Seminarvortrag

Assoziativspeicher

Alexander Hentschel
hentsche@informatik.hu-berlin.de

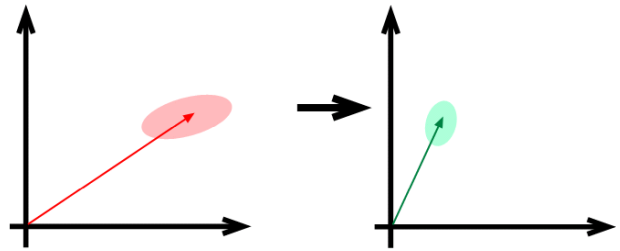
Assoziativspeicher

Inhalt

- 3 verschiedene Klassen von Assoziativspeichern
- allgemeine Funktionsweise
- Lernen in Assoziativspeichern: Hebbian-Learning
- empirische Resultate des Lernverfahrens
- Verbesserung der Erkennungsleistung von Assoziativspeichern

Bisher:

- neuronale Netze ohne Rückkopplung
- Backpropagation-Netze:
  Umgebung jeder bekannten Eingabe → Umgebung der jeweiligen Ausgabe

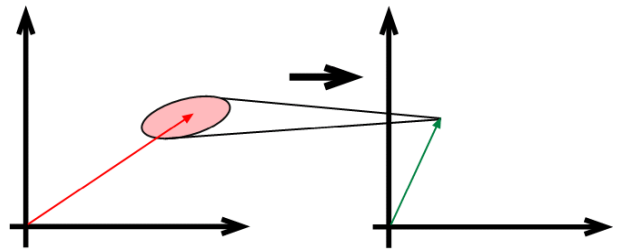


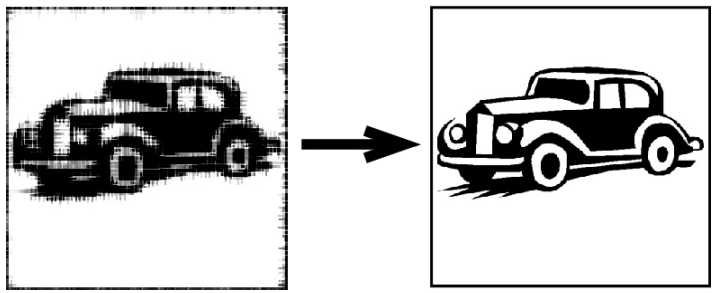
Jetzt:

- bestimmte Eingaben sollen mit zugehörigen Ausgaben **assoziiert** werden
  (daher Name: Assoziativspeicher)
- auch Umgebung jeder bekannten Eingabe **x** → jeweiligen Ausgabe **y**

Vorteil

- Assoziativspeicher kann leicht verrauschte
  Eingaben erkennen:

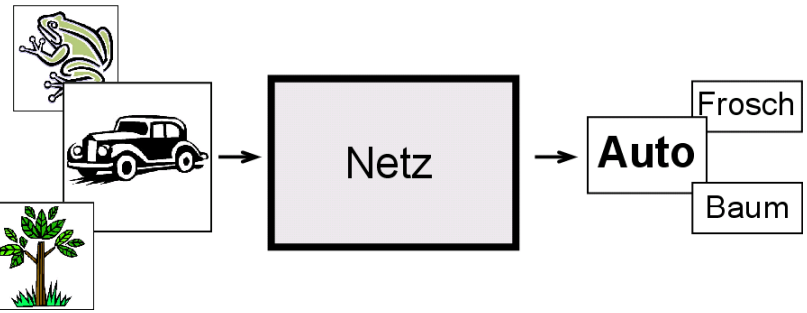




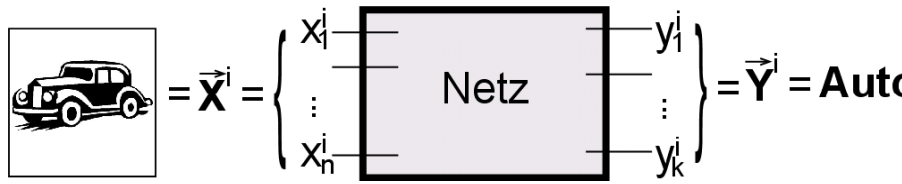
Klassen von Assoziativspeichern

Heteroassoziative Netze

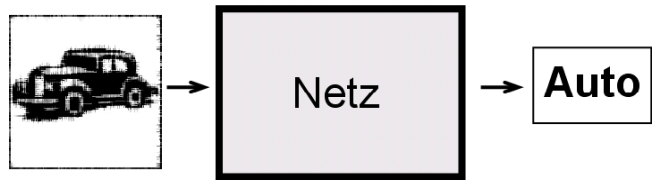
verknüpfen m Eingabevektoren $\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$
mit m Ausgabevektoren $\vec{y}^1, \vec{y}^2, \ldots \vec{y}^m$



Formal:



Eingabe $\tilde{\vec{x}}$ mit $\tilde{\vec{x}} \simeq \vec{x}^i$
soll ebenfalls Ausgabe $\vec{y}^i$ erzeugen

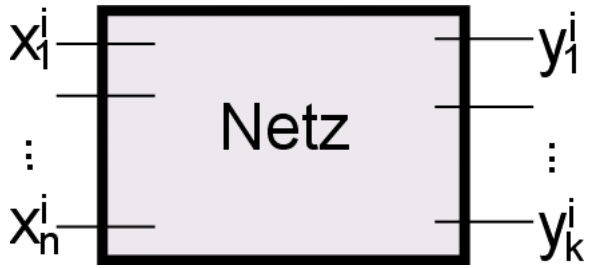


Klassen von Assoziativspeichern

Heteroassoziative Netze

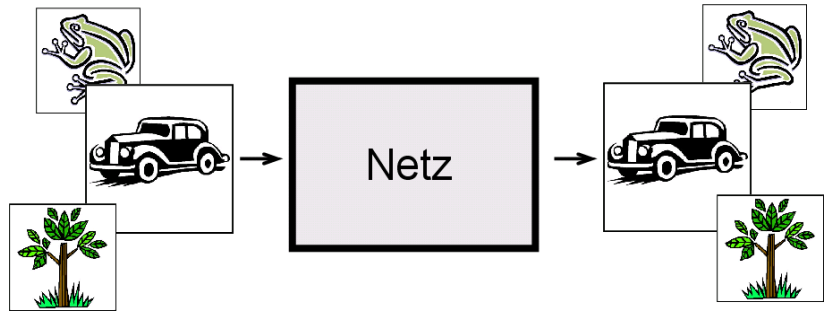
verknüpfen m Eingabevektoren $\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$
mit m Ausgabevektoren $\vec{y}^1, \vec{y}^2, \ldots \vec{y}^m$

Eingabe $\tilde{\vec{x}}$ mit $\tilde{\vec{x}} \simeq \vec{x}^i$
soll ebenfalls Ausgabe $\vec{y}^i$ erzeugen

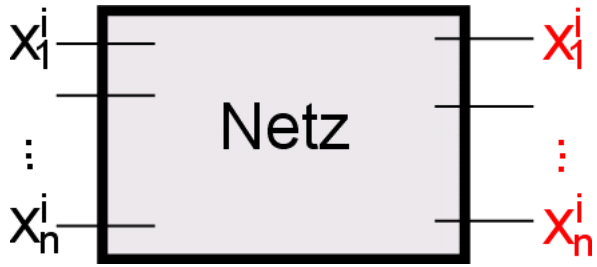


Autoassoziative Netze

Wie heteroassoziative Netze, aber
$\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$ werden mit sich selbst assoziiert: $\vec{y}^i \simeq \vec{x}^i$



Formal:

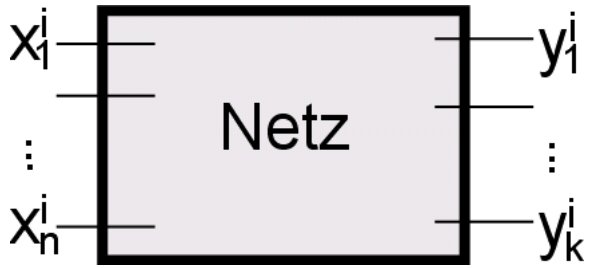


Klassen von Assoziativspeichern

Heteroassoziative Netze

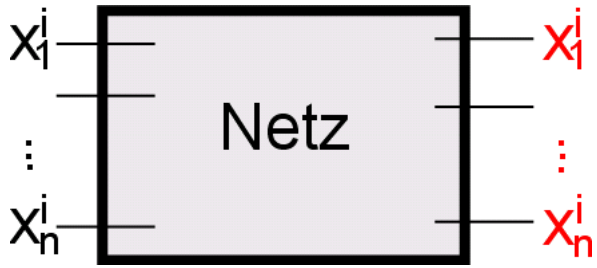
verknüpfen m Eingabevektoren $\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$
mit m Ausgabevektoren $\vec{y}^1, \vec{y}^2, \ldots \vec{y}^m$

Eingabe $\tilde{\vec{x}}$ mit $\tilde{\vec{x}} \simeq \vec{x}^i$
soll ebenfalls Ausgabe $\vec{y}^i$ erzeugen



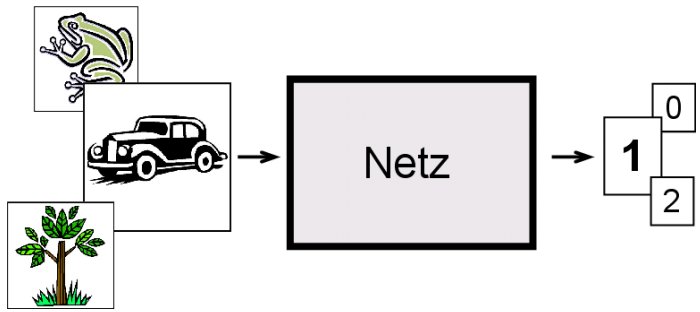
Autoassoziative Netze

Wie heteroassoziative Netze, aber
$\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$ werden mit sich selbst
assoziiert: $\vec{y}^i \simeq \vec{x}^i$



Mustererkennungsnetze

Spezialfall der heteroassoziativen Netze:
Eingabevektoren $\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$ mit
Nummern i = 1,2, ..., m assoziiert

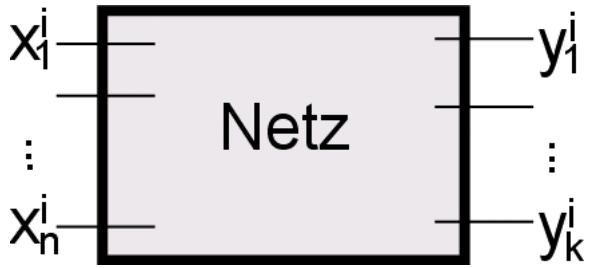


Klassen von Assoziativspeichern

Heteroassoziative Netze

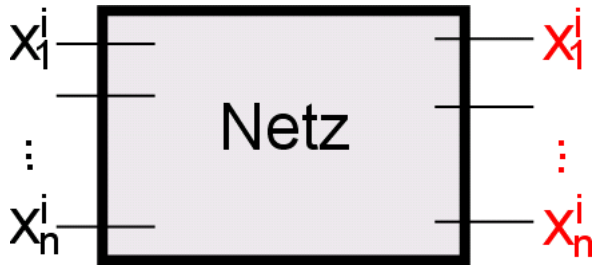
verknüpfen m Eingabevektoren $\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$
mit m Ausgabevektoren $\vec{y}^1, \vec{y}^2, \ldots \vec{y}^m$

Eingabe $\tilde{\vec{x}}$ mit $\tilde{\vec{x}} \simeq \vec{x}^i$
soll ebenfalls Ausgabe $\vec{y}^i$ erzeugen



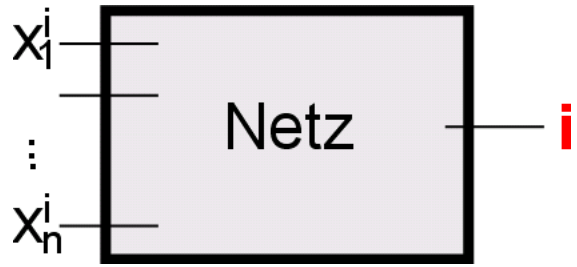
Autoassoziative Netze

Wie heteroassoziative Netze, aber
$\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$ werden mit sich selbst
assoziiert: $\vec{y}^i \simeq \vec{x}^i$



Mustererkennungsnetze

Spezialfall der heteroassoziativen Netze:
Eingabevektoren $\vec{x}^1, \vec{x}^2, \ldots \vec{x}^m$ mit
Nummern i = 1,2, ..., m assoziiert



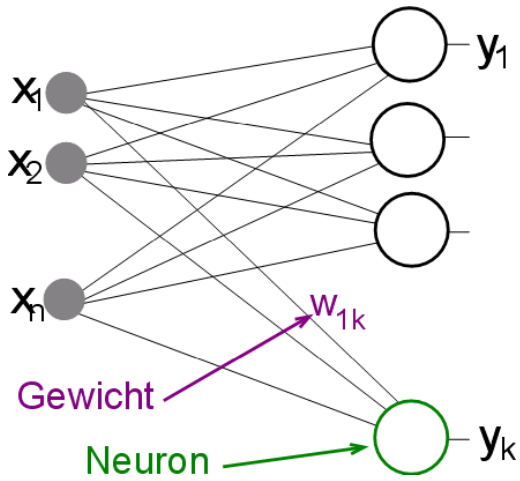
Struktur des Assoziativspeichers

Implementierbar mit nur einer Schicht von Neuronen

Leitungen von Eingabestellen zu Neuronen gewichtet durch w_{ij}

Gewichtematrix: $\mathbf{W} = \{w_{ij}\}$

Erregungen der Neuronen als Vektor: $\vec{\varepsilon} = (e_1, \ldots e_k)$



In Matrixschreibweise:

Eingabevektor $\vec{x} = (x_1, x_2, \ldots, x_n)$ erzeugt Erregungsvektor $\vec{\varepsilon}$

$$\vec{x}\mathbf{W} = \vec{\varepsilon}$$

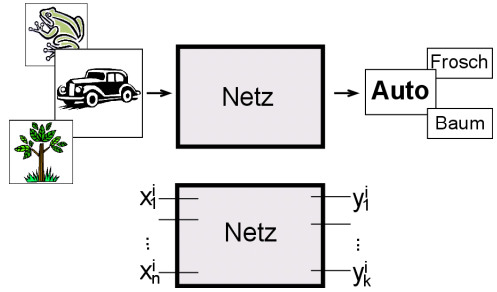
$$(x_1, x_2, x_3) \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} = (e_1, e_2, e_3)$$

mit $e_1 = x_1 w_{11} + x_2 w_{21} + x_3 w_{31}$

Lineare Assoziatoren: Neuronen geben als Ergebnis anliegende Erregung aus

$$\vec{x}\mathbf{W} = \vec{\varepsilon} = \vec{y}$$

Struktur des Assoziativspeichers



Ziel: **m** verschiedene n-dim. Eingabevektoren
assoziieren mit **m** k-dim. Ausgabevektoren

verwenden Lineare Assoziatoren: $\vec{x}\mathbf{W} = \vec{y}$

Verallgemeinerung der Matrixgleichung:

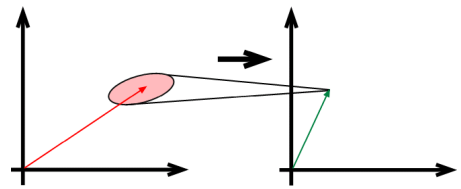
$$\underbrace{(x_1^i,\ldots,x_n^i)}_{\vec{x}^i}\underbrace{\begin{pmatrix} w_{11} & \cdots & w_{1k} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nk} \end{pmatrix}}_{\mathbf{W}} = \underbrace{(y_1^i,\ldots,y_k^i)}_{\vec{y}^i} \quad \longrightarrow \quad \underbrace{\begin{pmatrix} \vec{x}^1 \\ \vdots \\ \vec{x}^m \end{pmatrix}}_{\mathbf{X}}\underbrace{\begin{pmatrix} w_{11} & \cdots & w_{1k} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nk} \end{pmatrix}}_{\mathbf{W}} = \underbrace{\begin{pmatrix} \vec{y}^1 \\ \vdots \\ \vec{y}^m \end{pmatrix}}_{\mathbf{Y}}$$

$$\mathbf{XW} = \mathbf{Y}$$

Damit: Neuronales Netz (gegeben durch **W**) konstruiert, das Ziel erfüllt

Noch offen:

1) Netz soll verrauschte Eingaben erkennen
2) Wie findet man Matrix **W** (Lernalgorithmus) ?



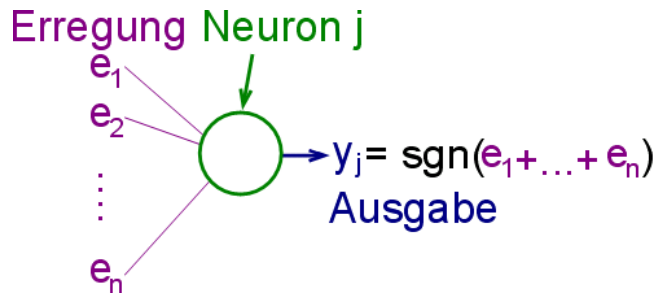
Erkennung verrauschter Eingaben

Codierung der Eingaben als **Bipolarverkoren:**

binär	0	1	0	0	1	0	1	1	1	0
bipolar	-1	1	-1	-1	1	-1	1	1	1	-1

Aktivierungsfunktion der Neuronen:

Vorzeichen-Funktion $\operatorname{sgn}(x) = \begin{cases} 1 \text{ für } x \geq 0 \\ -1 \text{ für } x < 0 \end{cases}$



In Kombination mit geeigneter Gewichtematrix **W** liefert dies gute Resultate

Lernalgorithmus **Hebbian-Learning:**

Gegeben:

- Netz aus einer einzigen Schicht von **k** Neuronen
- Vorzeichenfunktion als Aktivierungsfunktion der Neuronen
- n-dim. bipolarer Eingabevektor $\vec{x}$
- k-dim. bipolarer Ausgabevektor $\vec{y}$

Gesucht:

- passende Gewichte w_{ij}, so dass $\vec{x}$ auf $\vec{y}$ abgebildet wird

Lernalgorithmus **Hebbian-Learning:**

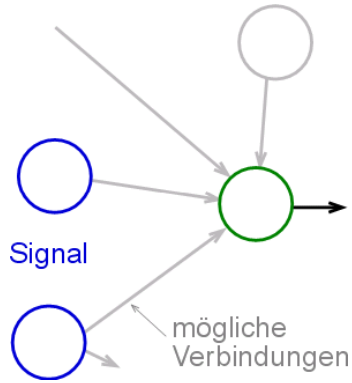
Gegeben:

- $\vec{x}, \vec{y}$

Gesucht:

- Matrix $\mathbf{W}=\{w_{ij}\}$ mit $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{y}$

Idee: zwei gleichzeitig aktive Neuronen gehen stärkere Verbindung ein als Neuronen, deren Aktivität nicht korreliert ist



Lernalgorithmus **Hebbian-Learning:**

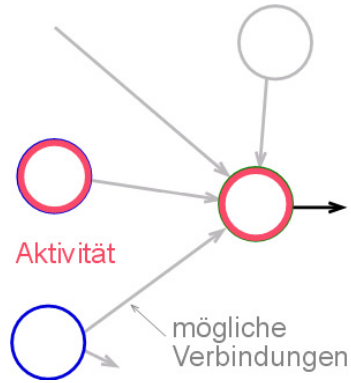
Gegeben:

- $\vec{x}, \vec{y}$

Gesucht:

- Matrix $\mathbf{W}=\{w_{ij}\}$ mit $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{y}$

Idee: zwei gleichzeitig aktive Neuronen gehen stärkere Verbindung ein als Neuronen, deren Aktivität nicht korreliert ist



Lernalgorithmus **Hebbian-Learning:**

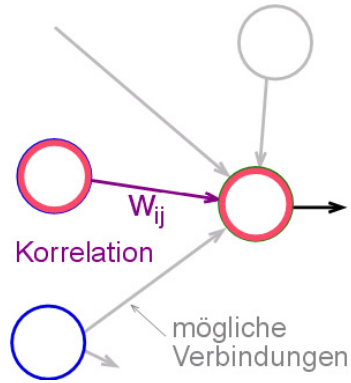
Gegeben:

- $\vec{x}, \vec{y}$

Gesucht:

- Matrix $\mathbf{W}=\{w_{ij}\}$ mit $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{y}$

Idee: zwei gleichzeitig aktive Neuronen gehen stärkere Verbindung ein als Neuronen, deren Aktivität nicht korreliert ist



für Assoziativspeicher

- Anfang: alle Gewichte $w_{ij}=0$
- in Lernphase: Ein- und Ausgabe von außen festgelegt
- liegt Erregung an einzelner Leitung an:
  Gewicht dieser Leitung um Δw_{ij} erhöhen

Verknüpfung von $\vec{x}$ mit $\vec{y}$ bei: $\Delta w_{ij} = x_i y_j$

Lernalgorithmus **Hebbian-Learning:**

Gegeben:

- $\vec{x}, \vec{y}$

Gesucht:

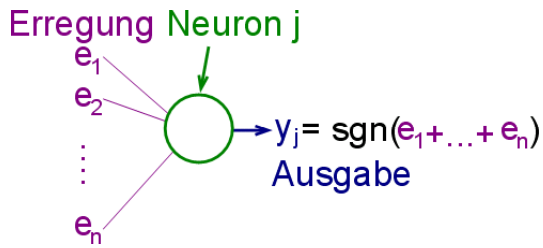
- Matrix $\mathbf{W}=\{w_{ij}\}$ mit $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{y}$

Verknüpfung von $\vec{x}$ mit $\vec{y}$ bei: $\Delta W_{ij} = x_i y_j$

Nach Lernphase: $\mathbf{W}=\{x_i y_j\}$ (Korrelationsmatrix)

Erregung: $$\vec{x}\mathbf{W} = (x_1,\ldots,x_n)\begin{pmatrix} x_1y_1 & \cdots & x_1y_k \\ \vdots & \ddots & \vdots \\ x_ny_1 & \cdots & x_ny_k \end{pmatrix} = (\underbrace{y_1\sum_{i=1}^{n} x_i x_i}_{e_1},\ldots,\underbrace{y_k\sum_{i=1}^{n} x_i x_i}_{e_k}) = \vec{y}\,(\vec{x}\cdot\vec{x})$$

Ausgabe des **k**-ten Neurons: $$\mathrm{sgn}(y_k \sum_{i=1}^{n} \underbrace{x_i x_i}_{>0}) = y_k$$ (± 1)



Ausgabe des Netzes: $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{y}$

Lernalgorithmus **Hebbian-Learning:**

Gegeben:
- $\vec{x}, \vec{y}$

Gesucht:
- Matrix $\mathbf{W}=\{w_{ij}\}$ mit $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{y}$

Verknüpfung von $\vec{x}$ mit $\vec{y}$ bei: $\Delta W_{ij} = x_i y_j$

Nach Lernphase: $\mathbf{W}=\{x_i y_j\}$ (Korrelationsmatrix)

Ziel: **m** verschiedene n-dim. Eingabevektoren
assoziieren mit **m** k-dim. Ausgabevektoren

Gegeben:
- $\vec{x}^1,\ldots,\vec{x}^m$ und $\vec{y}^1,\ldots,\vec{y}^m$

Gesucht:
- Matrix $\mathbf{W}=\{w_{ij}\}$ mit $\mathrm{sgn}(\vec{x}^i\mathbf{W}) = \vec{y}^i$

Hebb-Regel auf alle Vektorpaare $\vec{x}^i, \vec{y}^i$ anwenden:

Gewichtematrix:

$\mathbf{W}=\mathbf{W}^1+\mathbf{W}^2+\ldots+\mathbf{W}^m$ mit $\mathbf{W}^a=\{x_i^a y_j^a\}_{n\times k}$ n×k- Korrelationsmatrix der assoziierten Vektoren $\vec{x}^a, \vec{y}^a$

Lernalgorithmus **Hebbian-Learning:**

Gewichtematrix:

$$\mathbf{W} = \mathbf{W}^1 + \mathbf{W}^2 + \ldots + \mathbf{W}^m$$

mit $\mathbf{W}^a = \{x_i^a y_j^a\}_{n\times k}$ n×k- Korrelationsmatrix der assoziierten Vektoren $\vec{x}^i$, $\vec{y}^i$

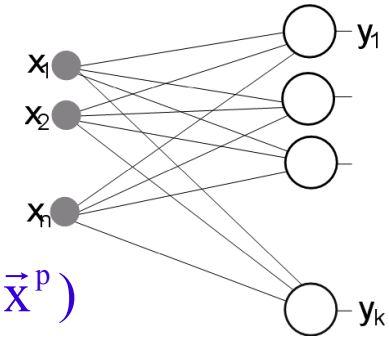
Eingabe: $\vec{x}^p$ mit $p \in \{1,\ldots,n\}$

Erregung:

$$\vec{x}^p \mathbf{W} = \vec{x}^p \left(\mathbf{W}^1 + \mathbf{W}^2 + \ldots + \mathbf{W}^m\right) = \vec{x}^p \mathbf{W}^p + \sum_{s\neq p} \vec{x}^p \mathbf{W}^s$$

$$= \vec{y}^p \underbrace{(\vec{x}^p \cdot \vec{x}^p)}_{\text{positive Konstante}} + \underbrace{\sum_{s\neq p} \vec{y}^s (\vec{x}^s \cdot \vec{x}^p)}_{\textbf{crosstalk}}$$

crosstalk
(Störfaktor)



Das Netz produziert gewünschte Ausgabe $\vec{y}^p$,falls

- crosstalk null ist (ist der Fall, wenn $\vec{x}^1,\ldots,\vec{x}^m$ paarweise orthogonal)
- crosstalk hinreichend klein zu $\vec{y}^p(\vec{x}^p \cdot \vec{x}^p)$

Lernalgorithmus **Hebbian-Learning:**

Eingabe: $\vec{x}^p$

Erregung: $\vec{x}^p \mathbf{W} = \vec{y}^p (\vec{x}^p \cdot \vec{x}^p) + \underbrace{\sum_{s \neq p} \vec{y}^s (\vec{x}^s \cdot \vec{x}^p)}_{\textbf{crosstalk}}$

(Störfaktor)

Ausgabe: $\operatorname{sgn}(\vec{x}^p \mathbf{W}) = \operatorname{sgn}\left(\vec{y}^p (\vec{x}^p \cdot \vec{x}^p) + \sum_{s \neq p} \vec{y}^s (\vec{x}^s \cdot \vec{x}^p) \right)$

$$= \operatorname{sgn}\left(\vec{y}^p + \sum_{s \neq p} \vec{y}^s \frac{(\vec{x}^s \cdot \vec{x}^p)}{(\vec{x}^p \cdot \vec{x}^p)} \right)$$

da $(\vec{x}^p \cdot \vec{x}^p)$ positive Konstante

gewollte Ausgabe: $\vec{y}^p \stackrel{!}{=} \operatorname{sgn}\left(\vec{y}^p + \sum_{s \neq p} \vec{y}^s \frac{(\vec{x}^s \cdot \vec{x}^p)}{(\vec{x}^p \cdot \vec{x}^p)} \right)$

Dafür muss gelten: absoluter Wert jeder Komponente des Störterms ≤ 1

d.h.: Skalarprodukt $\vec{x}^s \cdot \vec{x}^p$ $\ll$ Längenquadrat von $\vec{x}^p$

≈ 0
für nahezu
orthogonale Vektoren

$|\vec{x}^p|^2 = n$
mit n = Dimension von $\vec{x}^p$

Lernalgorithmus **Hebbian-Learning:**

$$\mathrm{sgn}(\vec{x}^{\,p}\mathbf{W}) = \mathrm{sgn}\left(\vec{y}^{\,p} + \sum_{s\neq p} \vec{y}^{\,s}\,\frac{(\vec{x}^{\,s}\cdot\vec{x}^{\,p})}{(\vec{x}^{\,p}\cdot\vec{x}^{\,p})}\right)$$

Beispiel:

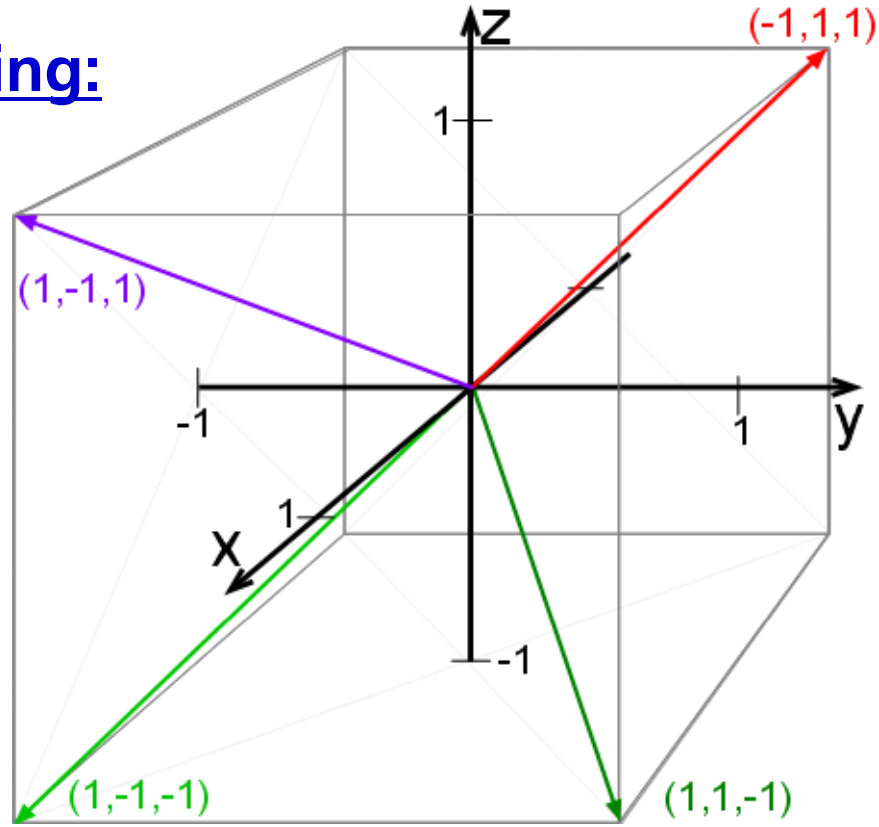
Erlernte Vektorpaare: $\vec{x}^{\,1} \rightarrow \vec{y}^{\,1}$

$\vec{x}^{\,2} \rightarrow \vec{y}^{\,2}$

Eingabe: $\vec{x}^{\,1}$

Ausgabe:

$$\mathrm{sgn}(\vec{x}^{\,1}\mathbf{W}) = \mathrm{sgn}\left(\vec{y}^{\,1} + \vec{y}^{\,2}\,\frac{(\vec{x}^{\,2}\cdot\vec{x}^{\,1})}{(\vec{x}^{\,1}\cdot\vec{x}^{\,1})}\right)$$



$$= \mathrm{sgn}\left(\vec{y}^{\,1} + \vec{y}^{\,2}\,\frac{-1}{3}\right)$$

$$= \mathrm{sgn}\left(\begin{pmatrix}1\\-1\\1\end{pmatrix} + \begin{pmatrix}1\\-1\\-1\end{pmatrix}\frac{-1}{3}\right) = \mathrm{sgn}\begin{pmatrix}2/3\\-2/3\\4/3\end{pmatrix} = \begin{pmatrix}1\\-1\\1\end{pmatrix} = \vec{y}^{\,1}$$

✓

Lernalgorithmus **Hebbian-Learning:**

Ziel:

- **m** verschiedene **n**-dim. Eingabevektoren assoziieren mit **m** **k**-dim. Ausgabevektoren
- Erkennung verrauschter Eingaben

Resultat:

- Lernverfahren liefert entsprechende Resultate für fast orthogonale Eingabevektoren
- für $\mathbf{m} \ll \mathbf{n}$ (Anzahl erlernter Vektorpaare $\ll$ Vektorlänge der Eingabe) zufällige Menge $\vec{x}^1,\ldots,\vec{x}^m$ von Bipolarvektoren mit hoher Wahrscheinlichkeit paarweise nahezu orthogonal

Anwendung: Bilderkennung
Bild mit 100kb $\approx$ 800.000 bit
bei $\mathbf{m}=0.01\mathbf{n}$ (1%): 8000 Bilder im Netz assoziierbar

Lernalgorithmus **Hebbian-Learning:**

verrauschte Eingabe: $\tilde{\vec{x}} \simeq \vec{x}^{\,p}$ ($\vec{x}^{\,p}$ erlernter Vektor der Dim. **n**)

Hemming-Abstand zwischen $\tilde{\vec{x}}$ und $\vec{x}^{\,p}$: H (Anzahl unterschiedlicher Komponenten)

Korrelationsmatrix $\mathbf{W}^a$ darstellbar als: $\vec{x}^{\,a^T} \otimes \vec{y}^{\,a}$ (⊗ - Tensorprodukt; $\vec{x}$, $\vec{y}$ - Zeilenvektoren)

Tensorprodukt: $$\vec{x}^{\,i^T} \otimes \vec{y}^{\,i} = \begin{pmatrix} x_1^i \cdot (y_1^i, \ldots, y_n^i) \\ \vdots \\ x_n^i \cdot (y_1^i, \ldots, y_n^i) \end{pmatrix}$$

Lernalgorithmus **Hebbian-Learning:**

verrauschte Eingabe: $\tilde{\vec{x}} \simeq \vec{x}^{\,p}$ ($\vec{x}^{\,p}$ erlernter Vektor der Dim. **n**)

Hemming-Abstand zwischen $\tilde{\vec{x}}$ und $\vec{x}^{\,p}$: H (Anzahl unterschiedlicher Komponenten)

Korrelationsmatrix $\mathbf{W}^a$ darstellbar als: $\vec{x}^{\,a^T} \otimes \vec{y}^{\,a}$ (⊗ - Tensorprodukt; $\vec{x}$, $\vec{y}$ - Zeilenvektoren)

Ausgabe:

$$\operatorname{sgn}\left(\tilde{\vec{x}}\mathbf{W}\right) = \operatorname{sgn}\left(\tilde{\vec{x}}\left(\mathbf{W}^1 + \mathbf{W}^2 + \ldots + \mathbf{W}^m\right)\right) = \operatorname{sgn}\left(\tilde{\vec{x}}\mathbf{W}^p + \sum_{s\neq p}\tilde{\vec{x}}\mathbf{W}^s\right)$$

$$= \operatorname{sgn}\left(\underbrace{\tilde{\vec{x}}\cdot\vec{x}^{\,p^T}}_{\text{= n-2H}} \otimes \vec{y}^{\,p} + \sum_{s\neq p}\underbrace{\tilde{\vec{x}}\cdot\vec{x}^{\,s^T}}_{\ll \mathbf{n}} \otimes \vec{y}^{\,s}\right)$$

Da $\tilde{\vec{x}} \simeq \vec{x}^{\,p}$ ist deren Hamming-Abstand klein verglichen mit der Vektorlänge

$\Rightarrow \mathbf{H} \ll \mathbf{n} \quad \Rightarrow \mathbf{n - 2H} \approx \mathbf{n}$

$\ll \mathbf{n}$

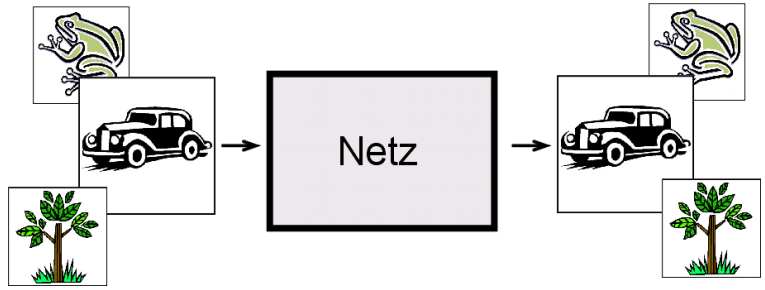
da $\tilde{\vec{x}} \simeq \vec{x}^{\,p}$ und $\vec{x}^{\,p}$ fast orthogonal zu $\vec{x}^{\,s}$

d.h. $\tilde{\vec{x}}\cdot\vec{x}^{\,sT} \simeq \vec{x}^{\,p}\cdot\vec{x}^{\,sT} \simeq 0$

$$\operatorname{sgn}\left(\tilde{\vec{x}}\mathbf{W}\right) \approx \operatorname{sgn}\left(\underbrace{\mathbf{n}\cdot\vec{y}^{\,p}}_{\text{Vektorkomponenten} = \pm\,\mathbf{n}} + \underbrace{\sum_{s\neq p}\tilde{\vec{x}}\cdot\vec{x}^{\,s^T} \otimes \vec{y}^{\,s}}_{\text{Vektorkomponenten} \ll \mathbf{n}}\right) = \vec{y}^{\,p}$$

✓

Lernalgorithmus **Hebbian-Learning:**



autoassoziatives Netz:

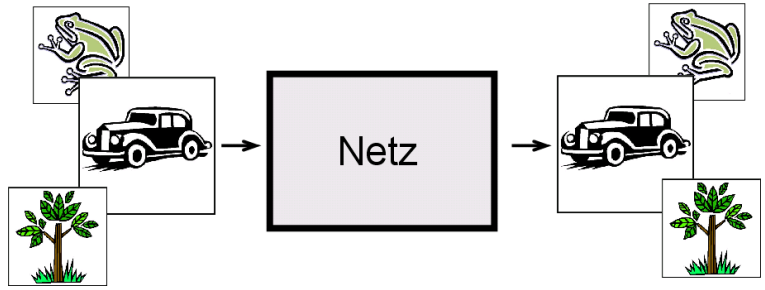
$\vec{x}^1,\ldots,\vec{x}^m$ sollen autoassoziiert werden:

Ziel: $\mathrm{sgn}\left(\vec{x}^i\mathbf{W}\right)=\vec{x}^i$ für i = 1,...,m

W = $\mathbb{1}$ unzureichend, da dies keine verrauschten Eingaben korrigiert

Was liefert das (rauschtolerante) Hebb-Verfahren ?

Lernalgorithmus **Hebbian-Learning:**



autoassoziatives Netz:

$\vec{x}^1,\ldots,\vec{x}^m$ sollen autoassoziiert werden:

$$\mathbf{W} = \vec{x}^{1^T}\vec{x}^1 + \vec{x}^{2^T}\vec{x}^2 + \ldots + \vec{x}^{m^T}\vec{x}^m$$

mit $\vec{x}^{i^T}\vec{x}^i \equiv \vec{x}^{i^T} \otimes \vec{x}^i = \begin{pmatrix} x_1^i \cdot (x_1^i,\ldots,x_n^i) \\ \vdots \\ x_n^i \cdot (x_1^i,\ldots,x_n^i) \end{pmatrix}$

es gilt: $\mathrm{sgn}\left(\vec{x}^i\mathbf{W}\right) = \vec{x}^i \quad (*)$ für i = 1,...,m

$$\mathrm{sgn}\left(\mathbf{XW}\right) = \mathbf{X}$$

$$\mathbf{X} = \begin{pmatrix} x_1^1 & \cdots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^m & \cdots & x_n^m \end{pmatrix} = \begin{pmatrix} \vec{x}^1 \\ \vdots \\ \vec{x}^m \end{pmatrix}$$

sgn(•W) stellt nichtlinearen Operator dar
(∗) entspricht Fixpunkten des Operators

Lernproblem für (auto-) Assoziativspeicher:

für gegebene Vektoren $\vec{x}^1,\ldots,\vec{x}^m$ Matrix **W** finden, dass diese Vektoren Fixpunkte des (nichtlinearen) Operators **sgn(•W)** bilden

Hebbian-Learning: **geometrische Deutung**

autoassoziatives Netz:

Matrizen $\mathbf{W}^1, \mathbf{W}^2, ..., \mathbf{W}^m$ mit $\mathbf{W}^i = \vec{x}^{i^T} \vec{x}^i$

am Beispiel $\mathbf{W}^1 = \vec{x}^{1^T} \vec{x}^1$

Eingabe in Netz: $\vec{z}$

Eingabevektor in Netz $\mathbf{W}^1$ wird in den durch Vektor $\vec{x}^1$ aufgespannten Raum projiziert:

$$\vec{z}\,\mathbf{W}^1 = \vec{z} \cdot (\vec{x}^{1^T} \vec{x}^1) = (\vec{z} \cdot \vec{x}^{1^T}) \vec{x}^1 = c_1 \vec{x}^1 \qquad c_1 \in \mathbb{R} \text{ Konstante}$$

nicht-orthogonale Projektion von $\vec{z}$ auf durch $\vec{x}^1$ aufgespannte Linie

allgemein:

$\mathbf{W} = \mathbf{W}^1 + ... + \mathbf{W}^m$ ist lineare Transformation, die **n**-dim. Vektor $\vec{z}$ auf den durch die **m** Vektoren $\vec{x}^1, ..., \vec{x}^m$ aufgespannten **m**-dim. Unterraum projiziert, da:

$$\vec{z}\,\mathbf{W} = \vec{z}\,\mathbf{W}^1 + ... + \vec{z}\,\mathbf{W}^m = c_1 \vec{x}^1 + c_2 \vec{x}^2 + ... + c_m \vec{x}^m$$

nicht-orthogonale (lineare) Projektion

Hebbian-Learning: **geometrische Deutung**

autoassoziatives Netz:

Matrizen $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^m$ mit $\mathbf{W}^i = \vec{x}^{i\,T} \vec{x}^i$

$\vec{x}^1, \ldots, \vec{x}^m$ fast orthogonal, d.h. $\vec{x}^a \cdot \vec{x}^{b\,T} \simeq 0$ für a ≠ b

verrauschte Eingabe: $\vec{z} \simeq \vec{x}^p$

Erregung:

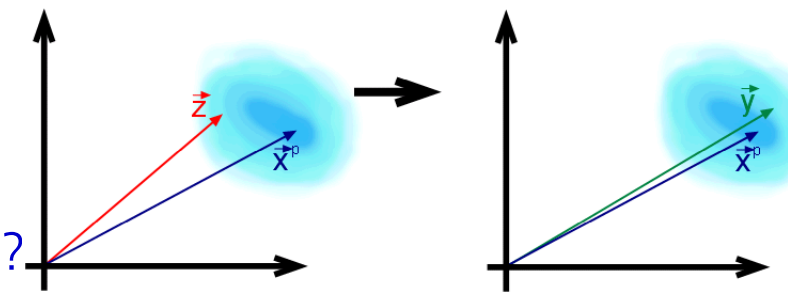
$$\vec{z}\,\mathbf{W} = \underbrace{(\vec{z} \cdot \vec{x}^{1\,T})\vec{x}^1}_{\mathbf{w}^1} + \ldots + \underbrace{(\vec{z} \cdot \vec{x}^{p\,T})\vec{x}^p}_{\mathbf{w}^p} + \ldots + \underbrace{(\vec{z} \cdot \vec{x}^{m\,T})\vec{x}^m}_{\mathbf{w}^m}$$

$$= c_1 \vec{x}^1 + \ldots + c_p \vec{x}^p + \ldots + c_m \vec{x}^m$$

klein wegen $c_i := \vec{z} \cdot \vec{x}^i \simeq \vec{x}^p \cdot \vec{x}^i \approx 0$ dominant, da $c_p := \vec{z} \cdot \vec{x}^p \simeq \vec{x}^p \cdot \vec{x}^p = n$

$$\vec{z}\,\mathbf{W} = c_p \vec{x}^p + \text{Störanteil}$$

Frage: in welchem Bereich um $\vec{x}^p$ liefert Netz bei Eingabe $\vec{z} \simeq \vec{x}^p$ einen Vektor $\vec{y}$ der näher an $\vec{x}^p$ liegt als $\vec{z}$?



Wie groß ist das **Attraktionsbecken** von $\vec{x}^p$?

Hebbian-Learning: **empirische Resultate**

- wollen **Größe der Attraktionsbecken** untersuchen
- verwenden als Abstandsmaß zwischen Bipolarvektoren **Hamming-Abstand** (Anzahl der unterschiedlichen Komponenten beider Vektoren)

wählen: **autoassoziatives Netz**
Dimension **n=10**
Anzahl der Vektoren **m = 10**: $\vec{x}^1,\ldots,\vec{x}^{10}$ zufällig gleichverteilt gewählt

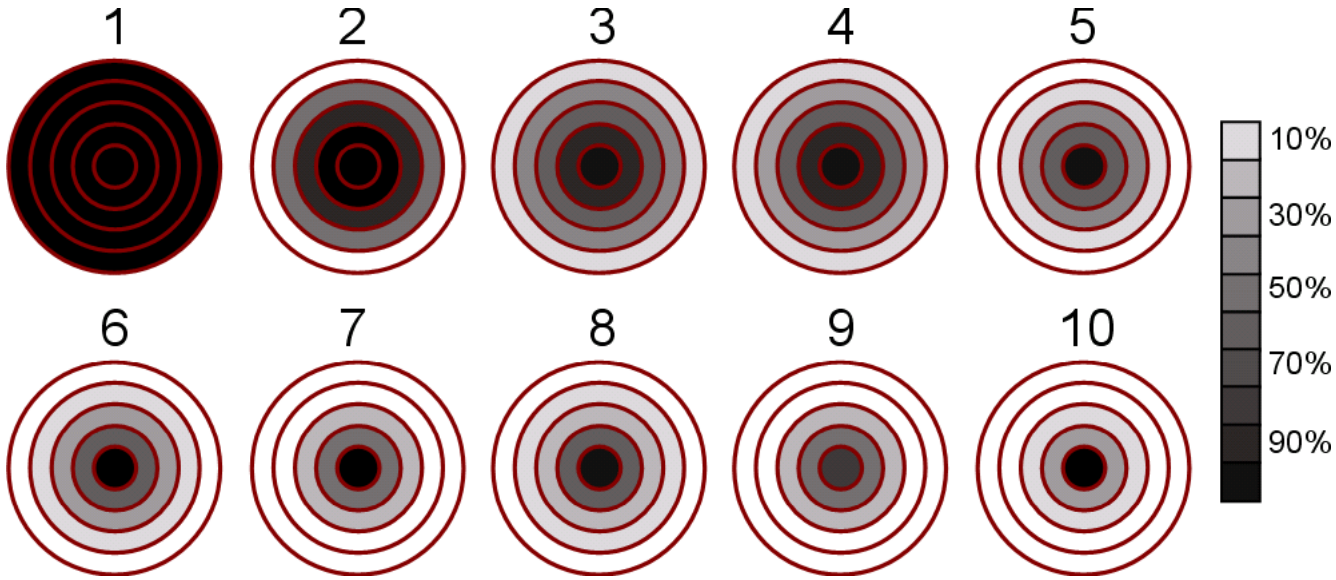
Computerexperiment:

1) $\vec{x}^1,\ldots,\vec{x}^{10}$ werden nacheinander mittels Hebb-Verfahren erlernt
2) nach jedem neu erlernten Vektor $\vec{x}^i$ (Neuberechnung der Gewichtematrix): welcher Prozentsatz der Vektoren mit Hamming-Abstand von 0 bis 5 zu bisher gespeicherten $\vec{x}^1,\ldots,\vec{x}^i$ Vektoren wird durch Netz auf diesen projiziert (wie gut werden verrauschte Eingabe erkannt)

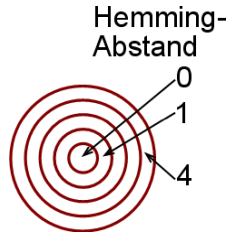
H \ i	1	2	3	4	5	6	7	8	9	10
0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	80,0	100,0
1	100,0	100,0	90,0	85,0	60,0	60,0	54,3	56,2	45,5	33,0
2	100,0	86,7	64,4	57,2	40,0	31,8	22,5	23,1	17,0	13,3
3	100,0	50,0	38,6	25,4	13,5	8,3	4,8	5,9	3,1	2,4
4	100,0	0,0	9,7	7,4	4,5	2,7	0,9	0,8	0,3	0,2
5	0,0	0,0	0,0	0,3	0,0	0,0	0,0	0,0	0,0	0,0

Hebbian-Learning: **empirische Resultate**

Änderung der Attraktionsbecken bei zunehmender Zahl i erlernter Vektoren



Prozentsatz der Vektoren $\vec{z}$, die mit einem Hamming-Abstand 0 bis 4 zu den gespeicherten Vektoren $\vec{x}^1,\ldots,\vec{x}^i$ auf diese abgebildet werden.



H \ i	1	2	3	4	5	6	7	8	9	10
0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	100,0	80,0	100,0
1	100,0	100,0	90,0	85,0	60,0	60,0	54,3	56,2	45,5	33,0
2	100,0	86,7	64,4	57,2	40,0	31,8	22,5	23,1	17,0	13,3
3	100,0	50,0	38,6	25,4	13,5	8,3	4,8	5,9	3,1	2,4
4	100,0	0,0	9,7	7,4	4,5	2,7	0,9	0,8	0,3	0,2
5	0,0	0,0	0,0	0,3	0,0	0,0	0,0	0,0	0,0	0,0

Hebbian-Learning: **empirische Resultate**

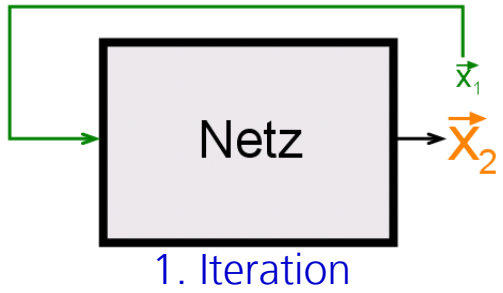
Verbesserungsmöglichkeiten zur Erkennung verrauschter Eingaben

- iterative Anwendung des Netzes
  auch bezeichnet als **Netz mit Rückkopplung**

Hebbian-Learning: **empirische Resultate**

Verbesserungsmöglichkeiten zur Erkennung verrauschter Eingaben

- iterative Anwendung des Netzes
  auch bezeichnet als **Netz mit Rückkopplung**



1. Iteration

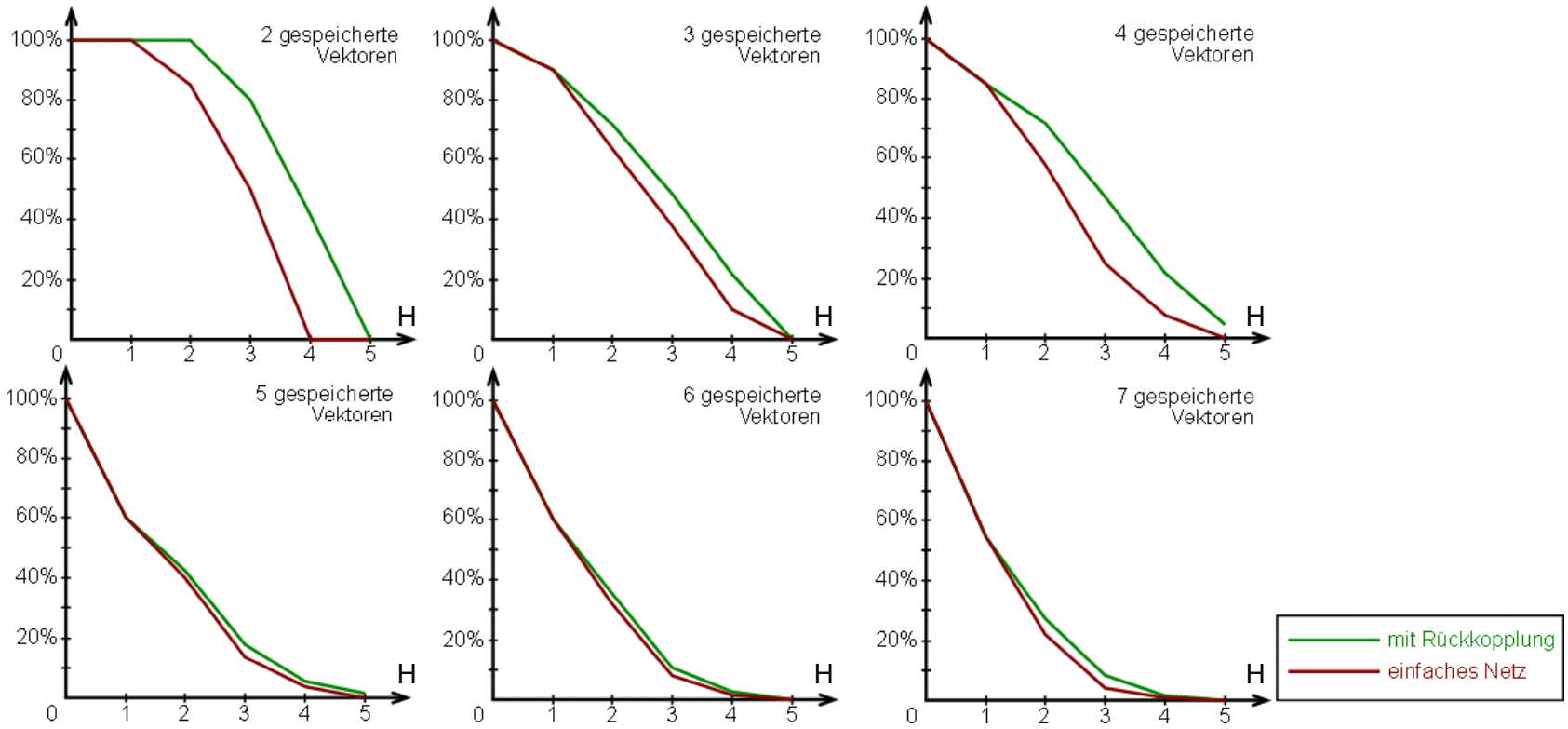
Experiment mit zuvor verwendeten Vektoren liefert nun:

H \ i	1	2	3	4	5	6	7
0	100,0	100,0	100,0	100,0	100,0	100,0	100,0
1	100,0	100,0	90,0	85,0	60,0	60,0	54,3
2	100,0	100,0	72,6	71,1	41,8	34,8	27,9
3	100,0	80,0	48,6	47,5	18,3	10,8	8,5
4	100,0	42,8	21,9	22,3	6,8	3,7	2,0
5	0,0	0,0	0,7	4,6	0,8	0,0	0,1

Prozentsatz der Vektoren $\vec{z}$, die mit einem Hamming-Abstand 0 bis 4 zu den gespeicherten Vektoren $\vec{x}^1,\ldots,\vec{x}^i$ nach 5 Iterationen auf diese abgebildet werden.

Hebbian-Learning: **empirische Resultate**

Vergleich der Erkennungsleistung von Netzen mit und ohne Rückkopplung:



Rekursives Netz:

- deutlich bessere Leistung bei kleiner Zahl von gespeicherten Vektoren
- für große Zahl ($m \precsim n$) gespeicherter Vektoren kaum Unterschiede

Hebbian-Learning: **empirische Resultate**

Ziel: Erkennungsleistung in Abhängigkeit der gespeicherten Vektoren **i**

benötigen: allgemeines Maß I für Erkennungsleistung bei festem **i**

Sei:

- **H** Hamming-Abstand zu einem der gespeicherten Vektoren
- p_H Prozentsatz der Vektoren mit Hamming-Abstand **H**, die erkannt werden

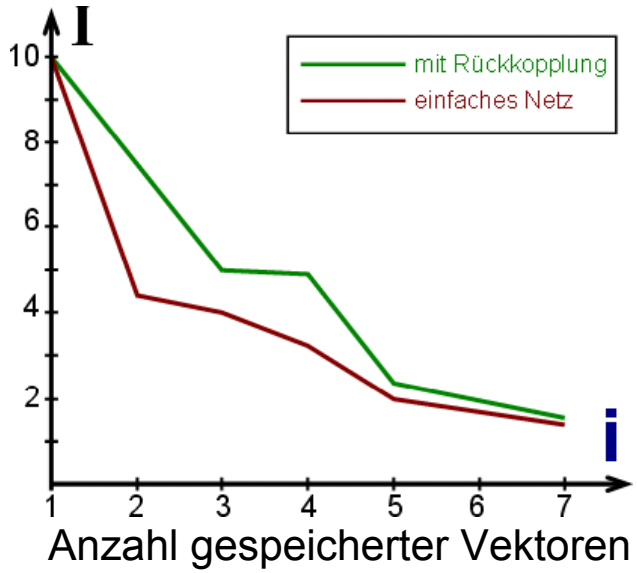
Anforderungen:

- Erkennung eines stark verrauschten Vektors (**H** groß) ist höher zu bewerten, als Erkennung eines wenig verrauschten Vektors (**H** klein)

Maß: $$I = \sum_{H=0}^{5} H p_H$$

gewichtet mit Stärke der Verrauschung **H**

Anteil erkannter Vektoren



Hebbian-Learning: **empirische Resultate**

Letzte Bemerkungen zum Hebb-Verfahren:

Attraktionsbecken reduzieren sich schnell mit steigendem **i**, warum?

- sei $\vec{x}$ gespeicherter Vektor: $\mathrm{sgn}(\vec{x}\mathbf{W}) = \vec{x}$ aber $\mathrm{sgn}(-\vec{x}\mathbf{W}) = -\vec{x}$

  $\vec{x}$ ist Fixpunkt $-\vec{x}$ ist auch Fixpunkt

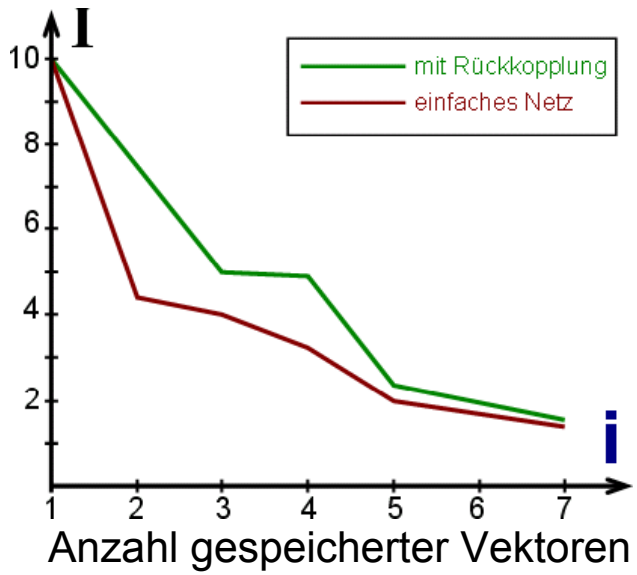
  aus **i** gespeicherten Vektoren entstehen **2i** Fixpunkte

- zudem entstehen auch sog. **falsche Fixpunkte**, die zwar $\mathrm{sgn}(\vec{v}\mathbf{W}) = \vec{v}$ erfüllen, $\vec{v}$ aber kein erlernter Vektor ist

Alle Fixpunkte müssen sich den zur Verfügung stehenden Raum teilen
⇒ kleinere Attraktionsbecken

gespeicherte Vektoren	2	3	4	5	6	7
negative Fixpunkte	2	3	4	5	6	7
falsche Fixpunkte	2	4	8	16	24	45
insgesamt	4	7	12	21	30	52

statistische Zahl unerwünschter Fixpunkte



beyond Hebbian-Learning

- Hebb-Verfahren liefert nur gute Ergebnisse bei fast **orthogonalen** Speichervektoren $\vec{x}^1,\ldots,\vec{x}^m$
- bei realen Anwendungen oft korrelierte Daten
  z.B. Schrifterkennung: Buchstaben Q und O nahezu identisch

  Ausgabe: $$\mathrm{sgn}(\vec{x}^i \mathbf{W}) = \mathrm{sgn}\left(\vec{y}^i + \underbrace{\sum_{s \neq i} \vec{y}^s \frac{(\vec{x}^s \cdot \vec{x}^i)}{(\vec{x}^i \cdot \vec{x}^i)}}_{\text{crosstalk}} \right)$$

  crosstalk wird zu gross, weil $\vec{x}^s \cdot \vec{x}^p$ nicht für alle s≠p klein genug ist

Frage: gibt es eine andere Matrix W, so dass cosstalk's möglichst klein werden ?

sog. **Pseudoinverse Matrix** minimiert Summe aller crosstalk's

Definition: es sei $\mathbf{X}$ eine reelle m×n-Matrix
man bezeichnet $\mathbf{X}^+$ als Pseudoinverse Matrix zu $\mathbf{X}$, falls

a) $\mathbf{X}\mathbf{X}^+\mathbf{X} = \mathbf{X}$
b) $\mathbf{X}^+\mathbf{X}\mathbf{X}^+ = \mathbf{X}^+$
c) $\mathbf{X}\mathbf{X}^+$ und $\mathbf{X}^+\mathbf{X}$ sind symmetrisch

Anmerkung: die Pseudoinverse ist Verallgemeinerung der inversen Matrix

beyond Hebbian-Learning

quadratische Norm einer Matrix $\mathbf{A}$: $\|\mathbf{A}\|^2 := Tr(\mathbf{A}\mathbf{A}^T)$

Es gilt: $\|\mathbf{A}\|^2 = \sum_{i=1}^{m} \vec{a}^i \cdot \vec{a}^{i^T} = \sum_{i=1}^{m} \|\vec{a}^i\|^2$

für: $\mathbf{A} = \begin{pmatrix} \vec{a}^1 \\ \vdots \\ \vec{a}^m \end{pmatrix} = \begin{pmatrix} a_1^1 & \cdots & a_n^1 \\ \vdots & \ddots & \vdots \\ a_1^m & \cdots & a_n^m \end{pmatrix}$

$$\mathbf{A}^T = \begin{pmatrix} \vec{a}^{1^T} & \cdots & \vec{a}^{m^T} \end{pmatrix}$$

$$\mathbf{A}\mathbf{A}^T = \begin{pmatrix} \vec{a}^1 \\ \vec{a}^2 \\ \vdots \\ \vec{a}^m \end{pmatrix} \cdot \begin{pmatrix} \vec{a}^{1^T} & \vec{a}^{2^T} & \cdots & \vec{a}^{m^T} \end{pmatrix} = \begin{pmatrix} \vec{a}^1\vec{a}^{1^T} & \vec{a}^1\vec{a}^{2^T} & \cdots & \vec{a}^1\vec{a}^{m^T} \\ \vec{a}^2\vec{a}^{1^T} & \vec{a}^2\vec{a}^{2^T} & & \\ \vdots & & \ddots & \\ \vec{a}^m\vec{a}^{1^T} & & & \vec{a}^m\vec{a}^{m^T} \end{pmatrix}$$

beyond Hebbian-Learning

quadratische Norm einer Matrix $\mathbf{A}$: $\|\mathbf{A}\|^2 := Tr(\mathbf{A}\mathbf{A}^T)$

Es gilt: $\|\mathbf{A}\|^2 = \sum_{i=1}^{m} \vec{a}^i \cdot \vec{a}^{iT} = \sum_{i=1}^{m} \|\vec{a}^i\|^2$ für: $\mathbf{A} = \begin{pmatrix} \leftarrow \vec{a}^1 \rightarrow \\ \vdots \\ \leftarrow \vec{a}^m \rightarrow \end{pmatrix} = \begin{pmatrix} a_1^1 & \cdots & a_n^1 \\ \vdots & \ddots & \vdots \\ a_1^m & \cdots & a_n^m \end{pmatrix}$

Satz: Sei $\mathbf{X}$ eine reelle **m**×**n**-Matrix und $\mathbf{Y}$ eine reelle **m**×**k**-Matrix.
Die **m**×**k**-Matrix $\mathbf{W} = \mathbf{X}^+\mathbf{Y}$ minimiert die quadratische Norm der Matrix $\mathbf{X}\mathbf{W} - \mathbf{Y}$ (mit $\mathbf{X}^+$ pseudoinverse Matrix zu $\mathbf{X}$).

damit: $\vec{y}^i = \vec{x}^i\mathbf{W} - \left(\vec{x}^i\mathbf{W} - \vec{y}^i\right)$

$\vec{x}^i\mathbf{W} = \vec{y}^i + \left(\vec{x}^i\mathbf{W} - \vec{y}^i\right)$

Ausgabe: $\mathrm{sgn}(\vec{x}^i\mathbf{W}) = \mathrm{sgn}\left(\vec{y}^i + \underbrace{\left(\vec{x}^i\mathbf{W} - \vec{y}^i\right)}_{=:\vec{c}^i}\right)$

crosstalk zu Vektor i

aus Vergleich mit

$\mathrm{sgn}(\vec{x}^i\mathbf{W}) = \mathrm{sgn}\left(\vec{y}^i + \underbrace{\sum_{s\neq i} \vec{y}^s \frac{(\vec{x}^s \cdot \vec{x}^i)}{(\vec{x}^i \cdot \vec{x}^i)}}_{\text{crosstalk}}\right)$

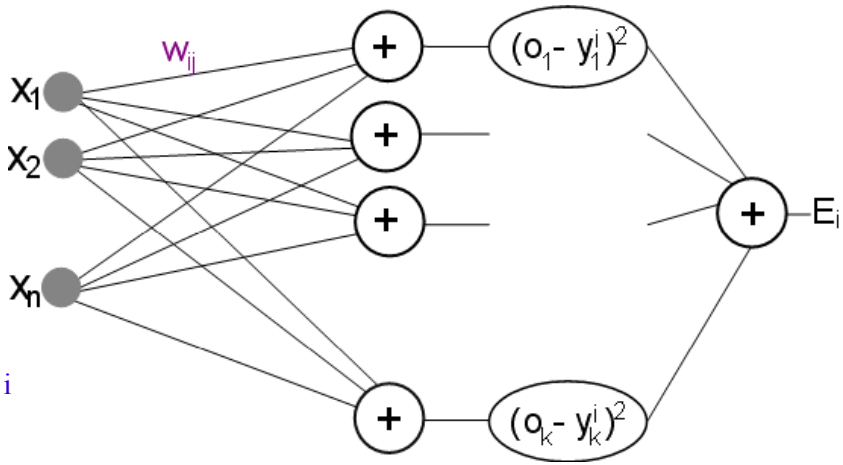
für $\mathbf{X}\mathbf{W} - \mathbf{Y} = \begin{pmatrix} \vec{x}^1\mathbf{W} - \vec{y}^1 \\ \vdots \\ \vec{x}^m\mathbf{W} - \vec{y}^m \end{pmatrix}$ folgt: $E = \|\mathbf{X}\mathbf{W} - \mathbf{Y}\|^2 = \sum_{i=1}^{m} \|\vec{x}^i\mathbf{W} - \vec{y}^i\|^2$

$\mathbf{W} = \mathbf{X}^+\mathbf{Y}$ minimiert Gesamtsumme aller Störterme eines Assoziativspeichers

beyond Hebbian-Learning

Berechnung der pseudoinversen Matrix

- effiziente Algorithmen aus linearer Algebra bekannt
- mittels eines Backpropagation-Netzes:
  - für Eingabe $\vec{x}^i$ wird Ausgabe des Netzes mit Komponenten des Vektors $\vec{y}^i$ verglichen und quadratischer Fehler E_i berechnet
  - gesamter quadratischer Fehler $E = \sum_i E_i$ ist Norm von $\mathbf{XW} - \mathbf{Y}$
  - mittels Backpropagation kann Matrix $\mathbf{W}$ gefunden werden die $\|\mathbf{XW} - \mathbf{Y}\|^2$ minimiert



Zusammenfassung:

Lösungen des Lernproblems für Assoziativspeicher :

- Hebbian-Learning: $\mathbf{W} = \mathbf{W}^1 + \mathbf{W}^2 + \ldots + \mathbf{W}^m$ mit $\mathbf{W}^i = \vec{x}^{i^T} \otimes \vec{y}^i$
  arbeitet nur bei hinreichend orthogonalen Vektoren $\vec{x}^1, \ldots, \vec{x}^m$

- Hebbian-Learning mit Rückkopplung:
  Iterative Anwendung des mit Hebb-Verfahren trainierten Netzes
  liefert bessere Rauschtoleranz bei kleiner Zahl von erlernten Vektoren

- Verfahren der pseudoinversen Matrix: $\mathbf{W} = \mathbf{X}^+ \mathbf{X}$
  arbeitet auch auf nicht-orthogonalen (korrellierten) Vektoren $\vec{x}^1, \ldots, \vec{x}^m$
  aber liefert i.a. schlechtere Resultate bei orthogonalen Vektoren

Ende