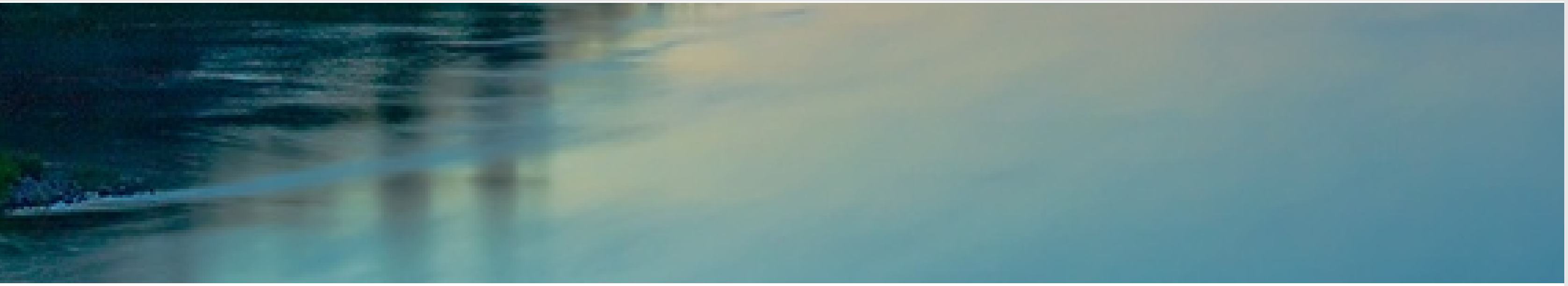


Neural Networks

Prof. Dr. Rudolf Kruse

Computational Intelligence Group
Faculty for Computer Science
kruse@iws.cs.uni-magdeburg.de

Hopfield Networks

A **Hopfield network** is a neural network with a graph $G = (U, C)$ that satisfies the following conditions:

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

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

- In a Hopfield network all neurons are input as well as output neurons.
- There are no hidden neurons.
- Each neuron receives input from all other neurons.
- A neuron is not connected to itself.

The connection weights are symmetric, i.e.

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

Hopfield Networks

The network input function of each neuron is the weighted sum of the outputs of all other neurons, i.e.

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

The activation function of each neuron is a threshold function, i.e.

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

The output function of each neuron is the identity, i.e.

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

Alternative activation function

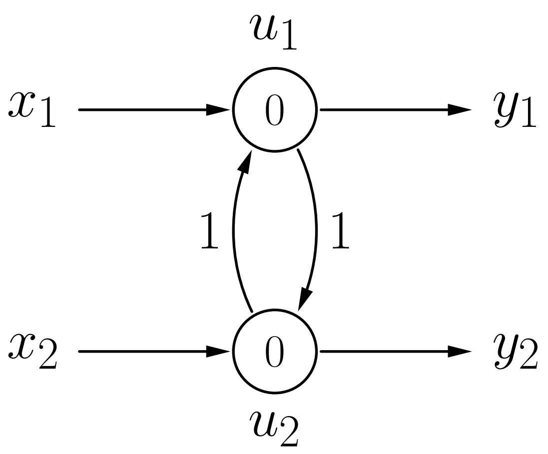
$$\forall u \in U: \quad f_{\text{act}}^{(u)}(\text{net}_u, \theta_u, \text{act}_u) = \begin{cases} 1, & \text{if} \quad \text{net}_u > \theta, \\ -1, & \text{if} \quad \text{net}_u < \theta, \\ \text{act}_u, & \text{if} \quad \text{net}_u = \theta. \end{cases}$$

This activation function has advantages w.r.t. the physical interpretation of a Hopfield network.

General weight matrix of a Hopfield network

$$\mathbf{W} = \begin{pmatrix} 0 & w_{u_1 u_2} & \dots & w_{u_1 u_n} \\ w_{u_1 u_2} & 0 & \dots & w_{u_2 u_n} \\ \vdots & \vdots & & \vdots \\ w_{u_1 u_n} & w_{u_1 u_n} & \dots & 0 \end{pmatrix}$$

Very simple Hopfield network



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

The behavior of a Hopfield network can depend on the update order.

- Computations can oscillate if neurons are updated in parallel.
- Computations always converge if neurons are updated sequentially.

Parallel update of neuron activations

	u_1	u_2
input phase	$-\mathbf{1}$	$\mathbf{1}$
work phase	$\mathbf{1}$	$-\mathbf{1}$
	$-\mathbf{1}$	$\mathbf{1}$
	$\mathbf{1}$	$-\mathbf{1}$
	$-\mathbf{1}$	$\mathbf{1}$
	$\mathbf{1}$	$-\mathbf{1}$
	$-\mathbf{1}$	$\mathbf{1}$

- The computations oscillate, no stable state is reached.
- Output depends on when the computations are terminated.

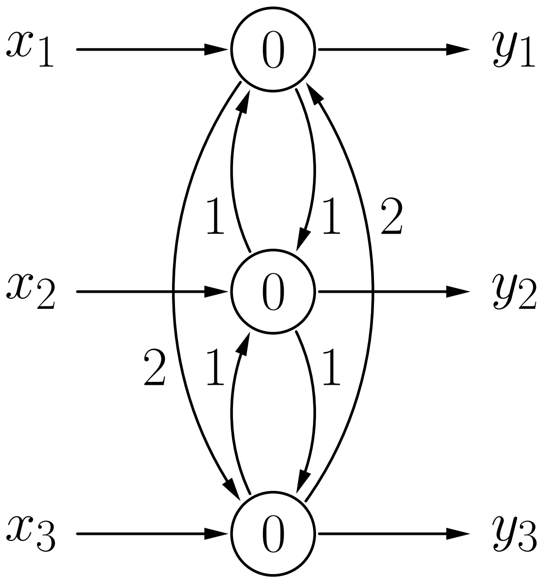
Sequential update of neuron activations

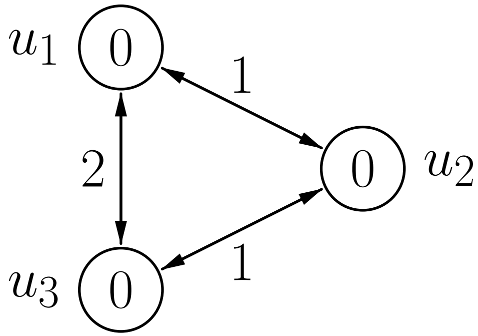
	u_1	u_2
input phase	$\mathbf{-1}$	$\mathbf{1}$
work phase	$\mathbf{1}$	1
	1	$\mathbf{1}$
	$\mathbf{1}$	1
	1	$\mathbf{1}$

	u_1	u_2
input phase	$\mathbf{-1}$	$\mathbf{1}$
work phase	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1
	-1	$\mathbf{-1}$
	$\mathbf{-1}$	-1

- Regardless of the update order a stable state is reached.
- Which state is reached depends on the update order.

Simplified representation of a Hopfield network

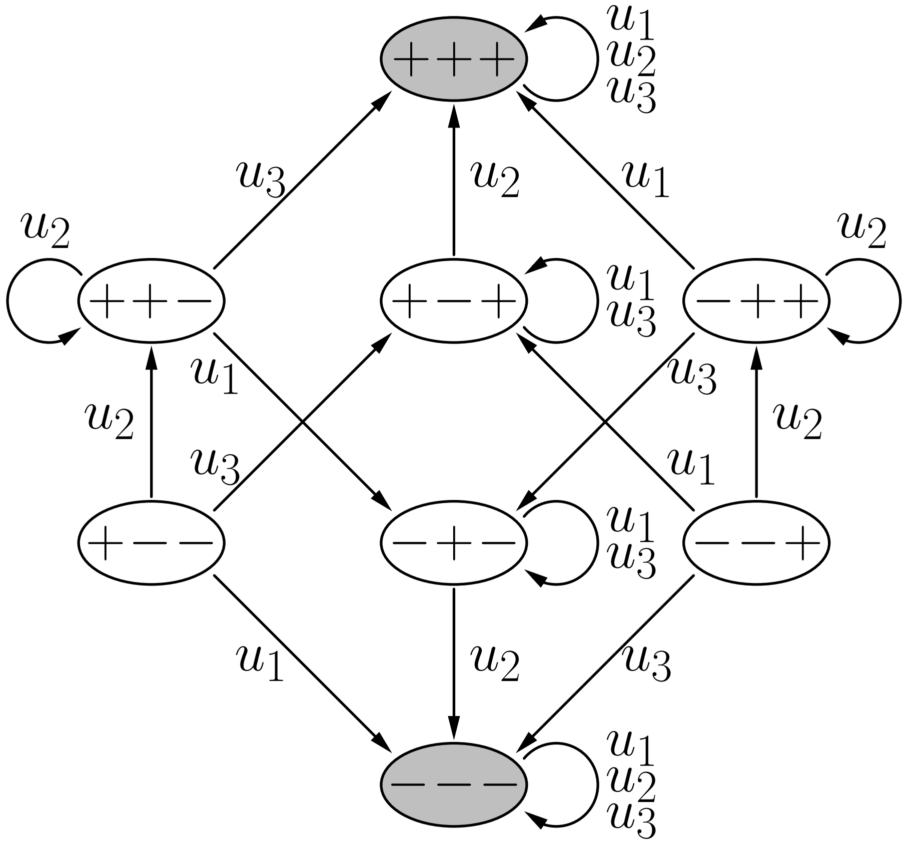




$$\mathbf{W} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

- Symmetric connections between neurons are combined.
- Inputs and outputs are not explicitely represented.

Graph of activation states and transitions



Convergence Theorem: If the activations of the neurons of a Hopfield network are updated sequentially (asynchronously), then a stable state is reached in a finite number of steps.

If the neurons are traversed cyclically in an arbitrary, but fixed order, at most $n \cdot 2^n$ steps (updates of individual neurons) are needed, where n is the number of neurons of the Hopfield network.

The proof is carried out with the help of an **energy function**.
The energy function of a Hopfield network with n neurons $u_1, \ldots, u_n$ is

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

Hopfield Networks: Convergence

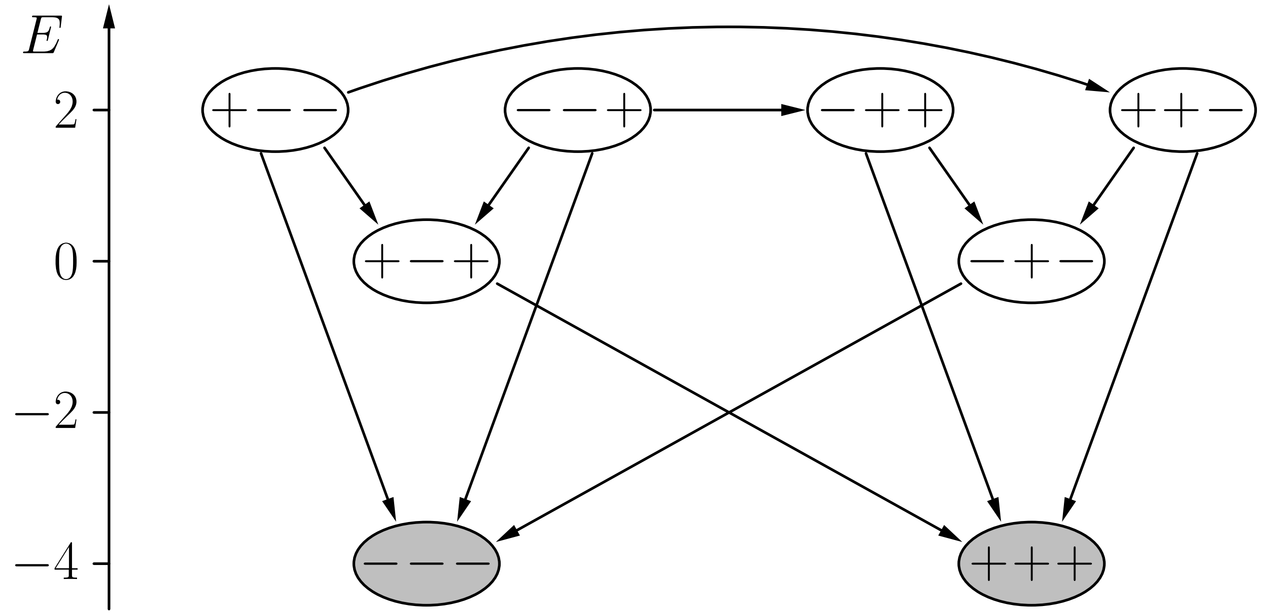
Consider the energy change resulting from an update that changes an activation:

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

- $\operatorname{net}_u < \theta_u$: Second factor is less than 0.
  $\operatorname{act}_u^{(\text{new})} = -1$ and $\operatorname{act}_u^{(\text{old})} = 1$, therefore first factor greater than 0.
  Result: $\Delta E < 0$.

- $\operatorname{net}_u \geq \theta_u$: Second factor greater than or equal to 0.
  $\operatorname{act}_u^{(\text{new})} = 1$ and $\operatorname{act}_u^{(\text{old})} = -1$, therefore first factor less than 0.
  Result: $\Delta E \leq 0$.

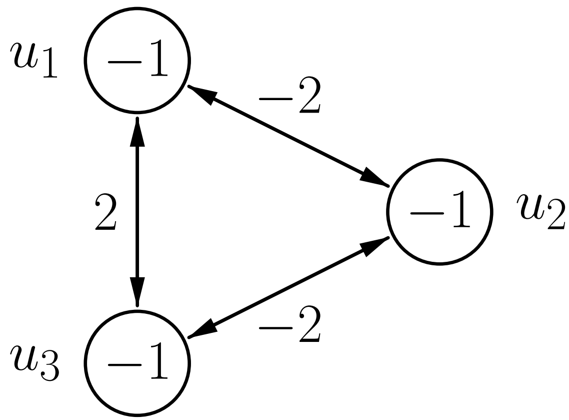
Arrange states in state graph according to their energy

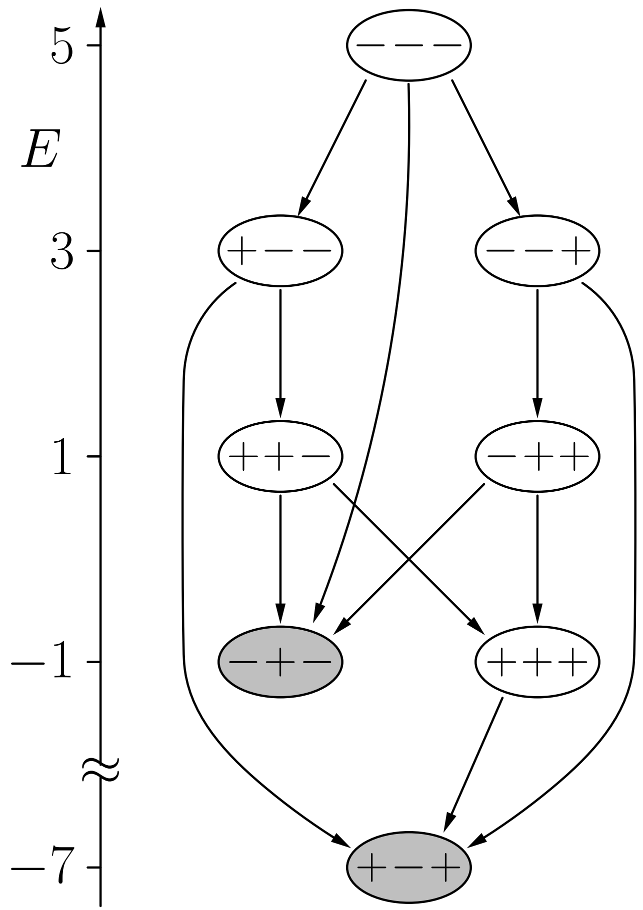


Energy function for example Hopfield network:

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

The state graph need not be symmetric





Physical interpretation: Magnetism

A Hopfield network can be seen as a (microscopic) model of magnetism (so-called Ising model, [Ising 1925]).

physical	neural
atom	neuron
magnetic moment (spin)	activation state
strength of outer magnetic field	threshold value
magnetic coupling of the atoms	connection weights
Hamilton operator of the magnetic field	energy function

Idea: Use stable states to store patterns

First: Store only one pattern $\vec{x} = (\text{act}_{u_1}^{(l)}, \ldots, \text{act}_{u_n}^{(l)})^\top \in \{-1, 1\}^n$, $n \geq 2$, i.e., find weights, so that pattern is a stable state.

Necessary and sufficient condition:

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

where

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

with

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

If $\vec{\theta} = \vec{0}$ an appropriate matrix $\mathbf{W}$ can easily be found. It suffices

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

Algebraically: Find a matrix $\mathbf{W}$ that has a positive eigenvalue w.r.t. $\vec{x}$.

Choose

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

where $\vec{x}\vec{x}^{\,T}$ is the so-called **outer product**.

With this matrix we have

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

Hebbian learning rule [Hebb 1949]

Written in individual weights the computation of the weight matrix reads:

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

- Originally derived from a biological analogy.
- Strengthen connection between neurons that are active at the same time.

Note that this learning rule also stores the complement of the pattern:

With $\mathbf{W}\vec{x} = (n-1)\vec{x}$ it is also $\mathbf{W}(-\vec{x}\,) = (n-1)(-\vec{x}\,)$.

Storing several patterns

Choose

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

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

If patterns are orthogonal, we have

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

and therefore

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

Storing several patterns

Result: As long as $m < n$, $\vec{x}$ is a stable state of the Hopfield network.

Note that the complements of the patterns are also stored.

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

But: Capacity is very small compared to the number of possible states (2^n).

Non-orthogonal patterns:

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

Example: Store patterns $\vec{x}_1 = (+1, +1, -1, -1)^\top$ and $\vec{x}_2 = (-1, +1, -1, +1)^\top$.

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

where

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

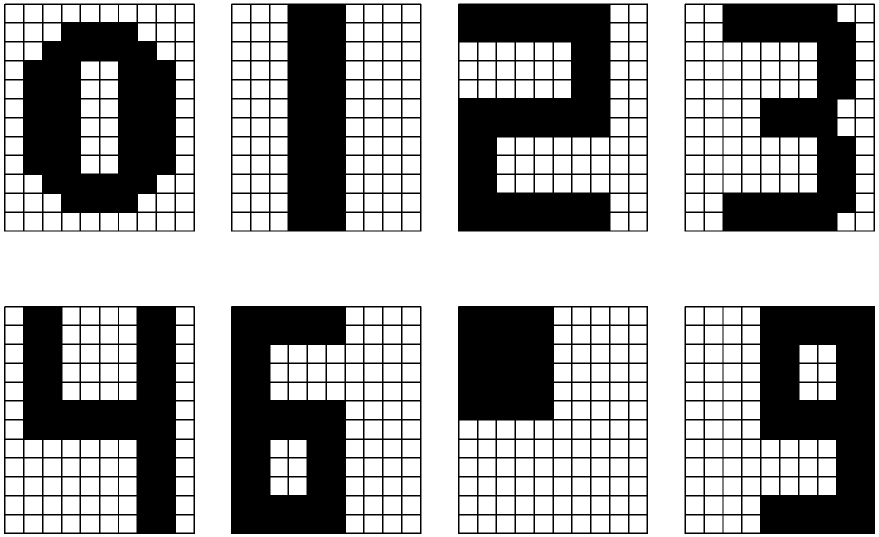
The full weight matrix is:

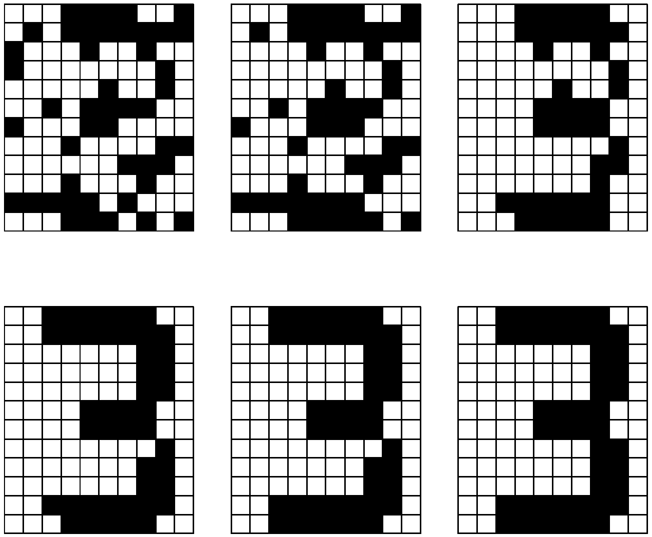
$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{pmatrix}.$$

Therefore it is

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

Example: Storing bit maps of numbers





- Left: Bit maps stored in a Hopfield network.
- Right: Reconstruction of a pattern from a random input.

Training a Hopfield network with the Delta rule

Necessary condition for pattern $\vec{x}$ being a stable state:

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

with the standard threshold function

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

Training a Hopfield network with the Delta rule

Turn weight matrix into a weight vector:

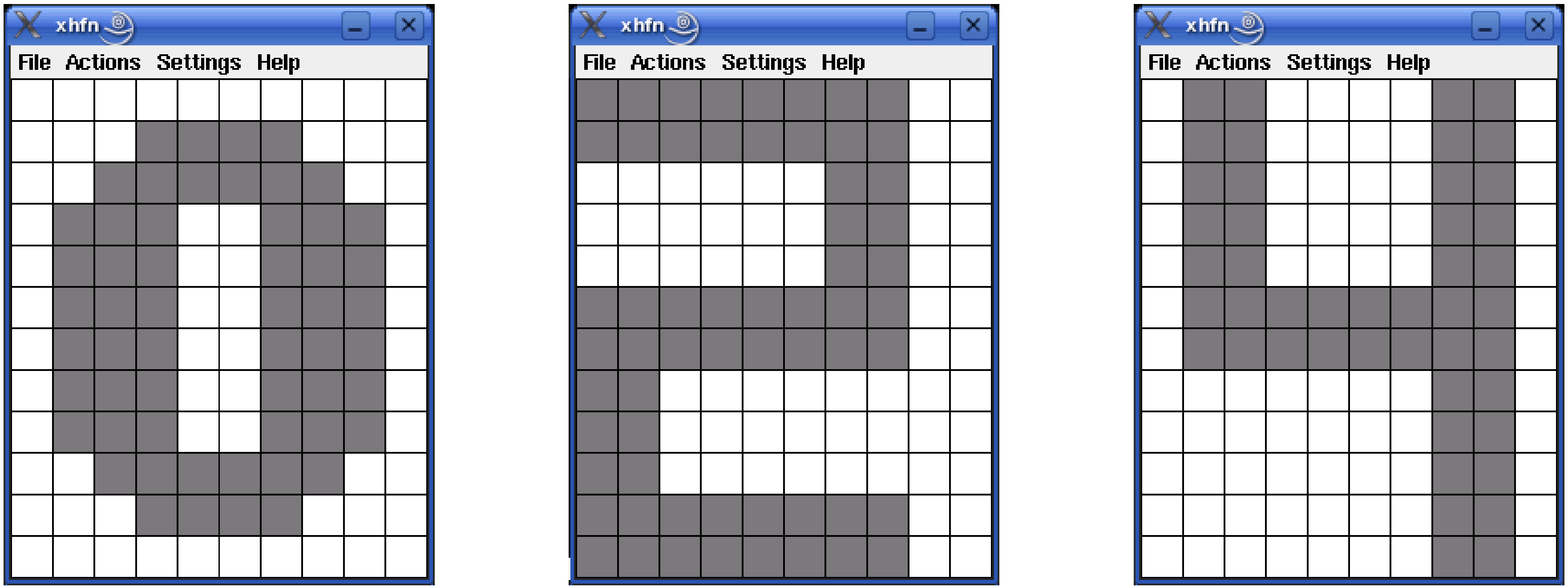
$$\begin{array}{rcccccl}
\vec{w} = (& w_{u_1 u_2}, & w_{u_1 u_3}, & \ldots, & w_{u_1 u_n}, & & \\
 & & w_{u_2 u_3}, & \ldots, & w_{u_2 u_n}, & & \\
 & & & \ddots & \vdots & & \\
 & & & & w_{u_{n-1} u_n}, & & \\
 & -\theta_{u_1}, & -\theta_{u_2}, & \ldots, & -\theta_{u_n} & &).
\end{array}$$

Construct input vectors for a threshold logic unit

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

Apply Delta rule training until convergence.

Demonstration Software: xhfn/whfn



Demonstration of Hopfield networks as associative memory:

- Visualization of the association/recognition process
- Two-dimensional networks of arbitrary size
- http://www.borgelt.net/hfnd.html

Use energy minimization to solve optimization problems

General procedure:

- Transform function to optimize into a function to minimize.
- Transform function into the form of an energy function of a Hopfield network.
- Read the weights and threshold values from the energy function.
- Construct the corresponding Hopfield network.
- Initialize Hopfield network randomly and update until convergence.
- Read solution from the stable state reached.
- Repeat several times and use best solution found.

Hopfield Networks: Activation Transformation

A Hopfield network may be defined either with activations -1 and 1 or with activations 0 and 1. The networks can be transformed into each other.

From $\text{act}_u \in \{-1, 1\}$ to $\text{act}_u \in \{0, 1\}$:

$$\begin{aligned} w_{uv}^0 &= 2w_{uv}^- \qquad\qquad \text{and} \\ \theta_u^0 &= \theta_u^- + \sum_{v \in U - \{u\}} w_{uv}^- \end{aligned}$$

From $\text{act}_u \in \{0, 1\}$ to $\text{act}_u \in \{-1, 1\}$:

$$\begin{aligned} w_{uv}^- &= \frac{1}{2} w_{uv}^0 \qquad\qquad \text{and} \\ \theta_u^- &= \theta_u^0 - \frac{1}{2} \sum_{v \in U - \{u\}} w_{uv}^0. \end{aligned}$$

Combination lemma: Let two Hopfield networks on the same set U of neurons with weights $w_{uv}^{(i)}$, threshold values $\theta_u^{(i)}$ and energy functions

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

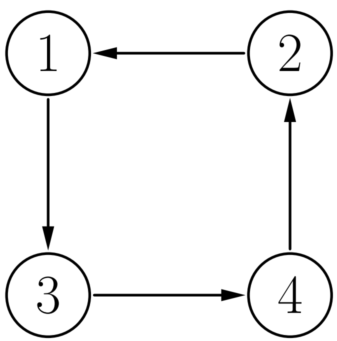
$i = 1, 2,$ be given. Furthermore let $a, b \in \mathbb{R}$. Then $E = aE_1 + bE_2$ is the energy function of the Hopfield network on the neurons in U that has the weights $w_{uv} = aw_{uv}^{(1)} + bw_{uv}^{(2)}$ and the threshold values $\theta_u = a\theta_u^{(1)} + b\theta_u^{(2)}$.

Proof: Just do the computations.

Idea: Additional conditions can be formalized separately and incorporated later.

Example: Traveling salesman problem

Idea: Represent tour by a matrix.



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

An element a_{ij} of the matrix is 1 if the i-th city is visited in the j-th step and 0 otherwise.

Each matrix element will be represented by a neuron.

Minimization of the tour length

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

Double summation over steps (index i) needed:

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

where

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

Symmetric version of the energy function:

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

Additional conditions that have to be satisfied:

- Each city is visited on exactly one step of the tour:

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

  i.e., each column of the matrix contains exactly one 1.

- On each step of the tour exactly one city is visited:

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

  i.e., each row of the matrix contains exactly one 1.

These conditions are incorporated by finding additional functions to optimize.

Formalization of first condition as a minimization problem:

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

Double summation over cities (index i) needed:

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

Resulting energy function:

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

Second additional condition is handled in a completely analogous way:

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

Combining the energy functions:

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

From the resulting energy function we can read the weights

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

and the threshold values:

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

Problem: Random initialization and update until convergence not always leads to a matrix that represents a tour, leave alone an optimal one.