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 –

Doc. RNDr. Iveta Mrázová, CSc.

Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics

Charles University in Prague

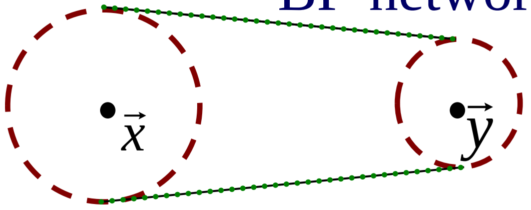
Associative networks and associative memories (1)

The goal of learning: associate known input vectors with given output vectors

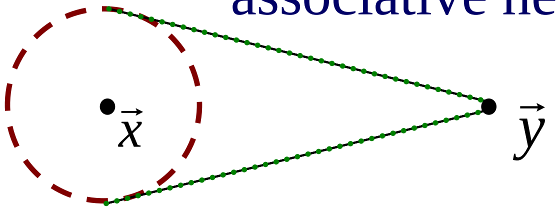
- The neighborhood of a known input pattern $\vec{x}$ should also be mapped to the image $\vec{y}$ of $\vec{x}$

  → „noisy“ input vectors can then be associated with the correct output

BP-network



associative network



Associative networks and associative memories (2)

- Associative memories can be implemented using networks with (or without) feedback

→ the simplest kind of **feedback**:

  - use the output of the network repetitively as a new input until the process converges to a stable state

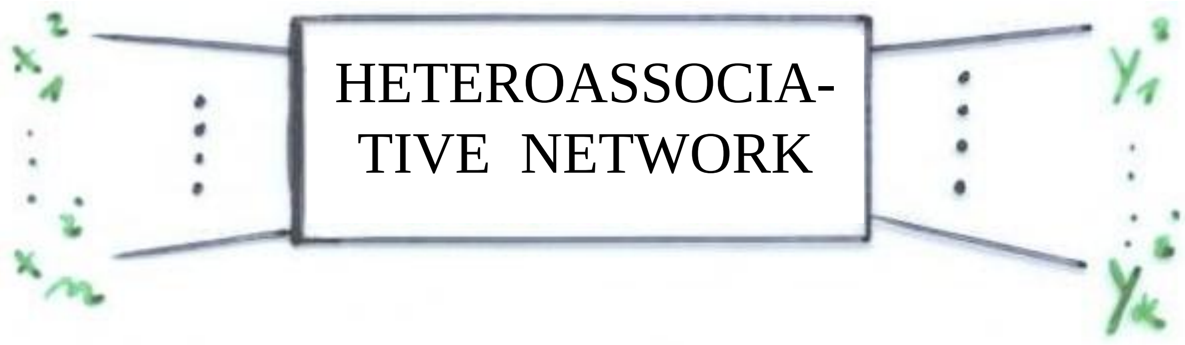
× **but not all networks converge to a stable state after presenting a new input pattern**

→ **additional restrictions on the network architecture are necessary**

Function of an associative memory

- Recognize previously learned input vectors, even if some noise has been added
- The respons of each neuron is determined exclusively by the information flowing through its own weights (Hebbian learning)
- <u>Three types of associative networks:</u>
  - heteroassociative, autoassociative and pattern recognition networks

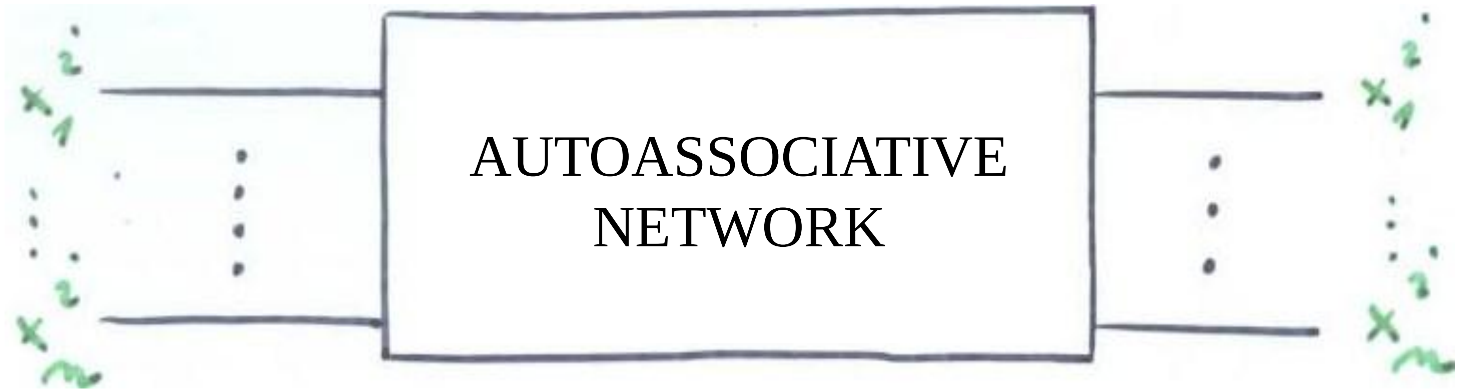
Heteroassociative networks



Map ***m*** input vectors $\vec{x}^1, \dots, \vec{x}^m$ from ***n***-dimensional space to ***m*** output vectors $\vec{y}^1, \dots, \vec{y}^m$ in ***k*** – dimensional space, so that $\vec{x}^i \mapsto \vec{y}^i$

If $\left\| \tilde{\vec{x}} - \vec{x}^i \right\|^2 < \varepsilon$, then $\tilde{\vec{x}} \mapsto \vec{y}^i$ $(\varepsilon > 0)$.

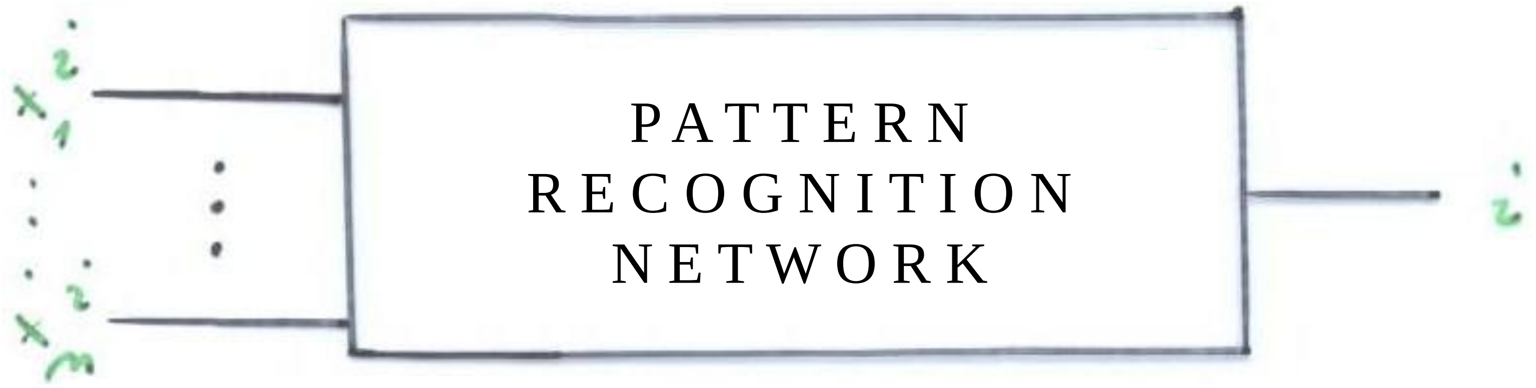
Autoassociative networks



A special subset of the heteroassociative networks (each vector is associated with itself: $\vec{y}^{\,i} = \vec{x}^{\,i}$ for $\boldsymbol{i = 1, \ldots, m}$)

the function of autoassociative networks is to „correct noisy input patterns“

Pattern recognition networks



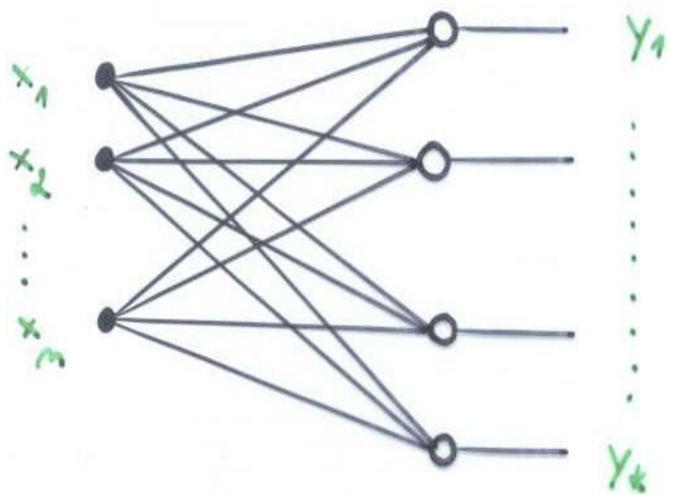
A special type of heteroassociative networks (each vector $\vec{x}^i$ is associated with with the scalar value $\boldsymbol{i}$)

The goal is to to identify the class of the input pattern

Structure of an associative memory

Associative memories can be implemented with a single layer of neurons

Heteroassociative network without feedback



Let: w_{ij} … weight between input i and neuron j

W …. $n \times k$ weight matrix

→ vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ yields excitation vector $\vec{e} = \vec{x} \cdot W$

→ Then the value of the transfer function is computed for each neuron

- For the identity, we get a linear associator and the output $\vec{y}$ is just $\vec{x} \cdot W$

Structure of an associative memory (2)

In general: $\boldsymbol{m}$ different $\boldsymbol{n}$ – dimensional vectors $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$ have to be associated with $\boldsymbol{m}$ $\boldsymbol{k}$ – dimensional vectors $\vec{y}^1, \ldots, \vec{y}^m$

$\rightarrow$ $\boldsymbol{X}$ …. $\boldsymbol{m} \times \boldsymbol{n}$ matrix (rows correspond to the respective input vectors)

$\boldsymbol{Y}$ …. $\boldsymbol{m} \times \boldsymbol{k}$ matrix (rows correspond to the output vectors)

Structure of an associative memory (3)

→ **we are looking for such a weight matrix W, for which $X \cdot W = Y$**

(and in the case of autoassociative memories: $X \cdot W = X$)

Remark: if $m = n$, then X is a square matrix

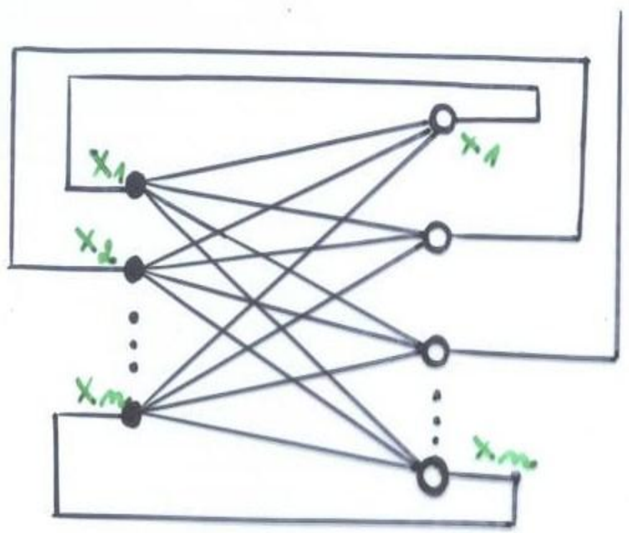
if it is invertible, the solution will be:

$$W = X^{-1} \cdot Y$$

Recurrent associative network

- Network output is used as the new input

Autoassociative network with feedback



- **Assumption:** all neurons compute their outputs simultaneously

  → in each step, the network is fed an input vector $\vec{x}(i)$ and produces a new output $\vec{x}(i+1)$

Recurrent associative network (2)

Question: **is there a fixed point $\vec{\xi}$ such that**

$$\vec{\xi} \cdot \boldsymbol{W} = \vec{\xi}$$

→ the vector $\vec{\xi}$ is an eigenvector of the matrix ***W*** with eigenvalue ***1***

→ the network behaves as a first-order dynamical system, since each new state $\vec{x}(i+1)$ is complet-ely determined by its most recent predecessor

Eigenvector automata

Let: ***W*** … weight matrix of an autoassociative network

the individual neurons are linear associators

→ look for fixed points of the dynamical system

Remark: not all weight matrices lead to a stable state

Example: rotation by 90° in two-dimensional space:

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

→ cycles of length 4

Eigenvector automata (2)

→ Quadratic matrices with a complete set of eigenvectors are more useful as memories

$\boldsymbol{n} \times \boldsymbol{n}$ matrix $\boldsymbol{W}$ has at most $\boldsymbol{n}$ linearly independent eigenvectors and $\boldsymbol{n}$ eigenvalues

→ the eigenvectors $\vec{x}^1, \ldots, \vec{x}^n$ satisfy then:

$$\vec{x}^i \cdot W = \lambda_i \vec{x}^i$$

for $i = 1, \ldots, n$ and the matrix eigenvalue s $\lambda_1, \ldots, \lambda_n$

Eigenvector automata (3)

- Each weight matrix with a full set of eigen-vectors defines an „eigenvector automaton“

  → given an initial input vector, the eigenvector with the largest eigenvalue can be found (if it exists)

- Assume w.l.o.g. that λ_1 is the eigenvalue of W with the largest magnitude:

$$|\lambda_1| > |\lambda_i| \qquad \forall\ i = 2, \ldots, n$$

Eigenvector automata (4)

- Let $\lambda_1 > 0$ and $\vec{a}_0$ is a non-zero randomly chosen ***n***-dimensional vector

→ $\vec{a}_0$ can be expressed as a linear combination of the ***n*** eigenvectors of the matrix ***W*** **:**

$$\vec{a}_0 = \alpha_1 \vec{x}^1 + \alpha_2 \vec{x}^2 + \ldots + \alpha_n \vec{x}^n$$

- **Assumption:** all constants α are non-zero

→ After the first iteration with the matrix ***W*** we get:

$$\vec{a}_1 = \vec{a}_0 \cdot W = \left(\alpha_1 \vec{x}^1 + \ldots + \alpha_n \vec{x}^n\right) \cdot W =$$
$$= \alpha_1 \lambda_1 \vec{x}^1 + \alpha_2 \lambda_2 \vec{x}^2 + \ldots + \alpha_n \lambda_n \vec{x}^n$$

Eigenvector automata (5)

→ After t iterations the result is:

$$\vec{a}_t = \alpha_1 \lambda_1^t \vec{x}^1 + \alpha_2 \lambda_2^t \vec{x}^2 + \ldots + \alpha_n \lambda_n^t \vec{x}^n$$

→ After a big enough number of iterations, the eigenvalue with the largest magnitude will dominate - λ_1

→ the vector $\vec{a}_t$ can thus be brought arbitrarily close to the eigenvector $\vec{x}^1$ (with respect to the direction, not length)

→ in each iteration, the vector $\vec{x}^1$ attracts any other vector $\vec{a}_0$ with a non-zero component for α_1

→ $\vec{x}^1$ is an **attractor**

Eigenvector automata (6)

Example:

Matrix $W = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ has $\boldsymbol{2}$ eigenvectors, $\boldsymbol{(1, 0)}$ and $\boldsymbol{(0, 1)}$ with respective eigenvalues $\boldsymbol{2}$ and $\boldsymbol{1}$

After $\boldsymbol{t}$ iterations, any initial vector $\boldsymbol{(x_1, x_2)\ ;\ x_1 \neq 0}$ will be transformed into the vector $\boldsymbol{(2^t x_1, x_2)}$

→ For large enough $\boldsymbol{t}$ this vector will come arbitrarily close to $\boldsymbol{(1, 0)}$

=> the vector $\boldsymbol{(1, 0)}$ is an attractor

Associative learning

Goal: use associative networks as dynamical systems, whose attractors are exactly those vectors we would like to store in the memory

- **During network design, locate as many attractors in the input space as possible**
  - each one of them should have a well-defined and bounded influence region

  × in the case of the linear eigenvector automaton, just one vector absorbs almost the whole input space

Associative learning (2)

→ nonlinear dynamical systems

- Nonlinear activation of neurons

  Hard-limiting transfer function:

$$\operatorname{sgn}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

- Bipolar coding is better than the binary one

  (bipolar vectors have a greater probability of being mutually orthogonal than binary vectors)

Hebbian learning

Assumption:

- single-layer network of ***k*** neurons with the ***sgn*** transfer function

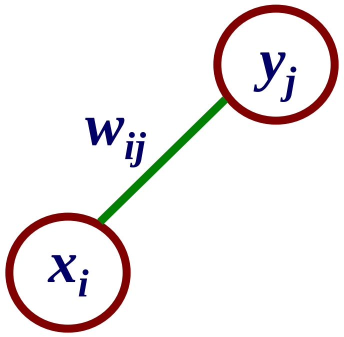
Goal:

- Find the appropriate weights to map the ***n*** – dimensional input vector $\vec{x}$ to the ***k*** – dimensional output vector $\vec{y}$

Idea: (Donald Hebb – 1949)

- Two neurons, which are simultaneously active, should develop a degree of interaction higher than those neurons, whose activities are uncorrelated. In the latter case, the interaction between the elements should be very low or zero.

Hebbian learning (2)



$$\Delta w_{ij} = \gamma\, x_i\, y_j$$

γ … learning parameter

W … weight matrix (initialized to zeroes)

- Adjustment rule will be applied to all weights
- at the input is the n – dimensional vector $\vec{x}^1$, at the output is the k – dimensional vector $\vec{y}^1$

→ **the updated weight matrix W is the correlation matrix for these two vectors**

$$W = [\, w_{ij} \,]_{n \times k} = [\, x_i^1\, y_j^1 \,]_{n \times k}$$

Hebbian learning (3)

- matrix W maps the non-zero vector $\vec{x}^1$ exactly to the vector $\vec{y}^1$

$$\vec{x}^1 \cdot W = \left(y_1^1 \sum_{i=1}^{n} x_i^1 x_i^1 , y_2^1 \sum_{i=1}^{n} x_i^1 x_i^1 , \dots , y_k^1 \sum_{i=1}^{n} x_i^1 x_i^1 \right) =$$

$$= \vec{y}^1 \left(\vec{x}^1 \cdot \vec{x}^1 \right)$$

- for $\vec{x}^1 \neq 0$ it holds that $\vec{x}^1 \cdot \vec{x}^1 > 0$ and the output of the network is:

$$\operatorname{sgn}\left(\vec{x}^1 \cdot W\right) = \left(y_1^1, \dots, y_k^1\right) = \vec{y}^1$$

- for $-\vec{x}^1$, the output of the network is:

$$\operatorname{sgn}\left(-\vec{x}^1 \cdot W\right) = -\operatorname{sgn}\left(\vec{x}^1 \cdot W\right) = -\vec{y}^1$$

Hebbian learning (4)

In general:

- If we want to associate ***m*** ***n*** – dimensional non-zero vectors $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$ with ***m*** ***k*** – dimensional vectors $\vec{y}^1, \ldots, \vec{y}^m$, we apply Hebbian learning to each pair INPUT/OUTPUT
- The resulting weight matrix ***W*** will have the form:

$$\boldsymbol{W} = \boldsymbol{W}^1 + \boldsymbol{W}^2 + \ldots + \boldsymbol{W}^m \quad ,$$

where each matrix $\boldsymbol{W}^l$ is the $\boldsymbol{n} \times \boldsymbol{k}$ correlation matrix of the vectors $\vec{x}^l$ and $\vec{y}^l$: $\boldsymbol{W}^l = [\, \mathbf{x}_i^l \, \mathbf{y}_j^l \,]_{n \times k}$

Hebbian learning (5)

- If the input to the network is then the vector $\vec{x}^p$, the excitation vector of the network will be equal to:

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

$$= \vec{y}^p \cdot \left(\vec{x}^p \cdot \vec{x}^p\right) + \sum_{l \neq p}^{m} \vec{y}^l \cdot \left(\vec{x}^l \cdot \vec{x}^p\right)$$

- The excitation vector thus coresponds to $\vec{y}^p$ (multiplied by a positive constant) plus a perturbation term

$$\sum_{l \neq p}^{m} \vec{y}^l \cdot \left(\vec{x}^l \cdot \vec{x}^p\right) \quad ,$$

that is called the **CROSSTALK**

Hebbian learning (6)

- The network produces the desired vector $\vec{y}^p$ as output when the crosstalk is zero

  $\rightarrow$ whenever the input patterns $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$ are pairwise orthogonal

- The network cann yield appropriate results even for non-zero crosstalks

  × crosstalk should be smaller than $\vec{y}^p \cdot \left(\vec{x}^p \cdot \vec{x}^p\right)$

$\rightarrow$ **<u>The output of the network is equal to:</u>**

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

Hebbian learning (7)

- Since $\vec{x}^p \cdot \vec{x}^p$ is a positive constant, it holds that:

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

- To produce the output $\vec{y}^p$, it must hold:

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

- This condition is satisfied, when the absolute value of all components of the perturbation term $\sum_{l \neq p}^{m} \vec{y}^l \cdot \frac{\left(\vec{x}^l \cdot \vec{x}^p\right)}{\left(\vec{x}^p \cdot \vec{x}^p\right)}$ is smaller than 1

Hebbian learning (8)

- → This means that the scalar product $\vec{x}^{l} \cdot \vec{x}^{p}$ must be smaller than the quadratic length of the vector $\vec{x}^{p}$ (equal to ***n*** for ***n***-dimensional bipolar vectors)
- → If randomly selected bipolar vectors are associated with other also randomly selected bipolar vectors, the probability is high that they will be nearly pairwise orthogonal (as long as not many of them are selected)
  - → In such a case, the crosstalk will be small and Hebbian learning will lead to an efficient set of weights for the associative network

Geometric interpretation of Hebbian learning

- For the matrices $\boldsymbol{W^i}$ from $\boldsymbol{W = W^1 + W^2 + \ldots + W^m}$ it holds in autoassociative networks: $W^i = \left(\vec{x}^i\right)^T \vec{x}^i$

$\rightarrow$ thus for $W^1 = \left(\vec{x}^1\right)^T \vec{x}^1$, the input vector $\vec{z}$ will be projected into the linear subspace $\boldsymbol{L_1}$ spanned by the vector $\vec{x}^1$, since

$$\vec{z} \cdot W^1 = \vec{z}\left(\vec{x}^1\right)^T \vec{x}^1 = \left(\vec{z}\left(\vec{x}^1\right)^T\right)\vec{x}^1 = c_1\, \vec{x}^1$$

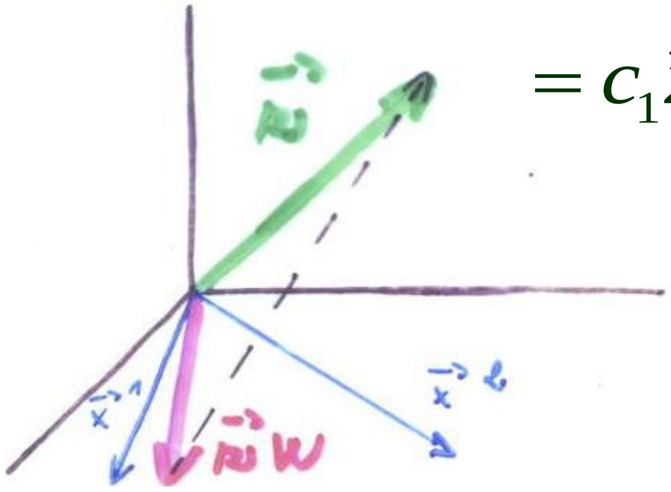
In general a non-orthogonal projection of the vector $\vec{z}$ into $\boldsymbol{L_1}$ ($\boldsymbol{c_1}$ represents a constant)

$\rightarrow$ similarly for other weight matrices $\boldsymbol{W^2, \ldots, W^m}$

Geometric interpretation of Hebbian learning (2)

- The matrix $W = \sum_{i=0}^{m} W^i$ projects a vector $\vec{z}$ into the linear subspace spanned by the vectors $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$, protože $\vec{z} \cdot W = \vec{z} \cdot W^1 + \vec{z} \cdot W^2 + \ldots + \vec{z} \cdot W^m =$

$$= c_1 \vec{x}^1 + c_2 \vec{x}^2 + \ldots + c_m \vec{x}^m$$



(in general a non-orthogonal projection)

Associative networks: behavior analysis

- Identification of attractors (fixed points of the system)
- The size of the basins of attraction
  - **Hamming distance**

    ~ number of different components in 2 bipolar vectors
  - Example:

    $$\begin{array}{cccc} 1 & \underline{-1} & \underline{1} & 1 \\ 1 & \underline{1} & \underline{-1} & 1 \end{array} \rightarrow \mathbf{2}$$
  - With the growimg number of stored patterns, the basins of attraction become smaler $\rightarrow$ **spurious stable states**
    - Big crosstalk
    - Patterns inverse to the stored ones:

$$\operatorname{sgn}\left(-\vec{x}\cdot W\right) = -\operatorname{sgn}\left(\vec{x}\cdot W\right) = -\vec{x}$$

Associative networks: behavior analysis (2)

- Recurrent networks (use feedback)
  - Improved convergence when compared to associative memories without feedback
  - Wider basins of attraction
    - × not too many patterns can be stored

    → PROBLEM: **Capacity of the weight matrix**
  - The sizes off the basins of attraction can be compared using an index

$$I = \sum_{h=0}^{n/2} h \, p_h$$

p_h … percentage of vectors with Hamming distance h from a stored pattern, which converged to it

The capacity problem

- The basins of attraction of stored patterns deteriorate with every new pattern to be stored in the memory
- If the crosstalk term becomes too large, previously stored patterns can be even „forgotten“
  - the probability that this could happen should be kept low

The capacity problem (2)

Assess the number of patterns ***m***, that can be stored safely in an autoassociative memory with a weight matrix ***W*** ***(n*** × ***n)***

Maximum capacity of the network: ***m ~ 0.18 n***

- The number of stored patterns should be smaller than *0.18 n* (***n*** is the dimension of the patterns)
- If the patterns are correlated, even ***m < 0.18 n*** can produce problems

Derivation of the network capacity – idea (1)

- For $W^i = \frac{1}{n}\left(\vec{x}^{\,i}\right)^T \vec{x}^{\,i}$
- Crosstalk for $\boldsymbol{n}$ – dimensional bipolar vectors and $\boldsymbol{m}$ patterns in the case of an autoassociative network:

$$\frac{1}{n}\sum_{l\neq p}^{m} \vec{x}^{\,l}\left(\vec{x}^{\,l}\cdot\vec{x}^{\,p}\right)$$

If the magnitude of this term is larger than $\boldsymbol{1}$ and the term has a sign opposite to the stored pattern, the considered pattern component can be flipped

Derivation of the network capacity – idea (2)

- Assume that the stored vectors are chosen randomly:
  - In this case the crosstalk term for bit ***i*** of the input vector is given by

$$\frac{1}{n} \sum_{l \neq p}^{m} x_i^l \left(\vec{x}^{\,l} \cdot \vec{x}^{\,p} \right) \qquad (*)$$

  - Since the components of each pattern have been selected randomly, we can think of $\boldsymbol{m \cdot n}$ random bit selections
  - The expected value of this sum (*) is ***0***

Derivation of the network capacity – idea (3)

- The sum ***(*)*** has a binomial distribution and for large $\boldsymbol{m \cdot n}$ we can approximate it by a normal distribution with the standard deviation $\sigma = \sqrt{m/n}$
- Probability of error $\boldsymbol{P}$, that the sum ***(*)*** becomes larger than ***1*** (or smaller than ***-1***), is given by

$$P = \frac{1}{\sqrt{2\pi}\,\sigma} \int_{1}^{\infty} e^{-x^2/(2\sigma^2)}\,dx$$

Derivation of the network capacity – idea (4)

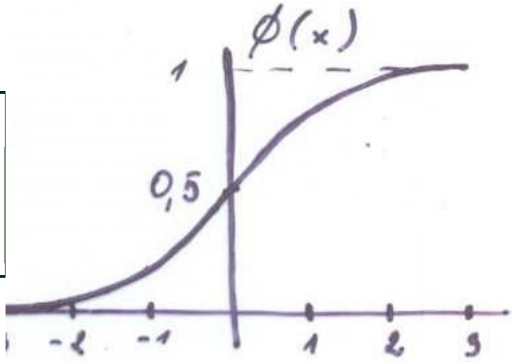
- That is: $P\left\{ \left| (*) \right| > 1 \right\} = 2\left[1 - \Phi\left(\frac{1}{\sqrt{m/n}} \right) \right]$

  where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$

$\rightarrow$ for the upper bound for one bit failure set to ***0.01***

we obtain:

$$0.01 = 2\left[1 - \Phi\left(\frac{1}{\sqrt{m/n}} \right) \right]$$



$\rightarrow$ $\boldsymbol{m \sim 0.18\, n}$

Associative memories: the pseudoinverse matrix (1)

Hebbian learning produces good results when the stored patterns are nearly orthogonal

- ~ when ***m*** bipolar vectors are selected randomly from an ***n*** – dimensional space, ***n*** is „large enough“ and ***m*** is „much smaller“ than ***n***

× in real applications the patterns are almost always correlated and the crosstalk in the expression

$$\vec{x}^p \cdot W = \vec{y}^p \cdot \left(\vec{x}^p \cdot \vec{x}^p\right) + \sum_{l \neq p}^{m} \vec{y}^l \cdot \left(\vec{x}^l \cdot \vec{x}^p\right)$$

affects the recall process because the scalar products $\vec{x}^l \cdot \vec{x}^p$ are not small enough for $\boldsymbol{l \neq p}$

Associative memories: the pseudoinverse matrix (2)

→ mutual correlation of stored patterns causes reduction in the capacity of the associative network

~ the number of patterns, that can be stored and recalled

the stored patterns do not occupy the input space homogeneously, but concentrate around a small region

→ look for alternative learning methods capable of mininizing the crosstalk between stored patterns

→ use pseudoinverse instead of correlation matrix

Associative memories: the pseudoinverse matrix (3)

Definition:

The pseudoinverse of a real $\boldsymbol{m \times n}$ matrix is the real matrix $\tilde{X}$ with the following properties:

1. $X \tilde{X} X = X$,
2. $\tilde{X} X \tilde{X} = \tilde{X}$,
3. $\tilde{X} X$ and $X \tilde{X}$ are symmetrical

The pseudoinverse always exists and is unique.

Pseudoinverse matrix - properties

Let $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$ be $\boldsymbol{n}$ – dimensional vectors and associate them with the $\boldsymbol{m}$ $\boldsymbol{k}$ – dimensional vectors $\vec{y}^1, \ldots, \vec{y}^m$

→ matrix notation:

$\boldsymbol{X}$ …. Matrix $\boldsymbol{m} \times \boldsymbol{n}$

the rows are the vectors $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$,

$\boldsymbol{Y}$ …. Matrix $\boldsymbol{m} \times \boldsymbol{k}$

the rows are the vectors $\vec{y}^1, \ldots, \vec{y}^m$

→ **Look for a weight matrix $\boldsymbol{W}$; $\boldsymbol{XW} = \boldsymbol{Y}$**

Pseudoinverse matrix – properties (2)

Since in general $\boldsymbol{m} \neq \boldsymbol{n}$ and vectors $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^m$ are not necessarily linearly independent, the matrix $\boldsymbol{X}$ does not have to be invertible

$\rightarrow$ look for a matrix $\boldsymbol{W}$, which minimizes $\|XW - Y\|^2$

(~ the sum of the squares of all its elements)

minimization by means of $W = \tilde{X}Y$

$\tilde{X}$…. pseudoinverse of the matrix $\boldsymbol{X}$

(~ the best approximation to an inverse of $\boldsymbol{X}$

if $\boldsymbol{X}^{-1}$ exists, then $\boldsymbol{X}^{-1} = \tilde{X}$)

Pseudoinverse matrix – properties (3)

Proposition:

Let $\boldsymbol{X}$ be an $\boldsymbol{m} \times \boldsymbol{n}$ real matrix and $\boldsymbol{Y}$ be an $\boldsymbol{m} \times \boldsymbol{k}$ real matrix. The $\boldsymbol{n} \times \boldsymbol{k}$ matrix $W = \tilde{X}Y$ minimizes the quadratic norm $\|XW - Y\|^2$.

(At the same time, $\tilde{X}$ minimizes $\|X\tilde{X} - I\|^2$.)

Proof:

Define $\boldsymbol{E} = \|XW - Y\|^2$

$\rightarrow$ $\boldsymbol{E}$ can be computed as $\boldsymbol{E} = \underset{\text{trace of a matrix}}{\boldsymbol{tr}}\,(\boldsymbol{S})$, where

$$\boldsymbol{S} = (XW - Y)^T (XW - Y)$$

($\boldsymbol{E}$ ~ sum of all the diagonal elements of $\boldsymbol{S}$)

Pseudoinverse matrix – properties (4)

Proof (continue):

→ $\boldsymbol{S}$ can be rewritten as

$$S = \left(\tilde{X}Y - W\right)^T X^T X \left(\tilde{X}Y - W\right) + Y^T \left(I - X\tilde{X}\right)Y$$

(Because:

$$S = \left(\tilde{X}Y - W\right)^T \left(X^T X\tilde{X}Y - X^T XW\right) + Y^T \left(I - X\tilde{X}\right)Y$$

Since the matrix $X\tilde{X}$ is symmetrical (def.), and therefore:

$$S = \left(\tilde{X}Y - W\right)^T \left(\left(\underbrace{X \ \tilde{X} \ X}_{= X \text{ (def.)}} \right)^T Y - X^T XW \right) + Y^T \left(I - X\tilde{X}\right)Y$$

Pseudoinverse matrix – properties (5)

<u>Proof (continue):</u>

(it follows: $S = \left(\tilde{X}Y - W\right)^T \left(X^T Y - X^T XW\right) + Y^T\left(I - X\tilde{X}\right)Y =$

$= \left(\tilde{X}Y - W\right)^T X^T \left(Y - XW\right) + Y^T\left(I - X\tilde{X}\right)Y =$

$= \left(X\tilde{X}Y - XW\right)^T \left(Y - XW\right) + Y^T\left(I - X\tilde{X}\right)Y =$

$= \left(-XW\right)^T\left(Y - XW\right) + Y^T X\tilde{X}\left(Y - XW\right) + Y^T\left(I - X\tilde{X}\right)Y =$

$= \left(-XW\right)^T\left(Y - XW\right) + Y^T\left(-XW\right) + Y^T Y =$

$= \left(Y - XW\right)^T\left(Y - XW\right)$)

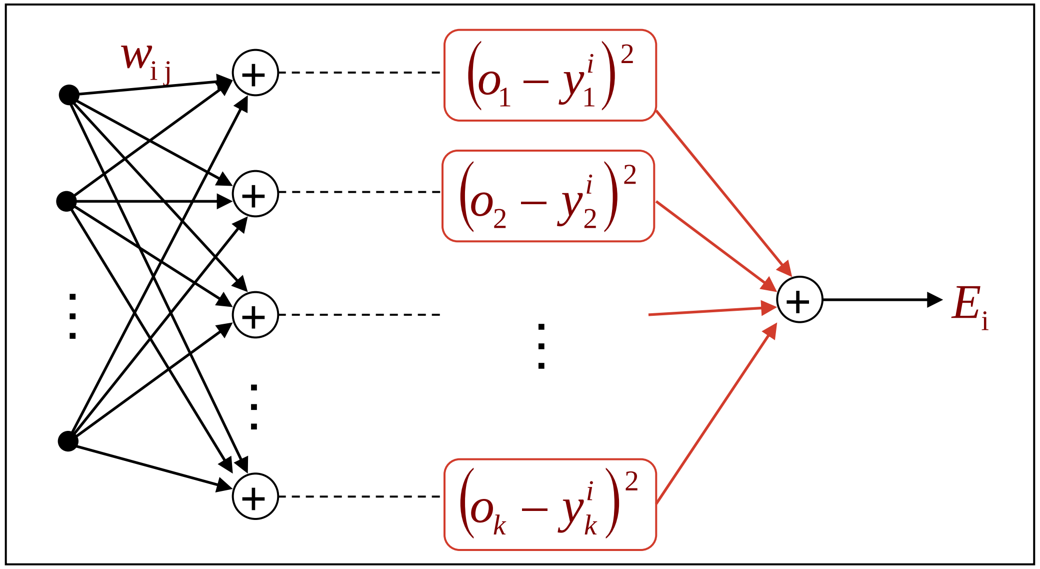
$\rightarrow$ E can be rewritten as

$$E = tr\left(\left(\tilde{X}Y - W\right)^T X^T X\left(\tilde{X}Y - W\right)\right) + \underbrace{tr\left(Y^T\left(I - X\tilde{X}\right)Y\right)}_{= const.}$$

$\rightarrow$ ***min E*** for $\boldsymbol{W = \tilde{X}Y}$ ***QED***

Computation of the pseudoinverse

- Compute an approximation of the pseudoinverse using a backpropagation network
- Network used to find the weights for associative memories



o – network output

y – desired association

Computation of the pseudoinverse (2)

- The goal of training: Find such a weight matrix **W** with elements $\mathbf{w}_{ij}$, that produces the best mapping from the vectors $\vec{\mathbf{x}}^{1},\ldots,\vec{\mathbf{x}}^{m}$ to the vectors $\vec{\mathbf{y}}^{1},\ldots,\vec{\mathbf{y}}^{m}$
- For the *i*-th input vector, network output will be compa-red with the vector $\vec{\mathbf{y}}^{i}$ and E_i will be computed
- The total quadratic error $E=\sum_{i=1}^{m} E_i$ then corresponds to: $\|\mathbf{XW}-\mathbf{Y}\|^2$
- The backpropagation algorithm then finds the matrix **W**, that is expected to minimize $\|\mathbf{XW}-\mathbf{Y}\|^2$