Neural networks

Doc. RNDr. Iveta Mrázová, CSc.

Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics

Charles University in Prague

Neural networks

– Associative memories –
– BAM and the Hopfield model –

Doc. RNDr. Iveta Mrázová, CSc.

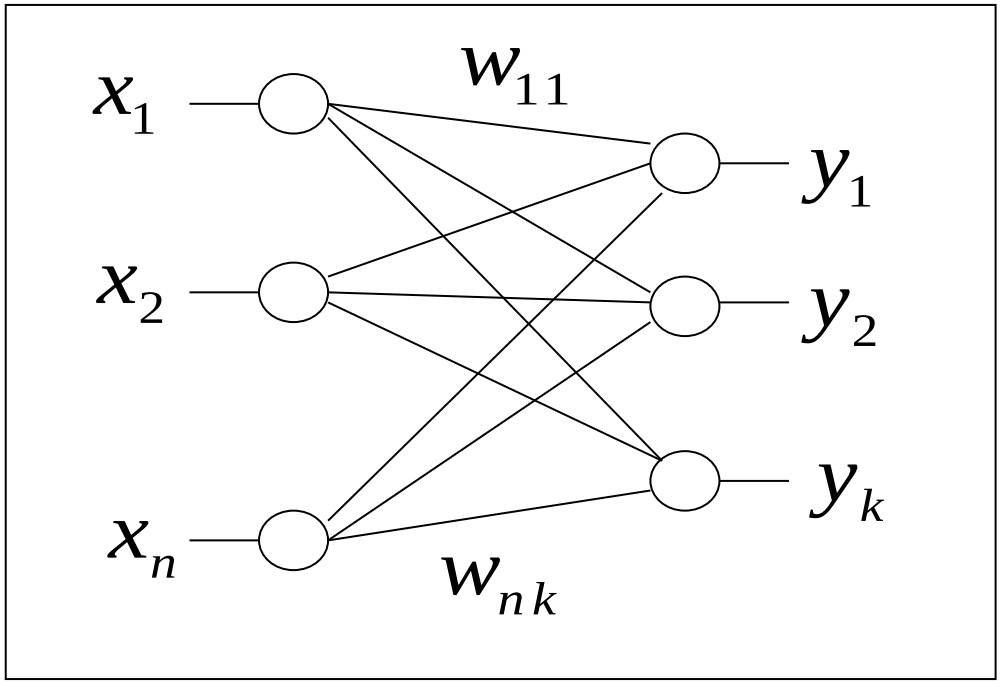
Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics

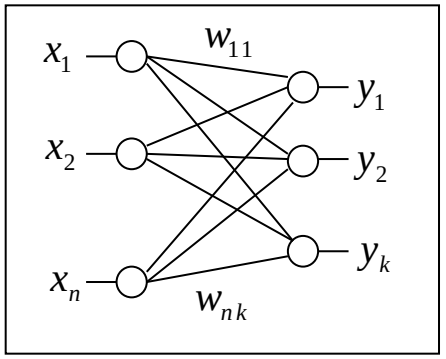
Charles University in Prague

Bidirectional associative memory

BAM – a synchronous associative model with bidirectional edges

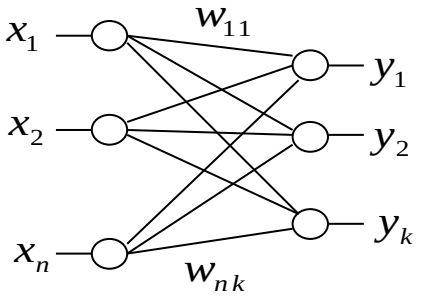


Bidirectional associative memory (2)



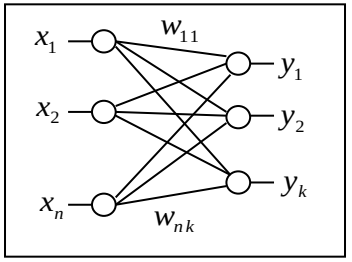
- **Recurrent associative memory**
  - Consists of 2 layers of units, which send information recursively between them.
  - The input layer sends the result of its computation to the output layer – by the weights.
  - The output layer then returns the result of its computation back to the input layer – by the same weights.
- **Question**: Will the network achieve a stable state, in which the information being sent back and forth does not change after a few iterations?

BAM ≈ Bidirectional Associative Memory (3)



- Resonance network
- **sgn** transfer function
- Information is coded using bipolar values
- Network maps an *n*-dimensional vector $\vec{\mathbf{x}}_0$ to a *k*-dimensional vector $\vec{\mathbf{y}}_0$
- The weight matrix of the network is the $n \times k$ matrix $\mathbf{W}$.
  - After the first passage, we obtain: $\vec{\mathbf{y}}_0 = \mathrm{sgn}\left(\vec{\mathbf{x}}_0 \mathbf{W}\right)$
  - After the feedback passage, the input will be: $\vec{\mathbf{x}}_1 = \mathrm{sgn}\left(\mathbf{W}\vec{\mathbf{y}}_0^{\mathrm{T}}\right)$
  - After the next forward passage, the output will be: $\vec{\mathbf{y}}_1 = \mathrm{sgn}\left(\vec{\mathbf{x}}_1 \mathbf{W}\right)$

BAM ≈ Bidirectional Associative Memory (4)



- After ***m*** iterations, we have ***m + 1*** pairs of vectors $(\vec{\mathbf{x}}_0, \vec{\mathbf{y}}_0), \ldots, (\vec{\mathbf{x}}_m, \vec{\mathbf{y}}_m)$, which fulfill the condition:

$$\vec{\mathbf{y}}_i = \mathrm{sgn}\left(\vec{\mathbf{x}}_i \mathbf{W}\right) \quad \text{and} \quad \vec{\mathbf{x}}_{i+1} = \mathrm{sgn}\left(\mathbf{W}\vec{\mathbf{y}}_i^{\mathrm{T}}\right)$$

- **Question:** Will the system find after some iterations a fixed point $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ such that it holds

$$\vec{\mathbf{y}} = \mathrm{sgn}\left(\vec{\mathbf{x}}\mathbf{W}\right) \quad \text{and} \quad \vec{\mathbf{x}} = \mathrm{sgn}\left(\mathbf{W}\vec{\mathbf{y}}^{\mathrm{T}}\right) \quad ?$$

BAM ≈ Bidirectional Associative Memory (5)

→ if a pair of vectors $(\vec{x}, \vec{y})$ is given and we want to set the weights of a BAM such that this pair of vectors will represent its fixed point, Hebbian learning can be used to compute an adequate weight matrix:

$$W = \vec{x}^T \vec{y}$$

→ $$\vec{y} = \mathrm{sgn}(\vec{x}W) = \mathrm{sgn}(\vec{x}\vec{x}^T \vec{y}) = \mathrm{sgn}(\|\vec{x}\|^2 \vec{y}) = \vec{y}$$

and also:

$$\vec{x}^T = \mathrm{sgn}(W \vec{y}^T) = \mathrm{sgn}(\vec{x}^T \vec{y} \vec{y}^T) = \mathrm{sgn}(\vec{x}^T \|\vec{y}\|^2) = \vec{x}^T$$

BAM ≈ Bidirectional Associative Memory (6)

If we want to store several patterns $(\vec{x}_1,\vec{y}_1),\ldots,(\vec{x}_m,\vec{y}_m)$, Hebbian learning will be more efficient, if the vectors $\vec{x}_1,\ldots,\vec{x}_m$ and $\vec{y}_1,\ldots,\vec{y}_m$ are pairwise orthogonal (negligible „crosstalk“)

→ for ***m*** pairs of vectors the matrix ***W*** will be set to:

$$W = \vec{x}_1^{\,T}\vec{y}_1 + \vec{x}_2^{\,T}\vec{y}_2 + \ldots + \vec{x}_m^{\,T}\vec{y}_m$$

→ a bidirectional associative memory can be used also to build autoassociative networks, because the weight matrices produced by the Hebb rule (or when computing the pseudoinverse) are symmetric

($X = X\,W$ and $X^T = W\,X^T$)

Energy function for BAM

- Assume that a BAM is given for which the vector pair $(\vec{x}, \vec{y})$ is a stable state
- The initial vector presented to the network from the left is $\vec{x}_0$ (and after some iterations, the network shall converge to $(\vec{x}, \vec{y})$)
- The vector $\vec{y}_0$ is computed according to:

$$\vec{y}_0 = \mathrm{sgn}(\vec{x}_0 W)$$

- The output vector $\vec{y}_0$ is now used for a new iteration (from the right)

Energy function for BAM (2)

- Excitation of the neurons on the left is determined by the excitation vector $\vec{e}^T = W \vec{y}_0^T$

  $\rightarrow$ $(\vec{x}_0, \vec{y}_0)$ corresponds to a stable state of the network, if $\mathrm{sgn}(\vec{e}) = \vec{x}_0$

  $\rightarrow$ i.e., if $\vec{e}$ is close enough

  ($\rightarrow$ the scalar product $\vec{x}_0 \ \vec{e}^T$ should be larger than for other vectors of the same length but further away from $\vec{x}_0$)

Energy function for BAM (3)

→ The product $E = -\vec{x}_0\ \vec{e}^T = -\vec{x}_0 W\ \vec{y}_0^T$ is thus smaller, if the vector $W\vec{y}_0$ lies closer to $\vec{x}_0$

→ **a kind of index of convergence to stable states of an associative memory**

***E* ~ energy function**

Local minima of the energy function correspond to stable states

Energy function for BAM (4)

Definition:

Let W be the weight matrix of a BAM and the output $\vec{y}_i$ of the right layer of neurons is computed in the i – th iteration as $\vec{y}_i = \mathrm{sgn}(\vec{x}_i W)$ and the output $\vec{x}_{i+1}$ of the left layer of neurons is computed as $\vec{x}_{i+1} = \mathrm{sgn}(W \vec{y}_i^T)$

The energy function of a BAM is then given by:

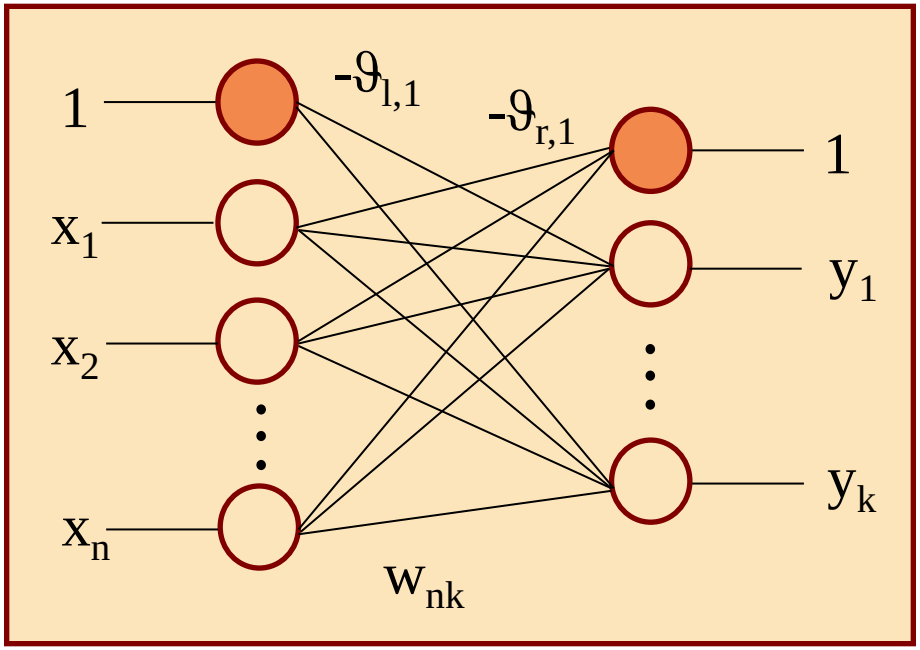
$$E(\vec{x}_i, \vec{y}_i) = -\frac{1}{2} \vec{x}_i \, W \, \vec{y}_i^T$$

Generalized energy function

Considers thresholds and a stepwise transfer function

- Each n – dimensional vector $\vec{x}$ will be transformed into the vector $(x_1, \ldots, x_n, 1)$
- Each k – dimensional vector $\vec{y}$, $y_j = \sum_i w_i x_i - \vartheta_{l,j}$ will be transformed into the vector $(y_1, \ldots, y_k, 1)$
- The weight matrix W will be extended to a new matrix W' with an additional row and column

Generalized energy function (2)



- Negative thresholds of the neurons in the right layer of the BAM form the ***(n+1)***–th row of ***W´***
- Negative thresholds of the neurons in the left layer form the ***(k+1)***–th column of ***W´***
- The entry ***(n+1 , k+1)*** of the matrix ***W´*** is ***0***

Generalized energy function (3)

- This transformation is equivalent to introduction of an additional unit with output ***1*** into both layers
  - The weights of these additional neurons correspond to negative thresholds of neurons fed by this information

→ **Energy function of the extended BAM:**

$$E(\vec{x}_i, \vec{y}_i) = -\frac{1}{2}\,\vec{x}_i\, W\, \vec{y}_i^{\,T} + \frac{1}{2}\vec{\theta}_r \vec{y}_i^{\,T} + \frac{1}{2}\vec{x}_i \vec{\theta}_l^{\,T}$$

$\vec{\theta}_l^{\,T}$ …. Vector of thresholds of the ***k*** neurons (in the left layer)

$\vec{\theta}_r$ …. Vector of thresholds of the ***n*** neurons (in the right layer)

Asynchronous BAM- networks

- Each unit computes its excitation at random
- Changes its state to 1 ot -1 independently of the others (but according to the sign of its total excitation)
- Probability of two units firing simul-taneously is zero
- Assumption: the state of a unit is not changed, if the total excitation is zero

Asynchronous BAM-networks (2)

BAM arrives at a stable state after a finite number of iterations (sequential choice of neurons)

- **Stable state**
  ~ vector pair $(\vec{x}, \vec{y})$; $\vec{y} = \mathrm{sgn}(\vec{x}W)$ and $\vec{x}^{\mathrm{T}} = \mathrm{sgn}(W\vec{y}^{T})$

Proposition:

A bidirectional associative memory with an arbitrary weight matrix $\boldsymbol{W}$ reaches a stable state in a finite number of iterations using either synchronous or asynchronous updates.

Asynchronous BAM- networks (3)

Proof:

- For vectors $\vec{x} = (x_1, x_2, \ldots, x_n)$ and $\vec{y} = (y_1, y_2, \ldots, y_k)$ and an $\boldsymbol{n \times k}$ weight matrix $\boldsymbol{W = \{w_{ij}\}}$ the energy function $E(\vec{x}, \vec{y})$ equals to:

$$E(\vec{x}, \vec{y}) = -\frac{1}{2}(x_1, \ldots, x_n)\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}$$

Asynchronous BAM- networks (4)

Proof (continue)**:**

- Product of the ***i***–th row of W and $\vec{y}^T$ represents the excitation of the ***i***–th neuron in the left layer g_i

  → then for the excitation vector of the left layer $(g_1,\ldots,g_n)$:

$$E(\vec{x},\vec{y}) = -\frac{1}{2}(x_1,\ldots,x_n)\begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix}$$

  → similarly it holds then for the right layer and its excitation vector $(e_1, \ldots, e_k)$:

$$E(\vec{x},\vec{y}) = -\frac{1}{2}(e_1,\ldots,e_k)\begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix}$$

Asynchronous BAM- networks (5)

Proof (continue):

- Energy function can be written in two equivalent forms:

$$E(\vec{x}, \vec{y}) = -\frac{1}{2}\sum_{i=1}^{k} e_i y_i$$

$$\text{and} \quad E(\vec{x}, \vec{y}) = -\frac{1}{2}\sum_{i=1}^{n} g_i x_i$$

- In asynchronous networks at each iteration we randomly select a neuron from the left or right layer:
  - For the selected neuron, excitation and new state are computed

Asynchronous BAM- networks (6)

Proof (continue)**:**

- If the state remains the same, also the energy of the network will not change
- The state of neuron i in the left layer will change only when the excitation g_i has a different sign than its present state x_i
- Since the other neurons do not change their states (asynchronous dynamics), the difference between the previous energy $E(\vec{x}, \vec{y})$ and the new energy $E(\vec{x}', \vec{y})$ will be:

$$E(\vec{x}, \vec{y}) - E(\vec{x}', \vec{y}) = -\frac{1}{2} g_i \left(x_i - x_i' \right)$$

Asynchronous BAM- networks (7)

Proof (continue)**:**

- Since $x_i - x_i'$ has a different sign than g_i, it follows that:

$$E(\vec{x}, \vec{y}) - E(\vec{x}', \vec{y}) > 0$$

(if $x_i - x_i'$ would have the same sign like g_i, no change of the neuron state had been occurred)

→ The new state $(\vec{x}', \vec{y})$ has thus a lower energy than the original state $(\vec{x}, \vec{y})$

- Analogically for the neurons from the right layer:

$$E(\vec{x}, \vec{y}) - E(\vec{x}, \vec{y}') > 0$$

(Whenever the state of the neurn has been flipped.)

Asynchronous BAM- networks (8)

Proof (continue)**:**

- Any update of the network state reduces the total energy
- Since there are only a finite number of possible combinations of bipolar states, the process must stop at some state $(\vec{a}, \vec{b})$, whose energy cannot be further reduced

→ **the network has fallen into a local minimum of the energy function and the state $(\vec{a}, \vec{b})$ is an attractor of the system**

QED

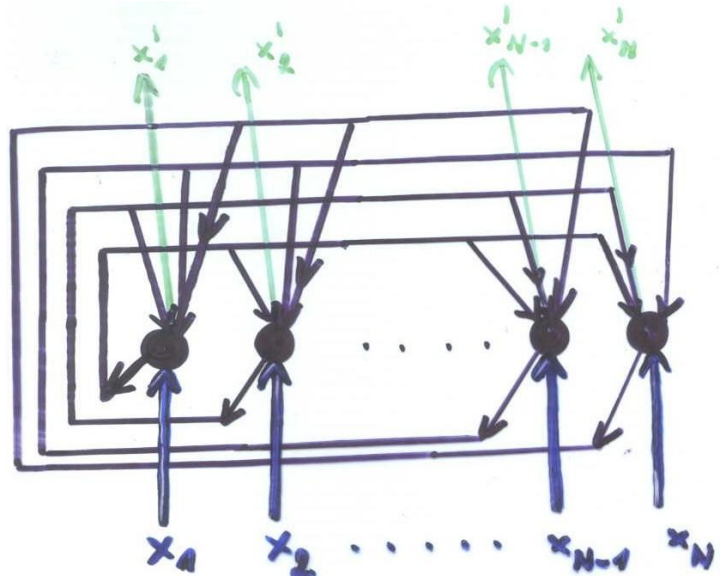
Asynchronous BAM- networks (9)

Remark:

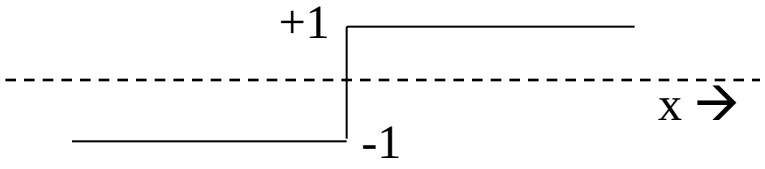
- Proposition holds also for synchronous networks

→ **any given real matrix *W* possesses bidirectional stable bipolar states**

Hopfield networks



Hard-limiting transfer function: f_h



- ***n*** neurons with the hard-limiting transfer function
- Bipolar inputs ant outputs ***{ + 1 , - 1 }***
- Synaptic weights w_{ij} (between all the neurons)
- ***m*** training patterns (classes)
- Supervised training
- Recall
- Applications:
  - Associative memory
  - Optimization tasks

The Hopfield model (bipolar)

Step 1: Training – set synaptic weights according to:

$$w_{ij} = \begin{cases} \sum_{s=1}^{m} x_i^s x_j^s & \text{for } i \neq j \\ 0 & \text{for } i = j \end{cases}$$

w_{ij} …. Synaptic weight between neuron i and j

$x_i^s \in \{-1,+1\}$ …. i–th component of the s–th pattern

$1 \leq i,j \leq n$

The Hopfield model (bipolar) (2)

Step 2: Initialization – present a new input pattern:

$$y_i(0) = x_i \qquad 1 \le i \le n$$

$y_i(t)$ …. Output of neuron i at time t

$x_i \in \{-1,+1\}$ …. i – th element of the presented pattern

Step 3: Iteration

$$y_j(t+1) = f_h\left[\sum_{i=1}^{n} w_{ij}\, y_i(t)\right] \qquad 1 \le j \le n$$

f_h …. Hard-limiting transfer function

The Hopfield model (bipolar) (3)

The iterative process is repeated during recall until neuron outputs stabilize. Neuron outputs then represent that training pattern, which best corresponds to the presented (new) pattern.

Step 4: Go to Step 2.

The Hopfield model (bipolar) (4)

Convergence (Hopfield):

- Symmetric weights: $w_{ij} = w_{ji}$
- Asynchronous output updates in single neurons

Drawbacks:

- Capacity ***(m < 0.15 n)*** ***n / 2 log n***
- Stability (→ orthogonalization)

The Hopfield model – example

Training:

- Patterns: [-1, -1, 1, 1]
  [1, -1, 1, -1]

- Weight setting:

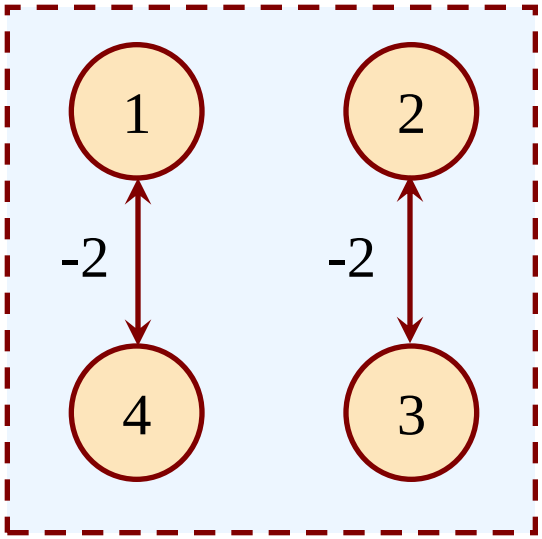
$$w_{ij} = \sum_{m=1}^{M} x_i^{(m)} \, x_j^{(m)} \qquad i \neq j$$

$$w_{ij} = 0 \qquad i = j$$

The Hopfield model – example (2)

- Weight matrix:

$$W = \begin{bmatrix} 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{bmatrix}$$



Recall:

- Pattern: [-1, -1, 1, -1]

[1, -1, 1, -1] [-1, -1, 1, 1]

The Hopfield model - recall

When initialized with $\vec{x}_1$, the vector of potentials is:

$$\vec{\xi} = \vec{x}_1 \cdot W = \vec{x}_1 \cdot \left(\vec{x}_1^T \vec{x}_1 + \ldots + \vec{x}_m^T \vec{x}_m - mI \right) =$$

$$= \underbrace{\vec{x}_1 \vec{x}_1^T}_{=n} \vec{x}_1 + \underbrace{\vec{x}_1 \vec{x}_2^T}_{=\alpha_{12}} \vec{x}_2 + \ldots + \underbrace{\vec{x}_1 \vec{x}_m^T}_{=\alpha_{1m}} \vec{x}_m - m\vec{x}_1 I =$$

$$= (n-m)\vec{x}_1 + \underbrace{\sum_{j=2}^{m} \alpha_{1j} \vec{x}_j}_{PERTURBATION}$$

The Hopfield model – recall (2)

$\alpha_{12}, \ldots, \alpha_{1m}$ …. Scalar product of $\vec{x}_1$ with each of the other vectors $\vec{x}_2, \ldots, \vec{x}_m$

→ **State $\vec{x}_1$ is stable, if $m < n$ and perturbation $\sum_{j=2}^{m} \alpha_{1j} \vec{x}_j$ is small**

$$\left(\Rightarrow \operatorname{sgn}\left(\vec{\xi} \right) = \operatorname{sgn}\left(\vec{x}_1 \right) \right)$$

➔ **A small number of orthogonal patterns**

The Hopfield model – recall (3)

- States of neurons not selected for update remain the same
- Random selection for update
- Neurons are fully interconnected
- Symmetric weights: $w_{ij} = w_{ji}$
- $w_{ii} = 0$

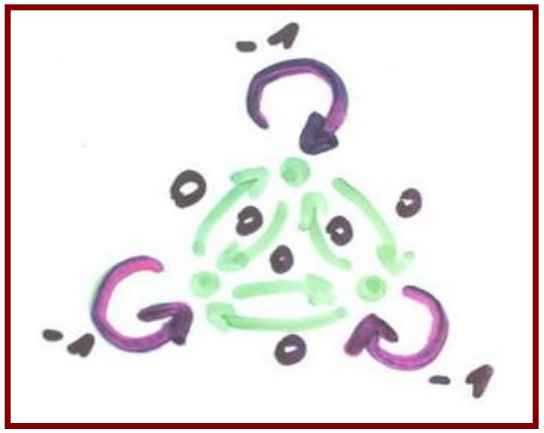
→ Convergence to a stable solution during recall
- necessary condition:

symmetric weight matrix with zero diagonal and asynchronous dynamics

The Hopfield model – examples

- A weight matrix with non zero diagonal does not have to yield stable states

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

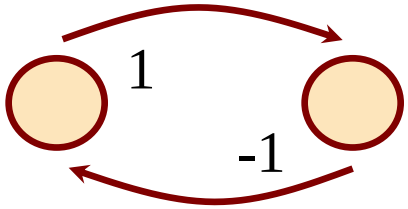


  - Synchronous dynamics: (-1, -1, -1) ↔ (1, 1, 1)
  - Asynchronous dynamics:
    - Random selection of one out of eight possible patterns

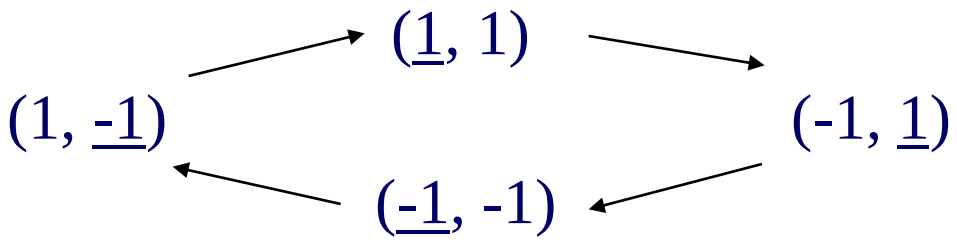
The Hopfield model – examples (2)

- Asymmetric weight matrix:

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



  - Asynchronous dynamics:



cyclic changes of states

Energy function

Energy function $E(\vec{x})$ of a Hopfield network with *n* neurons and the weight matrix *W* shows the energy of the network in state $\vec{x}$:

$$E(\vec{x}) = -\frac{1}{2}\,\vec{x}\,W\,\vec{x}^{T} = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}\,x_i\,x_j$$

(Similarly also for networks with thresholds:

$$E(\vec{x}) = -\frac{1}{2}\,\vec{x}\,W\,\vec{x}^{T} + \vec{\theta}\,\vec{x}^{T} =$$

$$= -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}\,x_i\,x_j + \sum_{i=1}^{n}\vartheta_i\,x_i \qquad)$$

Energy function (2)

Proposition:

A Hopfield network with *n* neurons and asynchronous dynamics, which starts from any given network state, eventually reaches a stable state at a local minimum of the energy function.

Proof (idea):

- Initial state
  - Presented pattern: $\vec{x} = (x_1, \ldots, \underline{x_k}, \ldots, x_n)$

Energy function (3)

Proof (idea - continue):

$$E(\vec{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}\ x_i\ x_j$$

Neuron ***k*** selected for adjustment

- if ***k*** does not change its state → $E(\vec{x})$ does not change
- if ***k*** changes its state → $\vec{x}' = \left(x_1, \ldots, \underline{x_k}', \ldots, x_n\right)$

Energy function (4)

Proof (idea - continue):

$$E(\vec{x}') = -\frac{1}{2}\sum_{\substack{j=1 \\ j\neq k}}^{n}\sum_{\substack{i=1 \\ i\neq k}}^{n} w_{ij}\, x_i\, x_j -$$

symmetric weights

$$-\sum_{i=1}^{n} w_{ik}\, x_i\, x_k' = \begin{cases} -\frac{1}{2}\sum_{j=1}^{n} w_{kj}\, x_k' x_j \\ -\frac{1}{2}\sum_{i=1}^{n} w_{ik}\, x_i\, x_k' \end{cases}$$

Energy function (5)

Proof (idea - continue):

- The difference between the old and new energies:

$$E(\vec{x}) - E(\vec{x}') = -\sum_{i=1}^{n} w_{ik}\, x_i\, x_k - \left(-\sum_{i=1}^{n} w_{ik}\, x_i\, x_k' \right) =$$

$$w_{kk} = 0 \longrightarrow = -\underbrace{\left(x_k - x_k' \right)}_{\text{different signs (otherwise no change of state would occur)}} \underbrace{\sum_{i=1}^{n} w_{ik}\, x_i}_{POTENTIAL} > 0$$

Energy function (6)

Proof (idea - continue):

→ **every time the state of a neuron is altered, the total energy of the network is reduced**

- A finite number of possible states

→ **the network must eventually reach a stable state for which the energy cannot be further reduced**

QED

Equivalence of Hebbian and perceptron learning for the Hopfield model

Sometimes Hebbian learning cannot find a weight matrix for which ***m*** given vectors are stable states (although such a matrix exists)

→ if the vectors to be stored lie near each other, the perturbation term can grow too large

→ **worse results of Hebbian learning**

Alternative: **Perceptron learning for Hopfield networks**

Equivalence of Hebbian and perceptron learning for the Hopfield model (2)

Perceptron learning in Hopfield networks

- Hopfield networks composed of neurons with a non-zero threshold and the hard-limiting transfer function
  - Neurons adopt state ***1*** for potentials greater than ***0***
  - Neurons adopt state ***- 1*** for potencials smaller than or equal to ***0***

Equivalence of Hebbian and perceptron learning for the Hopfield model (3)

- Let us consider a Hopfield network:

  n …….. the number of neurons

  $W = \{w_{ij}\}$ ….. $n \times n$ the weight matrix

  ϑ_i …….. the threshold of the neuron i

- If a vector $\vec{x} = (x_1, \ldots, x_n)$ is given to be „imprinted“ on the network, this vector will correspond to a stable state only if the network does not change its state after presenting this vector at its input

Equivalence of Hebbian and perceptron learning for the Hopfield model (4)

→ **neuron potentials should have the same sign like their previous states**

- Minus sign will be assigned to zero values
- **The following inequalities should thus hold:**

$$\begin{array}{lcll}
\text{Neuron 1:} & \operatorname{sgn}(x_1)(0 + x_2 w_{12} + \ldots + x_n w_{1n} - \vartheta_1) & > & 0 \\
\text{Neuron 2:} & \operatorname{sgn}(x_2)(x_1 w_{21} + 0 + \ldots + x_n w_{2n} - \vartheta_2) & > & 0 \\
\ldots & \ldots & \ldots & \ldots \\
\text{Neuron n:} & \operatorname{sgn}(x_n)(x_1 w_{n1} + x_2 w_{n2} + \ldots + 0 - \vartheta_1) & > & 0
\end{array}$$

Equivalence of Hebbian and perceptron learning for the Hopfield model (5)

- $w_{ij} = w_{ji}$ $\rightarrow$ $n \cdot (n-1)/2$ non-zero elements of the weight matrix and n thresholds

$\rightarrow$ let $\vec{v}$ denote a vector of dimension $n+n\cdot(n-1)/2$

(components of $\vec{v}$ correspond to the elements w_{ij} above the diagonal of the matrix W; $i < j$; and the n thresholds with minus sign)

$$\vec{v} = \left(\underbrace{w_{12}, w_{13}, \ldots, w_{1n}}_{n-1 \text{ components}}, \underbrace{w_{23}, w_{24}, \ldots, w_{2n}}_{n-2 \text{ components}}, \ldots, \underbrace{w_{n-1,n}}_{1 \text{ component}}, \underbrace{-\vartheta_1, \ldots, -\vartheta_n}_{n \text{ components}} \right)$$

Equivalence of Hebbian and perceptron learning for the Hopfield model (6)

→ transformation of the vector $\vec{x}$ into ***n*** „auxiliary“ vectors $\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_n$ of the dimension ***n+n·(n–1)/2*** :

$$\vec{z}_1 = \left(\underbrace{x_2, x_3, \ldots, x_n}_{n-1 \text{ components}}, 0, 0, \ldots, 0, \underbrace{1, 0, \ldots, 0}_{n \text{ components}} \right)$$

$$\vec{z}_2 = \left(\underbrace{x_1, 0, \ldots, 0}_{n-1 \text{ components}}, \underbrace{x_3, \ldots, x_n}_{n-2 \text{ components}}, 0, 0, \ldots, \underbrace{0, 1, \ldots, 0}_{n \text{ components}} \right)$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\vec{z}_n = \left(\underbrace{0, 0, \ldots, x_1}_{n-1 \text{ components}}, \underbrace{0, 0, \ldots, x_2}_{n-2 \text{ components}}, 0, 0, \ldots, \underbrace{0, 0, \ldots, 1}_{n \text{ components}} \right)$$

Equivalence of Hebbian and perceptron learning for the Hopfield model (7)

the components of the vectors $\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_n$ allow to write the above inequalities as:

$$\begin{array}{lcccc} \text{Neuron 1:} & \operatorname{sgn}(x_1)\ \vec{z}_1 \cdot \vec{v} & > & 0 \\ \text{Neuron 2:} & \operatorname{sgn}(x_2)\ \vec{z}_2 \cdot \vec{v} & > & 0 \\ \ldots & \ldots & \ldots & \ldots \\ \text{Neuron n:} & \operatorname{sgn}(x_n)\ \vec{z}_n \cdot \vec{v} & > & 0 \end{array}$$

Equivalence of Hebbian and perceptron learning for the Hopfield model (8)

→ to linearly separate the vectors $\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_n$ (based on ***sgn (x_i)***), we can use perceptron learning

→ Compute the weight vector $\vec{v}$ needed for the linear separation of the vectors $\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_n$ and set the weight matrix ***W***

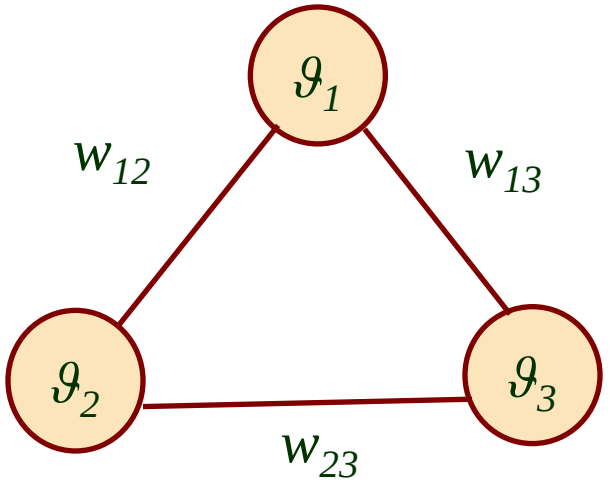
- If the Hopfield network has to „remember“ ***m*** vectors $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m$, we have to use the above transformation for every one of them

Equivalence of Hebbian and perceptron learning for the Hopfield model (9)

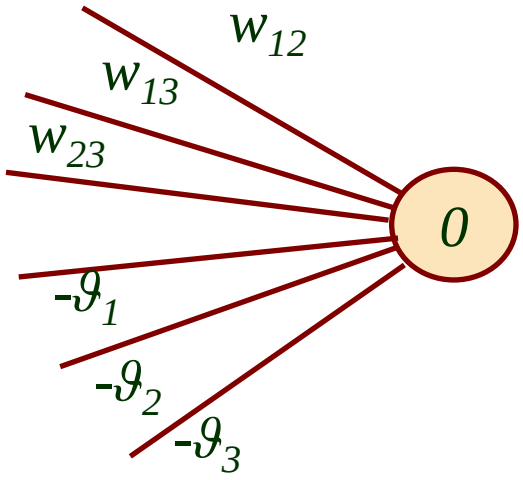
- $m \cdot n$ „auxiliary“ vectors, which must be linearly separated
- If the (auxiliary) vectors are actually linearly separable, perceptron learning will find the solution to the problem, coded as the vector $\vec{v}$

Equivalence of Hebbian and perceptron learning for the Hopfield model (10)

Example:



Training of a Hopfield network with ***n*** neurons



Perceptron learning with the input space dimension

$n+n\cdot(n-1)/2$ ($= n\cdot(n+1)/2$)

Remark: „local application of the delta-rule“

Hopfield networks used to solve optimization problems

- Binary coding: ***0 / 1***
- **Multiflop:**
  - $x_1, \ldots, x_n$ …. binary states of the individual neurons in the considered Hopfield network
  - The network should adopt a state, when exactly 1 neuron will be active; all other neurons will ne in state ***0***
  - **Objective:** find a minimum of $E(x_1, \ldots, x_n)$

$$E(x_1, \ldots, x_n) = \left(\sum_{i=1}^{n} x_i - 1 \right)^2$$

Hopfield networks used to solve optimization problems (2)

$$E(x_1,\ldots,x_n)=\left(\sum_{i=1}^{n}x_i-1\right)^2=\sum_{i=1}^{n}x_i^2+\sum_{i\neq j}^{n}x_i x_j-2\sum_{i=1}^{n}x_i+1=$$

$= x_i$ for binary states

$$=\sum_{i\neq j}^{n}x_i x_j-\sum_{i=1}^{n}x_i+1=$$

comparison with the energy function of the Hopfield model

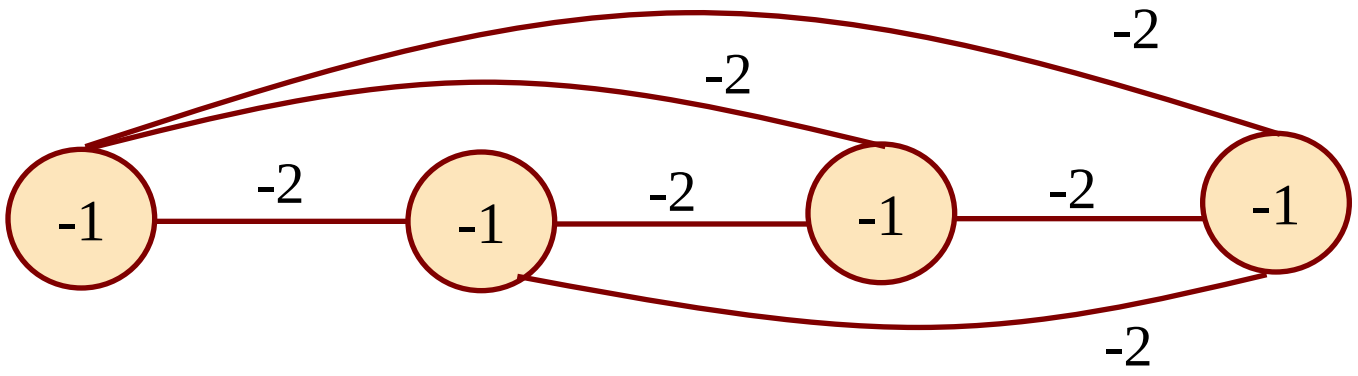
$$=-\frac{1}{2}\sum_{i\neq j}^{n}(-2)x_i x_j+\sum_{i=1}^{n}(-1)x_i+1=$$

→ **setting of network´s weights and thresholds**

Hopfield networks used to solve optimization problems (3)

$$E(x_1,\ldots,x_n) = -\frac{1}{2}\sum_{i\neq j}^{n}(-2)x_i x_j + \sum_{i=1}^{n}(-1)x_i + 1$$

setting of network´s weights and thresholds



Hopfield networks used to solve optimization problems (4)

- **The *n* rooks problem:**
  - Position ***n*** rooks in an ***n* × *n*** chessboard so that no one figure can take another

  → each rook shall be in a different row and column than the remaining ones

  → x_{ij} ……. state of the neuron at position ***ij*** of the board

  $\sum_{i=1}^{n} x_{ij}$ …. number of the states „1“ in column ***j***

  × in each column only a single „1“ is allowed

Hopfield networks used to solve optimization problems (5)

- **Minimize:**

$$E_1(x_{11},\ldots,x_{nn}) = \sum_{j=1}^{n} \underbrace{\left(\sum_{i=1}^{n} x_{ij} - 1\right)^2}_{\approx \mathrm{MULTIFLOP}}$$

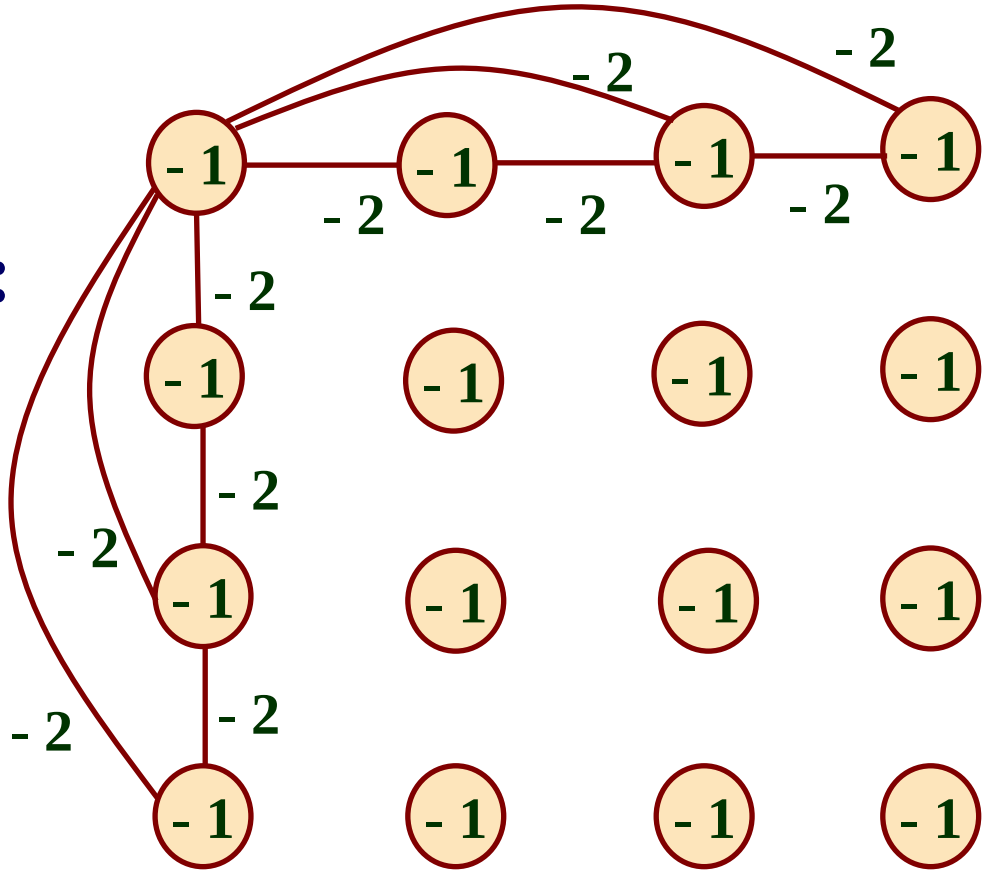
- Similarly for the rows:

$$E_2(x_{11},\ldots,x_{nn}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_{ij} - 1\right)^2$$

- **Minimize** $E = E_1 + E_2$

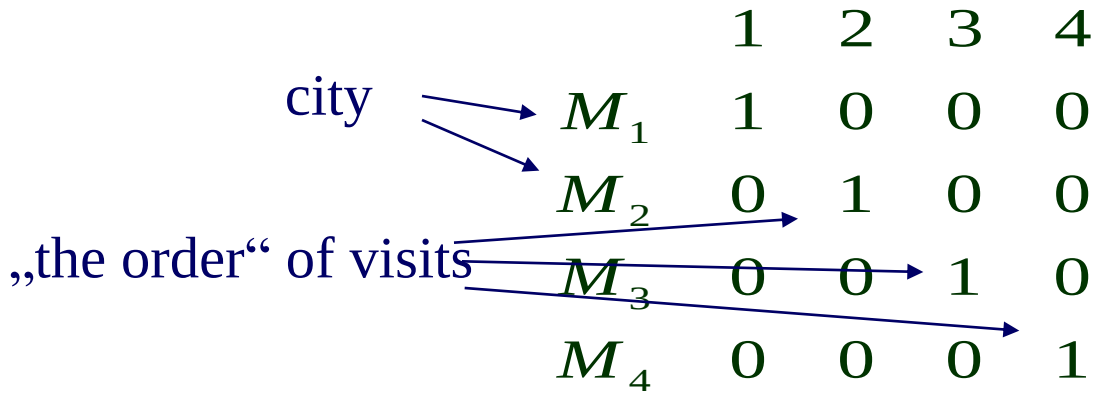
Hopfield networks used to solve optimization problems (6)

Set the weights and thresholds of the network:



Hopfield networks used to solve optimization problems (8)

- Representation using a matrix:



Convention:
(n+1)-st column is the same like ***n***
→ round trip

- x_{ik} ……neuron state ~ entry ***ik*** of the matrix
- $x_{ik} = x_{jk+1} = 1$ ... city M_i is visited in the ***k***-th step and M_j in the step ***(k+1)***
- d_{ij} …… Distance between M_i and M_j
  → add d_{ij} to the length of the round trip

Hopfield networks used to solve optimization problems (9)

- **Minimize the length of trip:** $L = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{ik} x_{jk+1}$

× a single „1“ is allowed in each row and column

→ **include the constraints for a valid round trip**

==> minimize *E:*

$$E = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{ik} x_{jk+1} + \frac{\gamma}{2} \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{n} x_{ij} - 1 \right)^2 + \sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2 \right)$$

Hopfield networks used to solve optimization problems (10)

- Set the weights and thresholds of the network as:

$w_{ik,jk+1} = - d_{ij} + t_{ik,jk+1}$; where

$t_{ik,jk+1} = - \gamma$ for neurons in the same row or column

$t_{ik,jk+1} = 0$ else

$\vartheta_{ij} = - \gamma / 2$

Stochastic models of neural networks

- Hopfield networks can be used to provide solutions to optimizatiom problems that can be expressed as the minimization of an energy function (although without guaranteeing global optimality)
- **Problem:** avoid „falling“ into local minima of the energy function

Stochastic models of neural networks (2)

Modifications of the Hopfield model:

1. strategy: increase the number of paths in the search space
   → also real-valued states become possible
   (sigmoidal transfer function)
   ==> **continuous model**

2. strategy: avoid local minima of the energy function by introducing „noise“ into the network dynamics
   → the network is occassionaly allowed to update its state despite of a temporarily increased energy level
   ==> **simulated annealing, Boltzmann machine**

Continuous Hopfield model

- The new state of neuron i selected for updating is given by: $x_i = s(u_i) = \frac{1}{1+e^{-u_i}}$

  u_i denotes the excitation (i.e., potential) of neuron i
- Additional assumption of slow changes in neuron´s excitation according to:

$$\frac{du_i}{dt} = \gamma\left(-u_i + \sum_{j=1}^{n} w_{ij} x_j\right) = \gamma\left(-u_i + \sum_{j=1}^{n} w_{ij} s(u_j)\right)$$

$\gamma > 0$ …. learning parameter

w_{ij} ……. weight between neuron i and j

Continuous Hopfield model (2)

- For a stimulation we will compute a discrete approximation of du_i which is added to the current value of u_i

  the result leads to the new state $x_i = s(u_i)$
- Asynchronous dynamics – leads to equilibrium
- **Energy function for the continuous model:**

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j + \sum_{i=1}^{n}\int_0^{x_i} s^{-1}(x)\,dx$$

Continuous Hopfield model (3)

- We have to show that the energy becomes lower after each state update; the change in time is:

$$\frac{dE}{dt} = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\,\frac{dx_i}{dt}\,x_j + \sum_{i=1}^{n} s^{-1}(x_i)\frac{dx_i}{dt}$$

- Since the network is symmetric, i.e., $w_{ij} = w_{ji}$ and simultaneously $u_i = s^{-1}(x_i)$

$$\frac{dE}{dt} = -\sum_{i=1}^{n} \frac{dx_i}{dt}\left(\sum_{j=1}^{n} w_{ij}\,x_j - u_i\right)$$

Continuous Hopfield model (4)

- Because: $\frac{du_i}{dt} = \gamma \left(\sum_{j=1}^{n} w_{ij} x_j - u_i \right)$

$$\Rightarrow \quad \frac{dE}{dt} = -\frac{1}{\gamma} \sum_{i=1}^{n} \frac{dx_i}{dt} \frac{du_i}{dt}$$

- Since $x_i = s(u_i)$

$$\frac{dE}{dt} = -\frac{1}{\gamma} \sum_{i=1}^{n} \frac{ds(u_i)}{dt} \frac{du_i}{dt} = -\frac{1}{\gamma} \sum_{i=1}^{n} \frac{ds(u_i)}{du_i} \frac{du_i}{dt} \frac{du_i}{dt} = -\frac{1}{\gamma} \sum_{i=1}^{n} s'(u_i) \left(\frac{du_i}{dt} \right)^2$$

Continuous Hopfield model (5)

- We know that $s'(x_i) > 0$

  (sigmoid is a strictly monotone function)

- Since $\gamma > 0$, $\frac{dE}{dt} \leq 0$

→ a stable state is reached when dE / dt vanishes

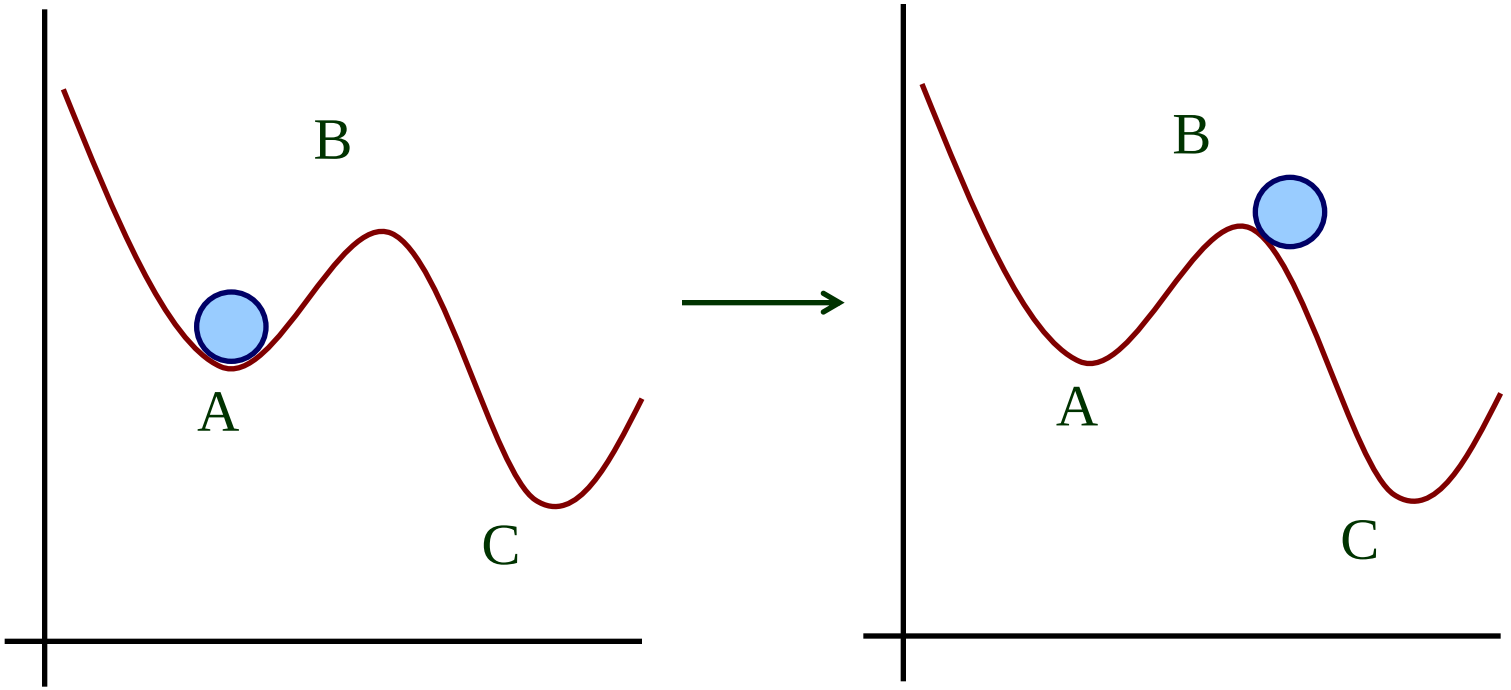
→ This happens when du_i / dt reaches the saturation region of the sigmoid where $du_i / dt \sim 0$

QED

Continuous Hopfield model (6)

- For combinatorial problems, the continuous Hopfield model can find better solutions than the discrete model
- However, in the case of really hard problems (e.g., the TSP) the continuous model has no definite advantage over the discrete model

Simulated annealing



Simulated annealing (2)

- When minimizing the energy function E, this phenomenon can be simulated as it follows:
  - the value of the variable x is changed always if the update Δx can reduce the value of the energy function E
  - But if the increment of x actually actually increases the value of E by ΔE, the new value for x (i.e., $x + \Delta x$) is accepted with probability $p_{\Delta E}$:

$$p_{\Delta E} = \frac{1}{1 + e^{\Delta E / T}}$$

where T is a temperature constant

Simulated annealing (3)

- For big values of T we get: $p_{\Delta E} \approx \frac{1}{2}$

  and the state update is accepted about half the time
- If $T = 0$ only those updates are accepted which reduce the value of E
- **A varying value of T from very large values down to zero corresponds to the heating and cooling phases in the annealing proces**

Simulated annealing (4)

- **It can even be shown that with this simulation strategy the global minimum of the energy function can be (asymptotically) reached**
- **The sigmoid best corresponds to those functions used in thermodynamics** (for the analysis of thermal equilibria)

Boltzmann machine

Definition:

A **Boltzmann machine** is a Hopfield network composed of n neurons with states $x_1, x_2, \ldots, x_n$.

The state of a neuron i is updated asynchronously according to the rule:

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{with probability } 1 - p_i \end{cases}$$

where $$p_i = \frac{1}{1 + e^{-\left(\sum_{j=1}^{n} w_{ij} x_j - \vartheta_i\right)/T}}$$

Boltzmann machine (2)

In the relationship: $p_i = \dfrac{1}{1 + e^{-\left(\sum_{j=1}^{n} w_{ij} x_j - \vartheta_i\right)/T}}$

T denotes a positive temperature constant, w_{ij} are the weights of the network and ϑ_i its thresholds

Energy function for a Boltzmann machine:

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}\, x_i\, x_j + \sum_{i=1}^{n} \vartheta_i\, x_i$$

Boltzmann machine (3)

- The difference between a Boltzmann machine and a Hopfield network consists in the stochastic activation of the neurons
- If T is **very small**, then $p_i \sim 1$, when

$$\sum_{j=1}^{n} w_{ij}\, x_j - \vartheta_i > 0$$

  - × if the net excitation is negative, then $p_i \sim 0$

→ the dynamics of the Boltzmann machine approximates the discrete Hopfield network and the Boltzmann machine finds a local minimum of the energy function

Boltzmann machine (4)

- If $T > 0$ the probability of a transition, or a sequence of transitions, from a network state $x_1, \ldots, x_n$ to another state is never zero

  → The Boltzmann machine cannot settle on a single state

  – possibility both to reduce and to increase the energy of the system

- For **very large values of** T the network visits almost the complete state space

  × During the cooling phase, the network begins to stay longer in the basins of attraction of the local minima

Boltzmann machine (5)

If the temperature is reduced according to the correct schedule, we can expect the system to reach a global minimum *with probability 1*