Neuronale Netze
Radiale-Basisfunktionen-Netze

Prof. Dr.-Ing. Sebastian Stober

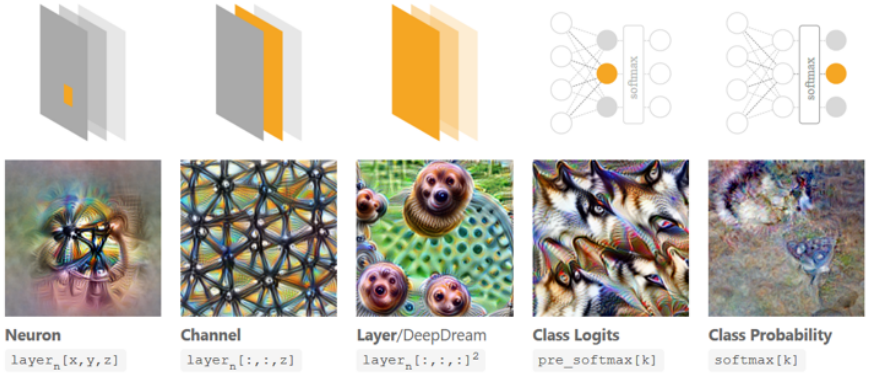
Artificial Intelligence Lab
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
stober@ovgu.de

FACULTY OF
COMPUTER SCIENCE

AiLAB

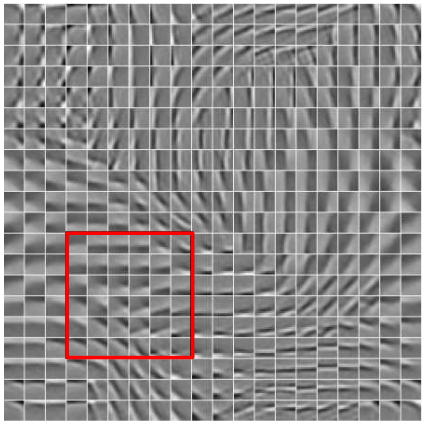
Recap: Introspection

feature visualization (optimize input)

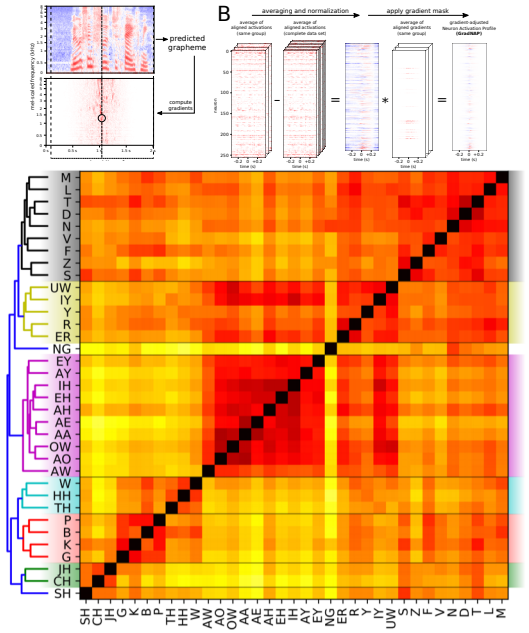


relevance / saliency analysis (for given input)

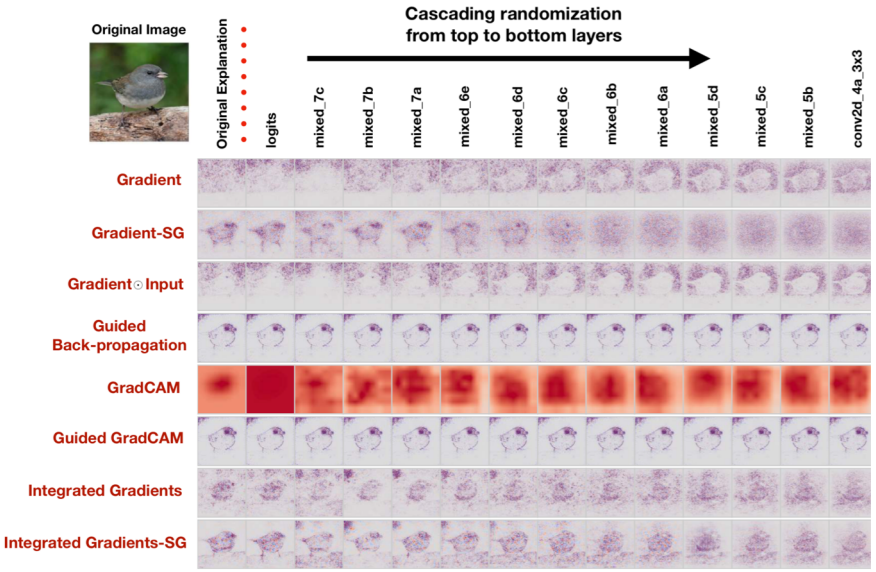
feature topography
(improve interpretability)



neuron activation profiles

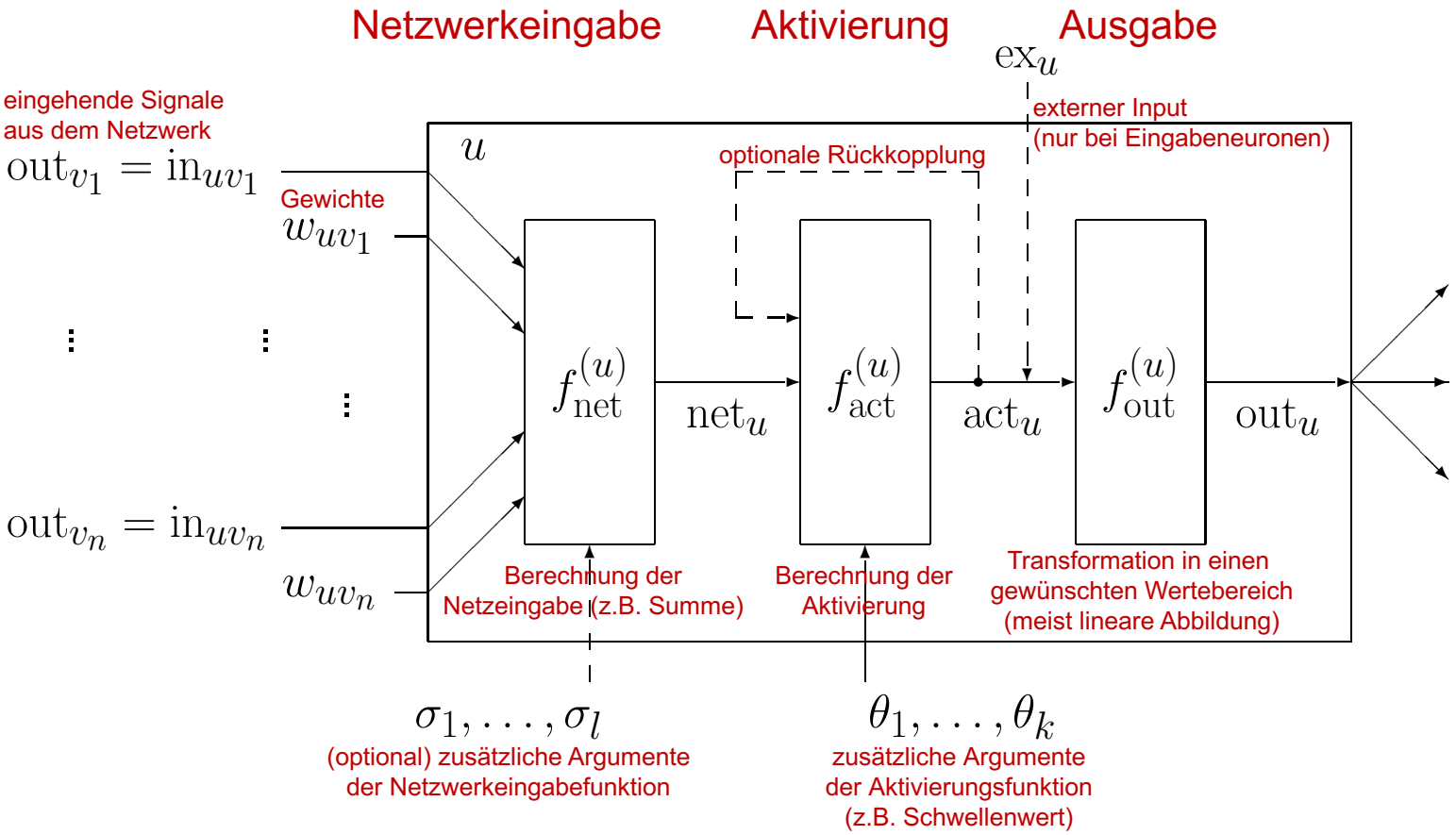


=> sanity checks!



Recap: Allgemeines Neuron

Ein verallgemeinertes Neuron verarbeitet numerische Werte



Radiale-Basisfunktionen-Netze

Eigenschaften von Radiale-Basisfunktionen-Netzen (RBF-Netzen)

RBF-Netze sind streng geschichtete, vorwärtsbetriebene neuronale Netze mit genau einer versteckten Schicht.

Als Netzeingabe- und Aktivierungsfunktion werden radiale Basisfunktionen verwendet.

Jedes Neuron erhält eine Art „Einzugsgebiet“.

Die Gewichte der Verbindungen von der Eingabeschicht zu einem Neuron geben das Zentrum an.

Radiale-Basisfunktionen-Netze

Ein **radiale-Basisfunktionen-Netz (RBF-Netz)** ist ein neuronales Netz mit einem Graph $G = (U, C)$, das die folgenden Bedingungen erfüllt:

(i) $U_{\text{in}} \cap U_{\text{out}} = \emptyset$,

(ii) $C = (U_{\text{in}} \times U_{\text{hidden}}) \cup C', \quad C' \subseteq (U_{\text{hidden}} \times U_{\text{out}})$

Die Netzeingabefunktion jedes versteckten Neurons ist eine **Abstandsfunktion** zwischen dem Eingabevektor und dem Gewichtsvektor, d.h.

$$\forall u \in U_{\text{hidden}}: \qquad f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = d(\vec{w}_u, \vec{\text{in}}_u),$$

wobei $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+$ eine Funktion ist, die $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^n$: erfüllt:

$$\begin{aligned}
&(i) && d(\vec{x}, \vec{y}) = 0 \quad \Leftrightarrow \quad \vec{x} = \vec{y}, && \\
&(ii) && d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x}) && \text{(Symmetrie)}, \\
&(iii) && d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) && \text{(Dreiecksungleichung)}.
\end{aligned}$$

Abstandsfunktionen

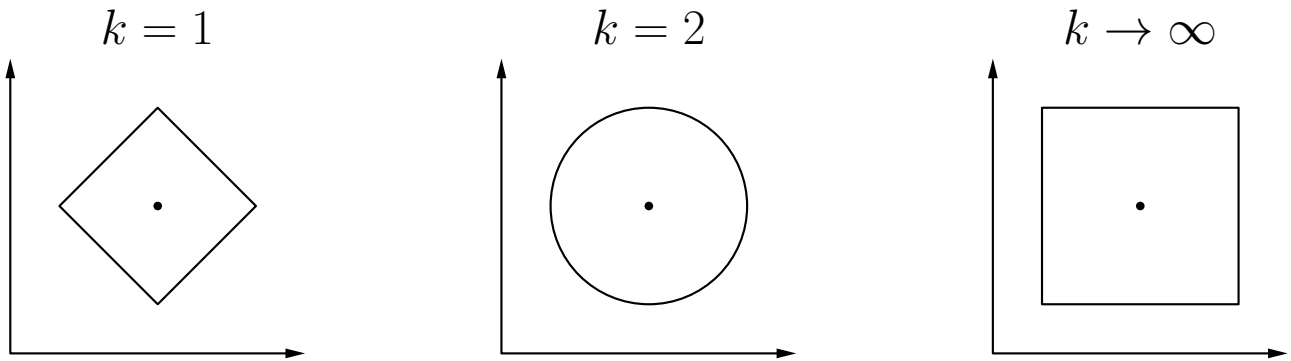
Veranschaulichung von Abstandsfunktionen

$$d_k(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^k \right)^{\frac{1}{k}}$$

Minkowski-Familie von Abstandsfunktionen

Bekannte Spezialfälle dieser Familie sind:

$k = 1$: Manhattan-Abstand ,
$k = 2$: Euklidischer Abstand,
$k \to \infty$: Maximum-Abstand, d.h. $d_\infty(\vec{x}, \vec{y}) = \max_{i=1}^{n} |x_i - y_i|$.



(alle Punkte auf dem Kreis bzw. den Vierecken haben denselben Abstand zum Mittelpunkt, entsprechend der jeweiligen Abstandsfunktion)

Radiale-Basisfunktionen-Netze

Die Netzeingabefunktion der Ausgabeneuronen ist die gewichtete Summe ihrer Eingaben, d.h.

$$\forall u \in U_{\text{out}}: \qquad f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = \vec{w}_u \vec{\text{in}}_u = \sum_{v \in \text{pred}(u)} w_{uv} \, \text{out}_v \, .$$

Die Aktivierungsfunktion jedes versteckten Neurons ist eine sogenannte **radiale Funktion**, d.h. eine monoton fallende Funktion

$$f : \mathbb{R}_0^+ \to [0,1] \quad \text{mit} \quad f(0) = 1 \quad \text{und} \quad \lim_{x \to \infty} f(x) = 0.$$

Die Aktivierungsfunktion jedes Ausgabeneurons ist eine lineare Funktion

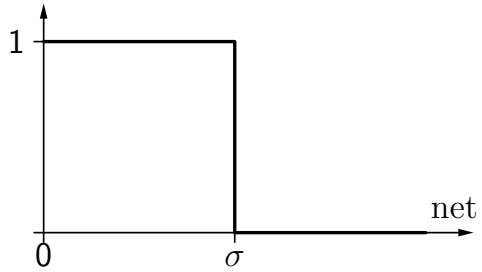
$$f_{\text{act}}^{(u)}(\text{net}_u, \theta_u) = \text{net}_u - \theta_u.$$

(Die lineare Aktivierungsfunktion ist wichtig für die Initialisierung.)

Radiale Aktivierungsfunktionen

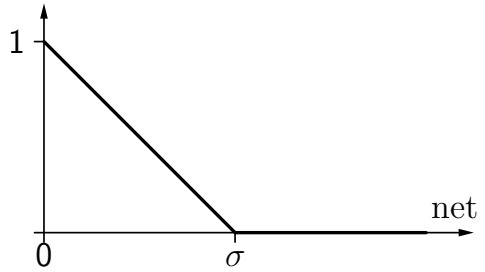
Rechteckfunktion:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{falls net} > \sigma, \\ 1, & \text{sonst.} \end{cases}$$



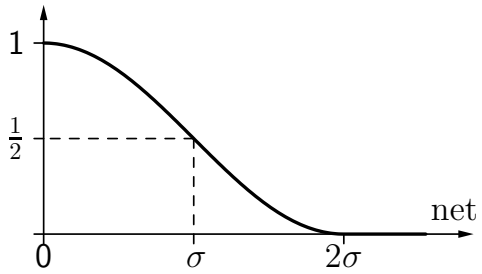
Dreiecksfunktion:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{falls net} > \sigma, \\ 1 - \frac{\text{net}}{\sigma}, & \text{sonst.} \end{cases}$$



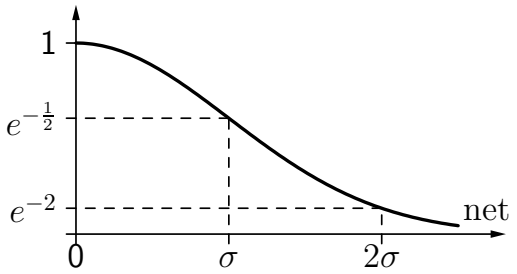
Kosinus bis Null:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{falls net} > 2\sigma, \\ \frac{\cos\left(\frac{\pi}{2\sigma}\,\text{net}\right)+1}{2}, & \text{sonst.} \end{cases}$$



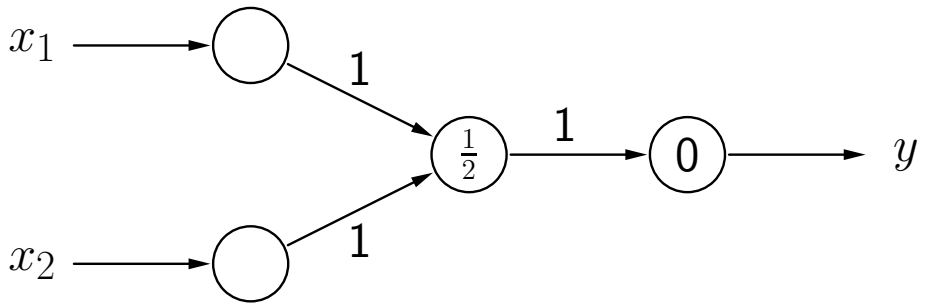
Gaußsche Funktion:

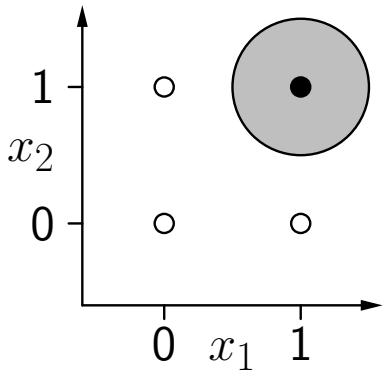
$$f_{\text{act}}(\text{net}, \sigma) = e^{-\frac{\text{net}^2}{2\sigma^2}}$$



Beispiele

Radiale-Basisfunktionen-Netz für die Konjunktion $x_1 \wedge x_2$



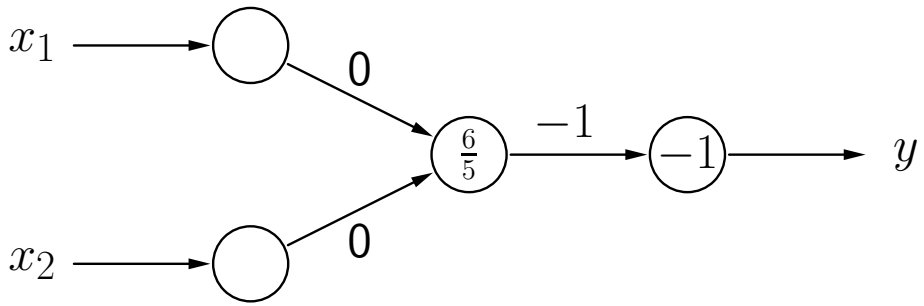


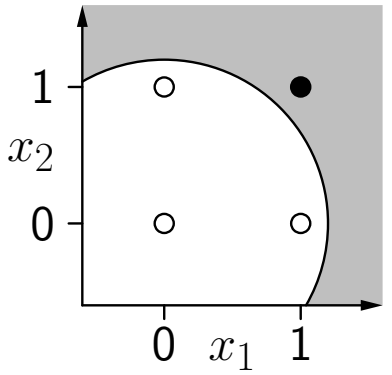
Radiale Funktion ist hier Rechtecksfunktion mit Zentrum (1,1)
und Referenzradius $\sigma = \frac{1}{2}$
Euklidischer Abstand $d(\vec{w}, \vec{x}) \leq \sigma$
Biaswert 0 im Ausgabeneuron

Für $(x_1, x_2) = (0.8, 0.9)$
$d((1.0, 1.0), (0.8, 0.9)) \leq \frac{1}{2}$
$\sqrt{(1.0 - 0.8)^2 + (1.0 - 0.9)^2} \leq \frac{1}{2}$
$\sqrt{0.014} \leq \frac{1}{2}$ gilt
$\Rightarrow$ RBF-Neuron feuert
$y = 1 \cdot 1 - 0 = 1$
$\Rightarrow$ Ausgabeneuron feuert

Beispiele

Radiale-Basisfunktionen-Netz für die Konjunktion $x_1 \wedge x_2$





Radiale Funktion ist hier Rechtecksfunktion mit Zentrum (0,0)
und Referenzradius $\sigma = \frac{6}{5}$
Euklidischer Abstand $d(\vec{w}, \vec{x}) \leq \sigma$
Biaswert -1 im Ausgabeneuron

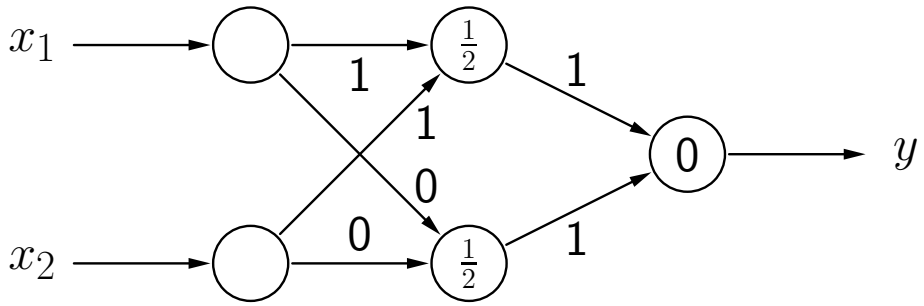
Für $(x_1, x_2) = (0.8, 0.9)$
$d((0.0, 0.0), (0.8, 0.9)) \leq \frac{6}{5}$
$\sqrt{(0.0 - 0.8)^2 + (0.0 - 0.9)^2} \leq \frac{6}{5}$
$\sqrt{1.45} \leq \frac{6}{5}$ gilt nicht
$\Rightarrow$ RBF-Neuron feuert nicht
$y = 0 \cdot (-1) - (-1) = 1$
$\Rightarrow$ Ausgabeneuron feuert

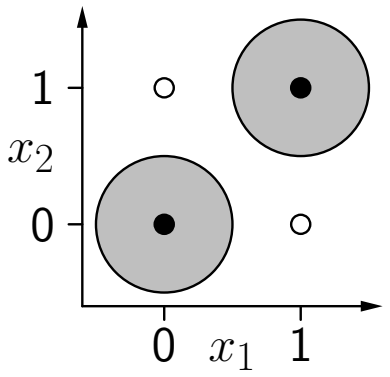
Beispiele

Radiale-Basisfunktionen-Netz für die Biimplikation $x_1 \leftrightarrow x_2$

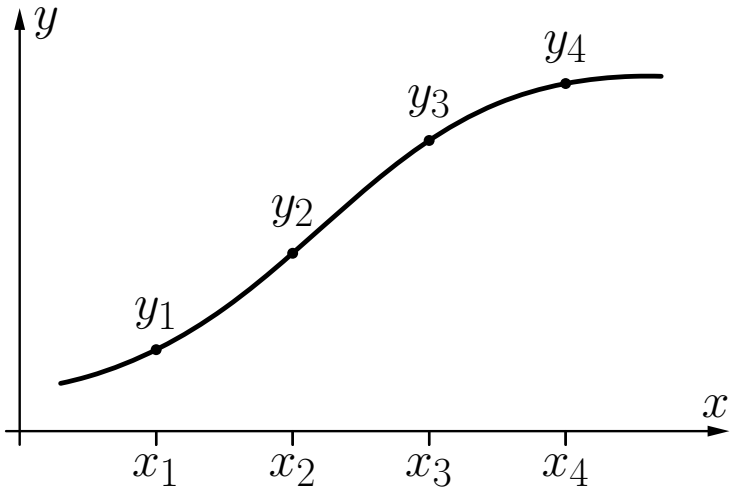
Idee: logische Zerlegung

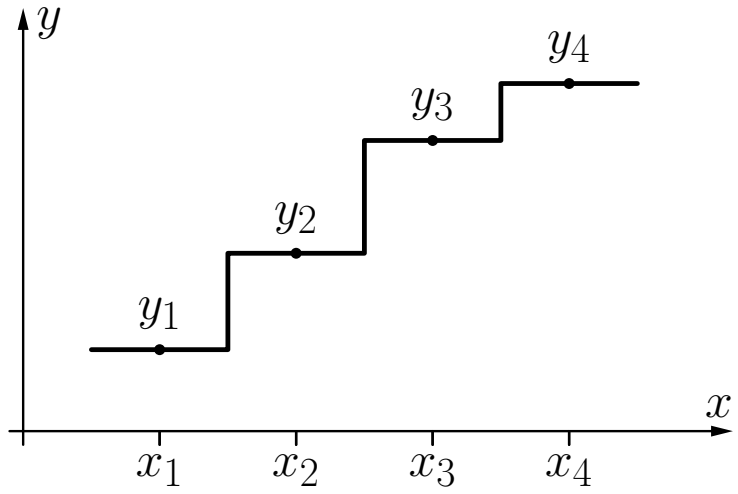
$$x_1 \leftrightarrow x_2 \quad \equiv \quad (x_1 \wedge x_2) \vee \neg(x_1 \vee x_2)$$

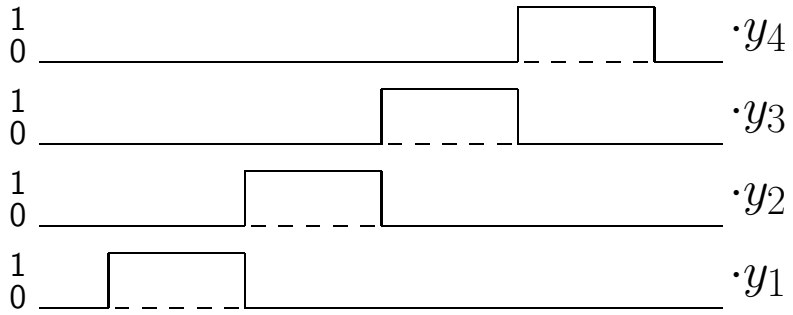




Funktionsapproximation

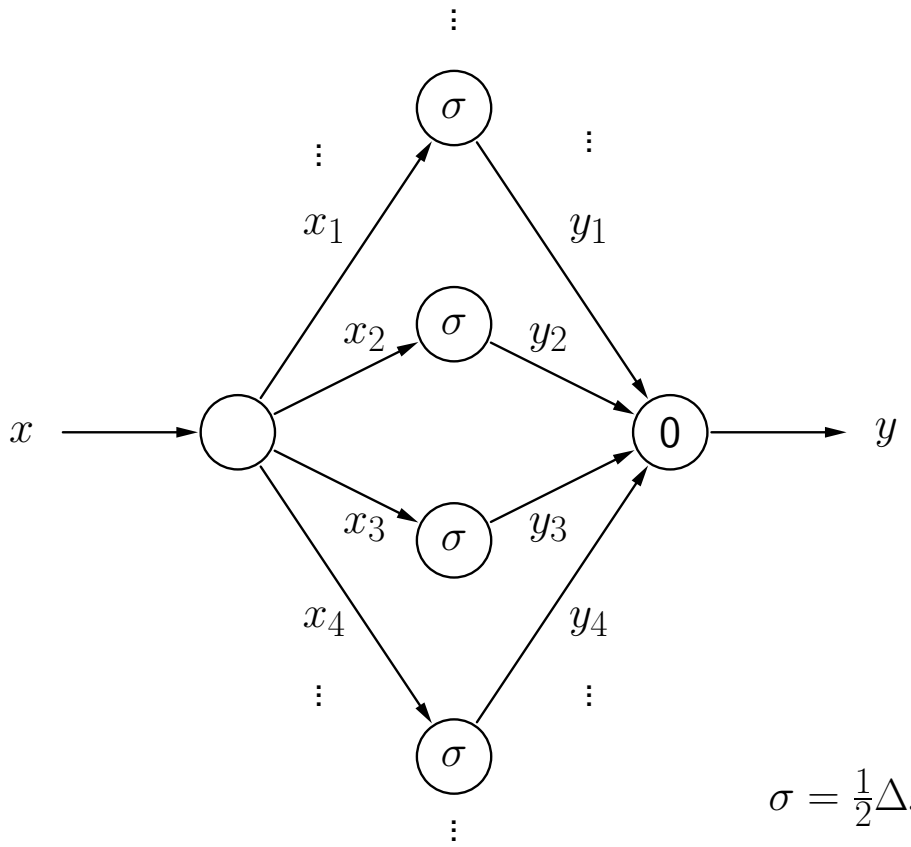






Annäherung der Originalfunktion durch Stufenfunktionen, deren Stufen durch einzelne Neuronen eines RBF-Netzes dargestellt werden können (vgl. MLPs).

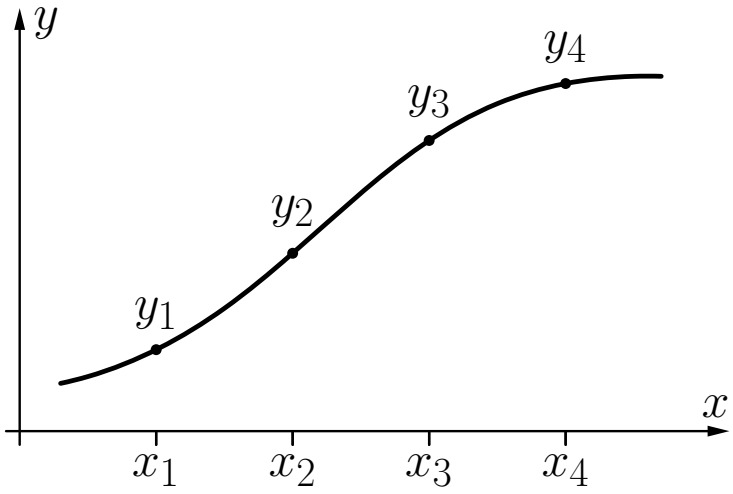
Funktionsapproximation

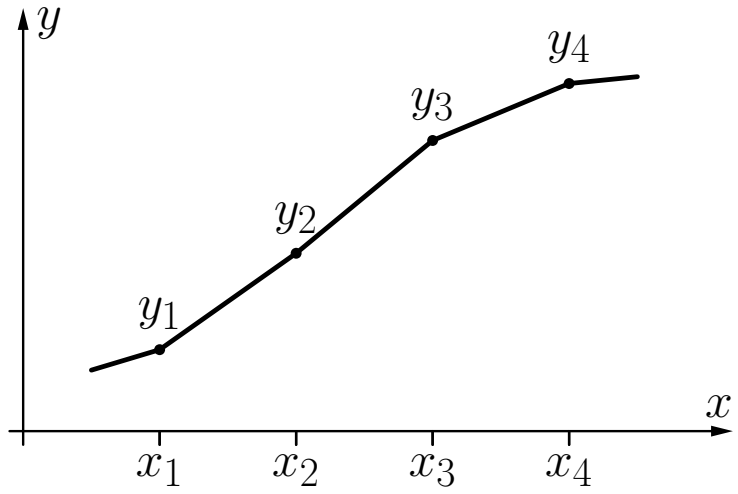


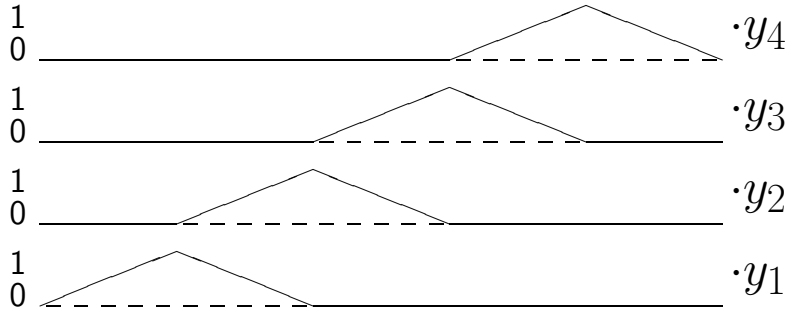
$$\sigma = \frac{1}{2}\Delta x = \frac{1}{2}(x_{i+1} - x_i)$$

Ein RBF-Netz, das die Treppenfunktion von der vorherigen Folie bzw. die stückweise lineare Funktion der folgenden Folie berechnet (dazu muss nur die Aktivierungsfunktion der versteckten Neuronen geändert werden).

Funktionsapproximation

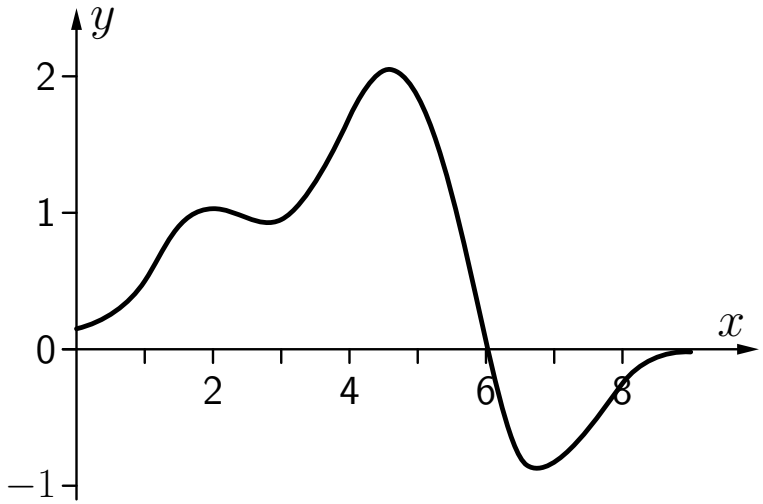


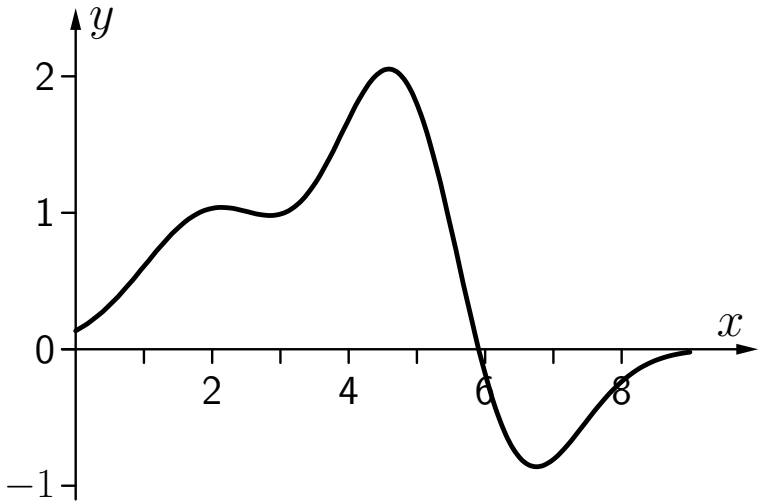


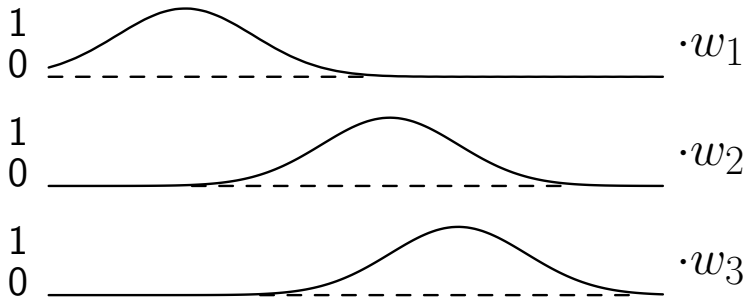


Darstellung einer stückweise linearen Funktion durch eine gewichtete Summe von Dreiecksfunktionen mit Zentren x_i.

Funktionsapproximation



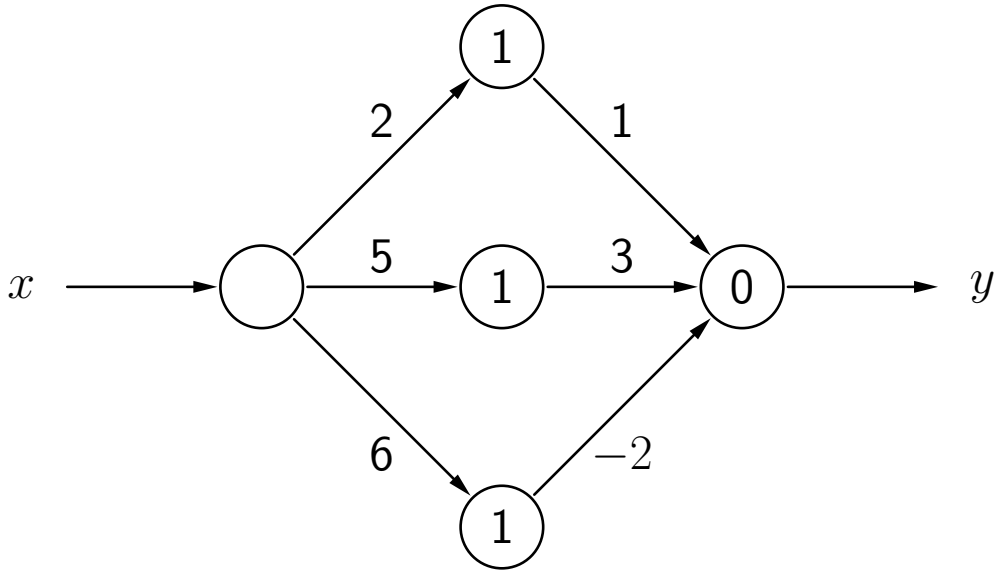




Annäherung einer Funktion durch eine Summe von Gaußkurven mit Radius $\sigma = 1$.
Es ist $w_1 = 2$, $w_2 = 3$ und $w_3 = -2$.

Funktionsapproximation

RBF-Netz für eine Summe dreier Gaußfunktionen



Training von RBF-Netzen
1. Initialisierung

Initialisierung

Sei $L_{\text{fixed}} = \{l_1, \ldots, l_m\}$ eine feste Lernaufgabe,
bestehend aus m Trainingsbeispielen $l = (\vec{\imath}^{\,(l)}, \vec{o}^{\,(l)})$.

Einfaches RBF-Netz:
Ein verstecktes Neuron v_k, $k = 1, \ldots, m$, für jedes Trainingsbeispiel

$$\forall k \in \{1, \ldots, m\}: \qquad \vec{w}_{v_k} = \vec{\imath}^{\,(l_k)}.$$

Falls die Aktivierungsfunktion die Gaußfunktion ist,
werden die Radien σ_k nach einer Heuristik gewählt

$$\forall k \in \{1, \ldots, m\}: \qquad \sigma_k = \frac{d_{\max}}{\sqrt{2m}},$$

wobei

$$d_{\max} = \max_{l_j, l_k \in L_{\text{fixed}}} d\left(\vec{\imath}^{\,(l_j)}, \vec{\imath}^{\,(l_k)}\right).$$

(max. Abstand zwischen Trainingsbeispielen)

Initialisierung

Initialisieren der Verbindungen von den versteckten zu den Ausgabeneuronen

$$\forall u : \sum_{k=1}^{m} w_{uv_m} \operatorname{out}_{v_m}^{(l)} - \theta_u = o_u^{(l)} \qquad \text{oder (abgekürzt)} \qquad \mathbf{A} \cdot \vec{w}_u = \vec{o}_u,$$

wobei $\vec{o}_u = (o_u^{(l_1)}, \ldots, o_u^{(l_m)})^\top$ der Vektor der gewünschten Ausgaben ist, $\theta_u = 0$, und

$$\mathbf{A} = \begin{pmatrix} \operatorname{out}_{v_1}^{(l_1)} & \operatorname{out}_{v_2}^{(l_1)} & \ldots & \operatorname{out}_{v_m}^{(l_1)} \\ \operatorname{out}_{v_1}^{(l_2)} & \operatorname{out}_{v_2}^{(l_2)} & \ldots & \operatorname{out}_{v_m}^{(l_2)} \\ \vdots & \vdots & & \vdots \\ \operatorname{out}_{v_1}^{(l_m)} & \operatorname{out}_{v_2}^{(l_m)} & \ldots & \operatorname{out}_{v_m}^{(l_m)} \end{pmatrix}$$

Ausgabewerte der versteckten Schicht für alle Trainingsbeispiele

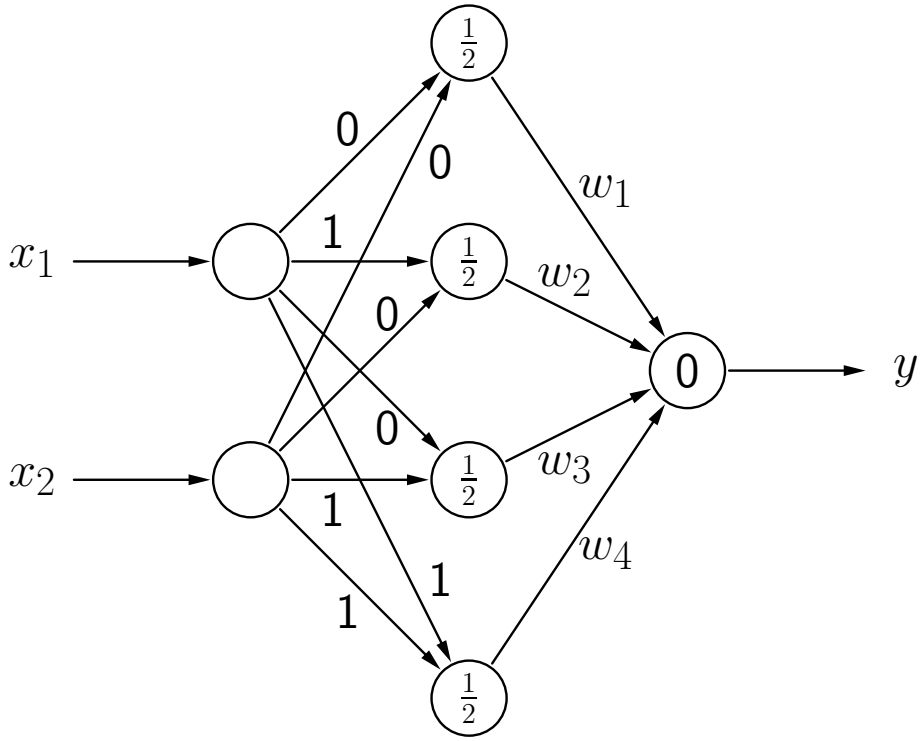
Ergebnis: Lineares Gleichungssystem, das durch Invertieren der Matrix $\mathbf{A}$ gelöst werden kann:

$$\vec{w}_u = \mathbf{A}^{-1} \cdot \vec{o}_u.$$

Beispiel

Einfaches RBF-Netz für die Biimplikation $x_1 \leftrightarrow x_2$

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



Beispiel

Einfaches RBF-Netz für die Biimplikation $x_1 \leftrightarrow x_2$

$$\mathbf{A} = \begin{pmatrix} 1 & e^{-2} & e^{-2} & e^{-4} \\ e^{-2} & 1 & e^{-4} & e^{-2} \\ e^{-2} & e^{-4} & 1 & e^{-2} \\ e^{-4} & e^{-2} & e^{-2} & 1 \end{pmatrix} \qquad \mathbf{A}^{-1} = \begin{pmatrix} \frac{a}{D} & \frac{b}{D} & \frac{b}{D} & \frac{c}{D} \\ \frac{b}{D} & \frac{a}{D} & \frac{c}{D} & \frac{b}{D} \\ \frac{b}{D} & \frac{c}{D} & \frac{a}{D} & \frac{b}{D} \\ \frac{c}{D} & \frac{b}{D} & \frac{b}{D} & \frac{a}{D} \end{pmatrix}$$

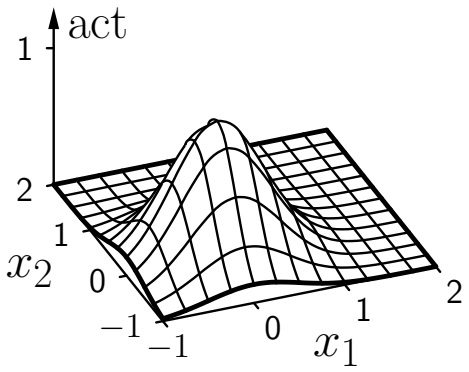
wobei

$$\begin{aligned} D &= 1 - 4e^{-4} + 6e^{-8} - 4e^{-12} + e^{-16} &&\approx \quad 0.9287 \\ a &= \quad 1 \quad - 2e^{-4} + e^{-8} &&\approx \quad 0.9637 \\ b &= -e^{-2} + 2e^{-6} - e^{-10} &&\approx -0.1304 \\ c &= \quad e^{-4} - 2e^{-8} + e^{-12} &&\approx \quad 0.0177 \end{aligned}$$

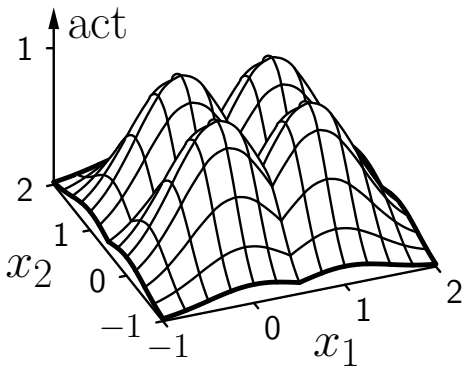
$$\vec{w}_u = \mathbf{A}^{-1} \cdot \vec{o}_u = \frac{1}{D} \begin{pmatrix} a+c \\ 2b \\ 2b \\ a+c \end{pmatrix} \approx \begin{pmatrix} 1.0567 \\ -0.2809 \\ -0.2809 \\ 1.0567 \end{pmatrix}$$

Beispiel

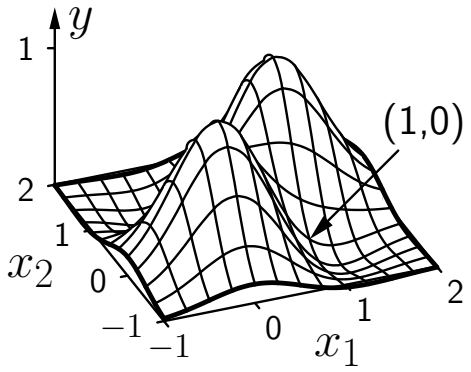
Einfaches RBF-Netz für die Biimplikation $x_1 \leftrightarrow x_2$



einzelne Basisfunktion



alle Basisfunktionen
(keine Summe!)



Ausgabe

Die Initialisierung führt bereits zu einer perfekten Lösung der Lernaufgabe.

Weiteres Trainieren ist nicht notwendig.

Initialisierung

Normale Radiale-Basisfunktionen-Netze:

Wähle Teilmenge von k Trainingsbeispielen als Zentren aus.

für Bias als Gewicht

$$\mathbf{A} = \begin{pmatrix} 1 & \mathrm{out}_{v_1}^{(l_1)} & \mathrm{out}_{v_2}^{(l_1)} & \dots & \mathrm{out}_{v_k}^{(l_1)} \\ 1 & \mathrm{out}_{v_1}^{(l_2)} & \mathrm{out}_{v_2}^{(l_2)} & \dots & \mathrm{out}_{v_k}^{(l_2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \mathrm{out}_{v_1}^{(l_m)} & \mathrm{out}_{v_2}^{(l_m)} & \dots & \mathrm{out}_{v_k}^{(l_m)} \end{pmatrix} \qquad \mathbf{A} \cdot \vec{w}_u = \vec{o}_u$$

Berechne (Moore–Penrose)-Pseudoinverse:

$$\mathbf{A}^+ = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top.$$

Die Gewichte können dann durch

$$\vec{w}_u = \mathbf{A}^+ \cdot \vec{o}_u = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \cdot \vec{o}_u$$

berechnet werden.

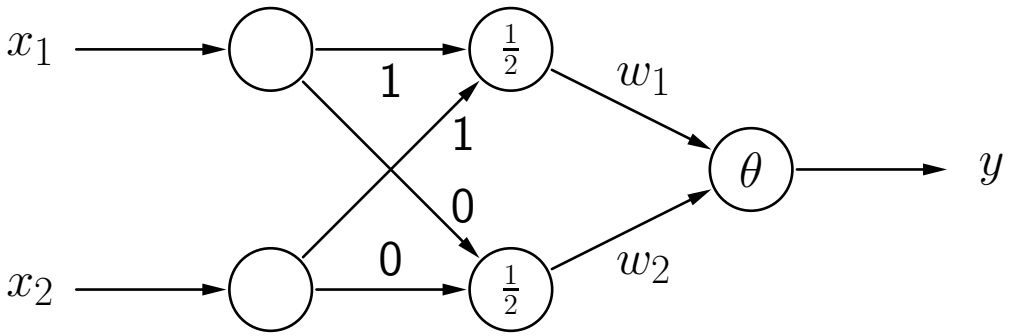
Beispiel

Normales RBF-Netz für die Biimplikation $x_1 \leftrightarrow x_2$

Wähle zwei Trainingsbeispiele aus:

$$l_1 = (\vec{\imath}^{(l_1)}, \vec{o}^{(l_1)}) = ((0,0),(1))$$

$$l_4 = (\vec{\imath}^{(l_4)}, \vec{o}^{(l_4)}) = ((1,1),(1))$$



Beispiel

Normales RBF-Netz für die Biimplikation $x_1 \leftrightarrow x_2$

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & e^{-4} \\ 1 & e^{-2} & e^{-2} \\ 1 & e^{-2} & e^{-2} \\ 1 & e^{-4} & 1 \end{pmatrix} \qquad \mathbf{A}^+ = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top = \begin{pmatrix} a & b & b & a \\ c & d & d & e \\ e & d & d & c \end{pmatrix}$$

wobei

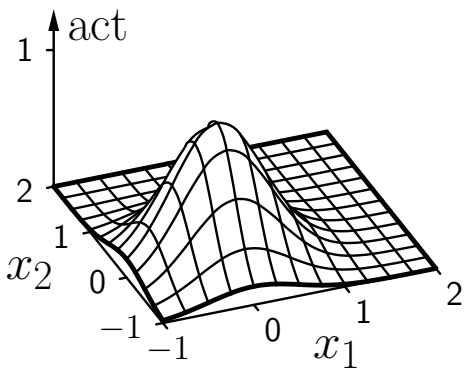
$$\begin{aligned} a &\approx -0.1810, & b &\approx \quad 0.6810, \\ c &\approx \quad 1.1781, & d &\approx -0.6688, & e &\approx 0.1594. \end{aligned}$$

Gewichte:

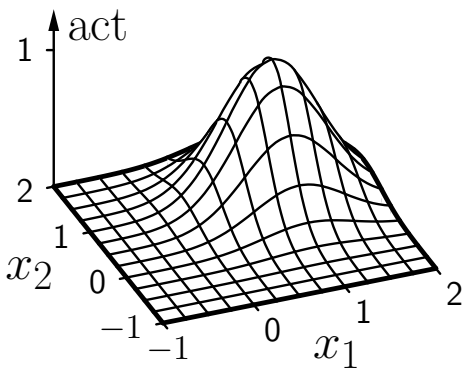
$$\vec{w}_u = \begin{pmatrix} -\theta \\ w_1 \\ w_2 \end{pmatrix} = \mathbf{A}^+ \cdot \vec{o}_u \approx \begin{pmatrix} -0.3620 \\ 1.3375 \\ 1.3375 \end{pmatrix}.$$

Beispiel

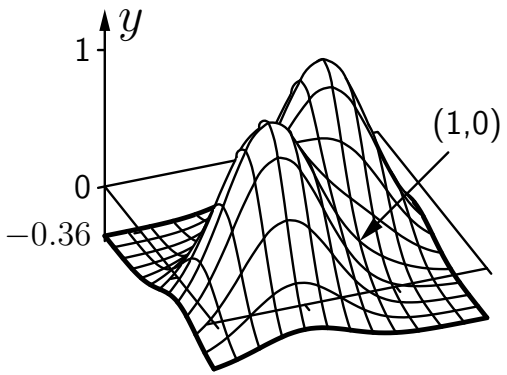
Normales RBF-Netz für die Biimplikation $x_1 \leftrightarrow x_2$



basis function (0,0)



basis function (1,1)



output

Die Initialisierung führt bereits zu einer perfekten Lösung der Lernaufgabe.

Dies ist Zufall, da das lineare Gleichungssystem wegen linear abhängiger Gleichungen nicht überbestimmt ist.

Initialisierung

Bestimmung passender Zentren für die RBFs

Ein Ansatz: **k-means-Clustering**

1. Wähle k zufällig ausgewählte Trainingsbeispiele als Zentren.
2. Weise jedem Zentrum die am nächsten liegenden Trainingsbeispiele zu.
3. Berechne neue Zentren als Schwerpunkt der dem Zentrum zugewiesenen Trainingsbeispiele.
4. Wiederhole Schritte 2 und 3 bis zur Konvergenz,
   d.h. bis sich die Zentren nicht mehr ändern.
5. Nutze die sich ergebenden Zentren für die Gewichtsvektoren der versteckten Neuronen.

Alternativer Ansatz: **Lernende Vektorquantisierung** (Details: nächste Vorlesung)

Training von RBF-Netzen
2. Training der Parameter

Training

Training von RBF-Netzen:
Herleitung der Update-Regeln ist analog zu der für MLPs.

Gewichte von den versteckten zu den Ausgabeneuronen.

Gradient:

$$\vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \frac{\partial e_u^{(l)}}{\partial \vec{w}_u} = -2(o_u^{(l)} - \mathrm{out}_u^{(l)})\, \vec{\mathrm{in}}_u^{(l)},$$

Gewichtsänderungsregel:

$$\Delta \vec{w}_u^{(l)} = -\frac{\eta_3}{2} \vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \eta_3 (o_u^{(l)} - \mathrm{out}_u^{(l)})\, \vec{\mathrm{in}}_u^{(l)}$$

(Zwei weitere Lernraten sind notwendig für die Positionen der Zentren und der Radien.)

Training

Training von RBF-Netzen:
Zentren: (Gewichte von Eingabe- zu versteckten Neuronen).

Gradient:

$$\vec{\nabla}_{\vec{w}_v} e^{(l)} = \frac{\partial e^{(l)}}{\partial \vec{w}_v} = -2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{su} \frac{\partial \mathrm{out}_v^{(l)}}{\partial \mathrm{net}_v^{(l)}} \frac{\partial \mathrm{net}_v^{(l)}}{\partial \vec{w}_v}$$

Gewichtsänderungsregel:

$$\Delta \vec{w}_v^{(l)} = -\frac{\eta_1}{2} \vec{\nabla}_{\vec{w}_v} e^{(l)} = \eta_1 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{sv} \frac{\partial \mathrm{out}_v^{(l)}}{\partial \mathrm{net}_v^{(l)}} \frac{\partial \mathrm{net}_v^{(l)}}{\partial \vec{w}_v}$$

Training

Training von RBF-Netzen:
Zentren: (Gewichte von Eingabe- zu versteckten Neuronen).

Spezialfall: **Euklidischer Abstand**

$$\frac{\partial \operatorname{net}_v^{(l)}}{\partial \vec{w}_v} = \left(\sum_{i=1}^{n} (w_{vp_i} - \operatorname{out}_{p_i}^{(l)})^2 \right)^{-\frac{1}{2}} (\vec{w}_v - \vec{\operatorname{in}}_v^{(l)}).$$

Spezialfall: **Gaußsche Aktivierungsfunktion**

$$\frac{\partial \operatorname{out}_v^{(l)}}{\partial \operatorname{net}_v^{(l)}} = \frac{\partial f_{\text{act}}(\operatorname{net}_v^{(l)}, \sigma_v)}{\partial \operatorname{net}_v^{(l)}} = \frac{\partial}{\partial \operatorname{net}_v^{(l)}} e^{-\frac{\left(\operatorname{net}_v^{(l)}\right)^2}{2\sigma_v^2}} = -\frac{\operatorname{net}_v^{(l)}}{\sigma_v^2} e^{-\frac{\left(\operatorname{net}_v^{(l)}\right)^2}{2\sigma_v^2}}.$$

Training

Training von RBF-Netzen:
Radien der radialen Basisfunktionen.

Gradient:

$$\frac{\partial e^{(l)}}{\partial \sigma_v} = -2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{su} \frac{\partial \mathrm{out}_v^{(l)}}{\partial \sigma_v}.$$

Gewichtsänderungsregel:

$$\Delta \sigma_v^{(l)} = -\frac{\eta_2}{2} \frac{\partial e^{(l)}}{\partial \sigma_v} = \eta_2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{sv} \frac{\partial \mathrm{out}_v^{(l)}}{\partial \sigma_v}.$$

Spezialfall: **Gaußsche Aktivierungsfunktion**

$$\frac{\partial \mathrm{out}_v^{(l)}}{\partial \sigma_v} = \frac{\partial}{\partial \sigma_v} e^{-\frac{\left(\mathrm{net}_v^{(l)}\right)^2}{2\sigma_v^2}} = \frac{\left(\mathrm{net}_v^{(l)}\right)^2}{\sigma_v^3} e^{-\frac{\left(\mathrm{net}_v^{(l)}\right)^2}{2\sigma_v^2}}.$$

Generelle Empfehlungen

- deutlich kleinere Lernraten als für MLPs (insbesondere für Verbindungsgewichte und Biaswerte der Ausgabeneuronen)

- von Online-Training wird abgeraten (wesentlichen instabiler als bei MLPs)

Verallgemeinerung

Verallgemeinerung der Abstandsfunktion

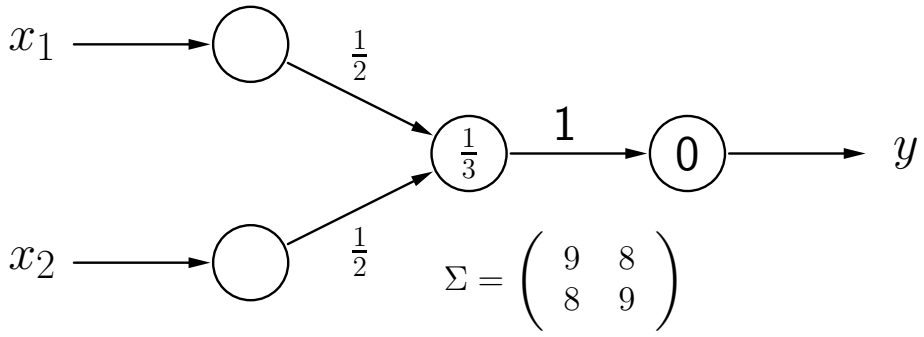
Idee: Benutze anisotrope (richtungsabhängige) Abstandsfunktion.

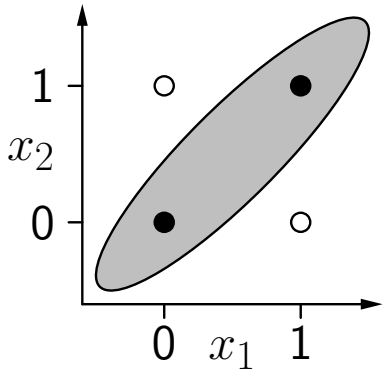
Beispiel: **Mahalanobis-Abstand**

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y})}.$$

Kovarianzmatrix (lernbar)

Beispiel: **Biimplikation**





Anwendung

Vorteile

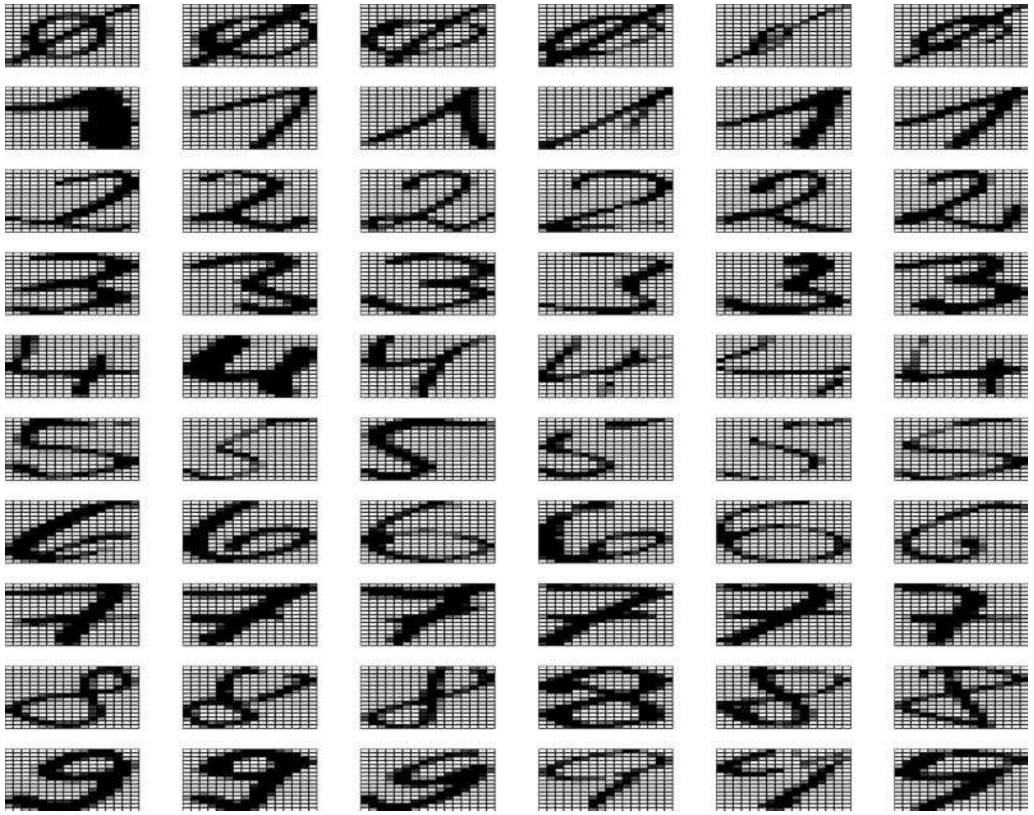
- einfache Feedforward-Architektur
- leichte Anpassbarkeit
- daher schnelle Optimierung und Berechnung

Anwendung

- kontinuierlich laufende Prozesse, die schnelle Anpassung erfordern
- Approximierung
- Mustererkennung
- Regelungstechnik

im Folgenden: Beispiel aus [Schwenker et al. 2001]

Beispiel: handgeschriebene Ziffern



Datensatz: 20.000 handgeschriebene Ziffern (2.000 Beispiele pro Klasse)

normiert in Höhe und Breite, dargestellt durch 16×16 Grauwerte $G_{ij} \in \{0, \ldots, 255\}$

Daten stammen vom EU-Projekt StatLog [Michie et al. 1994]

Ergebnisse mit RBF-Netzen

10.000 Trainingsbeispiele und 10.000 Testbeispiele (je 1.000 Beispiele pro Klasse)

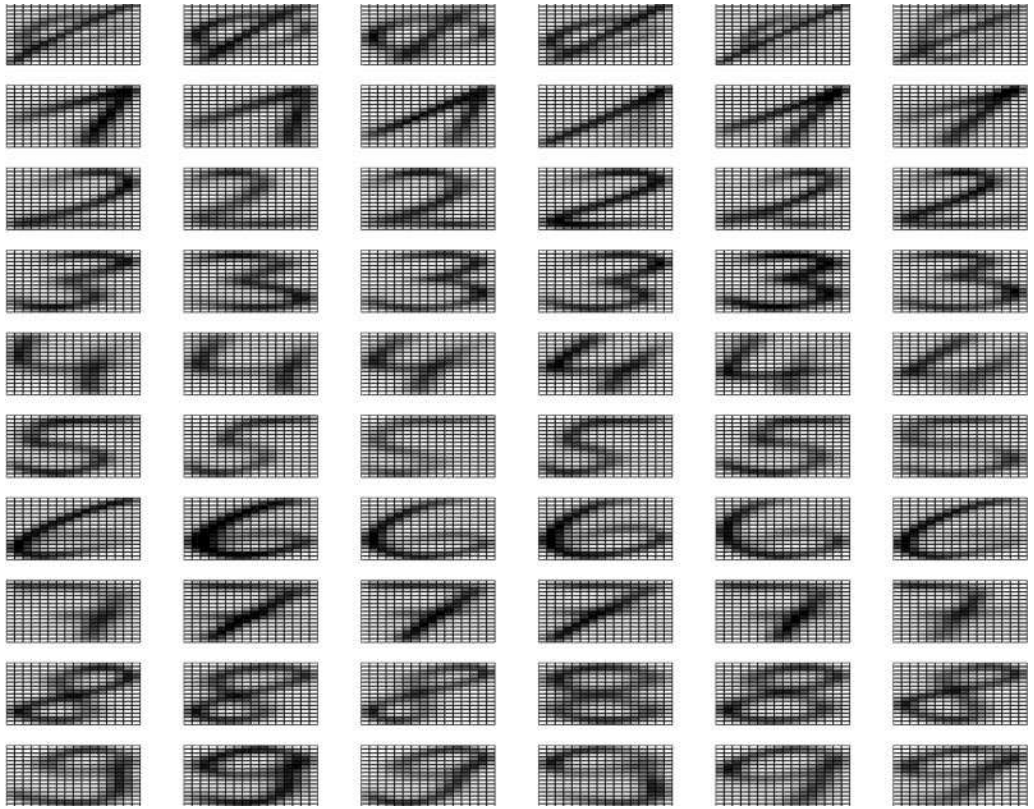
Trainingsdaten für Erlernen der Klassifikatoren, Testdaten zur Evaluierung

Initialisierung mittels k-means-Clustering mit 200 Prototypen (20 pro Klasse)

Methode	Erklärung	Testgenauigkeit
C4.5	Entscheidungsbaum	91.12%
RBF	k-means für RBF-Zentren	96.94%
MLP	eine versteckte Schicht mit 200 Neuronen	97.59%

hier: Median der Testgenauigkeiten von drei Trainings- und Testdurchläufen

k-means-Cluster zur RBF-Initialisierung

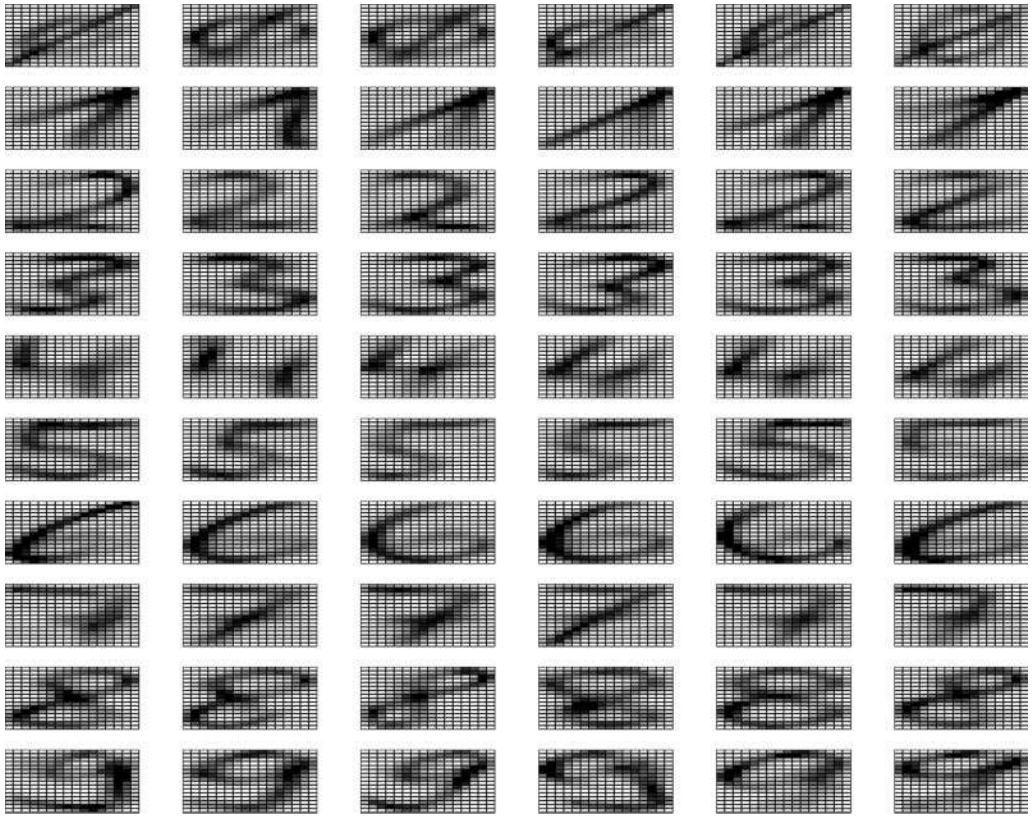


dargestellt: die 60 Cluster-Zentren der Ziffern nach k-means-Clustering

für jede Ziffer $k = 6$, wobei Algorithmus stets separat lief

Zentren wurden initialisiert durch zufällig gewählte Beispiele der Trainingsdaten

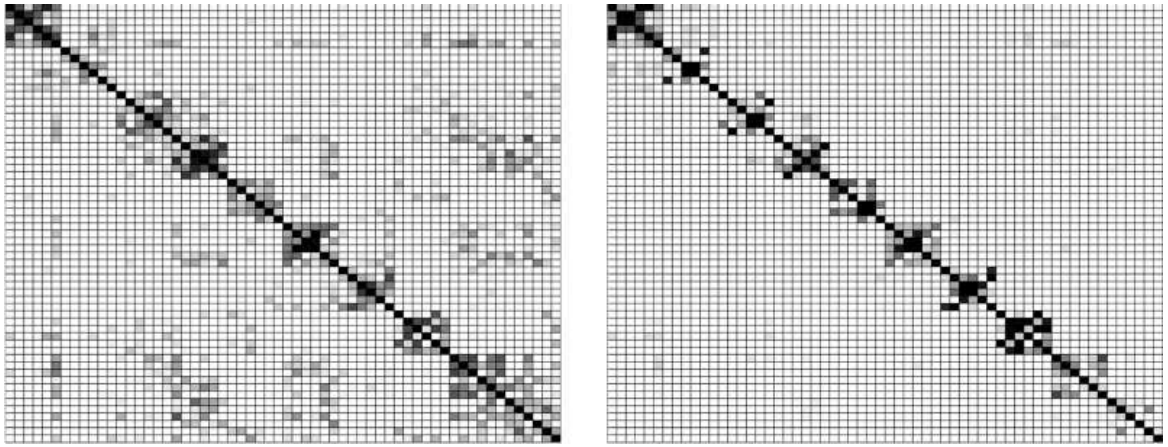
Gefundene RBF-Zentren nach 3 Durchläufen



dargestellt: die 60 RBF-Zentren der Ziffern nach 3 Rückpropagationsphasen des RBF-Netzes

kaum Unterschiede zu k-means erkennbar, aber...

Vergleich k-means und RBF-Netz



hier: Euklidischer Abstand der 60 RBF-Zentren vor und nach dem Training

Zentren sind so nach Klassen sortiert, dass ersten 6 Spalten/Zeilen die Ziffer 0 repräsentieren, nächsten 6 Spalten/Zeilen die Ziffer 1, usw.

Abstände kodiert als Grauwerte: je kleiner der Abstand, desto dunkler

Links: viele kleine Abstände zw. Zentren unterschiedlicher Klassen (z.B. 2–3 oder 8–9)

Diese kleinen Abstände führen oft zu Fehlklassifikationen

Rechts: nach dem Training lassen sich keine kleinen Abstände mehr finden