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

– Kohonen Maps and Hybrid Models –

Doc. RNDr. Iveta Mrázová, CSc.

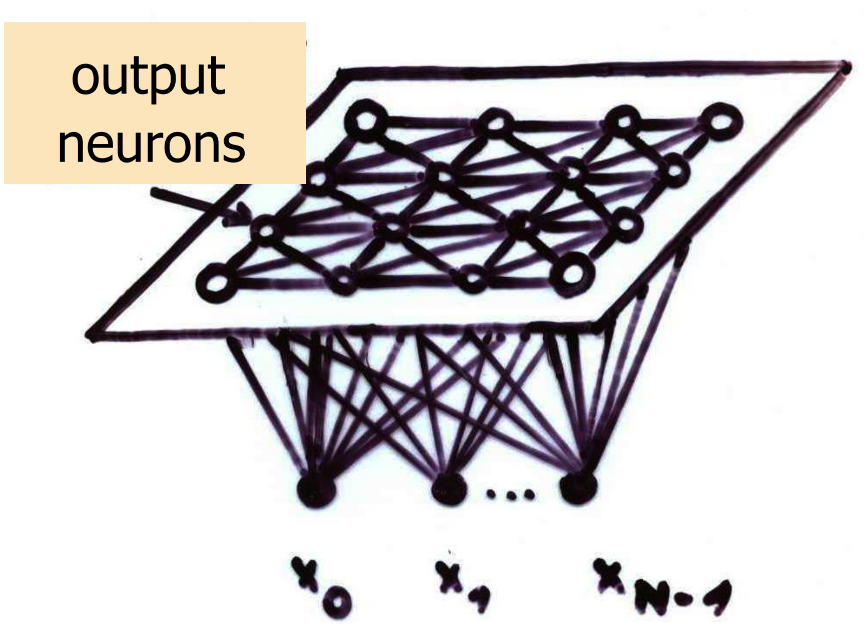
Department of Theoretical Computer Science and Mathematical Logic

Faculty of Mathematics and Physics

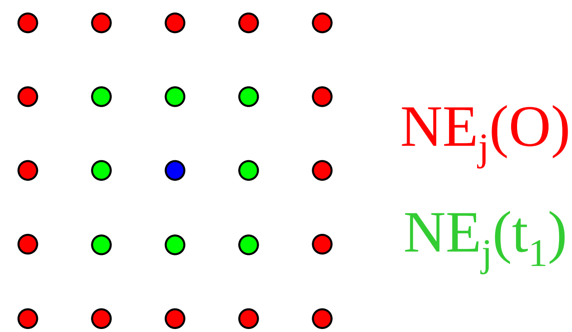
Charles University in Prague

Kohonen maps

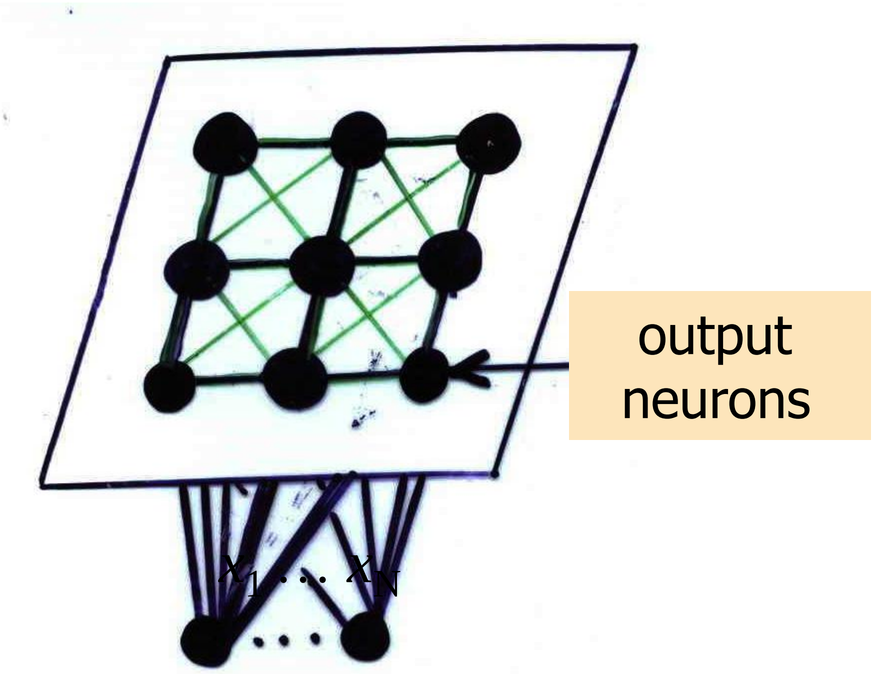
- Teuvo Kohonen – phonetic typewriter



topological neighborhood



Kohonen maps (2)



- **Training**
  - **unsupervised**
- **Recall**
- **Applications:**
  - Phonetic typewriter
  - Economics

Kohonen model – training algorithm

Motivation:

- The grid of (topologically ordered) neurons allows us to identify the immediate neighbors of a given neuron

  → during training, the weights of the respective neurons and their neighbors will be updated

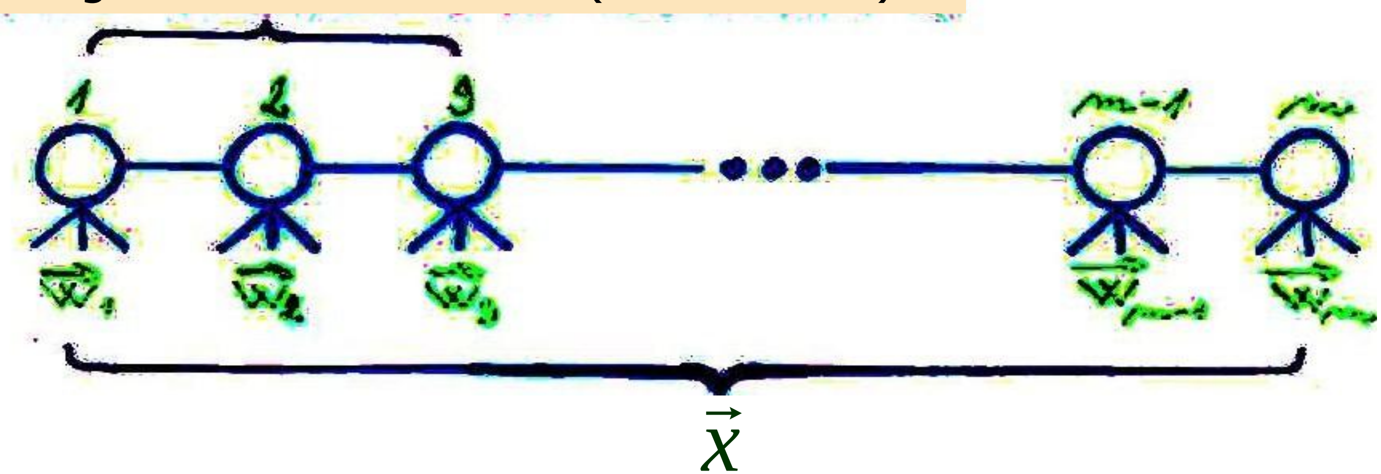
Objective: neighboring neurons should react to closely related signals

Kohonen model – training algorithm (2)

<u>Problem (1-dim):</u>

- Divide the ***n*** – dimensional space by means of a one-dimensional chain of „Kohonen neurons“
- neurons are arranged in sequence and numbered from ***1*** to ***n***

neighborhood of neuron 2 (with radius 1)



Kohonen model – training algorithm (3)

Problem (1-dim – continued**):**

- One-dimensional grid of neurons:
  - Each neuron receives an ***n***– dimensional input $\vec{x}$ based on an ***n***– dimensional weight vector $\vec{w} = (w_1, \ldots, w_n)$, it computes its excitation

Objective: „specialization“ of each neuron to a different region of the input space (this „specialization“ is characterized by maximum excitation of the respective neurons for patterns from the given region)

Kohonen model – training algorithm (4)

Problem (1-dimensional – continued**):**

→ „Kohonen" neurons compute the Euclidean distance between the input $\vec{x}$ and the correspon-ding weight vector $\vec{w}$

→ „the closest" neuron will be characterized by maximum excitation

Kohonen model – training algorithm (5)

Neighborhood definition:

- In a one-dimensional Kohonen map, the neighborhood of neuron ***k*** with radius ***1*** contains neuron ***k – 1*** and ***k + 1***
- Neurons on both ends of a one-dimensional Kohonen map have an asymmetric neighborhood
- In a ***1*** – dimensional Kohonen map, the neighborhood of neuron ***k*** with radius ***r*** contains all the neurons located up to ***r*** positions from ***k*** to the left or to the right
- Similarly for multidimensional Kohonen maps and the chosen grid metrics (rectangular, hexagonal, …)

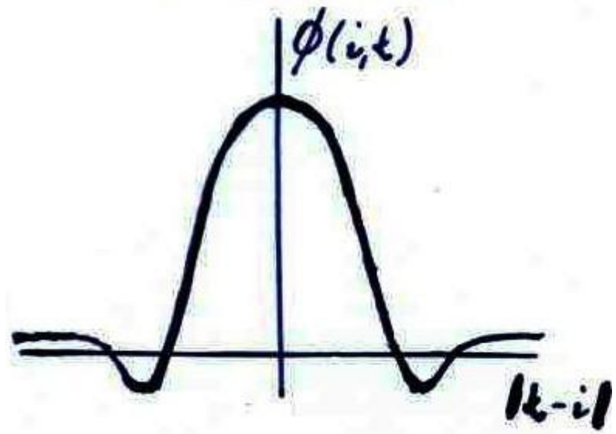
Kohonen model – training algorithm (6)

Lateral interaction function *Φ(i,k)*:

~ „the strength of the lateral interconnection“ between neuron ***i*** and ***k*** during training

Example:

- ***Φ(i,k)=1*** $\forall$ ***i*** from the neighbor-hood of ***k*** with radius ***r*** and ***Φ(i,k)=0*** $\forall$ remaining ***i***
- „Mexican hat“ function
- … and others …



Kohonen self-organizing feature maps: the training algorithm

Step 1: Initialize the weights between ***N*** input and ***M*** output neurons to small random values. Set the initial radius of the neighborhood and the lateral interaction function ***Φ***.

Step 2: Present a new training pattern to the network.

Step 3: Compute the distance d_j between the input pattern and the weight vectors of all the output neurons ***j*** by means:

$$d_j = \sum_{i=0}^{N-1} \left(x_i(t) - w_{ij}(t) \right)^2$$

where $x_i(t)$ denotes the input of the neuron ***i*** in time ***t*** and $w_{ij}(t)$ corresponds to the synaptic weight between the input neuron ***i*** and the output neuron neuron ***j*** in time ***t***. This distance can contain weight coefficients.

Kohonen self-organizing feature maps: the training algorithm (2)

Step 4: Select (e.g., by means of lateral inhibition) the output neuron c with the minimum distance d_j from the presented input pattern and denote it to be „the winner".

Step 5: Adjust the weights of the winning neuron c and all the neurons from its neighborhood N_c. The new weights are:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)\, \Phi(c,j)\, (\, x_i(t) - w_{ij}(t)\,)$$

For $j \in N_c$; $0 \le i \le N-1$

$\alpha(t)$ is the vigilance coefficient ($0 < \alpha(t) < 1$) decreasing with time.

Kohonen self-organizing feature maps: the training algorithm (3)

For the choice of $\alpha(t)$ it should hold:

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \quad \wedge \quad \sum_{t=1}^{\infty} \alpha^2(t) < \infty$$

During training, the winning neuron adjusts its weight vector towards current input patterns. The same holds also for neurons from the neighbourhood of the winner. The value of the function $\Phi(c,j)$ decreases with growing distance of the neurons from the center of the neighbourhood N_c.

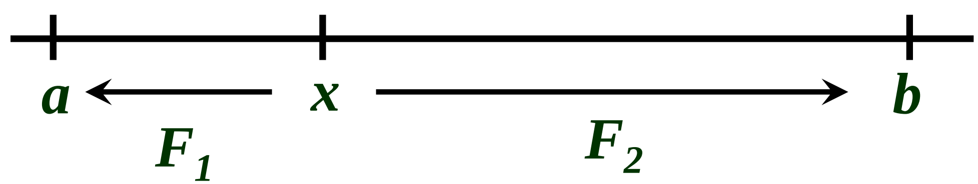
Step 6: Repeat by Going to Step 2.

Analysis of convergence ~ stability of the solution and an ordered state

Stability when suppposed that the network has already arrived at an ordered state:

1) <u>One-dimensional case:</u>

a) interval ***[a,b]*** , ***1*** neuron with the weight ***x***, no neighbourhood considered:



→ **convergence of *x* towards the center of *[a,b]***

Analysis of convergence ~ stability of the solution and an ordered state (2)

- The update rule: $x_n = x_{n-1} + \eta\,(\,\xi - x_{n-1}\,)$

  x_n, x_{n-1} …….... weight values in time n and $n-1$

  ξ ……….…... a random number from the interval $[\,a\,,\,b\,]$

- If $0 < \eta \leq 1$, the series $x_1, x_2, \ldots$ cannot leave $[\,a\,,\,b\,]$
- Bounded is also the expected value $\langle x \rangle$ of the weight x
- The expected value of the derivative of x with respect to t is zero: $\left\langle \frac{dx}{dt} \right\rangle = 0$, otherwise $\langle x \rangle$ would be $\langle x \rangle < a$ or $\langle x \rangle > b$
- Since: $$\left\langle \frac{dx}{dt} \right\rangle = \eta\,\left(\langle \xi \rangle - \langle x \rangle\right) = \eta\left(\frac{a+b}{2} - \langle x \rangle\right)$$

  it follows that: $\langle x \rangle = (\,a + b\,)\,/\,2$

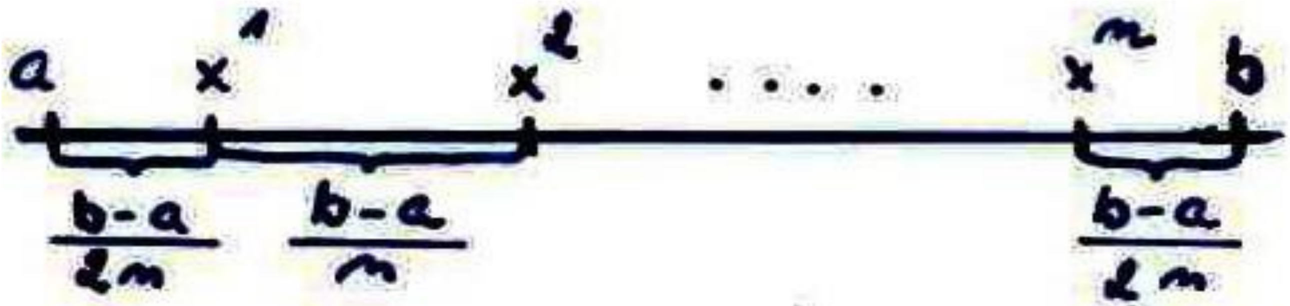
Analysis of convergence ~ stability of the solution and an ordered state (3)

b) interval ***[a,b]***, ***n*** neurons with weights $x^1, x^2, \ldots, x^n$

- no neighborhood considered,
- the weights are assumed to be monotonically ordered: $a < x^1 < x^2 < \ldots < x^n < b$

→ **the weights converge to**

$$\langle x^i \rangle = a + (2i-1)\frac{b-a}{2n}$$

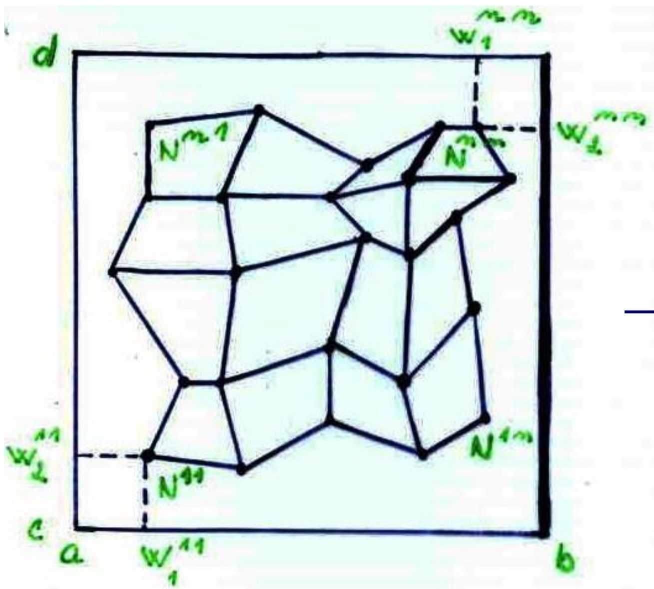


Analysis of convergence ~ stability of the solution and an ordered state (4)

2) Two-dimensional case:

- interval $[\,a, b\,] \times [\,c, d\,]$, $n \times n$ neurons
- no neighborhood considered, monotonically ordered weights:

$$w_1^{ij} < w_1^{ik} \qquad \text{for} \quad j < k$$
$$w_2^{ij} < w_2^{kj} \qquad \text{for} \quad j < k$$



→ **The problem will be reduced to 2 1-dimensional problems**

Analysis of convergence ~ stability of the solution and an ordered state (5)

2) **Two-dimensional case** (continued):

- Let $w_j^1 = \frac{1}{n}\sum_{i=1}^{n} w_1^{ij}$ denote the average weight value of the neurons from the $\boldsymbol{j}$ – th column
- Since $w_1^{ij} < w_1^{ik}$ for $j < k$, these average values w_1^j will be monotonically arranged: $a < w_1^1 < w_1^2 < \ldots < b$
- In the first column, the average weight value will oscillate around the expected value $\langle w_1^1 \rangle$
- Similarly for the average weight values in each row

→ **convergence to a stable state**

(for small enough learning rates)

Analysis of convergence ~ stability of the solution and an ordered state (6)

PROBLEMS:

- „rozvinutí“ planární mřížky a podmínky, za kterých k němu dojde
- „metastabilní stavy“ a nevhodná volba funkce laterální interakce (příliš rychlý pokles)

→ **convergence of 1-dimensional Kohonen networks to an ordered state if the input is selected from a uniform distribution and the following update rule is used**

$$w_k^{new} = w_k^{old} + \gamma \left(\xi - w_k^{old} \right)$$

where $\boldsymbol{k}$ denotes the winning neuron and its two neighbors

(Cottrell & Fort, 1986)

Variants of the training algorithm for Kohonen maps

Supervised training:

(LVQ ~ Learning Vector Quantization)

LVQ1:

- Motivation:

  -) $\vec{x}$ should belong to the same class like the closest $\vec{w}_i$
- let $c = \arg\min_i \{\|\vec{x} - \vec{w}_i\|\}$ denotes the $\vec{w}_i$ that is the closest one to $\vec{x}$ ($\boldsymbol{c}$ ~ the winning neuron)

Variants of the training algorithm for Kohonen maps (2)

LVQ1 (continued):

→ **adjustment rules (*0* < *α(t)* < *1*):**

$$\vec{w}_c(t+1) = \vec{w}_c(t) + \alpha(t)\left[\vec{x}(t) - \vec{w}_c(t)\right]$$

if $\vec{x}$ and $\vec{w}_c$ are classified identically

$$\vec{w}_c(t+1) = \vec{w}_c(t) - \alpha(t)\left[\vec{x}(t) - \vec{w}_c(t)\right]$$

if $\vec{x}$ and $\vec{w}_c$ are classified differently

$$\vec{w}_i(t+1) = \vec{w}_i(t) \quad \text{if } i \neq c$$

Variants of the training algorithm for Kohonen maps (3)

LVQ2.1:

- Motivation: mutual update of 2 nearest neighbors of $\vec{x}$
  - One must belong to the correct class, the other to the incorrect
  - Furthermore, $\vec{x}$ must be from the area close to the separating hyperplane between $\vec{w}_i$ and $\vec{w}_j$ (~ from „the window“)
  - If d_i (resp. d_j) denotes the Euclidean distance between $\vec{x}$ and $\vec{w}_i$ (resp. between $\vec{x}$ and $\vec{w}_j$), „the window“ can be defined by means of:

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s\,, \quad \text{where} \quad s = \frac{1-w}{1+w}$$

(recommended values for w (~ the width of the window): ***0.2 – 0.3***)

Variants of the training algorithm for Kohonen maps (4)

LVQ2.1 (continued):

→ **adjustment rules ($0 < \alpha(t) < 1$):**

- $\vec{w}_i(t+1) = \vec{w}_i(t) - \alpha(t)\left[\vec{x}(t) - \vec{w}_i(t)\right]$
- $\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha(t)\left[\vec{x}(t) - \vec{w}_j(t)\right]$

- $\vec{w}_i$ and $\vec{w}_j$ are the closest to $\vec{x}$
- at the same time, $\vec{x}$ and $\vec{w}_j$ belong to the same class
- and $\vec{x}$ and $\vec{w}_i$ belong to different classes
- $\vec{x}$ comes from the „window“

Variants of the training algorithm for Kohonen maps (5)

LVQ3 (motivation):

- aproximation of class distribution and stabilization of the solution

→ **adjustment rules (*0 < α(t) < 1*):**

- $$\vec{w}_i(t+1) = \vec{w}_i(t) - \alpha(t)\left[\vec{x}(t) - \vec{w}_i(t)\right]$$
- $$\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha(t)\left[\vec{x}(t) - \vec{w}_j(t)\right]$$

$\vec{w}_i$ and $\vec{w}_j$ are the closest to $\vec{x}$; $\vec{x}$ and $\vec{w}_j$ belong to the same class, while $\vec{x}$ and $\vec{w}_i$ belong to different classes and $\vec{x}$ is from „the window“

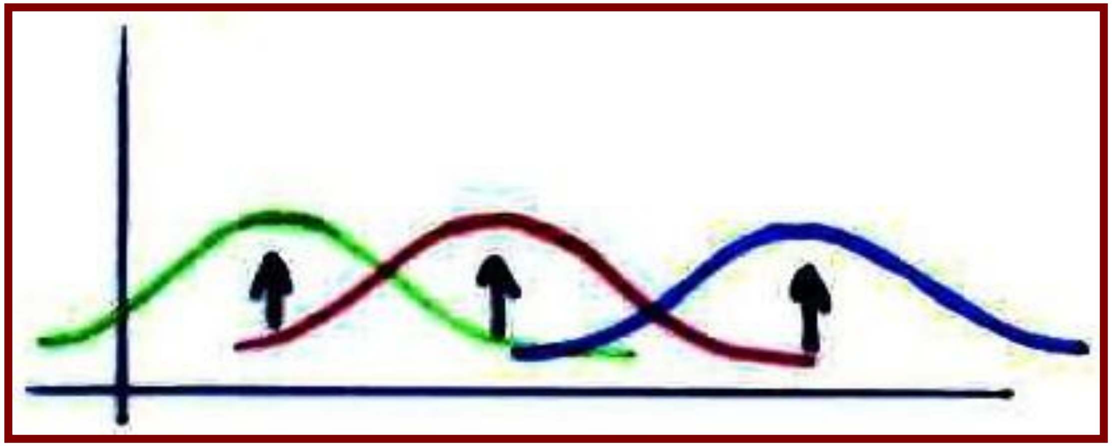
- $$\vec{w}_k(t+1) = \vec{w}_k(t) + \varepsilon\alpha(t)\left[\vec{x}(t) - \vec{w}_k(t)\right]$$

for $\boldsymbol{k \in \{i, j\}}$ if $\vec{x}$, $\vec{w}_i$ and $\vec{w}_j$ belong to the same class

Variants of the training algorithm for Kohonen maps (6)

LVQ3 (continued):

- choice of parameters:
  - $0.1 \leq \varepsilon \leq 0.5$
  - $0.2 \leq w \leq 0.3$

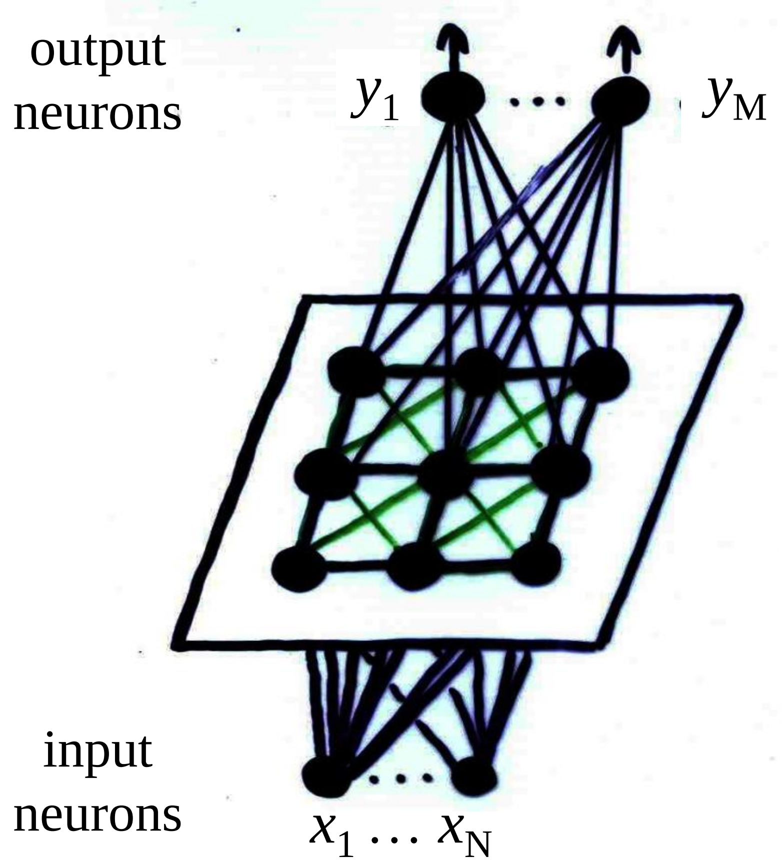


Variants of the training algorithm for Kohonen maps (7)

Further variants:

- Multi-layered Kohonen maps
  - Abstraction tree
- Counterpropagation networks
  - Supervised training – two phases
    - Kohonen (clustering) layer
    - Grossberg layer (weight adjustment only for the winning neurons from the Kohonen layer

Counterpropagation networks



Training: supervised

Recall

Application:

- Heteroassociative memory

The training algorithm for counterpropagation networks (1)

Step 1: Initialize synaptic weights to small random values.

Step 2: Present a new training pattern to the network as:

(input, desired output).

Step 3: In the Kohonen layer, find the neuron *c*, the synaptic weights of which best correspond to the presented pattern $\vec{x}(t)$. For this neuron, it thus follows that the distance e_k between the selected weight vector $\vec{v}_k(t)$ and the presented pattern $\vec{x}(t)$ is minimal. E.g., the Euclidean metrics can be used; then:

$$e_c = \min_k e_k = \min_k \sqrt{\sum_i \left(x_i(t) - v_{ik}(t)\right)^2}$$

The training algorithm for counterpropagation networks (2)

Step 4: Update the weights v_{ik} between the input neuron i and the neurons from the Kohonen layer, that belong to the neighborhood N_c of the neuron c to better correspond to the presented pattern $\vec{x}(t)$:

$$v_{ik}(t+1) = v_{ik}(t) + \alpha(t)\left(x_i(t) - v_{ik}(t)\right)$$

$\alpha(t)$, where $0 < \alpha(t) < 1$, is the training parameter for the weights between the input and Kohonen layer, that decreases with time.
t represents the current training step, $t + 1$ the following one.

The training algorithm for counterpropagation networks (3)

Step 5: Update the weights w_{ci} between the „winning“ neurone c from the Kohonen layer and the neurons of the Grossberg layer such that the output vector $\vec{y}$ better corresponds to the desired output $\vec{d}$:

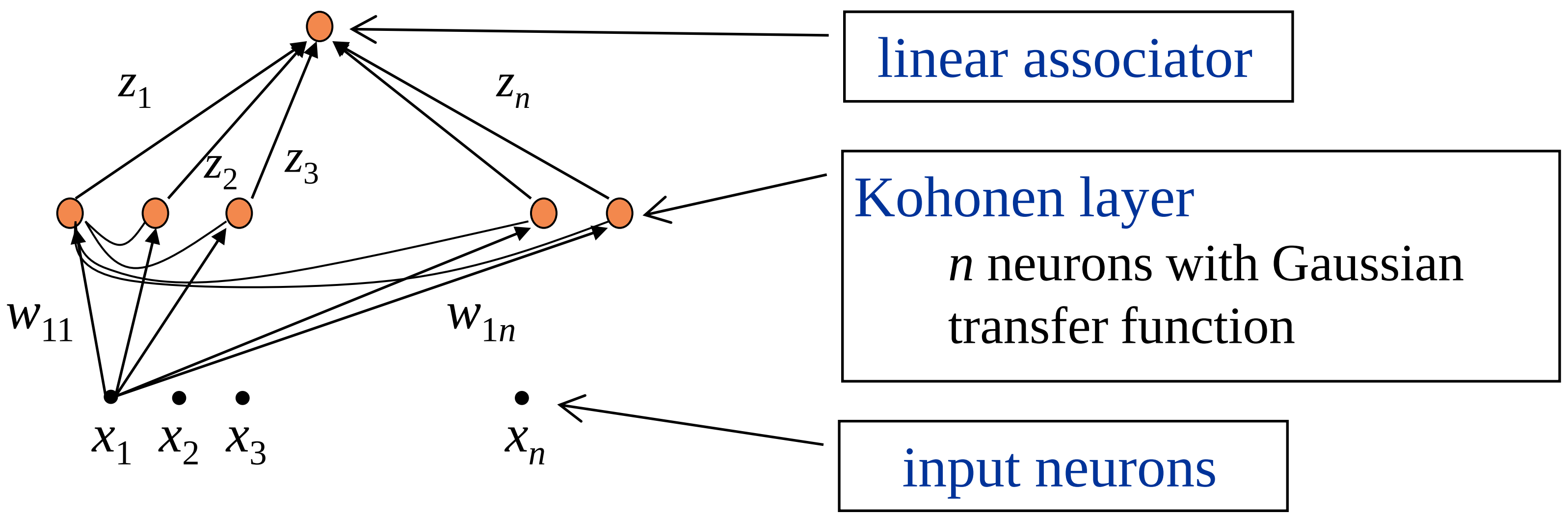
$$w_{cj}(t+1) = (1-\beta)w_{cj}(t) + \gamma z_c d_j$$

$w_{cj}(t)$ is the weight of the synaptic connection between the c-th neuron of the Kohonen layer and the j-th neuron of the Grossberg layer in time t, $w_{cj}(t + 1)$ denotes the value of this synapticconnection in time $t + 1$. β is a positive constant influencing the dependence of the new value of the synaptic weight on its value in the preceding training step. A positive constant γ represents learning rates for the weights between the Kohonen and Grossberg layer, z_c denotes the activity of the „winning“ neuron from the Kohonen layer.

Step 6: Go to Step 2.

RBF-networks (Radial Basis Functions)

- **Hybrid architecture** (Moody & Darken)
- **Supervised training**



RBF-networks (Radial Basis Functions)

- Every neuron j computes its output $g_j(t)$ according to:

$$g_j(\vec{x}) = \frac{\exp\left(\frac{(\vec{x}-\vec{w}_j)^2}{2\sigma_j^2}\right)}{\sum_k \exp\left(\frac{(\vec{x}-\vec{w}_k)^2}{2\sigma_k^2}\right)}$$

$\vec{x}$ ... input vector

$\vec{w}_1,\ldots,\vec{w}_m$ ... weight vectors of hidden neurons

$\sigma_1,\ldots,\sigma_m$ ... constants (set, e.g., according to the distance between the respective weight vector and its closest neighbor)

RBF-networks (Radial Basis Functions)

- the output of each hidden neuron is normalized
  - Mutual inter-connection of all the neurons
- the weights $z_1, \ldots, z_m$ can be found, e.g., by means of the back-propagation training algorithm:

$$E = \frac{1}{2} \sum_p \left(\sum_{i=1}^{n} g_i(\vec{x}_p)\, z_i - d_p \right)^2$$

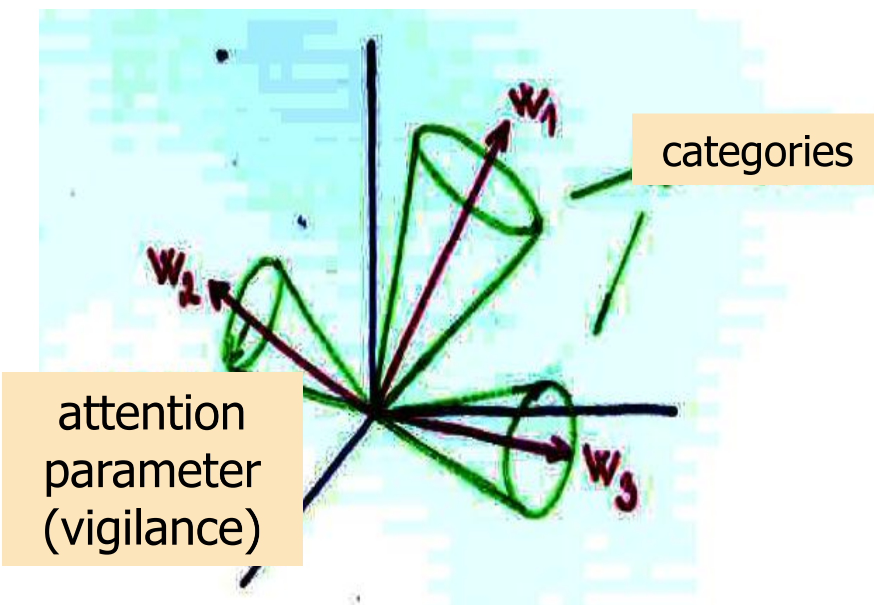
$$\Delta z_i \cong -\frac{\partial E}{\partial z_i} = \gamma\, g_i(\vec{x}) \left(d - \sum_{i=1}^{n} g_i(\vec{x})\, z_i \right)$$

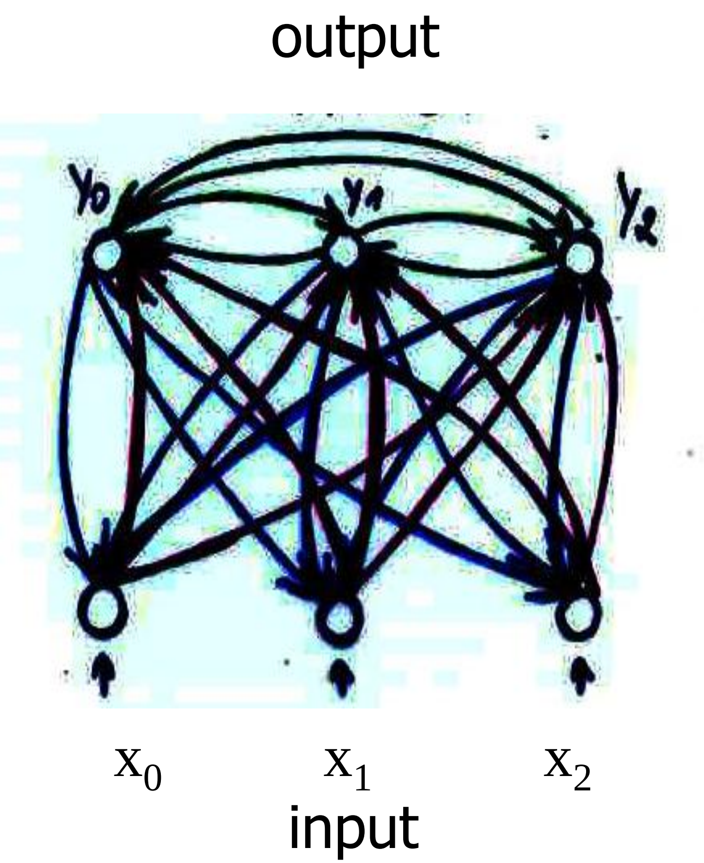
d ... the desired output výstup

p ... the number of training patterns

γ ... training parameter

ART – Adaptive Resonance Theory
(Carpenter & Grossberg)





ART – Adaptive Resonance Theory (2)
(Carpenter & Grossberg)

ART 1:

- Binary inputs, unsupervised training
- Lateral inhibition is used to find the output neuron with maximum response
- Feedback weights (from the output neurons to the input neurons) are used to compare the actual similarity of the processed input pattern with the recalled pattern (stored in the weights)

ART – Adaptive Resonance Theory (3)
(Carpenter & Grossberg)

ART 1 (continued):

- Vigilance test – attention parametr
- A mechanism to „switch off“ the output neuron with maximum response

  → **stability × plasticity of the network**

× the network has big problems even for „moderately noise-corrupted patterns“

  → **a growing number of stored patterns**

ART 1 – training algorithm

Step 1: **Initialization**

$$t_{ij}(0) = 1 \qquad 0 \leq i \leq N\text{-}1$$

$$b_{ij}(0) = 1 / (1 + N) \qquad 0 \leq j \leq M\text{-}1$$

$$\rho \qquad 0 \leq \rho \leq 1$$

$b_{ij}(t)$ ~ the weight between the input neuron i and the output neuron j in time t

$t_{ij}(t)$ ~ the weight between the output neuron j and ním the input neuron i in time t (determine the pattern specified by the output neuron j)

ρ ~ vigilance parameter (determines, how close should be the presented input to the stored pattern – to belong to the same category)

ART 1 – training algorithm (2)

Step 2: **Present a new input pattern**

Step 3: **Compute the activation of output neurons**

$$\mu_j = \sum_{i=0}^{N-1} b_{ij}(t)\, x_i \quad ; \quad 0 \le j \le M-1$$

$\boldsymbol{\mu_j}$ ~ output of the output neuron $\boldsymbol{j}$

$\boldsymbol{x_i}$ ~ $\boldsymbol{i}$ – th component of the input vector ($\in \{0,1\}$)

Step 4: **Find the stored pattern, that best represents the presented pattern** (e.g., by means of lateral inhibition):

$$\mu_{j^*} = \max_j \{ \mu_j \}$$

ART 1 – training algorithm (3)

Step 5: **Vigilance test**

$$\|\vec{x}\| = \sum_{i=0}^{N-1} x_i \quad \text{and} \quad \| T \cdot \vec{x} \| = \sum_{i=0}^{N-1} t_{ij^*} \, x_i$$

if $\dfrac{\| T \cdot \vec{x} \|}{\| \vec{x} \|} > \rho$, go to Step 7

else go to Step 6

Step 6: **Inhibit the best matching neuron**

the output of neuron ***j**** selected in Step 4 is temporarily set to zero (and not considered in the following maximization in Step 4). Afterwards go to Step 4.

ART 1 – training algorithm (4)

Step 7: **Adjustment of the best matching neuron**

$$t_{ij^*}(t+1) = t_{ij^*}(t) \cdot x_i$$

$$b_{ij^*}(t+1) = \frac{t_{ij^*}(t) \cdot x_i}{0.5 + \sum_{i=0}^{N-1} t_{ij^*}(t) \cdot x_i}$$

Step 8: **Go to Step 2 and repeat**

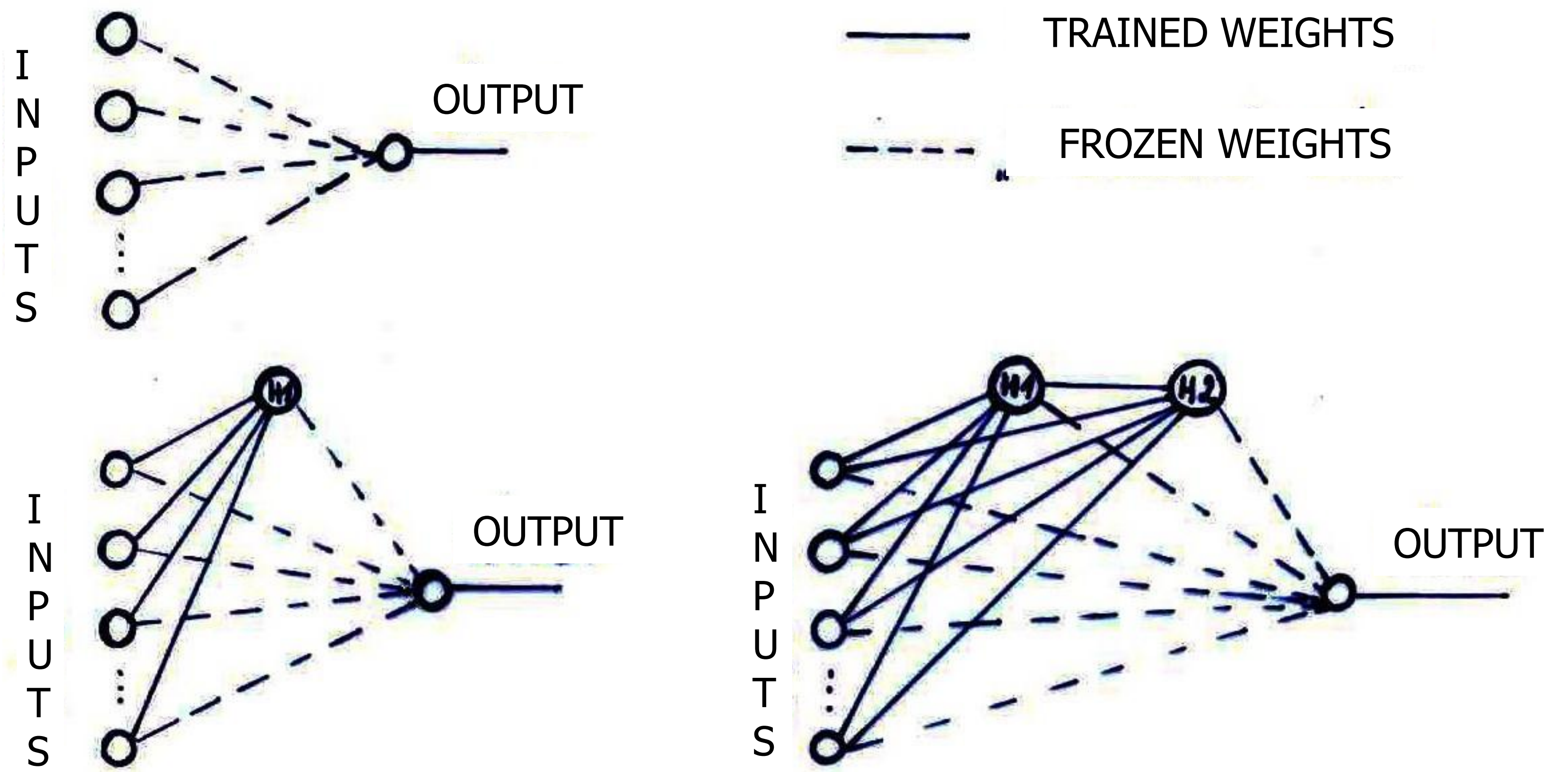
(Before that, „switch on“ all the neurons „switched off“ in Step 6)

Cascade correlation
(Fahlman & Lebiere, 1990)

~ a robust growing architecture

- The system starts training with a direct inter-connection between the inputs and outputs
- Subsequently, hidden neurons are added
- The inputs of each new neuron are inter-connected with all original inputs and previously created neurons

Cascade correlation (2)
(Fahlman & Lebiere, 1990)



Cascade correlation (3)
(Fahlman & Lebiere, 1990)

Training of the network (proceeds in 2 phases):

a) During the first phase, the already existing network is trained by means of Quickprop

- If the average quadratic error remains bigger than its desired level, a new neuron will be added to the network
- If the current error value is small enough, the algorithm stops

Cascade correlation (4)
(Fahlman & Lebiere, 1990)

Training of the network (continued):

b) The new added neuron belongs to a group of candidates trained to maximize correlation between their output and the error at network output

→ **the added neuron „has found“ a feature, that strongly correlates with the „residual“ error**

- The input weights of the added neuron will be frozen
- Only the weights from the added neuron to the output will be „retrained“

Cascade correlation (5)
(Fahlman & Lebiere, 1990)

Training of the network (continued):

While training hidden neurons, our objective is to maximilize ***S***:

$$S = \left| \sum_{i=1}^{p} \left(V_i - \overline{V} \right) \left(E_i - \overline{E} \right) \right|$$

p ….. the number of training patterns

V_i ….. output of the added neuron for the i – th pattern

$\overline{V}$ ….. average output of the added neuron

E_i ….. error for the i – th pattern

$\overline{E}$ ….. average error

Cascade correlation (6)
(Fahlman & Lebiere, 1990)

Training of the network (continued):

$$\frac{\partial S}{\partial w_k} = \sum_{i=1}^{p} \sigma \left(E_i - \bar{E} \right) f_i' I_{i,k}$$

$\boldsymbol{\sigma}$ ….. the sign of correlation between the added neuron and the output

f_i'….. the derivative of the transfer function for pattern ***i***

$I_{i,k}$ …. ***k*** – th input of the added neuron for pattern ***i***