Chapter 5:

Radial Basis Function Networks

Radial Basis Function Networks

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

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

(ii) $C = (U_{\text{in}} \times U_{\text{hidden}}) \cup C', \quad C' \subseteq (U_{\text{hidden}} \times U_{\text{out}})$

The network input function of each hidden neuron is a **distance function** of the input vector and the weight vector, i.e.

$$\forall u \in U_{\text{hidden}}: \qquad f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = d(\vec{w}_u, \vec{\text{in}}_u),$$

where $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+$ is a function satisfying $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^n :$

$$\begin{array}{lll} (i) & d(\vec{x}, \vec{y}) = 0 \;\Leftrightarrow\; \vec{x} = \vec{y}, & \\ (ii) & d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x}) & \text{(symmetry)}, \\ (iii) & d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) & \text{(triangle inequality)}. \end{array}$$

Radial Basis Function Networks

The network input function of the output neurons is the weighted sum of their inputs, i.e.

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

The activation function of each hidden neuron is a so-called **radial function**, i.e. a monotonously decreasing function

$$f: \mathbb{R}_0^+ \to [0,1] \quad \text{with} \quad f(0) = 1 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 0.$$

The activation function of each output neuron is a linear function, namely

$$f_{\text{act}}^{(u)}(\text{net}_u, \theta_u) = \text{net}_u - \theta_u.$$

(The linear activation function is important for the initialization.)

NEURO FUZZY

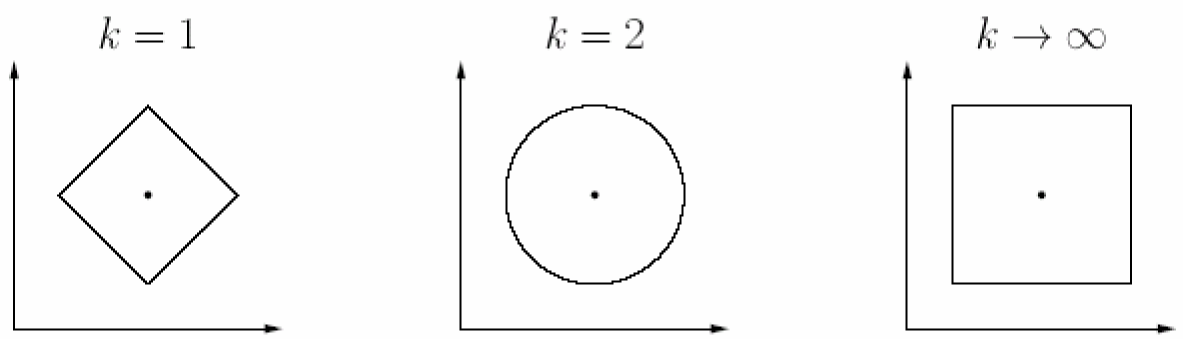
Distance Functions

Illustration of distance functions

$$d_k(\vec{x},\vec{y}) = \left(\sum_{i=1}^{n}(x_i - y_i)^k\right)^{\frac{1}{k}}$$

Well-known special cases from this family are:

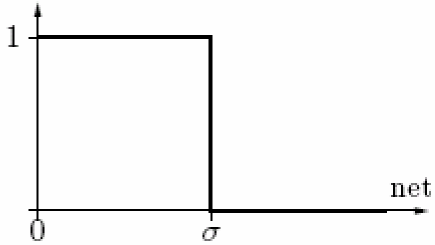
$k = 1:$	Manhattan or city block distance,
$k = 2:$	Euclidean distance,
$k \to \infty:$	maximum distance, i.e. $d_\infty(\vec{x},\vec{y}) = \max_{i=1}^{n}(x_i - y_i)$.



Radial Activation Functions

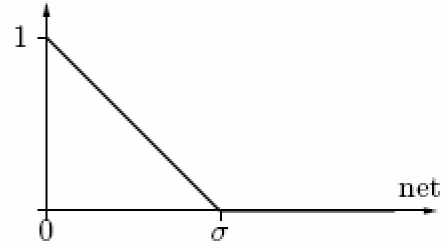
rectangle function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > \sigma, \\ 1, & \text{otherwise.} \end{cases}$$



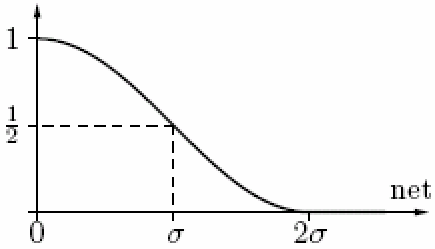
triangle function:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > \sigma, \\ 1 - \frac{\text{net}}{\sigma}, & \text{otherwise.} \end{cases}$$



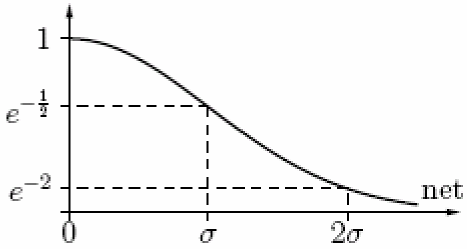
cosine until zero:

$$f_{\text{act}}(\text{net}, \sigma) = \begin{cases} 0, & \text{if net} > 2\sigma, \\ \frac{\cos(\frac{\pi}{2\sigma}\text{net})+1}{2}, & \text{otherwise.} \end{cases}$$



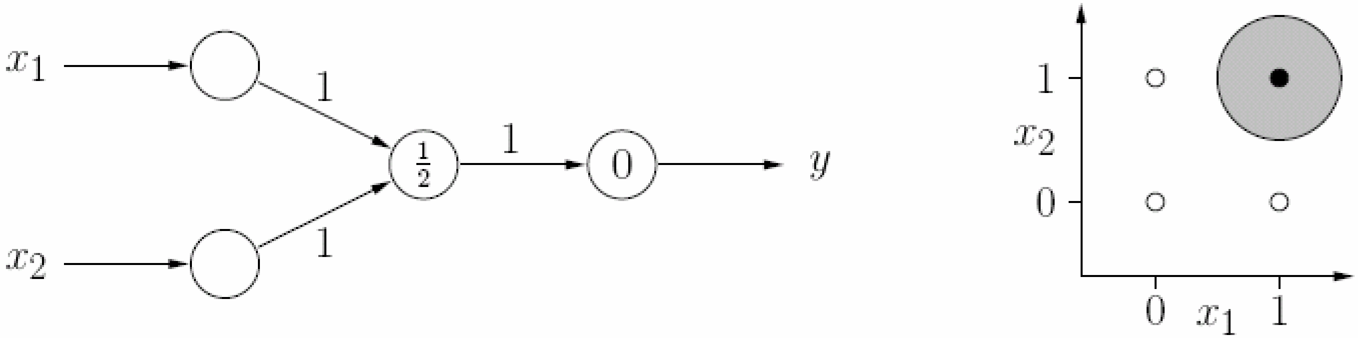
Gaussian function:

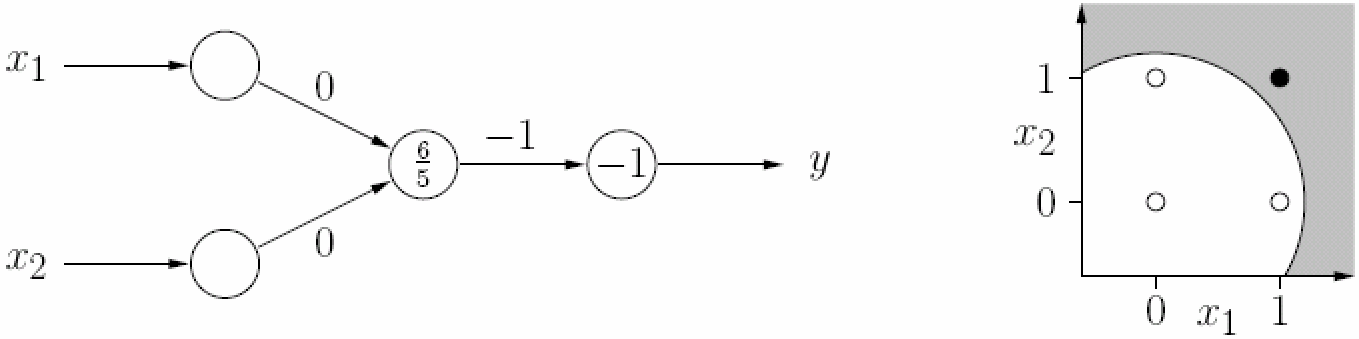
$$f_{\text{act}}(\text{net}, \sigma) = e^{-\frac{\text{net}^2}{2\sigma^2}}$$



Radial Basis Function Networks: Examples

Radial basis function networks for the conjunction $x_1 \wedge x_2$



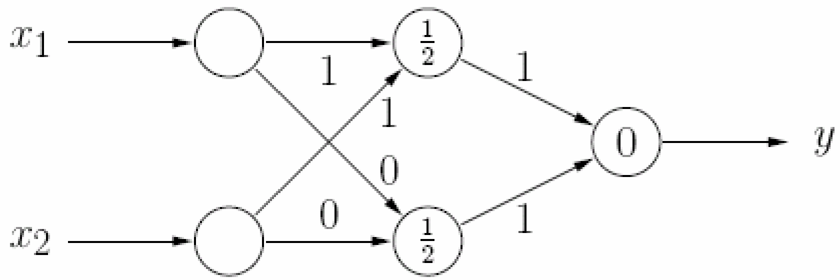


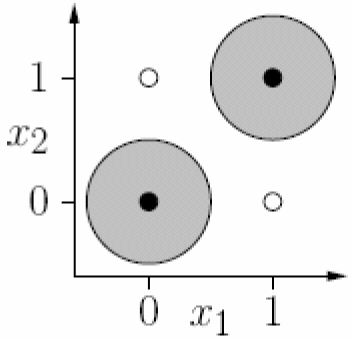
Radial Basis Function Networks: Examples

Radial basis function networks for the biimplication $x_1 \leftrightarrow x_2$

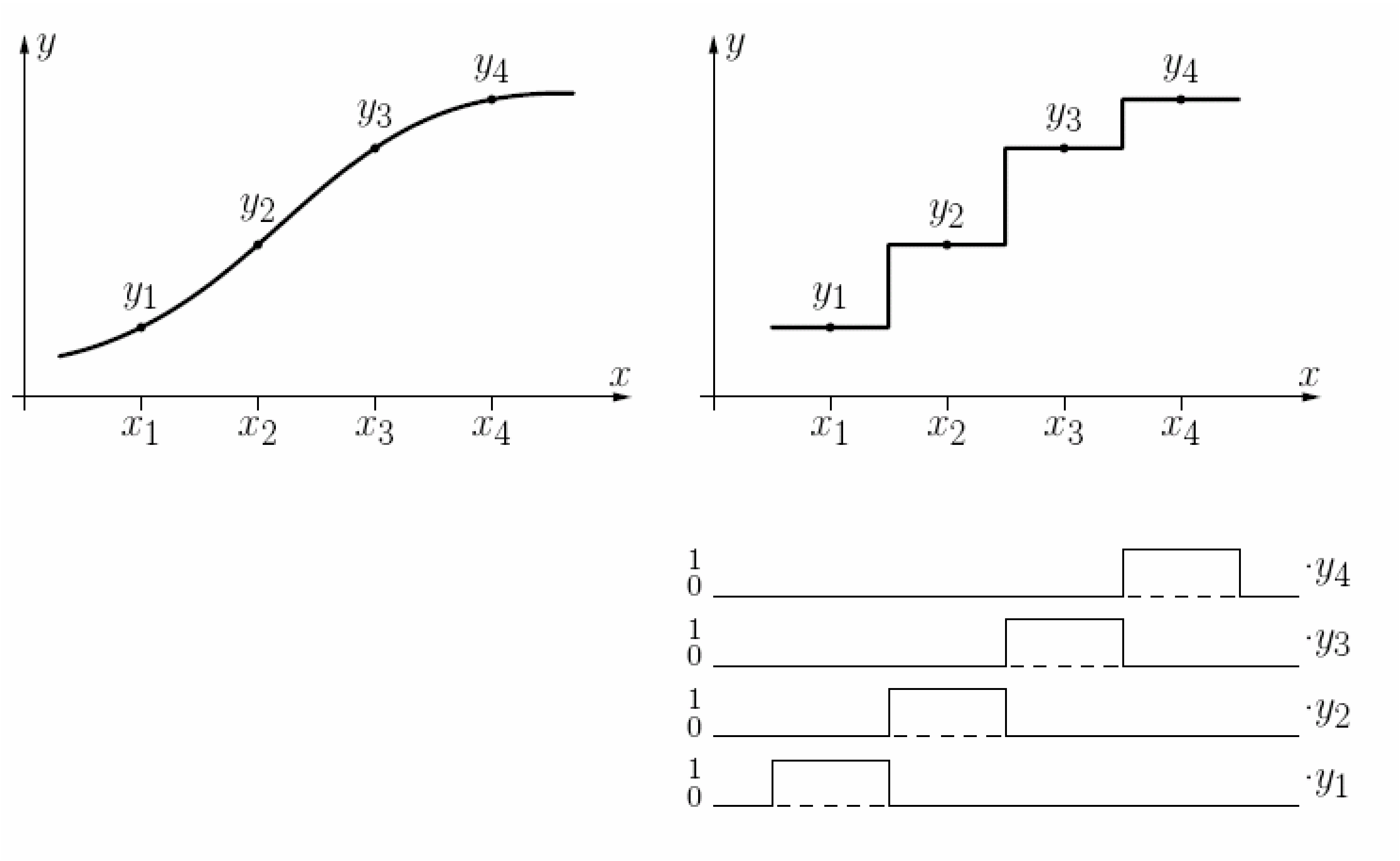
Idea: Logical decomposition

$$x_1 \leftrightarrow x_2 \quad \equiv \quad (x_1 \wedge x_2) \vee \neg(x_1 \vee x_2)$$

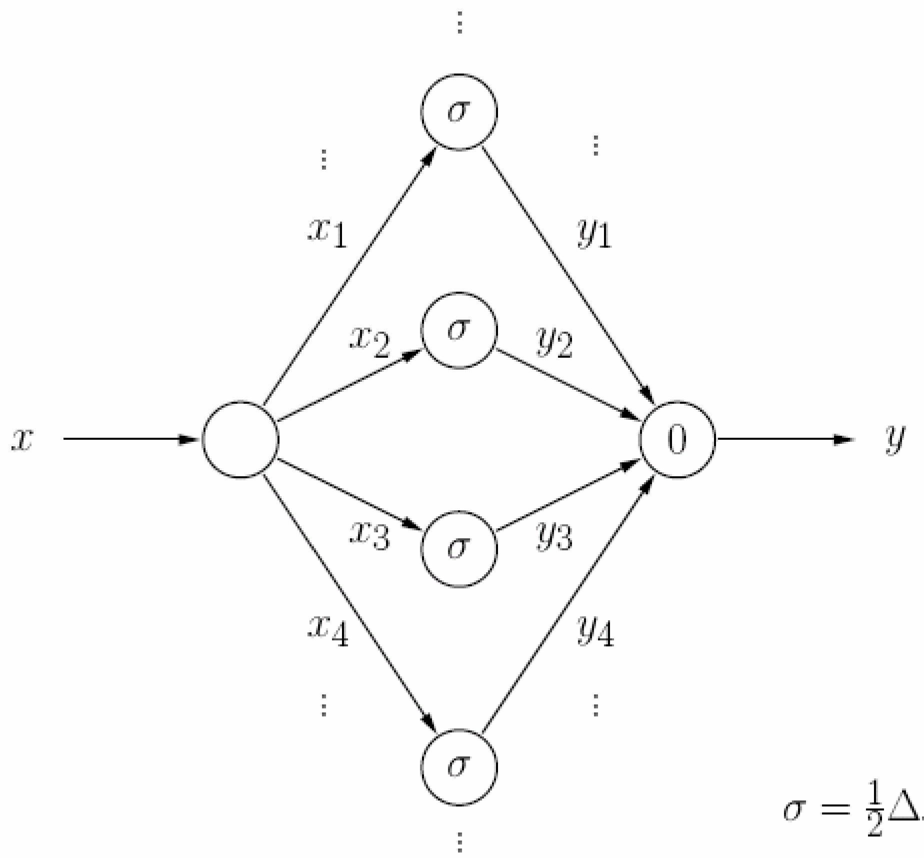




Radial Basis Function Networks: Function Approximation

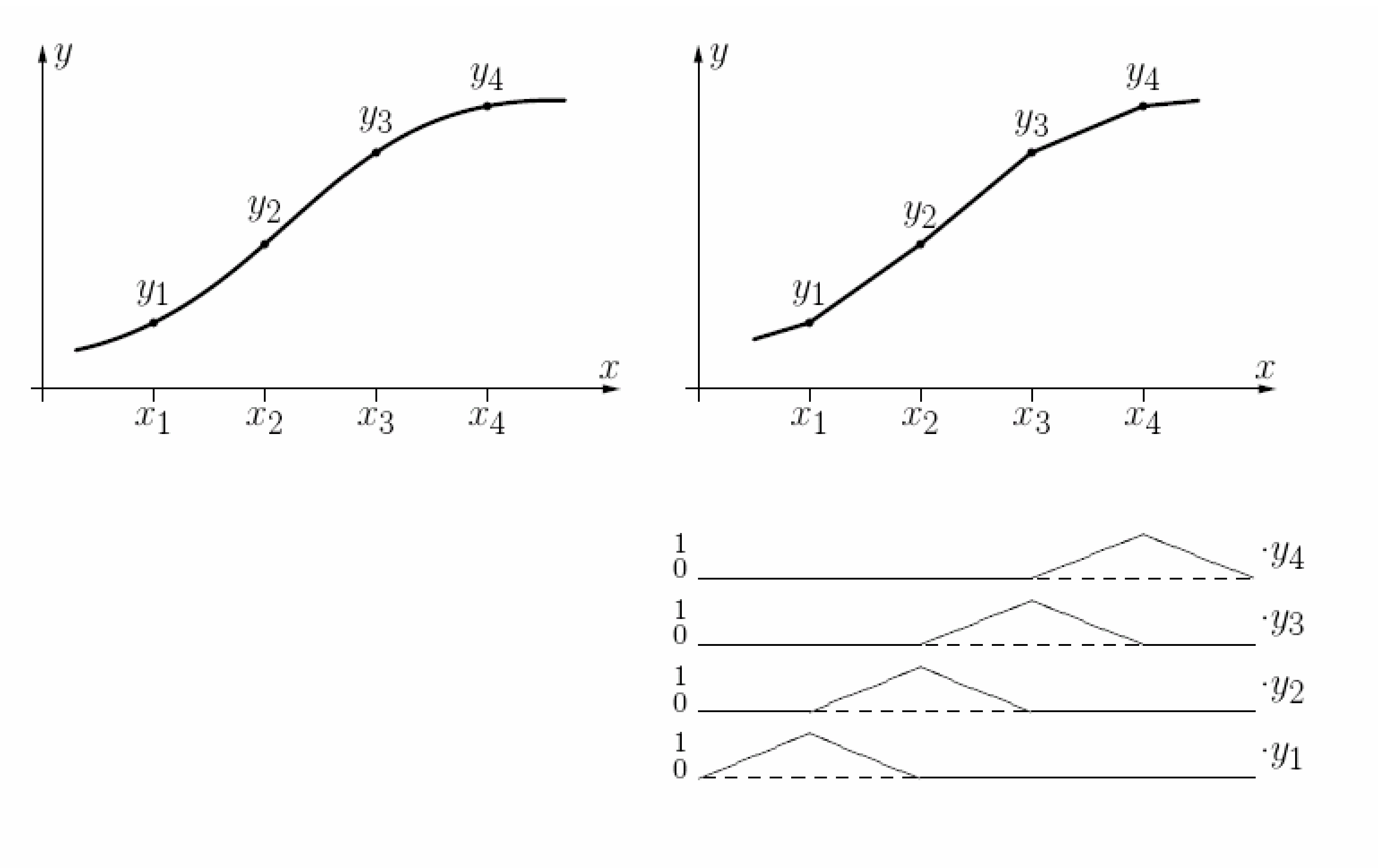


Radial Basis Function Networks: Function Approximation

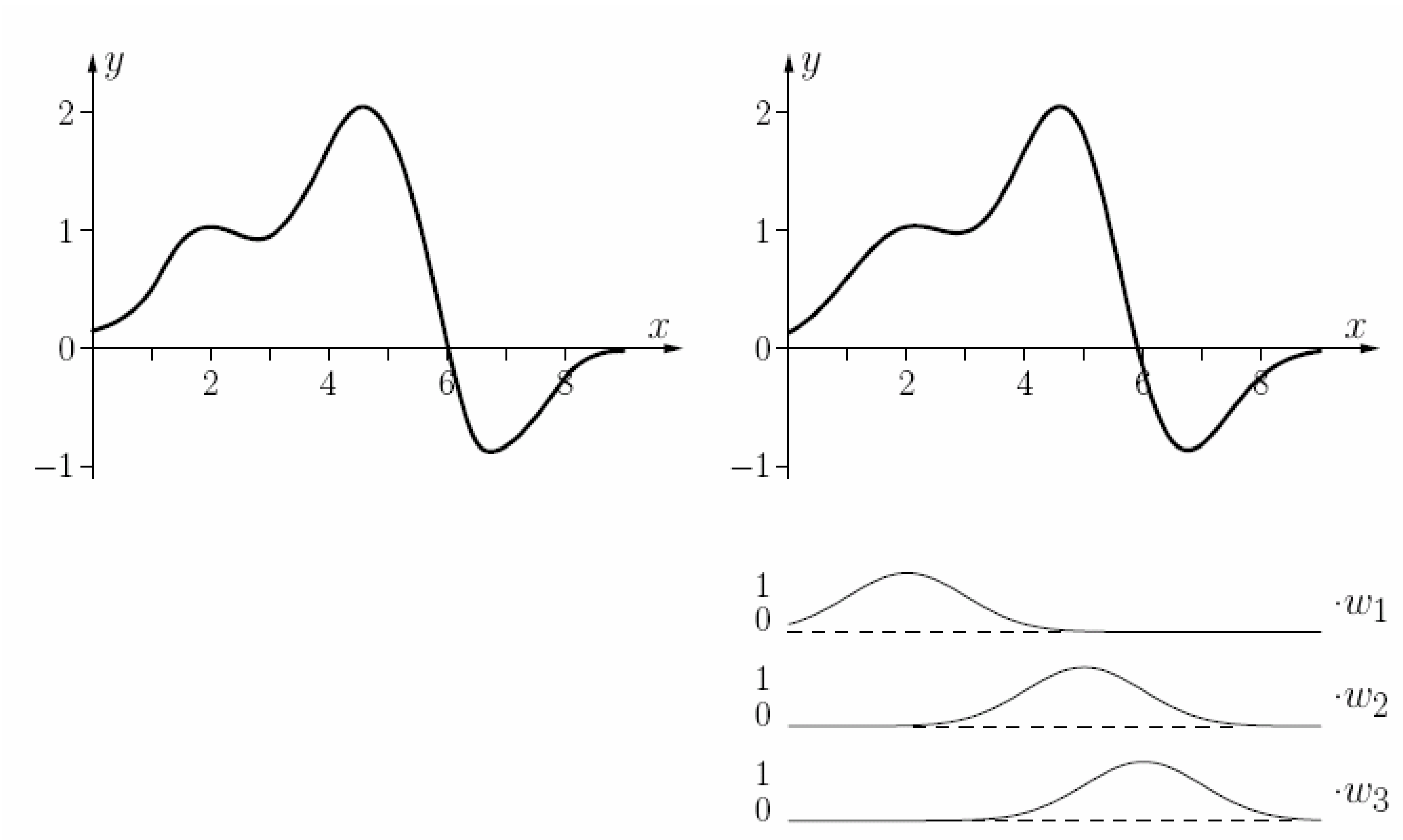


$$\sigma = \frac{1}{2}\Delta x = \frac{1}{2}(x_{i+1} - x_i)$$

Radial Basis Function Networks: Function Approximation

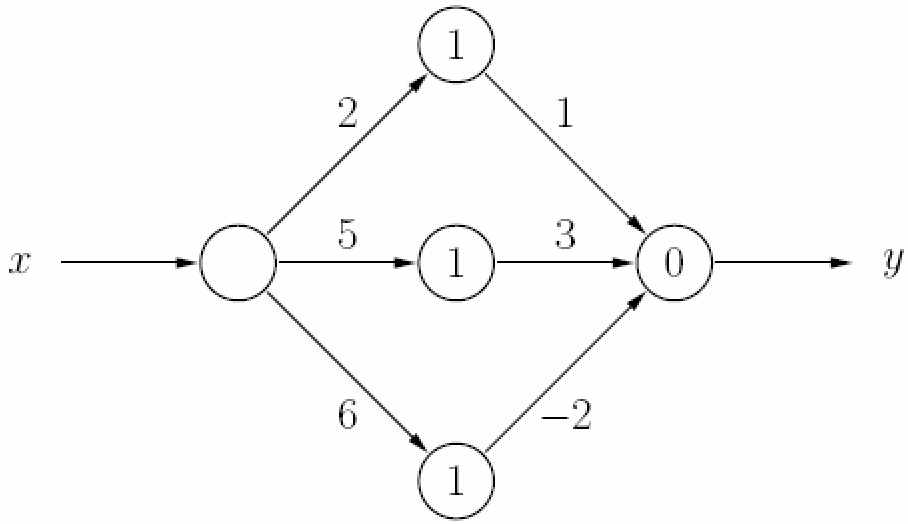


Radial Basis Function Networks: Function Approximation



Radial Basis Function Networks: Function Approximation

Radial basis function network for a sum of three Gaussian functions



Radial Basis Function Networks: Initialization

Let $L_{\text{fixed}} = \{l_1, \ldots, l_m\}$ be a fixed learning task, consisting of m training patterns $l = (\vec{\imath}^{\,(l)}, \vec{o}^{\,(l)})$.

Simple radial basis function network:
One hidden neuron v_k, $k = 1, \ldots, m$, for each training pattern:

$$\forall k \in \{1, \ldots, m\}: \qquad \vec{w}_{v_k} = \vec{\imath}^{\,(l_k)}.$$

If the activation function is the Gaussian function, the radii σ_k are chosen heuristically

$$\forall k \in \{1, \ldots, m\}: \qquad \sigma_k = \frac{d_{\max}}{\sqrt{2m}},$$

where

$$d_{\max} = \max_{l_j, l_k \in L_{\text{fixed}}} d\left(\vec{\imath}^{\,(l_j)}, \vec{\imath}^{\,(l_k)}\right).$$

Radial Basis Function Networks: Initialization

Initializing the connections from the hidden to the output neurons

$$\forall u : \sum_{k=1}^{m} w_{uv_m} \operatorname{out}_{v_m}^{(l)} - \theta_u = o_u^{(l)} \qquad \text{or abbreviated} \qquad \mathbf{A} \cdot \vec{w}_u = \vec{o}_u,$$

where $\vec{o}_u = (o_u^{(l_1)}, \ldots, o_u^{(l_m)})^T$ is the vector of desired outputs, $\theta_u = 0$, and

$$\mathbf{A} = \begin{pmatrix} \operatorname{out}_{v_1}^{(l_1)} & \operatorname{out}_{v_2}^{(l_1)} & \ldots & \operatorname{out}_{v_m}^{(l_1)} \\ \operatorname{out}_{v_1}^{(l_2)} & \operatorname{out}_{v_2}^{(l_2)} & \ldots & \operatorname{out}_{v_m}^{(l_2)} \\ \vdots & \vdots & & \vdots \\ \operatorname{out}_{v_1}^{(l_m)} & \operatorname{out}_{v_2}^{(l_m)} & \ldots & \operatorname{out}_{v_m}^{(l_m)} \end{pmatrix}.$$

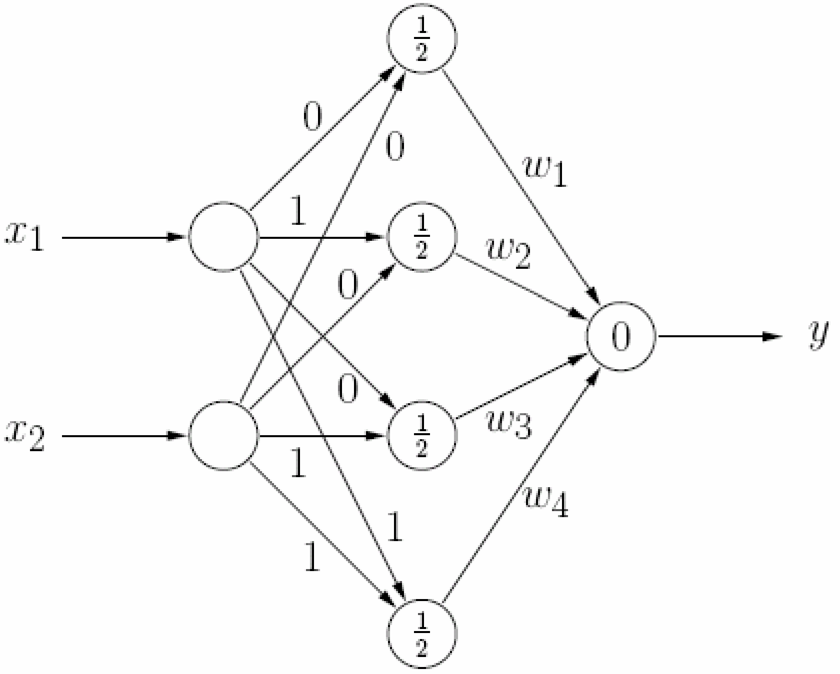
This is a linear equation system, that can be solved by inverting the matrix $\mathbf{A}$:

$$\vec{w}_u = \mathbf{A}^{-1} \cdot \vec{o}_u.$$

RBFN Initialization: Example

Simple radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



RBFN Initialization: Example

Simple radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

$$\mathbf{A} = \begin{pmatrix} 1 & e^{-2} & e^{-2} & e^{-4} \\ e^{-2} & 1 & e^{-4} & e^{-2} \\ e^{-2} & e^{-4} & 1 & e^{-2} \\ e^{-4} & e^{-2} & e^{-2} & 1 \end{pmatrix} \qquad \mathbf{A}^{-1} = \begin{pmatrix} \frac{a}{D} & \frac{b}{D} & \frac{b}{D} & \frac{c}{D} \\ \frac{b}{D} & \frac{a}{D} & \frac{c}{D} & \frac{b}{D} \\ \frac{b}{D} & \frac{c}{D} & \frac{a}{D} & \frac{b}{D} \\ \frac{c}{D} & \frac{b}{D} & \frac{b}{D} & \frac{a}{D} \end{pmatrix}$$

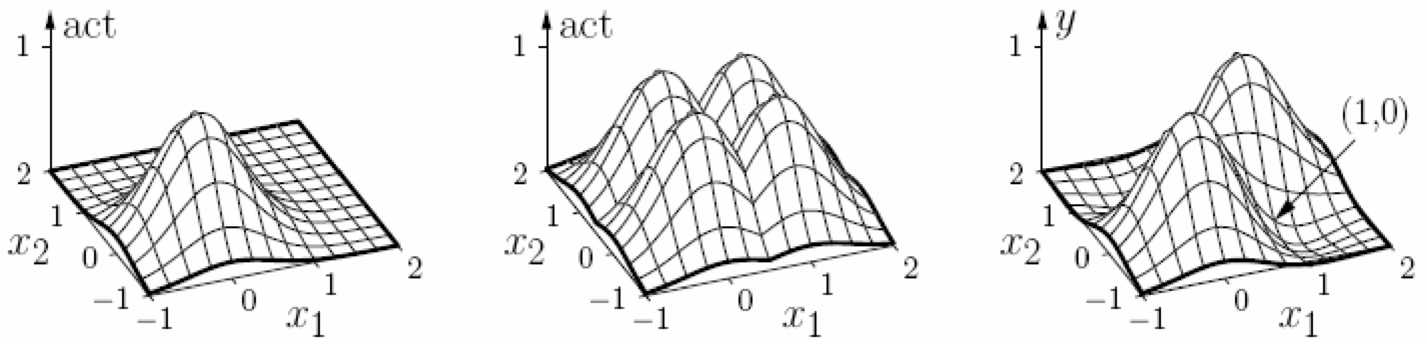
where

$$\begin{aligned} D &= 1 - 4e^{-4} + 6e^{-8} - 4e^{-12} + e^{-16} && \approx 0.9287 \\ a &= 1 - 2e^{-4} + e^{-8} && \approx 0.9637 \\ b &= -e^{-2} + 2e^{-6} - e^{-10} && \approx -0.1304 \\ c &= e^{-4} - 2e^{-8} + e^{-12} && \approx 0.0177 \end{aligned}$$

$$\vec{w}_u = \mathbf{A}^{-1} \cdot \vec{o}_u = \frac{1}{D} \begin{pmatrix} a+c \\ 2b \\ 2b \\ a+c \end{pmatrix} \approx \begin{pmatrix} 1.0567 \\ -0.2809 \\ -0.2809 \\ 1.0567 \end{pmatrix}$$

RBFN Initialization: Example

Simple radial basis function network for the biimplication $x_1 \leftrightarrow x_2$



single basis function — all basis functions — output

- Initialization leads already to a perfect solution of the learning task.
- Subsequent training is not necessary.

Radial Basis Function Networks: Initialization

Normal radial basis function networks:
Select subset of k training patterns as centers.

$$\mathbf{A} = \begin{pmatrix} 1 & \mathrm{out}_{v_1}^{(l_1)} & \mathrm{out}_{v_2}^{(l_1)} & \dots & \mathrm{out}_{v_k}^{(l_1)} \\ 1 & \mathrm{out}_{v_1}^{(l_2)} & \mathrm{out}_{v_2}^{(l_2)} & \dots & \mathrm{out}_{v_k}^{(l_2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \mathrm{out}_{v_1}^{(l_m)} & \mathrm{out}_{v_2}^{(l_m)} & \dots & \mathrm{out}_{v_k}^{(l_m)} \end{pmatrix} \qquad \mathbf{A} \cdot \vec{w}_u = \vec{o}_u$$

Compute (Moore–Penrose) pseudo inverse:

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T.$$

The weights can then be computed by

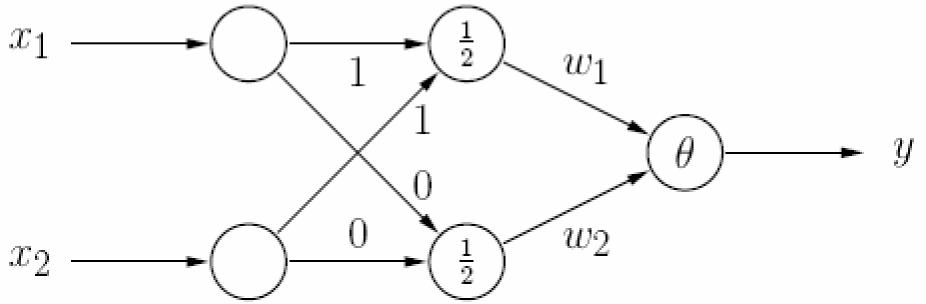
$$\vec{w}_u = \mathbf{A}^+ \cdot \vec{o}_u = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \cdot \vec{o}_u$$

RBFN Initialization: Example

Normal radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

Select two training patterns:

- $l_1 = (\vec{\imath}^{\,(l_1)}, \vec{o}^{\,(l_1)}) = ((0, 0), (1))$
- $l_4 = (\vec{\imath}^{\,(l_4)}, \vec{o}^{\,(l_4)}) = ((1, 1), (1))$



RBFN Initialization: Example

Normal radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & e^{-4} \\ 1 & e^{-2} & e^{-2} \\ 1 & e^{-2} & e^{-2} \\ 1 & e^{-4} & 1 \end{pmatrix} \qquad \mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \begin{pmatrix} a & b & b & a \\ c & d & d & e \\ e & d & d & c \end{pmatrix}$$

where

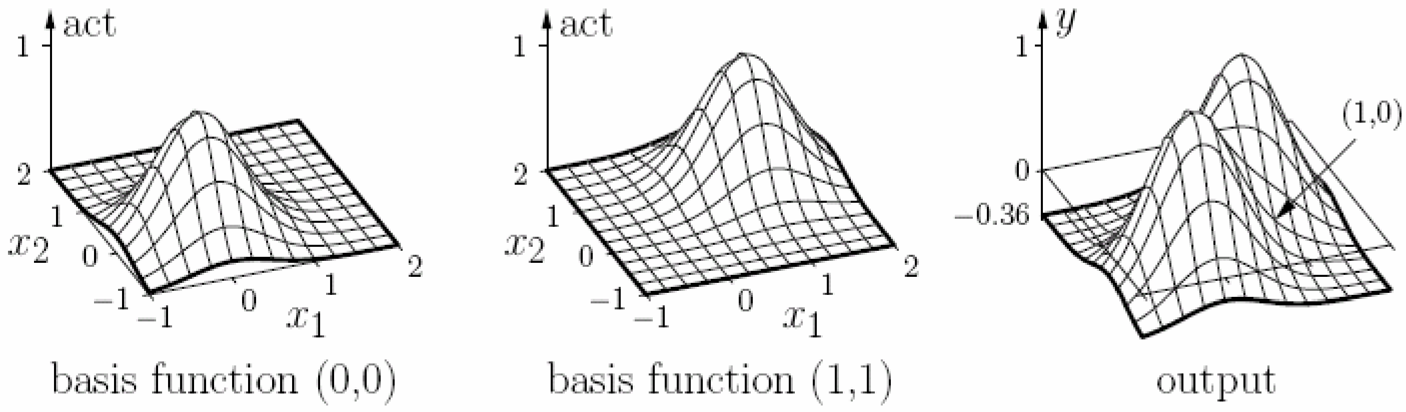
$$a \approx -0.1810, \qquad b \approx 0.6810,$$
$$c \approx 1.1781, \qquad d \approx -0.6688, \qquad e \approx 0.1594.$$

Resulting weights:

$$\vec{w}_u = \begin{pmatrix} -\theta \\ w_1 \\ w_2 \end{pmatrix} = \mathbf{A}^+ \cdot \vec{o}_u \approx \begin{pmatrix} -0.3620 \\ 1.3375 \\ 1.3375 \end{pmatrix}.$$

RBFN Initialization: Example

Normal radial basis function network for the biimplication $x_1 \leftrightarrow x_2$



- Initialization leads already to a perfect solution of the learning task.
- This is an accident, because the linear equation system is not over-determined, due to linearly dependent equations.

Radial Basis Function Networks: Initialization

Finding appropriate centers for the radial basis functions

One approach: **k-means clustering**

- Select randomly k training patterns as centers.
- Assign to each center those training patterns that are closest to it.
- Compute new centers as the center of gravity of the assigned training patterns
- Repeat previous two steps until convergence,
  i.e., until the centers do not change anymore.
- Use resulting centers for the weight vectors of the hidden neurons.

Alternative approach: **learning vector quantization**

Radial Basis Function Networks: Training

Training radial basis function networks:
Derivation of update rules is analogous to that of multilayer perceptrons.

Weights from the hidden to the output neurons.

Gradient:

$$\vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \frac{\partial e_u^{(l)}}{\partial \vec{w}_u} = -2(o_u^{(l)} - \mathrm{out}_u^{(l)}) \, \vec{\mathrm{in}}_u^{(l)},$$

Weight update rule:

$$\Delta \vec{w}_u^{(l)} = -\frac{\eta_3}{2} \vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \eta_3 (o_u^{(l)} - \mathrm{out}_u^{(l)}) \, \vec{\mathrm{in}}_u^{(l)}$$

(Two more learning rates are needed for the center coordinates and the radii.)

Radial Basis Function Networks: Training

Training radial basis function networks:
Center coordinates (weights from the input to the hidden neurons).

Gradient:

$$\vec{\nabla}_{\vec{w}_v} e^{(l)} = \frac{\partial e^{(l)}}{\partial \vec{w}_v} = -2 \sum_{s \in \text{succ}(v)} (o_s^{(l)} - \text{out}_s^{(l)}) w_{su} \frac{\partial \, \text{out}_v^{(l)}}{\partial \, \text{net}_v^{(l)}} \frac{\partial \, \text{net}_v^{(l)}}{\partial \vec{w}_v}$$

Weight update rule:

$$\Delta \vec{w}_v^{(l)} = -\frac{\eta_1}{2} \vec{\nabla}_{\vec{w}_v} e^{(l)} = \eta_1 \sum_{s \in \text{succ}(v)} (o_s^{(l)} - \text{out}_s^{(l)}) w_{sv} \frac{\partial \, \text{out}_v^{(l)}}{\partial \, \text{net}_v^{(l)}} \frac{\partial \, \text{net}_v^{(l)}}{\partial \vec{w}_v}$$

Radial Basis Function Networks: Training

Training radial basis function networks:
Center coordinates (weights from the input to the hidden neurons).

Special case: **Euclidean distance**

$$\frac{\partial\, \mathrm{net}_v^{(l)}}{\partial \vec{w}_v} = \left(\sum_{i=1}^{n} (w_{vp_i} - \mathrm{out}_{p_i}^{(l)})^2 \right)^{-\frac{1}{2}} (\vec{w}_v - \vec{\mathrm{in}}_v^{(l)}).$$

Special case: **Gaussian activation function**

$$\frac{\partial\, \mathrm{out}_v^{(l)}}{\partial\, \mathrm{net}_v^{(l)}} = \frac{\partial f_{\mathrm{act}}(\,\mathrm{net}_v^{(l)}, \sigma_v)}{\partial\, \mathrm{net}_v^{(l)}} = \frac{\partial}{\partial\, \mathrm{net}_v^{(l)}}\, e^{-\frac{\left(\mathrm{net}_v^{(l)}\right)^2}{2\sigma_v^2}} = -\frac{\mathrm{net}_v^{(l)}}{\sigma_v^2}\, e^{-\frac{\left(\mathrm{net}_v^{(l)}\right)^2}{2\sigma_v^2}}.$$

Radial Basis Function Networks: Training

Training radial basis function networks:
Radii of radial basis functions.

Gradient:

$$\frac{\partial e^{(l)}}{\partial \sigma_v} = -2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{su} \frac{\partial \, \mathrm{out}_v^{(l)}}{\partial \sigma_v}.$$

Weight update rule:

$$\Delta \sigma_v^{(l)} = -\frac{\eta_2}{2} \frac{\partial e^{(l)}}{\partial \sigma_v} = \eta_2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{sv} \frac{\partial \, \mathrm{out}_v^{(l)}}{\partial \sigma_v}.$$

Special case: **Gaussian activation function**

$$\frac{\partial \, \mathrm{out}_v^{(l)}}{\partial \sigma_v} = \frac{\partial}{\partial \sigma_v} e^{-\frac{\left(\mathrm{net}_v^{(l)}\right)^2}{2\sigma_v^2}} = \frac{\left(\mathrm{net}_v^{(l)}\right)^2}{\sigma_v^3} e^{-\frac{\left(\mathrm{net}_v^{(l)}\right)^2}{2\sigma_v^2}}.$$

Radial Basis Function Networks: Generalization

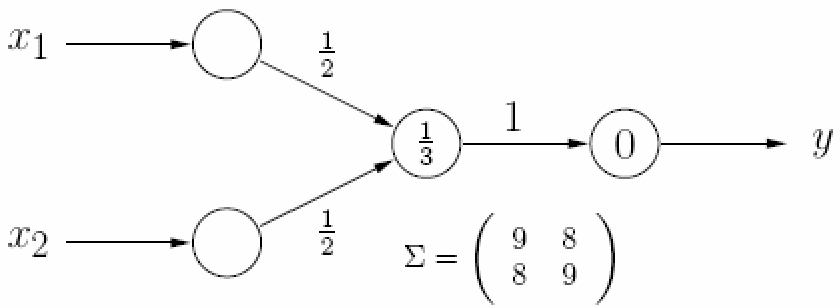
Generalization of the distance function

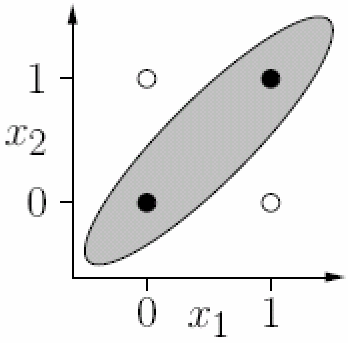
Idea: Use anisotropic distance function.

Example: **Mahalanobis distance**

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})}.$$

Example: **biimplication**





Interpretation of a Covariance Matrix

- A univariate normal distribution has the density function

$$f_X(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

μ:	expected value
σ^2:	variance
σ:	standard deviation

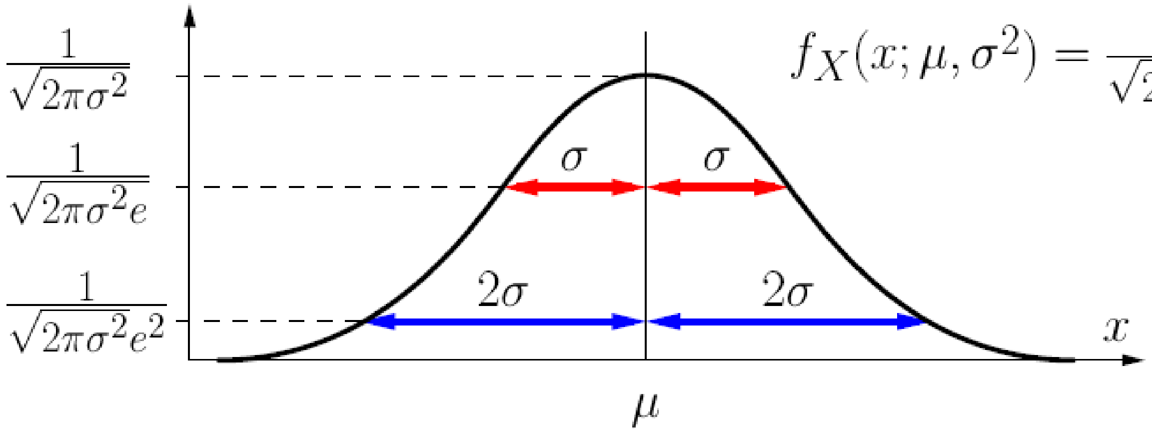
- A multivariate normal distribution has the density function

$$f_{\vec{X}}(\vec{x};\vec{\mu},\mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^m|\mathbf{\Sigma}|}} \cdot \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^\top \mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\right)$$

m:	size of the vector $\vec{x}$ (it is m-dimensional)
$\vec{\mu}$:	mean value vector (m-dimensional)
$\mathbf{\Sigma}$:	covariance matrix ($m \times m$ matrix)
$\lvert\mathbf{\Sigma}\rvert$:	determinant of the covariance matrix $\mathbf{\Sigma}$

Variance and Standard Deviation

- **Univariate Normal/Gaussian Distribution**
  The variance/standard deviation provides information about the height of the mode and the width of the curve.

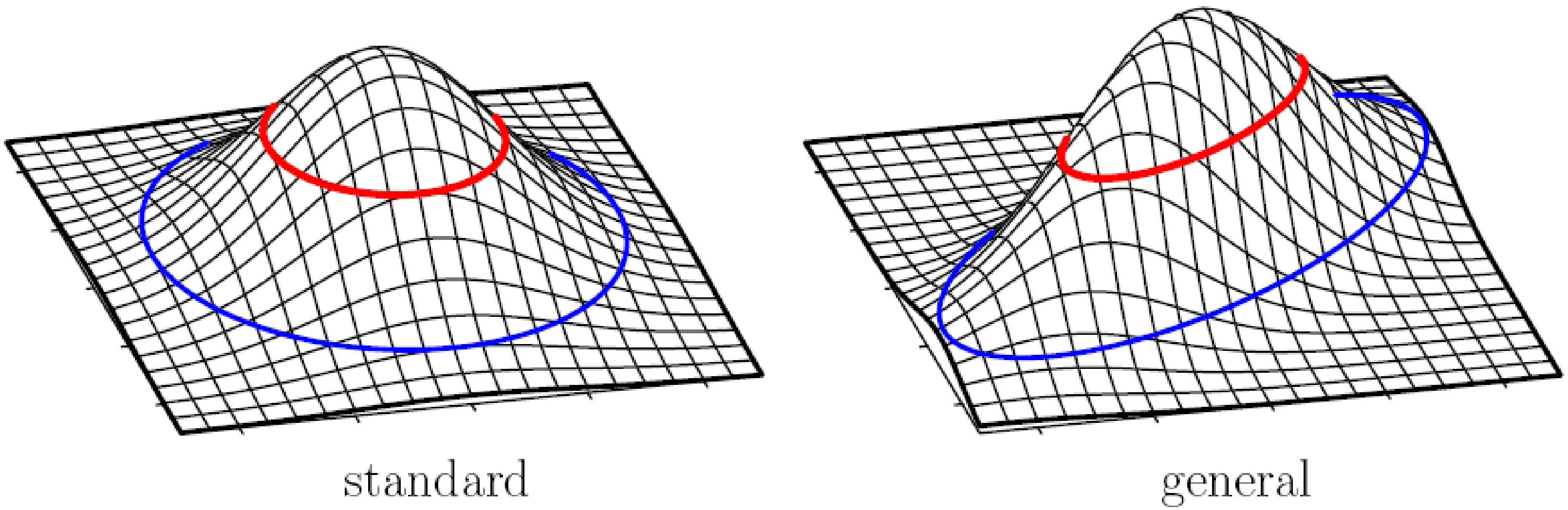


$$f_X(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- μ: expected value,
  σ^2: variance,
  σ: standard deviation,
  Important: standard deviation has same unit as expected value.

Interpretation of a Covariance Matrix

- The variance/standard deviation relates the spread of the distribution to the spread of a **standard normal distribution** $(\sigma^2 = \sigma = 1)$.
- The covariance matrix relates the spread of the distribution to the spread of a **multivariate standard normal distribution** $(\boldsymbol{\Sigma} = \mathbf{1})$.
- Example: bivariate normal distribution



- **Question:** Is there a multivariate analog of standard deviation?

Eigenvalue Decomposition

- Yields an analog of standard deviation.
- Let **S** be a symmetric, positive definite matrix (e.g. a covariance matrix).
  - **S** can be written as
    $$\mathbf{S} = \mathbf{R}\ \operatorname{diag}(\lambda_1, \ldots, \lambda_m)\ \mathbf{R}^{-1},$$
    where the λ_j, $j = 1, \ldots, m$, are the eigenvalues of $\mathbf{S}$ and the columns of $\mathbf{R}$ are the (normalized) eigenvectors of $\mathbf{S}$.
  - The eigenvalues λ_j, $j = 1, \ldots, m$, of $\mathbf{S}$ are all positive and the eigenvectors of $\mathbf{S}$ are orthonormal ($\rightarrow \mathbf{R}^{-1} = \mathbf{R}^\top$).
- Due to the above, $\mathbf{S}$ can be written as $\mathbf{S} = \mathbf{T}\,\mathbf{T}^\top$, where
  $$\mathbf{T} = \mathbf{R}\ \operatorname{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)$$

Eigenvalue Decomposition

Special Case: Two Dimensions

- Covariance matrix

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

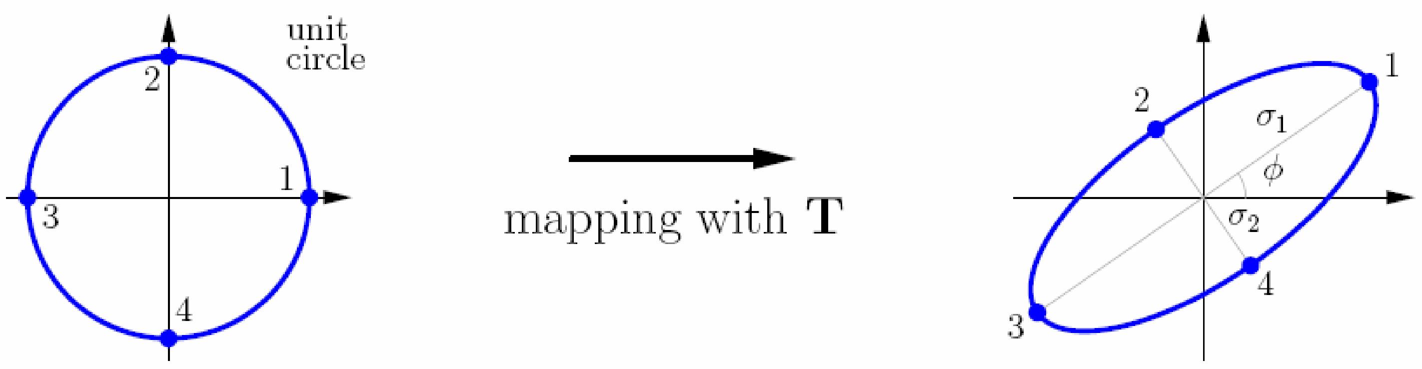
- Eigenvalue decomposition

$$\mathbf{T} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix},$$

$$s = \sin\phi, c = \cos\phi, \phi = \frac{1}{2}\arctan\frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2},$$

$$\sigma_1 = \sqrt{c^2\sigma_x^2 + s^2\sigma_y^2 + 2sc\sigma_{xy}},$$

$$\sigma_2 = \sqrt{s^2\sigma_x^2 + c^2\sigma_y^2 - 2sc\sigma_{xy}}.$$



Eigenvalue Decomposition

Eigenvalue decomposition enables us to write a covariance matrix $\mathbf{\Sigma}$ as

$$\mathbf{\Sigma} = \mathbf{T}\mathbf{T}^\top \qquad \text{with} \qquad \mathbf{T} = \mathbf{R}\,\text{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right).$$

As a consequence we can write its inverse $\mathbf{\Sigma}^{-1}$ as

$$\mathbf{\Sigma}^{-1} = \mathbf{U}^\top\mathbf{U} \qquad \text{with} \qquad \mathbf{U} = \text{diag}\left(\lambda_1^{-\frac{1}{2}}, \ldots, \lambda_m^{-\frac{1}{2}}\right)\mathbf{R}^\top.$$

$\mathbf{U}$ describes the inverse mapping of $\mathbf{T}$, i.e., rotates the ellipse so that its axes coincide with the coordinate axes and then scales the axes to unit length. Hence:

$$(\vec{x} - \vec{y})^\top\mathbf{\Sigma}^{-1}(\vec{x} - \vec{y}) \;=\; (\vec{x} - \vec{y})^\top\mathbf{U}^\top\mathbf{U}(\vec{x} - \vec{y}) \;=\; (\vec{x}\,' - \vec{y}\,')^\top(\vec{x}\,' - \vec{y}\,'),$$

where $\quad \vec{x}\,' = \mathbf{U}\vec{x} \quad$ and $\quad \vec{y}\,' = \mathbf{U}\vec{y}$.

Result: $(\vec{x} - \vec{y})^\top\mathbf{\Sigma}^{-1}(\vec{x} - \vec{y})$ is equivalent to the squared **Euclidean distance** in the properly scaled eigensystem of the covariance matrix $\mathbf{\Sigma}$.

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^\top\mathbf{\Sigma}^{-1}(\vec{x} - \vec{y})} \qquad \text{is called } \textbf{Mahalanobis distance}.$$

Eigenvalue Decomposition

Eigenvector decomposition also shows that the determinant of the covariance matrix $\mathbf{\Sigma}$ provides a measure of the (hyper-)volume of the (hyper-)ellipsoid. It is

$$|\mathbf{\Sigma}| = |\mathbf{R}|\ |\mathrm{diag}(\lambda_1, \ldots, \lambda_m)|\ |\mathbf{R}^\top| = |\mathrm{diag}(\lambda_1, \ldots, \lambda_m)| = \prod_{i=1}^{m} \lambda_i,$$

since $|\mathbf{R}| = |\mathbf{R}^\top| = 1$ as $\mathbf{R}$ is orthogonal with unit length columns, and thus

$$\sqrt{|\mathbf{\Sigma}|} = \prod_{i=1}^{m} \sqrt{\lambda_i},$$

which is proportional to the (hyper-)volume of the (hyper-)ellipsoid.

To be precise, the volume of the m-dimensional (hyper-)ellipsoid a (hyper-)sphere with radius r is mapped to with a covariance matrix $\mathbf{\Sigma}$ is

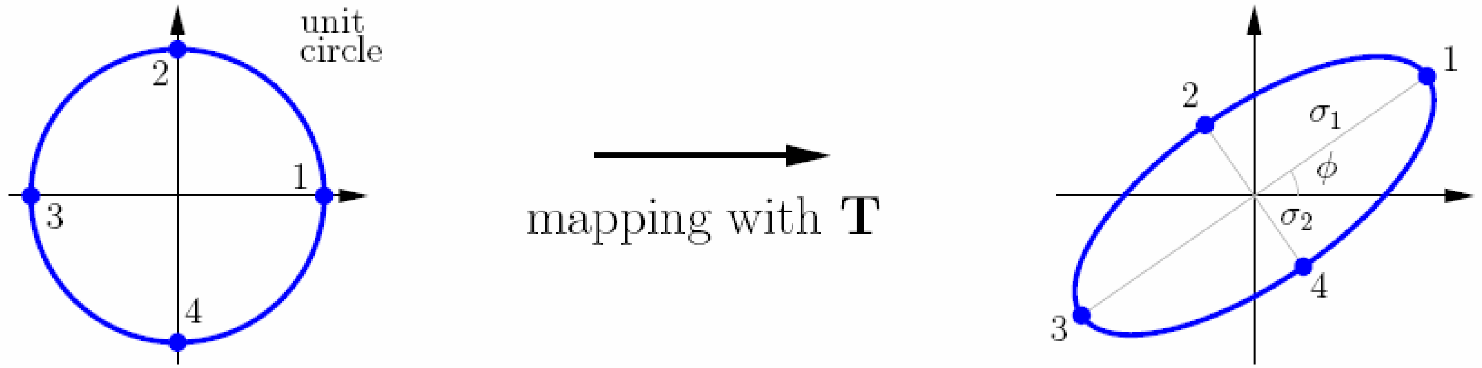
$$V_m(r) = \frac{\pi^{\frac{m}{2}} r^m}{\Gamma\left(\frac{m}{2} + 1\right)} \sqrt{|\mathbf{\Sigma}|}, \quad \text{where} \quad \begin{array}{l} \Gamma(x) = \int_0^\infty e^{-t} t^{x-1}\, \mathrm{d}t, \quad x > 0, \\ \Gamma(x+1) = x \cdot \Gamma(x), \quad \Gamma(\frac{1}{2}) = \sqrt{\pi}, \ \ \Gamma(1) = 1. \end{array}$$

Eigenvalue Decomposition

Special Case: Two Dimensions

- Covariance matrix and its eigenvalue decomposition:

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \qquad \text{and} \qquad \mathbf{T} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}.$$



- The area of the ellipse, to which the unit circle (area π) is mapped, is

$$A = \pi\sigma_1\sigma_2 = \pi\sqrt{|\mathbf{\Sigma}|}.$$

Cluster-Specific Distance Functions

The similarity of a data point to a prototype depends on their distance.

- If the cluster prototype is a simple cluster center, a general distance measure can be defined on the data space.

  In this case the **Euclidean distance** is most often used due to its rotation invariance. It leads to (hyper-)spherical clusters.

- However, more flexible clustering approaches (with size and shape parameters) use **cluster-specific distance functions**.

  The most common approach is to use a **Mahalanobis distance** with a **cluster-specific covariance matrix.**

$$d(\vec{x}, \vec{y}; \mathbf{\Sigma}) = \sqrt{(\vec{x} - \vec{y})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{y})}.$$

The covariance matrix comprises **shape** and **size** parameters.

The Euclidean distance is a special case that results for $\mathbf{\Sigma} = \mathbf{1}$.

Chapter 6:

Self-Organizing Maps

Self-Organizing Maps

A **self-organizing map** or **Kohonen feature map** is a neural network with a graph $G = (U, C)$ that satisfies the following conditions

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

(ii) $C = U_{\text{in}} \times U_{\text{out}}$.

The network input function of each output neuron is a **distance function** of input and weight vector. The activation function of each output neuron is a **radial function**, i.e. a monotonously decreasing function

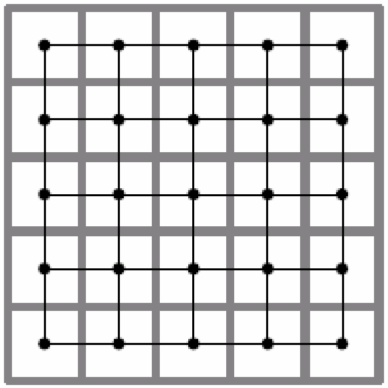
$$f : \mathbb{R}_0^+ \to [0, 1] \quad \text{with} \quad f(0) = 1 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 0.$$

The output function of each output neuron is the identity.
The output is often discretized according to the "**winner takes all**" principle.
On the output neurons a **neighborhood relationship** is defined:

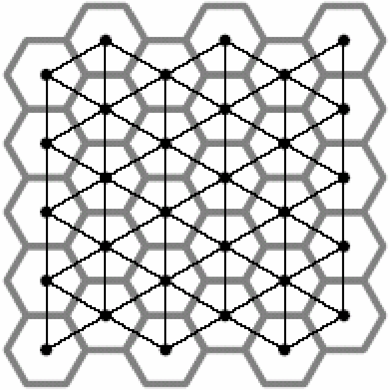
$$d_{\text{neurons}} : U_{\text{out}} \times U_{\text{out}} \to \mathbb{R}_0^+ .$$

Self-Organizing Maps: Neighborhood

Neighborhood of the output neurons: neurons form a grid



quadratic grid

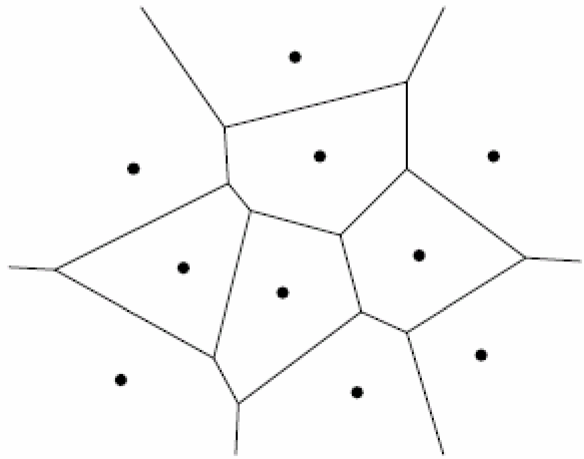


hexagonal grid

- Thin black lines: Indicate nearest neighbors of a neuron.
- Thick gray lines: Indicate regions assigned to a neuron for visualization.

Vector Quantization

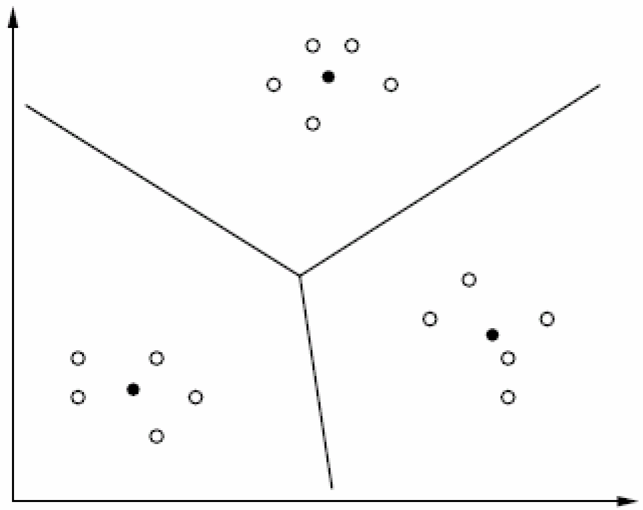
Voronoi diagram of a vector quantization



- Dots represent vectors that are used for quantizing the area.
- Lines are the boundaries of the regions of points that are closest to the enclosed vector.

Learning Vector Quantization

Finding clusters in a given set of data points



- Data points are represented by empty circles (∘).
- Cluster centers are represented by full circles (•).

Learning Vector Quantization

Adaptation of reference vectors / codebook vectors

- For each training pattern find the closest reference vector.
- Adapt only this reference vector (winner neuron).
- For classified data the class may be taken into account. (reference vectors are assigned to classes)

Attraction rule (data point and reference vector have same class)

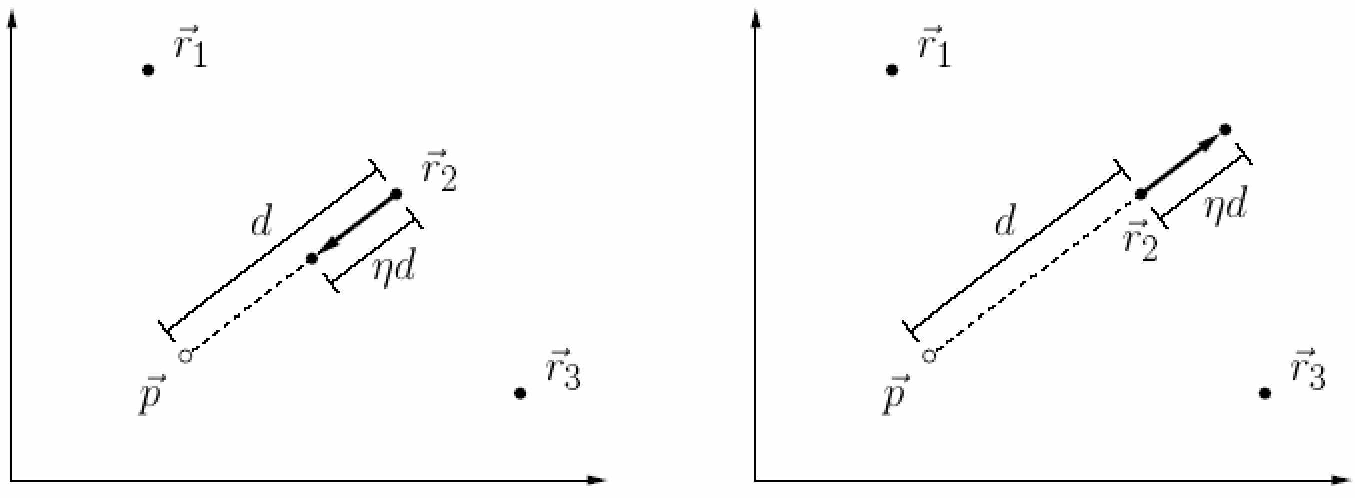
$$\vec{r}^{\,(\text{new})} = \vec{r}^{\,(\text{old})} + \eta(\vec{p} - \vec{r}^{\,(\text{old})}),$$

Repulsion rule (data point and reference vector have different class)

$$\vec{r}^{\,(\text{new})} = \vec{r}^{\,(\text{old})} - \eta(\vec{p} - \vec{r}^{\,(\text{old})}).$$

Learning Vector Quantization

Adaptation of reference vectors / codebook vectors

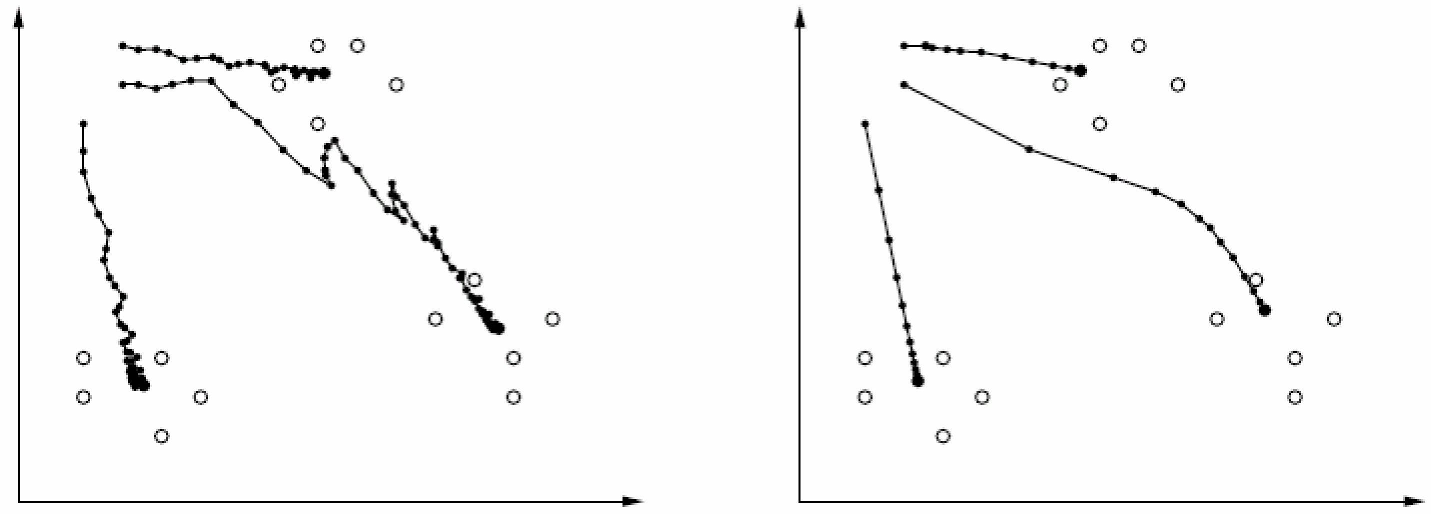


attraction rule | repulsion rule

- $\vec{p}$: data point, $\vec{r}_i$: reference vector
- $\eta = 0.4$ (learning rate)

Learning Vector Quantization: Example

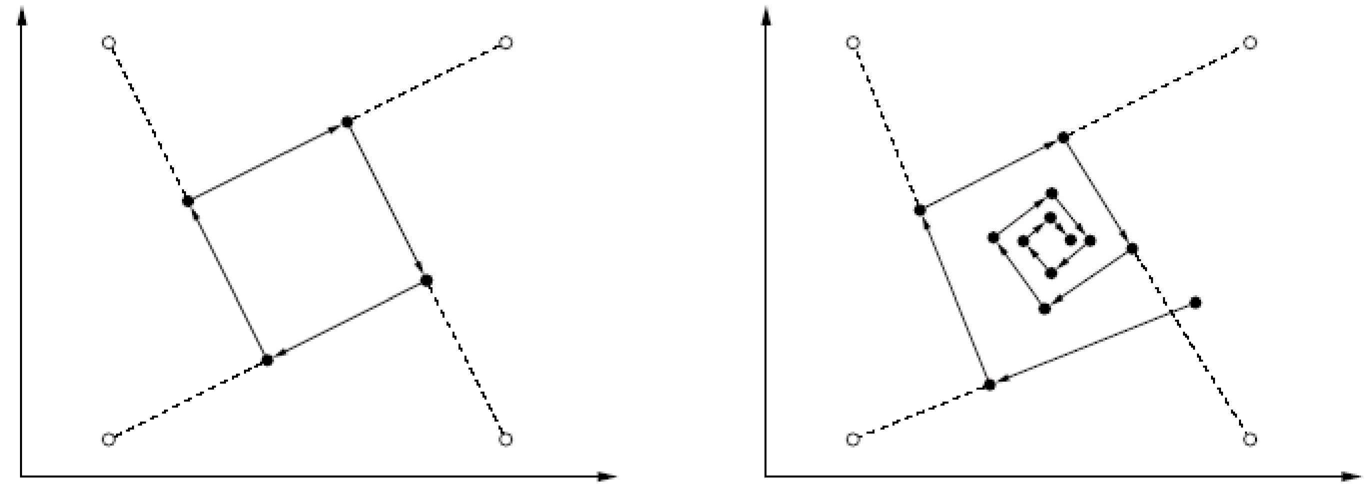
Adaptation of reference vectors / codebook vectors



- Left: Online training with learning rate $\eta = 0.1$,
- Right: Batch training with learning rate $\eta = 0.05$.

Learning Vector Quantization: Learning Rate Decay

Problem: fixed learning rate can lead to oscillations



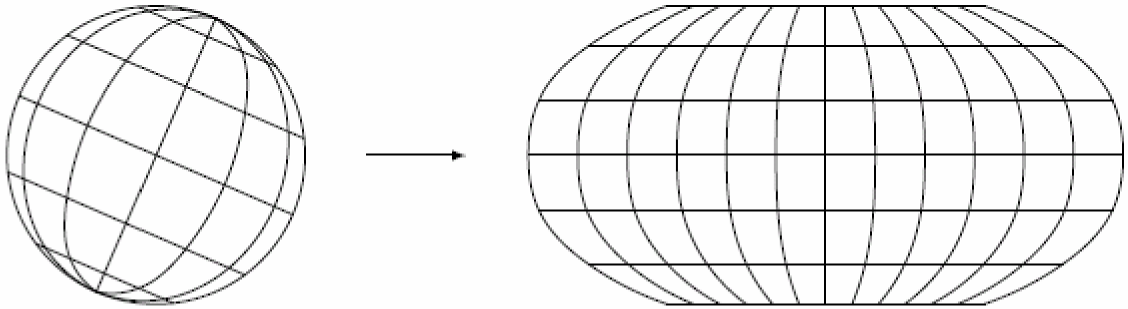
Solution: **time dependent learning rate**

$$\eta(t) = \eta_0 \alpha^t, \quad 0 < \alpha < 1, \qquad \text{or} \qquad \eta(t) = \eta_0 t^{\kappa}, \quad \kappa > 0.$$

Topology Preserving Mapping

Images of points close to each other in the original space should be close to each other in the image space.

Example: **Robinson projection** of the surface of a sphere



- Robinson projection is frequently used for world maps.

Self-Organizing Maps: Neighborhood

Find topology preserving mapping by respecting the neighborhood

Reference vector update rule:

$$\vec{r}_u^{\,(\text{new})} = \vec{r}_u^{\,(\text{old})} + \eta(t) \cdot f_{\text{nb}}(d_{\text{neurons}}(u, u_*), \varrho(t)) \cdot (\vec{p} - \vec{r}_u^{\,(\text{old})}),$$

- u_* is the winner neuron (reference vector closest to data point).
- The function f_{nb} is a radial function.

Time dependent learning rate

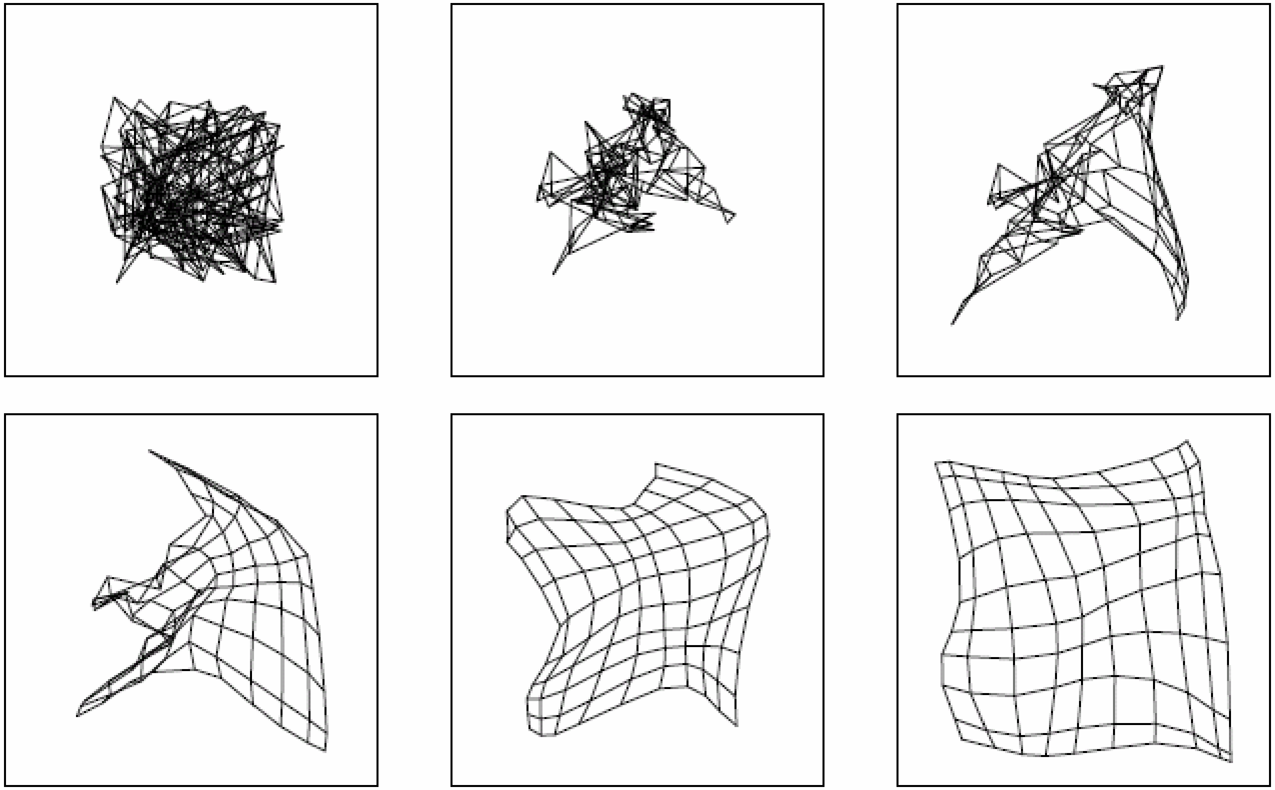
$$\eta(t) = \eta_0 \alpha_\eta^t, \quad 0 < \alpha_\eta < 1, \qquad \text{or} \qquad \eta(t) = \eta_0 t^{\kappa_\eta}, \quad \kappa_\eta > 0.$$

Time dependent neighborhood radius

$$\varrho(t) = \varrho_0 \alpha_\varrho^t, \quad 0 < \alpha_\varrho < 1, \qquad \text{or} \qquad \varrho(t) = \varrho_0 t^{\kappa_\varrho}, \quad \kappa_\varrho > 0.$$

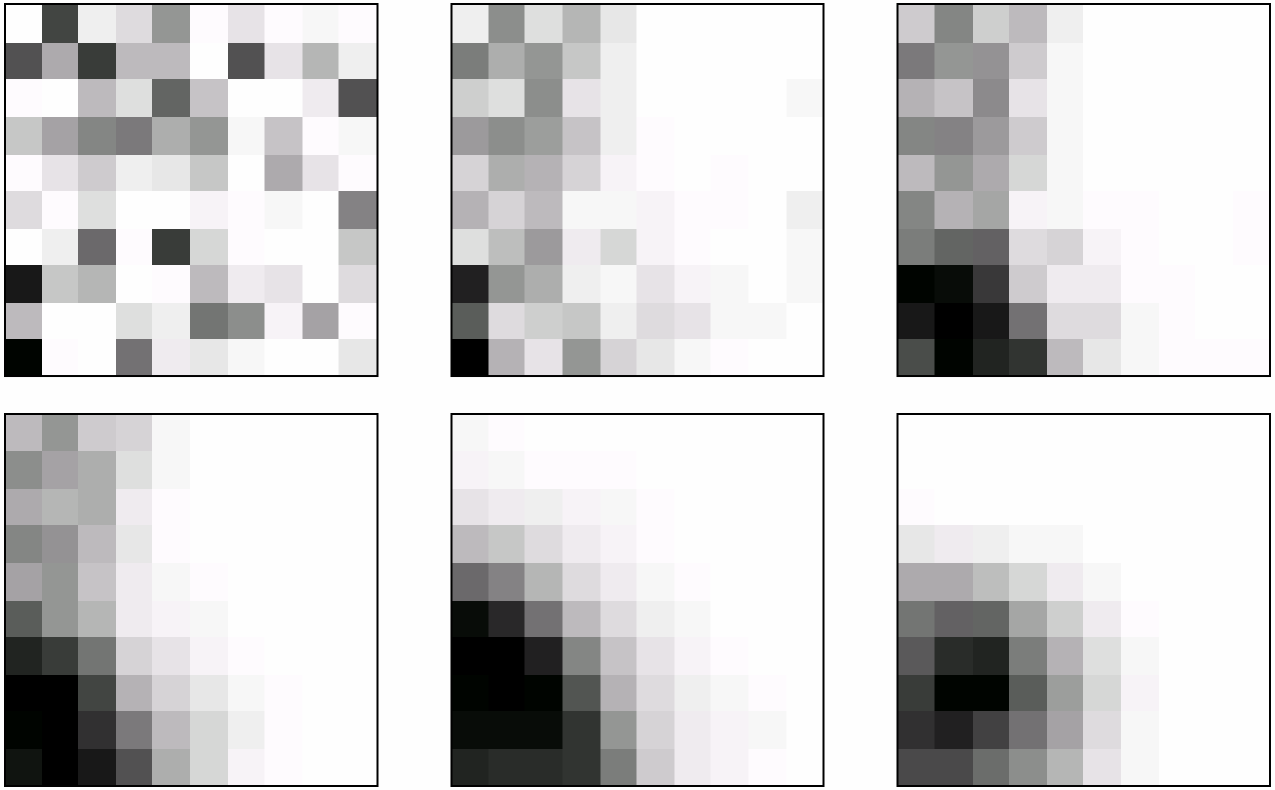
Self-Organizing Maps: Examples

Example: Unfolding of a two-dimensional self-organizing map.



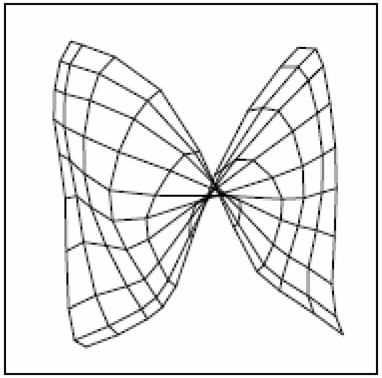
Self-Organizing Maps: Examples

Example: Unfolding of a two-dimensional self-organizing map.



Self-Organizing Maps: Examples

Example: Unfolding of a two-dimensional self-organizing map.

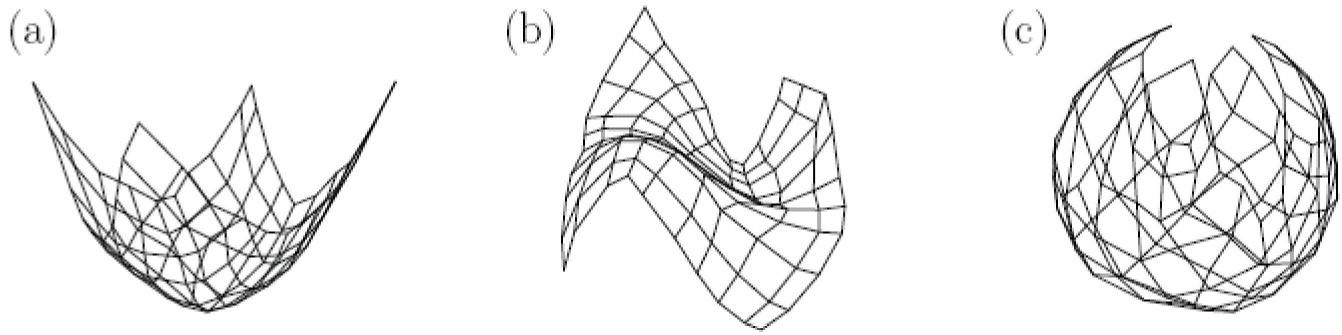


Training a self-organizing map may fail if

- the (initial) learning rate is chosen too small or
- or the (initial) neighbor is chosen too small.

Self-Organizing Maps: Examples

Example: Unfolding of a two-dimensional self-organizing map.



Self-organizing maps that have been trained with random points from
(a) a rotation parabola, (b) a simple cubic function, (c) the surface of a sphere.

- In this case original space and image space have different dimensionality.
- Self-organizing maps can be used for dimensionality reduction.

Phonemkarte

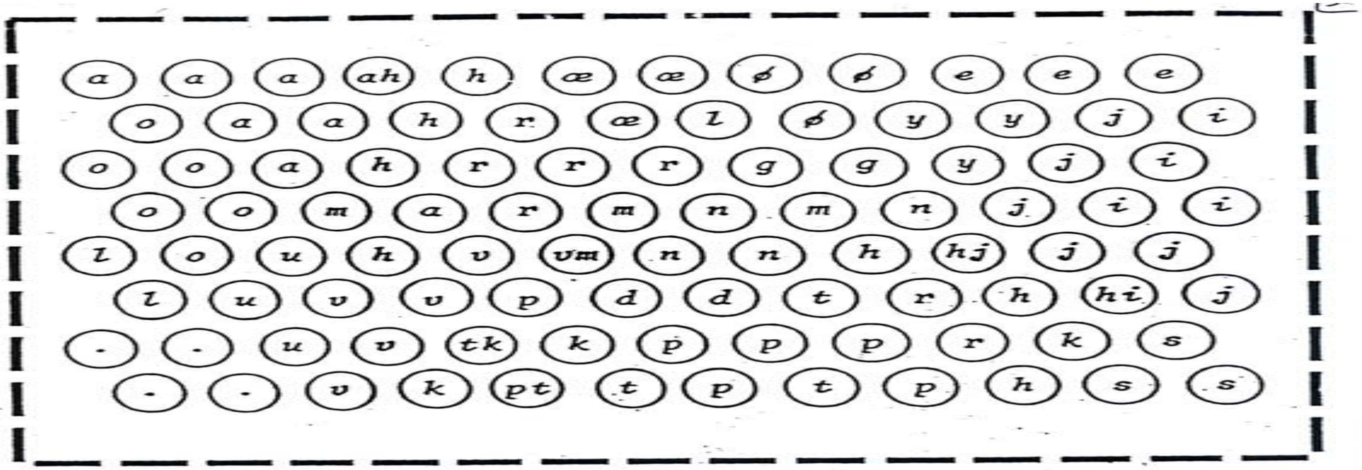


Abb. 2.6.7 Phonemkarte des Finnischen (nach [KOH88])

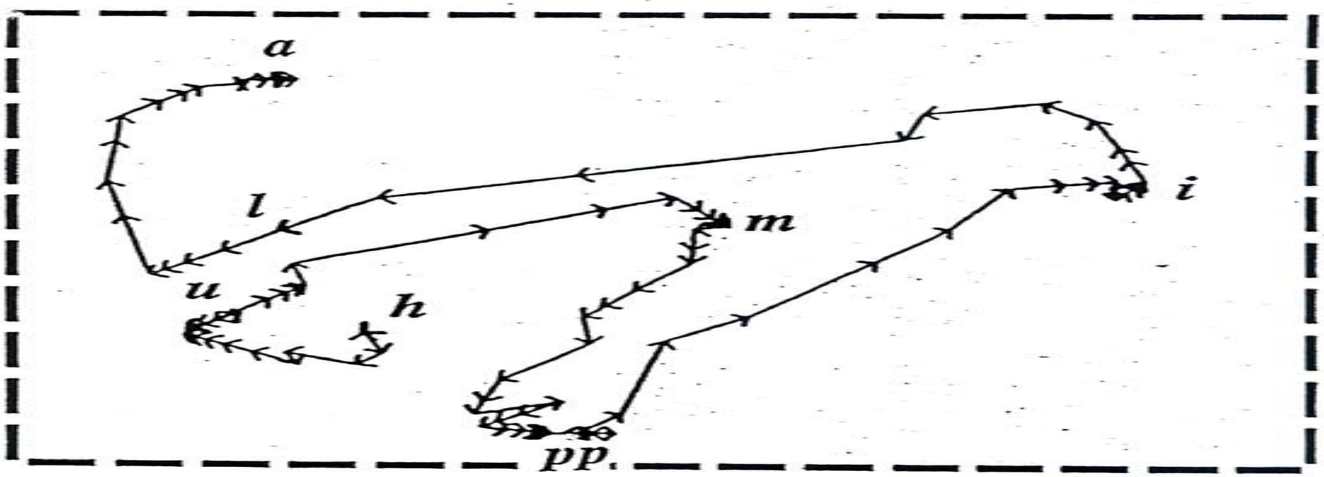
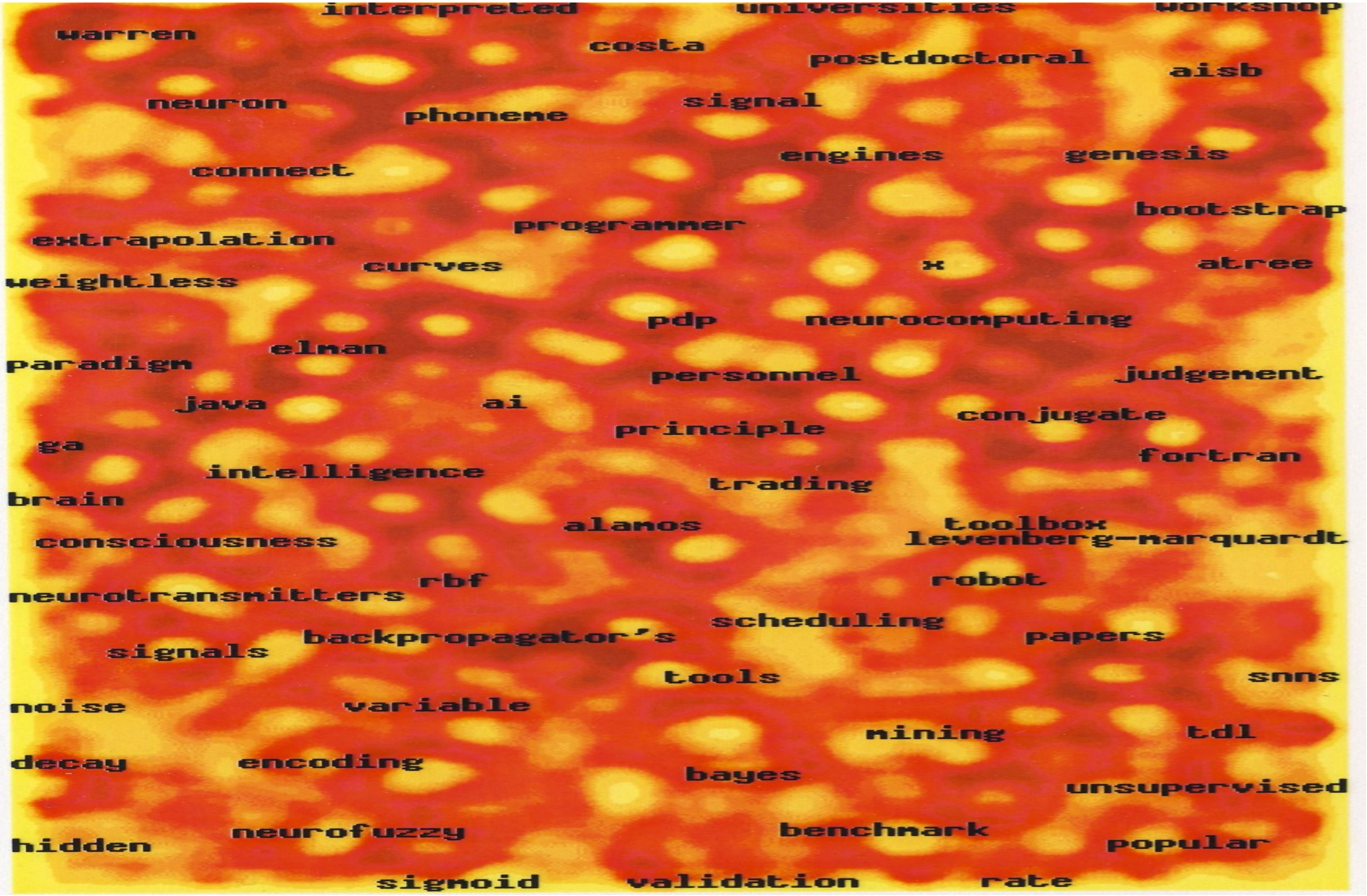


Abb. 2.6.8 Phonemsequenz für /humppila/ (nach [KOH88])

websom


interpreted
universities
workshop
warren
costa
postdoctoral
aisb
neuron
phoneme
signal
engines
genesis
connect
bootstrap
programmer
extrapolation
curves
x
atree
weightless
pdp
neurocomputing
elman
paradigm
personnel
judgement
java
ai
conjugate
principle
ga
fortran
intelligence
trading
brain
alamos
toolbox
levenberg-marquardt
consciousness
rbf
robot
neurotransmitters
scheduling
backpropagator's
papers
signals
tools
snns
noise
variable
mining
tdl
decay
encoding
bayes
unsupervised
neurofuzzy
benchmark
hidden
popular
sigmoid
validation
rate

Organising texts

Limitations of available text retrieval methods.

Ideas:

- Grouping documents based on a similarity measure
  ➔ Supports the user to navigate through similar documents.
- Navigation supported by conventional keyword search
  ➔ Important for the “first appropriate document”

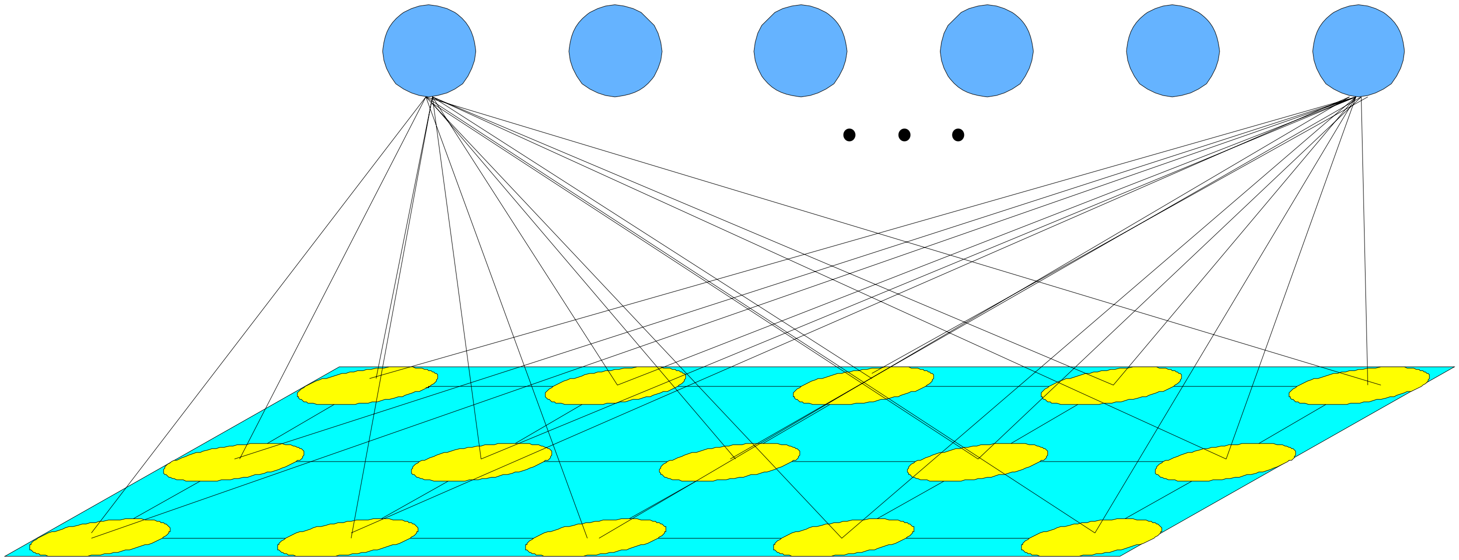
Realisation:

Interactive software tool based on self-organising maps:

➔ Interactive associative search

➔ Visualization for better overall view

Self Organising Map (SOM)

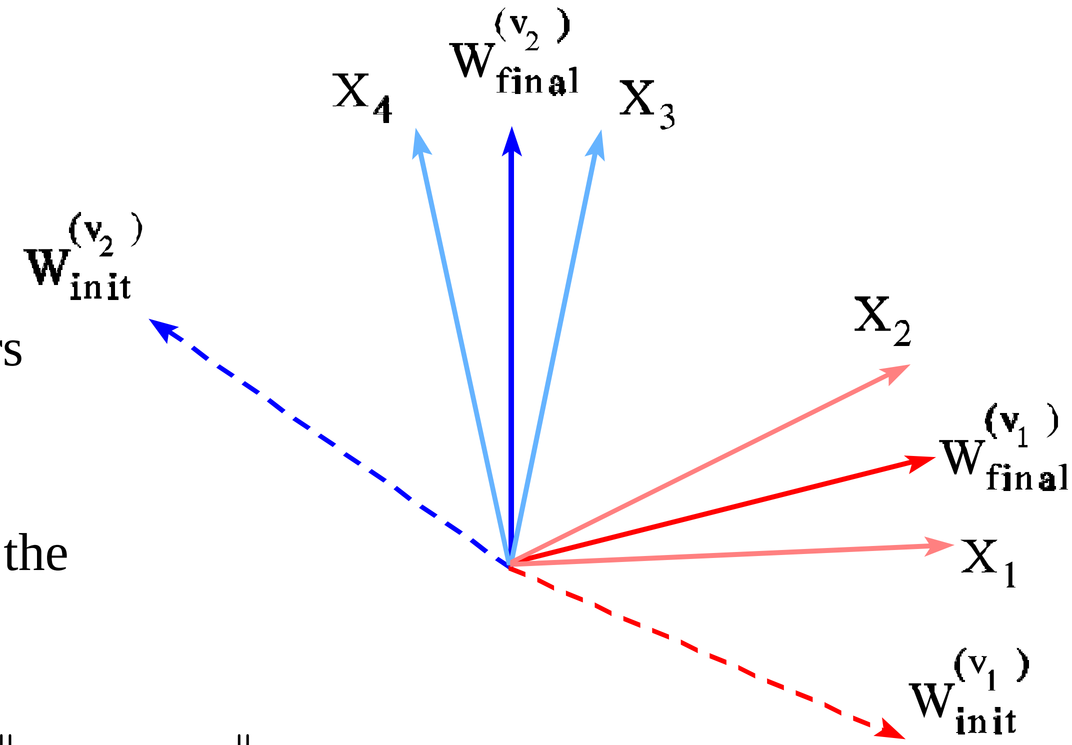


Artificial neural network model to project high-dimensional data vectors to lower dimensional data space (usually two dimensions) under preservation of neighbourhood relations.

SOM (Competitive learning)

Learning method (competitive learning):

- Weights (prototypes) w_i are randomly initialised.
- Adaptation of the model vectors carried out by a sequential regression process.
- For each input vector $x(t)$, first the winner index c (best match) is identified by the condition:



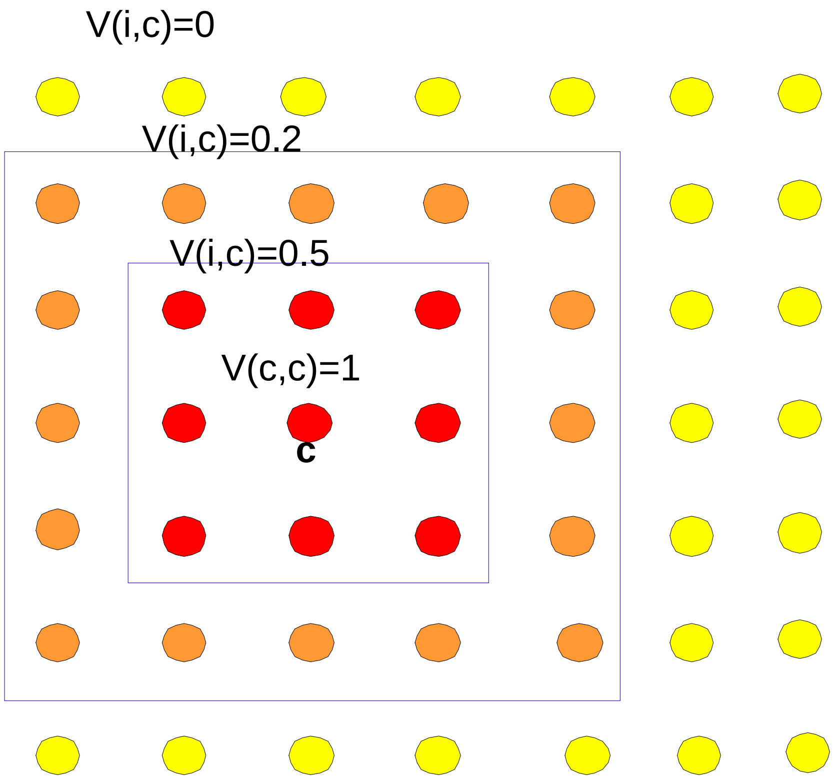
$$\forall i : \left\| \mathrm{w}_c - x(t) \right\| \le \left\| \mathrm{w}_i - x(t) \right\|$$

- The assigned vector w_c is adjusted such that for the next presentation of the same input vector a higher degree of similarity will be obtained:

$$\forall i : \mathrm{w}_i = \mathrm{w}_i + \delta \cdot (\mathrm{w}_i - x(t))$$

SOM Learning:

Competitive learning, additionally neighbourhood relations defined



All vectors *i* in a neighbourhood of the winner neuron *c* are adjusted:

$$\forall i : \mathrm{w}_i = \mathrm{w}_i + v(c,i) \cdot \delta \cdot (\mathrm{w}_i - x(t))$$

$v(i, c)$: neighbourhood function

δ : learning rate.

Self Organizing Map (SOM)

Properties:

- Topology preserving mapping
- Clustering of input data (unsupervised learning)
- Density of clusters is adjusted to density of input data
- Dimensionality reduction
- Good visualisation capabilities

Problems:

- Manual determination of structure and size

  Map to small: Grouping of different objects

  Map to large: Similar objects distributed in large area.

Document preprocessing and coding

➔ Reduce number of words to be considered

- **Filtering (stop word filtering):**

  Removing words that carry no or only little (content) information, e.g. articles, conjunctions, prepositions.

- **Stemming:**

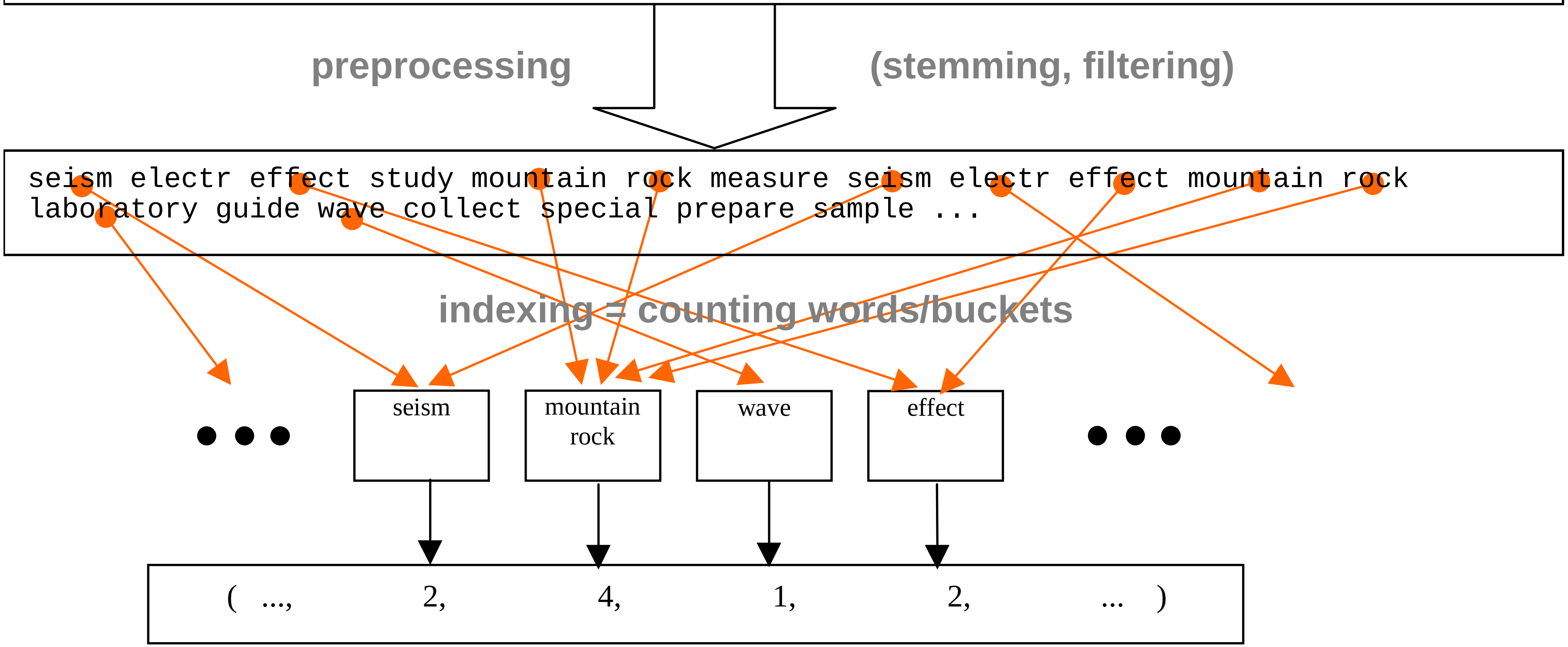
  Build the basic forms of words, e.g. strip plurals ‚s‘ from nouns and ‚ing‘ from verbs.

- **Coding:**

  Documents are indexed by remaining words.

Computation of ‚Fingerprints‘

Seismic-electric effect study of mountain rocks
Measurements of seismic-electric effect (SEE) of mountain rocks in laboratory on guided waves were continued with very wide collection of specially prepared samples ...



vector = “document fingerprint”

Defining the bins (Selection of index words based on entropy)

- **Calculate entropy for each word** as a measure for its importance:

$$W(w) = 1 + \frac{1}{\ln(m)} \sum_{i=1}^{m} p_i(w) \cdot \ln(p_i(w))$$

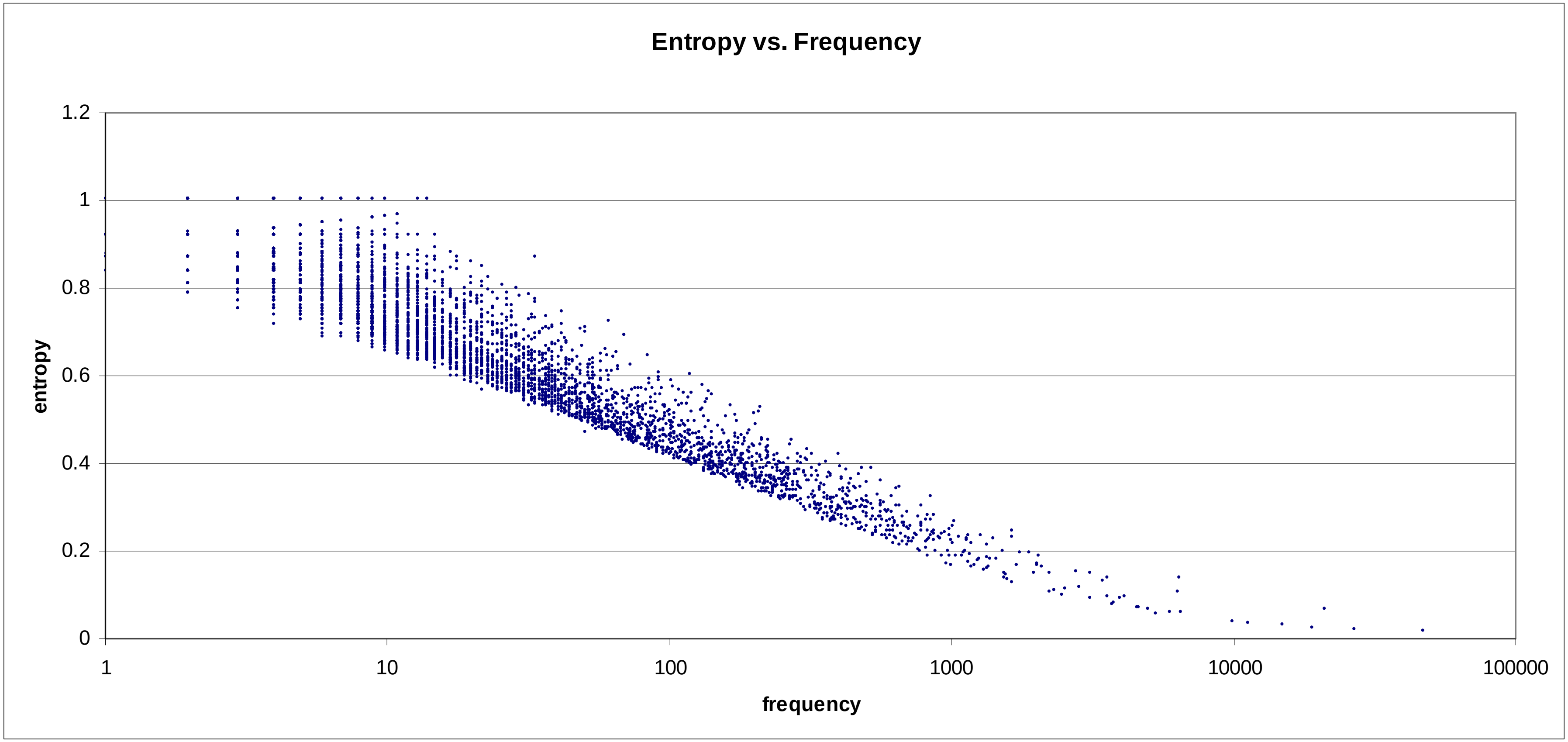
$$\text{with} \quad p_i(w) = \frac{n_i(w)}{\sum_{j=1}^{m} n_j(w)}$$

$n_i(w)$: frequency of word w in document i

m: number of documents

- Choose words that have a high entropy relative to their overall frequency
- Use these words as bins for fingerprint counting

Defining the bins (Selection of index words based on entropy)

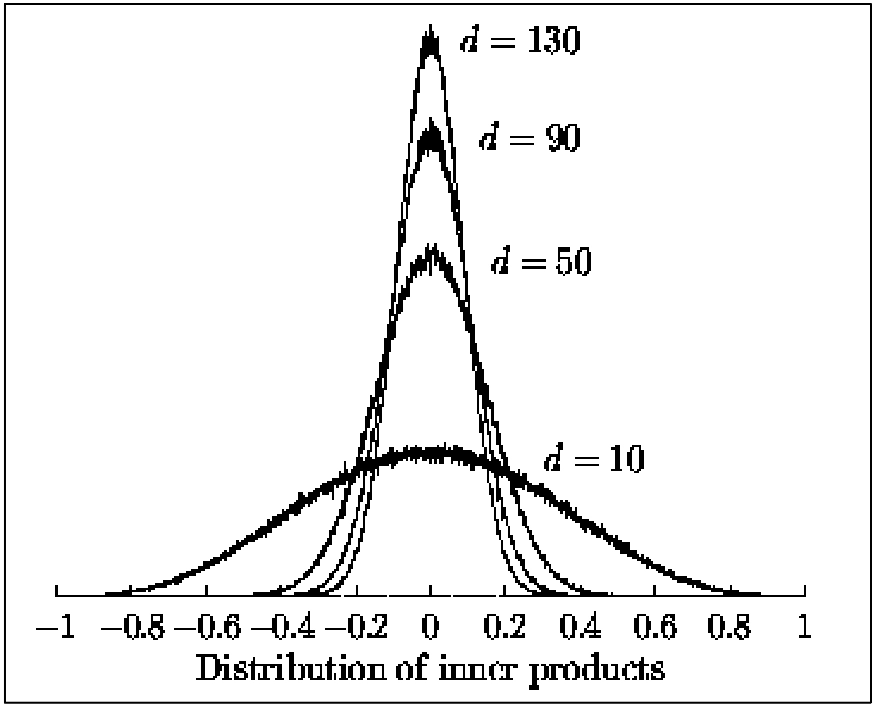


Defining the bins (Creating a word category map)

Encode words as high-dimensional random vectors

(Ritter and Kohonen, 1989)

➔ Encoding does not imply any word ordering: the vectors are “quasi-orthogonal"



Defining the bins (Creating a word category map)

Grouping similar words according to 3-word-contexts

For each word calculate the expectation value vectors e_1 and e_2 over all random vectors of enclosing words (in all documents) and create a context vector v based on these vectors and the random vector w of the considered word (Honkela et al., 1996):

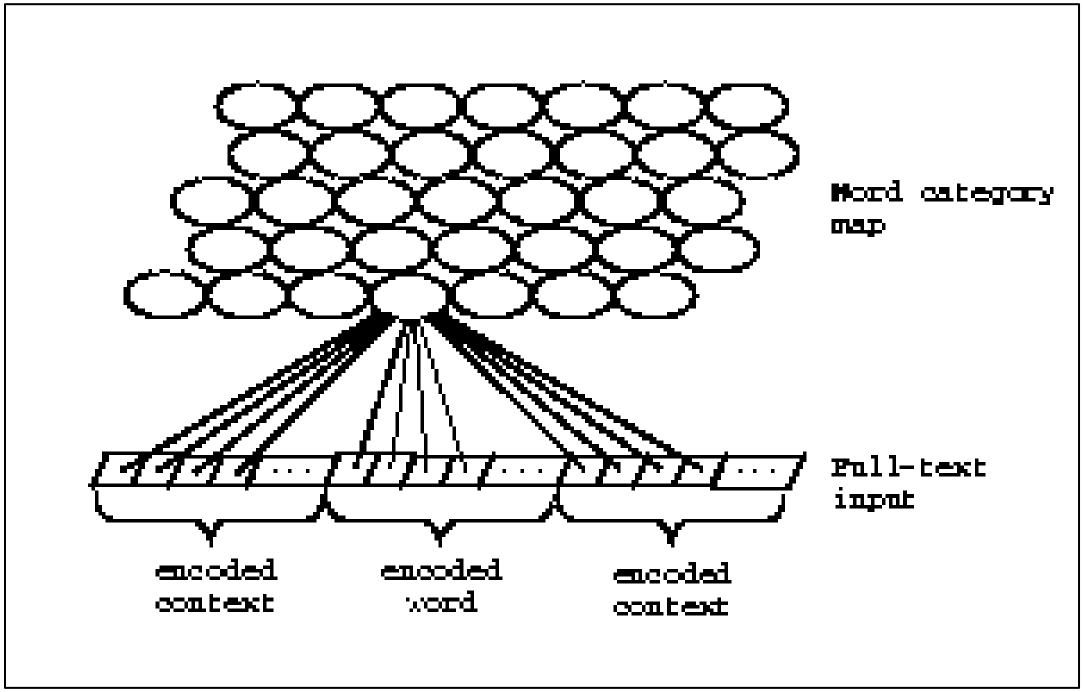
$$v = \{e_1 \, w \, e_2\}$$

➔ Words that occur in similar contexts have similar expectation values and therefore similar vectors v

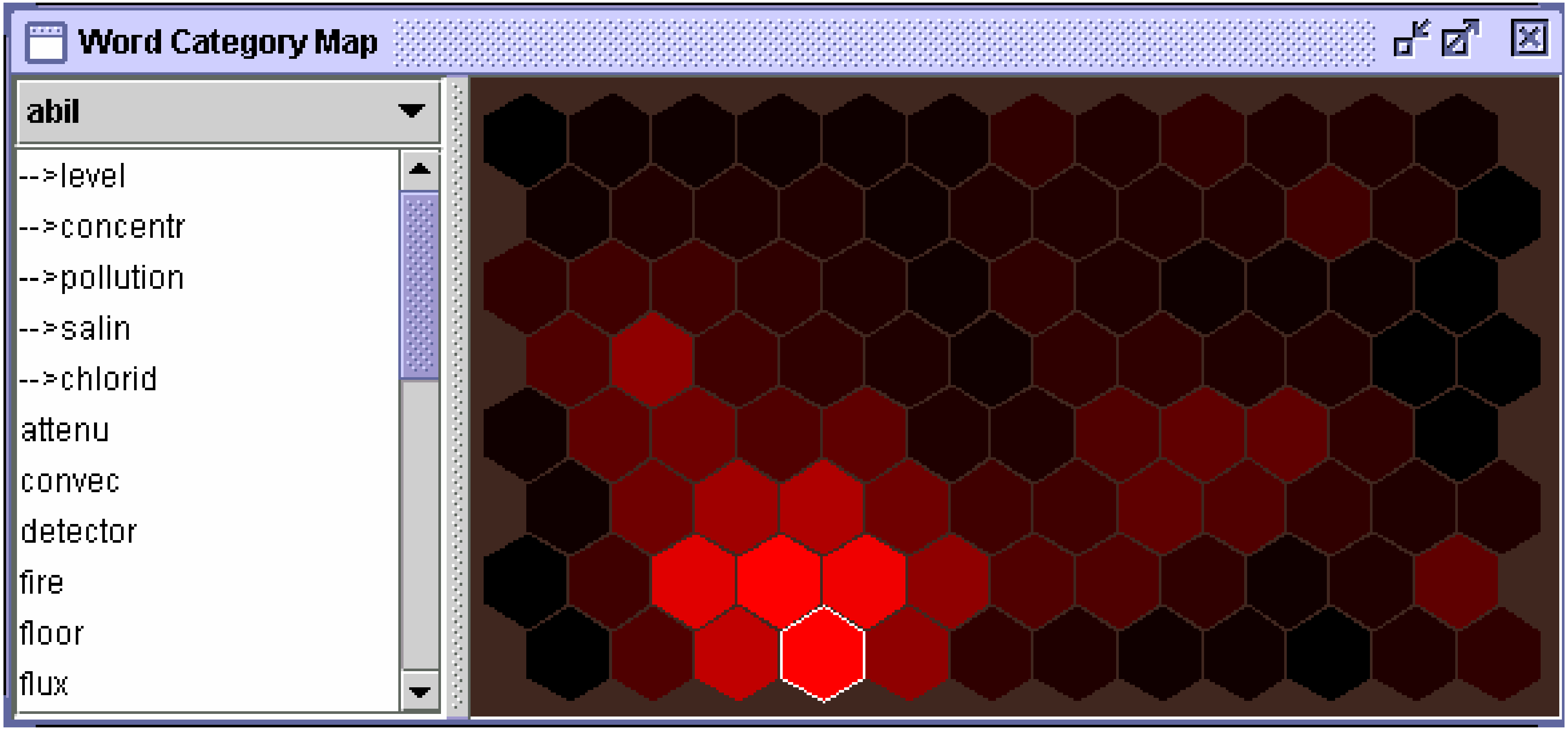
➔ Searching for lexical affinities

Defining the bins (Creating a word category map)

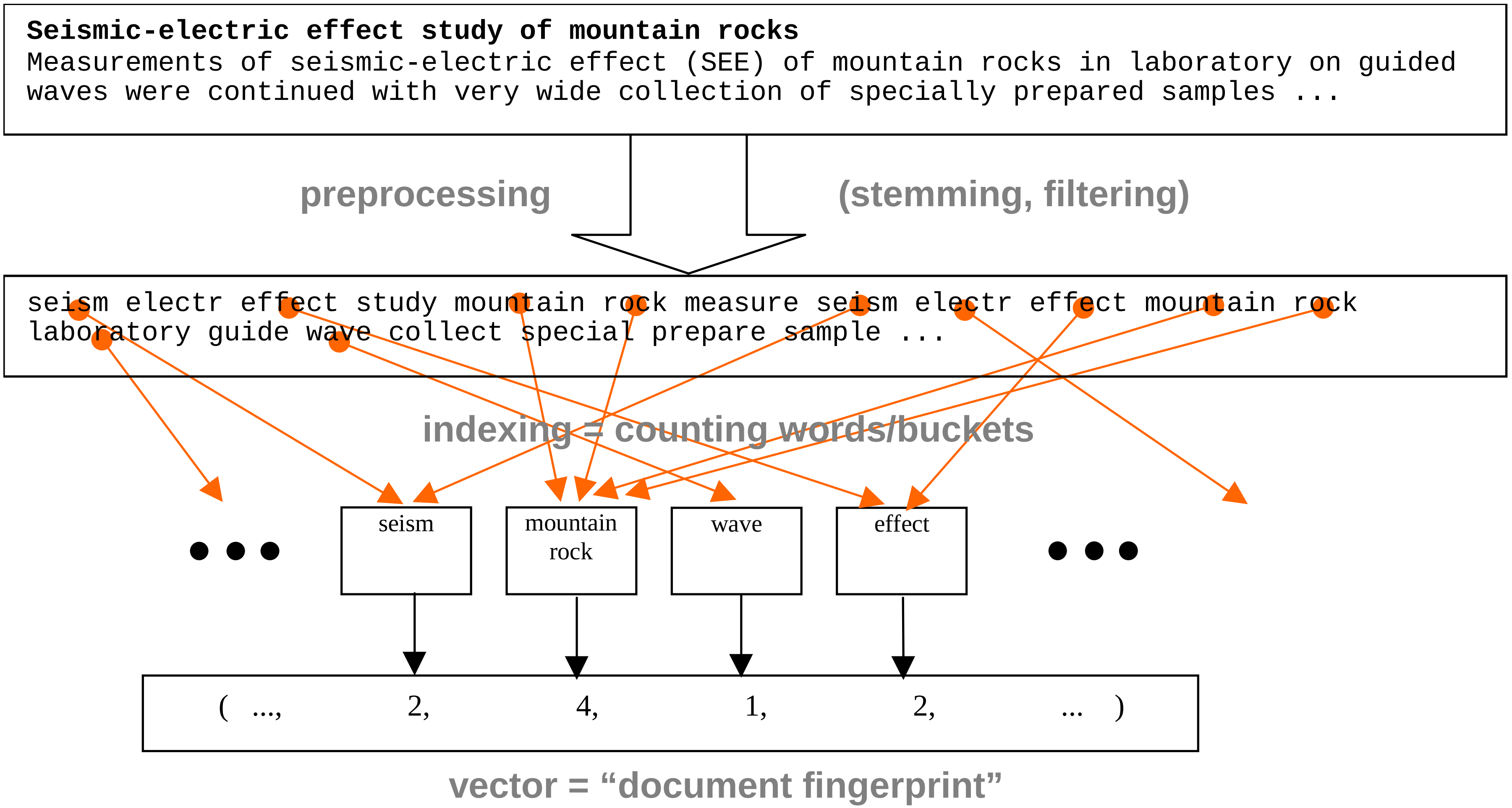
- Map vectors v_i to two dimensional space using a self organising map: Words frequently used in similar contexts are mapped to the same (or nearby) neuron.
- Each neuron of the resulting map is used as a bin for fingerprint counting.



The wordmap


Word Category Map
abil
-->level
-->concentr
-->pollution
-->salin
-->chlorid
attenu
convec
detector
fire
floor
flux

Computation of ‚Fingerprints‘



Arranging the documents (The document map)

- The fingerprints of the documents are used as input vectors for a two-dimensional self organising map.

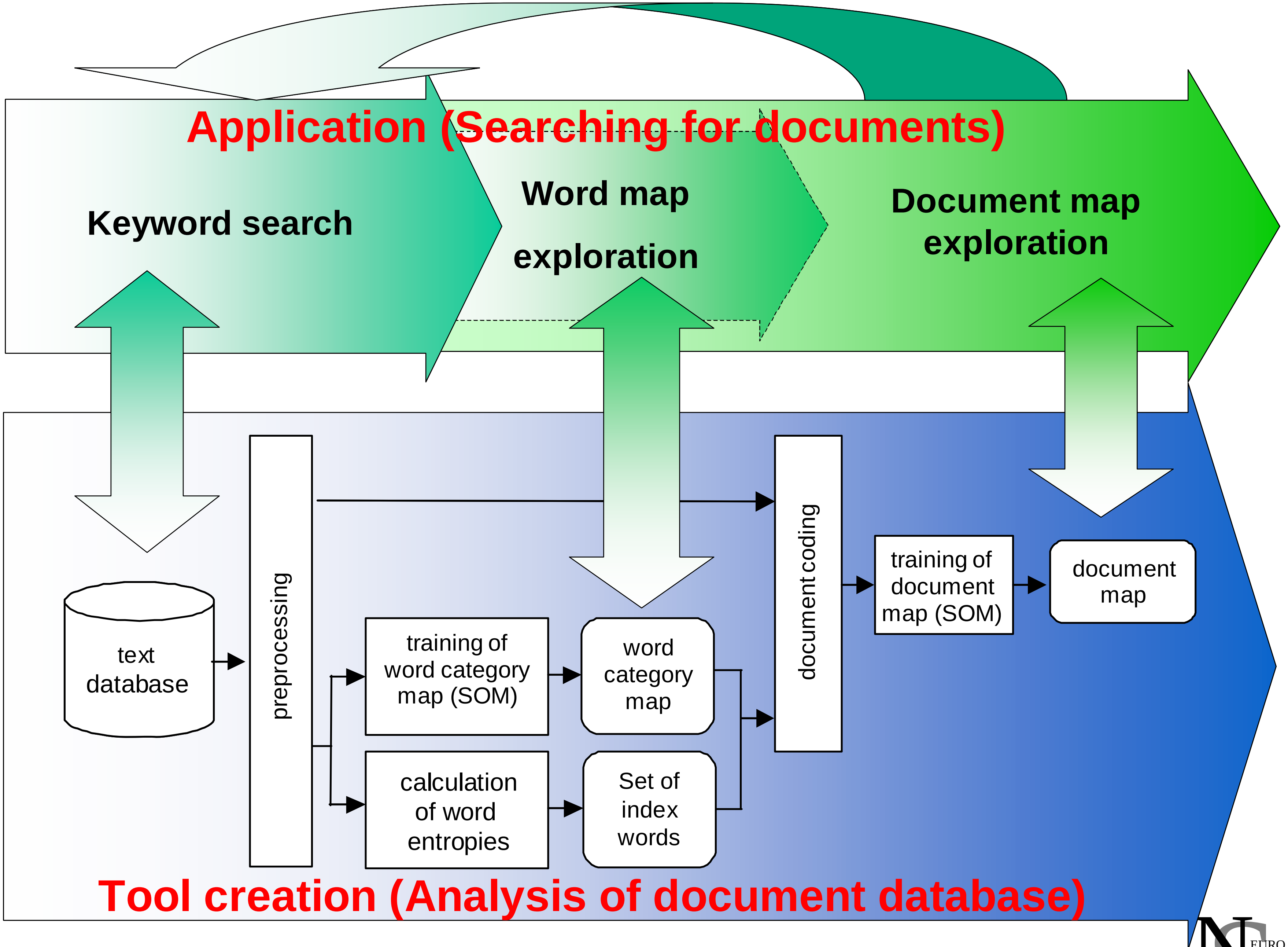
Dynamical Aspects

Changes in document database:

- **Small changes:** preprocess documents, compute buckets, and map documents on existing maps
- **Extensive changes:**

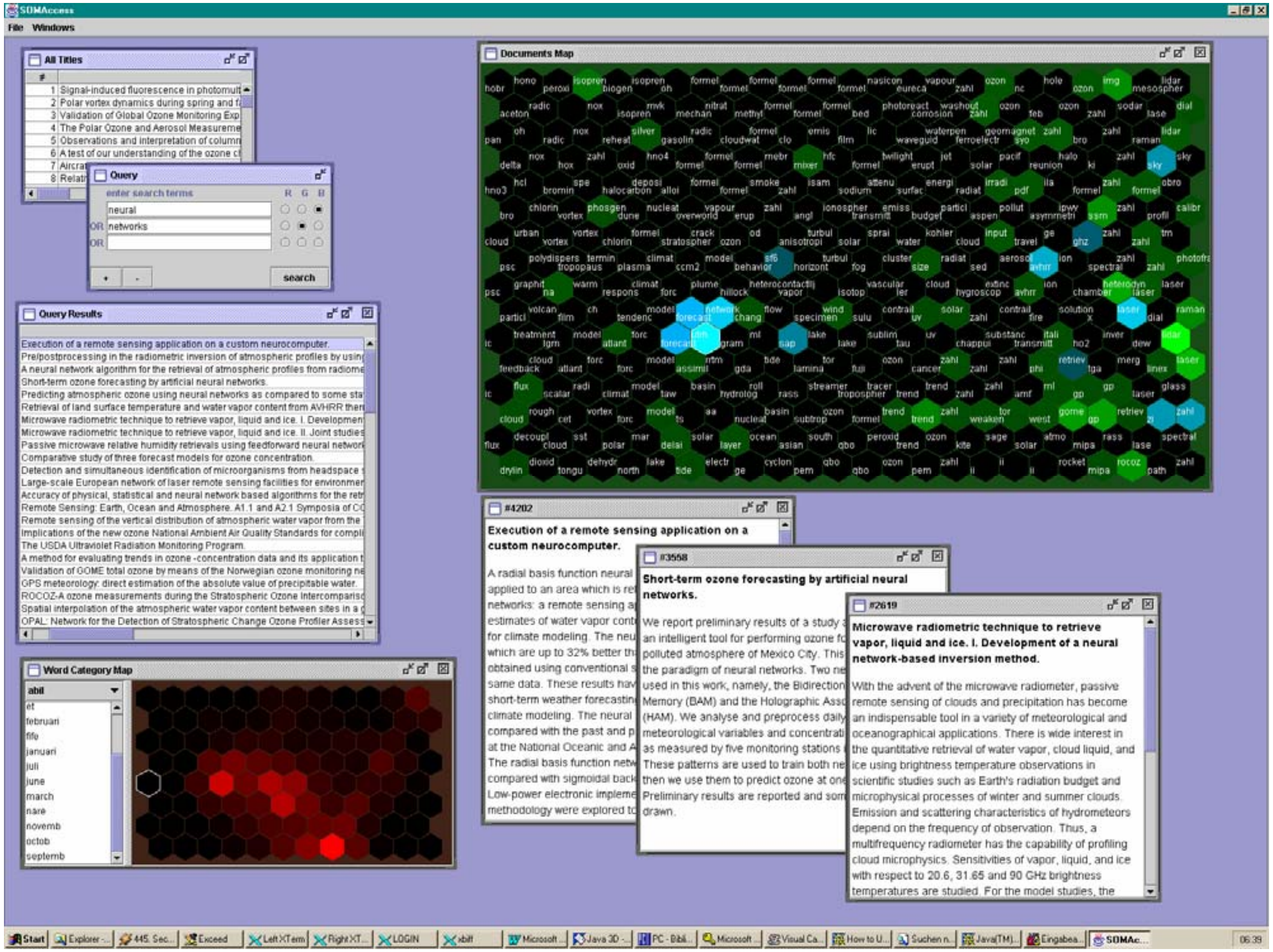
  Different approaches possible:

  - Retrain document map (incremental), keep buckets
  - Relearn document map from scratch, keep buckets (affects users which are already working with the map)
  - Retrain complete system (analysis of buckets and wordmap might yield hints on new topics)



Application (Searching for documents)
Keyword search
Word map exploration
Document map exploration
text database
preprocessing
training of word category map (SOM)
calculation of word entropies
word category map
Set of index words
document coding
training of document map (SOM)
document map
Tool creation (Analysis of document database)

SOMAccess V1.0



Available on CD-ROM: G. Hartmann, A. Nölle, M. Richards, and R. Leitinger (eds.), Data Utilization Software Tools 2 (DUST-2 CD-ROM), Copernicus Gesellschaft e.V., Katlenburg-Lindau, 2000 (ISBN 3-9804862-3-0)

Music Miner


Music Miner
Ein datenbionisches System
zur Organisation von Musiksammlungen
SoundMap von
200 Musikstücken
THE
BEATLES
mono
YESTERDAY