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:

Self-Organization

doc. RNDr. Iveta Mrázová, CSc.

DEPARTMENT OF THEORETICAL COMPUTER SCIENCE AND MATHEMATICAL LOGIC

FACULTY OF MATHEMATICS AND PHYSICS, CHARLES UNIVERSITY IN PRAGUE

Neural Networks:

Contents:

- **Associative Memories**
- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

Self-Organization

- Unsupervised training:
  - Self-organization and clustering
- Motivation:
  - The network decides by itself what response fits best for the presented input pattern and adjusts its weights accordingly
- Problem:
  - Determine the number and location of clusters present in the feature space

Self-Organization (2)

Competitive learning:

- Compete for the „right to represent the patterns“
- *Winner - Takes - All* rule (WTA)
- *Inhibition* of opponents
- Network *plasticity*
- Learning with conscience

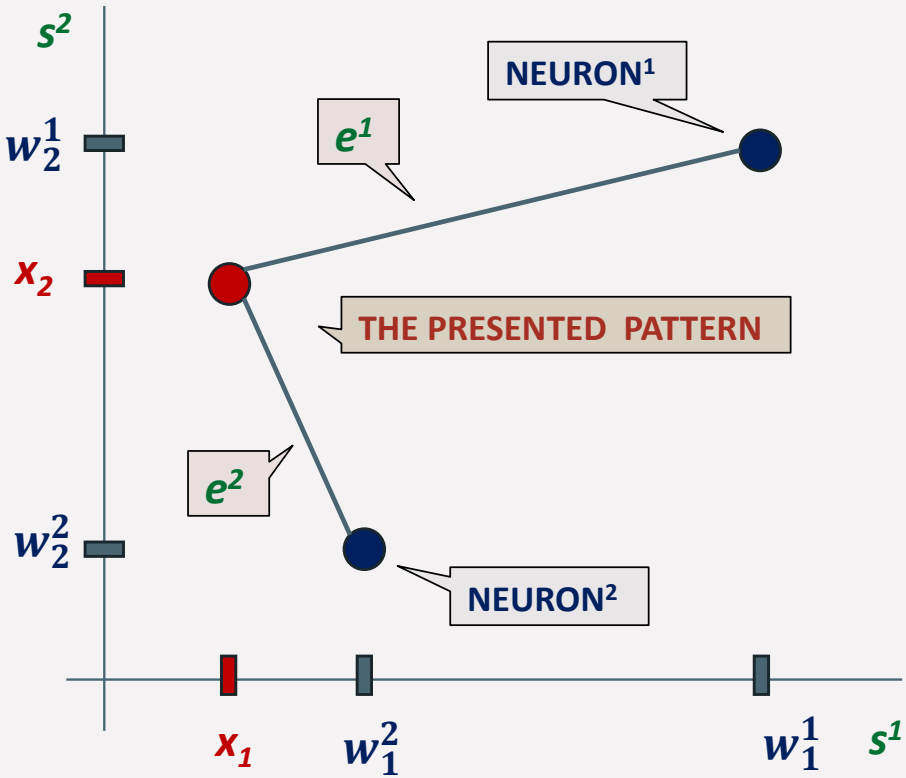
Reinforcement learning:

- Emphasizes the best reproduction of inputs

Unsupervised Competitive Learning

- n-dimensional input patterns are processed by such a number of neurons, that corresponds to the (assumed) number of clusters
- the neurons compute the (Euclidean) *distance* between the presented pattern and their weight vector

Unsupervised Competitive Learning (2)



- Competition „is won“ by the neuron situated the closest to the presented pattern
- The winning neuron becomes the most active one and will *inhibit* the activity of other neurons

Unsupervised Competitive Learning (3)

- Inhibition by means of „lateral connections“

  $\Rightarrow$ *lateral inhibition*

- Global information about the state of all the neurons in the network is necessary to decide whether a neuron will be active or not
- The activity of a neuron signals the membership of the presented input to the cluster of vectors represented by this neuron

Unsupervised Competitive Learning (4)

- The winning neuron adjusts its weights towards the presented pattern:

$$\Delta\, \vec{w} = \alpha \cdot (\vec{x} - \vec{w})$$

network plasticity (decays slowly during training)

Our objective:

- Position the neurons into the cluster centers
- Keep the already formed network structure

Unsupervised Competitive Learning (5)

- Strategies speeding-up the training:
  - An appropriate weight initialization, e.g., according to randomly selected patterns
- Problems:
  - Dead (never used) neurons
    - A grid in the Kohonen layer
    - Topological neighborhood of neurons
    - Controlled competition and the mechanism of conscience

Unsupervised Competitive Learning (6)

- During training, the weights of the neurons should be set in such a way that they correspond to the „centers of gravity of the respective clusters“
- The energy function of a set of n-dimensional ($n \geq 2$) normalized input patterns $X = \{\vec{x}_1, \ldots, \vec{x}_m\}$ is given for 1 neuron with the weight vector $\vec{w}$ by means of:

$$E_X(\vec{w}) = \sum_{i=1}^{m} \|\vec{x}_i - \vec{w}\|^2 ; \quad \vec{w} \in R^n$$

Unsupervised Competitive Learning (7)

⇒ in the optimum case, the weight vector is located in the center of the input pattern cluster:

$$E_X(\vec{w}) = \sum_{i=1}^{m} \|\vec{x}_i - \vec{w}\|^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(x_{ij} - w_j\right)^2 =$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} \left(x_{ij}^2 - 2x_{ij}w_j + w_j^2\right) =$$

$$= m\left(\sum_{j=1}^{n} w_j^2 - \frac{2}{m} \sum_{j=1}^{n} w_j\left(\sum_{i=1}^{m} x_{ij}\right)\right) + \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^2 =$$

$$m\left(\sum_{j=1}^{n} w_j^2 - \frac{2}{m}\sum_{j=1}^{n} w_j\left(\sum_{i=1}^{m} x_{ij}\right)\right) + \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^2$$

Unsupervised Competitive Learning (8)

$$E_X(\vec{w}) = m\sum_{j=1}^{n}\left(w_j^2 - \frac{2}{m}w_j\left(\sum_{i=1}^{m} x_{ij}\right) + \frac{1}{m^2}\left(\sum_{i=1}^{m} x_{ij}\right)\left(\sum_{i=1}^{m} x_{ij}\right)\right) -$$

$$\underbrace{- \frac{1}{m}\sum_{j=1}^{n}\left[\left(\sum_{i=1}^{m} x_{ij}\right)\left(\sum_{i=1}^{m} x_{ij}\right)\right] + \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^2}_{= K} =$$

$$= m\left(\sum_{j=1}^{n}\left(w_j - \frac{1}{m}\sum_{i=1}^{m} x_{ij}\right)^2\right) + K =$$

$$= m\,\|\vec{w} - \vec{x}^*\|^2 + K$$

Unsupervised Competitive Learning (9)

→ the vector $\vec{x}^*$ is the centroid of the cluster $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m\}$ and K is a constant

→ the energy function has a global minimum at $\vec{x}^*$

Unsupervised Competitive Learning (10)

Clustering methods for empirical multidimensional data:

- Two basic approaches:
  - k nearest neighbors – for labeled data
  - k-means algorithm – for unlabeled data
- k nearest neighbors (supervised training)
  - The training patterns are stored and classified into one of l different classes
  - A new input vector is classified into the class that contains the majority of its k nearest neighbors (from the stored set)

Unsupervised Competitive Learning (11)

- k-means clustering algorithm:
  - unsupervised learning
  - Input vectors are grouped into k different clusters

    (at the beginning, each cluster contains exactly 1 vector)
  - A new vector $\vec{x}$ is assigned to the cluster i, the centroid of which lies the closest to this pattern

Unsupervised Competitive Learning (12)

- k-means clustering algorithm (continue)
  - The centroid $\vec{c}_i$ is then adjusted by means of:

$$\vec{c}_i(new) = \vec{c}_i(old) + \frac{1}{n_i}\left(\vec{x} - \vec{c}_i(old)\right)$$

n_i … the number of vectors already assigned to cluster i

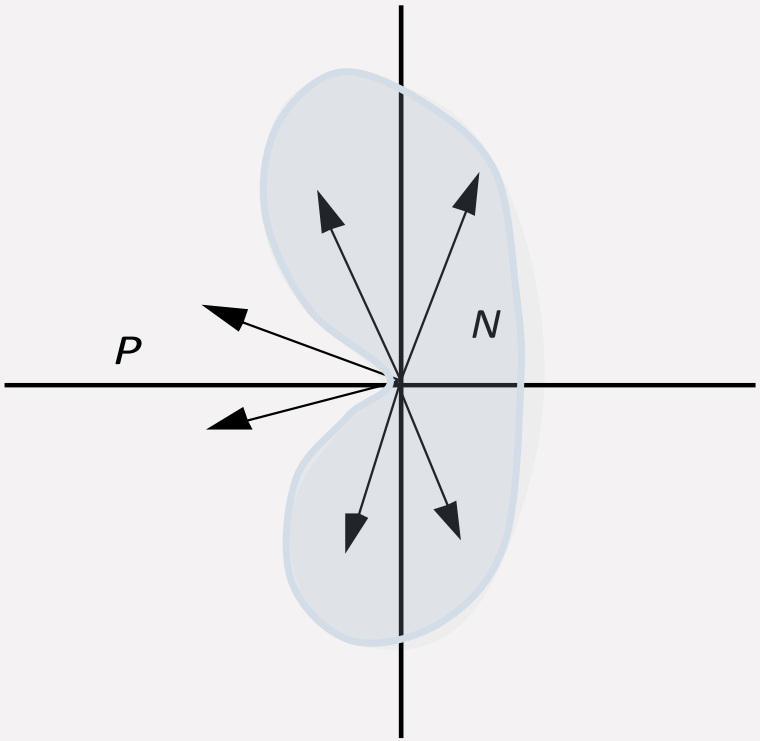
  - This procedure is iteratively repeated for the entire data set (its structure is then captured by the „weight vectors“ $\vec{c}_i$; $i = 1, \ldots, k$).

→ *Vector quantization*

The Clustering Problem

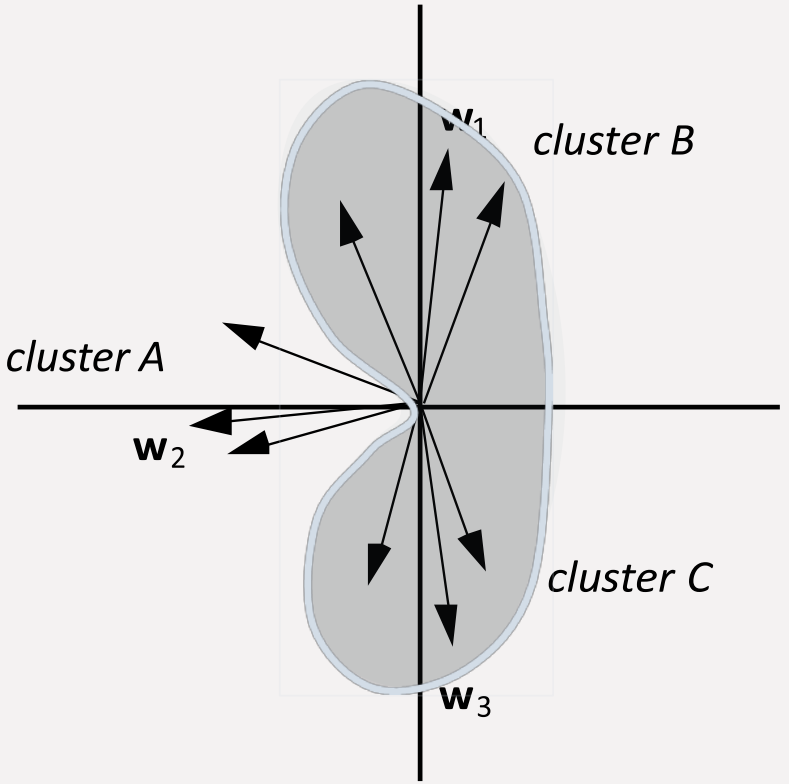
- Two sets of vectors: P and N
- Difficult to „separate“ the clusters by means of a simple perceptron such that:

$$\vec{w} \cdot \vec{p} \geq 0 \quad \forall \vec{p} \in P$$
$$\wedge \quad \vec{w} \cdot \vec{n} < 0 \quad \forall \vec{n} \in N$$



The Clustering Problem (2)

- Example: Three weight vectors for three clusters
- 3 different vectors $\vec{w}_1, \vec{w}_2$ and $\vec{w}_3$ can be used as „representants“ of the respective clusters A, B and C



The Clustering Problem (3)

- Each one of these vectors is „relatively close“ to any vector from the respective cluster
- Each weight vector corresponds to a single neuron which is active only if the input vector is close enough to its own weight vector

==> How could we determine the number and distribution of the clusters?

Neural Networks:

Contents:

- **Associative Memories**
- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

Competitive Learning: The Algorithm

- Let $X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_l\}$ be a set of normalized input vectors in the n-dimensional space which we want to classify into k different clusters
- The neural network consists of k neurons, each of which has n inputs and zero threshold

Initialization:

- The normalized weight vectors $\vec{w}_1, \dots, \vec{w}_k$ are generated randomly

Competitive Learning: The Algorithm (2)

Test:

- Select randomly a vector $\vec{x}_j \in X$
- Compute $\vec{w}_i \cdot \vec{x}_j$ for $i = 1, \ldots, k$
- Select $\vec{w}_m$ such that $\vec{w}_m \cdot \vec{x}_j \geq \vec{w}_i \cdot \vec{x}_j \quad (\forall i = 1, \ldots, k)$
- Continue with Update

Update:

- Substitute $\vec{w}_m(new)$ with $\vec{w}_m(old) + \vec{x}_j$ and normalize
- Continue with Test

Competitive Learning: The Algorithm (3)

- The algorithm can be stopped after a pre-determined number of steps
- The weight vectors of the k neurons are „attracted“ towards the centers of the respective clusters in the input space
- The algorithm is based on the principle known as *„winner-takes-all“*

Competitive Learning: The Algorithm (4)

- Normalized vectors prevent the weight vectors from becoming so large that they would win the competition too often
  - Other neurons would then never be updated and would remain useless → „*dead neurons*“
- Since both the input and weight vectors are normalized, the scalar product $\vec{w}_i \cdot \vec{x}_j$ of a weight and input vector is equal to the cosine of the angle between these two vectors

Competitive Learning: The Algorithm (5)

- The selection rule guarantees that the weight vector $\vec{w}_m$ of the cluster that is updated is the one that lies closest to the tested input vector
- The update rule rotates the weight vector $\vec{w}_m$ towards $\vec{x}_j$

Different learning rules:

- Update with a learning constant

  $\Delta\vec{w}_m = \eta\vec{x}_j;\ \ \eta \in (0,1)$ decays slowly in time

  → *network plasticity*

Competitive Learning: The Algorithm (6)

Different learning rules (continue):

- Difference update: $\Delta\vec{w}_m = \eta\left(\vec{x}_j - \vec{w}_m\right)$
  - „correction“ proportional to the difference of both vectors
- Batch update → a more stable learning process
  - Weight „corrections“ are computed for each respective pattern and then cumulated
  - After a number of iterations, the weight corrections are added to the weights at once

Neural Networks:

Contents:

- **Associative Memories**
- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

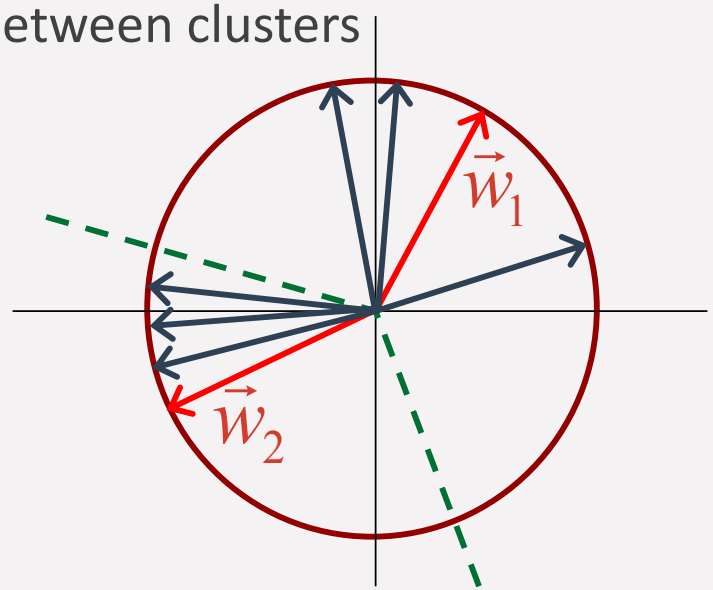
Competitive Learning: The Algorithm (7)

Stability of the solutions

- Necessity of a suitable measure for a „good clustering“

  → a simple approach: find the distance between clusters

- Two clusters of vectors and two „representative weight vectors“:
  - Both „representative vectors“ lie close to the vectors from their respective cluster

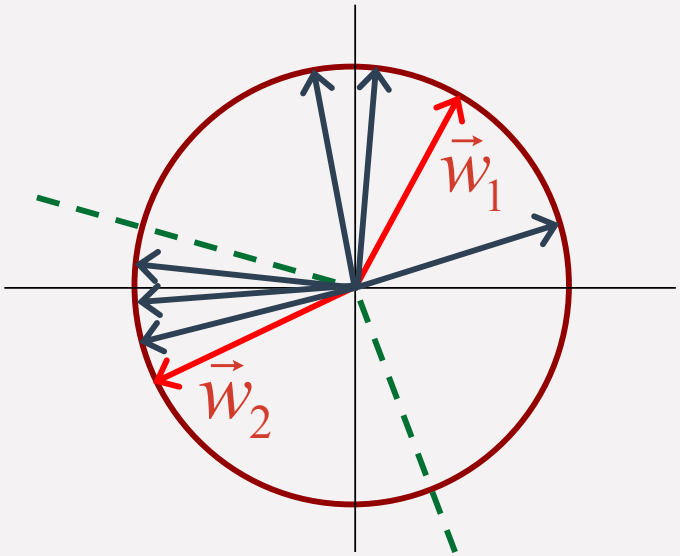


Competitive Learning: The Algorithm (8)

Stability of the solutions (continue):

- $\vec{w}_1$ lies inside a „cone“ defined by the vectors from its cluster
- $\vec{w}_2$ lies outside of the „cone“ of $\vec{w}_1$
- The vector $\vec{w}_1$ will not jump outside of its „cone“ in future iterations
- The vector $\vec{w}_2$ will jump inside the cone defined by its cluster at some point and will remain there

→ such a solution will be stable



Competitive Learning: The Algorithm (9)

The solution in a stable equilibrium:

- Intuitive idea:
  - A stable equilibrium requires clearly delimited clusters
- If the clusters overlap or are very extended, it can be the case that no stable solution can be found

  ==> unstable equilibrium

Stability of the Solutions: The Analysis

Definition:

Let P denote the set $\{\vec{p}_1, \ldots, \vec{p}_m\}$ of n-dimensional ($n \geq 2$) vectors located in the same half-space (~ a formal restriction of the cluster size).

The cone K defined by P is the set of all vectors $\vec{x}$ of the form $\vec{x} = \alpha_1\vec{p}_1 + \cdots + \alpha_m\vec{p}_m$, where $\alpha_1, \ldots, \alpha_m$ are positive real numbers.

- The cone of a cluster contains all vectors „within“ the cluster
- The diameter of a cone defined by normalized vectors is proportional to the maximum possible angle between two vectors in the cluster

Stability of the Solutions: The Analysis (2)

Definition:

The (angular) diameter φ of a cone K defined by normalized vectors $\{\vec{p}_1, \dots, \vec{p}_m\}$ corresponds to:

$$\phi = \sup\left\{\arccos\left(\vec{a}\cdot\vec{b}\right) \;\middle|\; \forall\vec{a},\vec{b}\in K;\ \|\vec{a}\| = \left\|\vec{b}\right\| = 1\right\}$$

where $0 \le \arccos\left(\vec{a}\cdot\vec{b}\right) \le \pi$

- A sufficient condition for stable equilibrium is that the angular diameter of the cluster´s cone must be smaller than the distance between clusters.

Stability of the Solutions: The Analysis (3)

Definition:

Let $P = \{\vec{p}_1, \ldots, \vec{p}_m\}$ and $N = \{\vec{n}_1, \ldots, \vec{n}_k\}$ be two non-empty sets of normalized vectors in an n-dimensional space ($n \geq 2$) that define the cones K_P and K_N

- If the intersection of the two cones is empty, the (angular) distance between K_N and K_P is given by:

$$\psi_{P,N} = \inf\{ \arccos(\vec{p} \cdot \vec{n}) ;\ \vec{p} \in K_P,\ \vec{n} \in K_N \text{ and } \|\vec{p}\| = \|\vec{n}\| = 1\}$$

  where $0 \leq \arccos(\vec{p} \cdot \vec{n}) \leq \pi$

- If the two cones K_P and K_N intersect, $\psi_{P,N} = 0$

Stability of the Solutions: The Analysis (4)

- If angular distance between clusters is greater than angular diameter of the clusters, a stable solution exists
  - The weight vectors will lie inside their respective cluster cones
  - Once inside their respective cluster cones, the weight vectors will not leave them

Clustering quality control:

- A smaller number of „more compact“ clusters is usually preferred
- A cost function penalizing a too big number of clusters

Neural Networks:

Contents:

- **Associative Memories**
- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

PCA – Principal Component Analysis

~ reduces the dimensionality of the input data

→ use fewer features without losing essential information

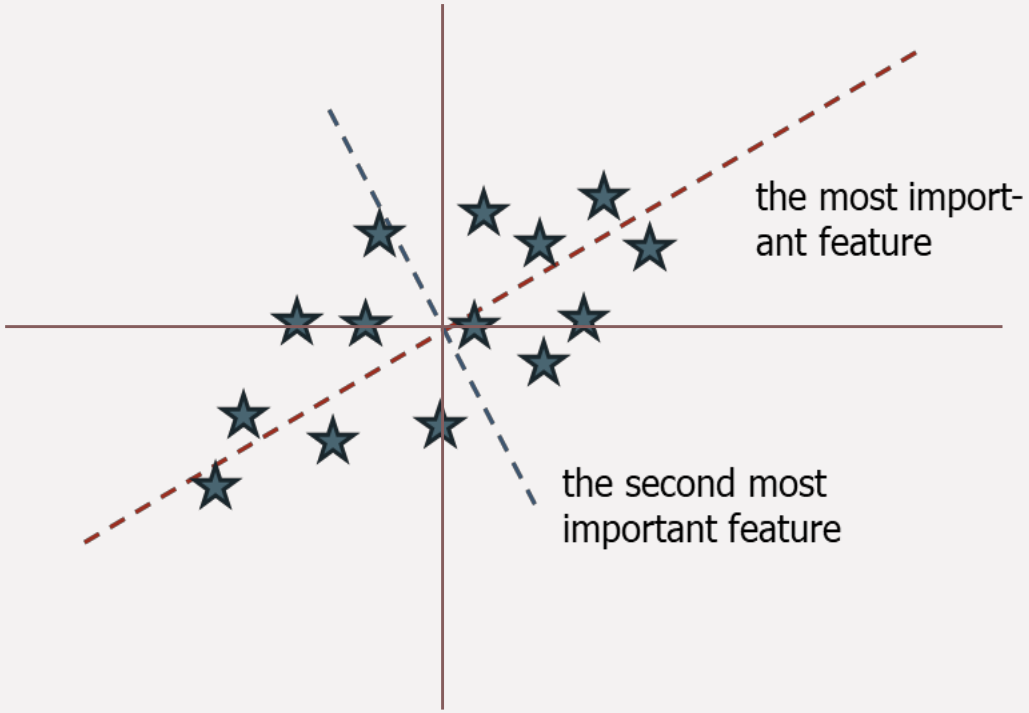
→ selection of the most important features

~ a set of m n-dimensional vectors is given: $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m\}$

~ the first principal component of this set of vectors is a vector $\vec{w}$ which maximizes the expression

$$\frac{1}{m} \sum_{i=1}^{m} (\vec{w} \cdot \vec{x}_i)^2$$

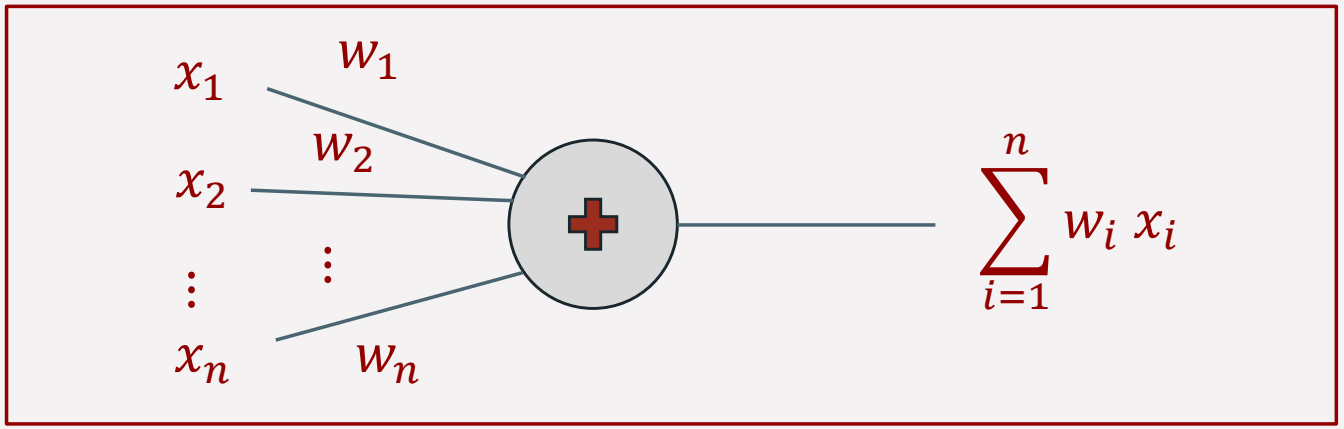
PCA – Principal Component Analysis (2)



- Distribution of the input data:
  - The first principal component: direction of maximum variance in the data
  - The second principal component: orthogonal to 1. PC and maximum variance

    (~ subtract from $\vec{x}$ its orthogonal projection on 1. PC)

PCA – Principal Component Analysis (3)



The applied model:

- Linear associator – outputs the weighted input as a result
- Unsupervised reinforcement learning – Oja's algorithm

Computation of the Principal Components with Artificial Neural Networks

Oja's learning algorithm (E. Oja, 1982)
(for the computation of the first principal component)

Assumption:

- The centroid of the input data is located at the origin

Start:

- Let X be a set of n-dimensional vectors
- The vector $\vec{w}$ is initialized randomly ($\vec{w} \neq 0$)
- A learning constant γ with $0 < \gamma \leq 1$ is selected

Computation of the Principal Components with Artificial Neural Networks (2)

Oja's learning algorithm (continue):

Update:

- From the set X a vector $\vec{x}$ is selected randomly
- The dot product $\Phi = \vec{x} \cdot \vec{w}$ is computed
- The new weight vector is set to: $\vec{w} + \gamma\, \Phi(\vec{x} - \Phi\, \vec{w})$
- Make γ smaller and go to „Update"

Stopping condition for update

- e.g., a predetermined number of iterations

Computation of the Principal Components with Artificial Neural Networks (3)

- The learning constant γ
  - The learning constant must be chosen small enough to guarantee adequate weight updates (limit big oscillations)
- „Automatic normalization“ of the weight vector
  - global information about all patterns is not necessary
  - local information about the updated weight, input, and scalar product of the associator is sufficient
- The first principal component is equivalent to the direction of the longest eigenvector of the correlation matrix of the considered input vectors

Neural Networks:

Contents:

- Associative Memories
- Self-Organization
- Kohonen Maps, Hybrid Models, and Genetic Algorithms

Contents:

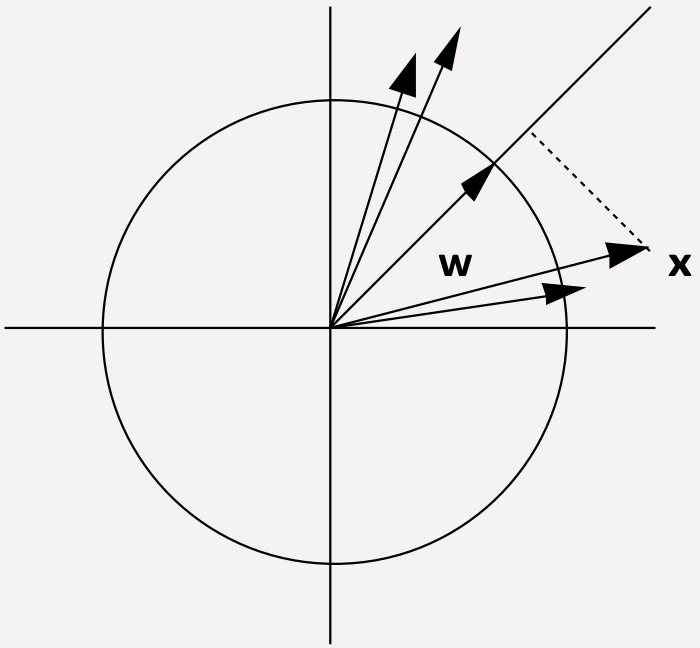
Convergence of The Oja's Algorithm

When a unique solution to the task exists, Oja's algorithm will converge:

Idea of the proof:

Update of $\vec{w}$ towards $\vec{x}$

- if Oja's algorithm is started for a weight vector inside a cone, it will oscillate in it, but will not leave it
- for $\|\vec{w}\| = 1$ the dot product $\Phi = \vec{x} \cdot \vec{w}$ corresponds to the length of the projection of $\vec{x}$ on $\vec{w}$



Convergence of The Oja's Algorithm (2)

Idea of the proof (continue):

-) the vector $\vec{x} - \Phi\vec{w}$ is orthogonal to $\vec{w}$
-) an iteration of Oja's algorithm attracts $\vec{w}$ to the vectors from cluster X
-) if the length of $\vec{w}$ remains equal to 1 (or close to 1), $\vec{w}$ will be brought into the middle of the cluster
-) further, it is necessary to show that the vector $\vec{w}$ is automatically normalized by the Oja's learning algorithm:

  Idea: a) the length of the vector $\vec{w}$ is bigger than 1

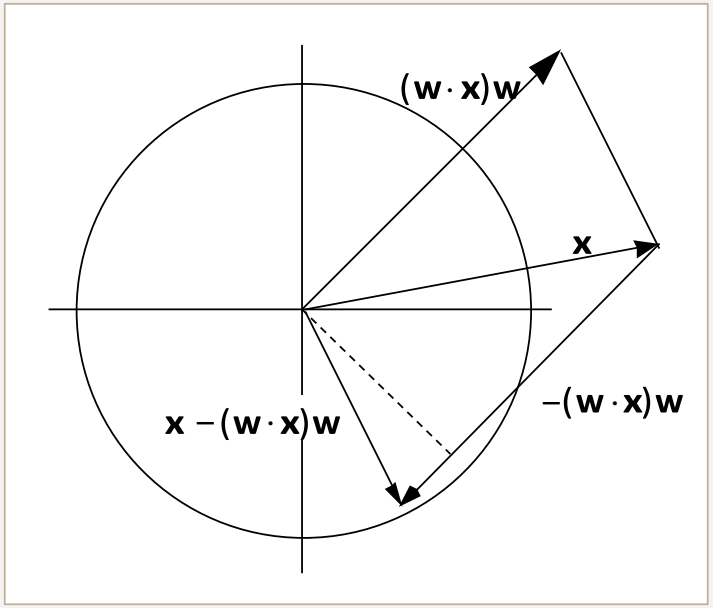
  b) the length of the vector $\vec{w}$ is smaller than $\boldsymbol{1}$

Convergence of The Oja's Algorithm (3)

a) the length of the vector $\vec{w}$ is bigger than 1

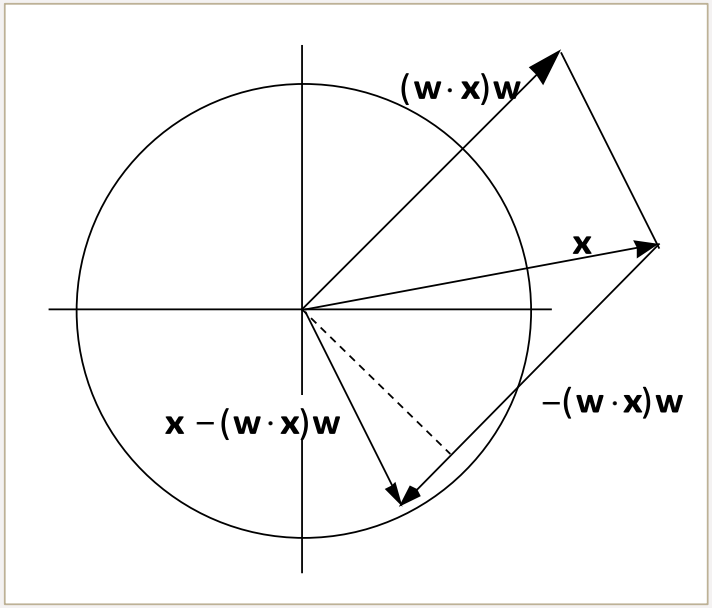
- The length of the vector $(\vec{x} \cdot \vec{w})\,\vec{w}$ is bigger than the length of the orthogonal projection of $\vec{x}$ on $\vec{w}$
- Further, assume that $\vec{x} \cdot \vec{w} > 0$, i.e., the vectors $\vec{x}$ and $\vec{w}$ are not too far away one from the other
- The vector $\vec{x} - (\vec{x} \cdot \vec{w})\,\vec{w}$ has a negative projection on $\vec{w}$, as:

$$(\vec{x} - (\vec{x} \cdot \vec{w})\vec{w}) \cdot \vec{w} = \vec{x} \cdot \vec{w} - \|\vec{w}\|^2 \vec{x} \cdot \vec{w} < 0$$



Convergence of The Oja's Algorithm (4)

- The result of many iterations:
  - (the vector $\vec{x} - (\vec{x} \cdot \vec{w})\vec{w}$ has one component normal to $\vec{w}$ and another one with the opposite direction of $\vec{w}$)
  - $\vec{w}$ will be brought into the middle of the cluster of vectors (and the normal component cancels in average)
  - $\vec{w}$ will become smaller with the growing number of iterations of this type (!avoid making $\vec{w}$ too small or reversing its direction in an iteration!)



Convergence of The Oja's Algorithm (5)

- A suitable choice of the learning parameter γ and normalization of the training vectors:
  - if the vector $\vec{x}$ has a positive dot product Φ with $\vec{w}$, then this should hold also for the new weight vector; it should thus hold:

$$\vec{x} \cdot \left(\vec{w} + \gamma\Phi(\vec{x} - \Phi\vec{w})\right) > 0$$

$$\Phi + \gamma\Phi\|\vec{x}\|^2 - \gamma\Phi\Phi^2 > 0$$

$$\underbrace{\Phi}_{>0}\ \underbrace{\left(1 + \gamma(\|\vec{x}\|^2 - \Phi^2)\right)}_{\Rightarrow \gamma(\|x^2\| - \Phi^2) > -1} > 0$$

  - For positive small enough γ, is always satisfied

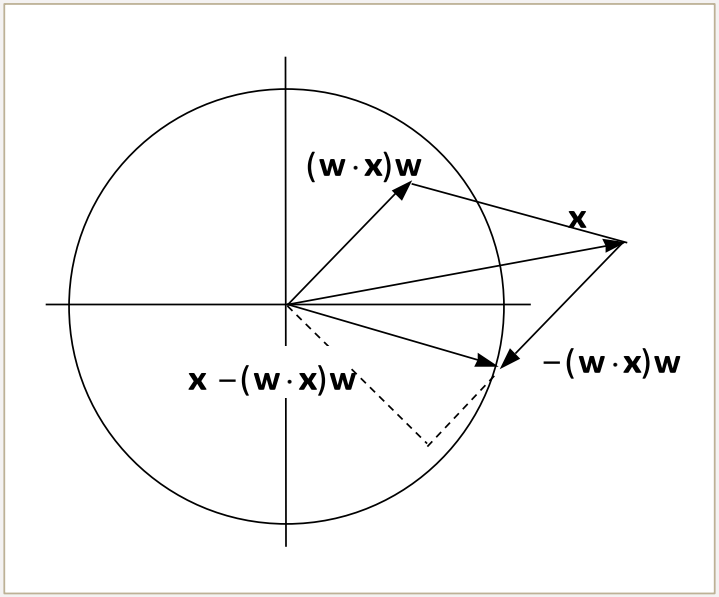
Convergence of The Oja's Algorithm (6)

b) the length of the vector $\vec{w}$ is smaller than 1

(similarly to a))

- The vector $\vec{x} - (\vec{x} \cdot \vec{w})\vec{w}$ has a positive projection on $\vec{w}$
- growing of $\vec{w}$

- **Combining a) and b)** $\Rightarrow$ $\vec{w}$ will be brought into the middle of the cluster and the length of $\vec{w}$ will oscillate around 1 (for a small enough γ)
- Problems: „sparse" clusters

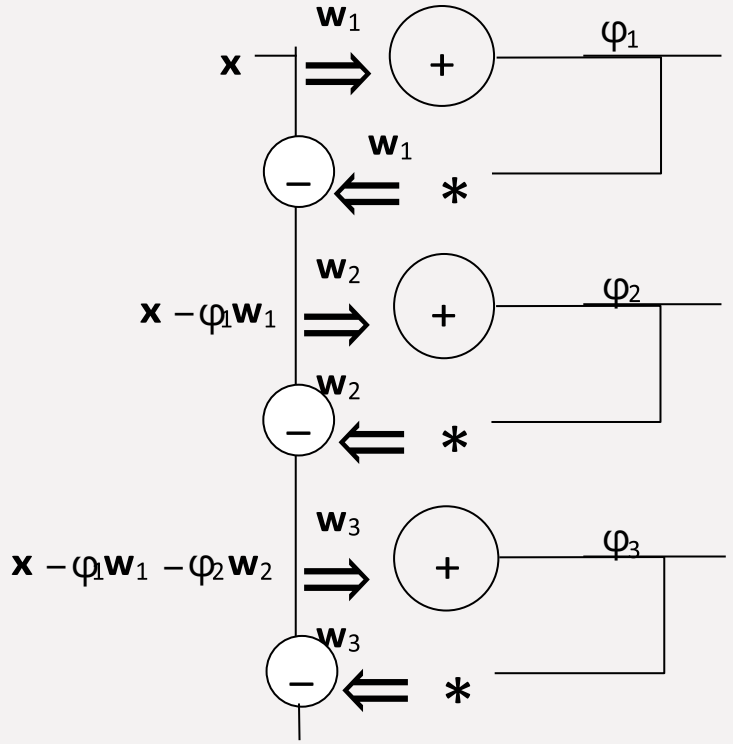


Oja's Learning Algorithm: Problems and Generalization

Problems:

- „sparse" clusters
- big differences in the length of the input vectors

Computation of more principal components:



Neural Networks:

Contents:

- Associative Memories
- Self-Organization
- Kohonen Maps, Hybrid Models, and Genetic Algorithms

Contents:

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

Neural Networks:

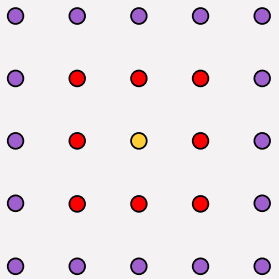
Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

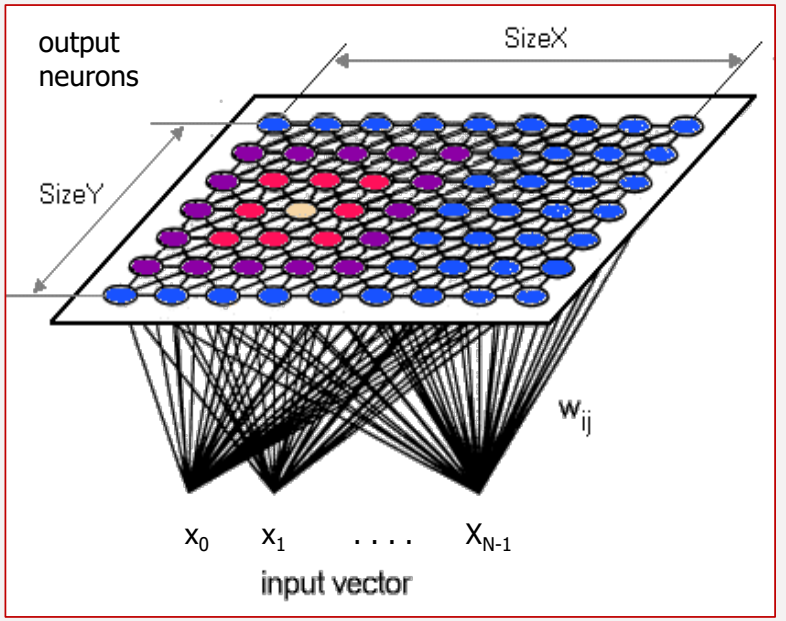
Kohonen Self-Organizing Feature Maps

- Teuvo Kohonen – phonetic typewriter
  - Unsupervised training
  - Recall
  - Applications: economics, text and image processing, etc.
  - topological neighborhood



$NE_j(O)$

$NE_j(t_1)$



Kohonen Model: The 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

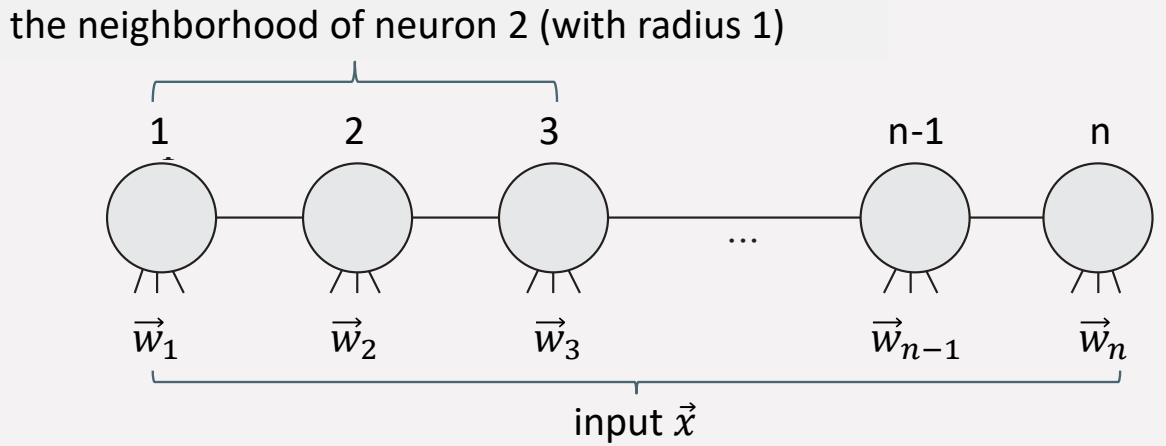
The objective: neighboring neurons should respond to closely related signals

Kohonen Model:
The Training Algorithm (2)

The problem (1-dimensional):

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

A one-dimensional lattice of neurons



Kohonen Model:
The Training algorithm (3)

Problem (1-dimensional – continued):

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

The 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:
The Training algorithm (4)

Problem (1-dimensional – continued):

→ „Kohonen“ neurons compute the Euclidean distance between the input $\vec{x}$ and the corresponding weight vector $\vec{w}$

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

Kohonen Model: The 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, …)

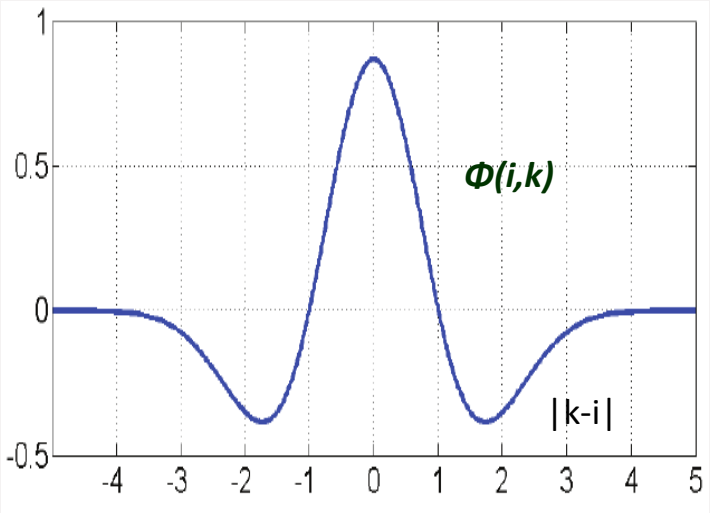
Kohonen Model:
The Training Algorithm (6)

Lateral interaction function $\Phi(i,k)$:

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

Example:

- $\Phi(i,k)=1 \ \forall i$ from the neighborhood of k with radius r and $\Phi(i,k)=0$ for all 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. Initialize also 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 output neurons j as:

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

where $x_i(t)$ denotes the input of neuron i at time t and $w_{ij}(t)$ corresponds to the synaptic weight between the input neuron i and the output neuron j at 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 $j \in N_c$; $0 \le i \le N\text{-}1$

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

Kohonen Self-Organizing Feature Maps: The Training Algorithm (3)

e.g., for $\alpha(t) = \frac{1}{t}$: $\sum_{t=1}^{\infty} 1/t = \infty$

$\sum_{t=1}^{\infty} 1/t^2 = \frac{\pi^2}{6}$

For the choice of *α(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 neighborhood of the winner.

The value of the function *Φ(c,j)* decreases with growing distance of the neurons from the center of the neighborhood N_c.

Step 6: Repeat by Going to Step 2.

Neural Networks:

Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

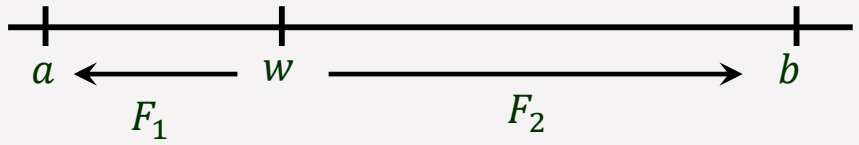
Contents:

Analysis of Convergence:
Stability of The Solution and an Ordered State

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

1) One-dimensional case:

   a) interval $[a, b]$, 1 neuron with the weight w, no neighborhood considered:



   → convergence of w towards the center of $[a, b]$

Analysis of Convergence:
Stability of The Solution and an Ordered State (2)

- The update rule: $w_n = w_{n-1} + \alpha(x - w_{n-1})$

  w_n, w_{n-1} … weight values at time n and $n-1$

  x …… a random number from the interval $[a, b]$

- If $0 < \alpha \leq 1$, the series $w_1, w_2, \ldots$ cannot leave $[a, b]$
- Bounded is also the expected value $\langle w \rangle$ of the weight w

- The expected value of the derivative of w with respect to t is zero: $\left\langle \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} \right\rangle = 0$, otherwise $\langle w \rangle$ would be $\langle w \rangle < a$ or $\langle w \rangle > b$

- As $\left\langle \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} \right\rangle = \alpha(\langle x \rangle - \langle w \rangle) = \alpha\left(\frac{\boldsymbol{a+b}}{2} - \langle w \rangle\right)$, it follows: $\langle w \rangle = (a+b)/2$

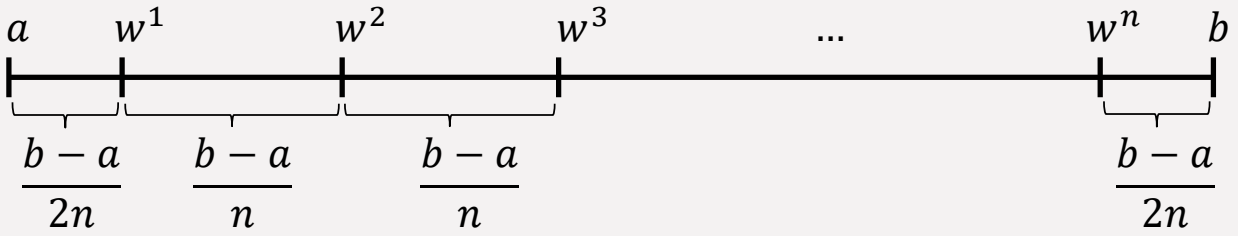
Analysis of Convergence:
Stability of The Solution and an Ordered State (3)

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

- no neighborhood considered,
- the weights are assumed to be monotonically ordered:

$$a < x^1 < x^2 < \cdots < x^n < b$$

→ the weights converge to: $\langle w^i \rangle = a + (2i - 1)\ \frac{b-a}{2n}$



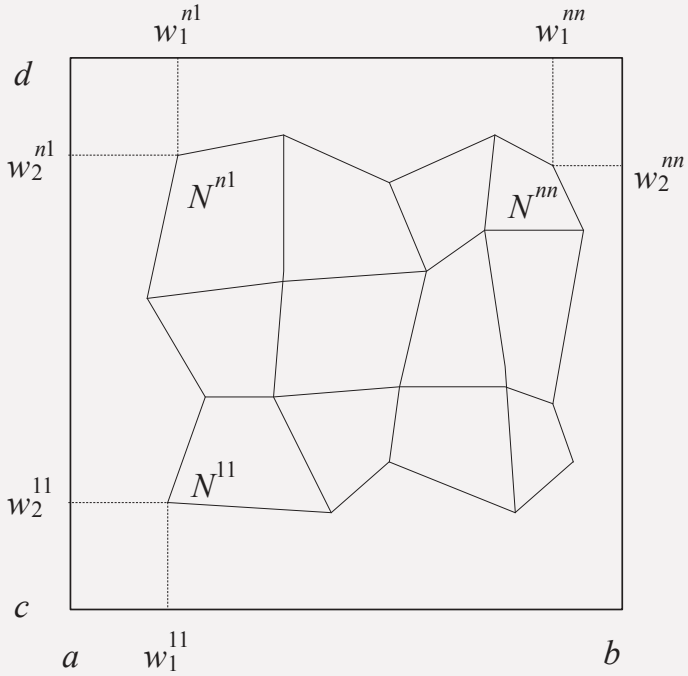
Analysis of Convergence:
Stability of The Solution and an Ordered State (4)

2) Two-dimensional case:

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

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

→ The problem will be reduced to two 1-dimensional problems



N^{ij} is the neuron at row i and column j with the weights w_1^{ij} and w_2^{ij}

Analysis of Convergence:
Stability of The Solution and an Ordered State (5)

2) Two-dimensional case (continued):

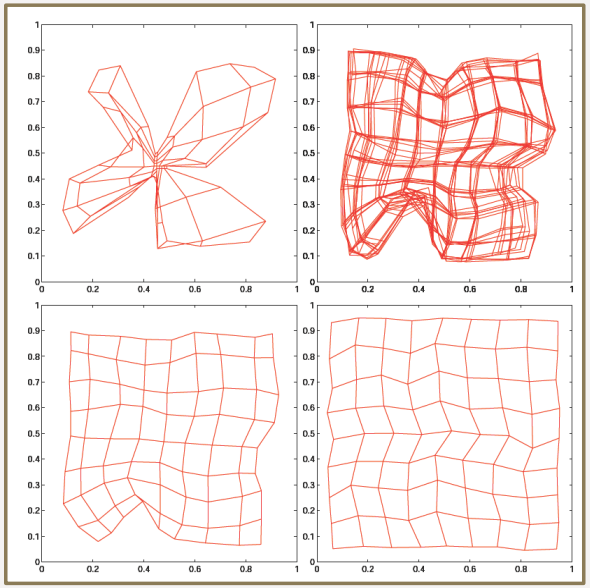
- Let $w_j^1 = \frac{1}{n}\sum_{i=1}^{n} w_1^{ij}$ denote the average weight value of the neurons from the j-th column
- Since $w_1^{ij} < w_1^{ik}$ for $j < k$, these average values w_1^j will be monotonically arranged: $a < w_1^1 < w_1^2 < \cdots < b$
- In the first column, the average weight value 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)

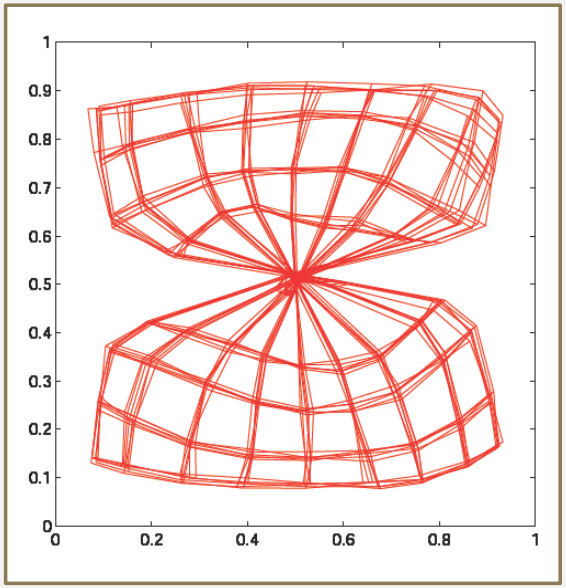
Analysis of Convergence:
Stability of The Solution and an Ordered State (6)

PROBLEMS:

- „unfolding“ a planar mesh and conditions, under which it will happen



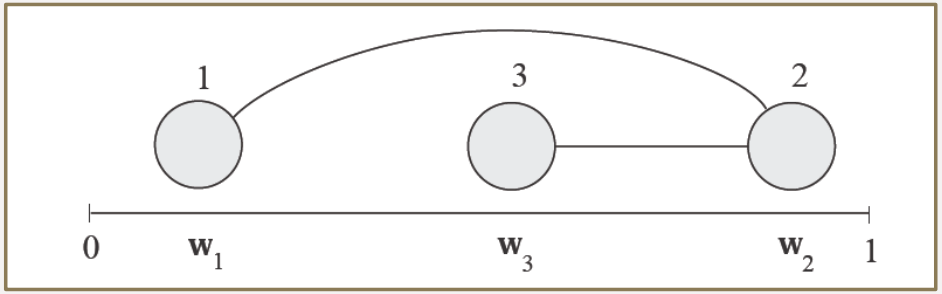
But sometimes



Analysis of Convergence:
Stability of The Solution and an Ordered State (7)

PROBLEMS:

- „meta-stable states“ and improper choice of lateral interaction (a too fast decrease)



→ 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

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

where k denotes the winning neuron and its two neighbors (Cottrell & Fort, 1986)

Neural Networks:

Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

Variants of The Training Algorithm for Kohonen Maps

Supervised training:

(LVQ ~ Learning Vector Quantization)

LVQ1:

- Motivation:

  -) $\vec{x}$ should belong to the same class as 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 $\vec{x}$ (c ~ the winning neuron)

Variants of The Training Algorithm for Kohonen Maps (2)

LVQ1 (continued):

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

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

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

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

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

- $\vec{w}_i(t+1) = \vec{w}_i(t)$ 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:

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

(recommended values for ω (~ the width of the window): $0.2 - 0.3$)

Variants of The Training Algorithm for Kohonen Maps (4)

LVQ2.1 (continued):

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

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

- approximation of the class distributions and stabilization of the solution

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

- $\vec{w}_i(t+1) = \vec{w}_i(t) - \alpha(t)[\vec{x}(t) - \vec{w}_i(t)]$
- $\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha(t)\left[\vec{x}(t) - \vec{w}_j(t)\right]$
  - $\vec{w}_i$ 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 $\vec{x}$ comes from the „window“
- $\vec{w}_k(t+1) = \vec{w}_k(t) + \varepsilon\alpha(t)[\vec{x}(t) - \vec{w}_k(t)]$
  - for $k \in \{i, j\}$ if $\vec{x}$, $\vec{w}_i$ and $\vec{w}_j$ belong to the same class
- The choice of parametersfor the „window“: $0.1 \leq \varepsilon \leq 0.5$ and $0.2 \leq \omega \leq 0.3$

Neural Networks:

Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

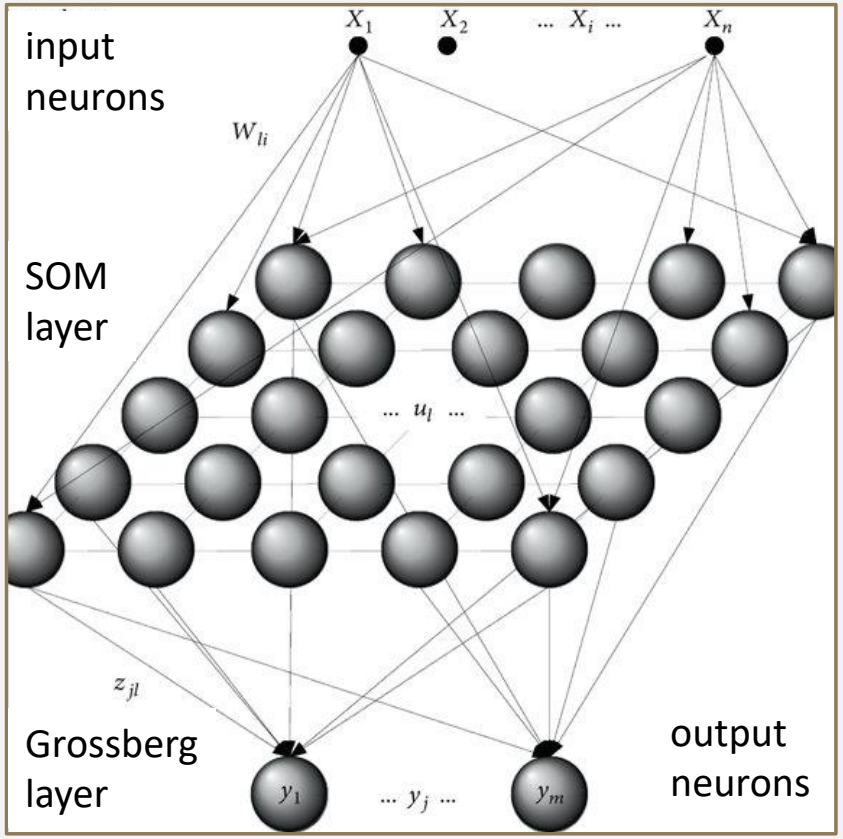
Variants of The Training Algorithm for Kohonen Maps (7)

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:

$$\textit{(input, desired output)} = \left(\vec{x}(t), \vec{d}(t)\right).$$

Step 3: In the Kohonen layer, find the winning 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_c between the selected weight vector $\vec{v}_c(t)$ 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 Kohonen layer that belong to the neighborhood N_c of neuron c to better correspond to the presented pattern $\vec{x}(t)$:

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

$\alpha(t)$, 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“ neuron 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 $\vec{d}(t)$:

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

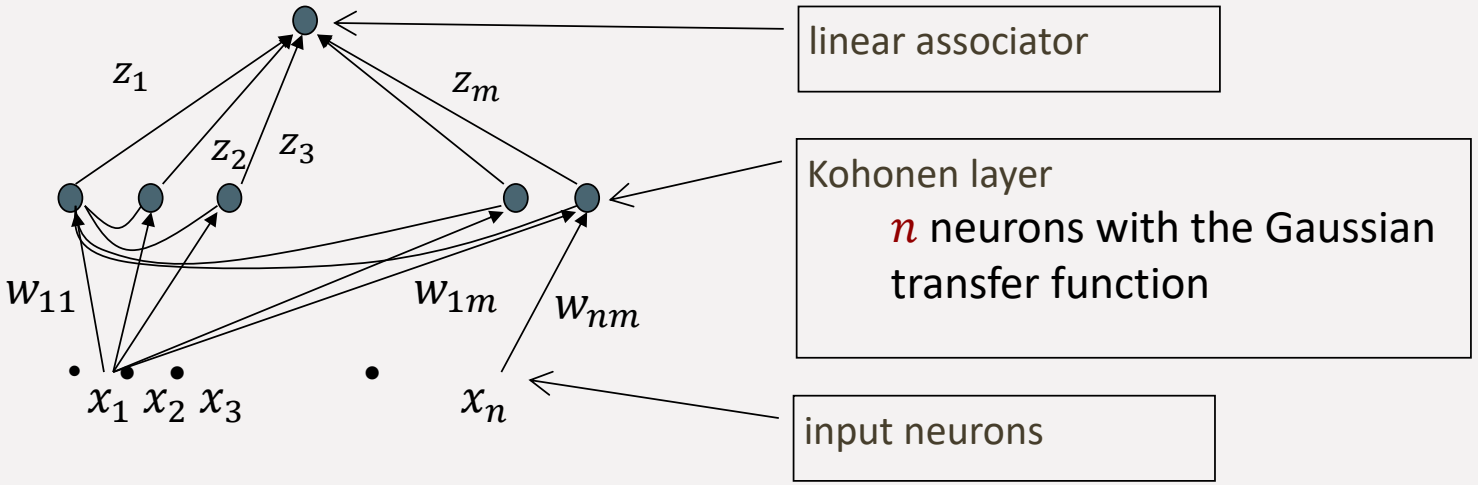
$w_{cj}(t)$ is the weight of the synaptic connection between the c-th neuron of the Kohonen layer and the j-th neuron of the Grossberg layer in time t, $w_{cj}(t+1)$ denotes the value of this synaptic connection in time $t+1$.

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 the learning rate 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

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

$\sigma_1, \ldots, \sigma_n$ ... 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 $z_1, \dots, z_m$ can be found, e.g., by means of the back-propagation training algorithm:

The error function on the whole training set

$$E = \frac{1}{2} \sum_p \left(\sum_{j=1}^{m} g_j(\vec{x}_p)\ z_j - d_p \right)^2$$

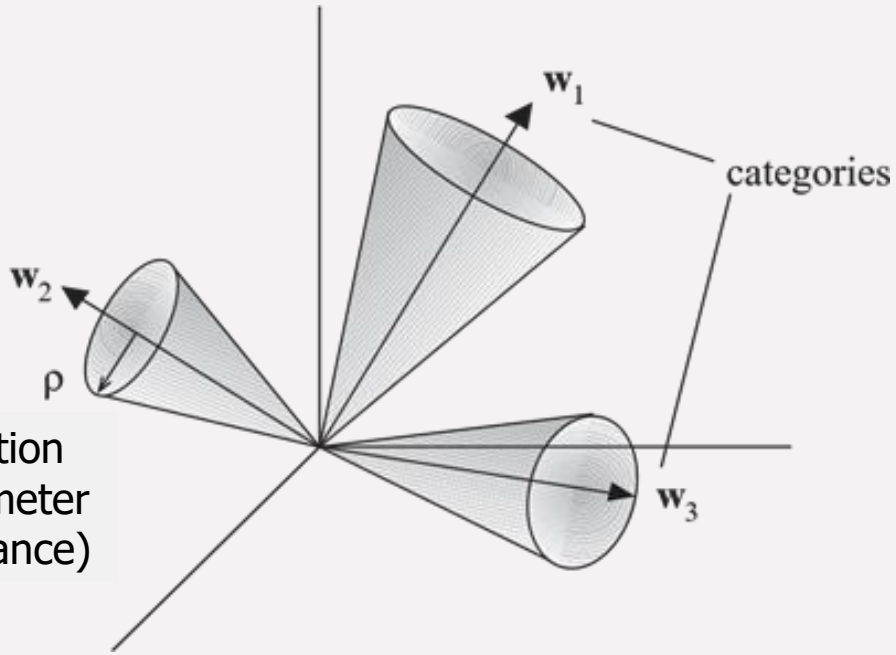
Adaptation for a single training pattern (x, d)

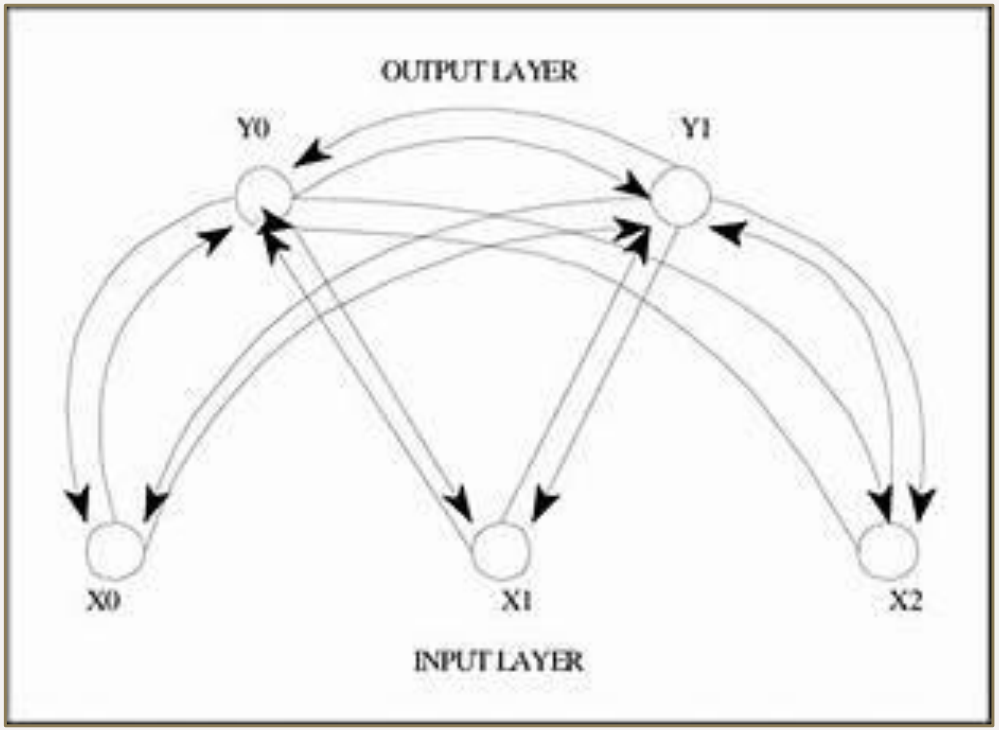
$$\Delta z_i \cong -\frac{\partial E}{\partial z_i} = \gamma g_i(\vec{x}) \left(d - \sum_{j=1}^{m} g_j(\vec{x})\ z_j \right)$$

d ... the desired output
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 parameter ρ
- 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 – The Training Algorithm

Step 1: Initialization

$$t_{ij}(0) = 1 \qquad 0 \le i \le N-1$$
$$b_{ij}(0) = 1/(1+N) \qquad 0 \le j \le M-1$$
$$\rho \qquad 0 \le \rho \le 1$$

$b_{ij}(t)$ ~ 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 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 – The Training Algorithm (2)

Step 2: Present a new input pattern

Step 3: Compute the activation of output neurons

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

μ_j ~ output of the output neuron j

x_i ~ 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 – The Training Algorithm (3)

Step 5: Vigilance test (using feedback weights)

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

if $\frac{\|T \cdot \vec{x}\|}{\|\vec{x}\|} > \rho$, 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 – The Training Algorithm (4)

Step 7: Adjustment of the best matching neuron

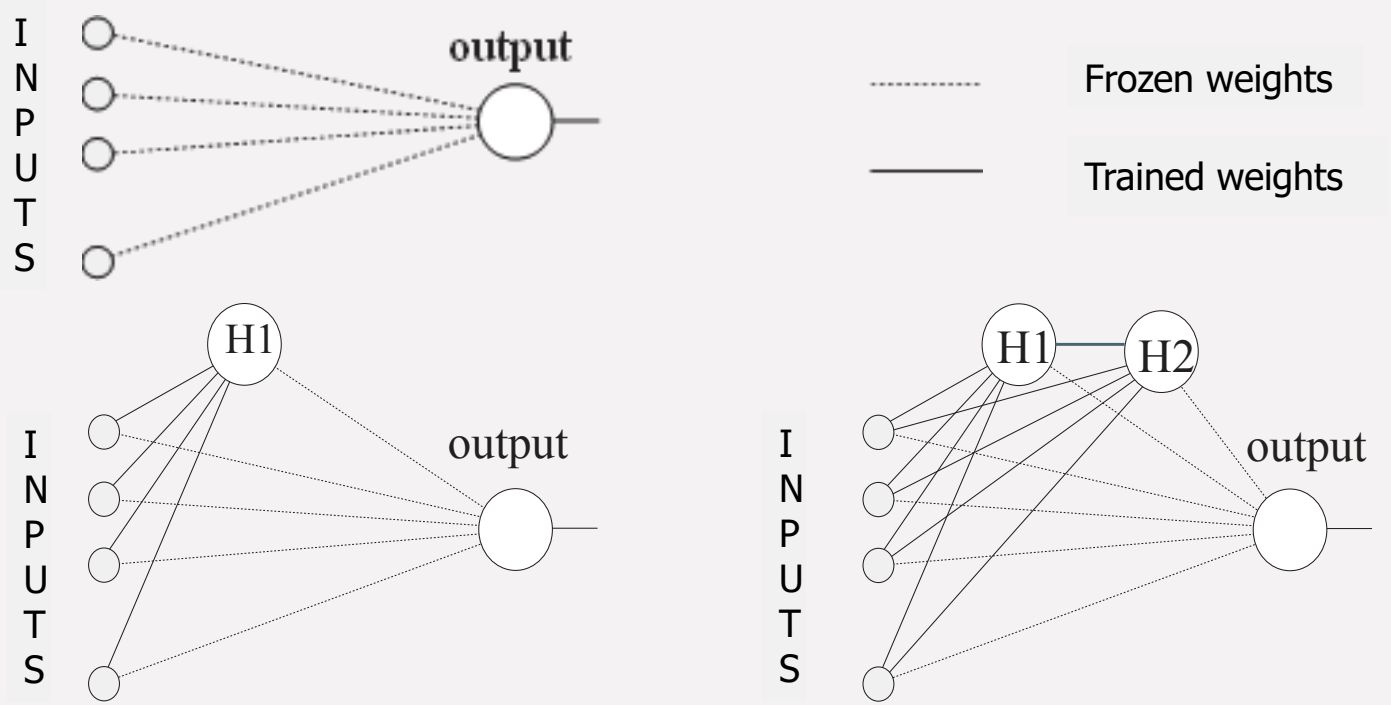
$$t_{ij^*}(t+1) = t_{ij^*}(t) \cdot x_i$$

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

Step 8: Go to Step 2 and repeat

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

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

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

Training of the network (continued):

b) The new added neuron is trained to maximize correlation between its output and the error at network output

→ **the output of the trained added neuron 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 maximize S:

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

p ..... the number of training patterns

V_i ..... output of the added neuron for the i-th pattern

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

E_i ..... error for the i-th pattern

$\bar{E}$ ..... average error

Cascade Correlation (Fahlman & Lebiere, 1990)

~ a robust growing architecture

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

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

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

Training of the network (continued):

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

σ ….. 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

Neural Networks:

Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

Genetic Algorithms (GA)

- Lawrence Fogel
  - studied ***recombination and mutating*** of populations
  - *L. Fogel et al.: Artificial Intelligence Through Simulated Evolution, John Wiley and Sons, New York, 1966*
- John Holland
  - proposed the ***concept of schemata*** and studied their diffusion dynamics
  - *J. Holland: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975*
- David Goldberg
  - summarized ***evolutionary methods*** for the solution of optimization problems
  - *D. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989*

Genetic Algorithms (GA)

- Artificial chromosome (genotype)

  ~ a string of symbols that code the properties of the individual (phenotype), e.g., a binary value of a variable or their sequence

  - many coding types – binary, Grey, real values, …
  - various alphabets – binary, ternary, …

with the Grey code, only one bit changes state from one position to another, i.e., if more than one bit changes, the data must be incorect

	Binary code	Grey code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110

Genetic Algorithms (GA)

- Population

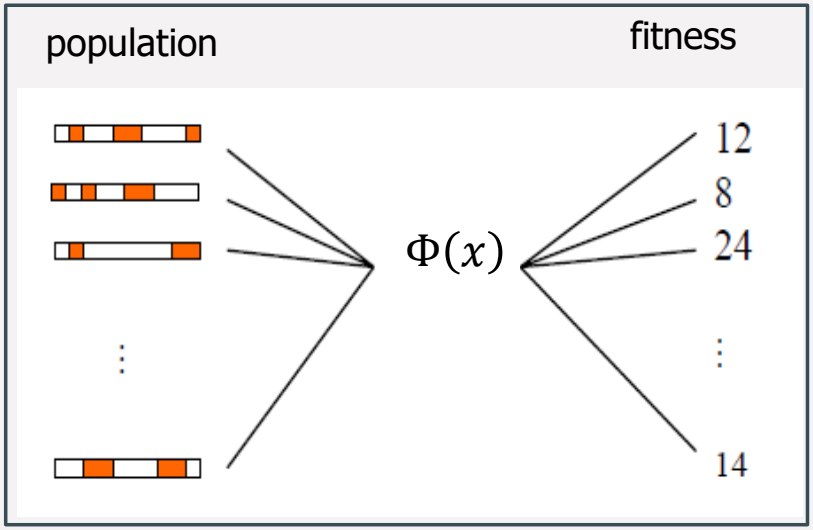
  ~ a set of artificial chromosomes that cyclically undergo selective reproduction with random changes

  - more efficient individuals are favored

- Fitness function (performance criterion)

  ~ a mapping Φ: genotype $\rightarrow \mathbb{R}$

  - better individuals get higher values



Genetic Algorithms (GA)

A general genetic algorithm (Goldberg, 1989)

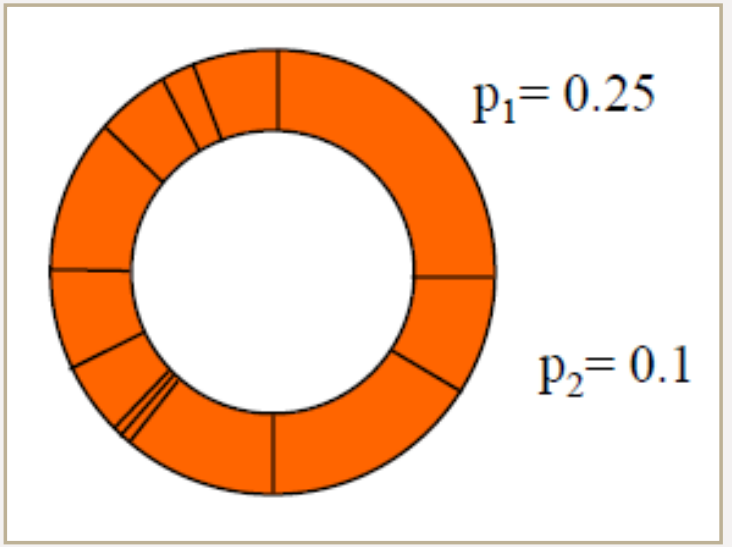
- create a population of N randomly generated chromosomes $x_1, x_2, \ldots, x_N$
- repeat
  - decode all the chromosomes and evaluate their fitness, $f_i = \Phi(x_i)$
  - form a new population by means of selective reproduction
  - recombine the chromosomes – crossover
  - mutate the chromosomes
- until the wanted individual has been found or the fitness of the so-far best individual does not improve

GA: Simple Selection

- a new population is created by copying the chromosomes from the previous population
  - the better is the copied individual, the more copies should be contained in the new population
- Roulette wheel selection:
  - the chance of an individual's being selected is proportional to its fitness:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

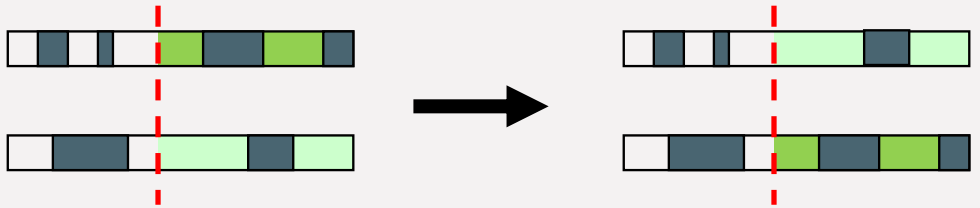
  - for a population of N individuals, the roulette wheel will be spun N-times



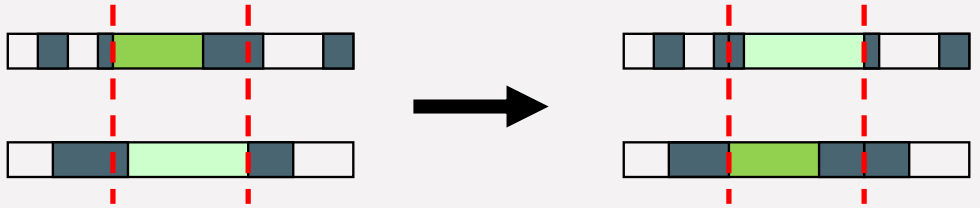
GA: Crossover

- individuals are randomly paired
- each pair of individuals will be „crossed over“ with a given probability

One-point crossover



More-point crossover



Uniform crossover a random choice of the bits from the parents

Mutation and distributed GA

- Mutation: ~ alter each element of the chromosome with a given probability
  - (binary-valued) bits will be negated
  - in the case of other ranges, a randomly chosen value can be used directly or in addition to the original value
  - probability of mutation is, in general, very low, e.g., 0.001
  - Exception: local minima (analogy to niche populations, e.g., hemophilia in European royalty, ectrodactyly in isolated tribes)

Mutation and distributed GA (2)

- Distributed genetic algorithms:
  - the individuals of a population are arranged, e.g., into a 2D- space
  - selection and crossover proceed only locally, yet individual sub-populations overlap

  ~ analogy to settler populations

==> propagation of „advantageous“ properties over entire populations

GA: Non-determinism

- The whole GA is non-deterministic and uses random variables
- Typically, we are interested in the average and maximum fitness
- GA stops after the desired fitness value has been achieved, a preset number of generations has been completed or if the fitness did not change during several past generations
- Afterwards, GA can be run again with other parameters
- The result represents the best individual selected from final generations over all the GA-runs

Neural Networks:

Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents:

GA: Schemata

- a schema is a mask describing a certain group of strings / chromosomes
  - $1{*}{*}1 = \{1001, 1011, 1101, 1111\}$; $*$ stands for any allowed value
- binary strings of length l allow for at most 3^l different schemata
- a population of N individuals represented by strings of length l yields between 2^l and $N2^l$ schemata
- a schema may define a property that guarantees high fitness
- it potentially allows to investigate more strings than actually contained within the population
- GA processes at most N^3 schemata in each step, even if it contains only N strings (an implicit parallelism in GA – Holland, 1975)

GA: Schemata (continued)

- some schemata yield a higher average fitness
- long schemata can be easily destroyed: $1*****1$ vs. $***01**$
- ***short schemata yielding a high average fitness grow exponentially during GA***
- schemata are considered to be the basic building blocks of evolution; crossover represents its main operator as it allows to investigate the combinations of schemata. In this respect, however, appropriate coding is required with short building blocks
- Note: sometimes, crossover worsens the results – in such a case, its probability shall be set to very small values or zero

GA: Schemata Theorem

- The order of the schema H, $o(H)$, is the number of fixed positions in the schema H (i.e., with values 0 or 1 for a binary alphabet), e.g.:
  - $o(011*1**) = 4$
  - $o(1******) = 1$
- The length of the schema H, $\delta(H)$, corresponds to the distance between the first and the last fixed position of H:
  - $\delta(011*1**) = 4$
  - $\delta\,(1******) = 0$
- *The Schemata Theorem analyzes the influence of reproduction, crossover and mutation on the number of strings corresponding to a given schema*

GA – Convergence Analysis: the Influence of Reproduction

The influence of reproduction on the expected number of individuals with a given schema in the population:

- At time t, the population contains m strings of the schema H ($m = m(H,t)$)
- During reproduction, a string A_i is selected to the following population (according to its fitness $f_i = \Phi(A_i)$) with the probability: $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$
- In the following population of the size n, the expected number of strings complying with the schema H ($m = m(H,t+1)$) can be estimated as:

$$m(H,t+1) = m(H,t) \cdot N \cdot \frac{f(H)}{\sum_{j=1}^{N} f_j}$$

where $f(H)$ is the average fitness of strings complying with the schema H

GA – Convergence Analysis: the Influence of Reproduction (2)

The influence of reproduction on the expected number of individuals with a given schema in the population (cont.):

- For the average fitness $f(H)$ of strings complying with the schema H at time t and for the average fitness of the entire population of n strings:

$$\bar{f} = \frac{\sum_{j=1}^{N} f_j}{N}$$

we obtain: $$m(H, t+1) = m(H, t) \cdot \frac{f(H)}{\bar{f}}$$

→ During fitness-based reproduction, the number of strings complying with a given schema grows / drops according to the average fitness

GA – Convergence Analysis: the Influence of Crossover

The influence of crossover on the expected number of individuals with a given schema in the population:

- for each schema, we can assess the probability p_s of surviving crossover as:

$$p_s = 1 - \frac{\delta(H)}{l-1}$$

- if crossover is performed randomly with probability p_c , the probability of surviving crossover corresponds to: $p_s = 1 - p_c \frac{\delta(H)}{l-1}$

→ During reproduction combined with crossover, the number of strings complying with a given schema grows / drops in the population ac-cording to the length of the schema and its average fitness:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \cdot \left[1 - p_c \frac{\delta(H)}{l-1}\right]$$

GA – Convergence Analysis: the Influence of Mutation

The influence of mutation on the expected number of individuals with a given schema in the population:

- a random change occurs at each position with probability p_m
  ⇒ each position survives mutation with probability $1 - p_m$
- all the respective mutations are mutually independent
- for a schema, each of the $o(H)$ fixed positions has to survive
  ⇒ the probability of surviving mutation is for schema H: $(1 - p_m)^{o(H)}$
- approximation for very small values of p_m ($\ll 1$):

$$(1 - p_m)^{o(h)} \sim 1 - o(H)p_m$$

GA – Convergence Analysis: the Influence of Mutation (2)

The influence of mutation on the expected number of individuals with a given schema in the population (cont.)**:**

- The expected number of strings complying with schema H in the population after reproduction, crossover, and mutation:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \cdot \left[1 - p_c \frac{\delta(H)}{l-1} - o(H) \cdot p_m\right]$$

→ During reproduction combined with crossover and mutation, the highest chance to survive possess short schemata with few fixed positions and above-average fitness

Neural Networks:

Contents:

- **Self-Organization**
- **Kohonen Maps, Hybrid Models, and Genetic Algorithms**

Contents: