Neuronale Netze
Hopfield-Netze

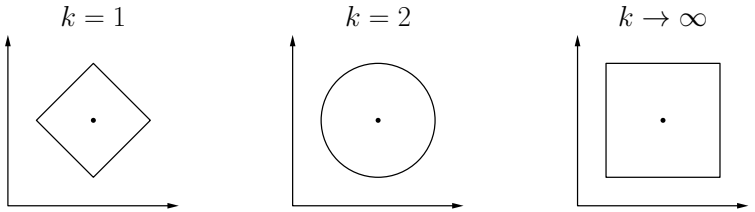
Prof. Dr.-Ing. Sebastian Stober

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

FACULTY OF
COMPUTER SCIENCE

Recap: RBF-Netze

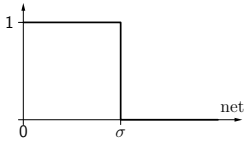
f_{net}: Abstandsfunktion



f_{act}: radiale Aktivierungsfunktion

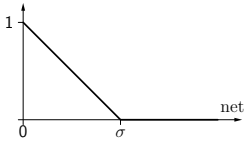
Rechteckfunktion:

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



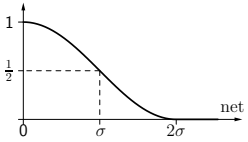
Dreiecksfunktion:

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



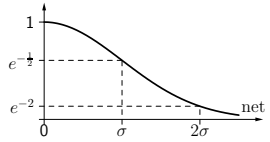
Kosinus bis Null:

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

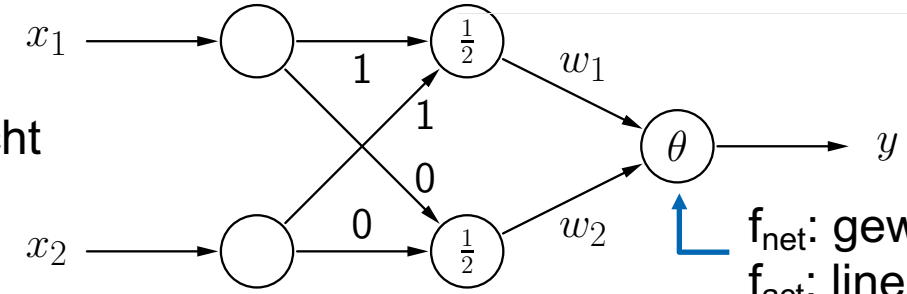


Gaußsche Funktion:

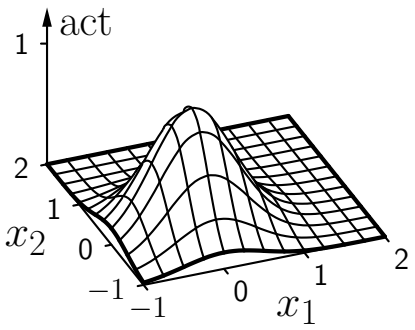
$$f_{act}(net, \sigma) = e^{-\frac{net^2}{2\sigma^2}}$$



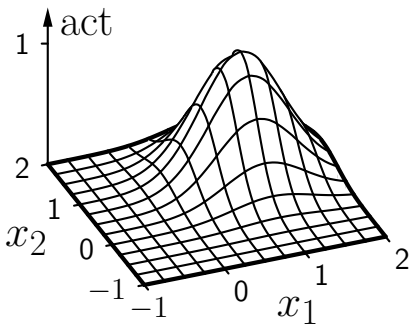
nur 1 versteckte Schicht



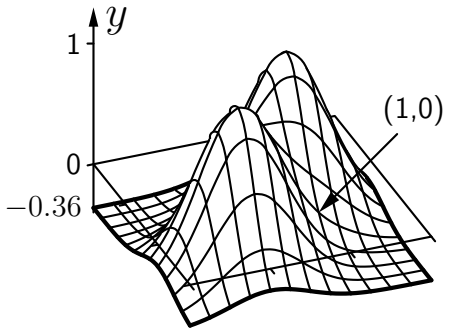
f_{net}: gewichtete Summe
f_{act}: lineare Funktion



basis function (0,0)



basis function (1,1)

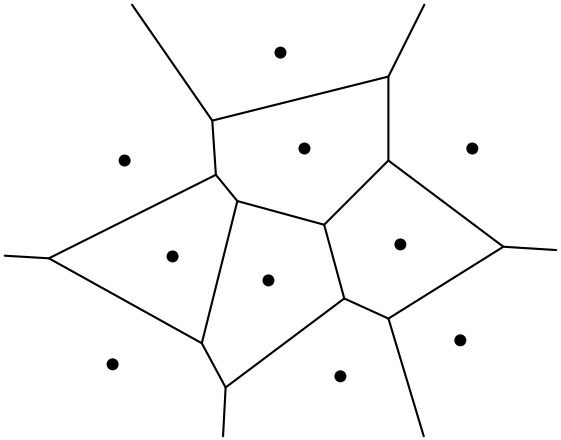


output

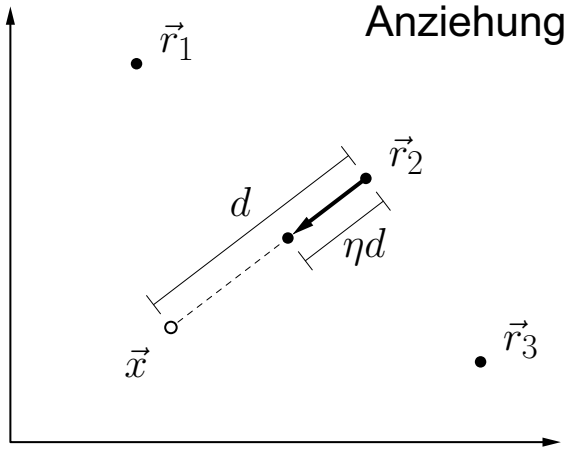
Recap: LVQ

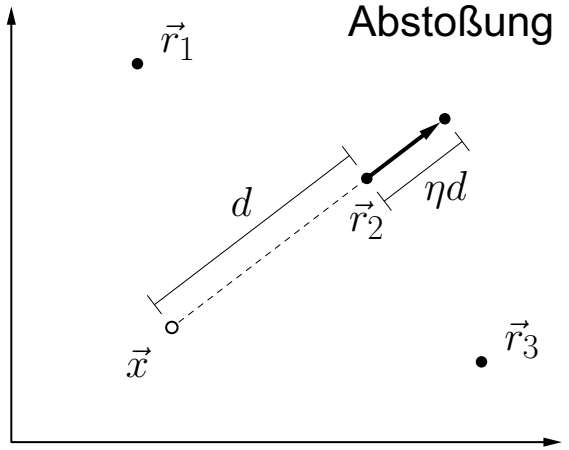
- wie RBF-Netz
  ohne Ausgabeschicht
- Winner-Takes-All
  Ausgabefunktion

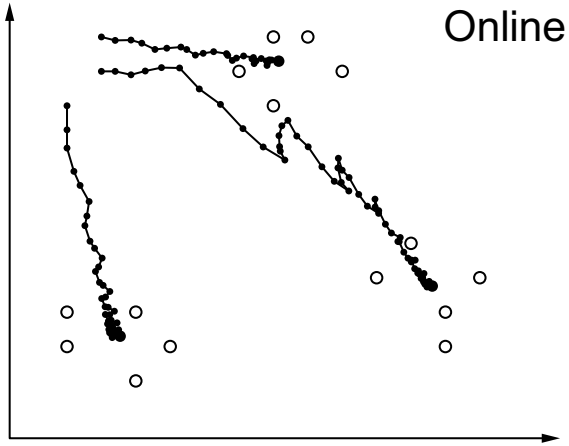
Voronoi-Diagramm
=> Quantisierung

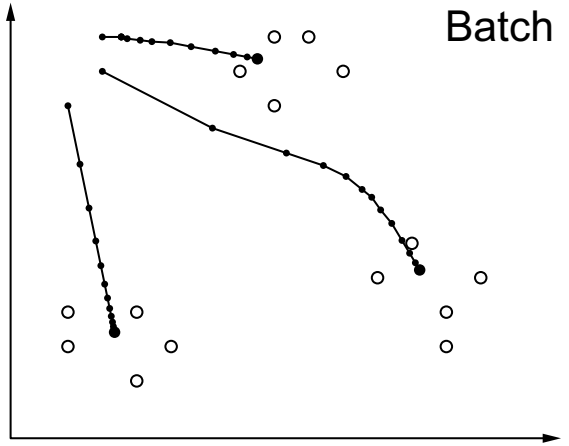


Anpassung der Refenrenzvektoren

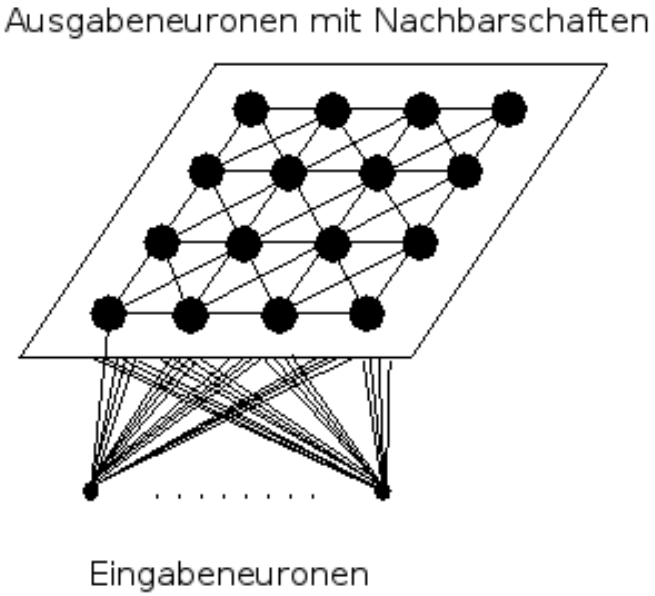








Recap: SOMs



Finde topologieerhaltende Abbildung durch Beachtung der Nachbarschaft

Anpassungsregel für Referenzvektor:

$$\vec{r}_u^{(\text{new})} = \vec{r}_u^{(\text{old})} + \eta(t) \cdot f_{\text{nb}}(d_{\text{neurons}}(u, u_*), \varrho(t)) \cdot (\vec{x} - \vec{r}_u^{(\text{old})}),$$

u_* ist das Gewinnerneuron (Referenzvektor am nächsten zum Datenpunkt).

Die Funktion f_{nb} ist eine radiale Funktion.

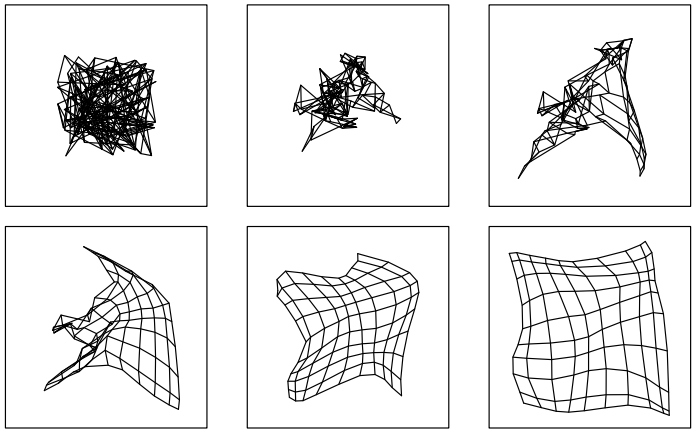
Zeitabhängige Lernrate

$$\eta(t) = \eta_0 \alpha_\eta^t, \quad 0 < \alpha_\eta < 1, \qquad \text{oder} \qquad \eta(t) = \eta_0 t^{\kappa_\eta}, \quad \kappa_\eta > 0.$$

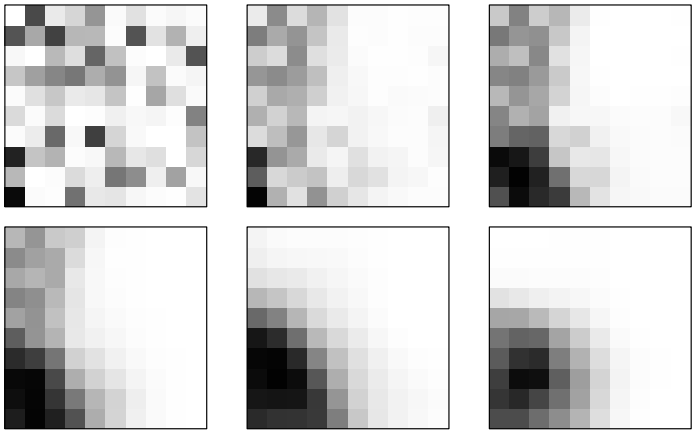
Zeitabhängiger Nachbarschaftsradius

$$\varrho(t) = \varrho_0 \alpha_\varrho^t, \quad 0 < \alpha_\varrho < 1, \qquad \text{oder} \qquad \varrho(t) = \varrho_0 t^{\kappa_\varrho}, \quad \kappa_\varrho > 0.$$

Visualisierung (2D-Eingaberaum)
Referenzvektoren => “Entfaltung”



Visualisierung (2D-Ausgaberaum)
Aktivierung für eine Beispieleingabe



Definition

Ein **Hopfield-Netz** ist ein neuronales Netz mit einem Graphen $G = (U, C)$, das die folgenden Bedingungen erfüllt:

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

(ii) $C = U \times U - \{(u, u) \mid u \in U\}$.

In einem Hopfield-Netz sind alle Neuronen sowohl Eingabe- als auch Ausgabeneuronen.

Es gibt keine versteckten Neuronen.

Jedes Neuron erhält seine Eingaben von allen anderen Neuronen.

Ein Neuron ist nicht mit sich selbst verbunden.

Die Verbindungsgewichte sind symmetrisch, d.h.

$$\forall u, v \in U, u \neq v : \qquad w_{uv} = w_{vu}.$$

Definition

Die Netzeingabefunktion jedes Neurons ist die gewichtete Summe der Ausgaben aller anderen Neuronen, d.h.

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

Die Aktivierungsfunktion jedes Neurons ist eine Sprungfunktion, d.h.

$$\forall u \in U: \quad f_{\text{act}}^{(u)}(\text{net}_u, \theta_u) = \begin{cases} 1, & \text{falls} \quad \text{net}_u \geq \theta_u, \\ -1, & \text{sonst.} \end{cases}$$

Die Ausgabefunktion jedes Neurons ist die Identität, d.h.

$$\forall u \in U: \quad f_{\text{out}}^{(u)}(\text{act}_u) = \text{act}_u \, .$$

Definition

Alternative Aktivierungsfunktion

$$\forall u \in U: \quad f_{\text{act}}^{(u)}(\text{net}_u, \theta_u, \text{act}_u) = \begin{cases} 1, & \text{falls} \quad \text{net}_u > \theta, \\ -1, & \text{falls} \quad \text{net}_u < \theta, \\ \text{act}_u, & \text{falls} \quad \text{net}_u = \theta. \end{cases}$$

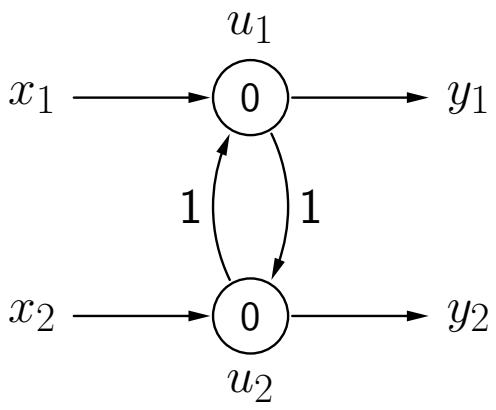
Diese Aktivierungsfunktion bietet einige Vorteile bei der späteren physikalischen Interpretation eines Hopfield-Netzes. Diese wird allerdings in der Vorlesung nicht weiter genutzt.

Allgemeine Gewichtsmatrix eines Hopfield-Netzes

$$\mathbf{W} = \begin{pmatrix} 0 & w_{u_1u_2} & \dots & w_{u_1u_n} \\ w_{u_1u_2} & 0 & \dots & w_{u_2u_n} \\ \vdots & \vdots & & \vdots \\ w_{u_1u_n} & w_{u_1u_n} & \dots & 0 \end{pmatrix}$$

Beispiele

Sehr einfaches Hopfield-Netz



$$\mathbf{W} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Das Verhalten eines Hopfield-Netzes kann von der Update-Reihenfolge abhängen.

Die Berechnungen können oszillieren, wenn Neuronen synchron aktualisiert werden.

Die Berechnung konvergiert immer, wenn die Neuronen asynchron in fester Reihenfolge aktualisiert werden.

Beispiele

Parallele Aktualisierung der Neuronenaktivierungen

	u_1	u_2
Eingabephase	$\mathbf{-1}$	$\mathbf{1}$
Arbeitsphase	$\mathbf{1}$	$\mathbf{-1}$
	$\mathbf{-1}$	$\mathbf{1}$
	$\mathbf{1}$	$\mathbf{-1}$
	$\mathbf{-1}$	$\mathbf{1}$
	$\mathbf{1}$	$\mathbf{-1}$
	$\mathbf{-1}$	$\mathbf{1}$

Die Berechnungen oszillieren, kein stabiler Zustand wird erreicht.

Die Ausgabe hängt davon ab, wann die Berechnung abgebrochen wird.

Beispiele

Sequentielle Aktualisierung der Neuronenaktivierungen

	u_1	u_2
Eingabephase	$\mathbf{-1}$	$\mathbf{1}$
Arbeitsphase	$\mathbf{1}$	1
	1	$\mathbf{1}$
	$\mathbf{1}$	1
	1	$\mathbf{1}$

	u_1	u_2
Eingabephase	$\mathbf{-1}$	$\mathbf{1}$
Arbeitsphase	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1
	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1

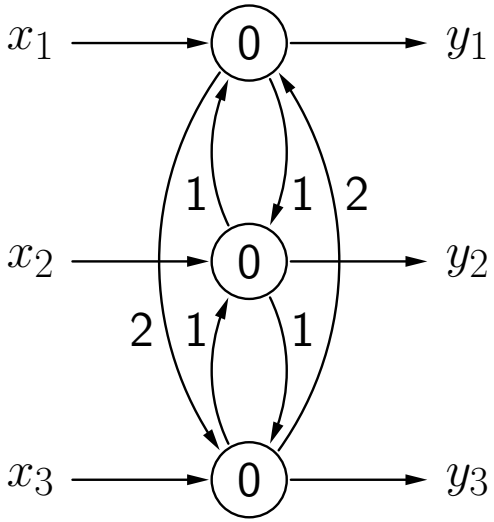
Aktivierungsreihenfolge u_1, u_2, u_1, ... bzw. u_2, u_1, u_1, ...

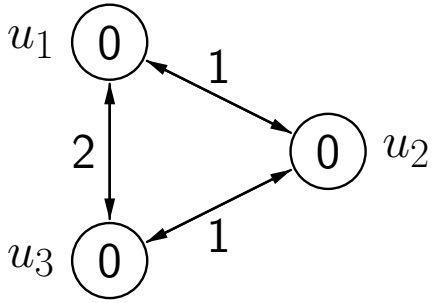
Unabhängig von der Reihenfolge wird ein stabiler Zustand erreicht.

Welcher Zustand stabil ist, hängt von der Reihenfolge ab.

Beispiele

Vereinfachte Darstellung eines Hopfield-Netzes





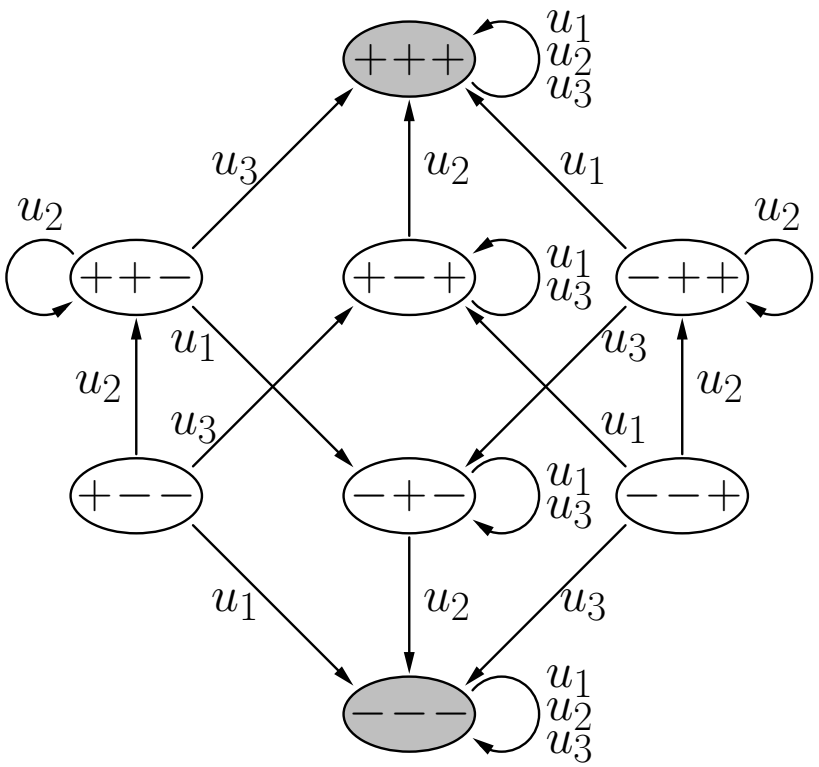
$$\mathbf{W} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

Symmetrische Verbindungen zwischen Neuronen werden kombiniert.

Eingaben und Ausgaben werden nicht explizit dargestellt.

Zustandsgraph

Graph der Aktivierungen und Übergänge



+/- Aktivierung der Neuronen:
+ entspricht $+1$, − entspricht -1

Pfeile: geben die Neuronen an, deren Aktualisierung zu dem jeweiligen Zustandsübergang führt

grau unterlegte Zustände: stabile Zustände

beliebige Aktualisierungsreihenfolgen ablesbar

(Zustandsgraph zum Netz auf der vorherigen Folie und Erläuterung)

Konvergenz der Berechnungen

Konvergenztheorem: Wenn die Aktivierungen der Neuronen eines Hopfield-Netzes asynchron (sequentiell) durchgeführt werden, wird ein stabiler Zustand nach einer endlichen Anzahl von Schritten erreicht.

Wenn die Neuronen zyklisch in einer beliebigen, aber festen Reihenfolge durchlaufen werden, sind höchstens $n \cdot 2^n$ Schritte (Aktualisierungen einzelner Neuronen) notwendig, wobei n die Anzahl der Neuronen im Netz ist.

Der Beweis erfolgt mit Hilfe einer **Energiefunktion**.
Die Energiefunktion eines Hopfield-Netzes mit n Neuronen $u_1, \ldots, u_n$ ist

$$\begin{aligned} E &= -\frac{1}{2} \vec{\mathrm{act}}^{\top} \mathbf{W} \vec{\mathrm{act}} + \vec{\theta}^{T} \vec{\mathrm{act}} \\ &= -\frac{1}{2} \sum_{u,v \in U, u \neq v} w_{uv} \, \mathrm{act}_u \, \mathrm{act}_v + \sum_{u \in U} \theta_u \, \mathrm{act}_u \, . \end{aligned}$$

wegen Symmetrie

Konvergenz

Man betrachte die Energieänderung die aus einer aktivierungsändernden Aktualisierung entsteht:

$$\begin{aligned}
\Delta E = E^{(\text{new})} - E^{(\text{old})} &= \Big(-\sum_{v \in U - \{u\}} w_{uv}\, \text{act}_u^{(\text{new})}\, \text{act}_v + \theta_u\, \text{act}_u^{(\text{new})} \Big) \\
&- \Big(-\sum_{v \in U - \{u\}} w_{uv}\, \text{act}_u^{(\text{old})}\, \text{act}_v + \theta_u\, \text{act}_u^{(\text{old})} \Big) \\
&= \left(\text{act}_u^{(\text{old})} - \text{act}_u^{(\text{new})} \right) \Big(\underbrace{\sum_{v \in U - \{u\}} w_{uv}\, \text{act}_v}_{= \text{net}_u} - \theta_u \Big).
\end{aligned}$$

$\text{net}_u < \theta_u$: Zweiter Faktor kleiner als 0.
$\text{act}_u^{(\text{new})} = -1$ and $\text{act}_u^{(\text{old})} = 1$, daher erster Faktor größer als 0.
Ergebnis: $\Delta E < 0$.

$\text{net}_u \geq \theta_u$: Zweiter Faktor größer als oder gleich 0.
$\text{act}_u^{(\text{new})} = 1$ und $\text{act}_u^{(\text{old})} = -1$, daher erster Faktor kleiner als 0.
Ergebnis: $\Delta E \leq 0$.

erhöht Anzahl +1-Aktivierungen um 1

Konvergenz

Höchstens $n \cdot 2^n$ Schritte bis zur Konvergenz:

die beliebige, aber feste Reihenfolge sorgt dafür, dass alle Neuronen zyklisch durchlaufen und neuberechnet werden

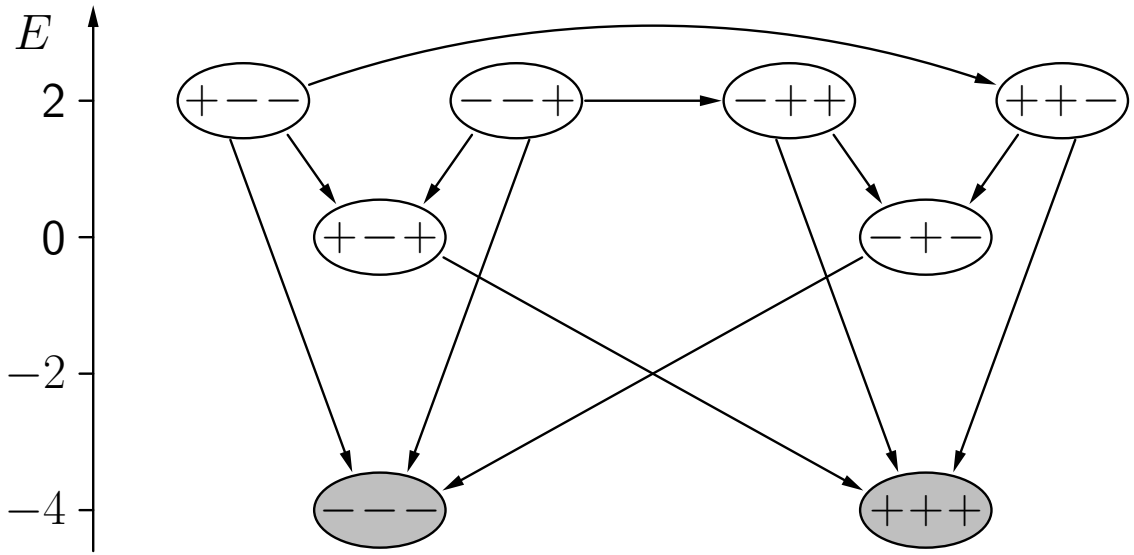
- a) es ändert sich keine Aktivierung – ein stabiler Zustand wurde erreicht
- b) es ändert sich mindestens eine Aktivierung: dann wurde damit mindestens einer der 2^n möglichen Aktivierungszustände ausgeschlossen.

Ein einmal verlassener Zustand kann nicht wieder erreicht werden (siehe vorherige Folien).

D.h. nach spätestens 2^n Durchläufen durch die n Neuronen ist ein stabiler Zustand erreicht.

Beispiele (Netz siehe Folie 11)

Ordne die Zustände im Graphen entsprechend ihrer Energie

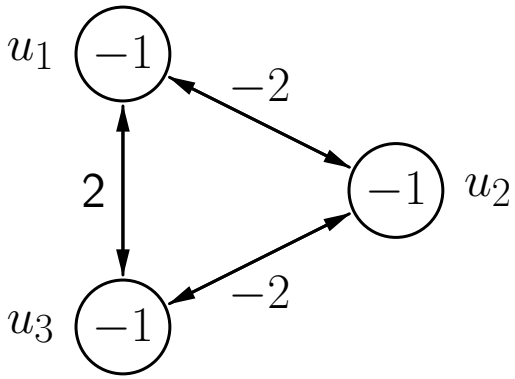


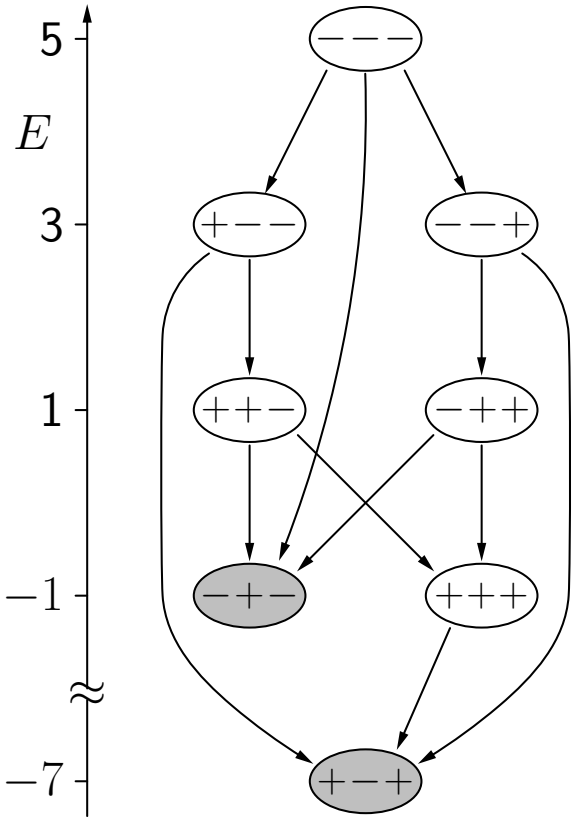
Energiefunktion für das Beispiel-Hopfield-Netz:

$$E = -\operatorname{act}_{u_1}\operatorname{act}_{u_2} - 2\operatorname{act}_{u_1}\operatorname{act}_{u_3} - \operatorname{act}_{u_2}\operatorname{act}_{u_3}\,.$$

Beispiele

Der Zustandsgraph muss nicht symmetrisch sein





Physikalische Interpretation

Physikalische Interpretation: Magnetismus

Ein Hopfield-Netz kann als (mikroskopisches) Modell von Magnetismus gesehen werden
(sogenanntes Ising-Modell, [Ising 1925]).

physikalisch	neuronal
Atom	Neuron
Magnetisches Moment (Spin)	Aktivierungszustand
Stärke des äußeren Magnetfeldes	Schwellenwert
Magnetische Kopplung der Atome	Verbindungsgewichte
Hamilton-Operator des Magnetfeldes	Energiefunktion

Assoziativspeicher

Idee: Nutze stabile Zustände, um Muster zu speichern

Zuerst: Speichere nur ein Muster $\vec{x} = (\mathrm{act}_{u_1}^{(l)}, \ldots, \mathrm{act}_{u_n}^{(l)})^\top \in \{-1, 1\}^n$, $n \geq 2$,
d.h. finde Gewichte, so dass der Zustand ein stabiler Zustand wird.

Notwendige und hinreichende Bedingung:

$$S(\mathbf{W}\vec{x} - \vec{\theta}\,) = \vec{x},$$

wobei

$$\begin{array}{rcll} S : \mathbb{R}^n & \to & \{-1, 1\}^n, & \text{elementweise} \\ \vec{x} & \mapsto & \vec{y} & \text{Schwellenwertfunktion} \end{array}$$

mit

$$\forall i \in \{1, \ldots, n\}: \quad y_i = \begin{cases} 1, & \text{falls } x_i \geq 0, \\ -1, & \text{sonst.} \end{cases}$$

Assoziativspeicher

Falls $\vec{\theta} = \vec{0}$, dann kann eine passende Matrix $\mathbf{W}$ leicht berechnet werden. Es reicht eine Matrix $\mathbf{W}$ zu finden mit

$$\mathbf{W}\vec{x} = c\vec{x} \qquad \text{mit } c \in \mathbb{R}^+.$$

Algebraisch: Finde eine Matrix $\mathbf{W}$ die einen positiven Eigenwert in Bezug auf $\vec{x}$ hat.

Wähle

setze Diagonalwerte auf 0

$$\mathbf{W} = \vec{x}\vec{x}^{\,T} - \mathbf{E}$$

wobei $\vec{x}\vec{x}^{\,T}$ das sogenannte **äußere Produkt** von $\vec{x}$ mit sich selbst ist.

Mit dieser Matrix erhalten wir

Matrix- und Vektormult. assoziativ

$$\begin{aligned} \mathbf{W}\vec{x} &= (\vec{x}\vec{x}^{\,T})\vec{x} - \underbrace{\mathbf{E}\vec{x}}_{=\vec{x}} \overset{(*)}{=} \vec{x}\,\underbrace{(\vec{x}^{\,T}\vec{x})}_{=|\vec{x}|^2=n} - \vec{x} \\ &= n\vec{x} - \vec{x} = (n-1)\vec{x}. \\ &\qquad = c \end{aligned}$$

(n ≥ 2)

Assoziativspeicher

Hebb'sche Lernregel [Hebb 1949]

In einzelnen Gewichten aufgeschrieben lautet die Berechnung der Gewichtsmatrix wie folgt:

$$w_{uv} = \begin{cases} 0, & \text{falls } u = v, \\ 1, & \text{falls } u \neq v,\ \mathrm{act}_u^{(p)} = \mathrm{act}_u^{(v)}, \\ -1, & \text{sonst.} \end{cases}$$

Ursprünglich von biologischer Analogie abgeleitet.

Verstärkt Verbindungen zwischen Neuronen, die zur selben Zeit aktiv sind.

Diese Lernregel speichert auch das Komplement des Musters:

Mit $\mathbf{W}\vec{x} = (n-1)\vec{x}$ ist daher auch $\mathbf{W}(-\vec{x}) = (n-1)(-\vec{x})$.

Assoziativspeicher

Speichern mehrerer Muster (m < n)

Wähle

$$\mathbf{W}\vec{x}_j = \sum_{i=1}^{m} \mathbf{W}_i \vec{x}_j = \left(\sum_{i=1}^{m} (\vec{x}_i \vec{x}_i^{\,T}) \vec{x}_j \right) - m \underbrace{\mathbf{E}\vec{x}_j}_{=\vec{x}_j}$$

$$= \left(\sum_{i=1}^{m} \vec{x}_i (\vec{x}_i^{\,T} \vec{x}_j) \right) - m\vec{x}_j$$

Wenn die Muster orthogonal sind, gilt

$$\vec{x}_i^{\,T} \vec{x}_j = \begin{cases} 0, & \text{falls} \quad i \neq j, \\ n, & \text{falls} \quad i = j, \end{cases}$$

Skalarprodukt orthogonaler Vektoren verschwindet

und daher

$$\mathbf{W}\vec{x}_j = (n - m)\vec{x}_j.$$

Assoziativspeicher

Speichern mehrerer Muster

Ergebnis: So lange $m < n$, ist $\vec{x}_j$ ein stabiler Zustand des Hopfield-Netzes.

Man beachte, dass die Komplemente der Muster ebenfalls gespeichert werden.

Mit $\mathbf{W}\vec{x}_j = (n-m)\vec{x}_j$ ist daher auch $\mathbf{W}(-\vec{x}_j) = (n-m)(-\vec{x}_j)$.

Aber: die Speicherkapazität ist verglichen mit der Anzahl möglicher Zustände sehr klein (2^n).

Assoziativspeicher

Nicht-orthogonale Muster:

$$\mathbf{W}\vec{x}_j = (n-m)\vec{x}_j + \underbrace{\sum_{\substack{i=1 \\ i\neq j}}^{m} \vec{x}_i(\vec{x}_i{}^T\vec{x}_j)}_{\text{“Störterm”}} .$$

$\vec{x}_j$ kann trotzdem stabil sein, wenn $n-m \geq 0$ gilt und der “Störterm” hinreichend klein ist.

Dieser Fall tritt ein, wenn die Muster “annähernd” orthogonal sind.

Je größer die Zahl der zu speichernden Muster ist, desto kleiner muß der Störterm sein.

Die theoretische Maximalkapazität eines Hopfield-Netzes wird praktisch nie erreicht.

Beispiele

Beispiel: Speichere Muster $\vec{x}_1 = (+1, +1, -1, -1)^\top$ und $\vec{x}_2 = (-1, +1, -1, +1)^\top$.

$$\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 = \vec{x}_1 \vec{x}_1^T + \vec{x}_2 \vec{x}_2^T - 2\mathbf{E}$$

wobei

$$\mathbf{W}_1 = \begin{pmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{pmatrix}, \qquad \mathbf{W}_2 = \begin{pmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{pmatrix}.$$

Die vollständige Gewichtsmatrix ist:

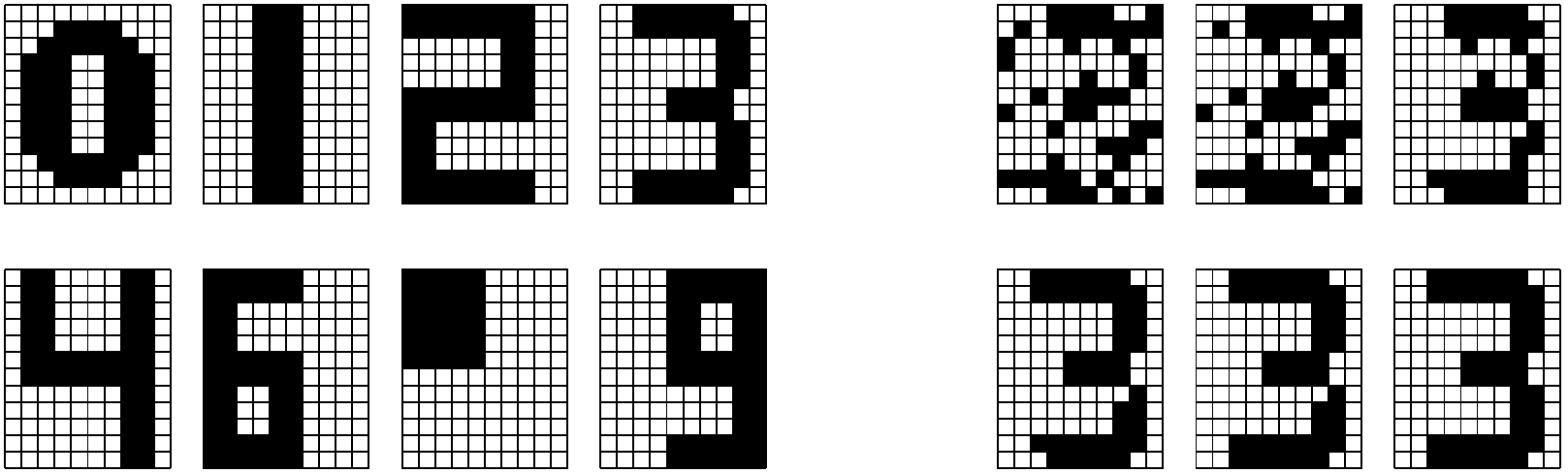
$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{pmatrix}.$$

Daher ist

$$\mathbf{W}\vec{x}_1 = (+2, +2, -2, -2)^\top \qquad \text{und} \qquad \mathbf{W}\vec{x}_1 = (-2, +2, -2, +2)^\top.$$

Beispiele

Beispiel: Speichere Bitmaps von Zahlen



Links: Bitmaps, die im Hopfield-Netz gespeichert sind.

Rechts: Rekonstruktion eines Musters aus einer zufälligen Eingabe.

Assoziativspeicher

Trainieren eines Hopfield-Netzes mit der Delta-Regel

Notwendige Bedingung, dass ein Muster x einem stabilen Zustand entspricht:

$$\begin{array}{llllll}
s(0 & + w_{u_1u_2}\,\mathrm{act}_{u_2}^{(x)} & + \ldots + w_{u_1u_n}\,\mathrm{act}_{u_n}^{(x)} & - \theta_{u_1}) = \mathrm{act}_{u_1}^{(x)}, \\
s(w_{u_2u_1}\,\mathrm{act}_{u_1}^{(x)} & + 0 & + \ldots + w_{u_2u_n}\,\mathrm{act}_{u_n}^{(x)} & - \theta_{u_2}) = \mathrm{act}_{u_2}^{(x)}, \\
\vdots & \vdots & \vdots & \vdots \quad \vdots \\
s(w_{u_nu_1}\,\mathrm{act}_{u_1}^{(x)} & + w_{u_nu_2}\,\mathrm{act}_{u_2}^{(x)} & + \ldots + 0 & - \theta_{u_n}) = \mathrm{act}_{u_n}^{(x)}.
\end{array}$$

mit der standardmäßigen Schwellenwertfunktion

$$s(x) = \begin{cases} 1, & \text{falls} \quad x \geq 0, \\ -1, & \text{sonst.} \end{cases}$$

Assoziativspeicher

Trainieren eines Hopfield-Netzes mit der Delta-Regel

Überführe Gewichtsmatrix in einen Gewichtsvektor:

$$\begin{array}{llll} \vec{w} = (& w_{u_1u_2}, & w_{u_1u_3}, & \ldots, \quad w_{u_1u_n}, \\ & & w_{u_2u_3}, & \ldots, \quad w_{u_2u_n}, \\ & & & \ddots \quad \vdots \\ & & & w_{u_{n-1}u_n}, \\ & -\theta_{u_1}, & -\theta_{u_2}, & \ldots, \quad -\theta_{u_n} \qquad). \end{array}$$

Konstruiere Eingabevektoren für ein Schwellenwertelement

$$\vec{z}_2 = (\mathrm{act}_{u_1}^{(p)}, \underbrace{0, \ldots, 0,}_{n-2 \text{ Nullen}} \mathrm{act}_{u_3}^{(p)}, \ldots, \mathrm{act}_{u_n}^{(p)}, \ldots 0, 1, \underbrace{0, \ldots, 0}_{n-2 \text{ Nullen}}).$$

Wende die Deltaregel auf diesen Gewichtsvektor und die Eingabevektoren an, bis sich Konvergenz einstellt.

Lösen von Optimierungsproblemen

Nutze Energieminimierung, um Optimierungsprobleme zu lösen

Allgemeine Vorgehensweise:

- Transformiere die zu optimierende Funktion in eine zu minimierende.
- Transformiere Funktion in die Form einer Energiefunktion eines Hopfield-Netzes.
- Lies die Gewichte und Schwellenwerte der Energiefunktion ab.
- Konstruiere das zugehörige Hopfield-Netz.
- Initialisiere das Hopfield-Netz zufällig und aktualisiere es solange, bis sich Konvergenz einstellt.
- Lies die Lösung aus dem erreichten stabilen Zustand ab.
- Wiederhole mehrmals und nutze die beste gefundene Lösung.

Achtung: Optimiere Zustand (Aktivierungen) – nicht die Netzwerkparameter!

Aktivierungstransformation

(Herleitung siehe Abschnitt 28.3 im Buch)

Ein Hopfield-Netz kann entweder mit Aktivierungen -1 und 1 oder mit Aktivierungen 0 and 1 definiert werden. Die Netze können ineinander umgewandelt werden.

Von $\mathrm{act}_u \in \{-1, 1\}$ in $\mathrm{act}_u \in \{0, 1\}$:

$$\begin{aligned} w^0_{uv} &= 2w^-_{uv} && \text{und} \\ \theta^0_u &= \theta^-_u + \sum_{v \in U - \{u\}} w^-_{uv} \end{aligned}$$

Von $\mathrm{act}_u \in \{0, 1\}$ in $\mathrm{act}_u \in \{-1, 1\}$:

$$\begin{aligned} w^-_{uv} &= \frac{1}{2} w^0_{uv} && \text{und} \\ \theta^-_u &= \theta^0_u - \frac{1}{2} \sum_{v \in U - \{u\}} w^0_{uv}. \end{aligned}$$

Notation:	^{0}	:	Größe aus Netz mit $\mathrm{act}_u \in \{0, 1\}$,
	$^-$	:	Größe aus Netz mit $\mathrm{act}_u \in \{-1, 1\}$.

Lösen von Optimierungsproblemen

Kombinationslemma: Gegeben seien zwei Hopfield-Netze auf derselben Menge U Neuronen mit Gewichten $w_{uv}^{(i)}$, Schwellenwerten $\theta_u^{(i)}$ und Energiefunktionen

$$E_i = -\frac{1}{2} \sum_{u \in U} \sum_{v \in U - \{u\}} w_{uv}^{(i)} \operatorname{act}_u \operatorname{act}_v + \sum_{u \in U} \theta_u^{(i)} \operatorname{act}_u,$$

$i = 1, 2$. Weiterhin sei $a, b \in \mathbb{R}$. Dann ist $E = aE_1 + bE_2$ die Energiefunktion des Hopfield-Netzes auf den Neuronen in U das die Gewichte $w_{uv} = aw_{uv}^{(1)} + bw_{uv}^{(2)}$ und die Schwellenwerte $\theta_u = a\theta_u^{(1)} + b\theta_u^{(2)}$ hat.

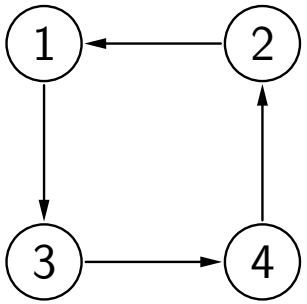
Beweis: Einfach Berechnungen durchführen.

Idee: Zusätzliche Bedingungen können separat formuliert und später mit einbezogen werden.

Lösen von Optimierungsproblemen

Beispiel: Problem des Handlungsreisenden
(TSP – Traveling Salesman Problem)

Idee: Stelle Tour durch Matrix dar.



$$\begin{array}{c} \text{Stadt} \\ \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{array}{l} 1. \\ 2. \\ 3. \\ 4. \end{array} \end{array} \quad \text{Schritt}$$

Ein Element m_{ij} der Matrix ist 1 wenn die i-te Stadt im j-ten Schritt besucht wird und 0 sonst.

Jeder Matrixeintrag wird durch ein Neuron repräsentiert.

Lösen von Optimierungsproblemen

Minimierung der Tourlänge

Distanz von Stadt j_1 nach j_2

$$E_1 = \sum_{j_1=1}^{n} \sum_{j_2=1}^{n} \sum_{i=1}^{n} d_{j_1 j_2} \cdot m_{i j_1} \cdot m_{(i \bmod n)+1, j_2}.$$

nur > 0 wenn Städte aufeinander folgen

Index-"Überlauf"

Doppelsumme über die benötigten Schritte (Index i):

$$E_1 = \sum_{(i_1,j_1)\in\{1,\ldots,n\}^2} \sum_{(i_2,j_2)\in\{1,\ldots,n\}^2} d_{j_1 j_2} \cdot \delta_{(i_1 \bmod n)+1, i_2} \cdot m_{i_1 j_1} \cdot m_{i_2 j_2},$$

wobei

$$\delta_{ab} = \begin{cases} 1, & \text{falls } a = b, \\ 0, & \text{sonst.} \end{cases}$$

Test, ob direkter Nachfolger

Symmetrische Version der Energiefunktion:

$$E_1 = -\frac{1}{2} \sum_{\substack{(i_1,j_1)\in\{1,\ldots,n\}^2 \\ (i_2,j_2)\in\{1,\ldots,n\}^2}} -d_{j_1 j_2} \cdot (\delta_{(i_1 \bmod n)+1, i_2} + \delta_{i_1, (i_2 \bmod n)+1}) \cdot m_{i_1 j_1} \cdot m_{i_2 j_2}$$

jeder Schritt wird doppelt gezählt (gleicht Faktor ½ aus)

Lösen von Optimierungsproblemen

Zusätzliche Bedingungen, die erfüllt werden müssen:

- Jede Stadt wird in genau einem Schritt der Tour besucht:

$$\forall j \in \{1, \ldots, n\} : \qquad \sum_{i=1}^{n} m_{ij} = 1,$$

  d.h. jede Spalte der Matrix enthält genau eine 1.

- In jedem Schritt der Tour wird genau eine Stadt besucht:

$$\forall i \in \{1, \ldots, n\} : \qquad \sum_{j=1}^{n} m_{ij} = 1,$$

  d.h. jede Zeile der Matrix enthält genau eine 1.

Diese Bedingungen werden erfüllt durch zusätzlich zu optimierende Funktionen.

Lösen von Optimierungsproblemen

Formalisierung der ersten Bedingung als Minimierungsproblem: $E_2^* = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} m_{ij} - 1 \right)^2 = 0$

$$
\begin{aligned}
E_2^* &= \sum_{j=1}^{n} \left(\left(\sum_{i=1}^{n} m_{ij} \right)^2 - 2 \sum_{i=1}^{n} m_{ij} + 1 \right) \\
&= \sum_{j=1}^{n} \left(\left(\sum_{i_1=1}^{n} m_{i_1 j} \right) \left(\sum_{i_2=1}^{n} m_{i_2 j} \right) - 2 \sum_{i=1}^{n} m_{ij} + 1 \right) \\
&= \sum_{j=1}^{n} \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} m_{i_1 j} m_{i_2 j} - 2 \sum_{j=1}^{n} \sum_{i=1}^{n} m_{ij} + n.
\end{aligned}
$$

Doppelsumme über benötigte Städte (Index i): (wie auf Folie 33)

$$
E_2 = \sum_{(i_1,j_1) \in \{1,\ldots,n\}^2} \sum_{(i_2,j_2) \in \{1,\ldots,n\}^2} \delta_{j_1 j_2} \cdot m_{i_1 j_1} \cdot m_{i_2 j_2} - 2 \sum_{(i,j) \in \{1,\ldots,n\}^2} m_{ij}.
$$

Lösen von Optimierungsproblemen

Sich ergebende Energiefunktion:

$$E_2 = -\frac{1}{2} \sum_{\substack{(i_1,j_1)\in\{1,\dots,n\}^2 \\ (i_2,j_2)\in\{1,\dots,n\}^2}} -2\delta_{j_1 j_2} \cdot m_{i_1 j_1} \cdot m_{i_2 j_2} + \sum_{(i,j)\in\{1,\dots,n\}^2} -2m_{ij}$$

Die zweite zusätzliche Bedingung wird analog gehandhabt:

$$E_3 = -\frac{1}{2} \sum_{\substack{(i_1,j_1)\in\{1,\dots,n\}^2 \\ (i_2,j_2)\in\{1,\dots,n\}^2}} -2\delta_{i_1 i_2} \cdot m_{i_1 j_1} \cdot m_{i_2 j_2} + \sum_{(i,j)\in\{1,\dots,n\}^2} -2m_{ij}.$$

Kombinieren der Energiefunktionen:

$$E = aE_1 + bE_2 + cE_3 \qquad \text{wobei} \qquad \frac{b}{a} = \frac{c}{a} > 2 \max_{(j_1,j_2)\in\{1,\dots,n\}^2} d_{j_1 j_2}.$$

Die größtmögliche Verbesserung, die durch eine (lokale) Änderung der Reiseroute erzielt werden kann, ist kleiner als die minimale Verschlechterung, die sich aus einer Verletzung einer Nebenbedingung ergibt.

Lösen von Optimierungsproblemen

Aus der resultierenden Energiefunktionen können wir die Gewichte

$$w_{(i_1,j_1)(i_2,j_2)} = \underbrace{-ad_{j_1j_2} \cdot (\delta_{(i_1 \bmod n)+1,i_2} + \delta_{i_1,(i_2 \bmod n)+1})}_{\text{von } E_1} \underbrace{-2b\delta_{j_1j_2}}_{\text{von } E_2} \underbrace{-2c\delta_{i_1i_2}}_{\text{von } E_3}$$

und die Schwellenwerte:

$$\theta_{(i,j)} = \underbrace{0a}_{\text{von } E_1} \quad \underbrace{-2b}_{\text{von } E_2} \quad \underbrace{-2c}_{\text{von } E_3} = -2(b+c)$$

ablesen.

Problem: die zufällige Initialisierung und die Aktualisierung bis zur Konvergenz führen nicht immer zu einer Matrix, die tatsächlich eine Tour repräsentiert, geschweige denn eine optimale Tour.

Schwachstellen

Problem: Wechsel von einer gültigen Tour zu einer anderen gültigen Tour schwierig

Grund: Die benötigte Transformation der Matrix, der aktuellen Lösung bräuchte mindestens vier Änderungen

Wenn Änderungen asynchron durchgeführt werden widerspricht mindestens eine einem Constraint, führt also zur Erhöhung der Energie

Nur alle vier Änderungen zusammen reduzieren die Energie

Daher wird eine gefundene gültige Lösung nicht mehr geändert

Dieses Problem tritt auch bei anderen Optimierungsmethoden auf. So liegt es nahe, Lösungsideen, die für andere Optimierungsmethoden entwickelt wurden, auf Hopfield-Netze zu übertragen, z.B. das sogenannte simulierte Ausglühen.

Lokale Optima

Erweiterung:

- Verwende keine diskreten Hopfield Netze mit Aktivierungen $[-1, 1]$ oder $[0, 1]$, sondern
- kontinuierliche Hopfield Netze mit Aktivierungen in $[-1, 1]$ (oder $[0, 1]$)

Kontinuierliche Hopfield Netze liefern etwas bessere Ergebnisse, lösen aber das Grundproblem nicht.

Generelles Problem: Steckenbleiben im lokalen Optimum

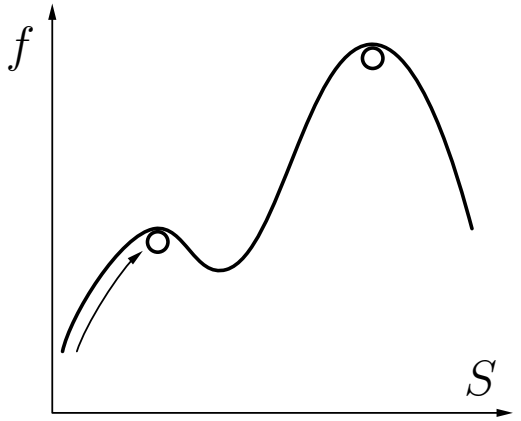
Ein beliebter Ansatz ist das sogenanntes simulierte Ausglühen

Simuliertes Ausglühen

[Metropolis 1953, Kirkpatrick 1983]

Kann als Erweiterung des Zufalls- und Gradientenaufstiegs gesehen werden, die ein Hängenbleiben vermeidet.

Idee: Übergänge von niedrigeren zu höheren (lokalen) Maxima sollen wahrscheinlicher sein als umgekehrt.



Prinzip des simulierten Ausglühens:

- Zufällige Varianten der aktuellen Lösung werden erzeugt.
- Bessere Lösungen werden immer übernommen.
- Schlechtere Lösungen werden mit einer bestimmten Wahrscheinlichkeit übernommen, die abhängt von
  - der Qualitätsdifferenz der aktuellen und der neuen Lösung und
  - einem Temperaturparameter, der im Laufe der Zeit verringert wird.

Simuliertes Ausglühen

Motivation: (Minimierung statt Maximierung)

Physikalische Minimierung der Energie (genauer: der Atomgitterenergie), wenn ein erhitztes Stück Metall langsam abgekühlt wird.

Dieser Prozess wird **Ausglühen** (engl.: annealing) genannt. Er dient dazu, ein Metall weicher zu machen, indem innere Spannungen und Instabilitäten aufgehoben werden, um es dann leichter bearbeiten zu können.

Alternative Motivation: (ebenfalls Minimierung)

Eine Kugel rollt auf einer unregelmäßig gewellten Oberfläche.
Die zu minimierende Funktion ist die potentielle Energie der Kugel.

Am Anfang hat die Kugel eine gewisse kinetische Energie, die es ihr erlaubt, Anstiege zu überwinden. Durch Reibung sinkt die Energie der Kugel, sodass sie schließlich in einem Tal zur Ruhe kommt.

Achtung: Es gibt keine Garantie, dass das globale Optimum gefunden wird.

Simuliertes Ausglühen

1. Wähle einen (zufälligen) Startpunkt $s_0 \in S$.
2. Wähle einen Punkt $s' \in S$ „in der Nähe“ des aktuellen Punktes s_i.
   (z.B. durch zufällige, aber nicht zu große Veränderung von s_i)
3. Setze

$$s_{i+1} = \begin{cases} s', & \text{falls } f(s') \geq f(s_i), \\ s' & \text{mit Wahrscheinlichkeit } p = e^{-\frac{\Delta f}{kT}}, \\ s_i & \text{mit Wahrscheinlichkeit } 1-p, \end{cases} \quad \text{sonst.}$$

$\Delta f = f(s_i) - f(s')$	Qualitätsverringerung der Lösung
$k = \Delta f_{\max}$	(Schätzung der) Spannweite der Funktionswerte
T	Temperaturparameter; wird im Laufe der Zeit gesenkt

4. Wiederhole Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist.

Für kleine T geht das Verfahren nahezu in einen Zufallsaufstieg über.

Simuliertes Ausglühen

Anwendung des simulierten Ausglühens auf Hopfield-Netze ist simpel:

- zufällige Initialisierung der Aktivierungen
- Neuronen des Hopfield-Netzes werden durchlaufen (z.B. in zufälliger Reihenfolge) und es wird bestimmt, ob eine Änderung ihrer Aktivierung zu einer Verringerung der Energie führt oder nicht.
- Aktivierungsänderung, die die Energie vermindert, wird immer ausgeführt. Eine die Energie erhöhende Änderung wird nur mit einer Wahrscheinlichkeit ausgeführt.

Man beachte, dass in diesem Fall einfach

$$\Delta f = \Delta E = |\operatorname{net}_u - \theta_u|$$

ist.