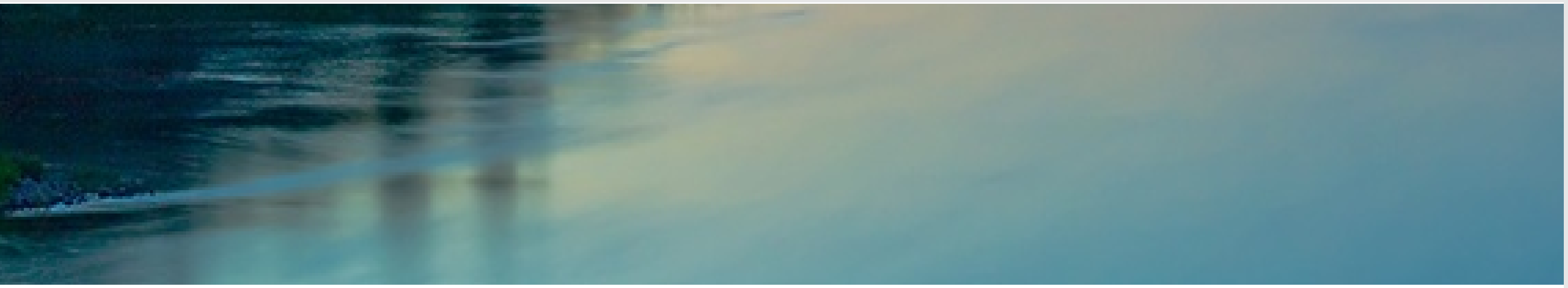


Neural Networks

Prof. Dr. Rudolf Kruse

Computational Intelligence Group
Faculty for Computer Science
`kruse@iws.cs.uni-magdeburg.de`

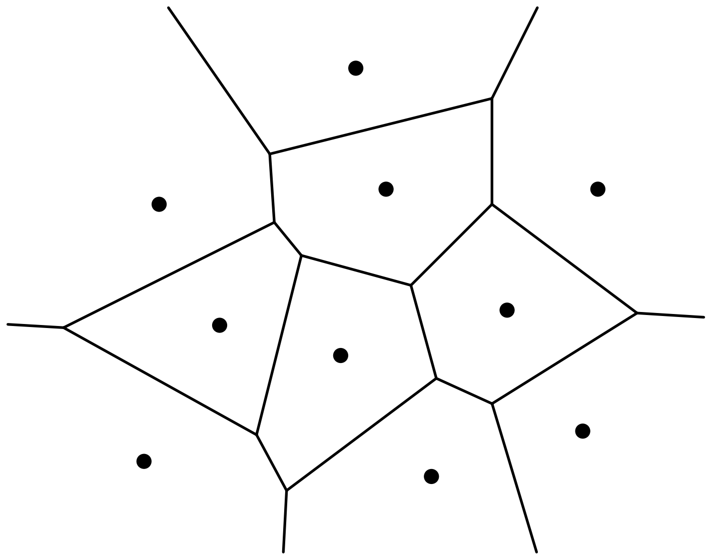
Learning Vector Quantization

Motivation

- previously: static learning
  now: „free“ learning, i.e. without known class labels or target values as training samples
- main idea: similar inputs leading to similar outputs
- like clustering: close data points in the input space are close together in the output space, too.

Vector Quantization

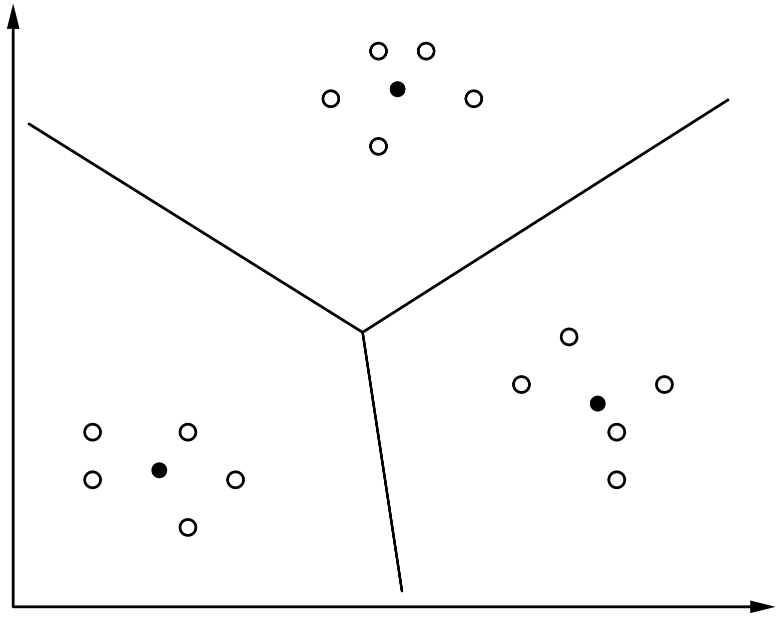
Voronoi diagram of a vector quantization



- 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 ($\circ$).
- Cluster centers are represented by full circles ($\bullet$).

Learning Vector Quantization Networks

A **learning vector quantization network (LVQ)** is a neural network with a graph $G = (U, C)$ that satisfies the following conditions

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

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

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

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

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

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

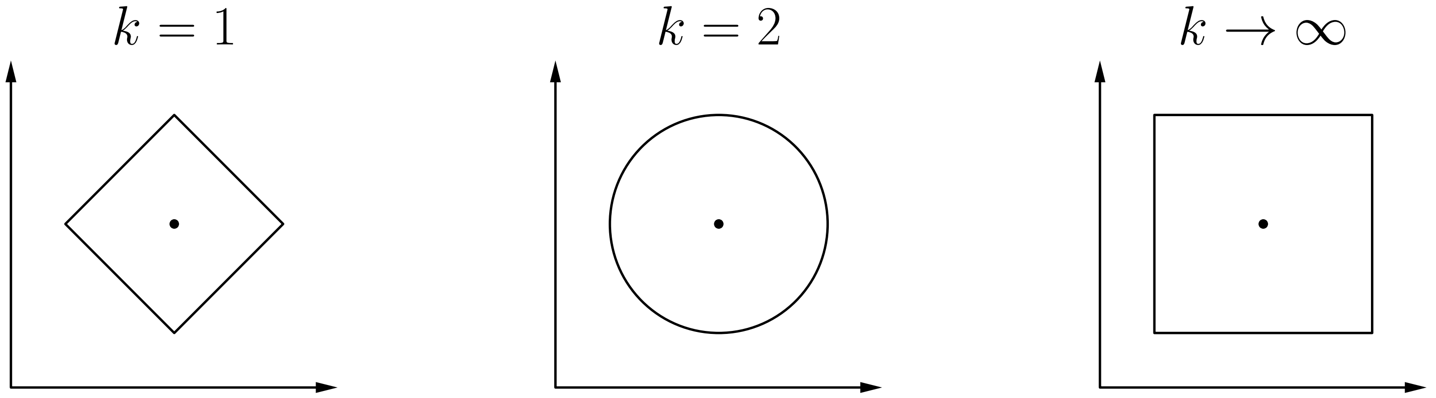
Distance Functions

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. $d_\infty(\vec{x},\vec{y}) = \max_{i=1}^{n}\|x_i - y_i\|$.



(all points lying on a circle or rectangle are sharing the same distance to the center point, according to the corresponding distance function)

Learning Vector Quantization

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

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

Sometimes the range of values is restricted to the interval $[0, 1]$.
However, due to the special output function this restriction is irrelevant.

The output function of each output neuron is not a simple function of the activation of the neuron. Rather it takes into account the activations of all output neurons:

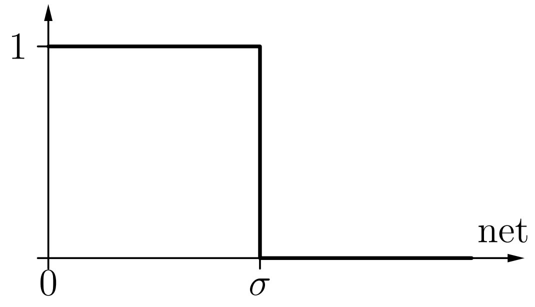
$$f_{\text{out}}^{(u)}(\text{act}_u) = \begin{cases} 1, & \text{if } \text{act}_u = \max\limits_{v \in U_{\text{out}}} \text{act}_v, \\ 0, & \text{otherwise.} \end{cases}$$

If more than one unit has the maximal activation, one is selected at random to have an output of 1, all others are set to output 0: **winner-takes-all principle**.

Radial Activation Functions

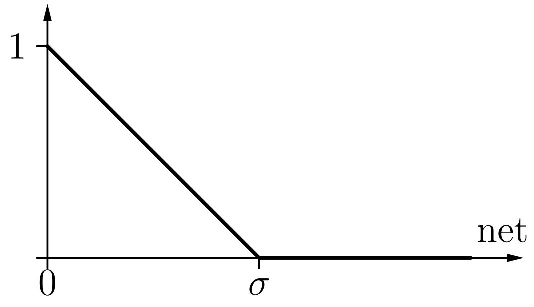
rectangle function:

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



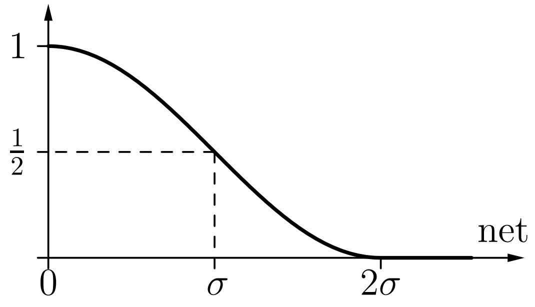
triangle function:

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



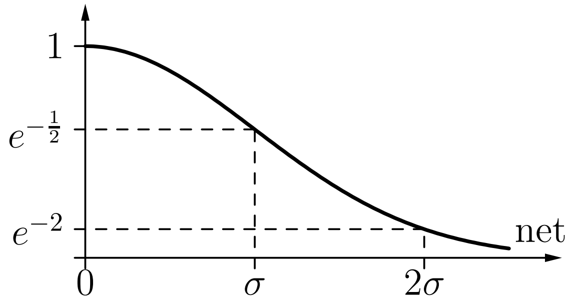
cosine until zero:

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



Gaussian function:

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



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:
  Each reference vector is assigned to a class.

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

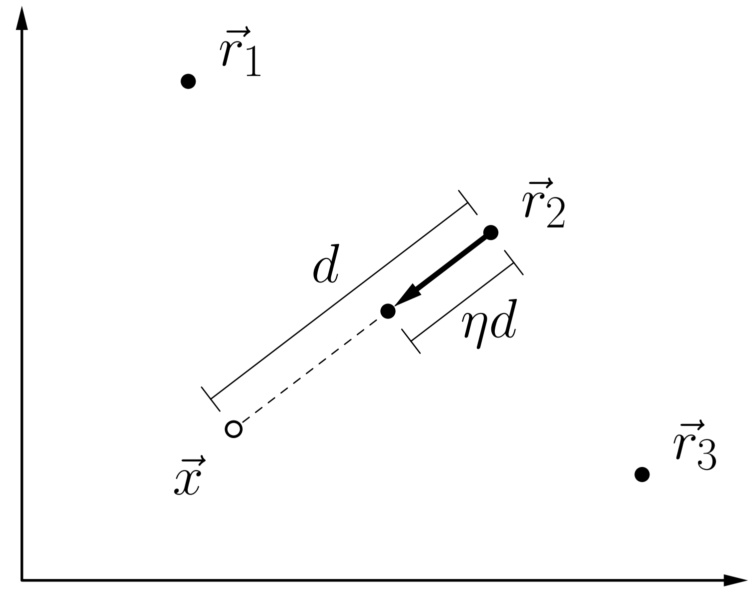
$$\vec{r}^{\,(\text{new})} = \vec{r}^{\,(\text{old})} + \eta(\vec{x} - \vec{r}^{\,(\text{old})}),$$

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

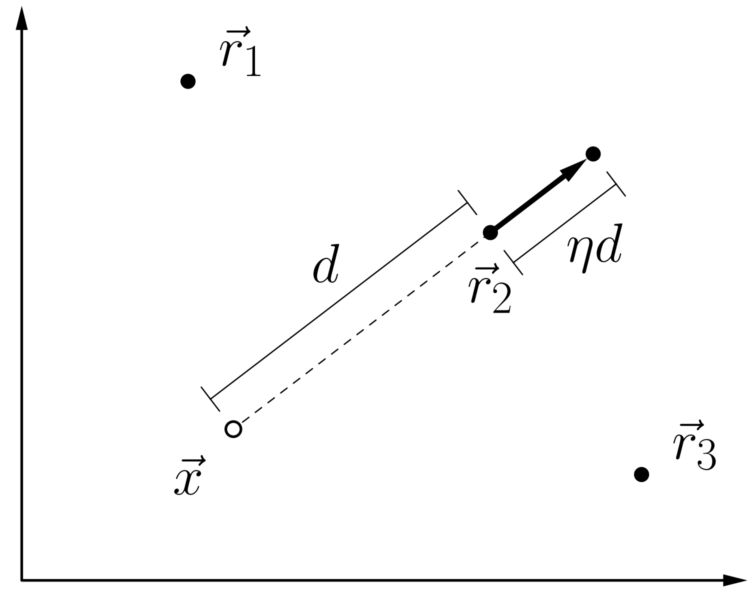
$$\vec{r}^{\,(\text{new})} = \vec{r}^{\,(\text{old})} - \eta(\vec{x} - \vec{r}^{\,(\text{old})}).$$

Learning Vector Quantization

Adaptation of reference vectors / codebook vectors



attraction rule

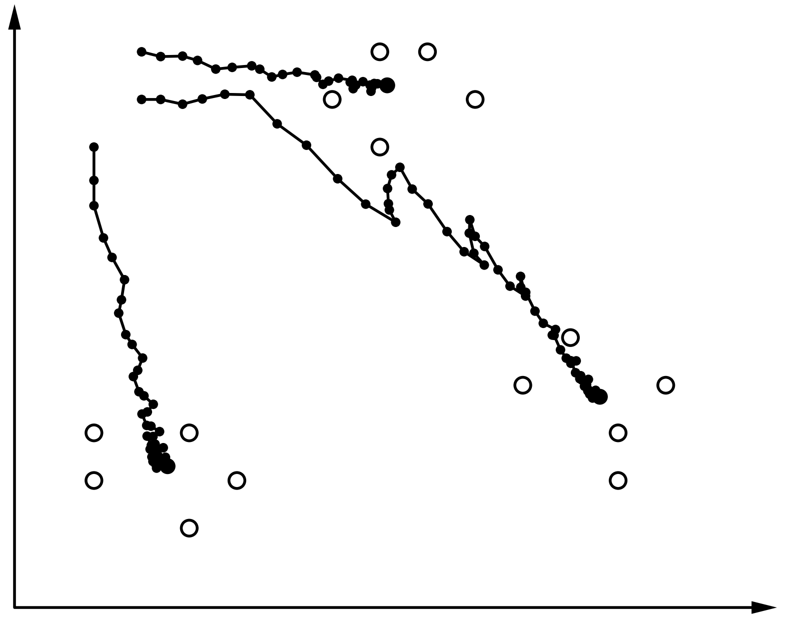


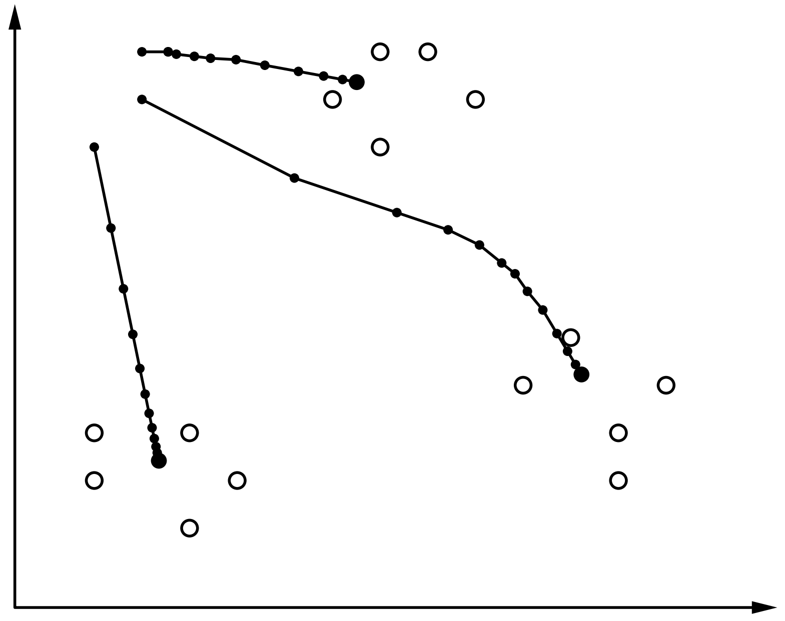
repulsion rule

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

Learning Vector Quantization: Example

Adaptation of reference vectors / codebook vectors

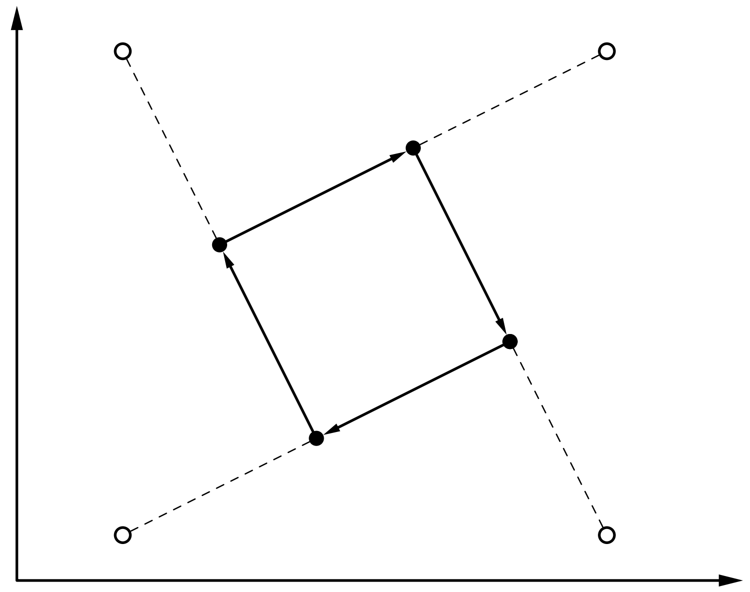


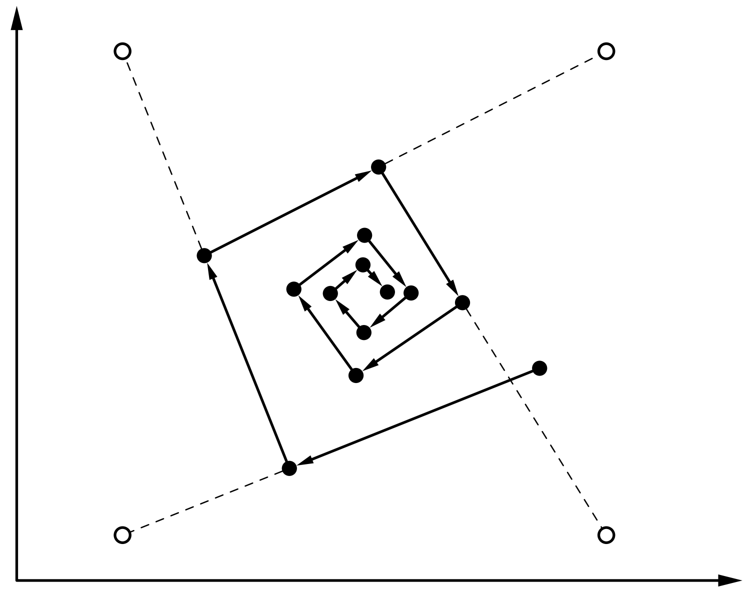


- Left: Online training with learning rate $\eta = 0.1$,
- Right: Batch training with learning rate $\eta = 0.05$.

Learning Vector Quantization: Learning Rate Decay

Problem: fixed learning rate can lead to oscillations





Solution: **time dependent learning rate**

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

Learning Vector Quantization: Classified Data

Improved update rule for classified data

- **Idea:** Update not only the one reference vector that is closest to the data point (the winner neuron), but **update the two closest reference vectors**.

- Let $\vec{x}$ be the currently processed data point and c its class.
  Let $\vec{r}_j$ and $\vec{r}_k$ be the two closest reference vectors and z_j and z_k their classes.

- Reference vectors are updated only if $z_j \neq z_k$ and either $c = z_j$ or $c = z_k$.
  (Without loss of generality we assume $c = z_j$.)
  The **update rules** for the two closest reference vectors are:

$$\begin{aligned} \vec{r}_j^{\,(\text{new})} &= \vec{r}_j^{\,(\text{old})} + \eta(\vec{x} - \vec{r}_j^{\,(\text{old})}) \qquad \text{and} \\ \vec{r}_k^{\,(\text{new})} &= \vec{r}_k^{\,(\text{old})} - \eta(\vec{x} - \vec{r}_k^{\,(\text{old})}), \end{aligned}$$

  while all other reference vectors remain unchanged.

Learning Vector Quantization: Window Rule

- It was observed in practical tests that standard learning vector quantization may drive the reference vectors further and further apart.
- To counteract this undesired behavior a **window rule** was introduced: update only if the data point $\vec{x}$ is close to the classification boundary.
- "Close to the boundary" is made formally precise by requiring

$$\min\left(\frac{d(\vec{x},\vec{r}_j)}{d(\vec{x},\vec{r}_k)}, \frac{d(\vec{x},\vec{r}_k)}{d(\vec{x},\vec{r}_j)}\right) > \theta, \qquad \text{where} \qquad \theta = \frac{1-\xi}{1+\xi}.$$

  ξ is a parameter that has to be specified by a user.

- Intuitively, ξ describes the "width" of the window around the classification boundary, in which the data point has to lie in order to lead to an update.
- Using it prevents divergence, because the update ceases for a data point once the classification boundary has been moved far enough away.

Soft Learning Vector Quantization

Idea: Use soft assignments instead of winner-takes-all.

Assumption: Given data was sampled from a mixture of normal distributions.
Each reference vector describes one normal distribution.

Objective: Maximize the log-likelihood ratio of the data, that is, maximize

$$\begin{aligned}\ln L_{\text{ratio}} &= \sum_{j=1}^{n} \ln \sum_{\vec{r}\in R(c_j)} \exp\left(-\frac{(\vec{x}_j-\vec{r})^\top(\vec{x}_j-\vec{r})}{2\sigma^2}\right)\\ &\quad - \sum_{j=1}^{n} \ln \sum_{\vec{r}\in Q(c_j)} \exp\left(-\frac{(\vec{x}_j-\vec{r})^\top(\vec{x}_j-\vec{r})}{2\sigma^2}\right).\end{aligned}$$

Here σ is a parameter specifying the "size" of each normal distribution.
$R(c)$ is the set of reference vectors assigned to class c and $Q(c)$ its complement.

Intuitively: at each data point the probability density for its class should be as large as possible while the density for all other classes should be as small as possible.

Soft Learning Vector Quantization

Update rule derived from a maximum log-likelihood approach:

$$\vec{r}_i^{\,(\text{new})} = \vec{r}_i^{\,(\text{old})} + \eta \cdot \begin{cases} u_{ij}^{\oplus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j = z_i, \\ -u_{ij}^{\ominus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j \neq z_i, \end{cases}$$

where z_i is the class associated with the reference vector $\vec{r}_i$ and

$$u_{ij}^{\oplus} = \frac{\exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}_i^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}_i^{\,(\text{old})})\right)}{\sum\limits_{\vec{r} \in R(c_j)} \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}^{\,(\text{old})})\right)} \qquad \text{and}$$

$$u_{ij}^{\ominus} = \frac{\exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}_i^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}_i^{\,(\text{old})})\right)}{\sum\limits_{\vec{r} \in Q(c_j)} \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_j - \vec{r}^{\,(\text{old})})^\top (\vec{x}_j - \vec{r}^{\,(\text{old})})\right)}.$$

$R(c)$ is the set of reference vectors assigned to class c and $Q(c)$ its complement.

Hard Learning Vector Quantization

Idea: Derive a scheme with hard assignments from the soft version.

Approach: Let the size parameter σ of the Gaussian function go to zero.

The resulting update rule is in this case:

$$\vec{r}_i^{\,(\text{new})} = \vec{r}_i^{\,(\text{old})} + \eta \cdot \begin{cases} u_{ij}^{\oplus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j = z_i, \\ -u_{ij}^{\ominus} \cdot (\vec{x}_j - \vec{r}_i^{\,(\text{old})}), & \text{if } c_j \neq z_i, \end{cases}$$

where

$$u_{ij}^{\oplus} = \begin{cases} 1, & \text{if } \vec{r}_i = \operatorname*{argmin}_{\vec{r} \in R(c_j)} d(\vec{x}_j, \vec{r}), \\ 0, & \text{otherwise,} \end{cases} \qquad u_{ij}^{\ominus} = \begin{cases} 1, & \text{if } \vec{r}_i = \operatorname*{argmin}_{\vec{r} \in Q(c_j)} d(\vec{x}_j, \vec{r}), \\ 0, & \text{otherwise.} \end{cases}$$

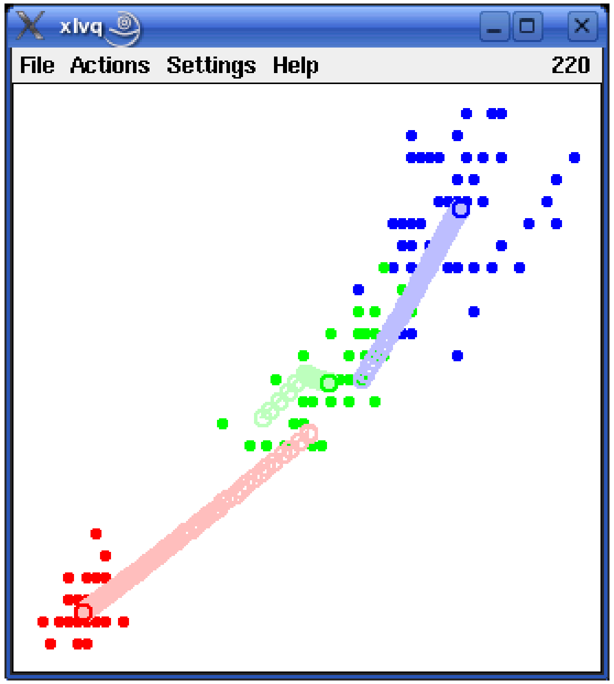
$\vec{r}_i$ is closest vector of same class $\qquad$ $\vec{r}_i$ is closest vector of different class

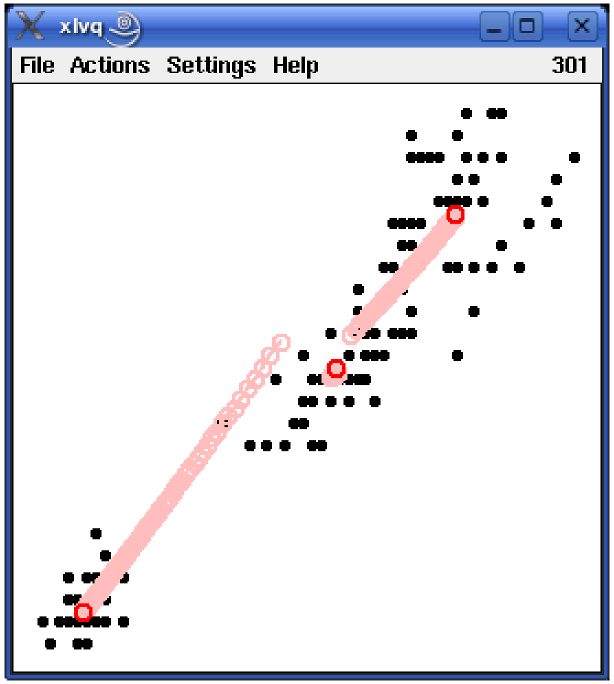
This update rule is stable without a *window rule* restricting the update.

Learning Vector Quantization: Extensions

- **Frequency Sensitive Competitive Learning**
  - The distance to a reference vector is modified according to
    the number of data points that are assigned to this reference vector.
- **Fuzzy Learning Vector Quantization**
  - Exploits the close relationship to fuzzy clustering.
  - Can be seen as an online version of fuzzy clustering.
  - Leads to faster clustering.
- **Size and Shape Parameters**
  - Associate each reference vector with a cluster radius.
    Update this radius depending on how close the data points are.
  - Associate each reference vector with a covariance matrix.
    Update this matrix depending on the distribution of the data points.

Demonstration Software: xlvq/wlvq





Demonstration of learning vector quantization:

- Visualization of the training process
- Arbitrary datasets, but training only in two dimensions
- http://www.borgelt.net/lvqd.html

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

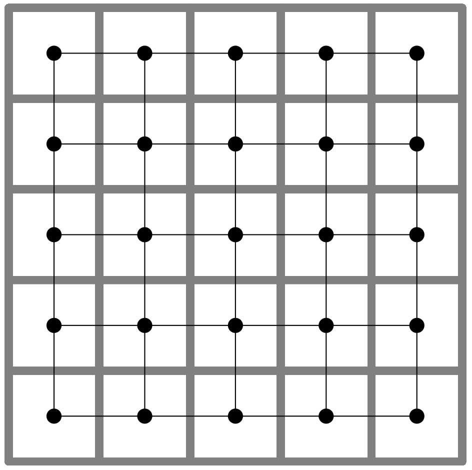
$$f : \mathbb{R}_0^+ \to [0, 1] \quad \text{with} \quad f(0) = 1 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 0.$$

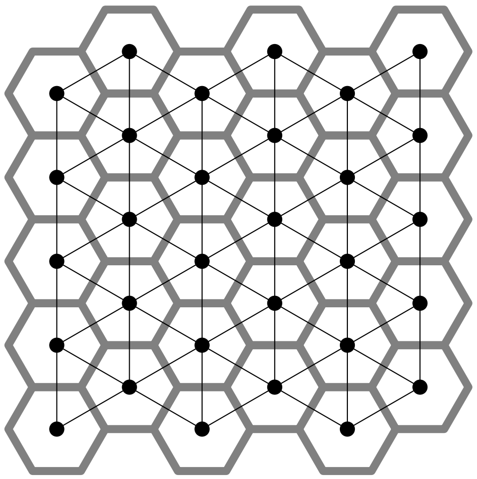
The output function of each output neuron is the identity.
The output is often discretized according to the "**winner takes all**" principle.
On the output neurons a **neighborhood relationship** is defined:

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

Self-Organizing Maps: Neighborhood

Neighborhood of the output neurons: neurons form a grid

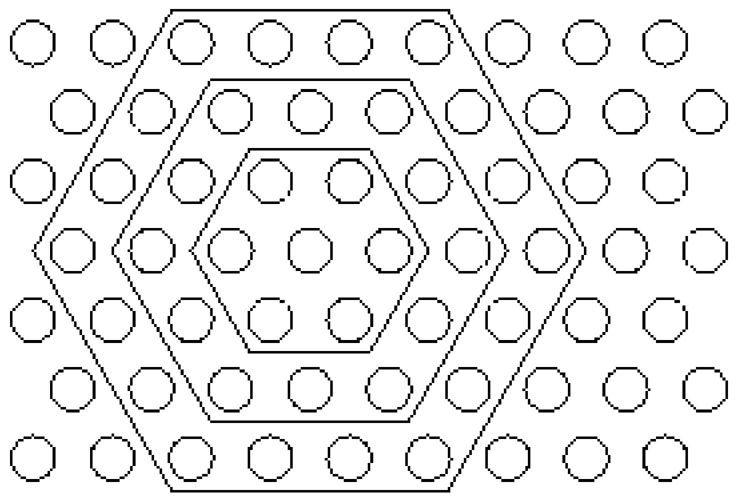

quadratic grid

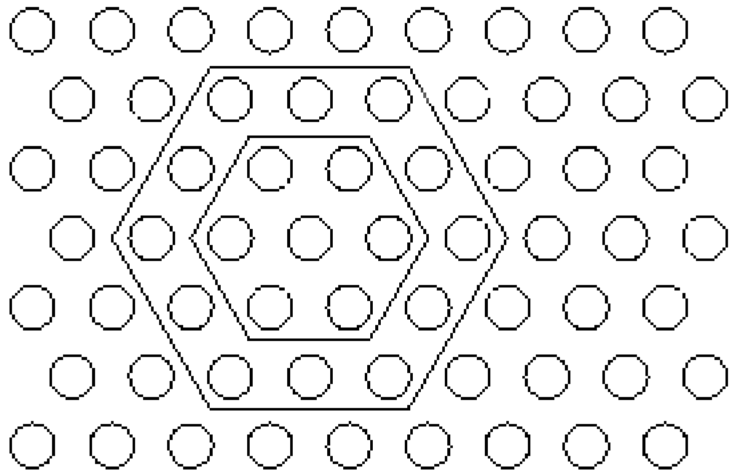

hexagonal grid

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

Self-Organizing Maps: Neighborhood

Neighborhood of the winner neuron

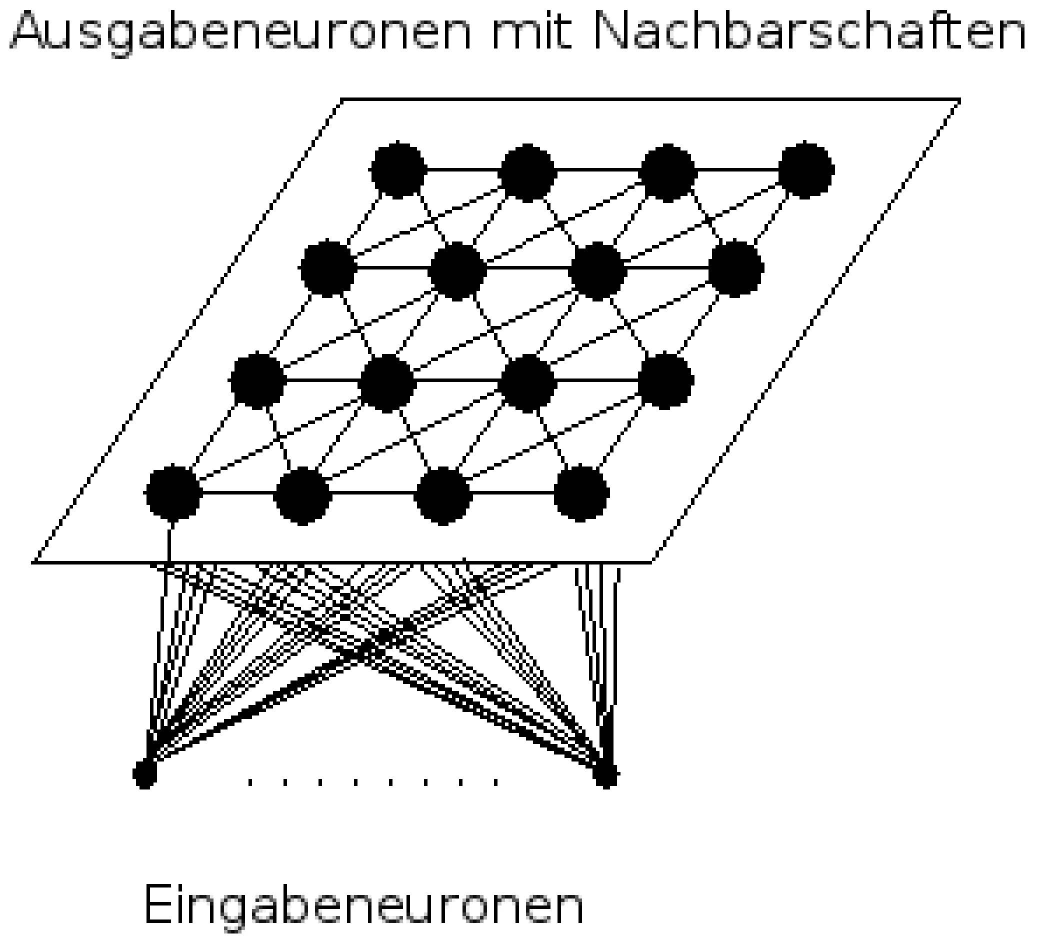




The neighborhood radius is decreasing by time.

Self-Organizing Maps: Structure

the „map“ shows the output neurons and their neighbors.



Self-Organizing Maps: Structure

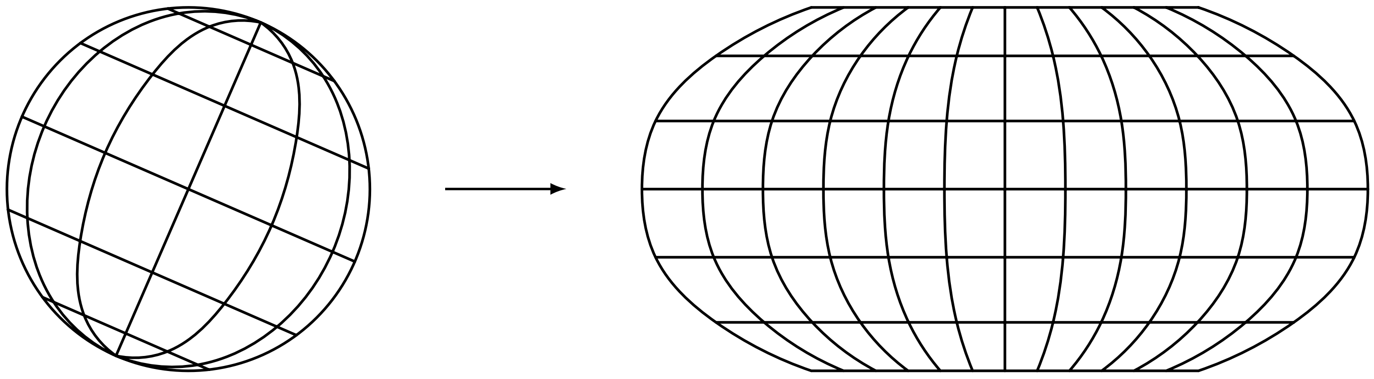
The process of SOM-learning

1. initialize the SOM's weight vectors
2. randomly choose the input vector from the training set
3. determine winner neuron according to distance function
4. determine time-dependent radius of the neighboring neurons within the radius of the winner neuron
5. tune these neighboring neurons time-dependently, then follow step 2 again.

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.
- $\rightarrow$ a SOM carries out a topology-preserving mapping

Self-Organizing Maps: Neighborhood

Find topology preserving mapping by respecting the neighborhood

Reference vector update rule:

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

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

Time dependent learning rate

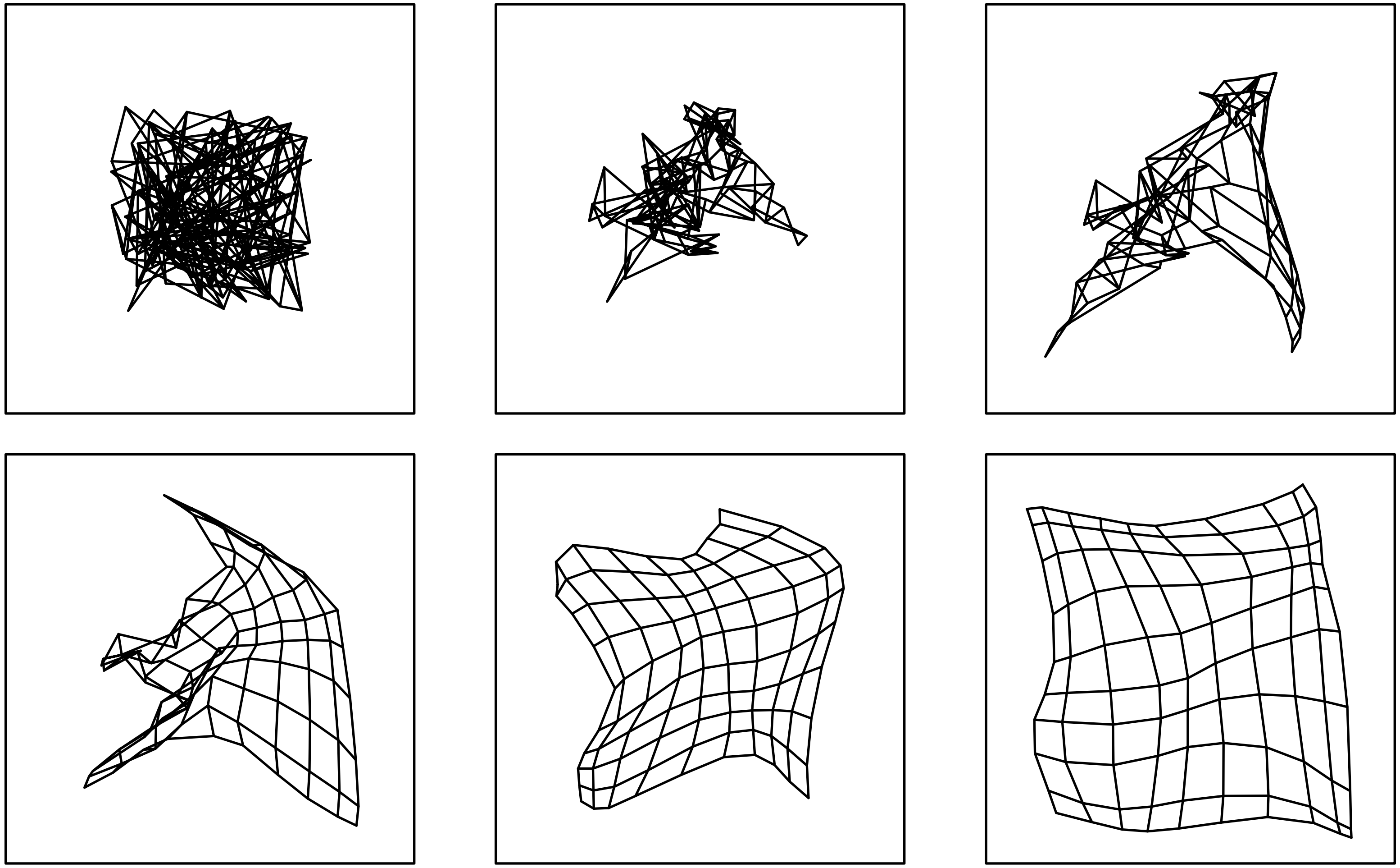
$$\eta(t) = \eta_0 \alpha_\eta^t, \quad 0 < \alpha_\eta < 1, \qquad \text{or} \qquad \eta(t) = \eta_0 t^{\kappa_\eta}, \quad \kappa_\eta > 0.$$

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.



The figures illustrate (left to right, top to bottom) the current state of the SOM (it's input space) after 10, 20, 40, 80 and 160 learning iterations. One training sample is being processed during each iteration.

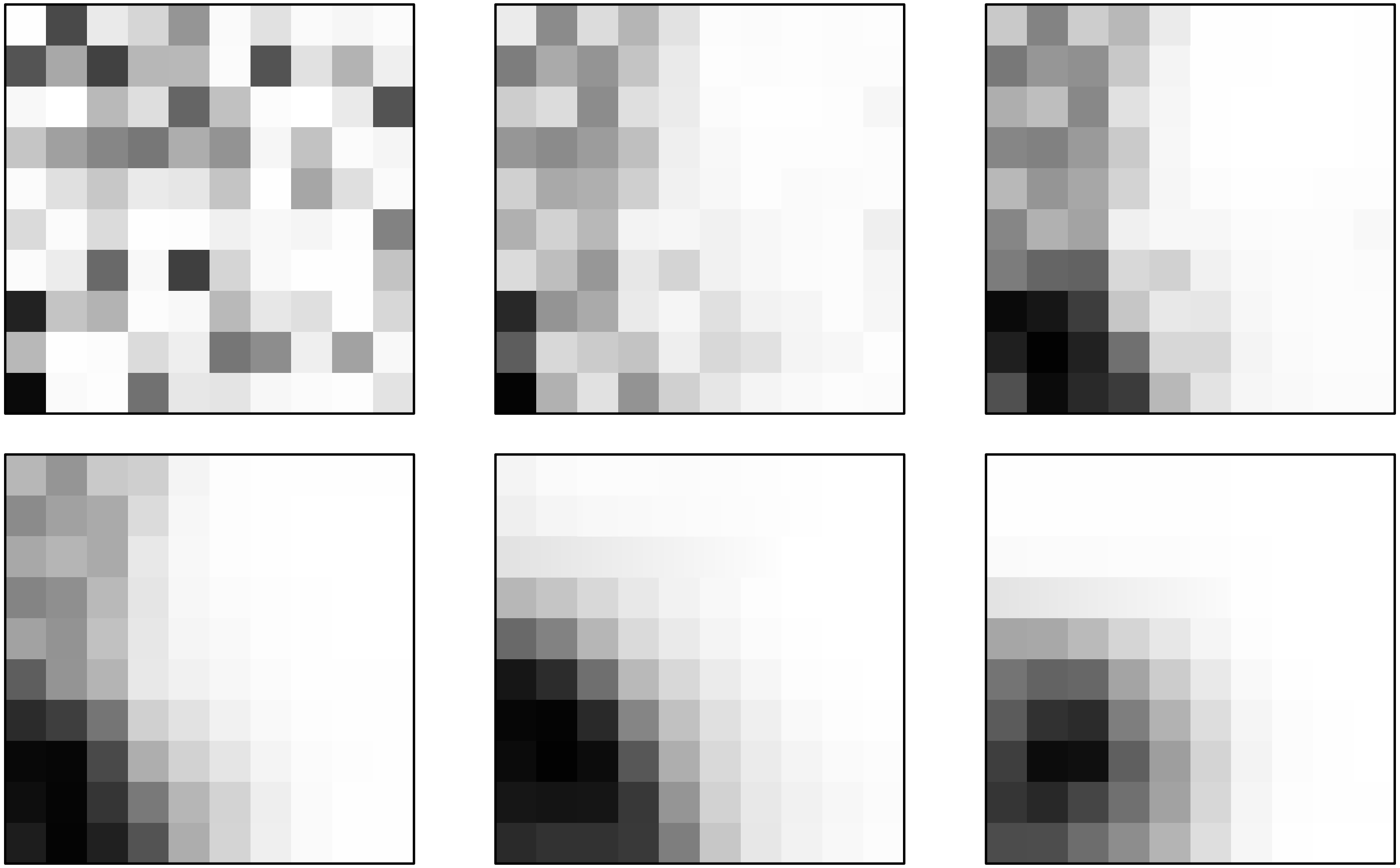
Self-Organizing Maps: Examples

Example: Unfolding a two-dimensional self-organizing map. (comments)

- Unfolding a 10x10-map trained by random samples $\in [-1, 1] \times [-1, 1]$
- initialization with reference vectors $\in [-0.5, 0.5]$
- lines connecting direct neighbors (grid)
- learning rate $\eta(t) = 0.6 * t$
- Gaussian neighborhood function f_{nb}
- radius $\rho(t) = 2.5 * t^{-0.1}$

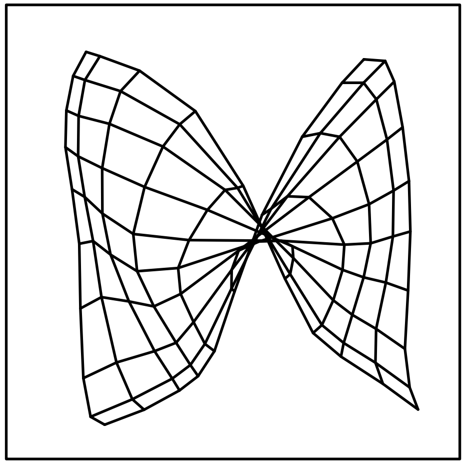
Self-Organizing Maps: Examples

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



Self-Organizing Maps: Examples

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

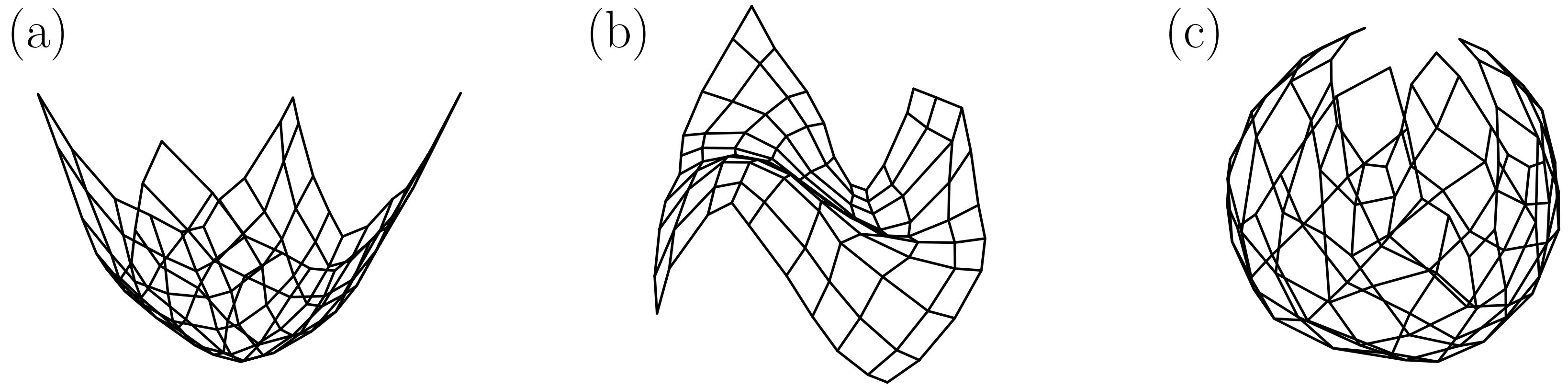


Training a self-organizing map may fail because of

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

Self-Organizing Maps: Examples

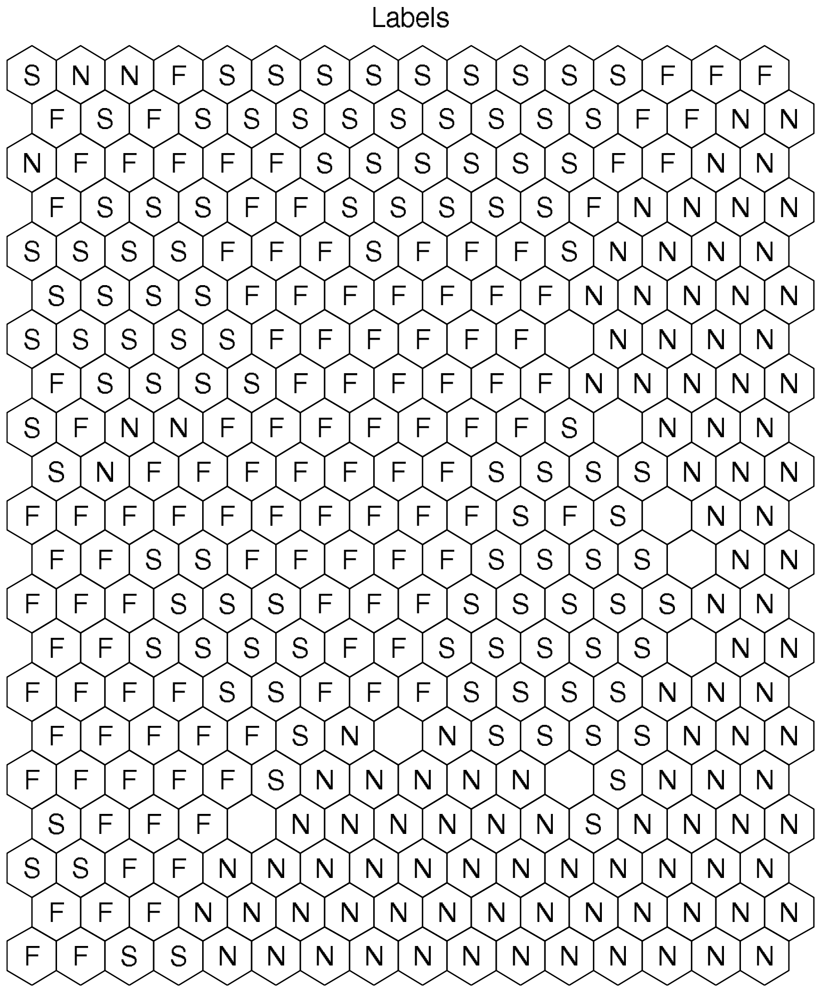
Example: Unfolding of a two-dimensional self-organizing map.

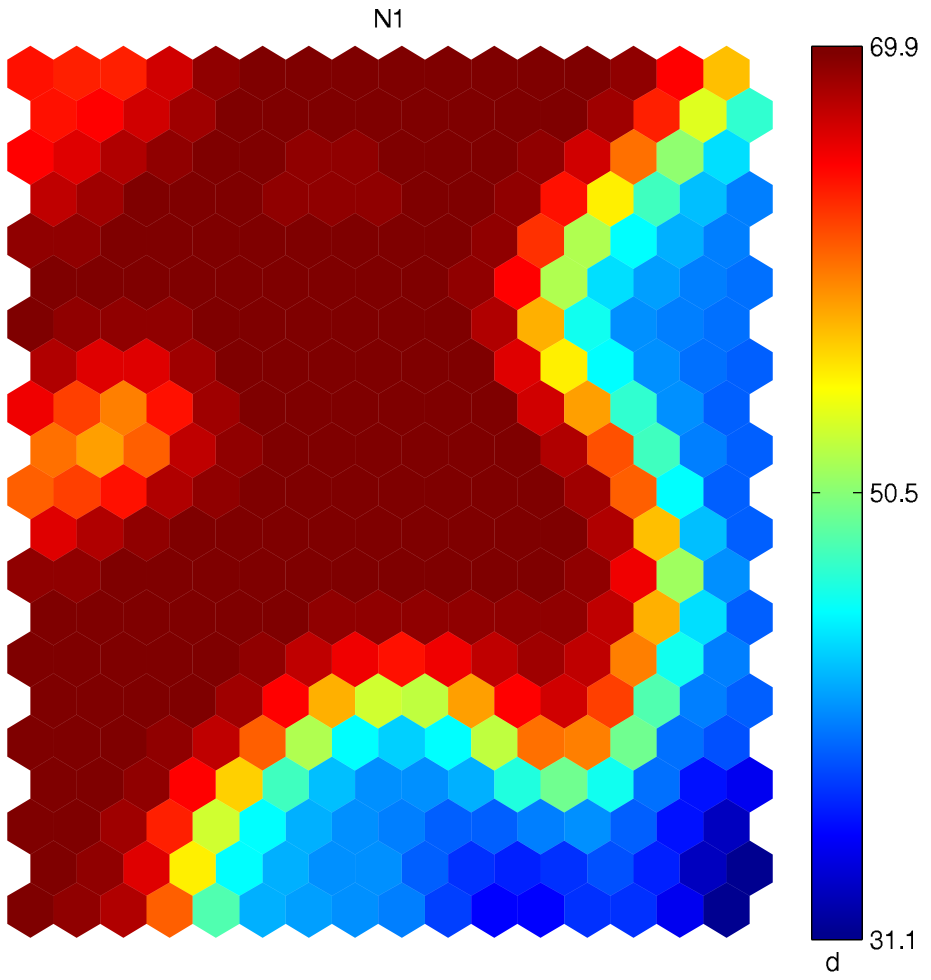


Self-organizing maps that have been trained with random points from
(a) a rotation parabola, (b) a simple cubic function, (c) the surface of a sphere.

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

SOM, Example Clustering of Strategies for editing fields


Labels
S N N F S S S S S S S S S F F F
F S F S S S S S S S S S F F N N
N F F F F F S S S S S S F F N N
F S S S F F S S S S S F N N N N
S S S S F F F S F F F S N N N N
S S S S F F F F F F F N N N N N
S S S S S F F F F F F N N N N
F S S S S F F F F F F N N N N N
S F N N F F F F F F F S N N N
S N F F F F F F F S S S S N N N
F F F F F F F F F F S F S N N
F F S S F F F F F S S S S N N
F F F S S S F F F S S S S S N N
F F S S S S F F S S S S S N N
F F F F S S F F F S S S S N N N
F F F F F S N N S S S S N N N
F F F F F S N N N N N S N N N
S F F F N N N N N N S N N N N
S S F F N N N N N N N N N N N N
F F F N N N N N N N N N N N N N
F F S S N N N N N N N N N N N N



Left: self-organizing map including class labels
Right: a variable that has been used for learning

SOM, Phoneme mape of Finnish

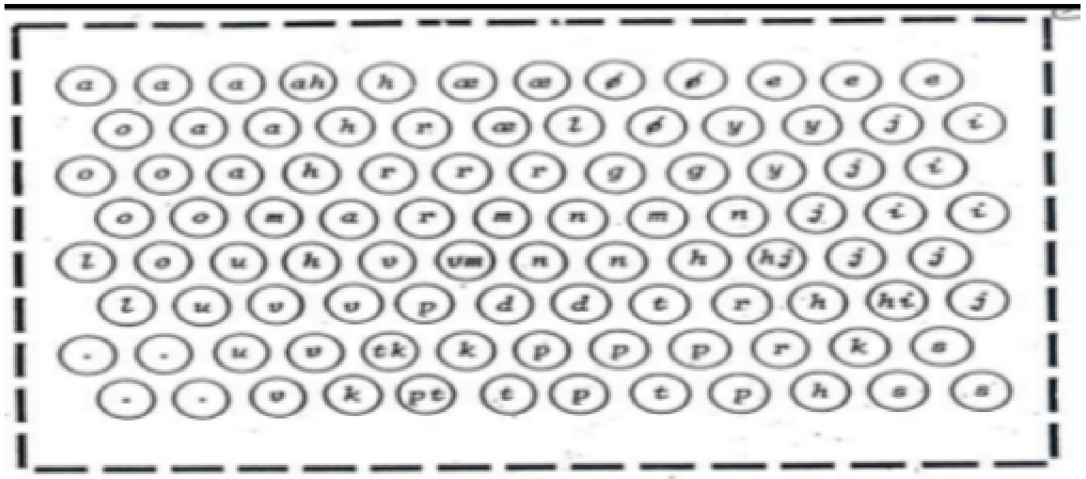


Abb. 2.6.7 Phonemkarte des Finnischen (nach [KOH88])

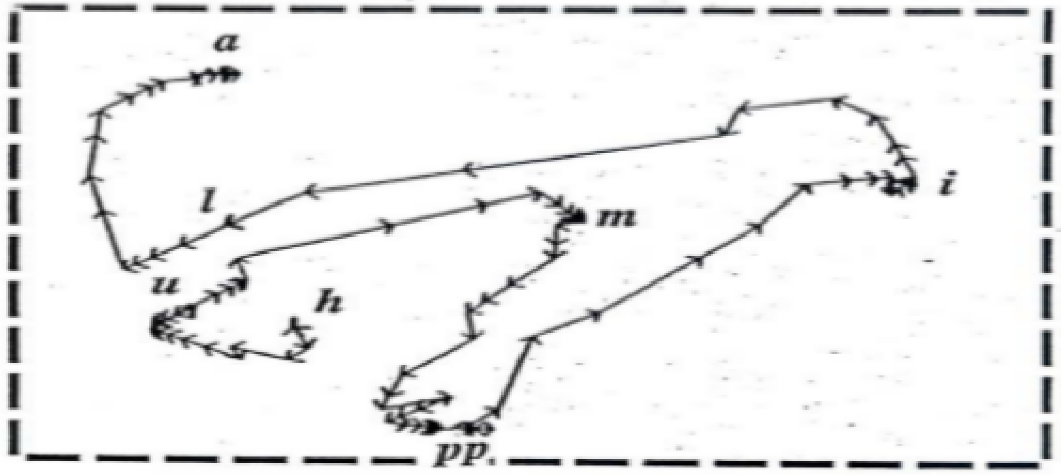
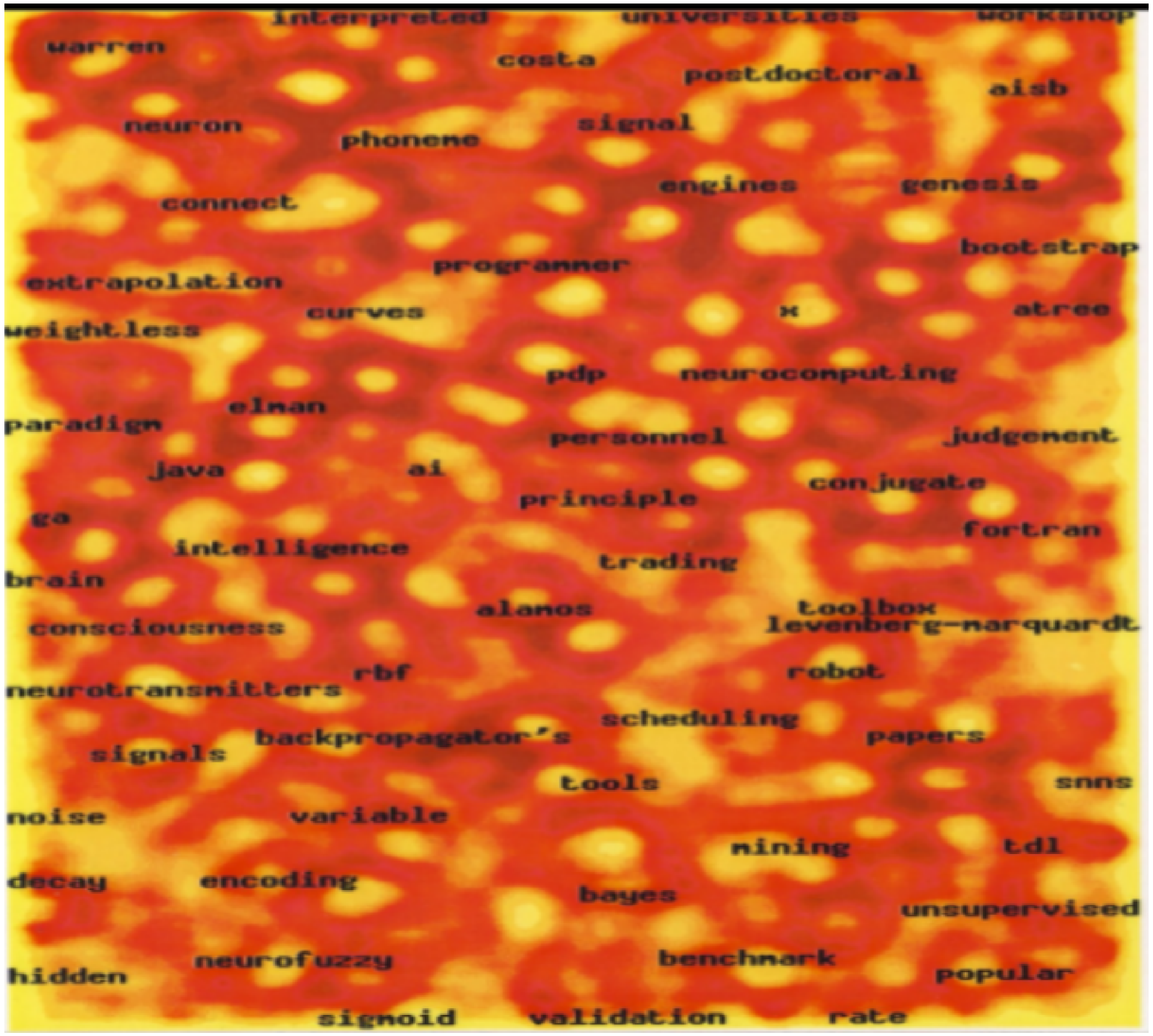
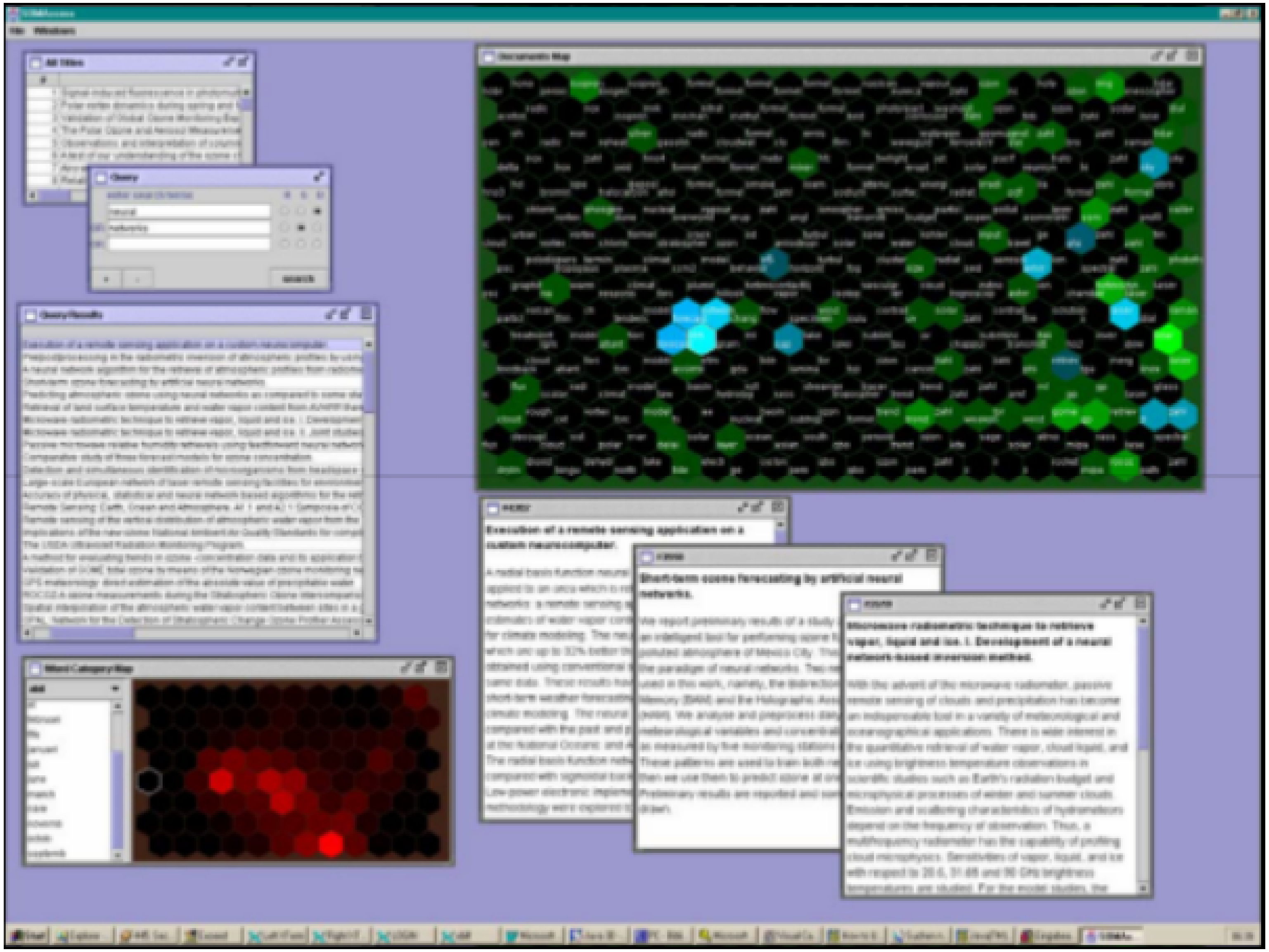


Abb. 2.6.8 Phonemsequenz für */humppila/* (nach [KOH88])

SOM, 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
principle
conjugate
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
popular
hidden
sigmoid
validation
rate

SOM, Websom



Execution of a remote sensing application on a custom neurocomputer.

[illegible]

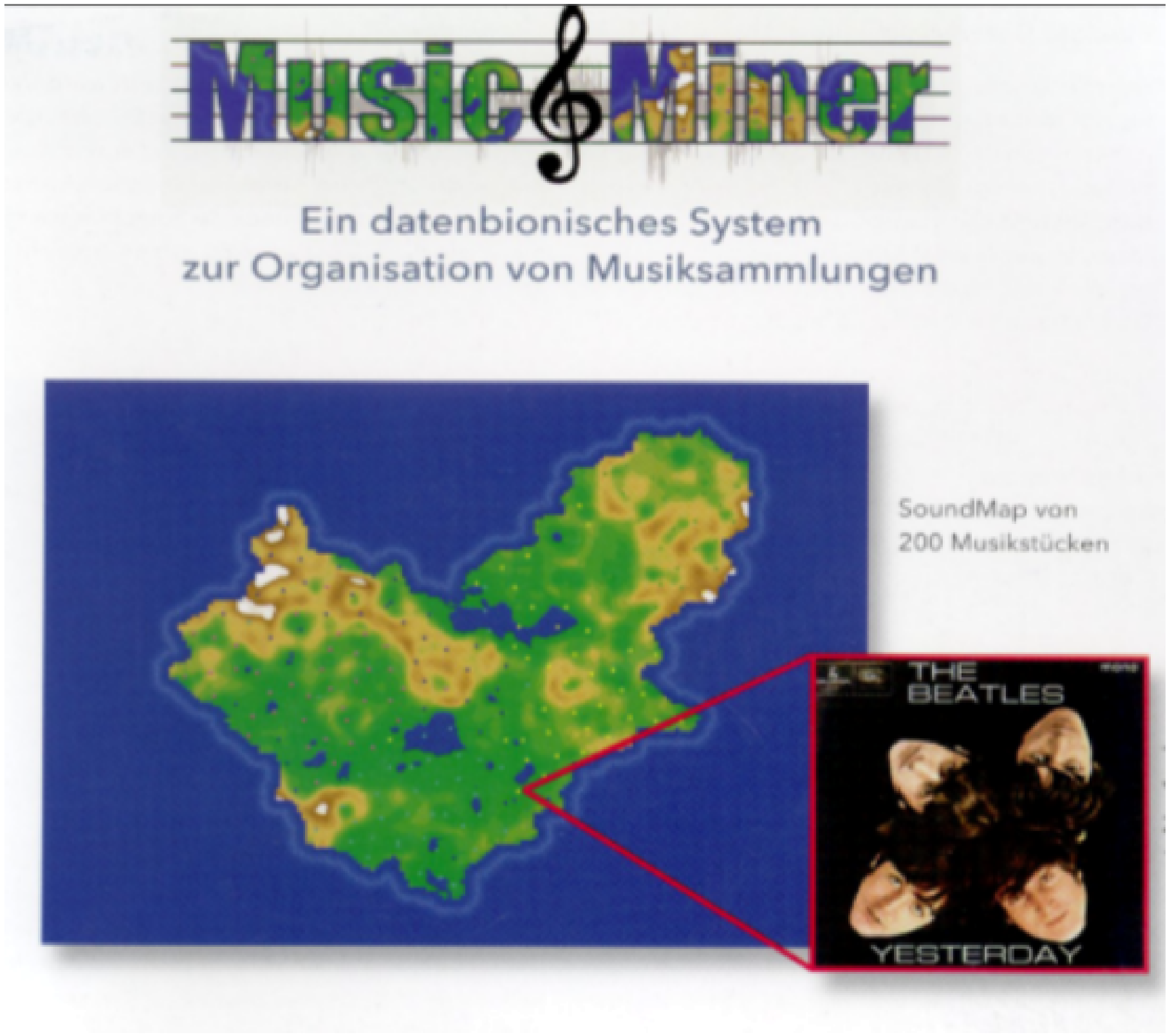
Short-term ozone forecasting by artificial neural networks.

[illegible]

Microwave radiometric technique to retrieve vapor, liquid and ice. I. Development of a neural network-based inversion method.

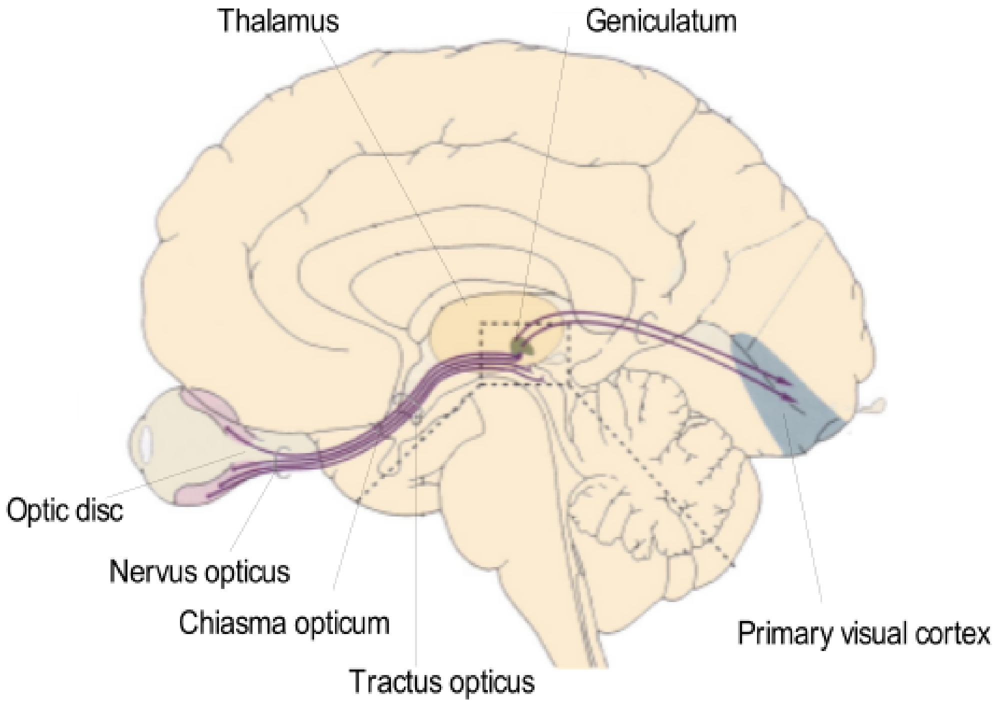
[illegible]

SOM, MusicMiner

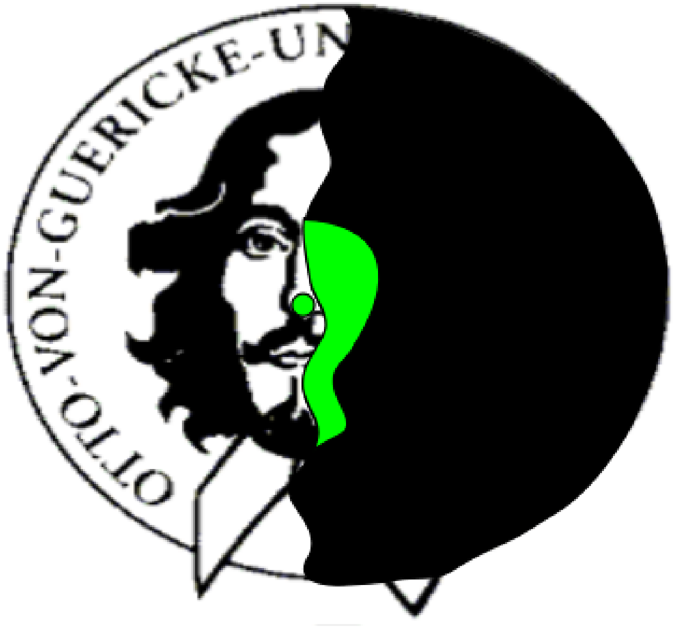

MusicMiner
Ein datenbionisches System
zur Organisation von Musiksammlungen
SoundMap von
200 Musikstücken
THE
BEATLES
YESTERDAY

SOM, Lack of visual field

Gesichtsfeldausfall



E.Kandel, J. Schwartz, T. Jessell: Neurowissenschaften, 1996.

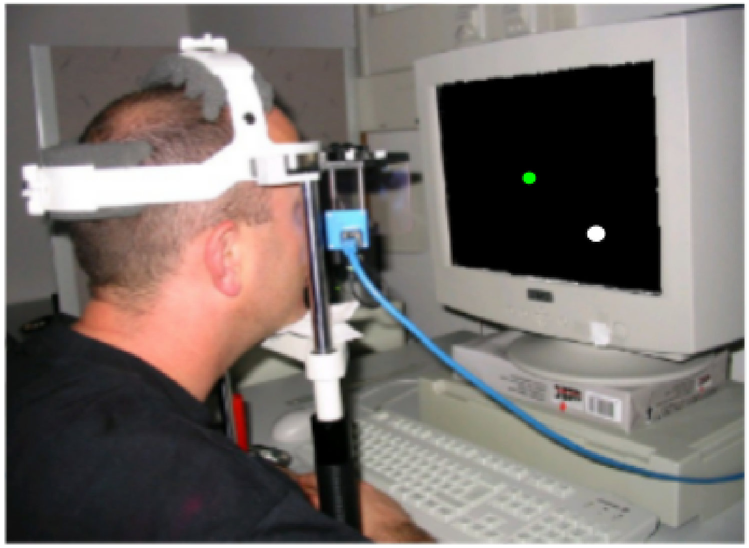


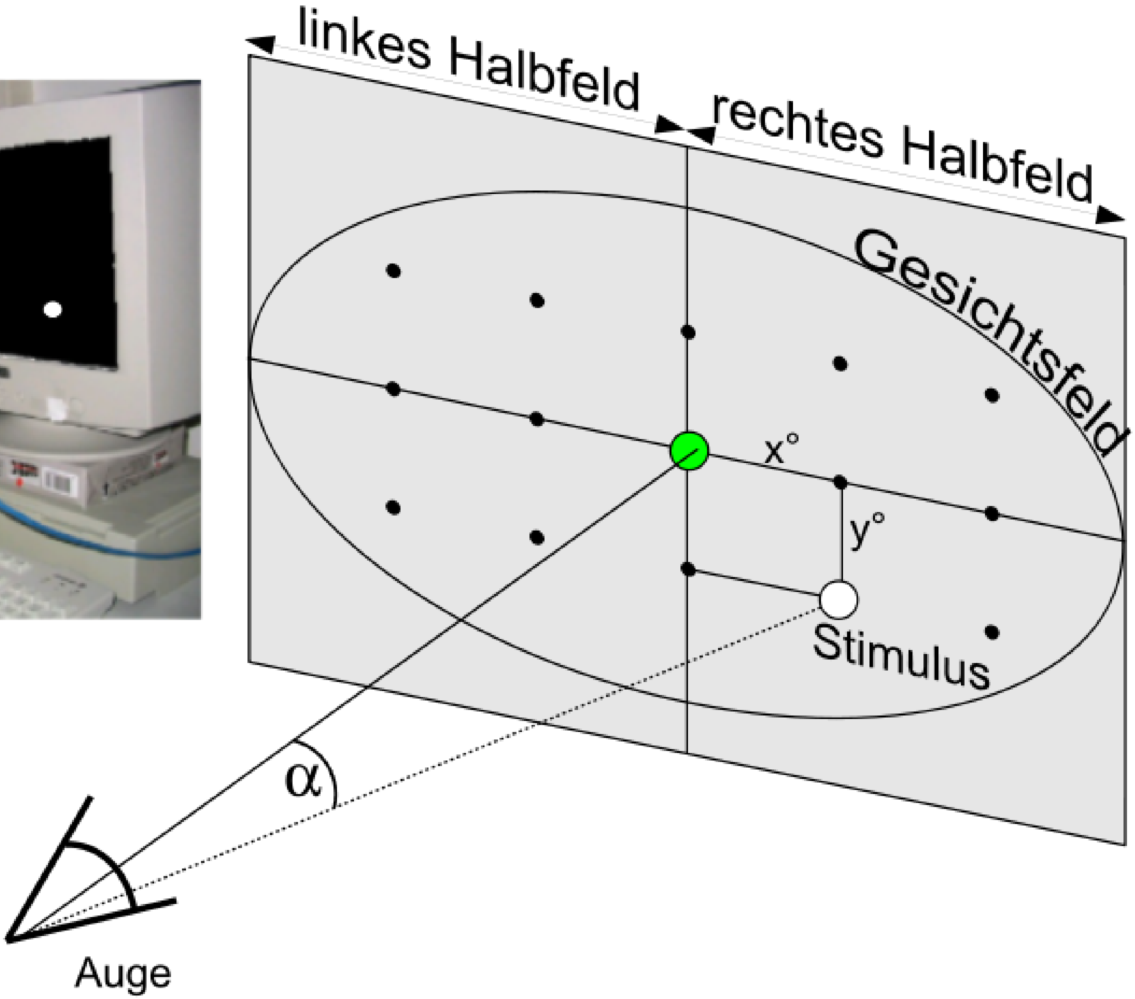
Visuelle Restitutionstherapie
(6 Monate à 1h pro Tag)

E. Kasten, S. Wuest, W. Behrens-Baumann, and B. A. Sabel. Computerbased training for the treatment of partial blindness. Nat Med, 4(9):1083–7,1998.

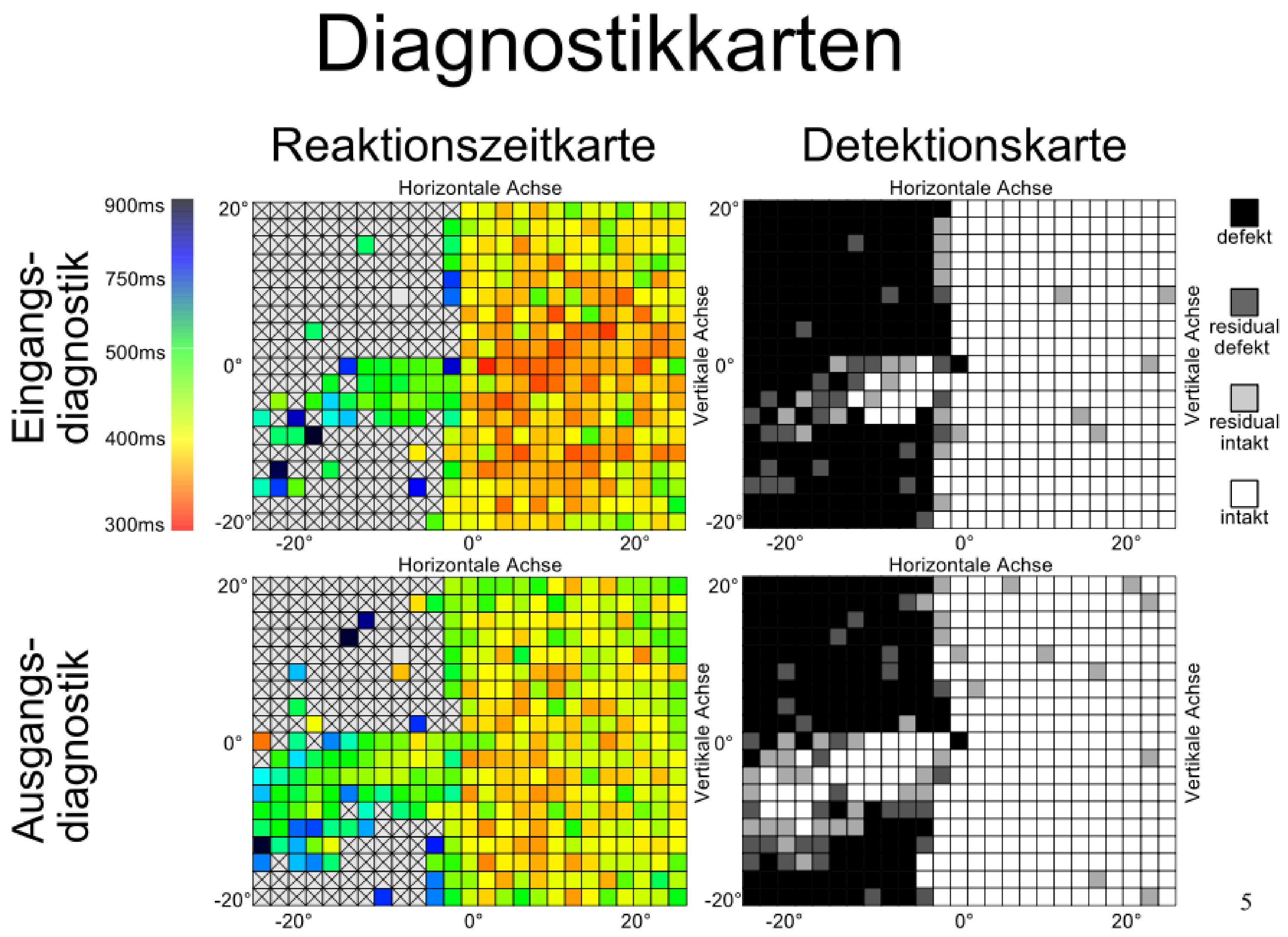
SOM, Lack of visual field

Gesichtsfelddiagnostik

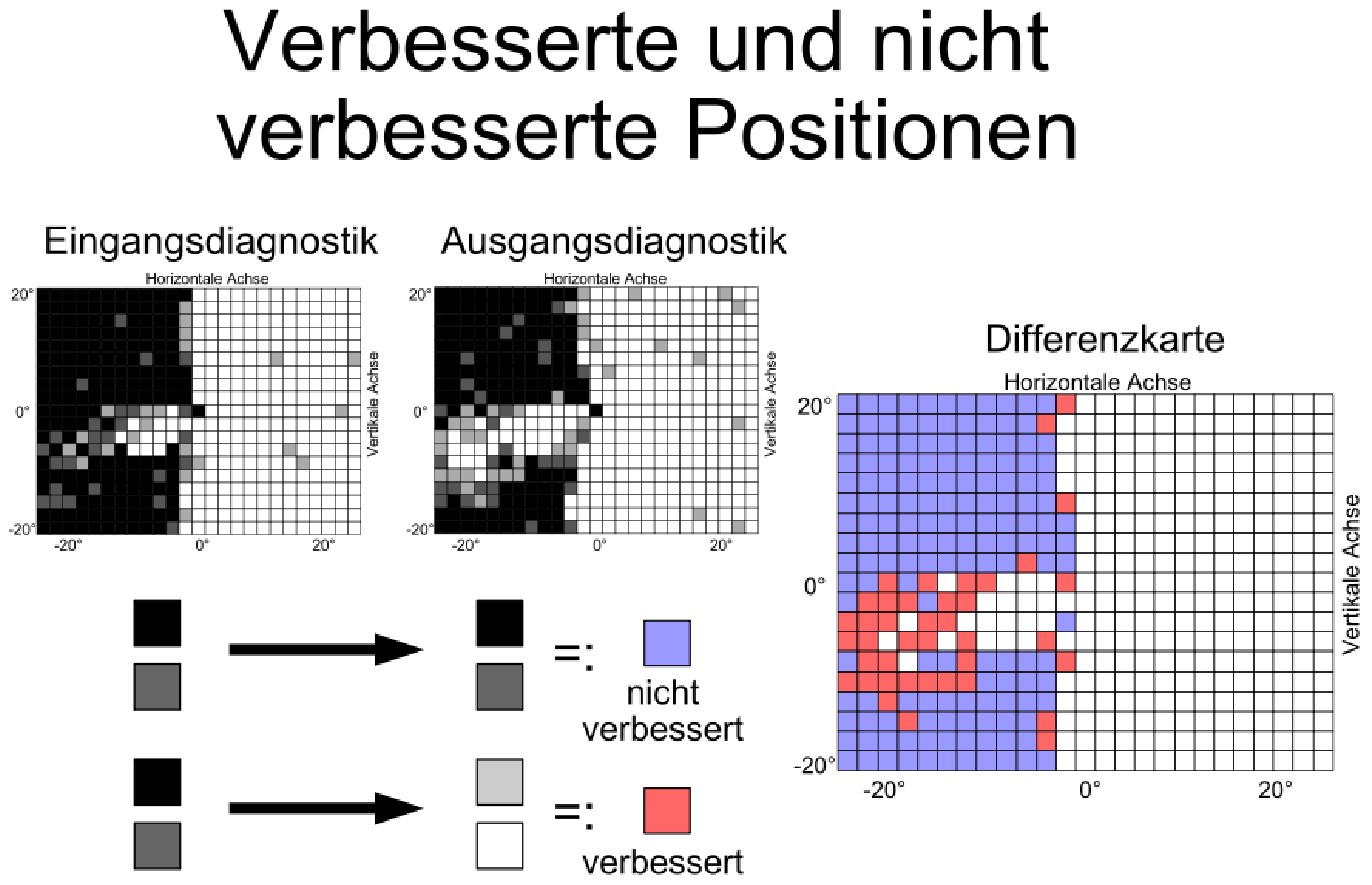




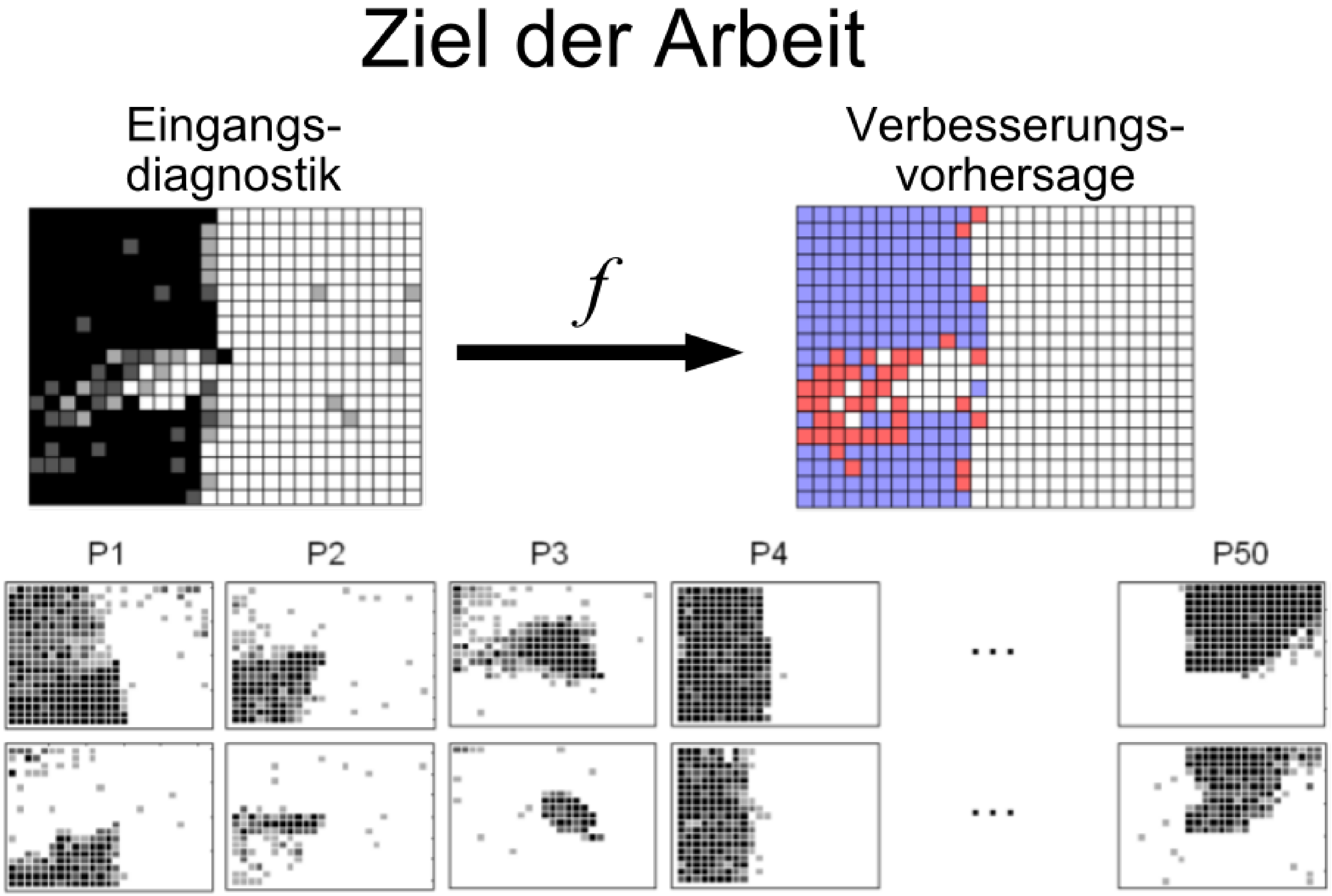
SOM, Lack of visual field



SOM, Lack of visual field



SOM, Lack of visual field


Ziel der Arbeit
Eingangs-diagnostik
Verbesserungs-vorhersage
f
P1
P2
P3
P4
P50
...