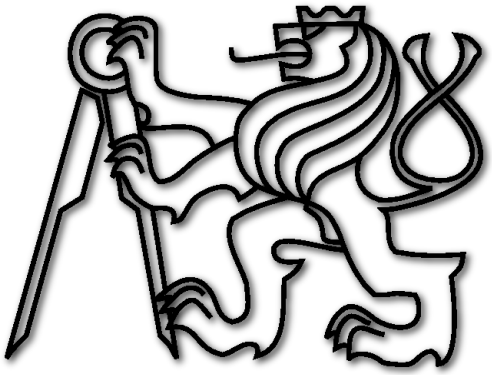
Artificial Neural Networks
SOM, RBF & GMDH

Jan Drchal
drchajan@fel.cvut.cz



Computational Intelligence Group
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague

Self Organising Map: SOM

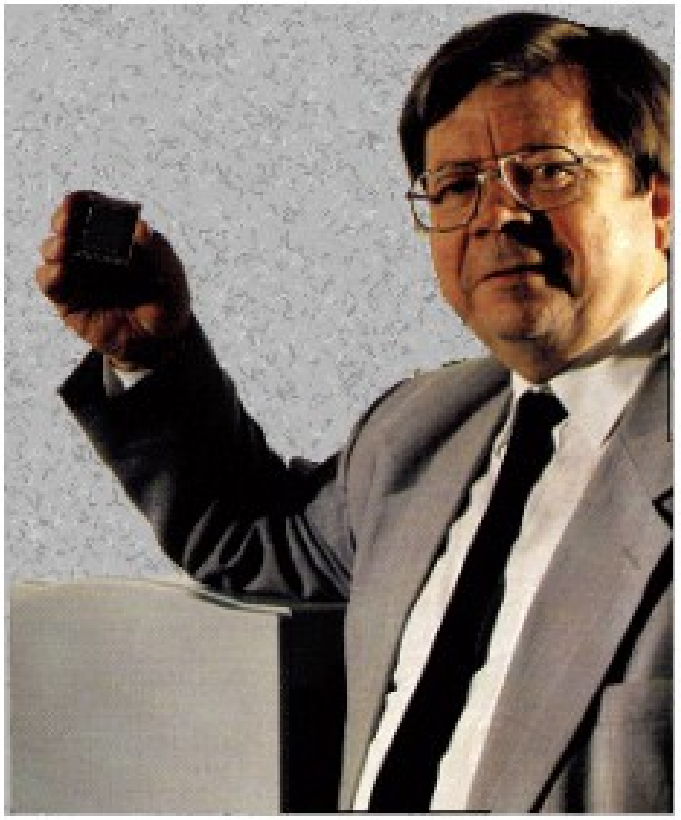
Competitive Learning

- Nature inspired.
- No arbiter needed – unsupervised learning.
- Individuals (units, neurons) learn from examples.
- System **self-organizes**.
- Now we are going to apply this to **cluster analysis**.

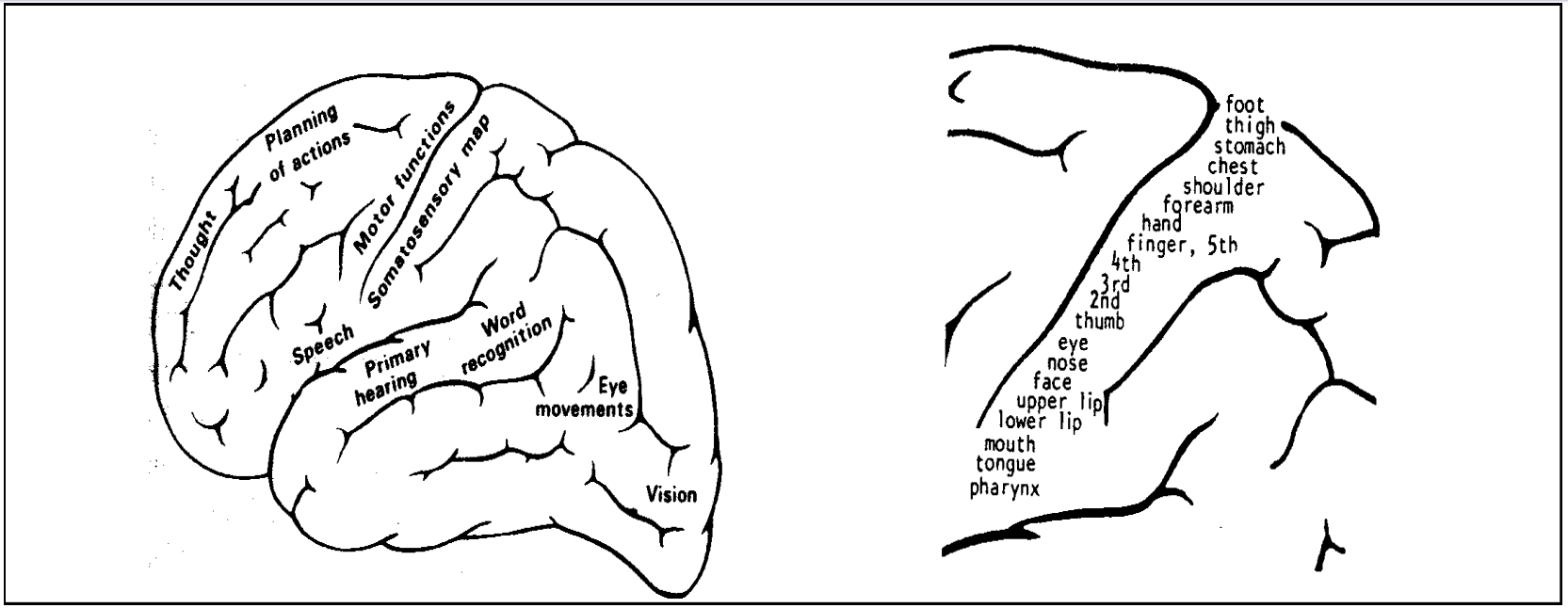
SOM

- SOM = Self Organizing Maps.
- Prof. Teuvo Kohonen, Finsko,

  TU Helsinki, 1981, several thousands scientific publications since...

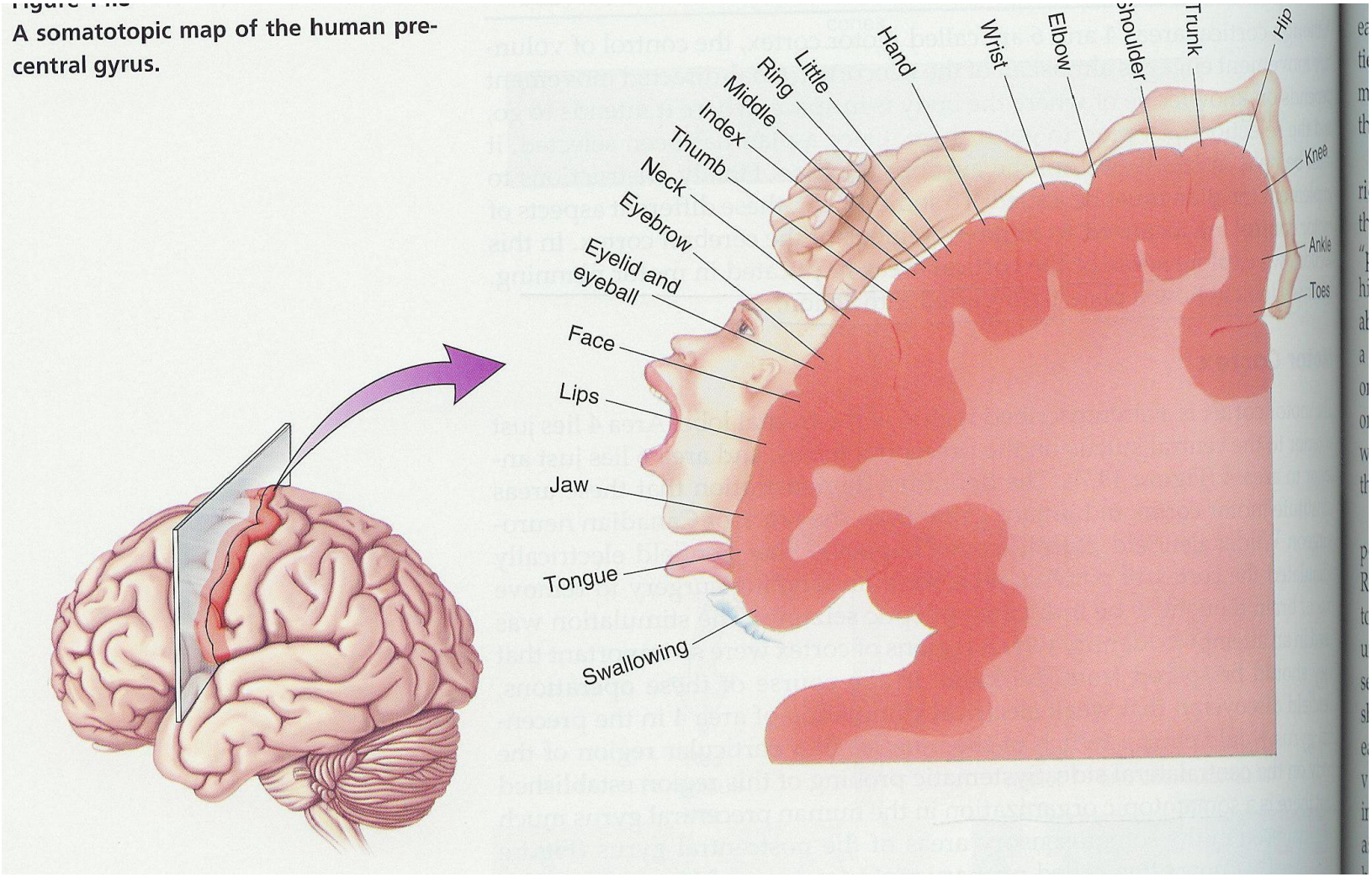


SOM Inspiration



- Brain represents the world in a **topological way**.
- Exterior spatial relations are mapped to similar spatial relations in the brain:
  - i.e. signals from hand and arm are processed nearby.

SOM Inspiration II



Bear, Connors & Paradiso (2001). *Neuroscience: Exploring The Brain*. Pg. 474.

SOM Overview

- Single layer, feed-forward.
- Unsupervised, **self-organization**.
- No output, instead **Winner-takes-all**.
- Used for **cluster analysis**.
- Performs **vector quantization**.
- Not a classifier!
  - But can be simply transformed into one by adding another layer.

What is Self-Organization?

- Self-organization of a system is a process which leads to a rise of a quality of its inner configuration while not using any information from outside.
- Self-organization clears up relationships between parts of a system.

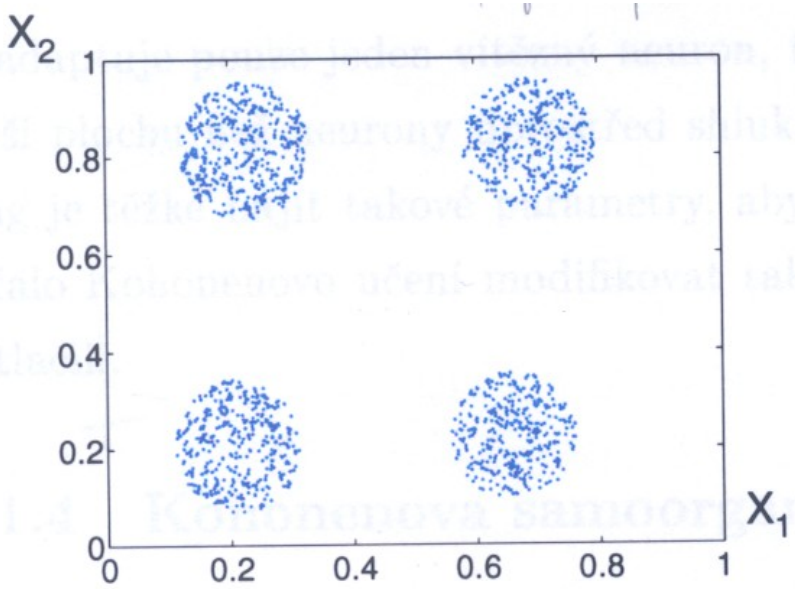
What is Vector Quantization

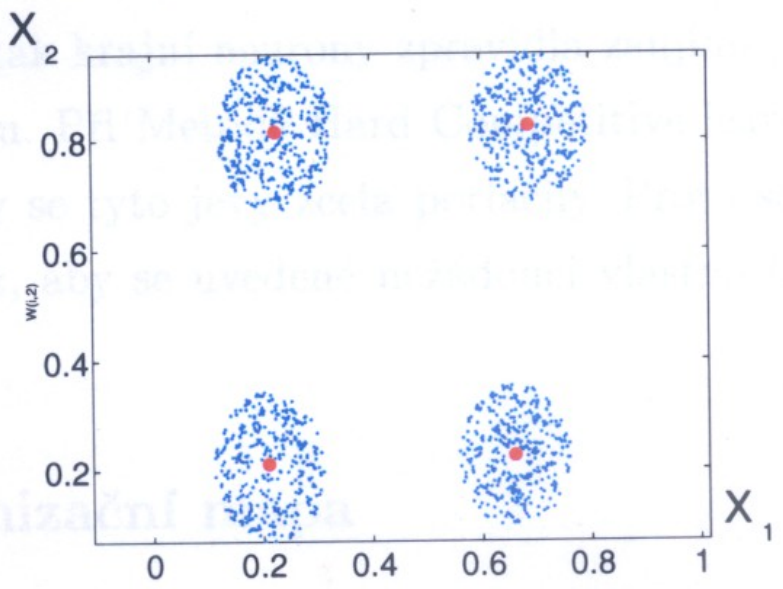
The goal of Vector Quantization is to approximate the probability density $p(x)$ of real input vectors $\boldsymbol{x} \in \boldsymbol{R}^n$ distribution using finite number of representatives $\boldsymbol{w}_i \in \boldsymbol{R}^n$.

The representative vectors tend to drift there where the data is dense, while there tends to be only a few of them where data is sparsely located. In this manner, the net tends to approximate the probability density of the input data. *Hollmen '96*

Vector Quantization Example

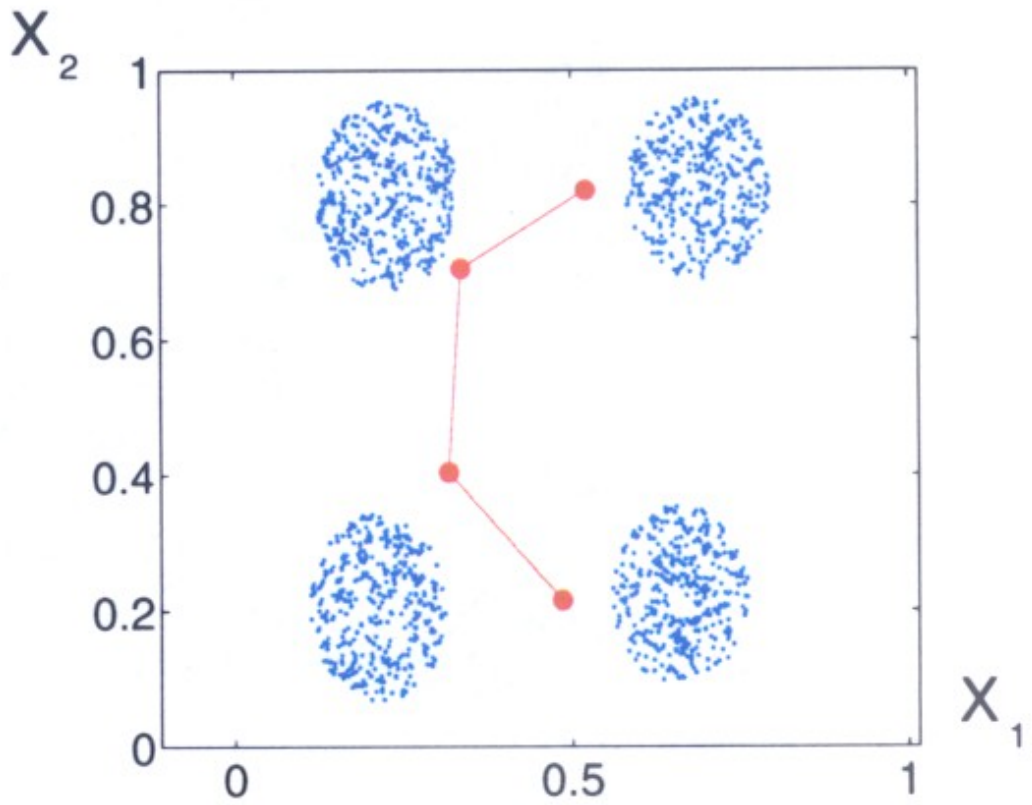
Blue points are the input vectors.





Red points are the representatives.

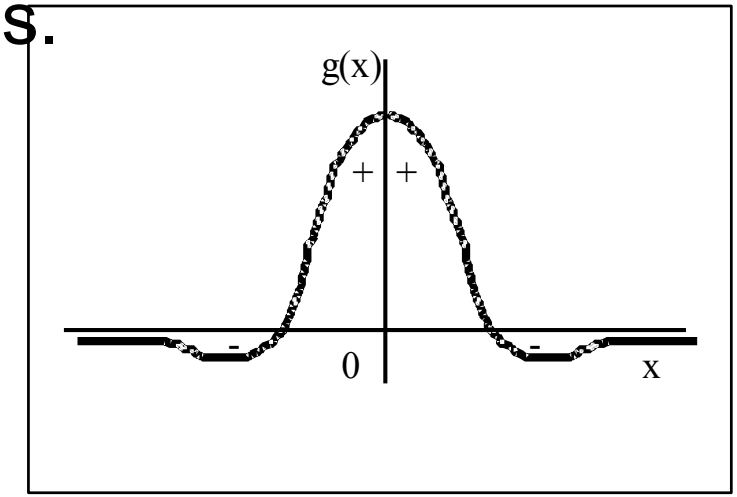
VQ by SOM



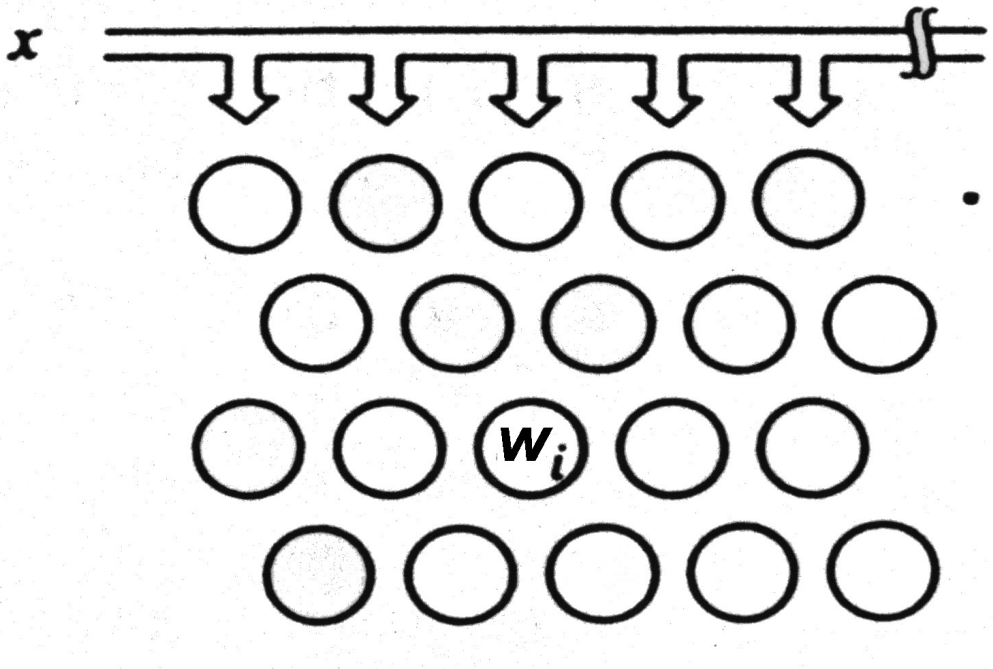
1D SOM of 4 neurons

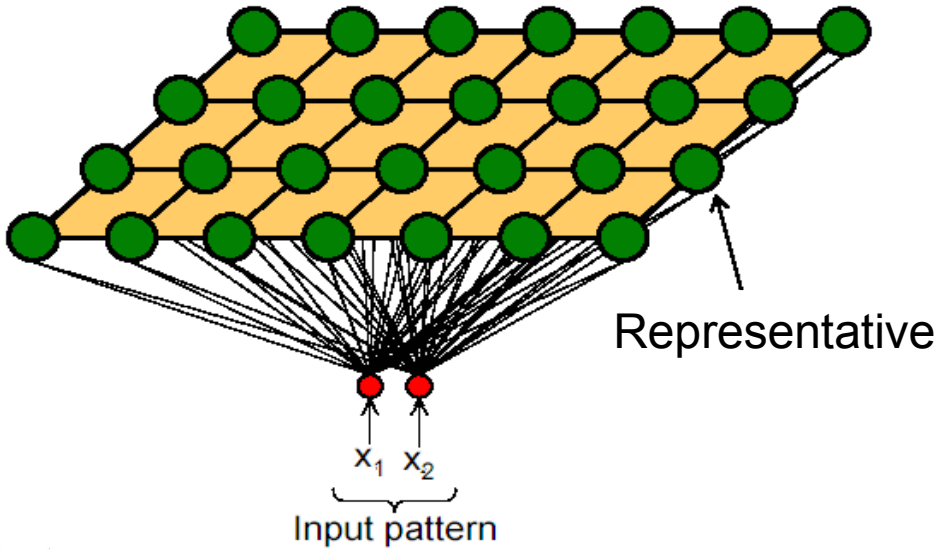
Why the Different Result?

- SOM works with neighbourhood.
- Representatives influence each other.
- They form “elastic”:
  - chain for 1D SOM,
  - mesh for higher dimensions.



SOM Architecture 1/3





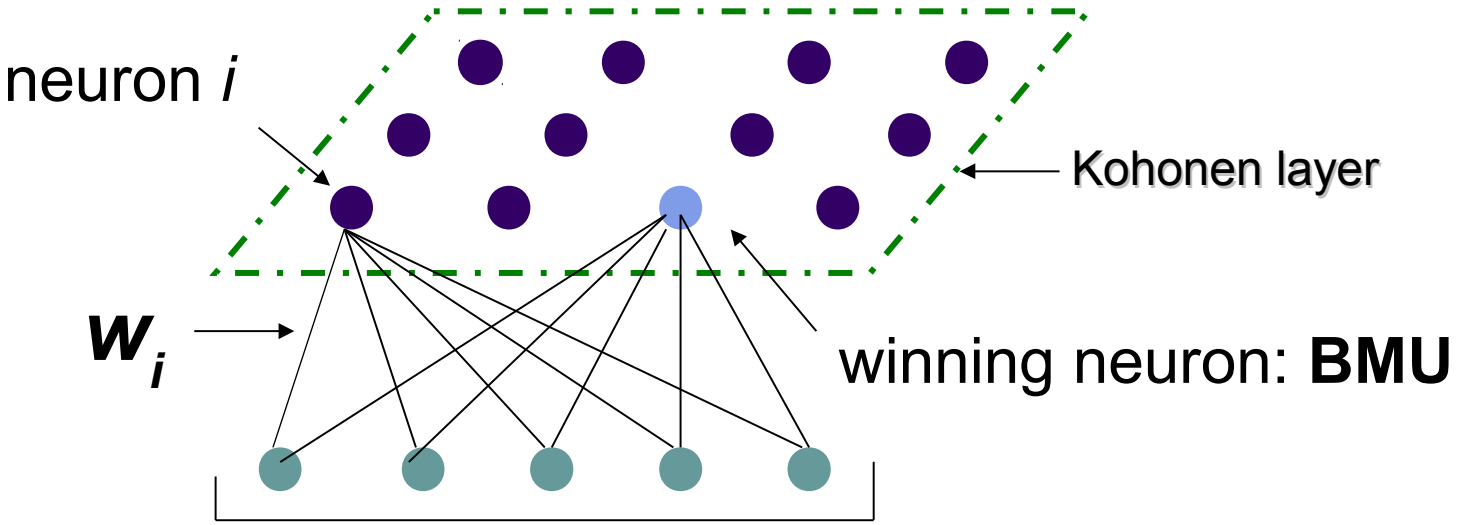
Typically: 2D mesh of representatives (neurons)

SOM Architecture 2/3

- Arrangements:
  - 1D linear quite often,
  - 2D mesh most frequently,
  - 3D (and higher dimensions) exceptionally – problematic visualization.
- The arrangement defines **neighbourhood** of a neuron.
- Kohonen suggests: rectangular SOM!

SOM Architecture 3/3

- Input vector ***x*** has a dimension *N*.
- Each neuron has a weight vector ***w*** of the same dimension *N*.
- Weight vectors of all neurons are compared to ***x***.
- The most similar is chosen → BMU (Best Matching Unit).
- BMU becomes a representative of vector ***x***.



SOM Neuron 1/2

Evaluates the similarity of input vector ***x*** and weight vector $\boldsymbol{w}_i$.

Similarity: i.e., Euclidean

The most similar neuron to a input vector is chosen (BMU):

$$j^* = \underset{i}{\operatorname{argmin}} \left\{ \| x - w_i \| \right\},$$

SOM neuron is a representative of a cluster.

SOM Neuron 2/2

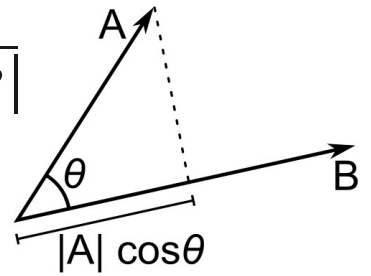
- Note, we don't have to use Euclidean distance.
- We can use directional similarity expressed by the dot product:

$$j^* = \arg\max_i \left\{ x^T(t) w_i(t) \right\} .$$

Why max here?

Note:

$$\cos\theta = \frac{A.B}{|A||B|}$$



http://en.wikipedia.org/wiki/Dot_product

Learning SOM

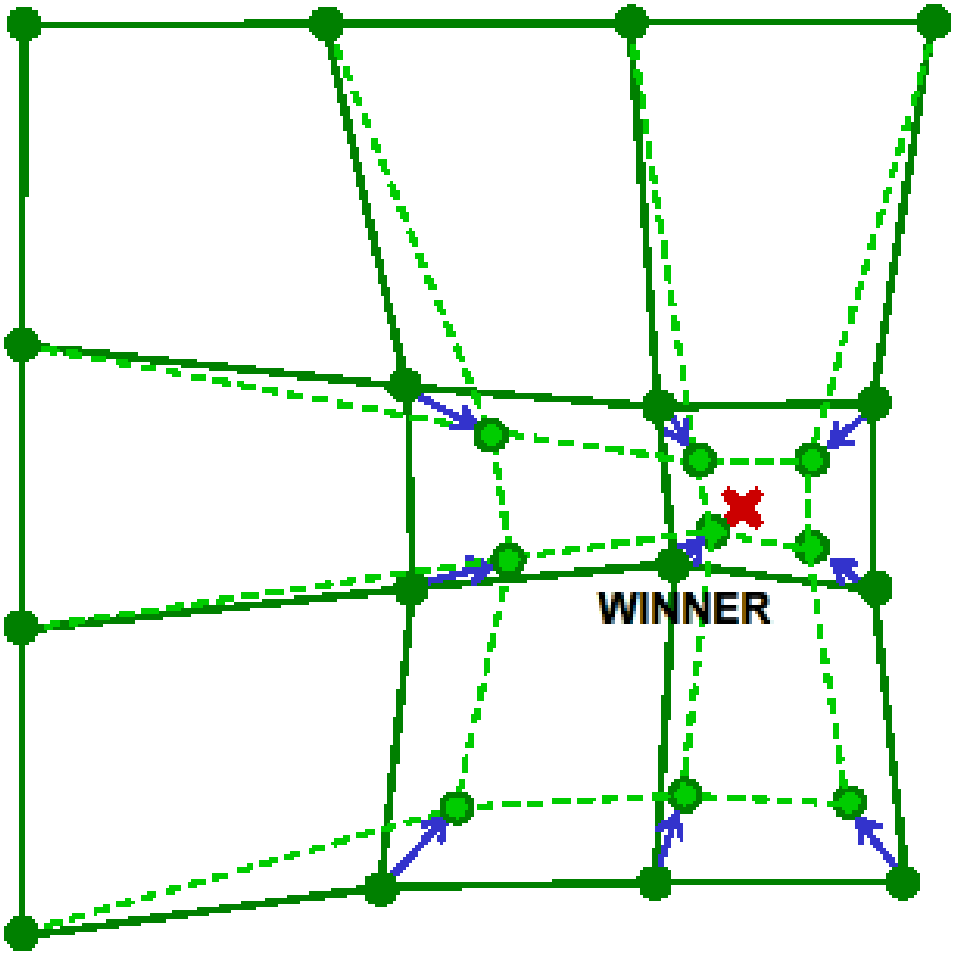
- Initialization (random weights).
- Apply input pattern $\boldsymbol{x} = (x_1, x_2, \ldots x_N)$.
- Compute distances.
- Select BMU – neuron *j*.
- Adjust weights for all neurons *i*:

$$w_i(t+1) = w_i(t) + \eta_{ij}(t)\left[x(t) - w_i(t)\right]$$

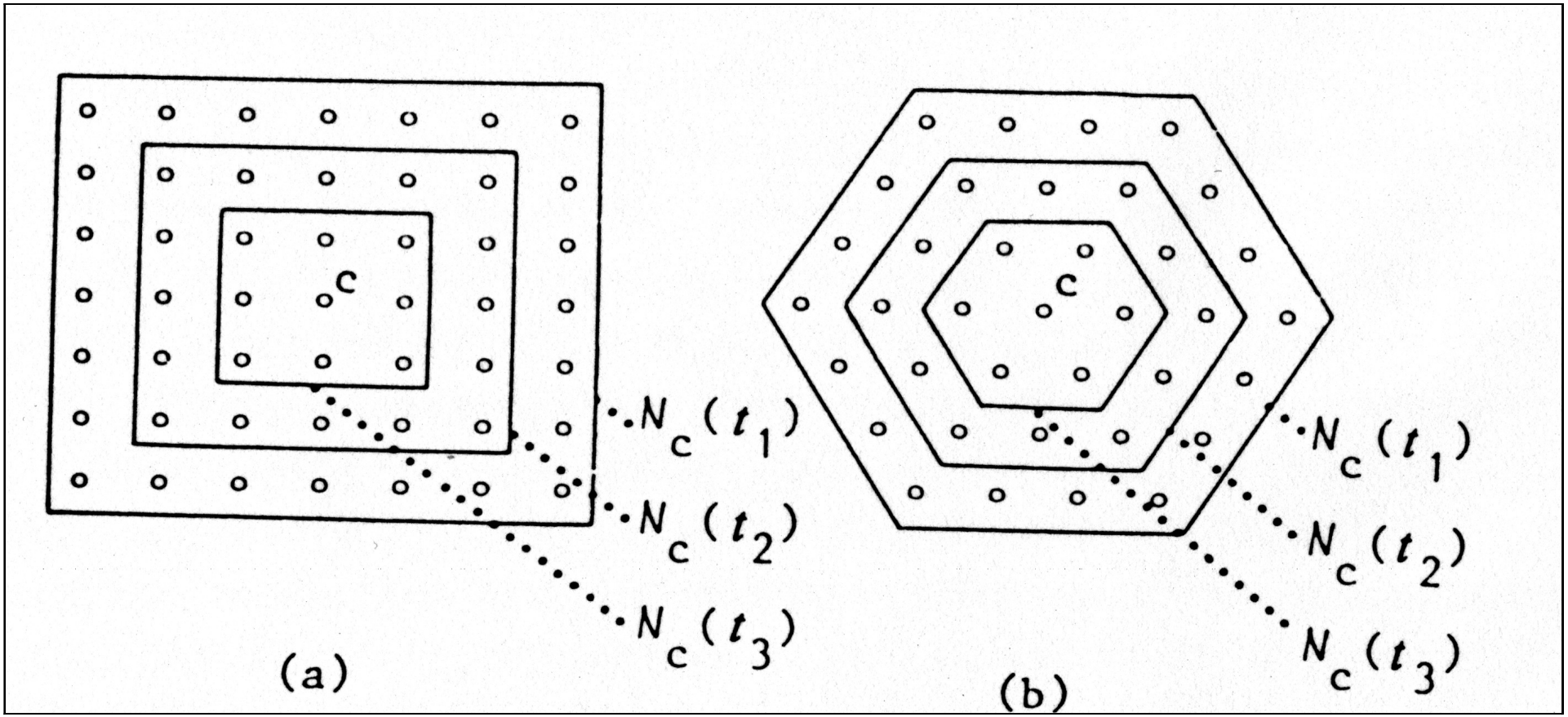
Neighbourhood function

- Continue with next pattern.

What About Updating Also Neurons in the Neighbourhood?



Common Neighbourhoods



square hexagonal

T. Kohonen: Self Organizing Maps

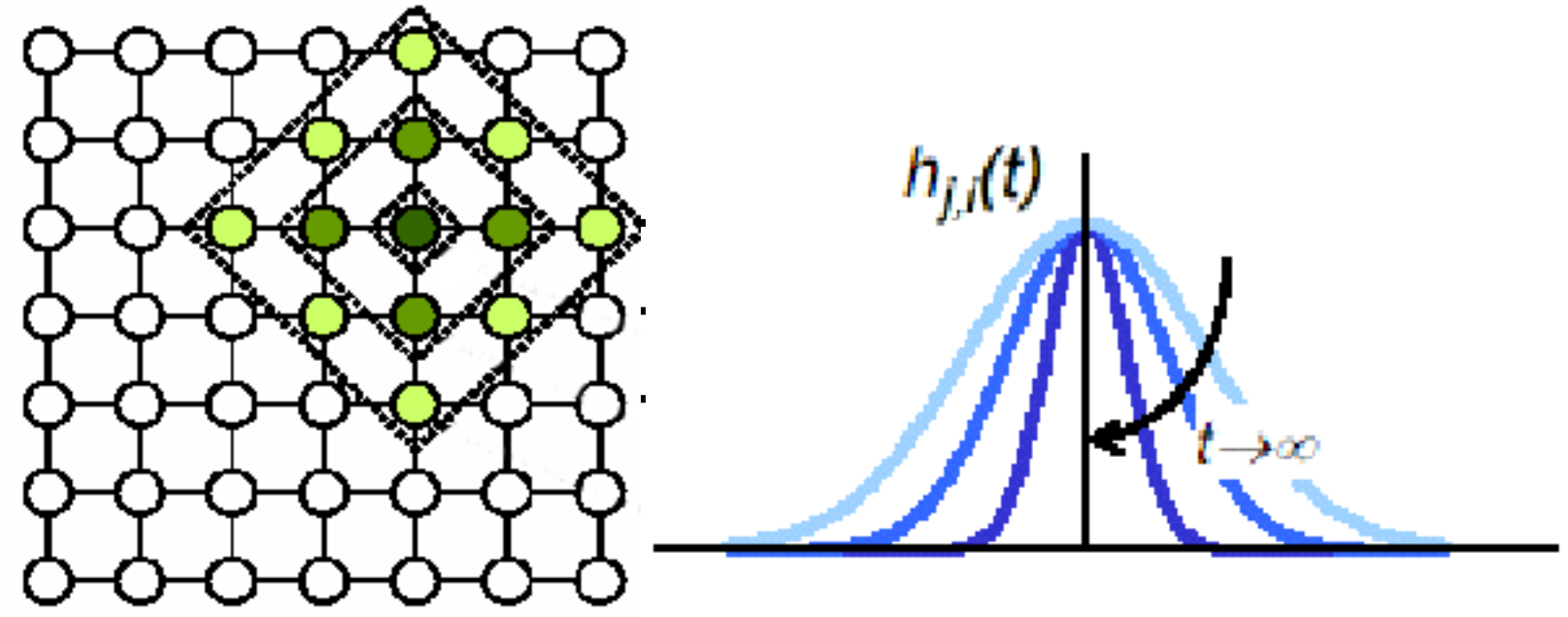
Gaussian Neighbourhood

- Neighbourhood function for neuron *i*.

$$\eta_{ij^*}(t) = \alpha(t) . \exp\left(-\frac{\|r_{j^*} - r_i\|^2}{2\sigma^2(t)}\right)$$

- Where j^* is the BMU,
  r the position of neuron in map,
  and function $\alpha(t)$: learning rate.
- The *exp* expression represents neighbourhood shape.

Gaussian Neighbourhood in Time

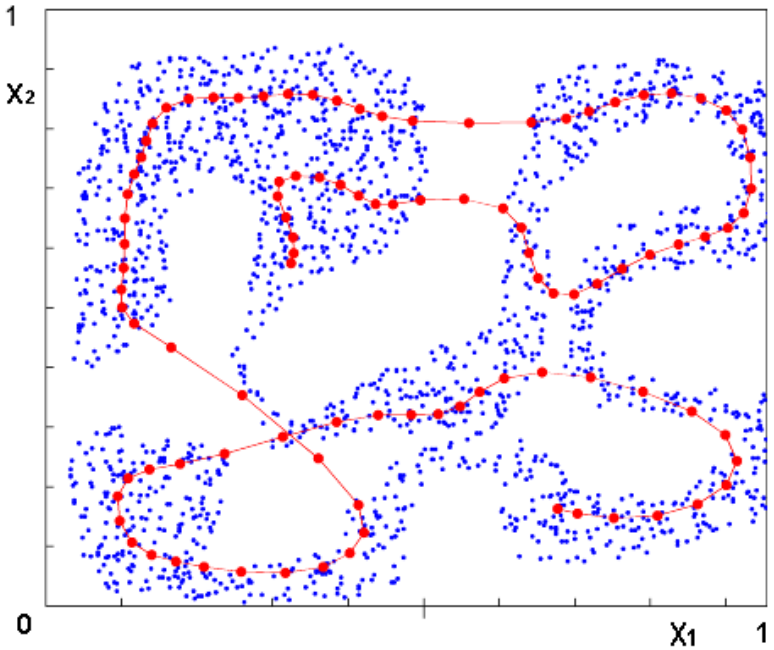


Distance related learning

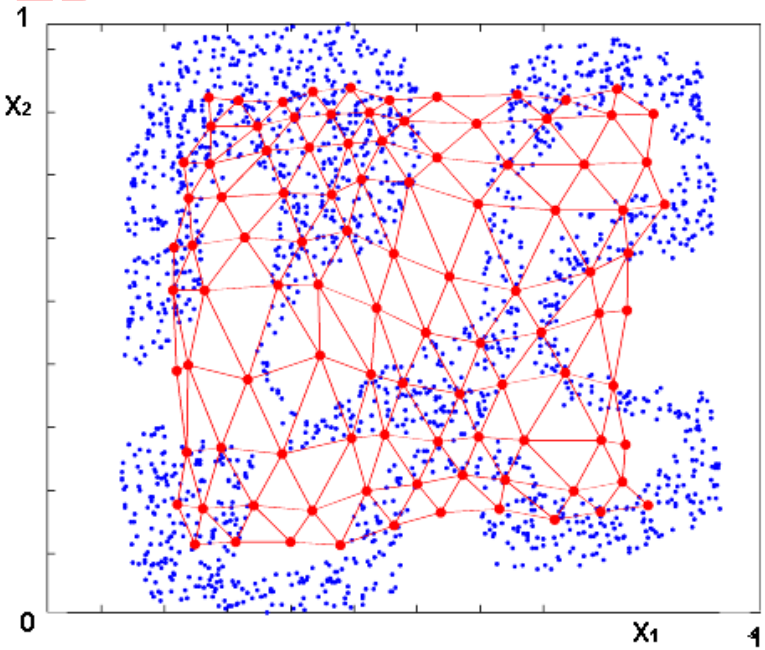
SOM Applications

- To visualize data.
- To cover the input space by representatives.

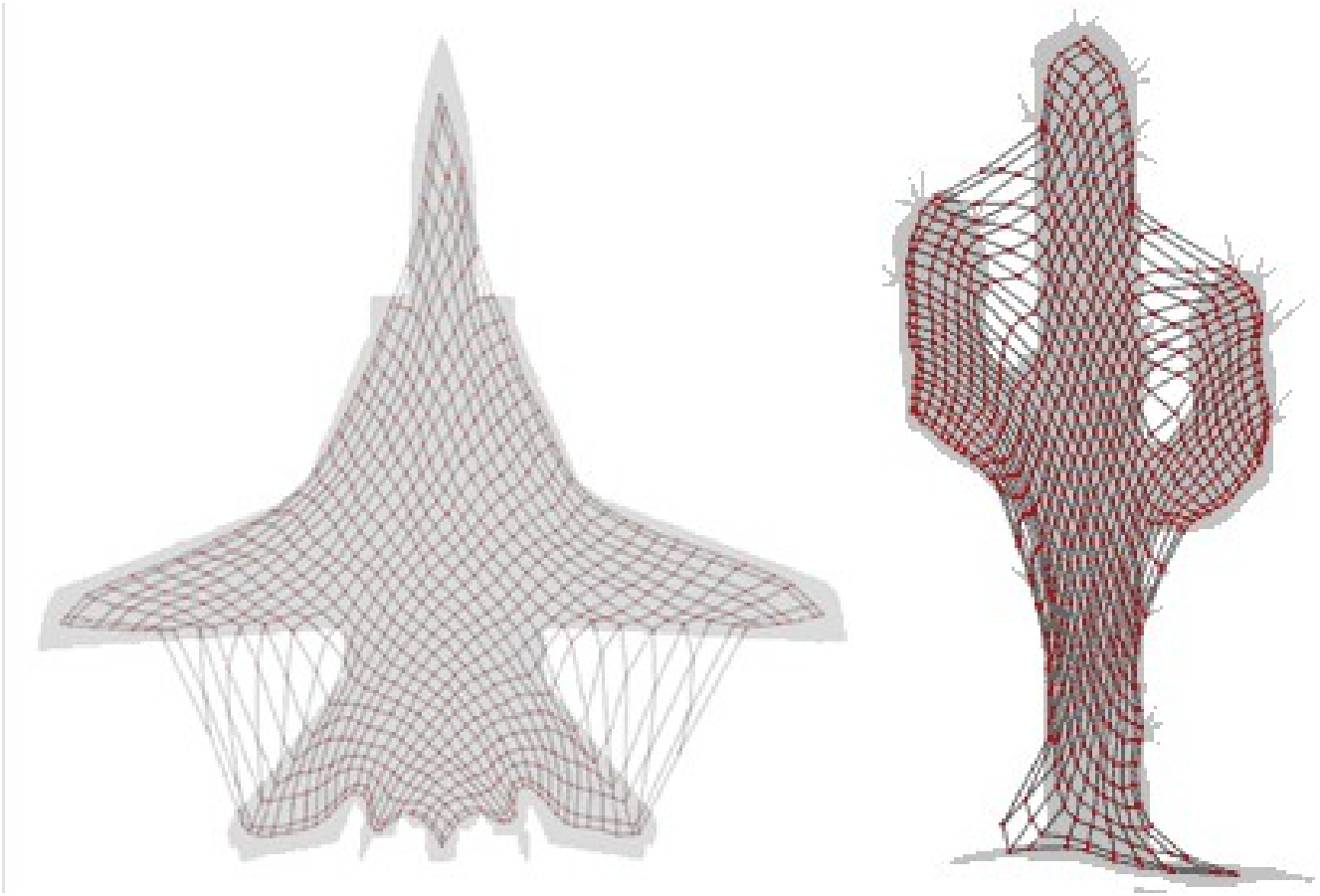
1D



2D



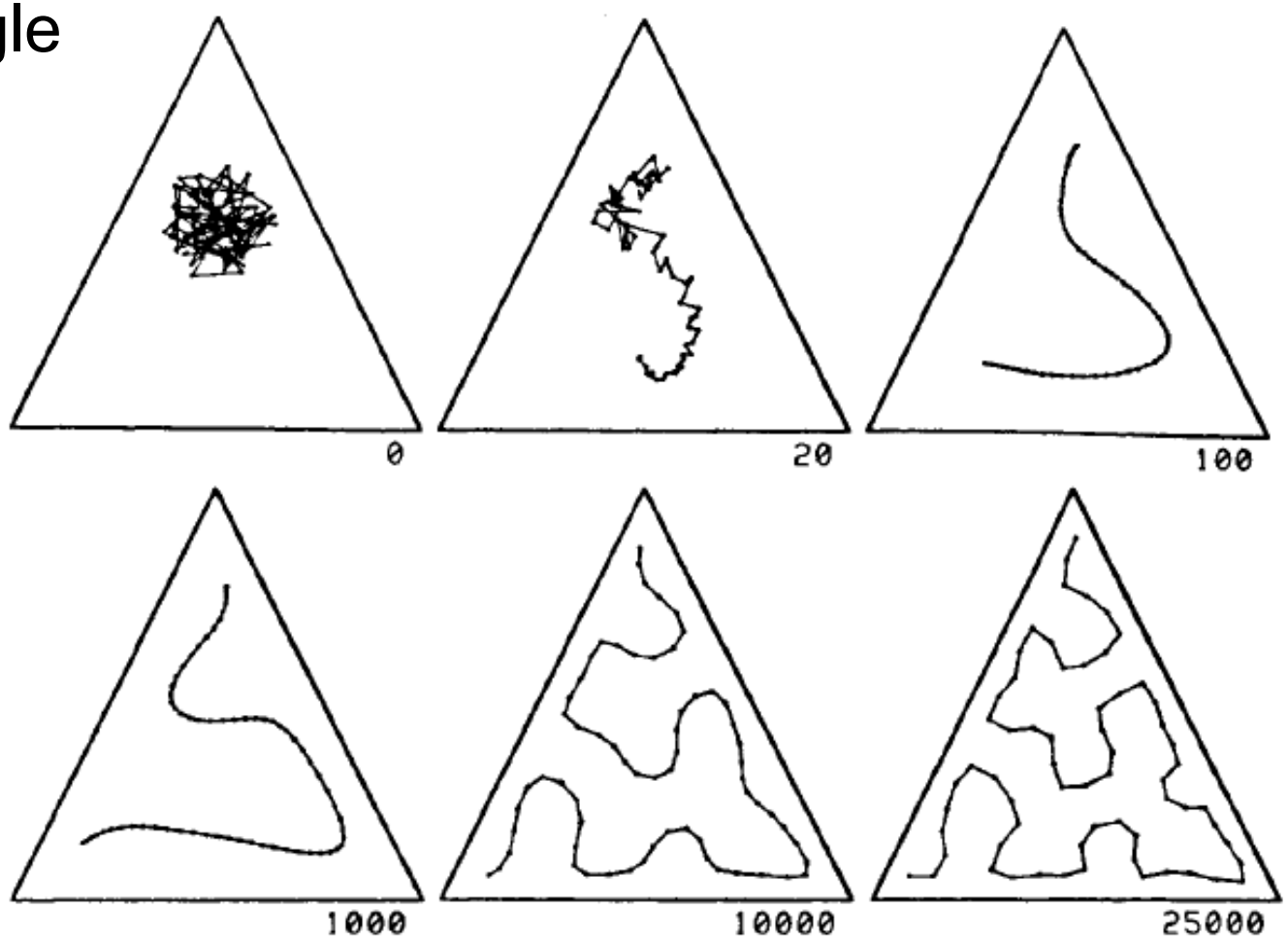
Or …



Slide by Johan Everts

More Examples

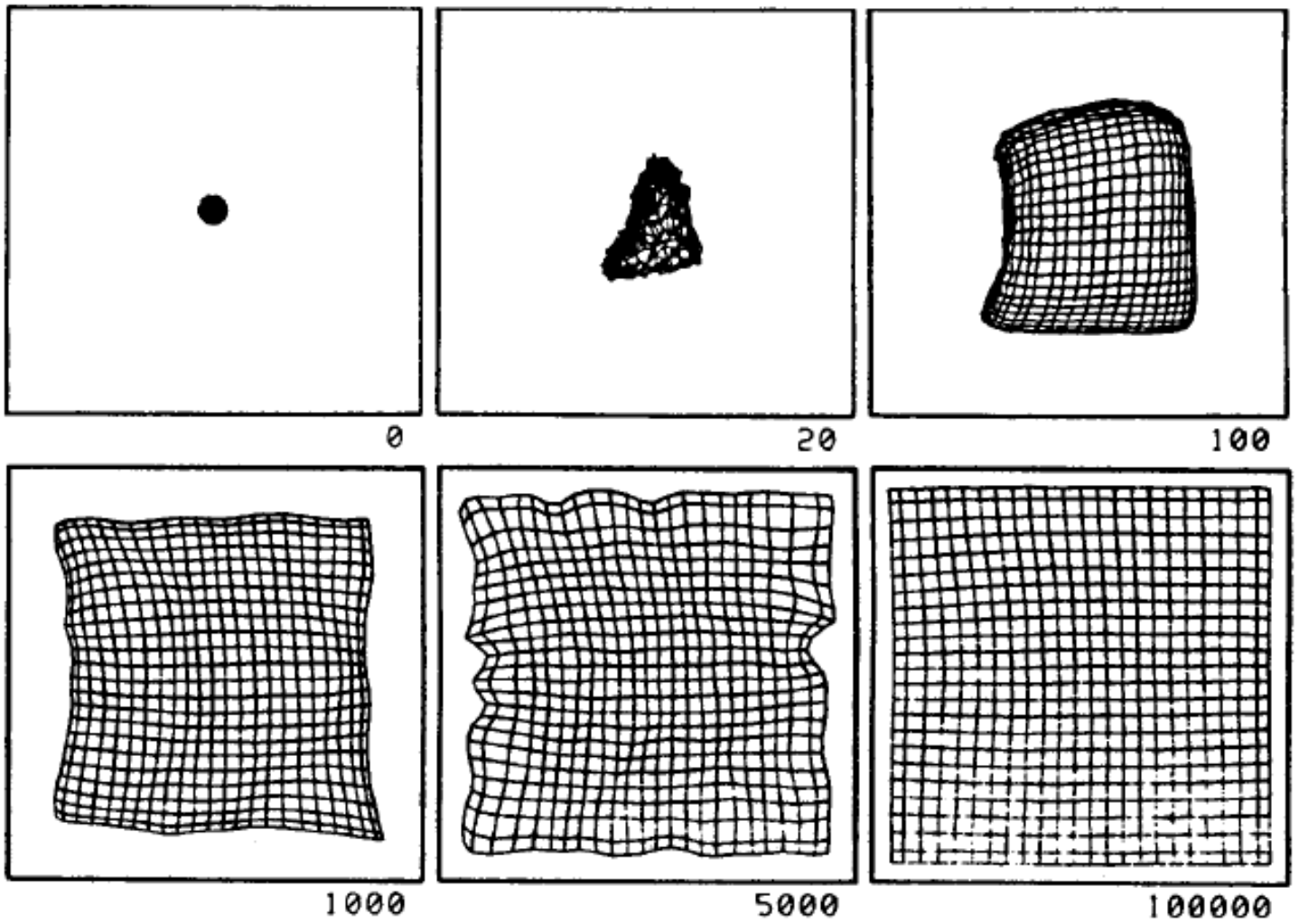
Covering a triangle by 1D SOM.



T. Kohonen: Self Organizing Maps

More Examples contd.

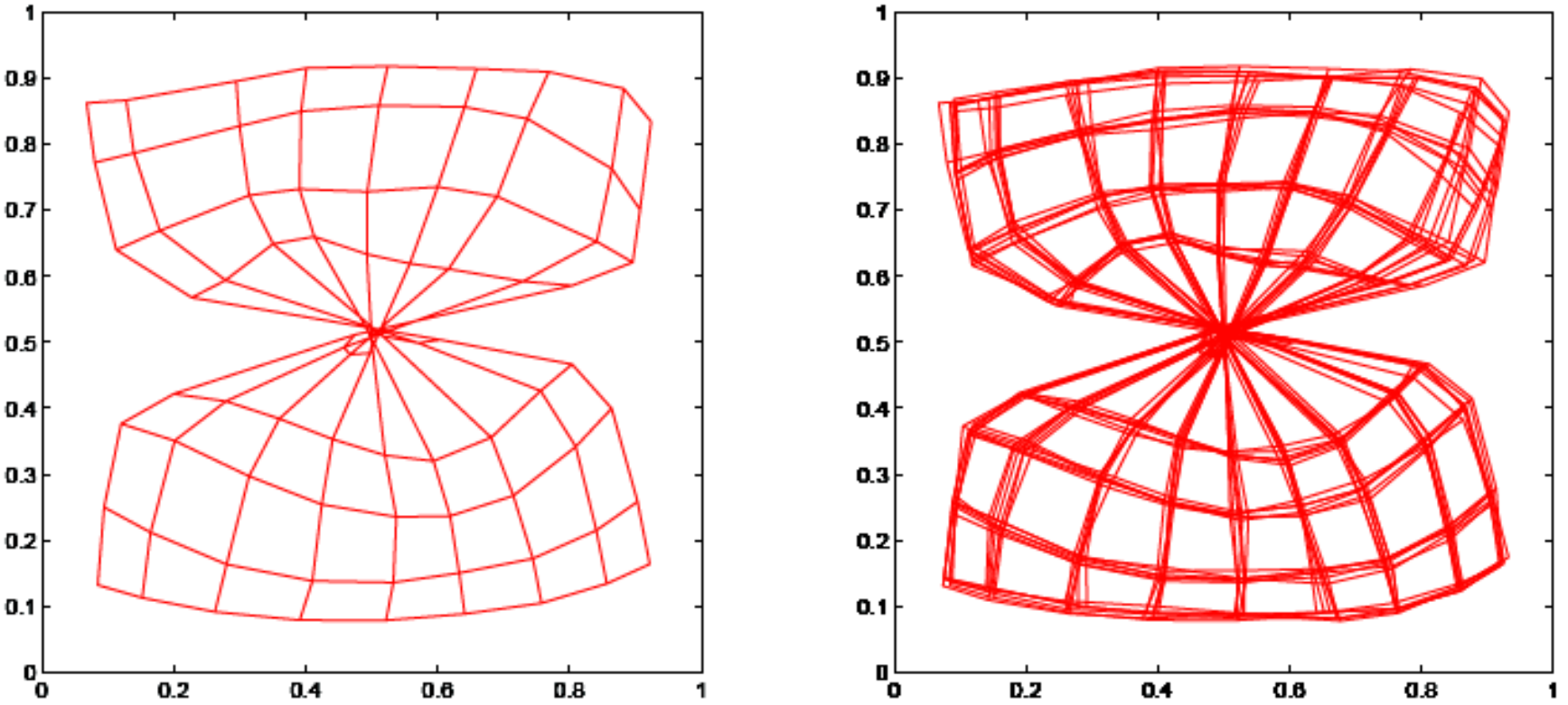
Covering a square by 2D SOM.



T. Kohonen: Self Organizing Maps

Possible Problem: Knots

- This problem is not likely to be corrected by further learning if the *plasticity* is low:



Rojas: Neural Networks - A Systematic Introduction

What is the Cause?

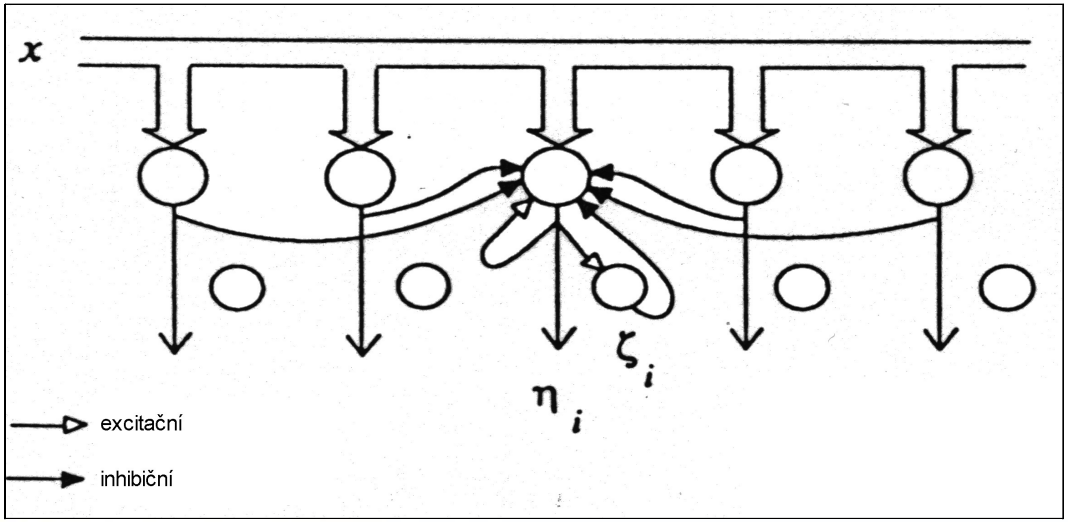
- There are many:
  - Random initialization of weights → we are unable the change bad initial position/orientation of vectors.
  - Choice of a neighbourhood function.
  - Scheduling of neighbourhood modification in time.
  - Input data of course...

What Can Help?

- Same weights for all neurons initially → each neuron has a same chance to represent a pattern.
- Add random noise to input patterns at start.
- Lateral inhibition...

Lateral Inhibition

- When choosing the BMU we do not pick isolated winner.
- The choice does not depend on an activation of a single neuron but also on activity of its neighbours...



Lateral Inhibition II

$$I_j = I^l_j + I^f_j = d_j + \sum_k g_{jk} I_k$$

j-th neuron response

local response

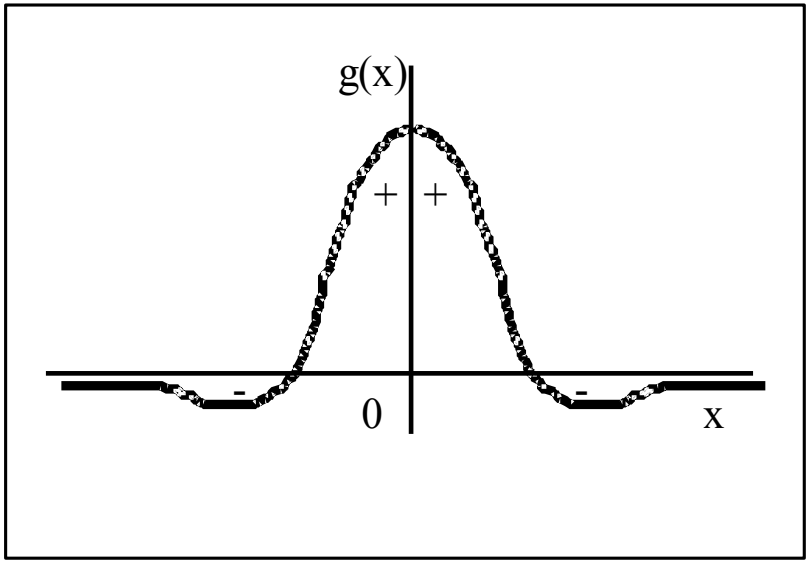
neighbourhood response

distance from input vector

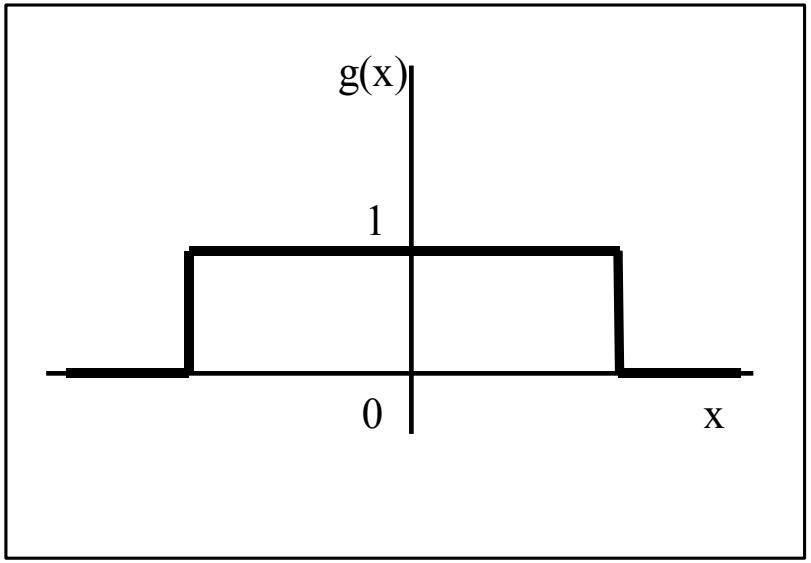
neighbours

lateral inhibition interaction

Lateral Inhibition Functions



biological



simplified

SOM Visualization

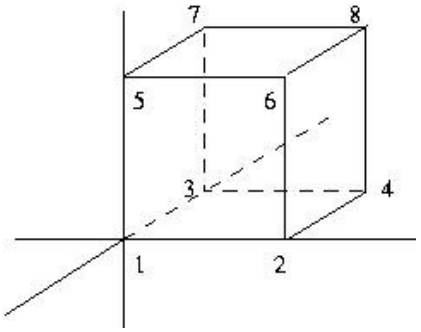
- How to visualize representatives?
- Weight dimension = input vector dimension.
- How to show in 2D?
  - U-matrix (P-matrix and others),
  - PCA (linear projection),
  - Sammon's projection (non-linear).

U-matrix (UMAT)

- Visualizes distances between neurons:
  - Dark coloring between neurons → large distance.
  - Light → close in input space.
- Dark gaps separate clusters.
- Neuron colour reflects the distance of its weight vector to all other weight vectors, again:
  - dark → large distance,
  - light → close distance.

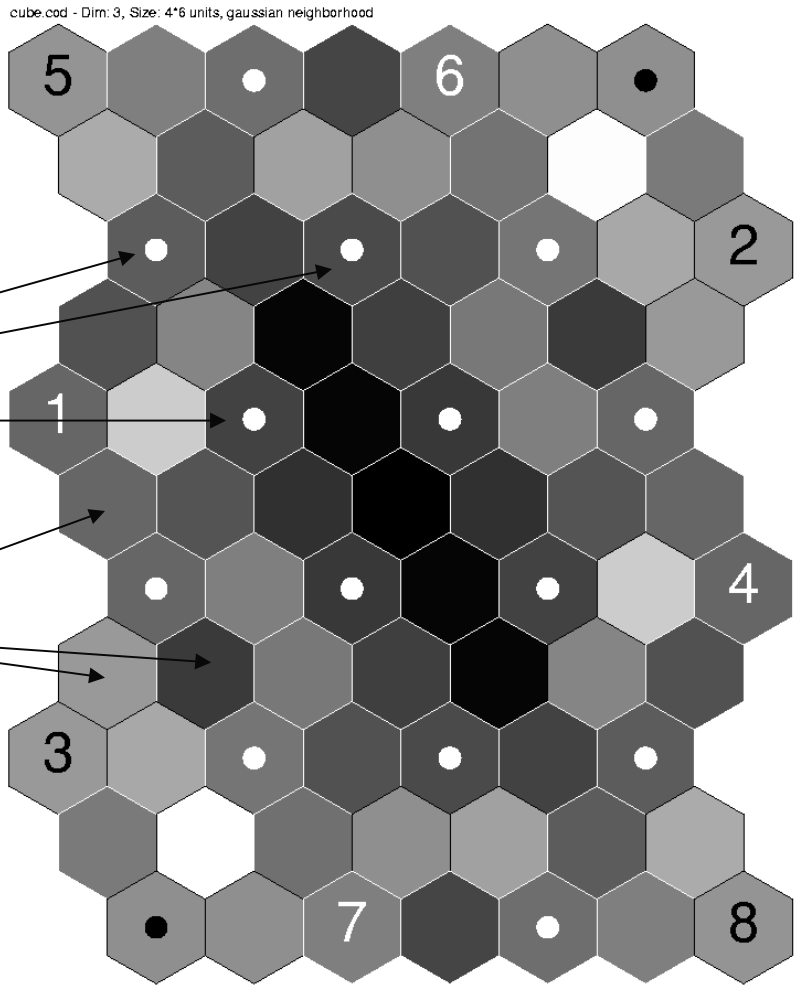
U-matrix Example

data



neurons

distance between
adjacent neurons



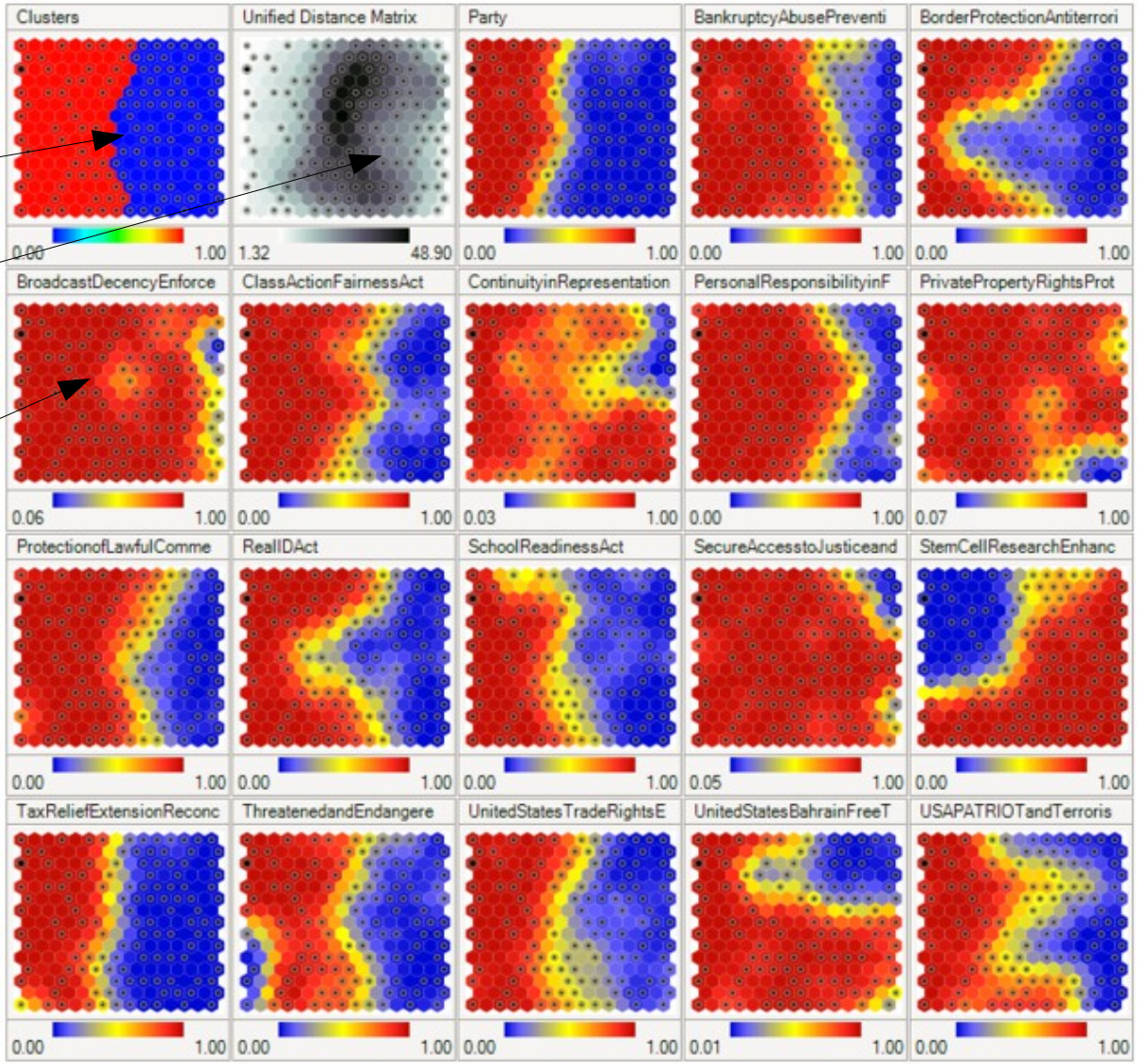
Feature Plots

clustering

UMAP

feature plot
shows a value
of a single
component (feature)
of a weight vector

can be used to
check if two
components
correlate



http://en.wikipedia.org/wiki/Self-organizing_map

Drawbacks of UMAT (PMAT)

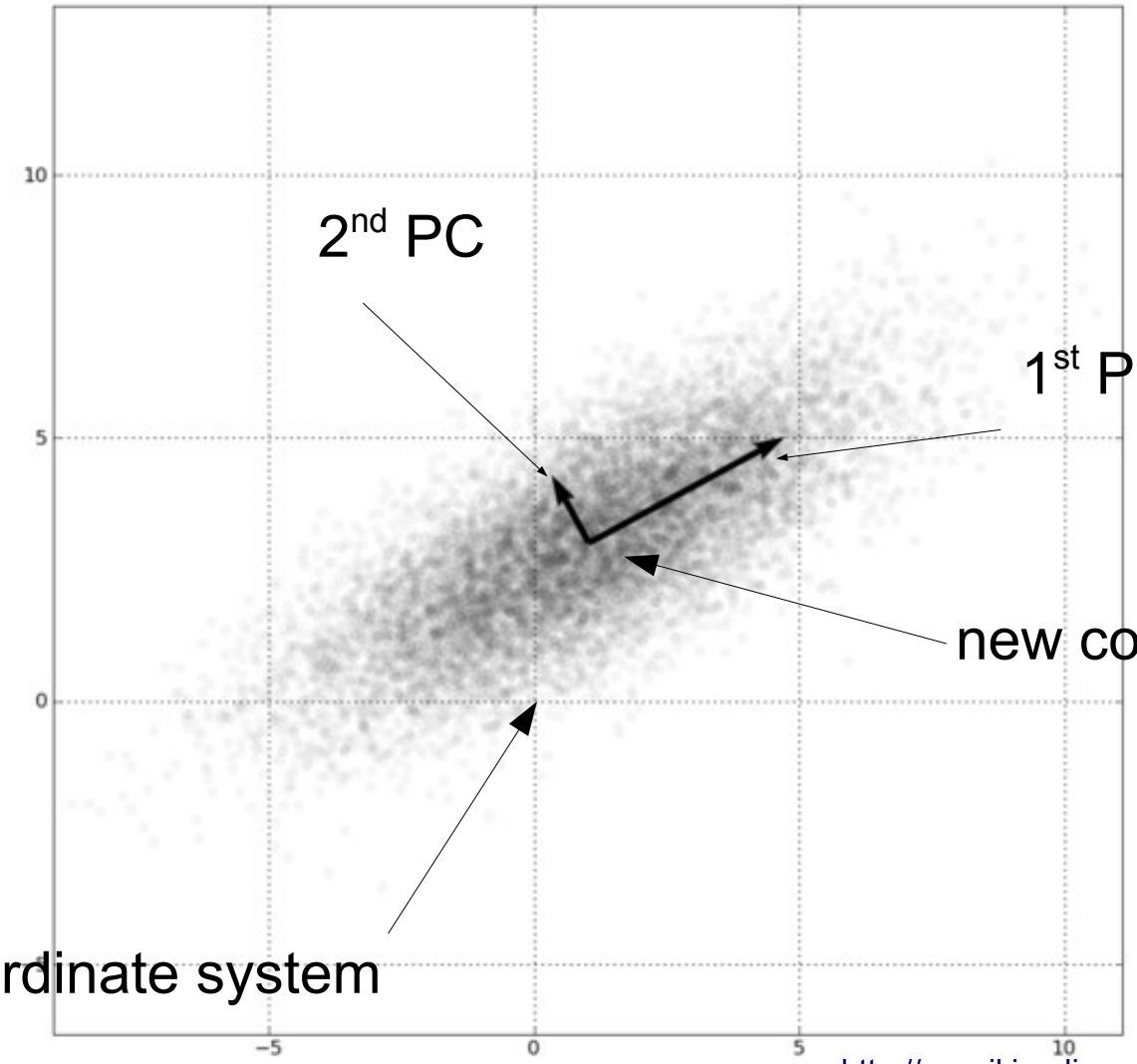
- Only distances between neighbours.
- New learning on the same data may give different results: (e.g., 90 degrees rotation)
- Not intuitive.

- **How can we show high-dimensional data in 2D(3D) keeping notion of original distances?**

PCA

- Principal Component Analysis.
- Linear transformation to a new coordinate system such that:
  - 1st coordinate (principal component)→ greatest variance by any projection of the data
  - 2nd coordinate → 2nd greatest variance
  - etc.
- Dimension reduction → use only *N* first coordinates, **throw the rest away**...

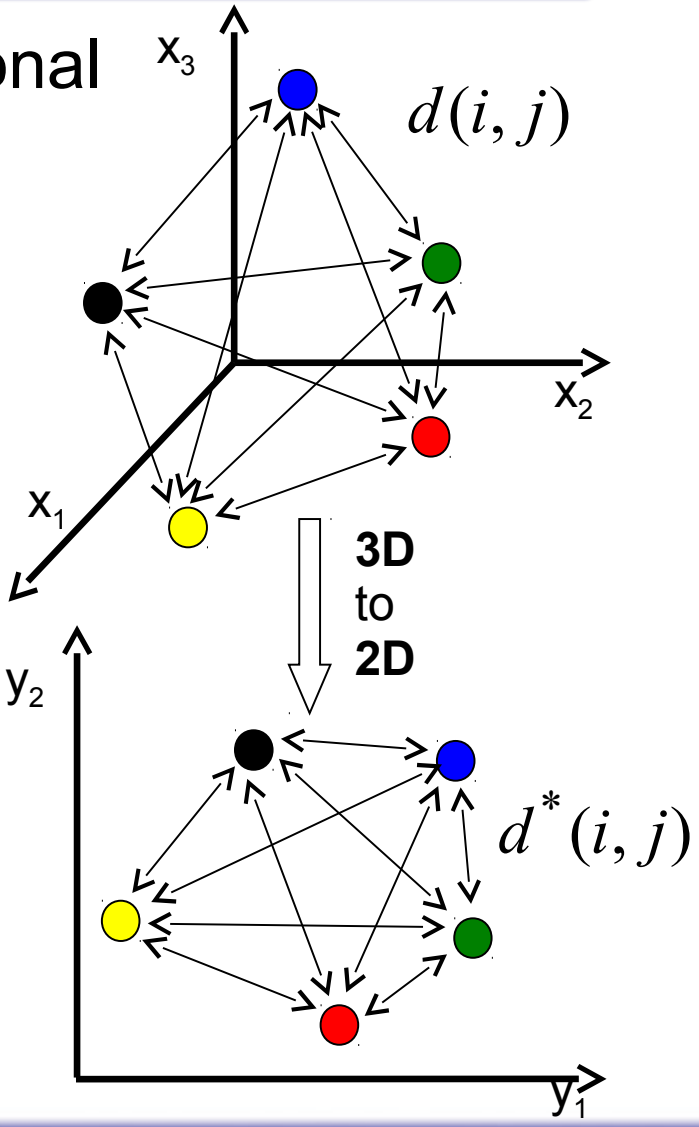
Principal Components Example



http://en.wikipedia.org/wiki/Principal_component_analysis

Sammon's Projection

- Non-linear reduction of higher-dimensional space to lower-dimensional space.
- Tries to preserve distances.



energy function → low for similar distances in both spaces.

distance in high dim. space

distance in low dim. space

$$E = \frac{1}{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} d(i,j)} \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \frac{\left(d(i,j) - d^*(i,j)\right)^2}{d(i,j)}$$

- Energy function is a subject to minimization (originally using gradient descent).

SOM Evaluation

- VQ – vector quantization, more input vectors mapped into a single neuron → **quantization error or distortion**.
- Compression of an input space dimension.
- Preserves data topology – neighbour vectors (from an input space) are mapped to neighbour neurons (in the mesh) → **topographic error**.

SOM Quantization Error & Distortion

- Quantization Error → average distance between input vector and its BMU (computed over all input vectors).
  - precision of mapping.
- Distortion → count with neighbours:

$$E = \sum_{i \in N} \sum_{j \in I} \eta_{i,bmu(j)} \left\| w(i) - x(j) \right\|^2$$

neurons

input vectors

Energy function again!

Topographic Error of SOM

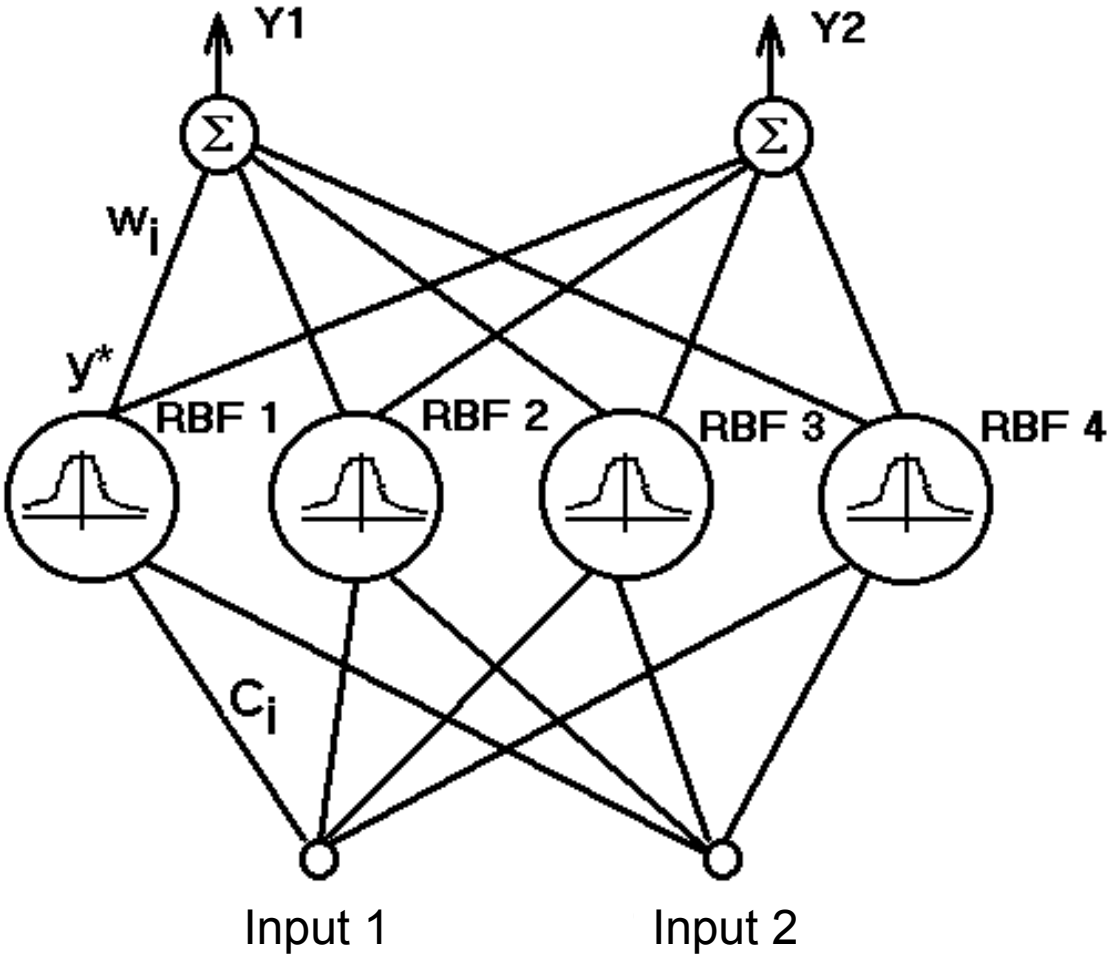
- # of input vectors, for which the winner (BMU) and the second best neuron are not adjacent in the mesh.

RBF Networks

Radial Basis Function (RBF) Networks

- Two layers, different types of neurons in each layer.
- Feed-forward.
- Supervised Learning.
- Broomhead, Lowe (1988).

RBF Architecture



RBF Neurons

- **Hidden layer:**
  - inner potential

$$\phi = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2}$$

$C = (c_1, c_2, \ldots, c_n)$
center (prototype)

  - non-linear activation function

$$y^* = f(\Phi)$$

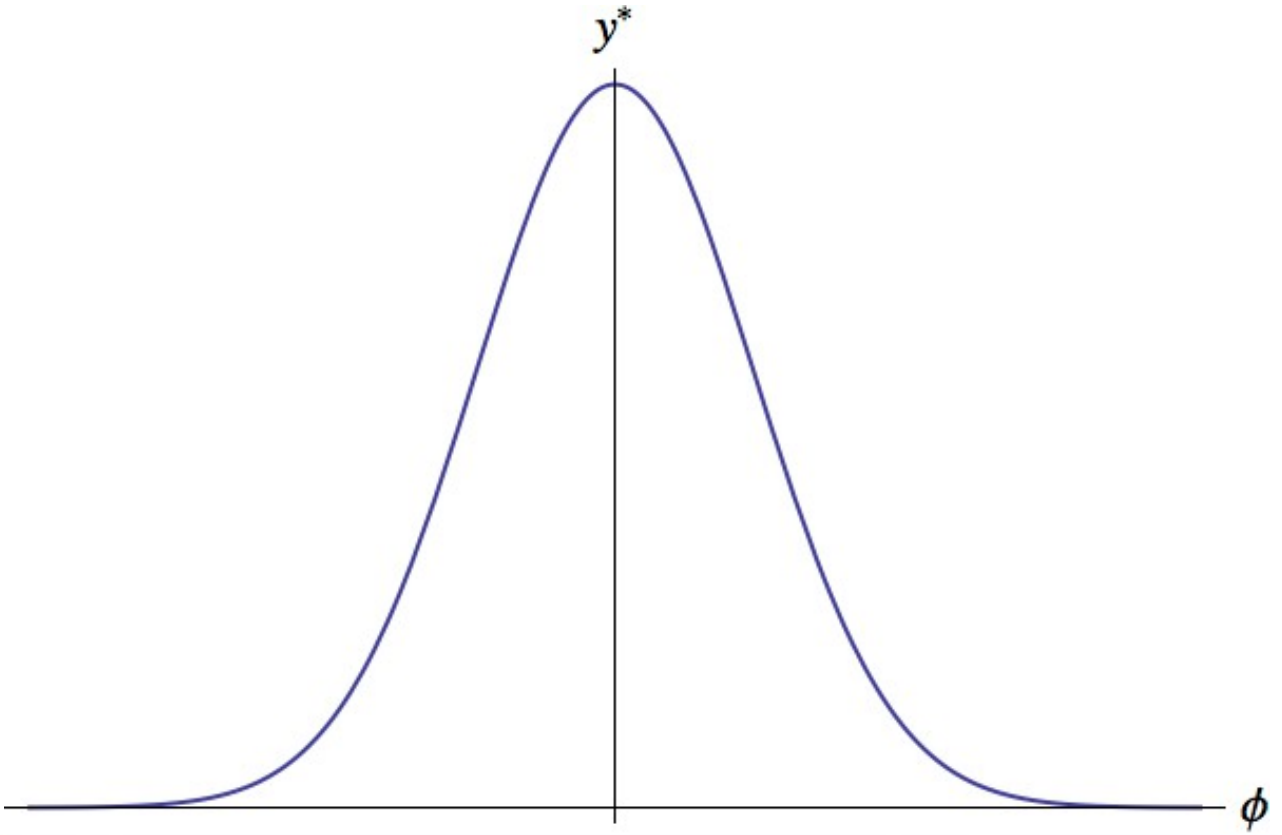
- **Output layer:**
  - linear perceptron

$$y = \sum_{i=1}^{n} w_i y_i^*$$

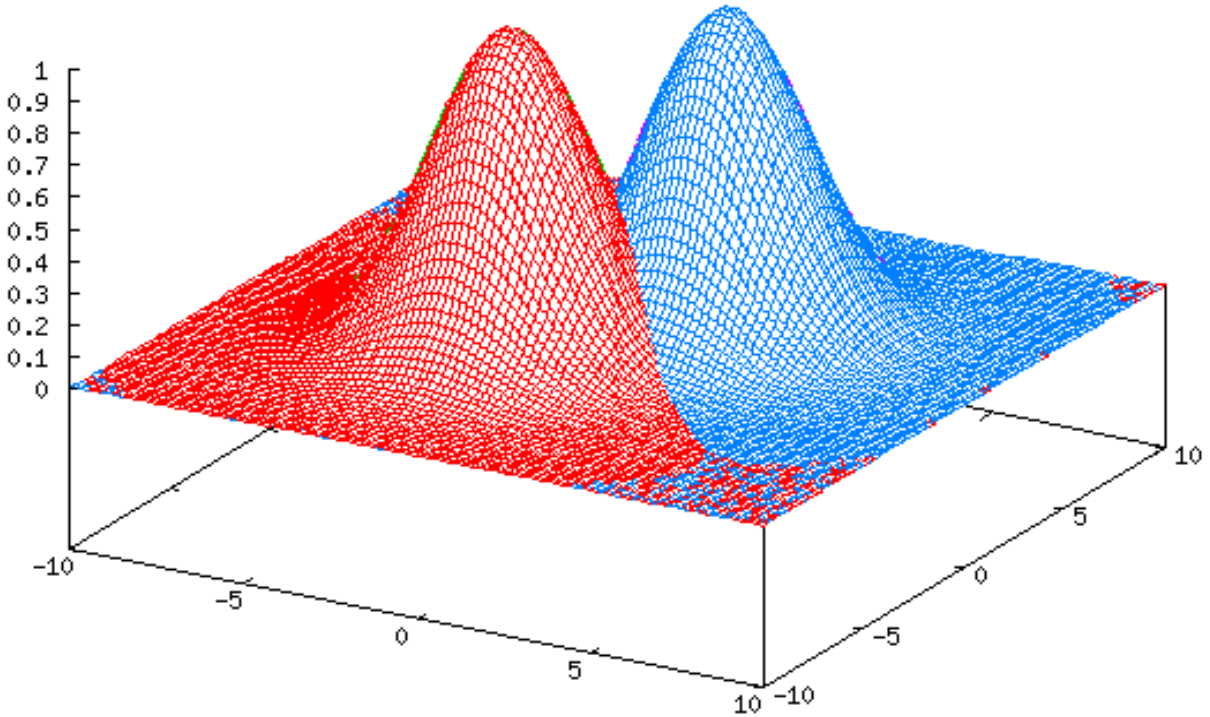
Typical RBF Activation Function

Gaussian with zero mean and variance σ^2:

$$y^* = e^{-\frac{\phi^2}{\sigma^2}}, \sigma > 0$$



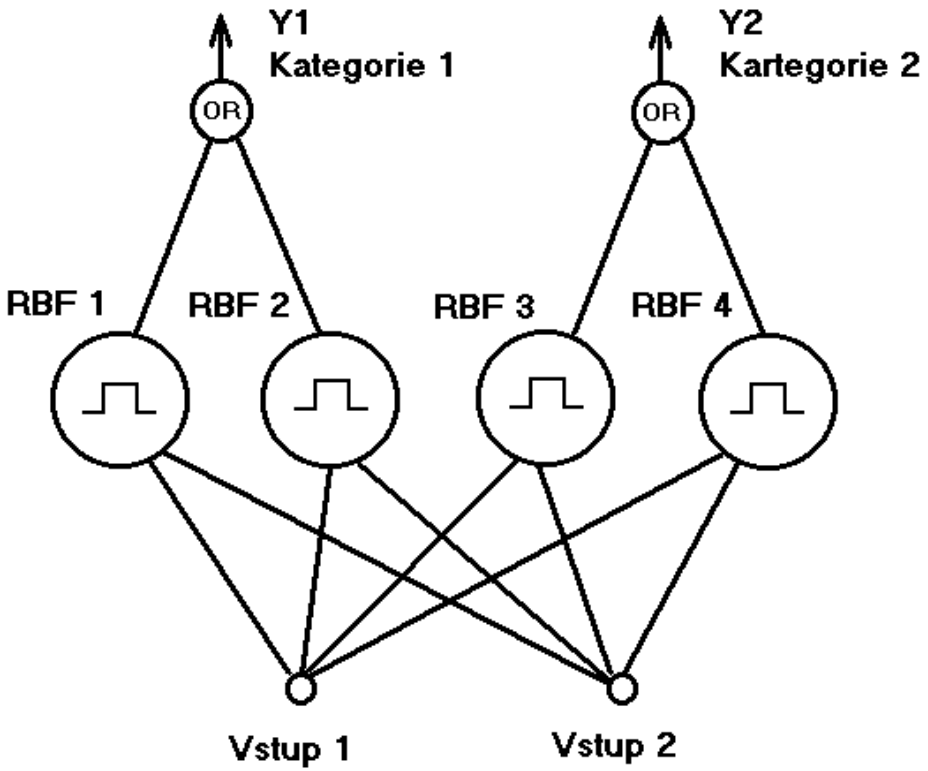
Sphere of Influence: Example

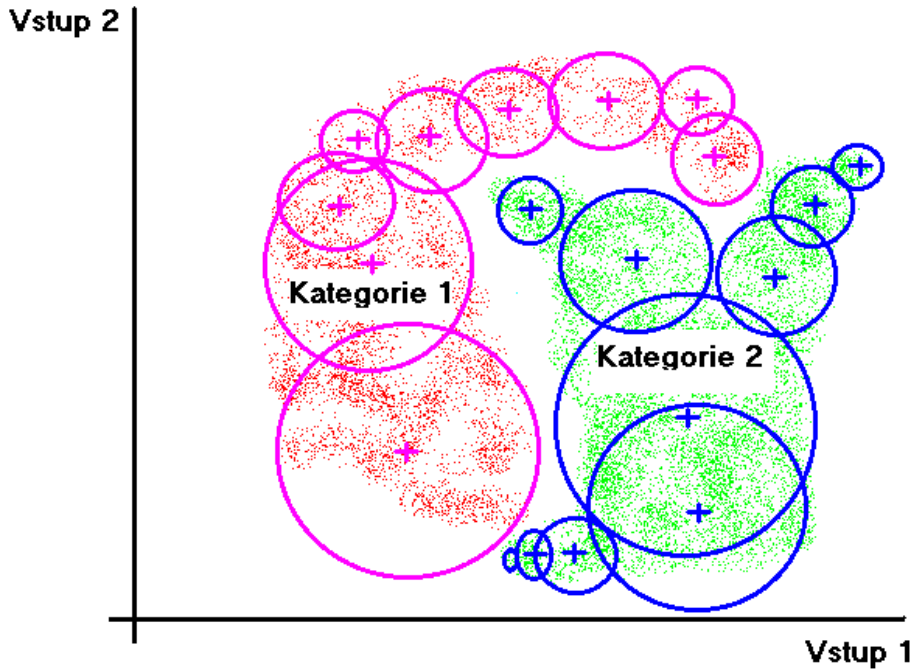


Neuron Types Discussion

- RBF neurons:
  - inner potential is a measure of distance between input vector and the prototype (represented by the weights),
  - activation function delimits the sphere of influence.
- Output neurons:
  - accumulate RBF neurons' outputs to achieve accurate approximation, OR
  - perform union of sets for classification.

RBF as a Classifier

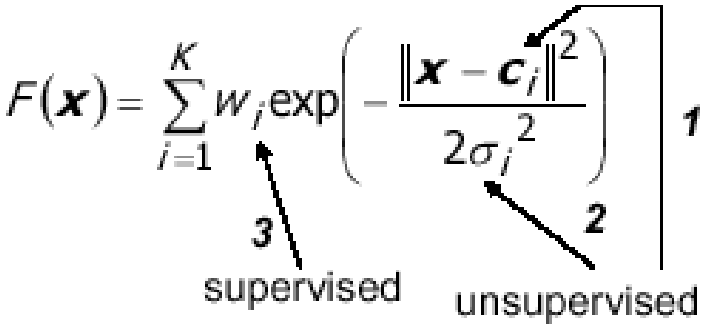




Training RBF Networks

- Two phases:
  - training prototypes,
  - training output neurons.
- Training prototypes:
  - Unsupervised: Cluster Analysis.
- Training output neurons:
  - Supervised.

3-steps method:



Training Prototypes I

- Estimate the number of clusters in input data :(
- Define a *membership function* ***m(x)***:
  - determines whether an input pattern belongs to a certain cluster,
  - choose cluster by the closest prototype.
- Estimate coordinates of all vectors $\mathbf{C}_p$ which correspond to cluster centers:
  → we use a well-known K-means algorithm.

Training Prototypes II: K-means

- **Steps of K-means algorithm**:
  - randomly initialize centers $\boldsymbol{C}_i$ of RBF neurons,
  - *) evaluate ***m()*** for all input patterns,
  - determine a new center $\boldsymbol{C}_k$ as an average of all patterns which belong to cluster ***k*** according to ***m()***.
  - stop when ***m()*** does not change anymore, otherwise go to *)

Training Prototypes III: Determine σ

- Parameter σ is determined by a root mean squared deviation (RMSD) of distance between patterns and cluster center:

$$\sigma_k = \sqrt{\frac{1}{Q}\sum_{q=1}^{Q} \|\bar{C}_k - \bar{X}_q\|^2}$$

- where $\boldsymbol{X}_q$ is q-th pattern which belongs to cluster with center $\boldsymbol{c}_k$.

Other approach uses fixed σ.

Training Output Neurons

- Find weights by minimization of this energy function:

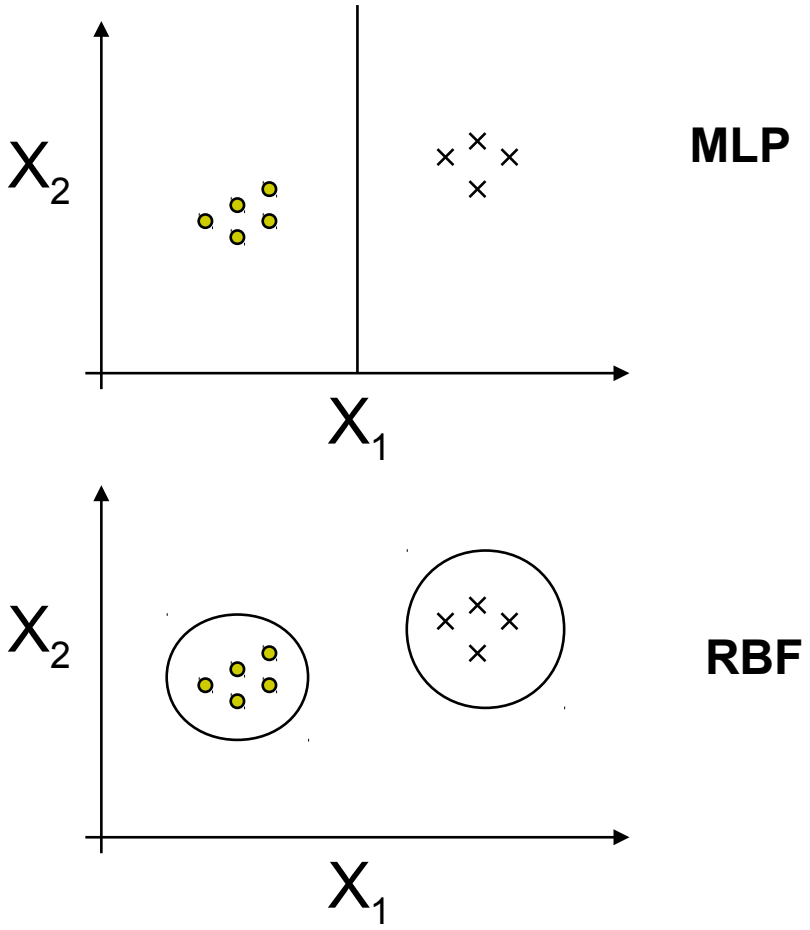
$$\Delta \bar{w}^{(t)} = -\eta \nabla E^{(t)} = \eta \left(D^{(t)} - Y^{(t)} \right) Y^{*(t)}$$

- Is it familiar?

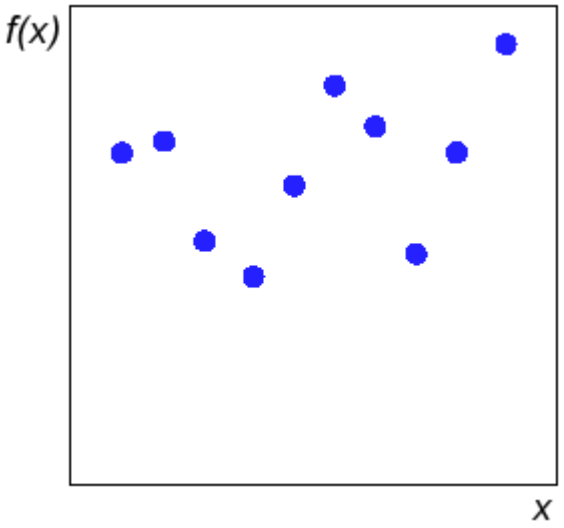
$$E = \frac{1}{2} \sum_{t=1}^{m} \sum_{i=1}^{n} \left(d_i^{(t)} - y_i^{(t)} \right)^2$$

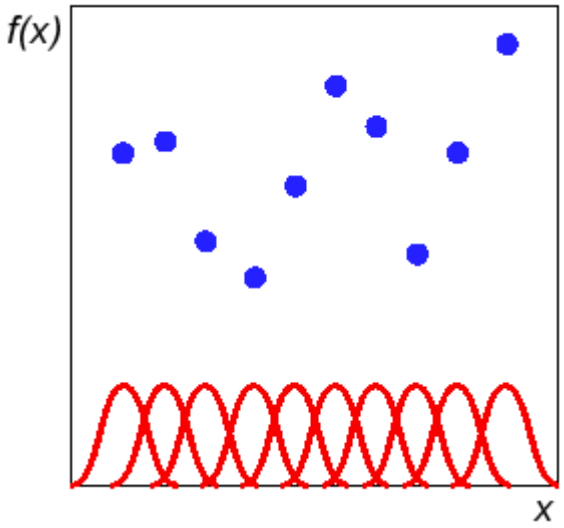
MLP vs. RBF

- **Learning**
  - RBF learns faster.
- **Applications**
  - MLP/RBF usable for regression & classification
- **Properties**
  - both are universal approximators.
- **As classifiers:**
  - MLPs - hyperplanes,
  - RBFs - hyperspheres.



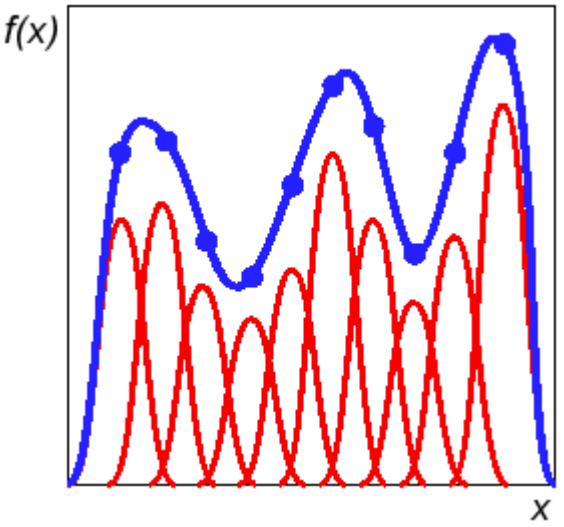
Approximation by RBF: Example





RBF neurons

Output neurons



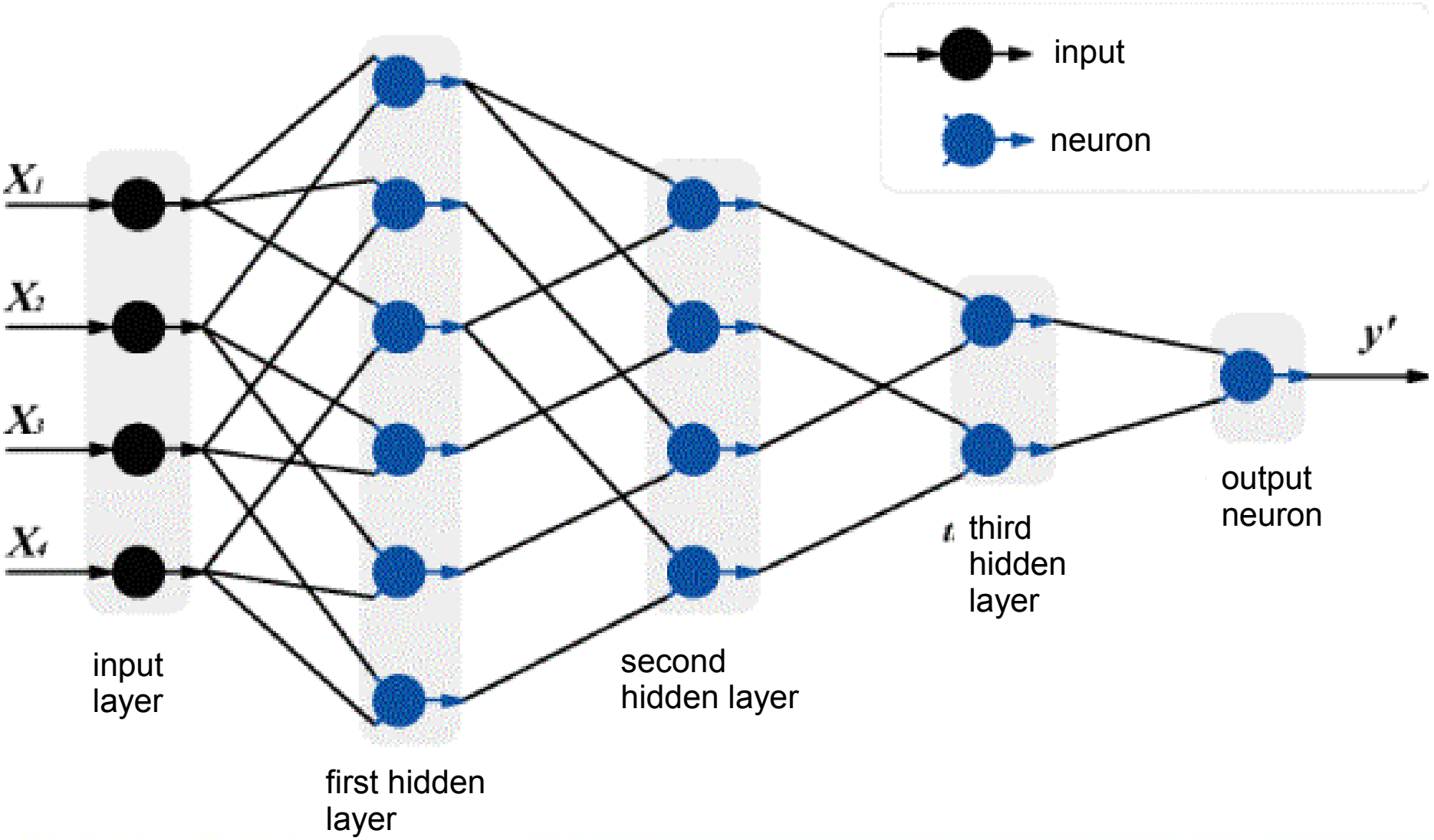
GMDH

- **G**roup **M**ethod of **D**ata **H**andling.
- By A.G. Ivakhnenko, 1968.
- Network is constructed by **induction**.
- Multiple layers.
- Feed-forward.
- Supervised.
- See http://www.gmdh.net ...

GMDH Types

- Parametric
  - parameters are set during training,
  - MIA (Multilayer Iterative Algorithm) → partial induction
  - COMBI (Combinatorial Algorithm) → full induction
- Other variants are non-parametric.

GMDH MIA Architecture

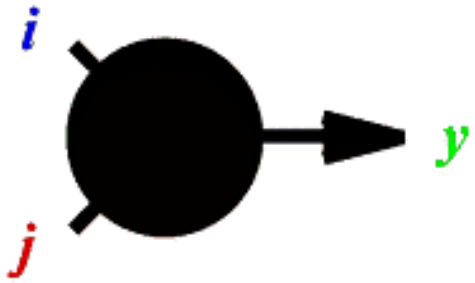


Remarks

- The structure (topology) of this network forms itself during learning process.
- # of neurons and even # of hidden layers changes during learning,
  → network self-organizes (but in a different way than SOM)

GMDH MIA Neuron

$$y = ai^2 + bij + cj^2 + di + ej + f$$



Ivakhnenko's polynomial.

Example of other Ivakhnenko's polynomials:

$$y = a + bi + cj \ ,$$
$$y = ai + bjk \ .$$

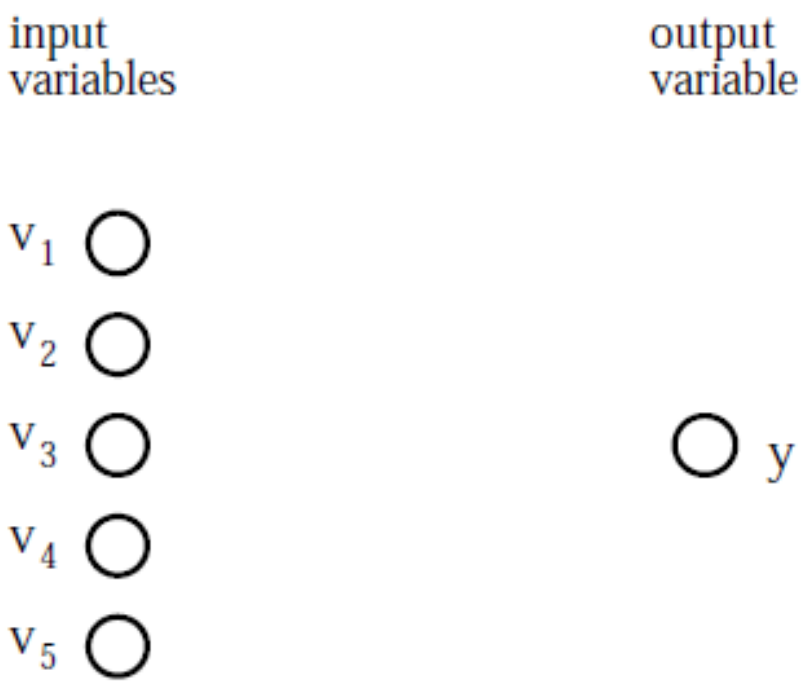
MIA Characteristics

- MIA network uses supervised learning.
- MIA uses single type of neurons (unlike other GMDH variants).
- Training:
  - network is built layer by layer,
  - layers are added until some error criterion is met.
- Network evaluation is simple:
  - feed-forward, only single output.

MIA Training

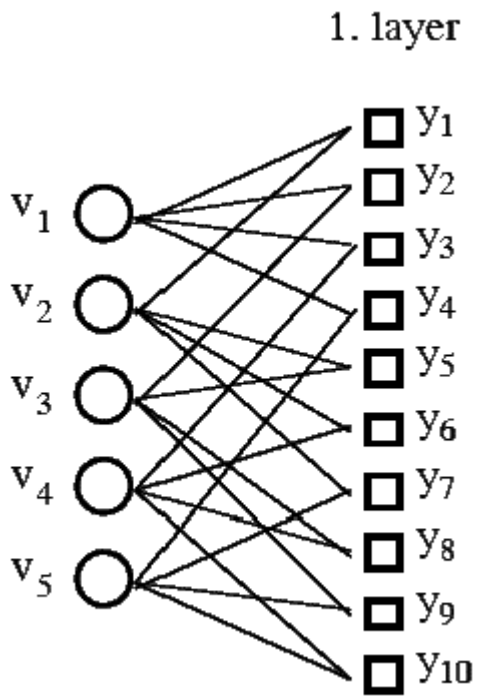
- **Configure k-th layer** (*k* denotes actual layer)
  - create new neurons in the layer,
  - compute all six coefficients of polynomial.
- **Selection of neurons in k-th layer**
  - elimination of unsuccessful neurons.
- Repeat or **finish training.**

MIA Training II



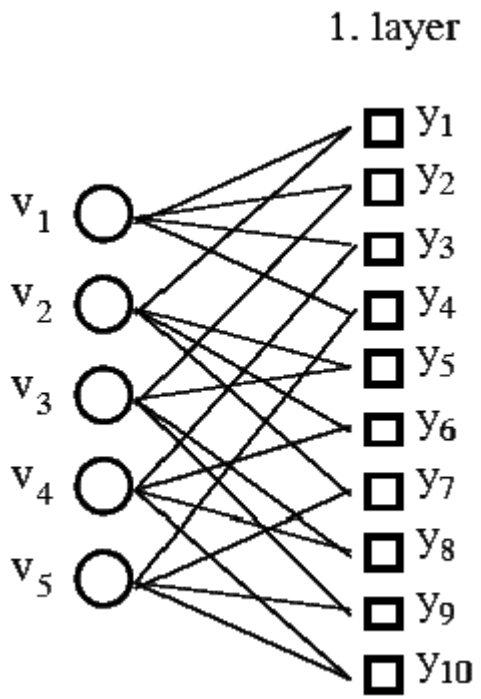
Lemke,Müller:SELF-ORGANIZING DATA MINING BASED ON GMDH PRINCIPLE

MIA Training III



- 1st layer creation:
- Active neurons with **optimized** transfer function $y_k=f_k(v_i,v_j)$.
- Two inputs only!
- They form “initial population”.
- How many neurons?

MIA Training III



- 1st layer creation:
- Active neurons with **optimized** transfer function $y_k=f_k(v_i,v_j)$.
- Two inputs only!
- They form “initial population”.
- How many neurons?

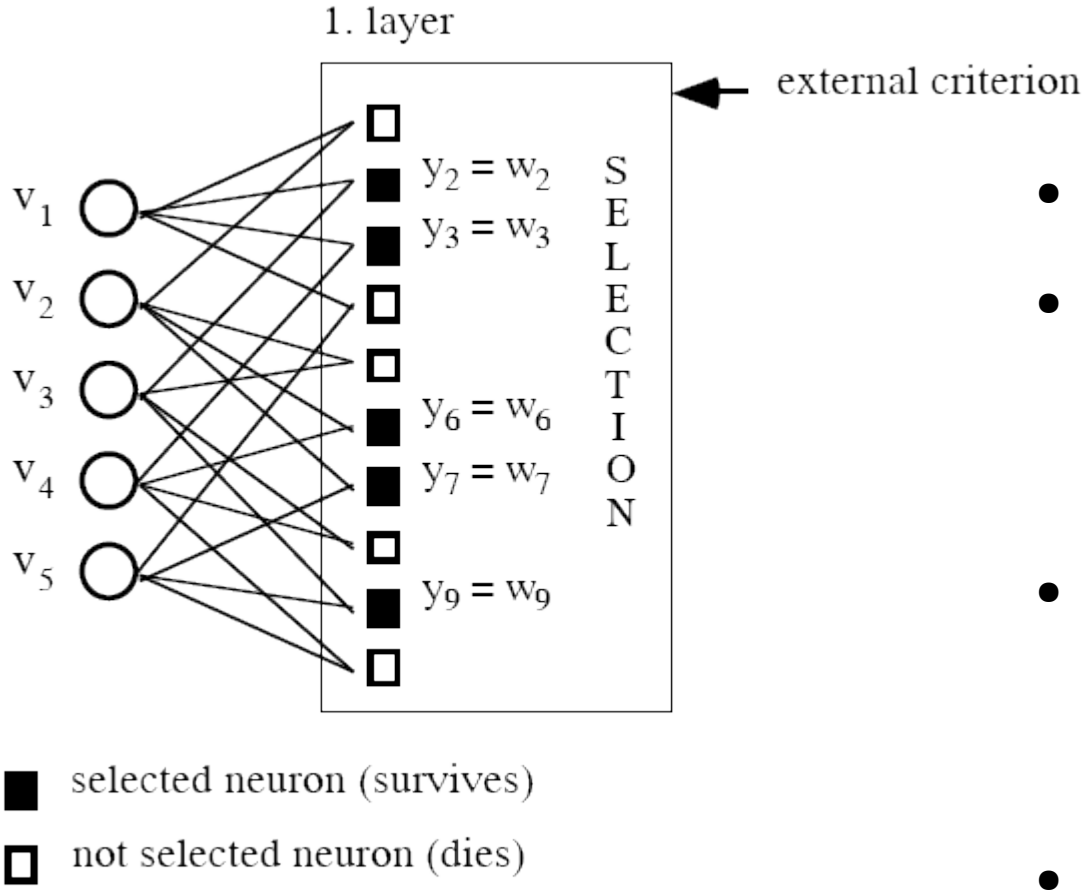
$$\binom{N}{2}=\frac{N(N-1)}{2}$$

MIA Training IV

- How to optimize neuron's coefficients?
  - Least Mean Squares (LMS),
  - we have to optimize six coefficients
    → choose 6 random input vectors
    → solve system of 6 linear equations

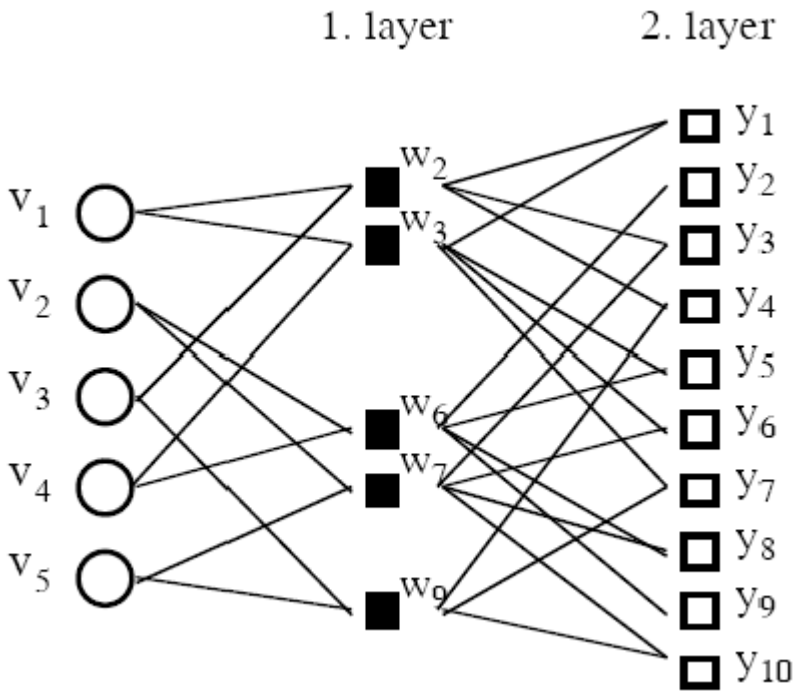
$$y = ai^2 + bij + cj^2 + di + ej + f$$

MIA Training V



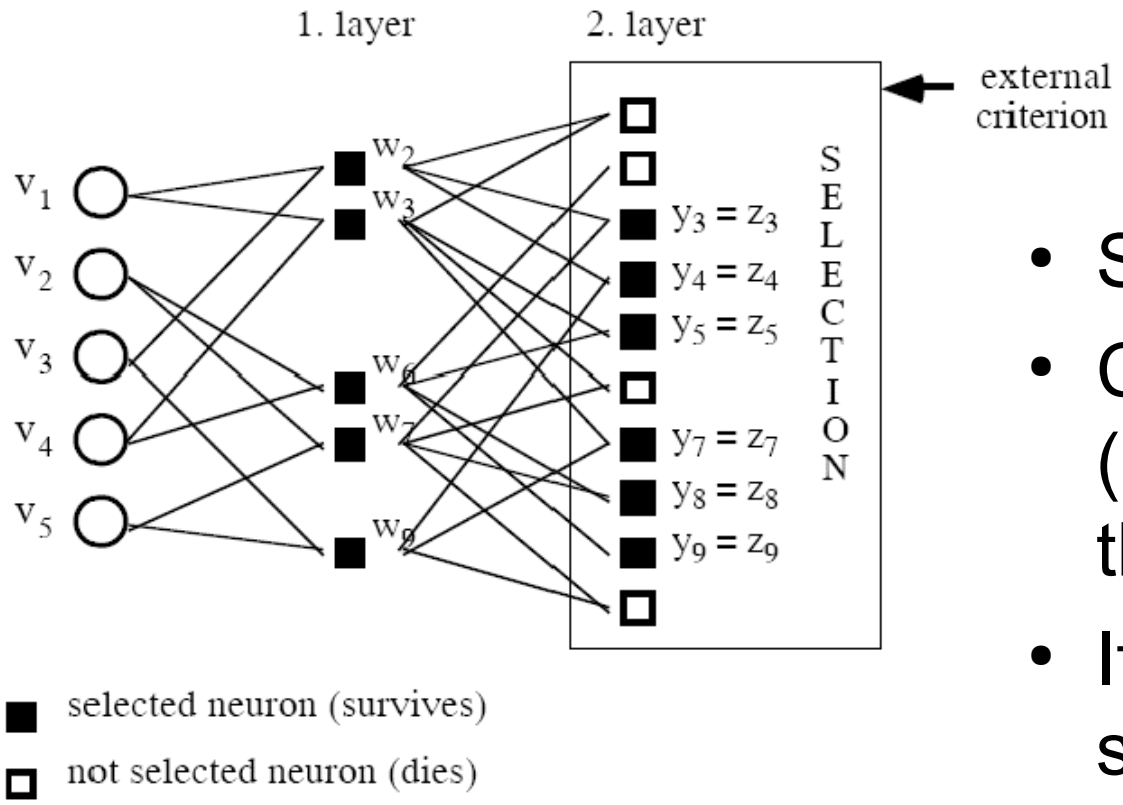
- Next step: Selection.
- According to a given criterion some neurons survive, others not...
- If we are satisfied with the output (low error) → stop.
- Similar to evolutionary algorithms...

MIA Training VI



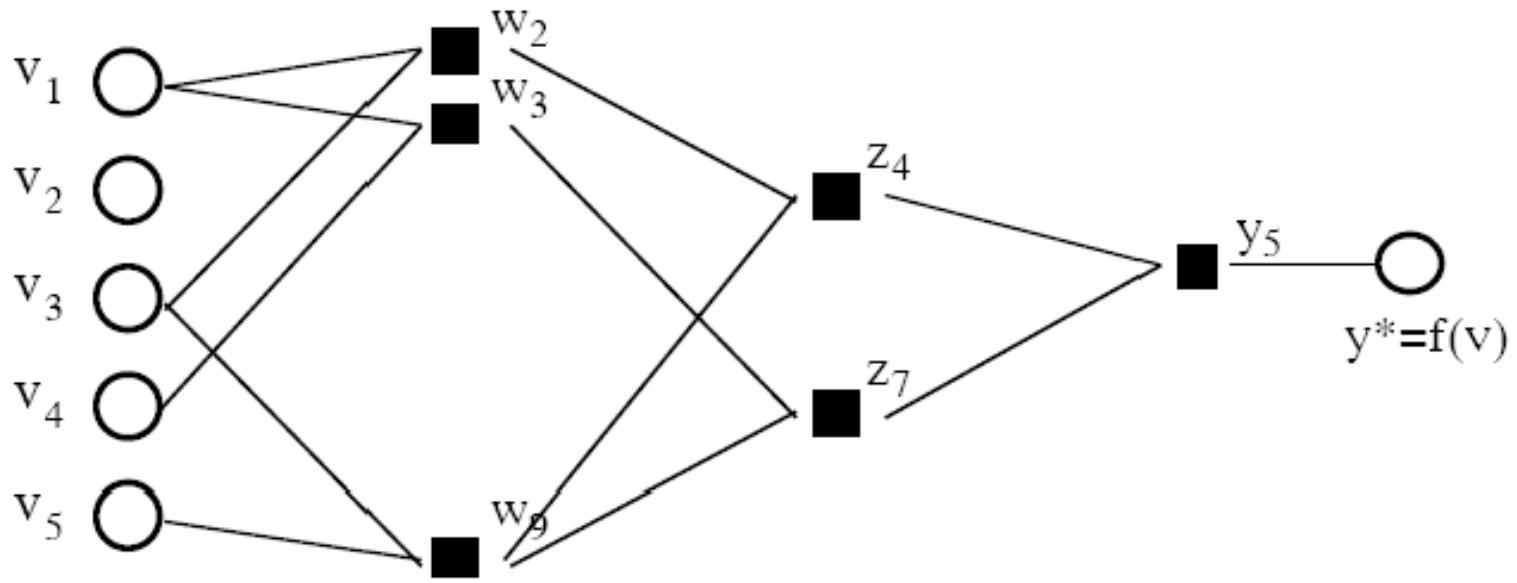
- 2nd layer:
- Again, initial population has to be created.
- Active neurons form the possible network output.
- Next step?

MIA Training VII



- Selection again.
- Competing neurons (survive and die) are of the same complexity.
- If we are still not satisfied with them → next layer is created.

MIA Training VII

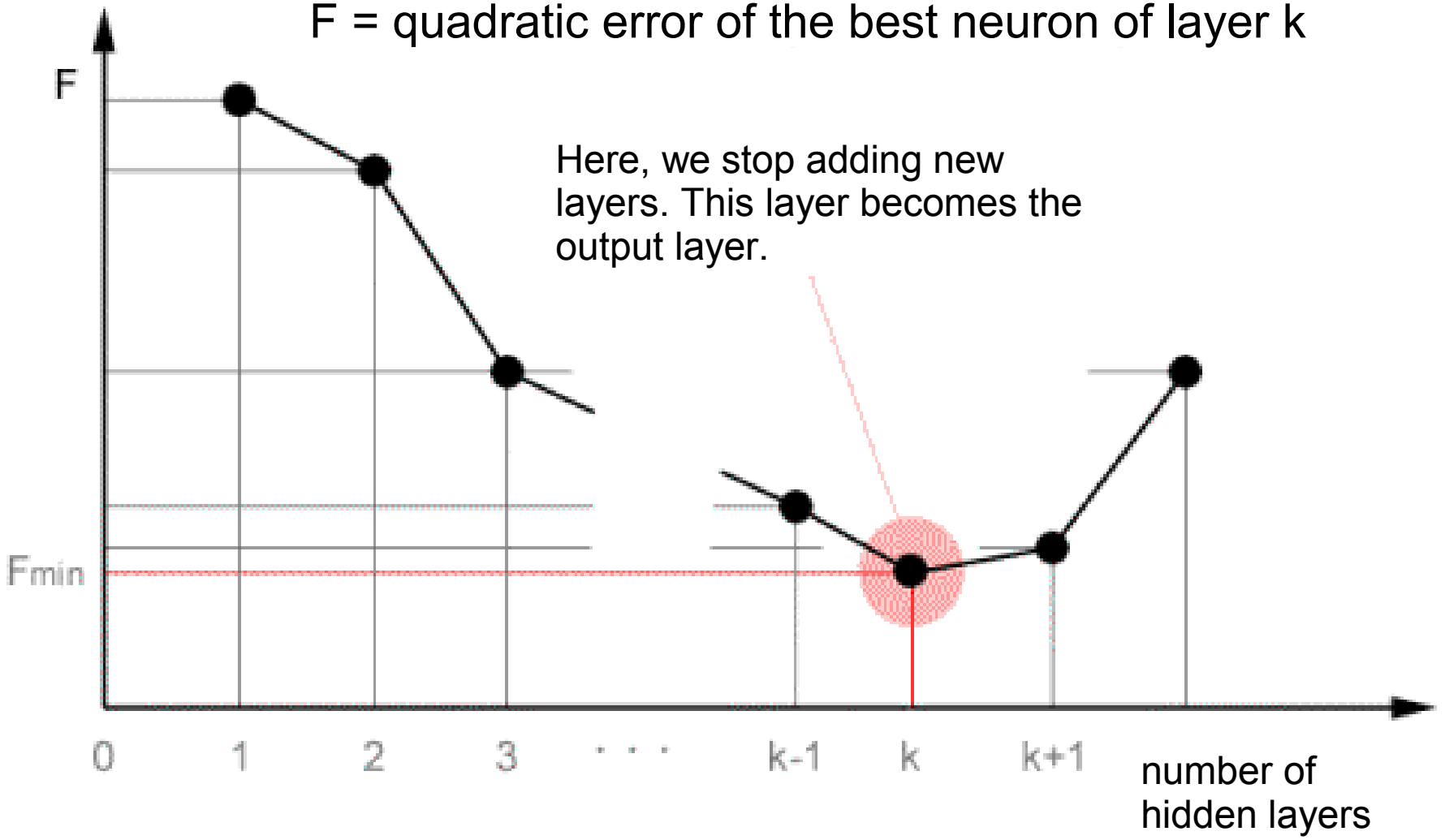


explicit analytically available optimal complex model.

network status:
order of regression model y*: ≤ 8
number of variables v in model y*: 4 (from initially 5 variables)

When to End Training?

F = quadratic error of the best neuron of layer k



Possibilities to Determine Fmin :)

$$\frac{N}{N-p}\,s_e^2 \qquad s_{e,p}^2$$

$$\frac{N+p}{N-p}\,s_e^2 \qquad FPE$$

$$s_e^2 + 2\sigma_p^2\,\frac{p}{N} \qquad PSE$$

$$\frac{Ns_e^2}{\sigma_c^2} + 2p - N \qquad Mallow\ C_p$$

$$N\ln s_e^2 + 2p + c \qquad AIC$$

$$N\ln s_{e,p}^2 + p\ln\left(\frac{s_{y^m}^2}{s_{e,p}^2}\,\frac{N}{p}\right) \qquad BIC$$

$$s_e^2 + s_y^2 \ln N\,\frac{p}{N} \qquad Schwarz$$

$$N\ln s_e^2 + p\ln N \qquad Rissanen$$

$$\frac{2ps_e^2}{N-p} - Ns_{y^m}^2 \qquad Kocerka$$

$$\frac{\sum_{t=1}^{N}(\bar{y}_t^m - y_t)^2}{\sum_{t=1}^{N} y_t^2} \qquad i^2(N)$$

$$\frac{s_e^2}{\left[1-\lambda\sqrt{\frac{k(\ln(\frac{N}{k})+1)-\ln\alpha}{N}}\left(1-\frac{1}{4}\,\frac{\ln(\frac{N}{k})+1)-\ln\alpha}{N}\right)\right]} \qquad Vapnik$$

Regularity criterion

$$AB = \frac{1}{N_B}\sum_{i=1}^{N_B}\left[y_i - \hat{y}_i(A)\right]^2 \to \min.$$

Discrimination criterion

$$\delta^2 = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} < 1.0$$

Validation criterion

$$R_{ij} = 1 - \sqrt{\frac{\frac{1}{N-P_i}\sum_{t=1}^{N}(y_t - y_t^i)^2}{\frac{1}{N-P_j}\sum_{t=1}^{N}(y_t - y_t^j)^2}}$$

$$s_e^2 = \frac{1}{N}\sum_{t=1}^{N}(y_t - y_t^m)^2 = \frac{1}{N}\sum_{t=1}^{N} e_t^2; \quad s_y^2 = \frac{1}{N}\sum_{t=1}^{N}(y_t - \bar{y})^2; \quad s_{y^m}^2 = \frac{1}{N}\sum_{t=1}^{N}(y_t^m)^2$$

Most Common

Regularity Criterion:

$$AB = \frac{1}{N_B} \sum_{i=1}^{N_B} \left[y_i - \hat{y}_i(A) \right]^2 \rightarrow \min.$$

$\hat{y}_i(A)$ … neuron‘s output trained on the A-subset.

Neurons are then ordered according to their error on validation data B-subset.

Other Inductive Approaches

- Cascade Correlation:
  diploma thesis by Minh Duc Do:
  https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13136&y=2009&a=dom1&t=dipl
- NEAT and other TWEANNs (Topology and Weight Evolving Neural Networks)
  → we will see later...