Neuronale Netze

Prof. Dr. Rudolf Kruse

Computational Intelligence
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
rudolf.kruse@ovgu.de

Hopfield-Netze

Hopfield-Netze

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.

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

Hopfield-Netze

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.

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

Alternative Aktivierungsfunktion

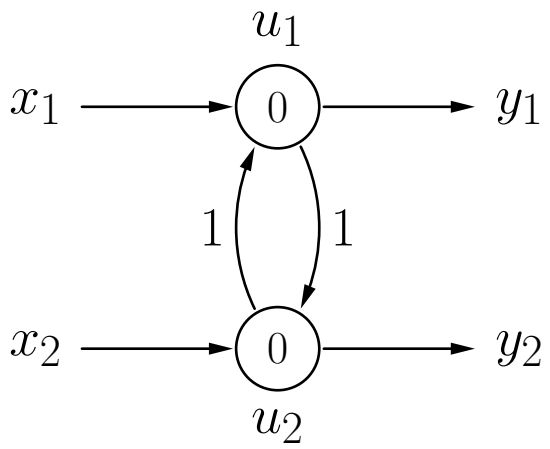
$$\forall u \in U: \quad f_{\text{act}}^{(u)}(\text{net}_u, \theta_u, \text{act}_u) = \begin{cases} 1, & \text{falls} \quad \text{net}_u > \theta, \\ -1, & \text{falls} \quad \text{net}_u < \theta, \\ \text{act}_u, & \text{falls} \quad \text{net}_u = \theta. \end{cases}$$

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

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

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.

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.

Sequentielle Aktualisierung der Neuronenaktivierungen

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

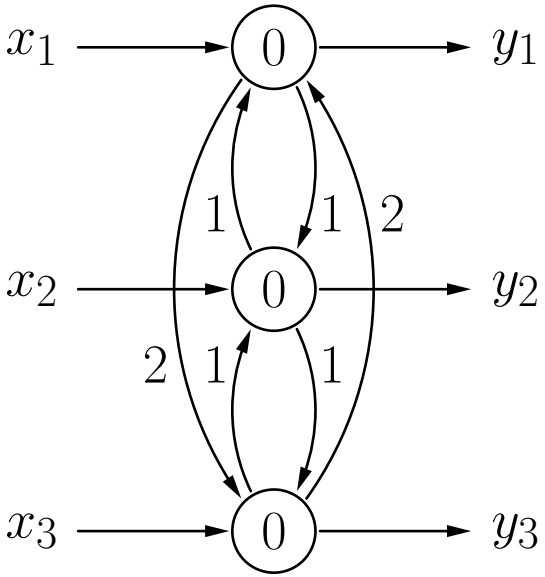
	u_1	u_2
Eingabephase	$\mathbf{-1}$	$\mathbf{1}$
Arbeitsphase	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1
	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1

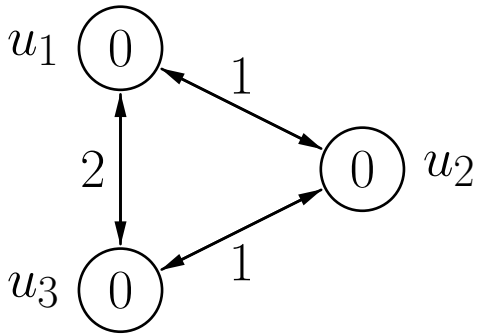
Aktivierungsreihenfolge u_1, u_2, u_1, ... bzw. u_2, u_1, u_1, ...

Unabhängig von der Reihenfolge wird ein stabiler Zustand erreicht.

Welcher Zustand stabil ist, hängt von der Reihenfolge ab.

Vereinfachte Darstellung eines Hopfield-Netzes



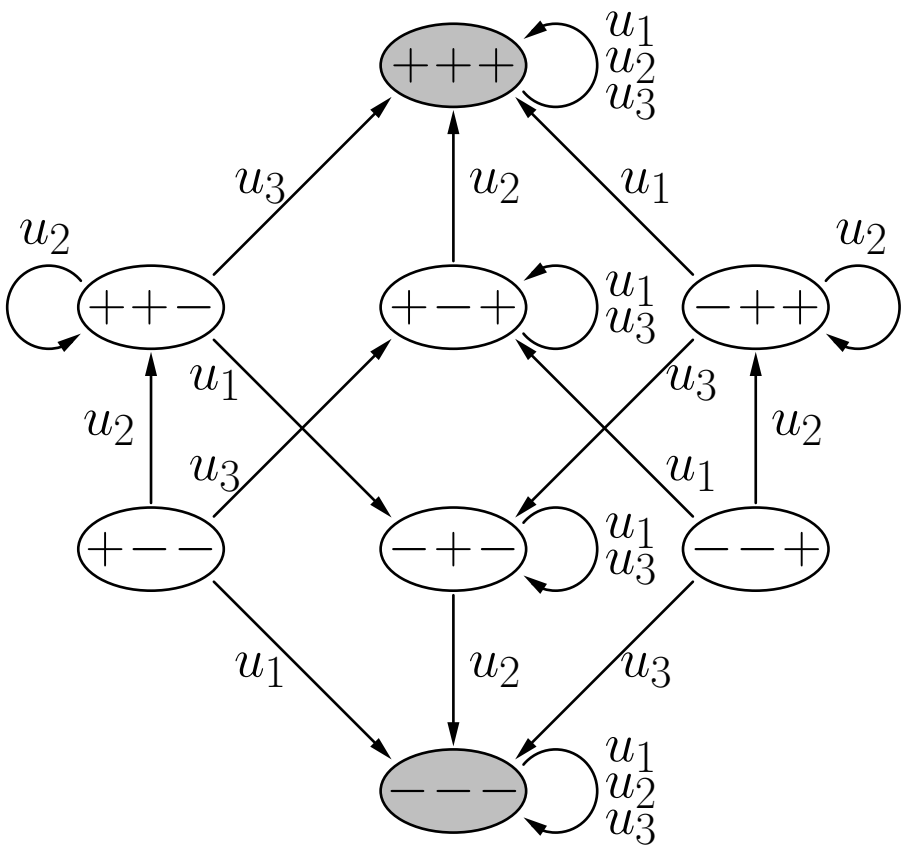


$$\mathbf{W} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

Symmetrische Verbindungen zwischen Neuronen werden kombiniert.

Eingaben und Ausgaben werden nicht explizit dargestellt.

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)

Hopfield-Netze: 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

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

Hopfield-Netze: 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: $\Delta E \leq 0$.

Hopfield-Netze: 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

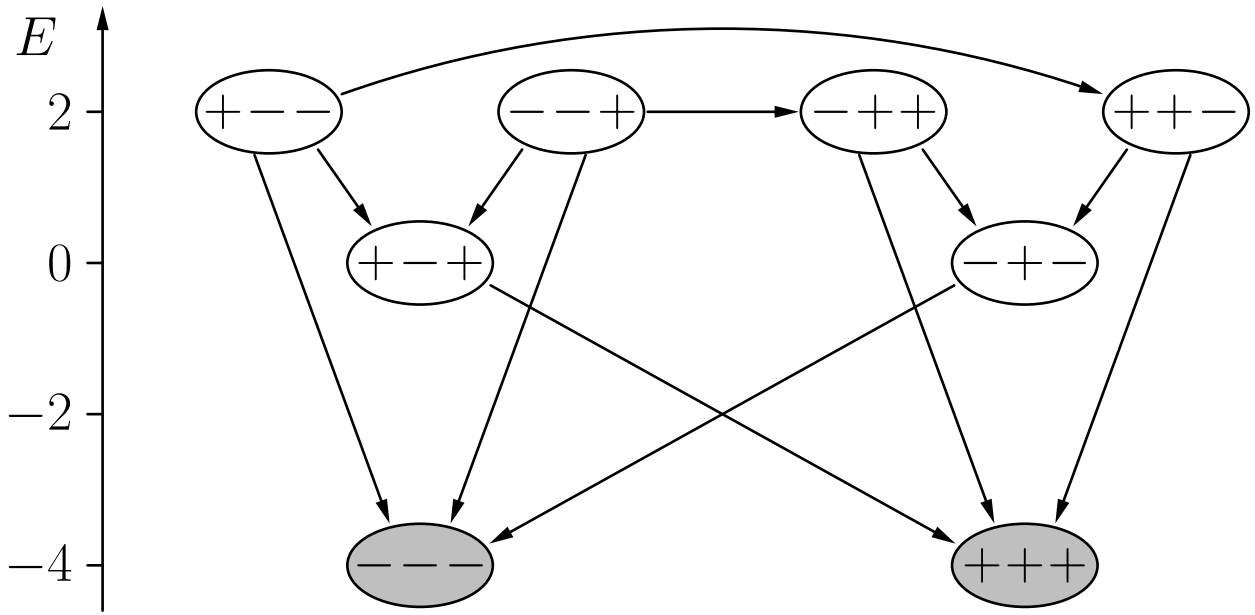
- a) es ändert sich keine Aktivierung – ein stabiler Zustand wurde erreicht
- b) es ändert sich mindestens eine Aktivierung: dann wurde damit mindestens einer der 2^n möglichen Aktivierungszustände ausgeschlossen.

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.

Hopfield-Netze: Beispiele

Ordne die Zustände im Graphen entsprechend ihrer Energie

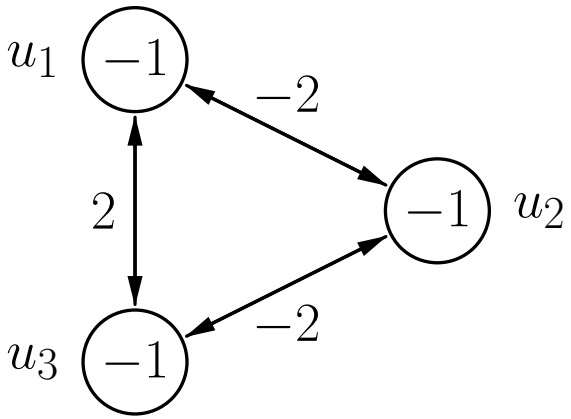


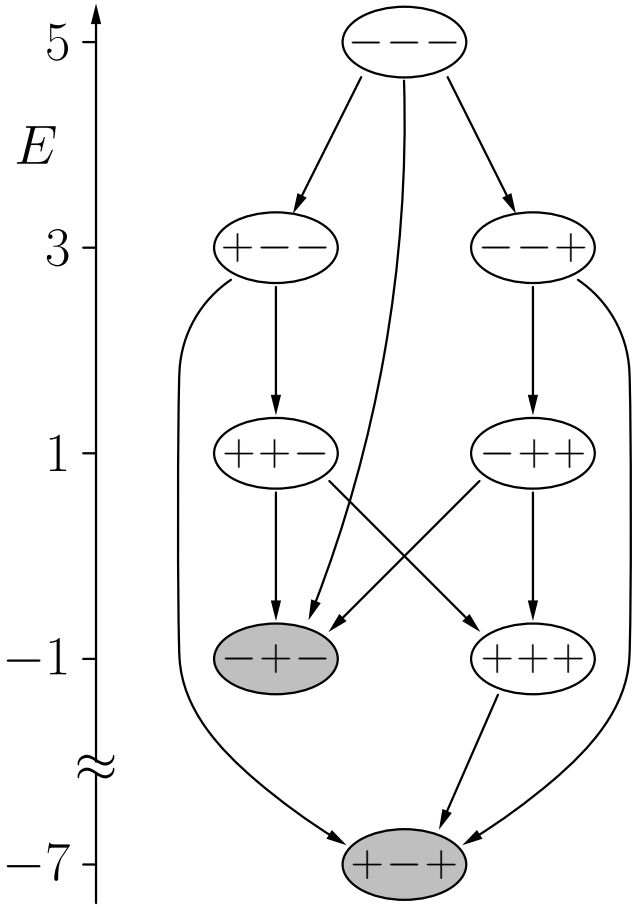
Energiefunktion für das Beispiel-Hopfield-Netz:

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

Hopfield-Netze: Beispiele

Der Zustandsgraph muss nicht symmetrisch sein





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

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

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

mit

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

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

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

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

Mit dieser Matrix erhalten wir

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

Hebb'sche Lernregel [Hebb 1949]

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

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

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 $\mathbf{W}(-\vec{x}) = (n-1)(-\vec{x})$.

Speichern mehrerer Muster

Wähle

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

Wenn die Muster orthogonal sind, gilt

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

und daher

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

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).

Nicht-orthogonale Muster:

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

p_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.

Assoziativspeicher: Beispiel

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:

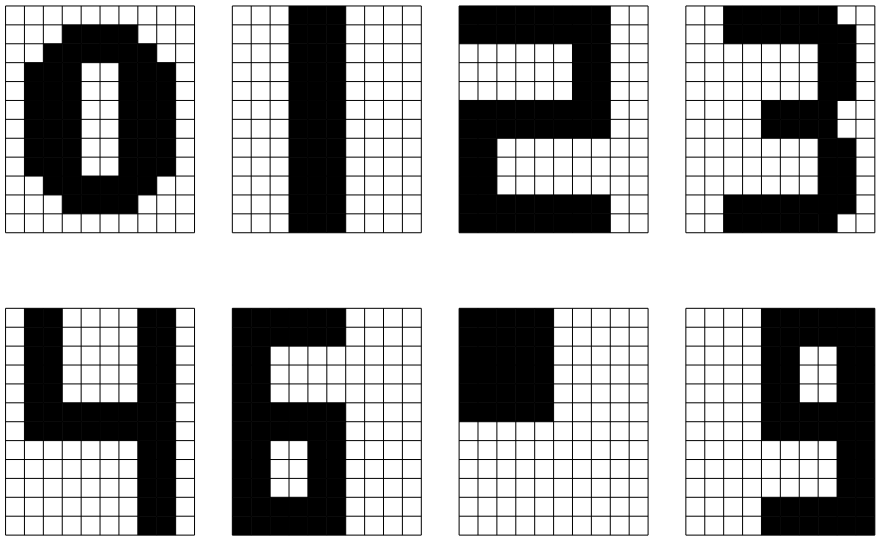
$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{pmatrix}.$$

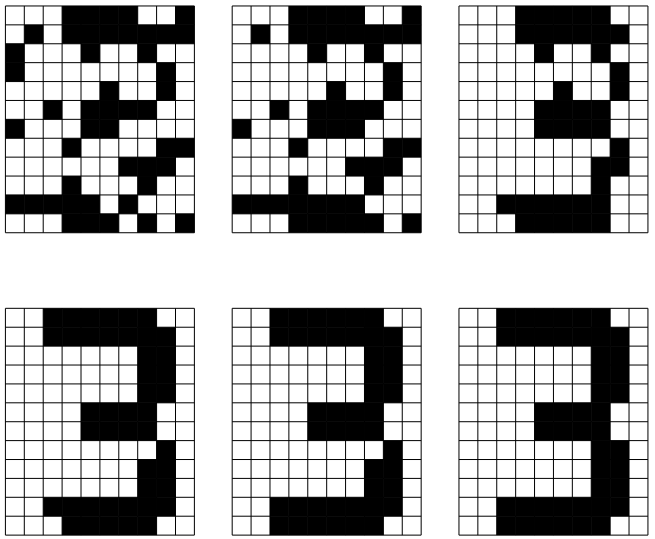
Daher ist

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

Assoziativspeicher: Beispiele

Beispiel: Speichere Bitmaps von Zahlen





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

Rechts: Rekonstruktion eines Musters aus einer zufälligen Eingabe.

Trainieren eines Hopfield-Netzes mit der Delta-Regel

Notwendige Bedingung, dass ein Muster $\vec{x}$ einem stabilen Zustand entspricht:

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

mit der standardmäßigen Schwellenwertfunktion

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

Trainieren eines Hopfield-Netzes mit der Delta-Regel

Überführe Gewichtsmatrix in einen Gewichtsvektor:

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

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.

Hopfield-Netze: 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.

Hopfield-Netze: Aktivierungstransformation

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

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

Hopfield-Netze: 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

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

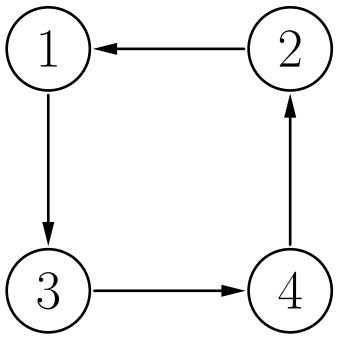
$i = 1, 2$. Weiterhin sei $a, b \in \mathbb{R}$. Dann ist $E = aE_1 + bE_2$ die Energiefunktion des Hopfield-Netzes auf den Neuronen in U das die Gewichte $w_{uv} = aw_{uv}^{(1)} + bw_{uv}^{(2)}$ und die Schwellenwerte $\theta_u = a\theta_u^{(1)} + b\theta_u^{(2)}$ hat.

Beweis: Einfach Berechnungen durchführen.

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

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

Idee: Stelle Tour durch Matrix dar.



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

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.

Minimierung der Tourlänge

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

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

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

Symmetrische Version der Energiefunktion:

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

Hopfield-Netze: 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.

Hopfield-Netze: Lösen von Optimierungsproblemen

Formalisierung der ersten Bedingung als Minimierungsproblem:

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

Doppelsumme über benötigte Städte (Index i):

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

Hopfield-Netze: Lösen von Optimierungsproblemen

Sich ergebende Energiefunktion:

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

Die zweite zusätzliche Bedingung wird analog gehandhabt:

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

Kombinieren der Energiefunktionen:

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

Hopfield-Netze: Lösen von Optimierungsproblemen

Aus der resultierenden Energiefunktionen können wir die Gewichte

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

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.

Hopfield-Netze: 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*.

Hopfield Netze: 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.

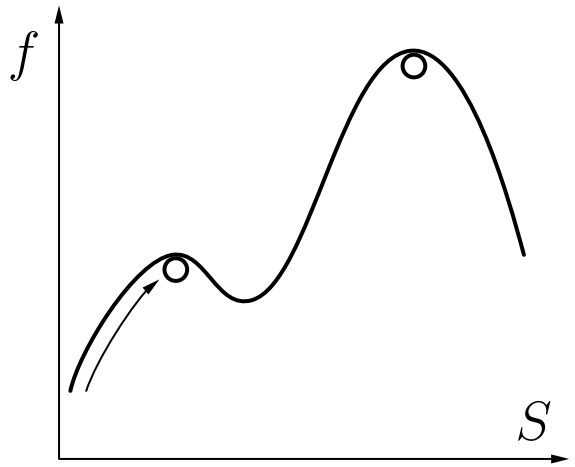
Generelles Problem: Steckenbleiben im lokalen Optimum

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

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

$\Delta f = f(s_i) - f(s')$	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.

Hopfield-Netze: 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.

Boltzmann Maschinen

Boltzmann Maschinen sind eng verwandt mit Hopfield Netzen

Hauptunterschied: Update der Neuronen

Gemeinsamkeit: Energiefunktion, die jedem Status des Netzes einen numerischen Wert zuordnet.

Mit Hilfe der Energiefunktion wird eine Wahrscheinlichkeitsverteilung der Netzwerkzustände definiert, die auf der Boltzmann-Verteilung (auch Gibbs-Verteilung) basiert:

$$P(\vec{s}) = \frac{1}{c}\, e^{-\frac{E(\vec{s})}{kT}}.$$

$\vec{s}$	beschreibt den (diskreten) Zustand des Systems,
c	ist eine Konstante zur Normalisierung,
E	ist die Funktion, die die Energie des Zustands $\vec{s}$ bestimmt,
T	ist die thermodynamische Temperatur des Systems,
k	ist die Boltzmann Konstante ($k \approx 1.38 \cdot 10^{-23}\, J/K$).

Boltzmann Maschinen

In der Praxis wird meist das Produkt kT zu einem Parameter zusammengefasst

Der Zustand $\vec{s}$ besteht aus dem Vektor $\vec{\text{act}}$ der Aktivierung der Neuronen

Die Energiefunktion der Boltzmann Maschine ist

$$E(\vec{\text{act}}) = -\frac{1}{2}\,\vec{\text{act}}^{\top}\mathbf{W}\vec{\text{act}} + \vec{\theta}^{\top}\vec{\text{act}},$$

wobei $\mathbf{W}$: Gewichtsmatrix; $\vec{\theta}$: Vektor der Schwellenwerte.

Betrachte die Energieänderung durch die Änderung eines Neurons u:

$$\Delta E_u = E_{\text{act}_u=1} - E_{\text{act}_u=0} = \sum_{v \in U - \{u\}} w_{uv}\,\text{act}_v - \theta_u$$

Es gilt:

$$\Delta E_u = -kT\ln(P(\text{act}_u = 1) - (-kT\ln(P(\text{act}_u = 0)).$$

Umstellen ergibt

$$\begin{aligned}\frac{\Delta E_u}{kT} &= \ln(P(\text{act}_u = 1)) - \ln(P(\text{act}_u = 0)) \\ &= \ln(P(\text{act}_u = 1)) - \ln(1 - P(\text{act}_u = 1))\end{aligned}$$

(da offensichtlich $P(\text{act}_u = 0) + P(\text{act}_u = 1) = 1$).

Das Lösen der Gleichung für $P(\text{act}_u = 1)$ ergibt:

$$P(\text{act}_u = 1) = \frac{1}{1 + e^{-\frac{\Delta E_u}{kT}}}.$$

Das bedeutet: Wahrscheinlichkeit, dass ein Neuron aktiv ist, wird beschrieben durch die logistische Funktion der (skalierten) Energiedifferenz vom aktiven zum inaktiven Zustand

Die Energiedifferenz hängt ab von der Netzeingabe:

$$\Delta E_u = \sum_{v \in U - \{u\}} w_{uv}\, \text{act}_v - \theta_u = \text{net}_u - \theta_u,$$

Diese Formel legt eine stochastische Updateprozedur nahe

Boltzmann Maschinen: Update Prozess

Für ein zufällig gewähltes Neuron u werden Netzeingabe, Energiedifferenz ΔE_u und zuletzt die Wahrscheinlichkeit $P(\text{act}_u = 1)$ berechnet

Aktivierung wird mit $P(\text{act}_u = 1)$ auf 1 gesetzt, andernfalls auf 0.

Schritte werden mehrfach für zufällige Neuronen wiederholt.

Es wird Simuliertes Ausglühen angewandt, wobei die Temperatur langsam reduziert wird.

Dieser Update Prozess stellt eine **Markov-Ketten Monte Carlo Simulation** dar.

Nach hinreichend vielen Schritten hängt Wahrscheinlichkeit eines bestimmten Aktivierungszustandes nur noch von seiner Energie ab, nicht mehr von der Initialisierung.

Dieser finale Zustand wird als *thermales Equilibrium* bezeichnet

Boltzmann Maschinen sind Repräsentationen und Samplingmechanismen der Boltzmann-Verteilung

Boltzmann Maschinen: Training

Idee:
Entwickle ein Lernverfahren, mit dem die Wahrscheinlichkeitsverteilung der Boltzmann Maschine mit Hilfe der Energiefunktion an ein gegebenes Sample von Daten Punkten angepasst werden kann. Auf diese Weise wird ein **probabilistisches Modell** der Daten generiert.

Dies kann nur dann erfolgreich erreicht werden, wenn die Datenpunkte tatsächlich das Sample einer Boltzmann-Verteilung sind. Andernfalls ist es prinzipiell nicht möglich ein Modell zu erhalten, das zu den gegebenen Daten passt.

Um diese Einschränkung bezüglich der Boltzmann-Verteilung abzuschwächen, wird eine Variante der Struktur von Hopfield-Netzen verwendet.

Neuronen werden unterteilt in zwei Gruppen:

- **sichtbare Neuronen**, die Datenpunkte als Eingabe erhalten, und
- **versteckte Neuronen**, die durch die Datenpunkte nicht festgelegt sind.

Hopfield-Netze haben nur sichtbare Neuronen.

Boltzmann Maschinen: Training

Ziel des Trainings:
Passe Verbindungsgewichte und Schwellenwerte so an, dass die wahre Verteilung der Datenpunkte gut von den Wahrscheinlichkeitsverteilungen der sichtbaren Neuronen der Boltzmann-Maschine angenähert wird.

Natürlicher Ansatz zum Training:

- Wähle ein Maß für den Differenz zwischen zwei Wahrscheinlichkeitsverteilungen.
- Führe Gradientenabstieg durch, um diese Differenz zu minimieren.

Bekanntes Maß: **Kullback–Leibler-Divergenz**.
Für zwei Wahrscheinlichkeitsverteilungen p_1 und p_2 auf selben Beispielraum Ω:

$$KL(p_1, p_2) = \sum_{\omega \in \Omega} p_1(\omega) \ln \frac{p_1(\omega)}{p_2(\omega)}.$$

Angewendet auf Boltzmann Maschinen: p_1 stellt die Beispieldaten dar, p_2 stellt die sichtbaren Neuronen der Boltzmann Maschine dar.

Boltzmann Maschinen: Training

Das Training wird in zwei Phasen durchgeführt:

"Positive Phase": Sichtbare Neuronen werden auf einen zufällig gewählten Datenpunkt fixiert; nur versteckte Neuronen werden bis zum Erreichen des thermalen Equilibriums aktualisiert.

"Negative Phase": Alle Neuronen werden aktualisiert, bis das thermale Equilibrium erreicht ist.

In den beiden Phasen werden Statistiken über einzelne Neuronen und Paare von Neuronen (sichtbar und versteckt) wie häufig diese (gemeinsam) aktiv sind.

Das Update wird durch die folgenden beiden Gleichungen durchgeführt:

$$\Delta w_{uv} = \frac{1}{\eta}(p^+_{uv} - p^-_{uv}) \qquad \text{und} \qquad \Delta\theta_u = -\frac{1}{\eta}(p^+_u - p^-_u).$$

p_u	Die Wahrscheinlichkeit, dass Neuron u aktiv ist,
p_{uv}	Die Wahrscheinlichkeit, dass Neuronen u und v gemeinsam aktiv sind,
$+\ -$	als obere Indizes: bedeuten die zugehörige Phase

(Alle Wahrscheinlichkeiten werden über die beobachteten relativen Häufigkeiten geschätzt)

Boltzmann Maschinen: Training

Intuitive Erklärung der Update-Regel:

- Wenn ein Neuron öfter aktiv ist, wenn Beispieldaten gegeben sind, als wenn das Netzwerk frei schalten kann,
  dann ist die Wahrscheinlichkeit für Aktivität des Neurons zu gering und der Schwellenwert sollte reduziert werden.

- Wenn Neuronen häufiger zusammen aktiv sind, wenn Beispieldaten gegeben sind, als wenn das Netzwerk frei schalten kann,
  dann sollte ihr Verbindungsgewicht erhöht werden,
  so dass es wahrscheinlicher wird, dass sie zusammen aktiv sind.

Dieses Trainingsverfahren ist sehr ähnlich zur **Hebb'schen Lernregel**

Diese besagt dass die Verbindungen zwischen zwei Neuronen, die gleichzeitig aktiv sind, verstärkt werden

Umgangssprachlich: "cells that fire together, wire together".

Boltzmann Maschinen: Training

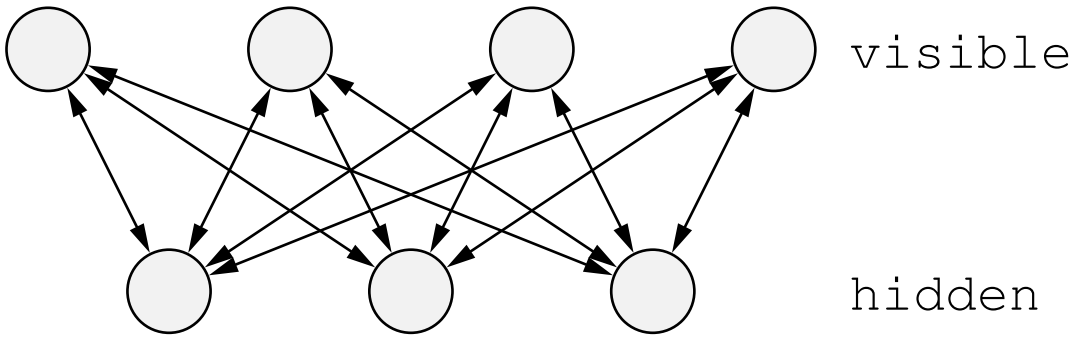
Diese Prozedur sehr aufwändig, wenn die Netze nicht sehr klein sind

Hauptgrund hierfür: bei großen Netzen müssen mehr Updates durchgeführt werden, um **verlässliche** Statistiken über die Neuronenpaare zu erhalten

Effizientes Training ist möglich für **Restricted Boltzmann Machines**.

Diese sind eingeschränkt, dahingehend, dass sie keinen vollständigen Graphen, sondern einen bipartiten Graphen verwenden:

- Knoten werden in zwei Gruppen aufgeteilt, in sichtbare und versteckte Neuronen;
- Verbindungen existieren nur zwischen Neuronen verschiedener Gruppen



Restricted Boltzmann Machines

Eine **restricted Boltzmann Machine (RBM)** oder **Harmonium** ist ein neuronales Netz mit einem Graphen $G = (U, C)$ und folgenden Einschränkungen:

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

(ii) $C = U_{\text{in}} \times U_{\text{hidden}}.$

In einer restricted Boltzmann Machine sind alle Eingabeneuronen auch Ausgabeneuronen und umgekehrt.

Es gibt versteckte Neuronen, die weder Ein- noch Ausgabeneuronen sind

Jedes Ein-/Ausgabeneuron hat alle versteckten Neuronen als Eingabe.
Jedes versteckte Neuron hat alle Ein-/Ausgabeneuronen als Eingabe.

Die Verbindungsgewichte zwischen Eingabe- und versteckten Neuronen sind symmetrisch:

$$\forall u \in U_{\text{in}}, v \in U_{\text{hidden}} : \qquad w_{uv} = w_{vu}.$$

Restricted Boltzmann Machines: Training

Wegen der fehlenden Verbindungen zwischen sichtbaren und versteckten Neuronen erfolgt das Training durch Wiederholen der folgenden drei Schritte:

1) Sichtbare Neuronen werden auf ein zufälliges Trainingsbeispiel $\vec{x}$ fixiert
   Versteckte Neuronen werden einmal parallel aktualisiert mit Resultat: $\vec{y}$

   $\vec{x}\vec{y}^\top$ wird der **positive Gradient** der Gewichtsmatrix genannt.

2) Versteckte Neuronen werden auf den berechneten Vektor $\vec{y}$ fixiert
   Sichtbare Neuronen werden einmal parallel aktualisiert mit Resultat: $\vec{x}^*$).

   Sichtbare Neuronen werden darauf fixiert, $\vec{x}^*$ zu "rekonstruieren";
   Versteckte Neuronen werden noch einmal aktualisiert mit Resultat: $\vec{y}^*$).

   $\vec{x}^*\vec{y}^{*\top}$ wird der **negative Gradient** der Gewichtsmatrix genannt.

3) Verbindungsgewichte werden mit der Differenz des positiven und negativen Gradienten aktualisiert:

$$\Delta w_{uv} = \eta(\vec{x}_u\vec{y}_v - \vec{x}^*_u\vec{y}^*_v) \qquad \eta \text{ ist dabei die Lernrate.}$$

Restriced Boltzmann Machines: Training und Deep Learning

Es gibt viele Verbesserungen dieser Basisprozedur [Hinton 2010]:

- Nutze Momentterm beim Training,
- Nutze echte Wahrscheinlichkeiten statt binärer Rekonstruktionen,
- Nutze online-artige Verfahren mit kleinen Batches

Restricted Boltzmann Machines wurden auch verwendet um tiefe Netze zu erzeugen – ähnlich wie stacked Autoencoders.

Idee: Trainiere eine restricted Boltzmann machine und erstelle mit dem Datensatz der Aktivierungen der versteckten Neuronen eine weitere RBM.

Dies kann mehrfach wiederholt werden, so dass die resultierenden RBMs einfach hintereinander geschaltet werden können.

Das entstandene Netz wird dann mit Fehlerrückprapation weiter trainiert [Hinton 2006].

Mehr dazu im Kapitel zum Thema *Deep Learning*