

---

# Artificial Neural Networks

## MLP, RBF & GMDH

*Jan Drchal*

*drchajan@fel.cvut.cz*

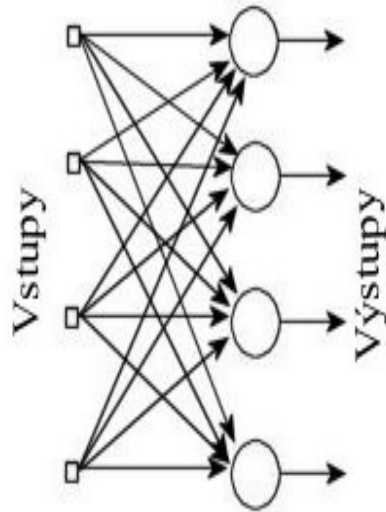
*Computational Intelligence Group  
Department of Computer Science and Engineering  
Faculty of Electrical Engineering  
Czech Technical University in Prague*

# Outline

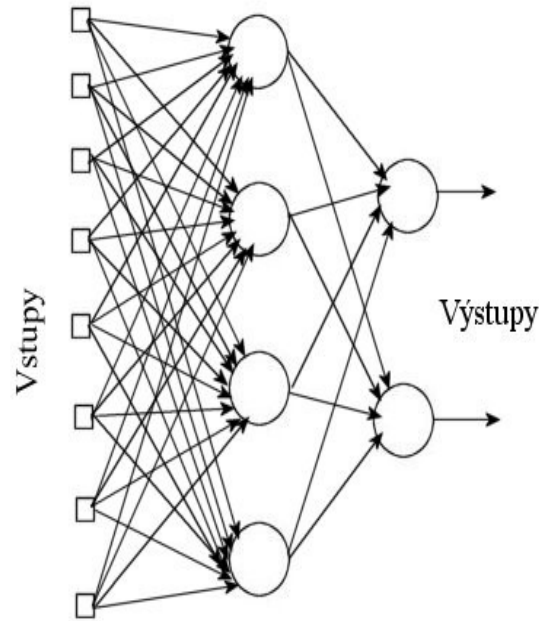
---

- MultiLayer Perceptron (MLP).
- How many layers and neurons?
- How to train ANNs and preprocess data?
- Radial Basis Function (RBF).
- Group Method of Data Handling (GMDH).

# Layered ANNs

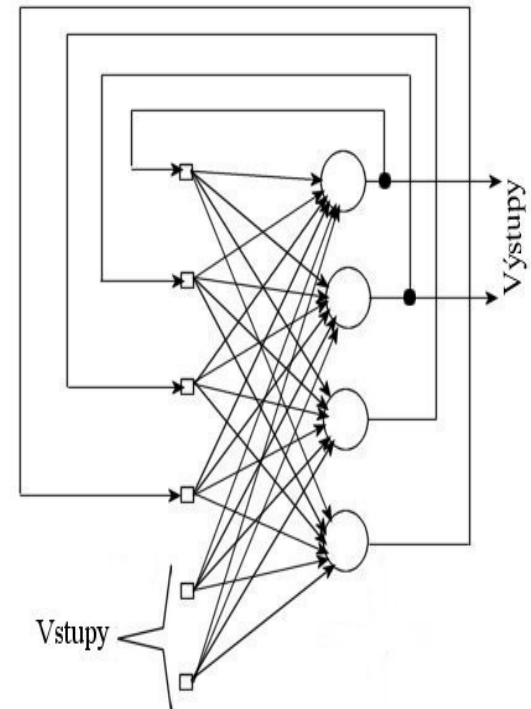


single layer



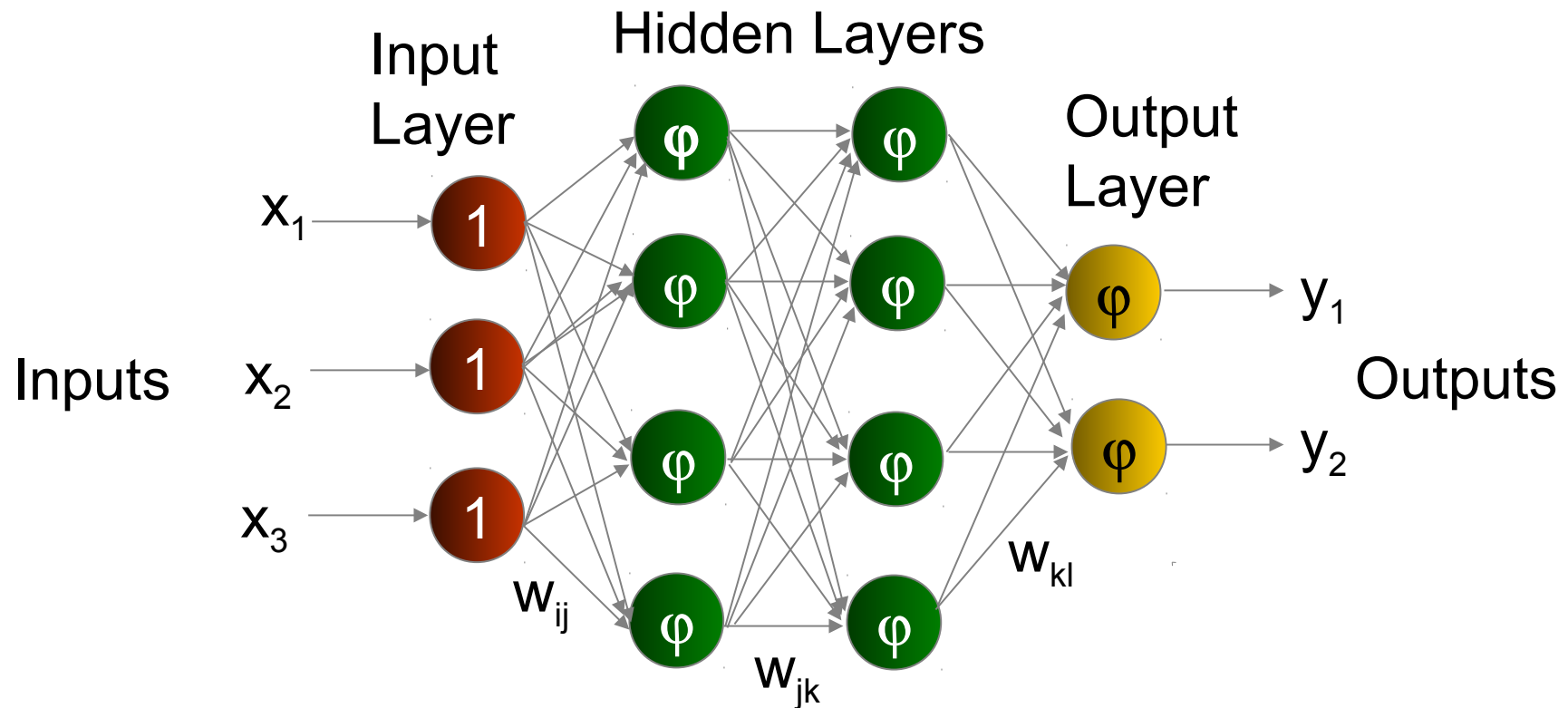
two layers

feed-forward networks

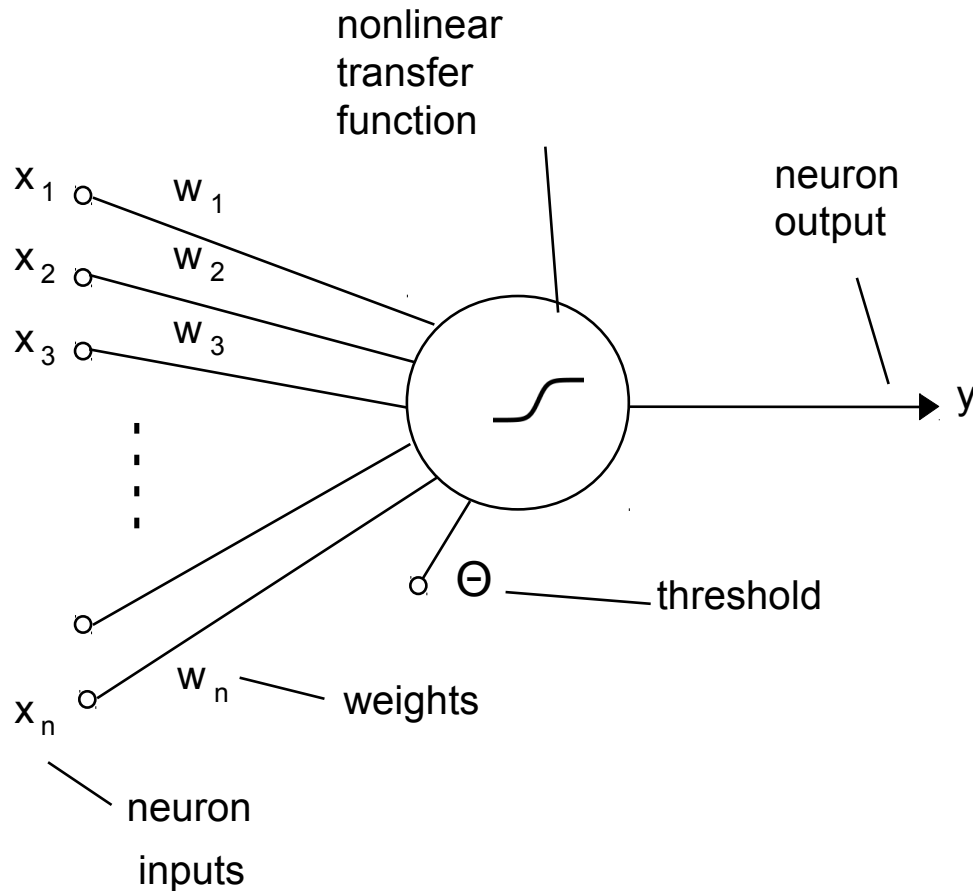


recurrent network

# MultiLayer Perceptron (MLP)

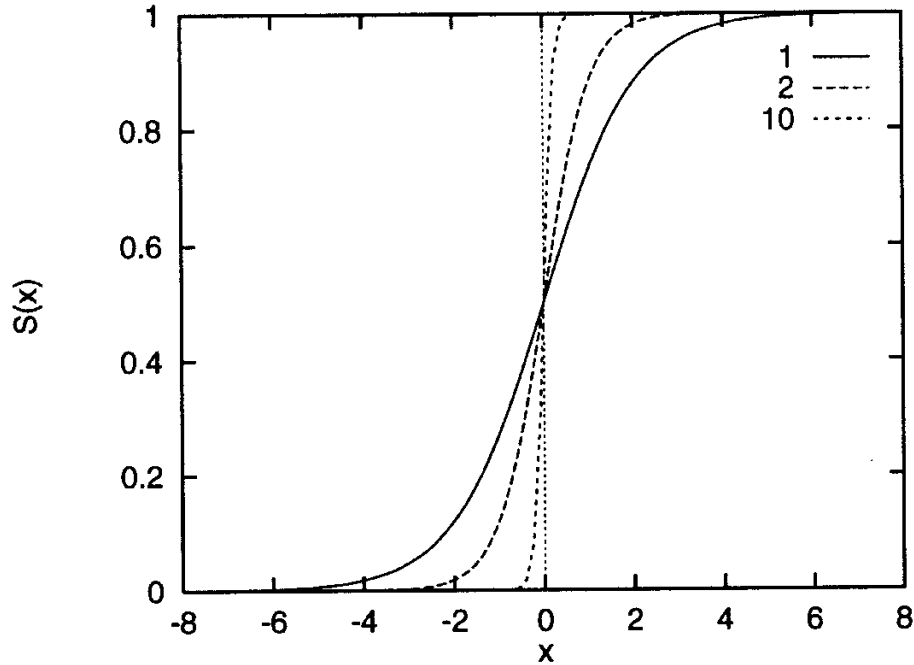


# Neurons in MLPs



McCulloch-Pitts perceptron.

# Logistic Sigmoid Function



Sigmoid for different gain/slope parameter  $\gamma$ .

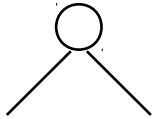
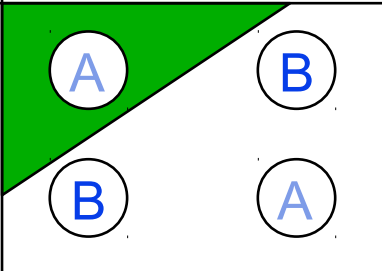
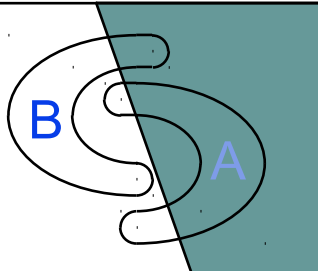
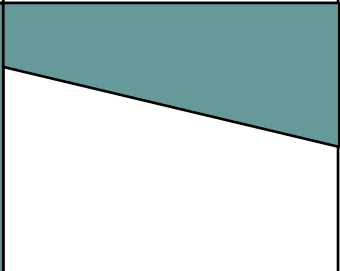
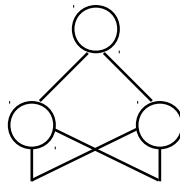
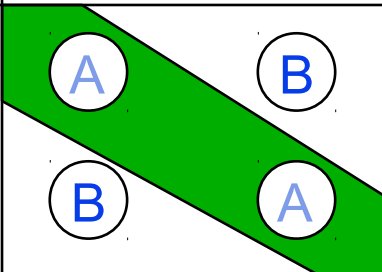
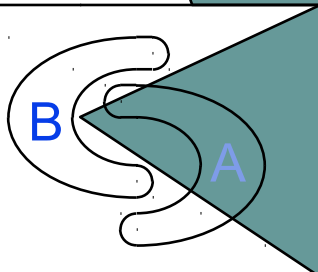
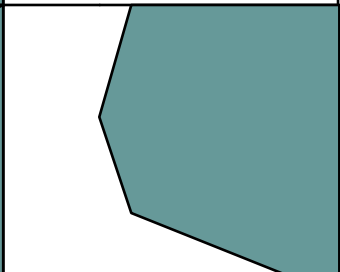
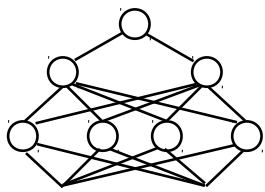
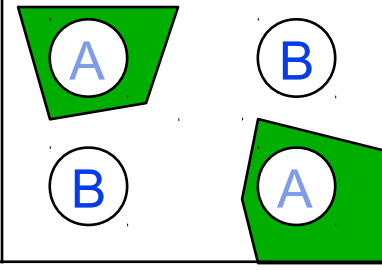
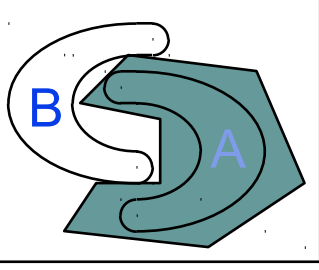
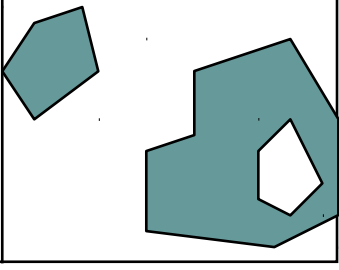
$$S(s) = \frac{1}{1 + e^{-\gamma s}}$$

- But also many other (non)-linear functions...

# How Many Hidden Layers?

## MLPs with Discrete Activation Functions

see [ftp://ftp.sas.com/pub/neural/FAQ3.html#A\\_hl](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hl) for overview

<b>Structure</b>	<b>Types of Decision Regions</b>	<b>Exclusive-OR Problem</b>	<b>Classes with Meshed regions</b>	<b>Most General Region Shapes</b>
<b>Single-Layer</b> 	<i>Half Plane Bounded By Hyperplane</i>			
<b>Two-Layer</b> 	<i>Convex Open Or Closed Regions</i>			
<b>Three-Layer</b> 	<b>Arbitrary</b> (Complexity Limited by No. of Nodes)			

# How Many Hidden Layers? Continuous MLPs

---

- **Universal Approximation** property.
- Kurt Hornik: “For MLP using **continuous, bounded, and non-constant** activation functions a **single hidden layer is enough** to approximate any function.”
- Leshno, Lin, Pinkus & Schocken 1993: **non-polynomial activation function suffices...**



# Continuous MLPs

---

- Although one hidden layer is enough for a continuous MLP:
    - we don't know how many neurons to use,
    - **fewer neurons are often sufficient for ANN architectures with two (or more) hidden layers.**
- See [ftp://ftp.sas.com/pub/neural/FAQ3.html#A\\_hl](ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hl) for example.

# How Many Neurons?

No one knows :( we have only rough estimates (upper bounds):

ANN with a **single hidden layer**:

$$N_{\text{hid}} = \sqrt{N_{\text{in}} \cdot N_{\text{out}}} ,$$

ANN with two **hidden layers**:

$$N_{\text{hid}-1} = N_{\text{out}} \cdot \left( \sqrt[3]{\frac{N_{\text{in}}}{N_{\text{out}}}} \right)^2 , \quad N_{\text{hid}-2} = N_{\text{out}} \cdot \left( \sqrt[3]{\frac{N_{\text{in}}}{N_{\text{out}}}} \right) .$$

**You have to experiment!**

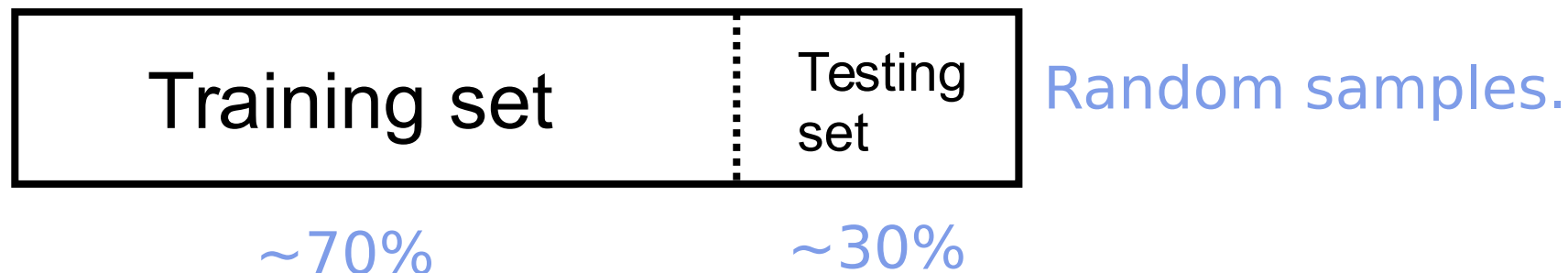
# Generalization vs. Overfitting

---

- When training ANNs we typically want them to perform accurately on new previously unseen data.
- This ability is known as the **generalization**.
- When ANN rather memorizes the training data while giving bad results on new data, we talk about **overfitting (overtraining)**.

# Training/Testing Sets

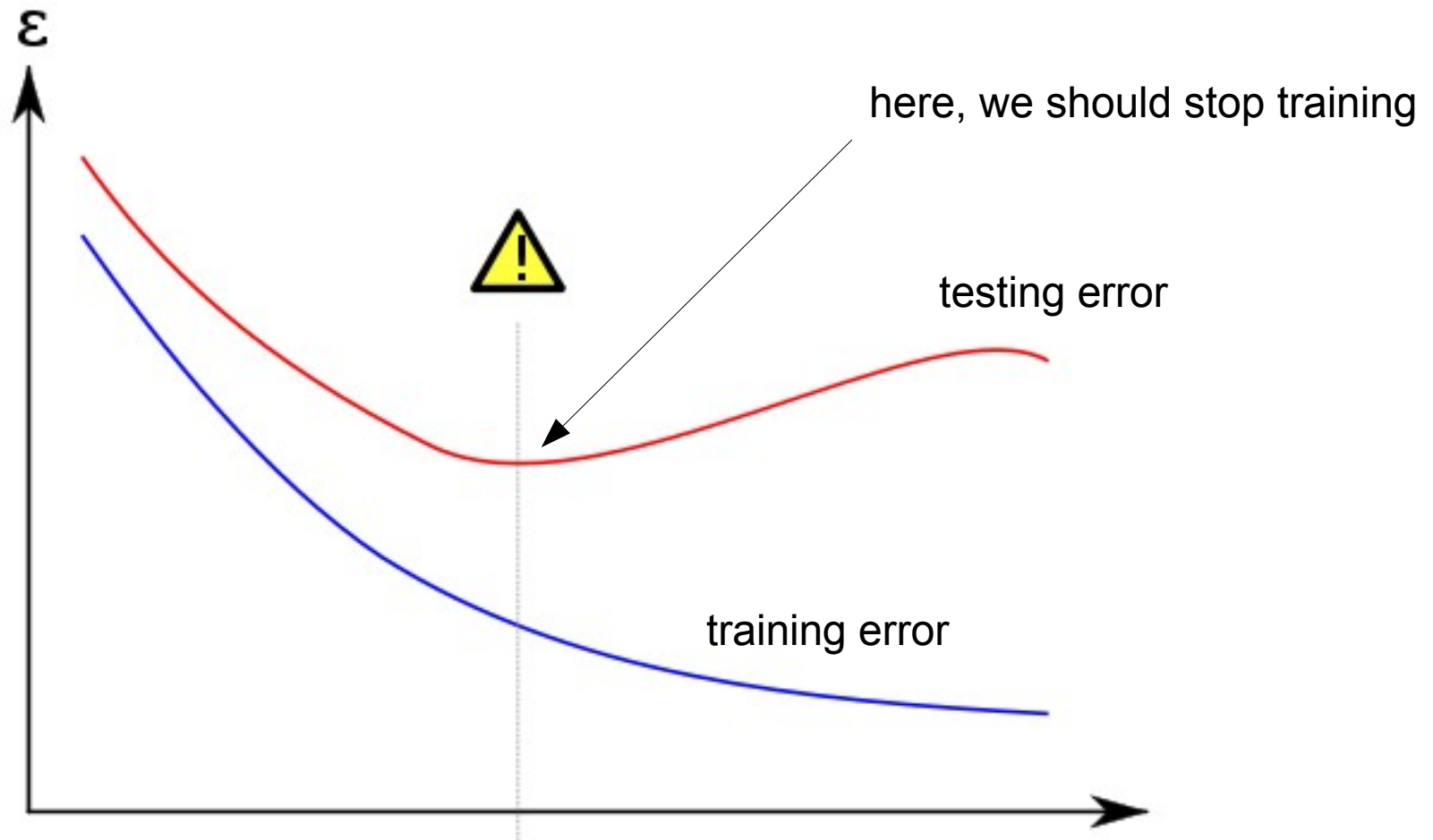
---



Used to train ANN model.

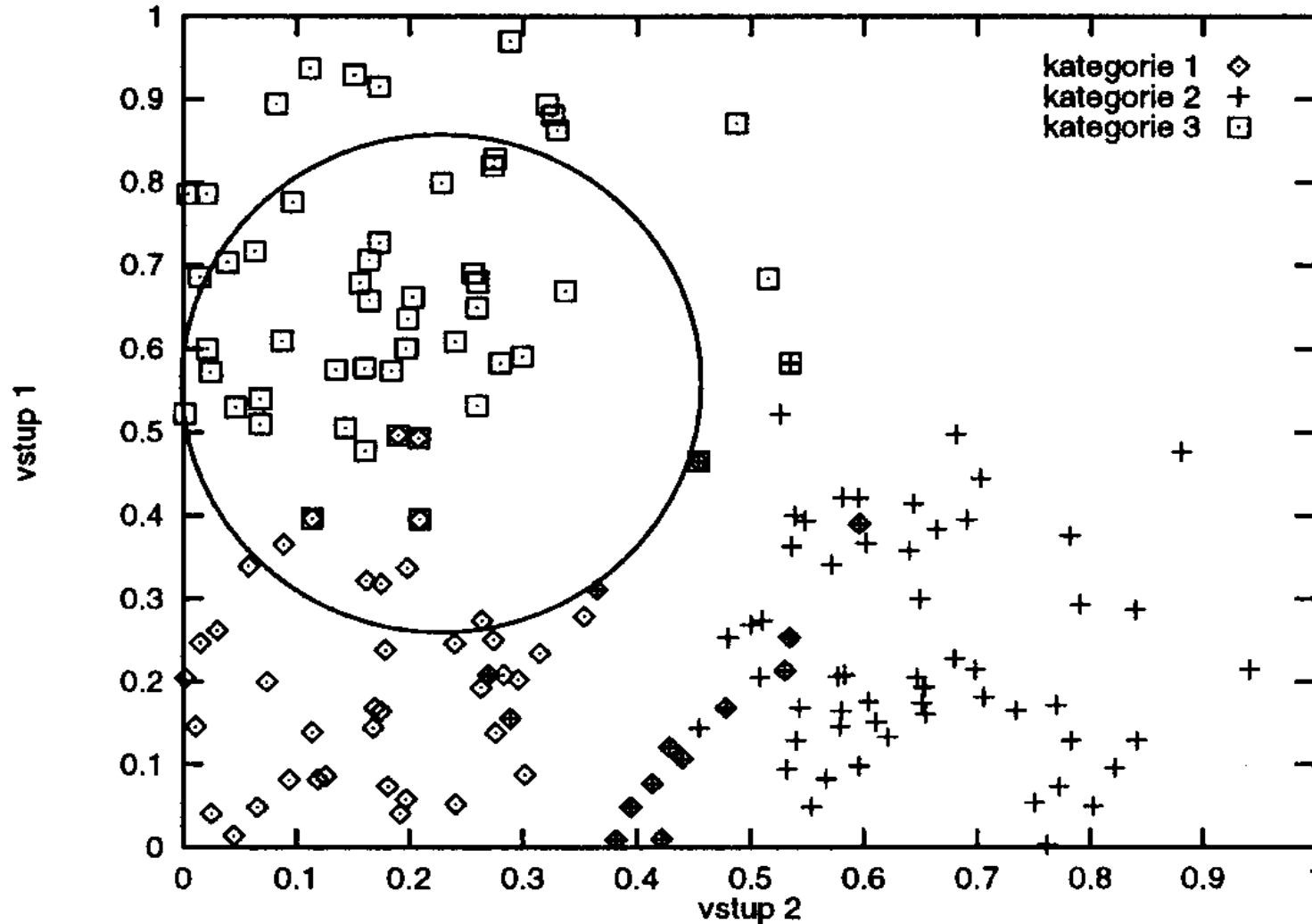
Testing set error  
→ generalization

# Overfitting Example



[http://upload.wikimedia.org/wikipedia/commons/1/1f/Overfitting\\_svg.svg](http://upload.wikimedia.org/wikipedia/commons/1/1f/Overfitting_svg.svg)

# Example: Bad Choice of A Training Set



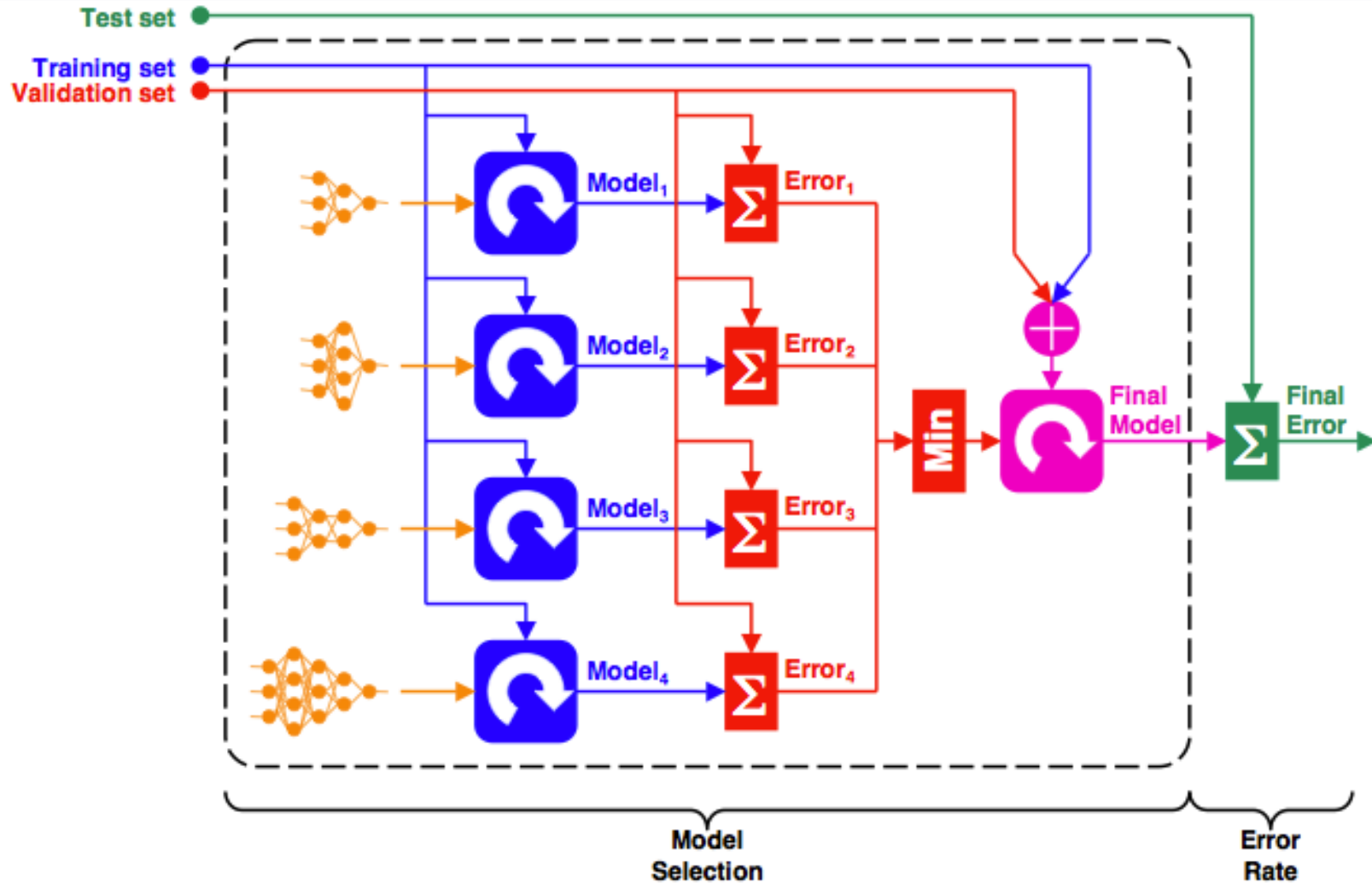
# Training/Validation/Testing Sets

---

- Ripley “Pattern Recognition and Neural Networks”, 1996:
  - **Training set:** A set of examples used for learning, that is to fit the parameters [i.e., weights] of the ANN.
  - **Validation set:** A set of examples used to tune the parameters [i.e., architecture, not weights] of an ANN, for example to choose the number of hidden units.
  - **Test set:** A set of examples used only to assess the performance (generalization) of a fully-specified ANN.
- Separated: ~60%, ~20%, ~20%.
- Note: meaning of the validation and test sets is often reversed in literature (machine-learning vs. statistics).

For example see Priddy, Keller: Artificial neural networks: an introduction (Google books, p. 44)

# Training/Validation/Testing Sets II



• Taken from Ricardo Gutierrez-Osuna's slides: [http://courses.cs.tamu.edu/rgutier/ceg499\\_s02/l13.pdf](http://courses.cs.tamu.edu/rgutier/ceg499_s02/l13.pdf)

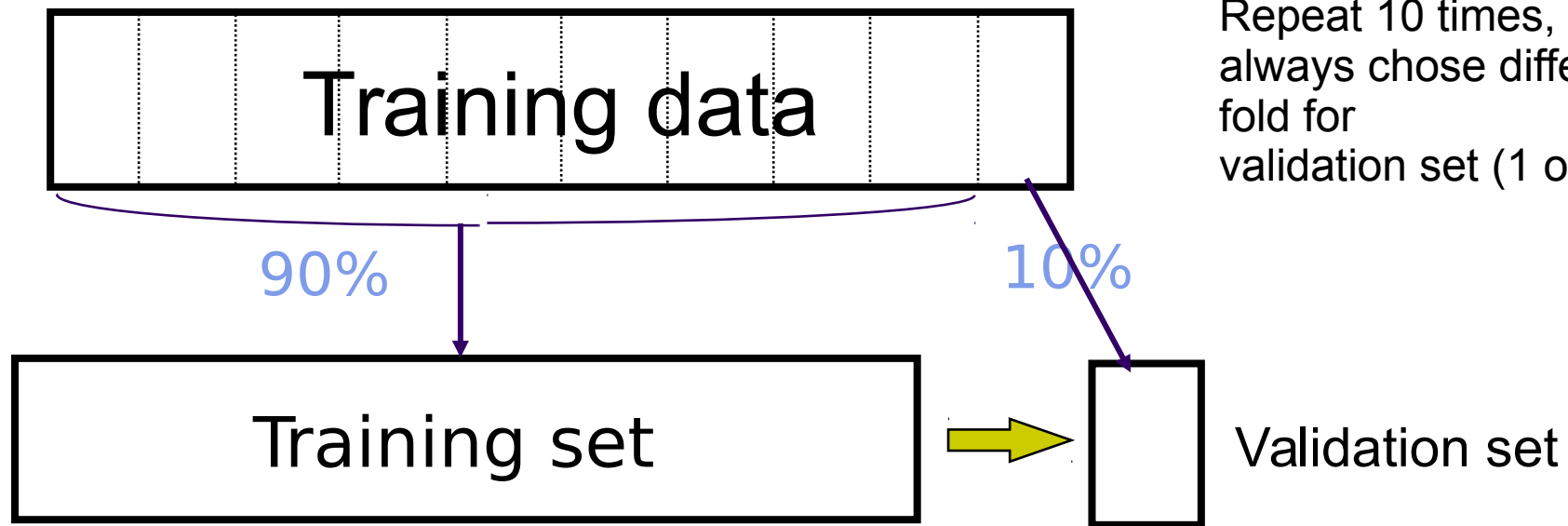


# k-fold Cross-validation

Example 10-fold cross-validation:

Split training data to 10 folds of equal size.

Repeat 10 times,  
always chose different  
fold for  
validation set (1 of 10).



Create 10 ANN models.

The cross-validation  
error is the average  
over all (10) validation sets.

Suitable for small datasets, reduces the problems  
caused by random selection of training/testing sets

# “Cleaning” Dataset

---

- Imputing missing values.
- Outlier identification:
  - the instance of the data distant from the rest of the data
- Smoothing-out the noisy data.

# Data Reduction

---

- Often needed for large data sets.
- The reduced dataset should be representative sample of the original dataset.
- The simplest algorithm: randomly remove data instances.

# Data Transformation

---

- Normalization:
  - scaling/shifting values to fit given interval or distribution
- Aggregation:
  - i.e. “binning” - discretization (continuous values to classes).

# Learning Process Notes

---

- We don't have to choose instances sequentially  
→ random selection.
- We can apply certain instances more frequently than others.
- We need often hundreds to thousands epochs to train the network.
- Good strategy might speed things up.

# Backpropagation (BP)

---

- Paul Werbos,
- 1974, Harvard, PhD thesis.
- Still popular method,
- many modifications.
- **BP is a learning method for MLP:**
  - **continuous, differentiable activation functions!**



# ANN Energy

**Backpropagation is based on a minimalization of ANN energy (= error).** Energy is a measure describing how the network is trained on given data. For BP we define the energy function:

$$E_{TOTAL} = \sum_p E_p$$

The total sum computed over all patterns of the training set.

where

$$E_p = \frac{1}{2} \sum_{i=1}^{N_o} (d_i^o - y_i^o)^2$$

we will omit "p" in following slides

Note,  $\frac{1}{2}$  – only for convenience – we will see later...

# ANN Energy II

---

The energy/error is a function of:

$$E = f \left( \vec{X}, \vec{W} \right)$$

$\vec{W}$  weights (thresholds) → **variable**,

$\vec{X}$  inputs → **fixed (for given pattern)**.



# Weight Update

- We want to update weights in opposite direction to the gradient:

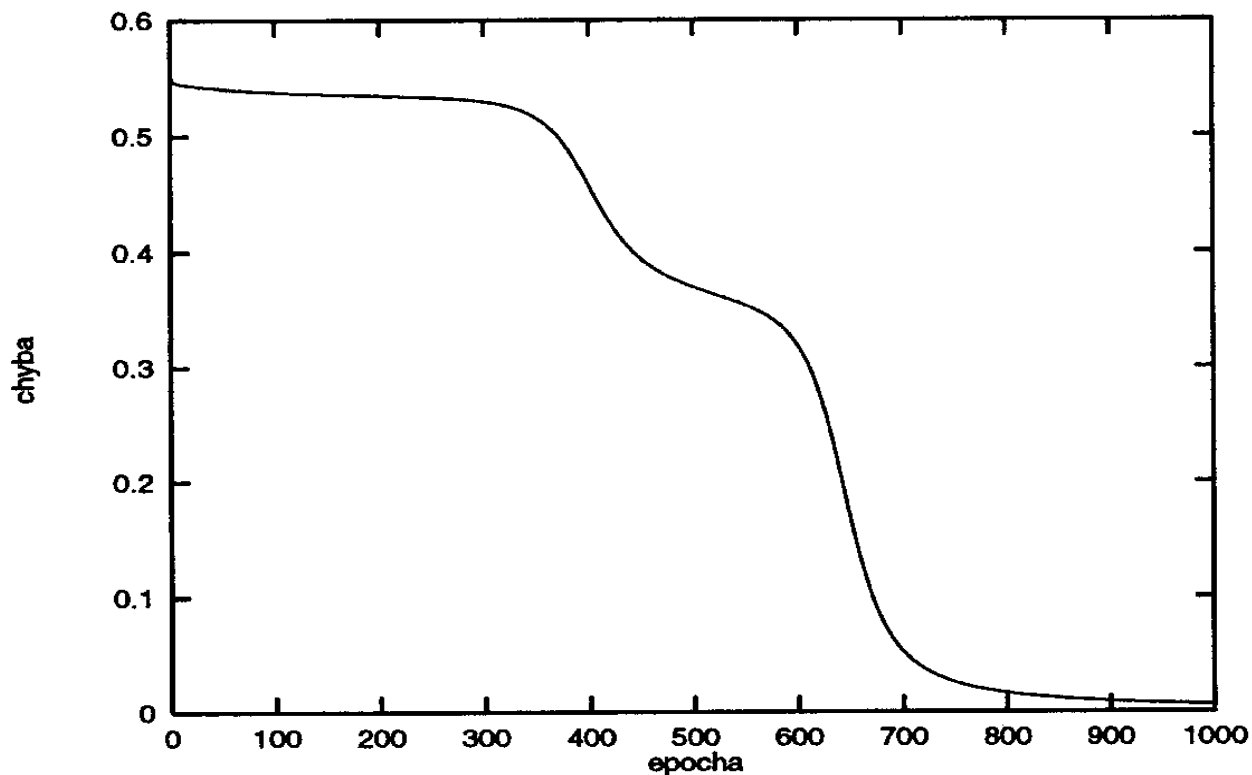
$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

weight "delta"      learning rate

Note: gradient of energy function is a vector which contains partial derivatives for all weights (thresholds)

# Typical Energy Behaviour During Learning

---

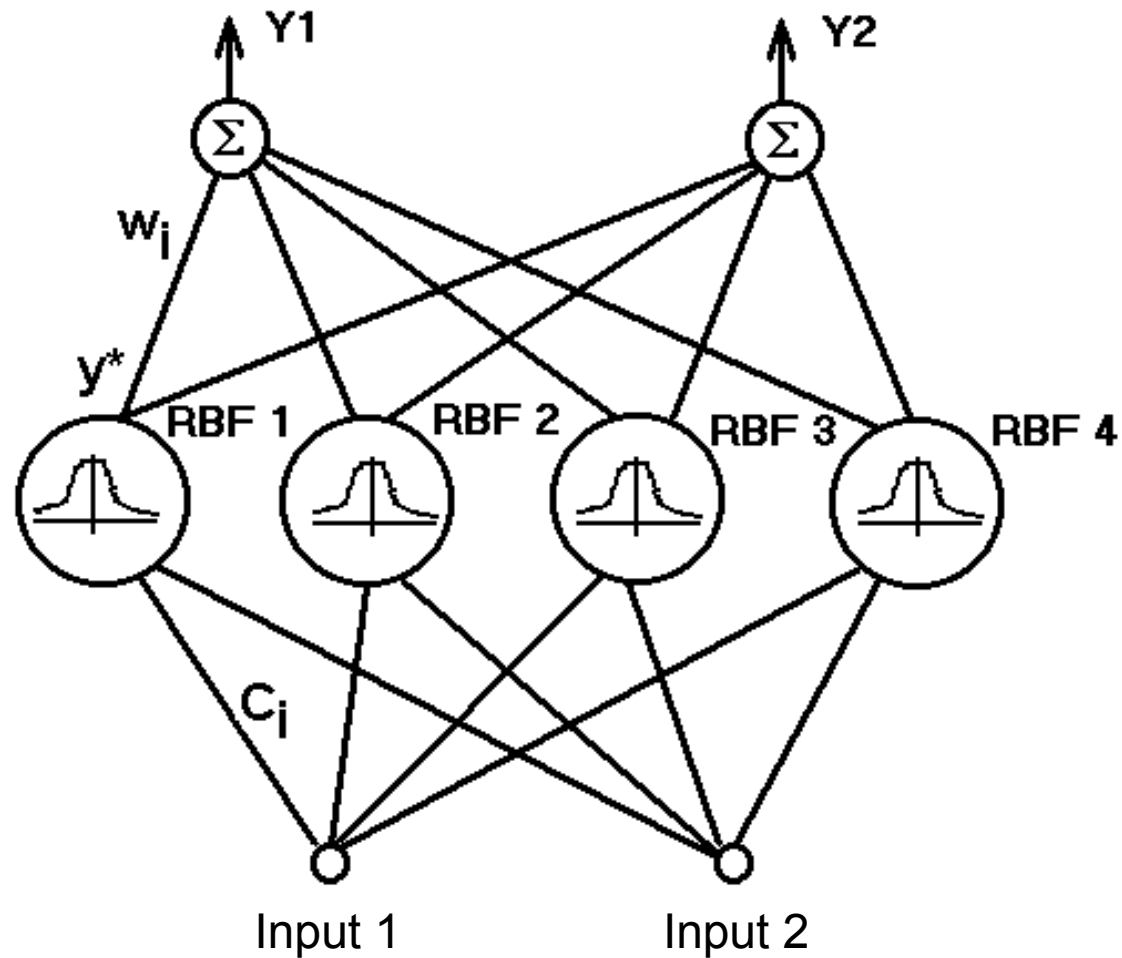


# Radial Basis Function (RBF) Networks

---

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

# RBF Architecture



# RBF Neurons

---

- **Hidden layer:**

- inner potential

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

- non-linear activation function

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

- **Output layer:**

- linear perceptron

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

# Typical RBF Activation Functions

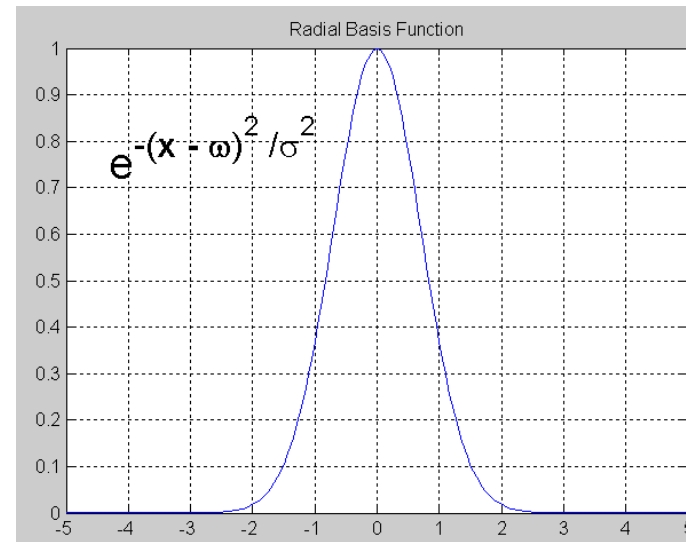
---

$$y^* = \varphi$$

$$y^* = \varphi^2 \log \varphi$$

$$y^* = (\varphi^2 + \beta)^\alpha, \beta \geq 0, 0 < \alpha < 1$$

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



# Sphere of Influence

---

- Hypersphere with center  $C$  and radius  $R$ .
- Determined by euclidean metric.
- The center is called **the prototype**.
  
- The prototype represents a cluster of input data.

# How to Determine the Sphere of Influence?

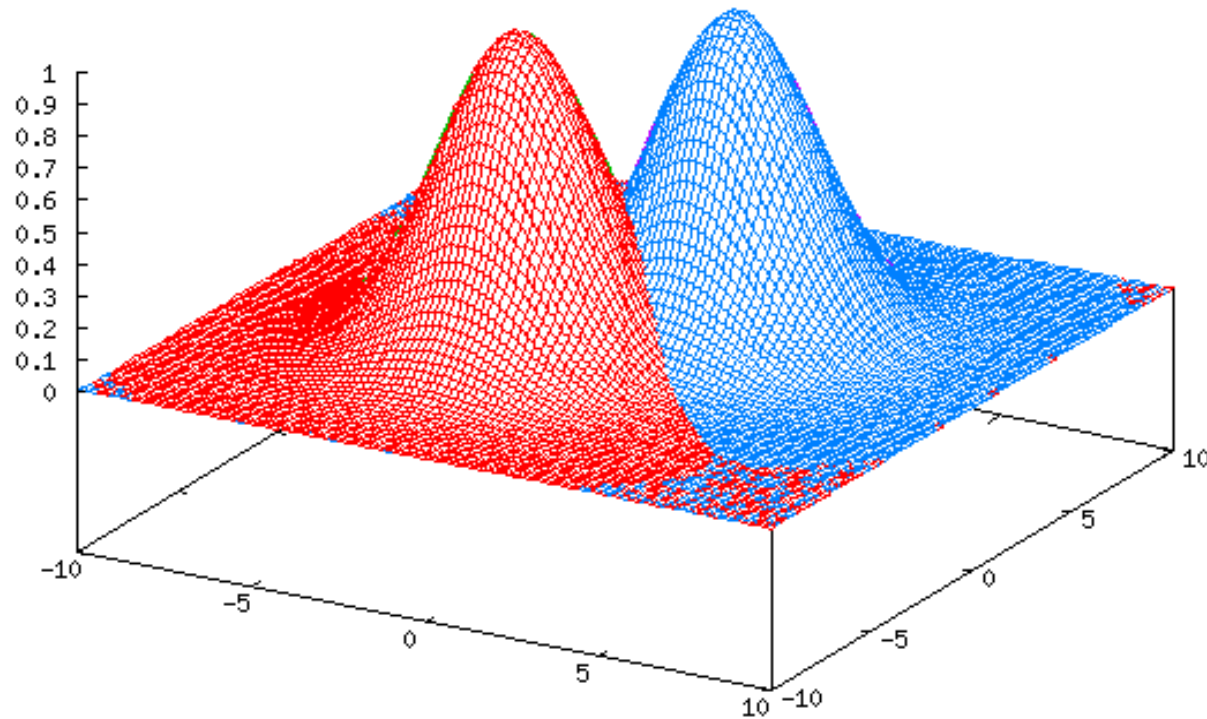
---

- Mostly we use Gauss function:
  - If an input vector has same weights as the prototype ( $\varphi = 0$ ), the function gains maximum ( $= 1$ ). This is also the maximum activation value of the neuron.
- With increasing distance of input vector from the prototype the activity of neuron decreases.
- $\sigma$  (which corresponds to a variance of normal distribution) determines a gain (width) of the activation function.



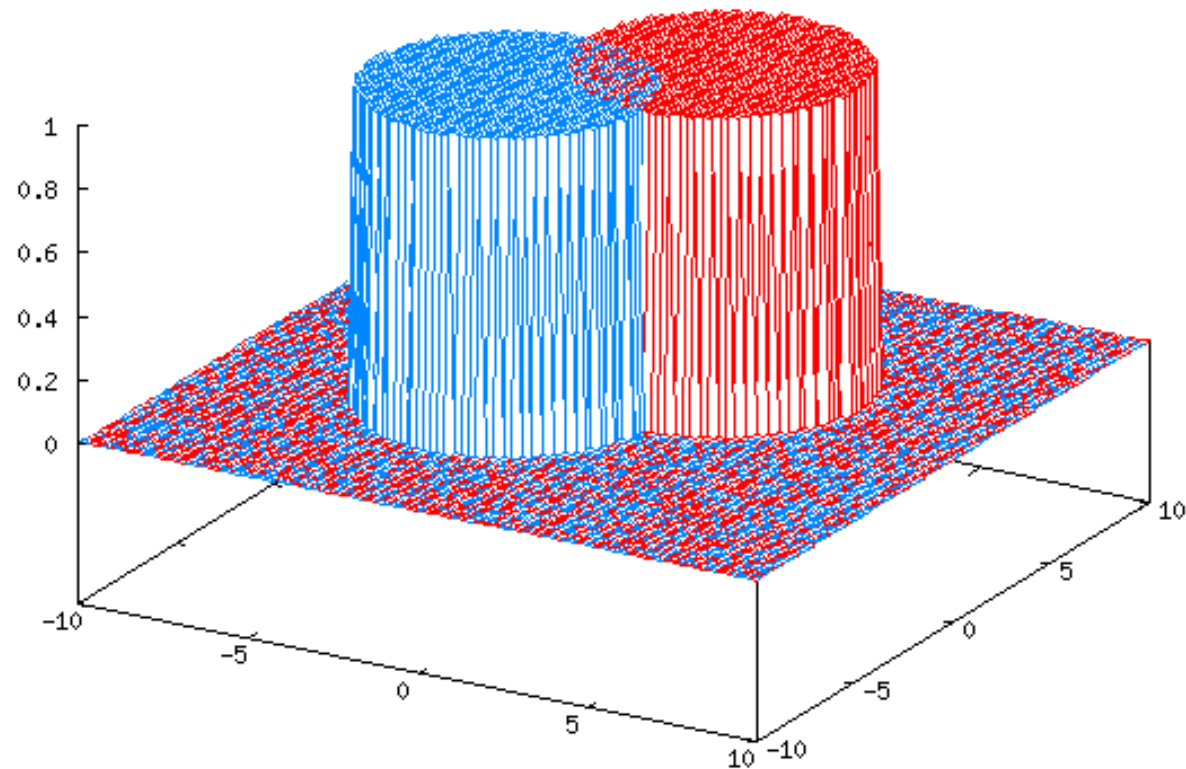
# Sphere of Influence: Example

---



# Sphere of Influence for Hardware Implementations

---

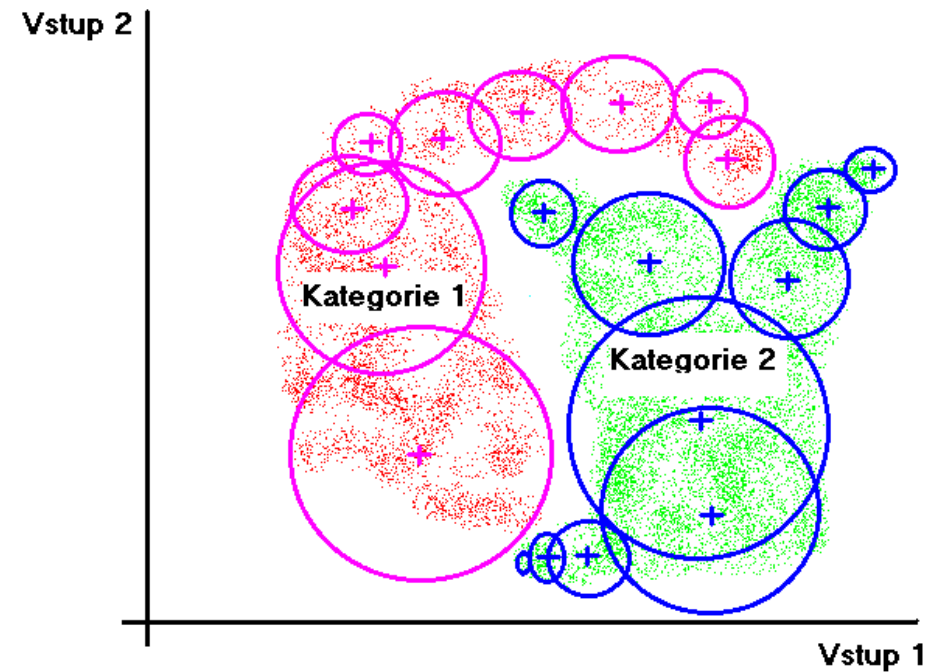
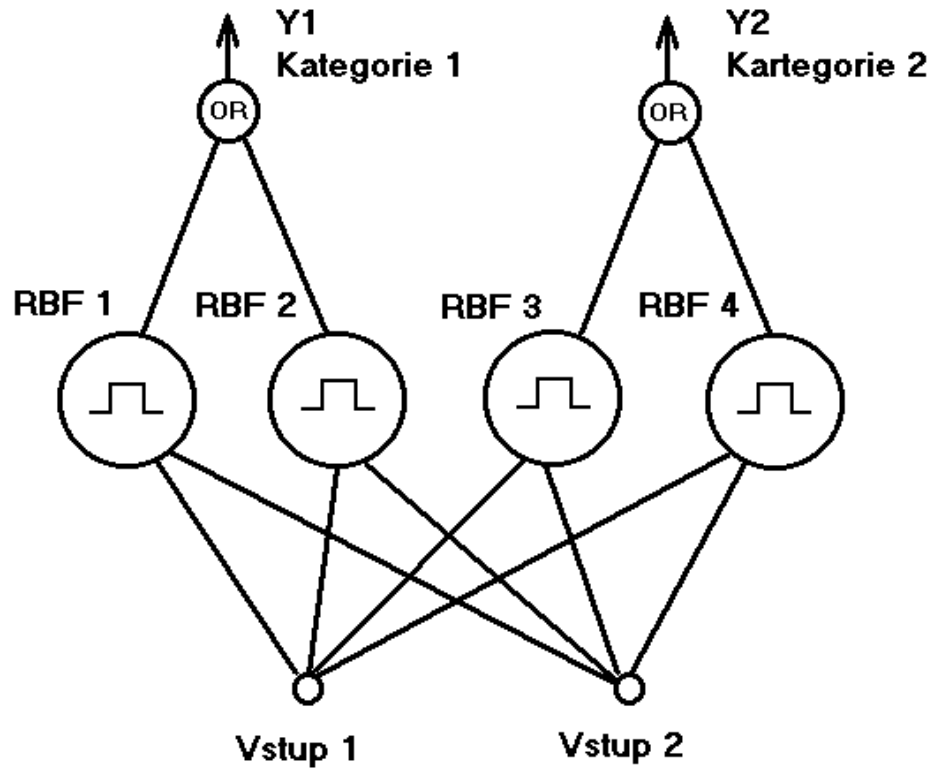


# Neuron Types Discussion

---

- 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



# Application Areas

---

- Same as MLPs.
- Universal approximator (1991: Park, Sandberg).

# Training RBF Networks

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

$$F(\mathbf{x}) = \sum_{i=1}^K w_i \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2}\right)$$

Diagram illustrating the 3-steps method for training RBF networks. The equation shows the output function  $F(\mathbf{x})$  as a sum of  $K$  Gaussian functions. The weights  $w_i$  are labeled as "supervised" (3), the centers  $\mathbf{c}_i$  are labeled as "unsupervised" (2), and the spread parameters  $\sigma_i$  are labeled as "unsupervised" (1).

3-steps method:

# Training Prototypes I

---

- Estimate the number of clusters in input data :(
- Define a *membership function*  $m(\mathbf{x})$ :
  - 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  $\mathbf{C}_i$  of RBF neurons,
  - \*) evaluate  $m()$  for all input patterns,
  - determine a new center  $\mathbf{C}_k$  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

---

- There exists also an *adaptive version* of the previous algorithm:
  - reminds perceptron learning algorithms (i.e. delta rule),
  - moves a center closer to input vectors.

# Training Prototypes IV: Adaptive K-means

---

- **Steps of adaptive K-means:**

- randomly initialize centers  $\mathbf{C}_i$ ,
- \*) take input pattern  $\mathbf{X}$ ,
- determine closest center  $\mathbf{C}_k$  and change its coordinates according to:

$$\bar{C}_k^{(t+1)} = \bar{C}_k^t + \eta (\bar{X}^{(t)} - \bar{C}_k^t)$$

where  $\eta$  denotes adaptation rate, which decreases with successive iterations.

- stop when  $\eta = 0$  or after certain # of iterations. Otherwise continue with \*).

# Training Prototypes V: Estimate # of Clusters

---

- Start with zero number of clusters,
- \*) take input pattern  $\mathbf{X}$ ,
- Find the closest cluster  $\mathbf{k}$ , If the distance between  $\mathbf{C}_k$  and  $\mathbf{X} <$  some constant  $\mathbf{r}$ , modify:

$$\bar{C}_k^{(t+1)} = \bar{C}_k^t + \eta (\bar{X}^{(t)} - \bar{C}_k^t)$$

- If the distance  $> \mathbf{r}$ , create a new center with coordinates of  $\mathbf{X}$ :

$$\bar{C}_k^{t+1} = \bar{X}^{(t)}$$

stop when  $\eta = 0$  or after certain # of iterations. Otherwise continue with \*).

- How to choose  $\mathbf{r}$  ?

# Training Prototypes V: Determine $\sigma$

---

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

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

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

Other approach uses fixed  $\sigma$ .

# Training Output Neurons

---

- Find weights by minimization of this energy function:

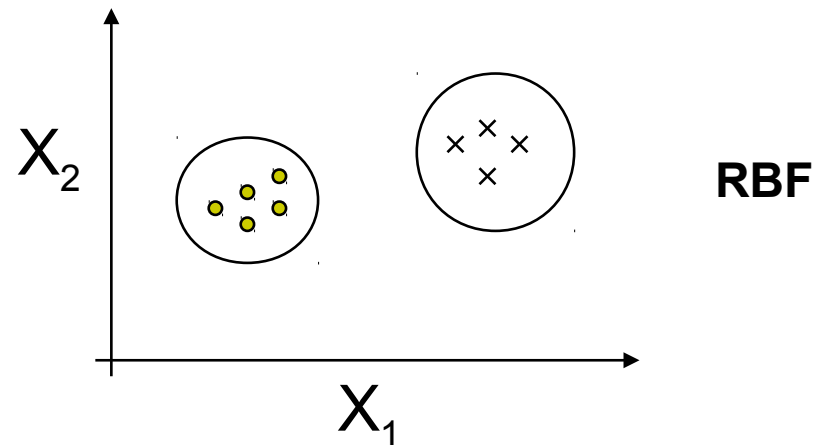
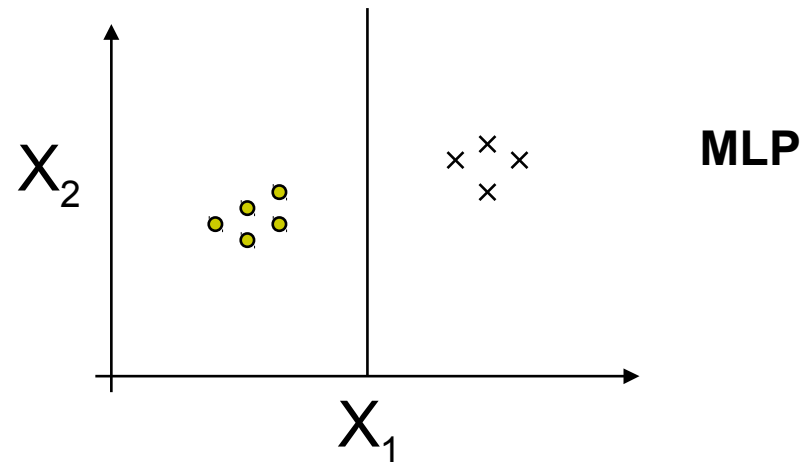
$$\Delta \bar{w}^{(t)} = -\eta \nabla E^{(t)} = \eta (D^{(t)} - Y^{(t)}) Y^{*(t)}$$

- 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.

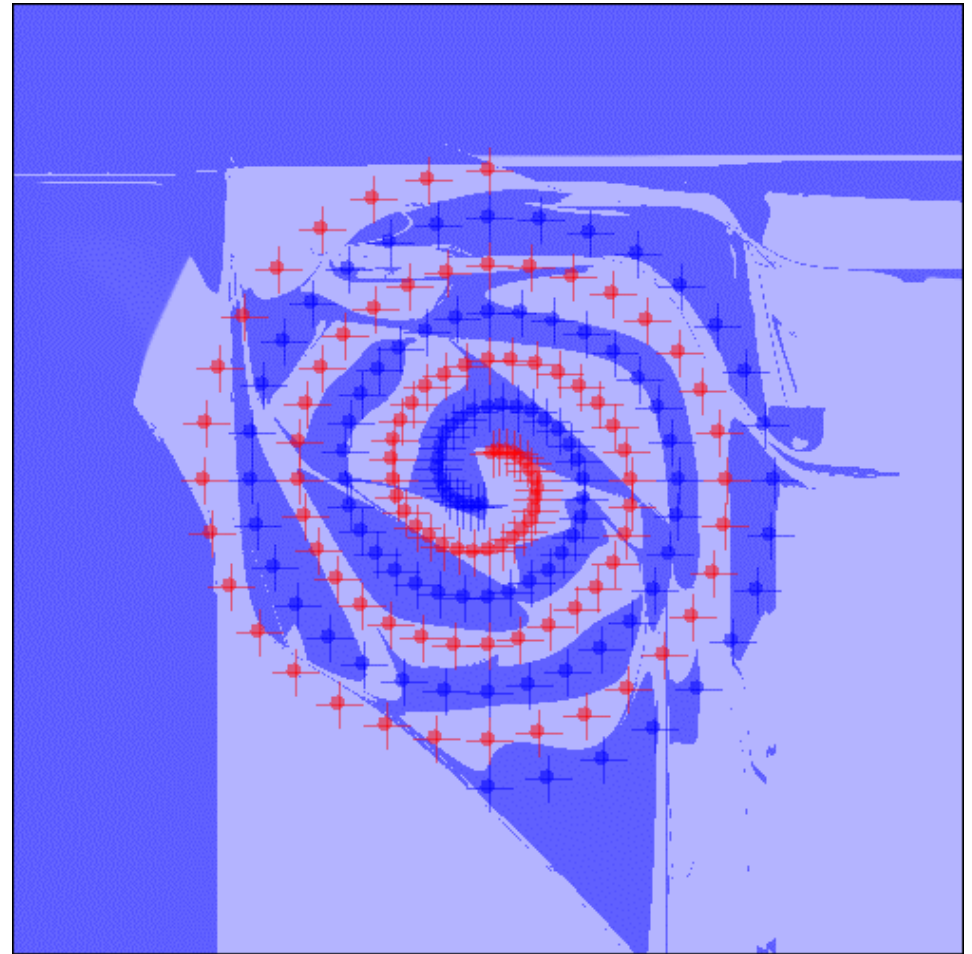


# The Spiral problem

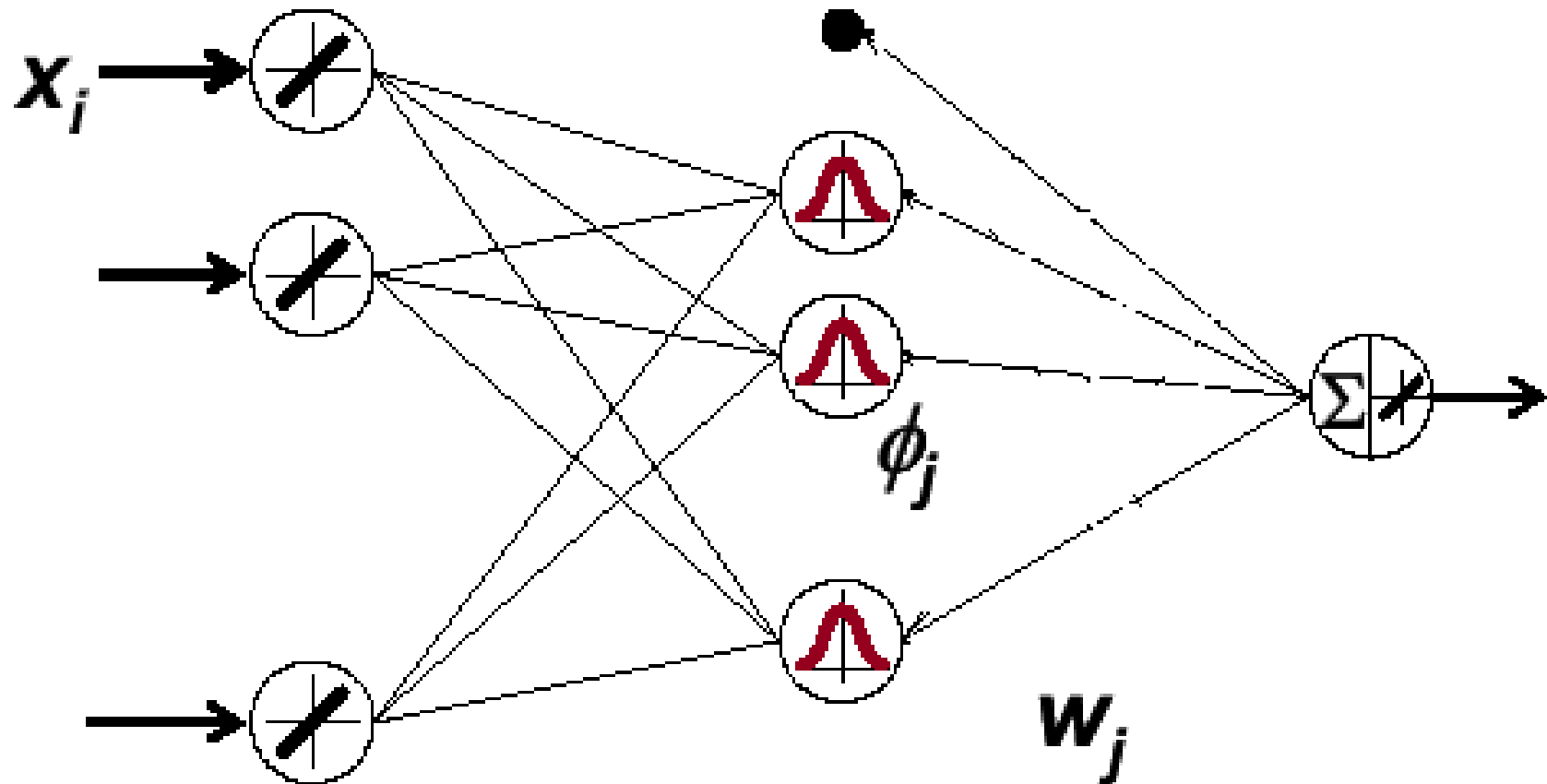
---

One of the most challenging artificial benchmarks: separate two inter-winded spirals.

How can MLP solve it?  
How about RBF?

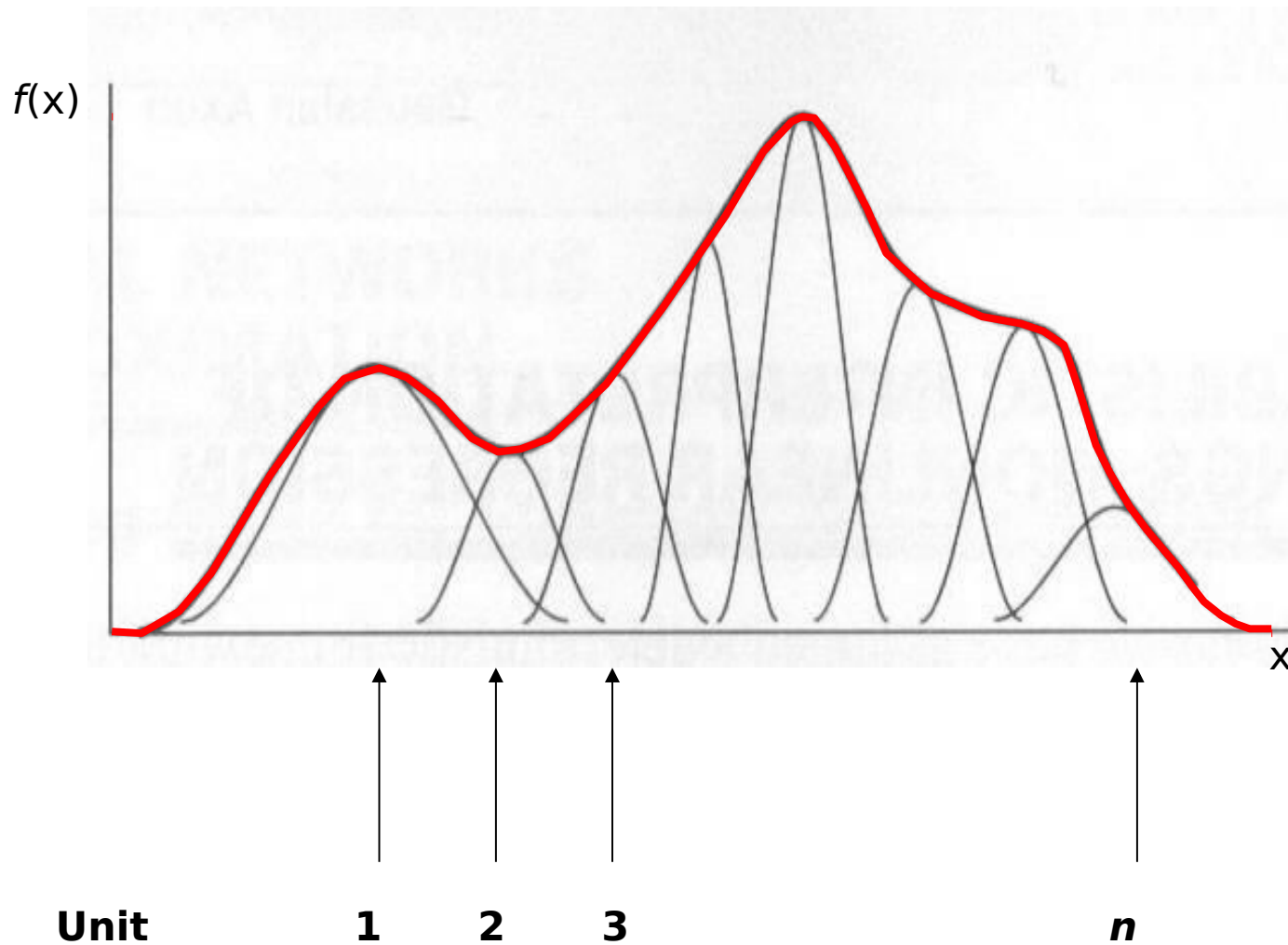


# RBF for Approximation





# Approximation by RBF: Example



# GMDH

- **Group Method of Data Handling.**
- By A.G. Ivakhnenko, 1968.
- Network is constructed by **induction**.
- Multiple layers.
- Feed-forward.
- Supervised.
- See <http://www.gmdh.net> ...



# Induction vs. Deduction

---

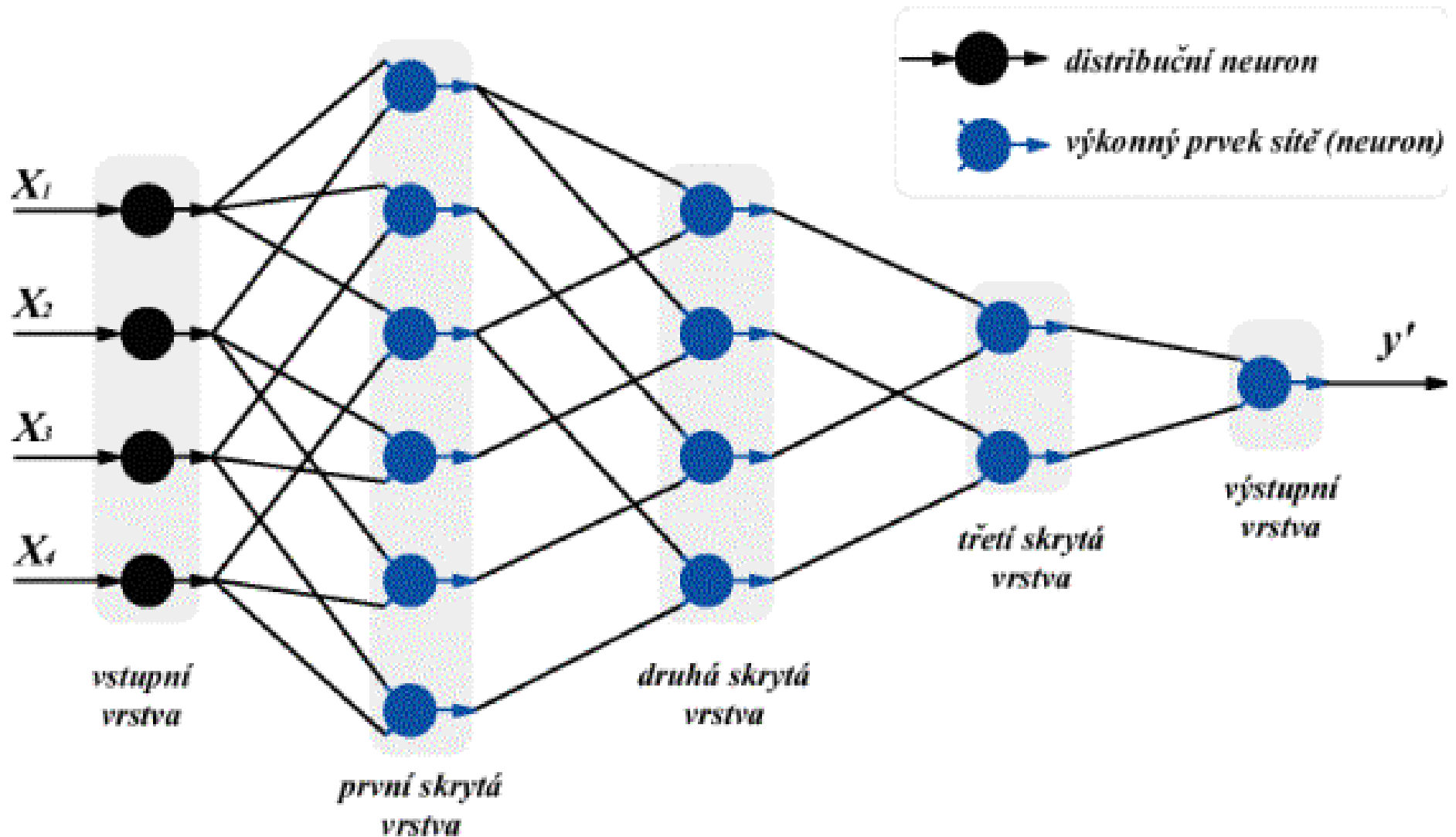
- There are two approaches to create a model of a system:
  - **data-driven**: based on observations of a real system. The task is to infer general rule from specific cases → this the way GMDH model is built → **induction**,
  - **theory-driven**: based on theoretical knowledge of system functionality → the way to create mathematical models → **deduction**.

# 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

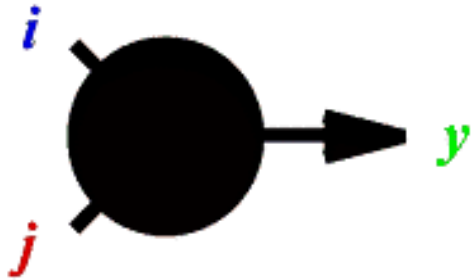
---

- 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

---

input  
variables

output  
variable

$v_1$  ○

$v_2$  ○

$v_3$  ○

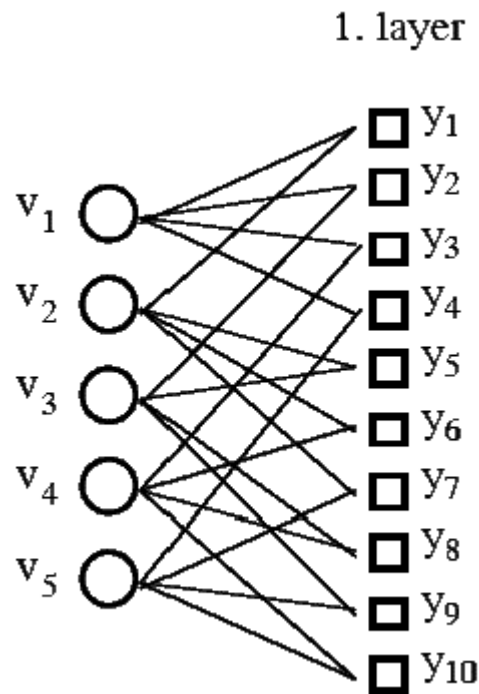
$v_4$  ○

$v_5$  ○

○  $y$

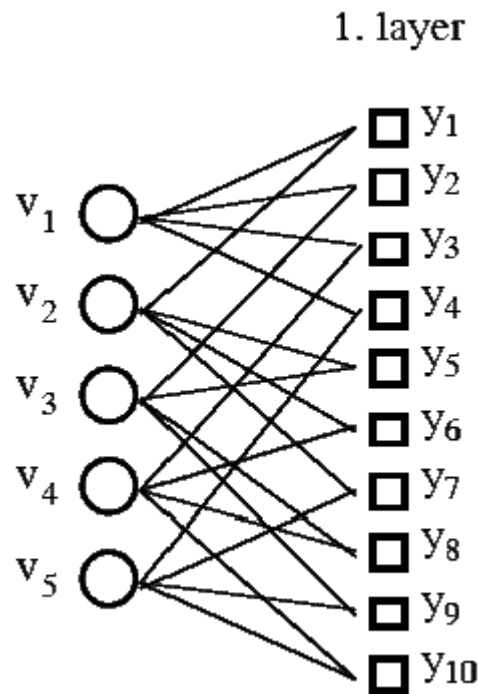
Lemke,Müller:SELF-ORGANIZING DATA MINING BASED ON GMDH PRINCIPLE

# MIA Training III



- 1<sup>st</sup> 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



- 1<sup>st</sup> 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?

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

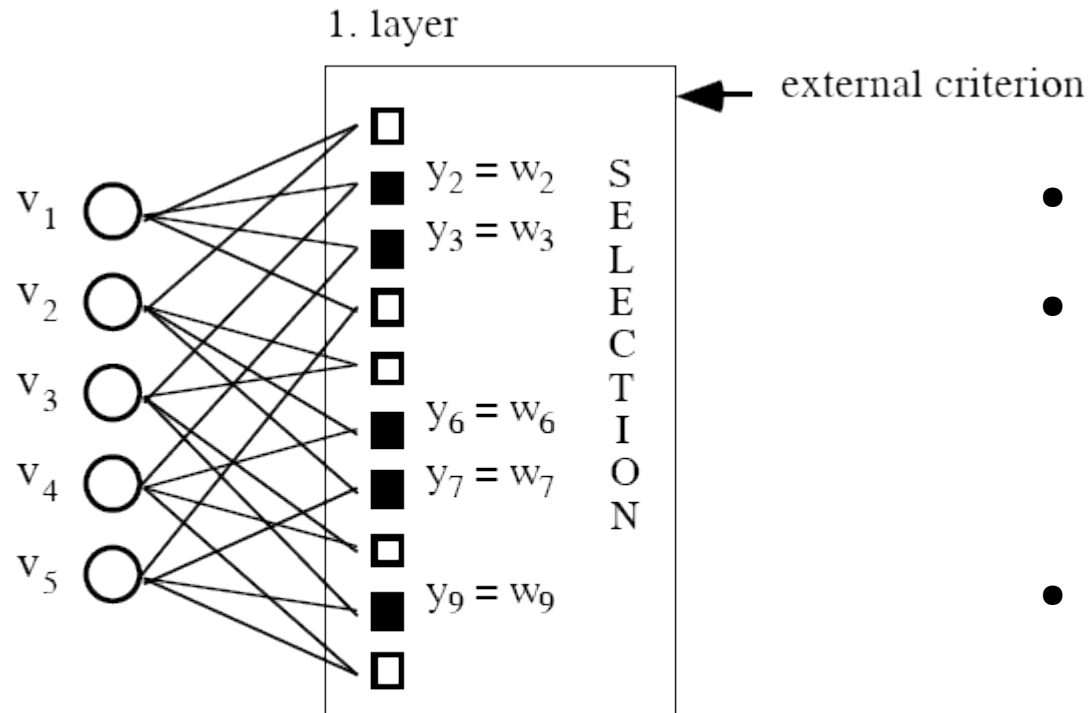
# MIA Training IV

---

- 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

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

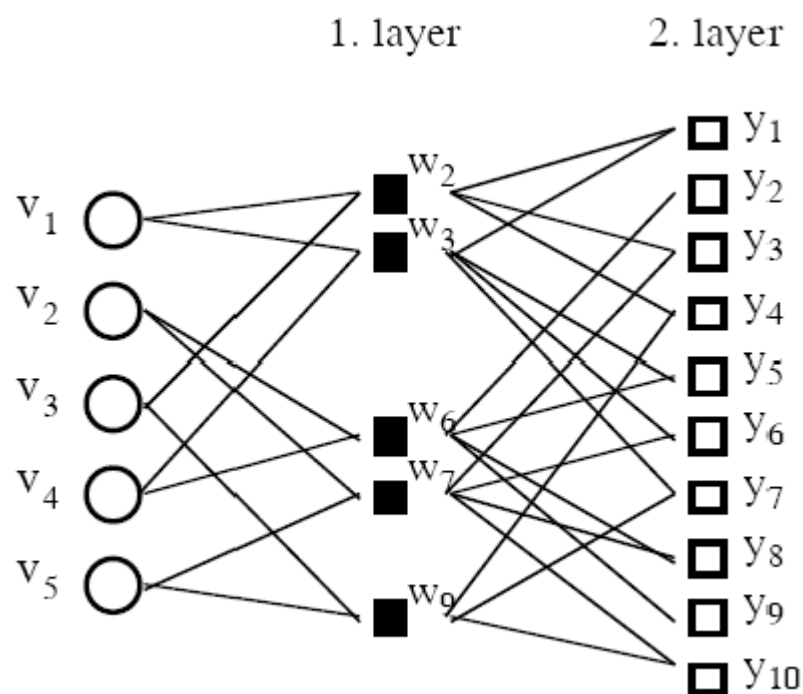
# MIA Training V



- selected neuron (survives)
- not selected neuron (dies)

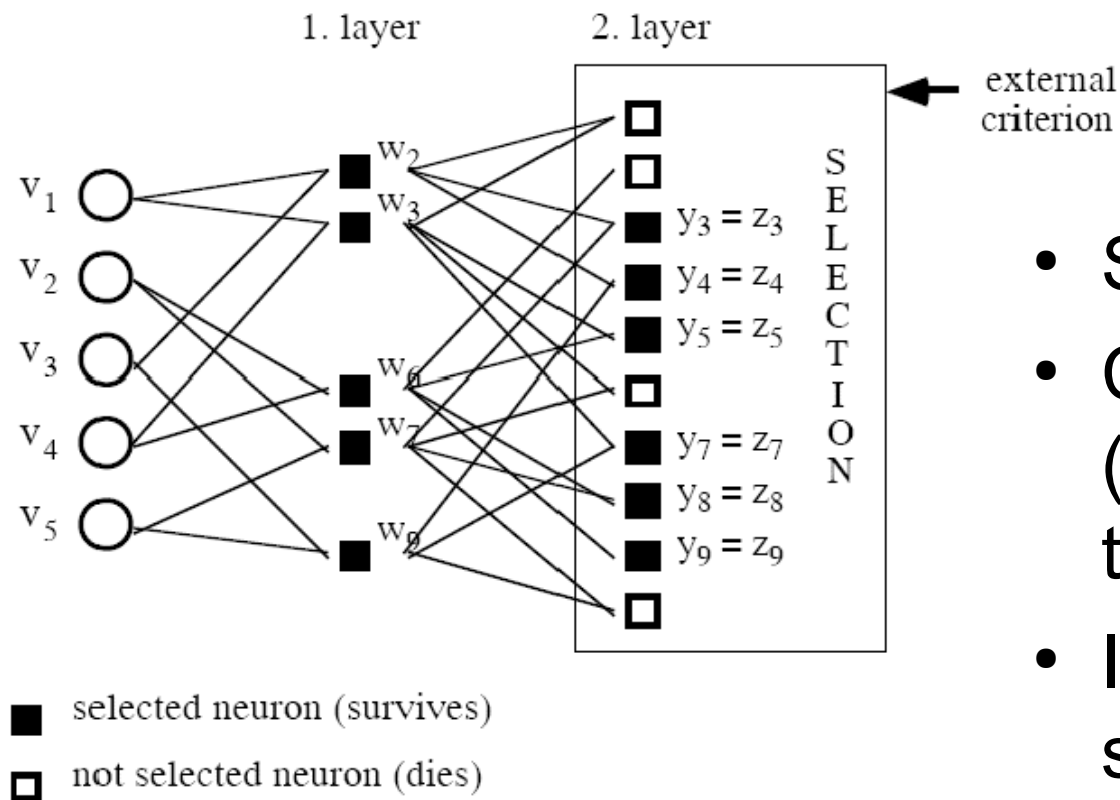
- Next step: Selection.
- According to a criterion (see later) some neurons survive, others not...
- If we are satisfied with the output (low error) → stop.
- Similar to evolutionary algorithms...

# MIA Training VI



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

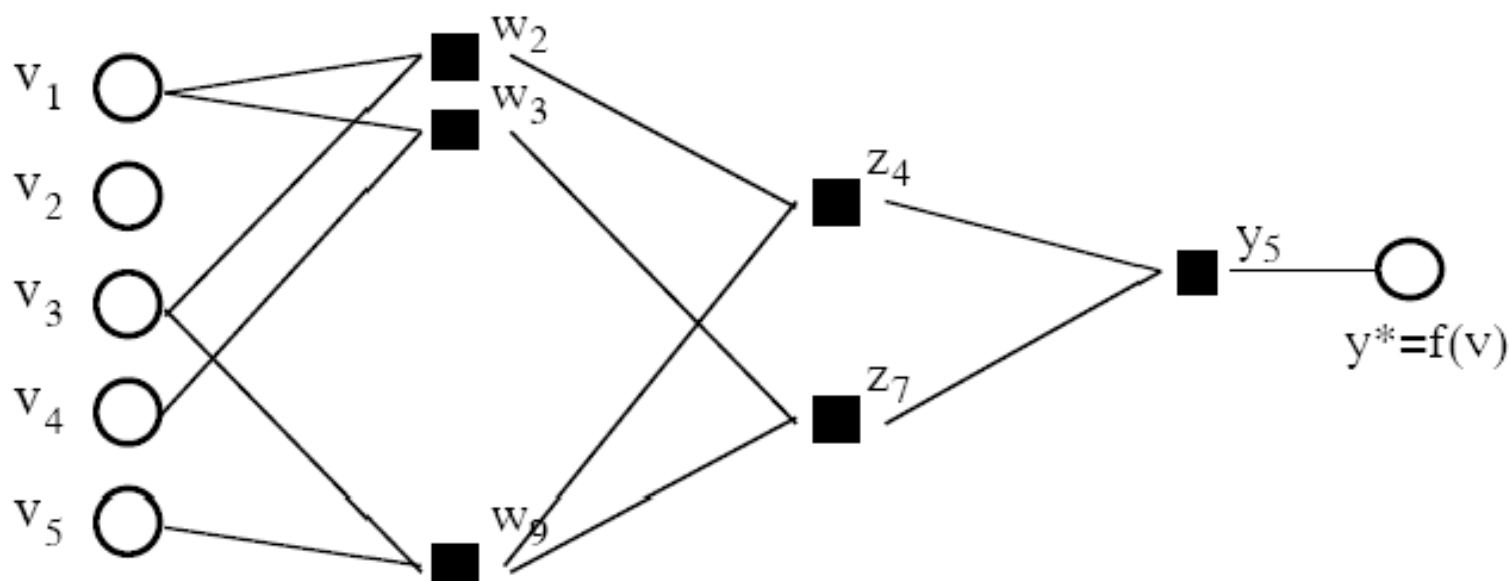
# MIA Training VII



- 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



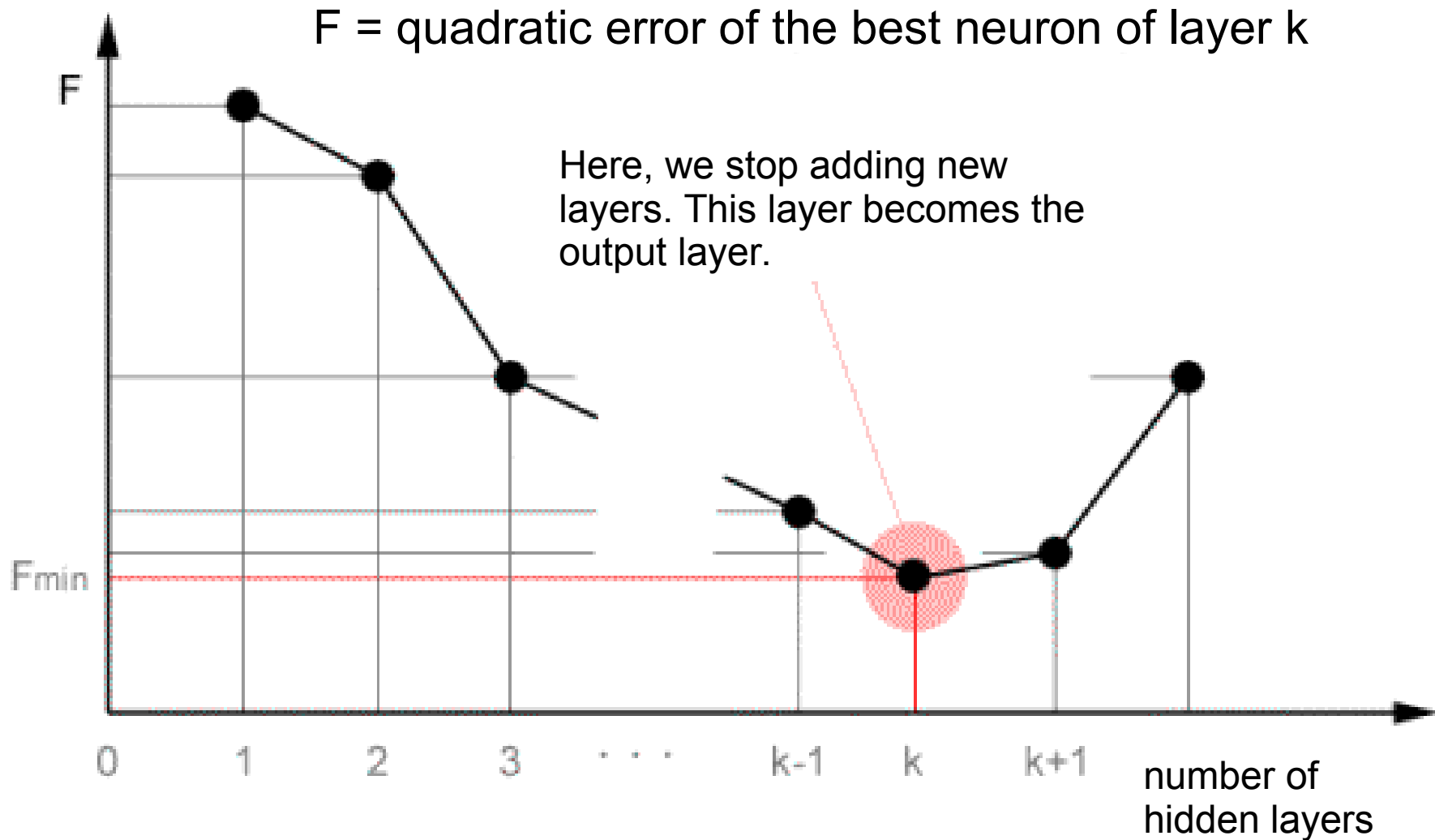
explicit analytically available optimal complex model.

*network status:*

order of regression model  $y^*$ :  $\leq 8$

number of variables  $v$  in model  $y^*$ : 4 (from initially 5 variables)

# When to End Training?



# Possibilities to Determine Fmin :)

$\frac{N}{N-p} s_e^2$	$s_{e,p}^2$	$s_e^2 + s_y^2 \ln N$	$\frac{P}{N}$	Schwarz	<i>Regularity criterion</i>	$AB = \frac{1}{N_B} \sum_{i=1}^{N_B} [y_i - \hat{y}_i(A)]^2 \rightarrow \min.$
$\frac{N+p}{N-p} s_e^2$	$FPE$	$N \ln s_e^2 + p \ln N$		Rissanen	<i>Discrimination criterion</i>	$\sum_{i=1}^N (y_i - \hat{y}_i)^2$
$s_e^2 + 2\sigma_p^2 \frac{P}{N}$	$PSE$	$\frac{2ps_e^2}{N-p} - Ns_{y^m}^2$		Kocerka		$\delta^2 = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} < 1.0$
$\frac{Ns_e^2}{\sigma_c^2} + 2p - N$	Mallow $C_p$	$\frac{\sum_{t=1}^N (\bar{y}_t^m - y_t)^2}{N}$		$i^2(N)$	<i>Validation criterion</i>	$R_{ij} = 1 - \sqrt{\frac{\frac{1}{N-P_i} \sum_{t=1}^N (y_t - y_t^i)^2}{\frac{1}{N-P_j} \sum_{t=1}^N (y_t - y_t^j)^2}}$
$N \ln s_e^2 + 2p + c$	AIC	$\sum_{t=1}^N y_t^2$				
$N \ln s_{e,p}^2 + p \ln \left( \frac{s_{y^m}^2}{s_{e,p}^2} \frac{N}{p} \right)$	BIC	$s_e^2$		$\left[ I - \lambda \sqrt{\frac{k(\ln(\frac{N}{k}) + I) - \ln \alpha}{N}} \left( I - \frac{I}{4} \frac{\ln(\frac{N}{k}) + I - \ln \alpha}{N} \right) \right]$		Vapnik
$s_e^2 = \frac{1}{N} \sum_{t=1}^N (y_t - y_t^m)^2 = \frac{1}{N} \sum_{t=1}^N e_t^2; \quad s_y^2 = \frac{1}{N} \sum_{t=1}^N (y_t - \bar{y})^2; \quad s_{y^m}^2 = \frac{1}{N} \sum_{t=1}^N (y_t^m)^2$						

# Most Common

---

**Regularity Criterion:**

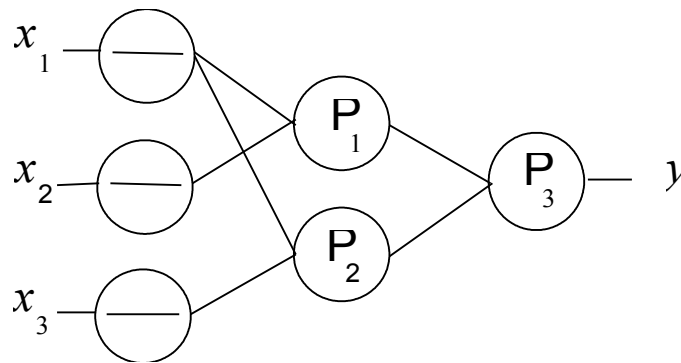
$$AB = \frac{1}{N_B} \sum_{i=1}^{N_B} [y_i - \hat{y}_i(A)]^2 \rightarrow \min.$$

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

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

# GMDH → Mathematical Model

- Example:



$P_i$  are linear

$$y = P_3(P_2(x_1, x_3), P_1(x_1, x_2)) = a_{31}(a_{11}x_1 + a_{12}x_3 + a_{13}) + a_{32}(a_{21}x_1 + a_{22}x_2 + a_{23}) + a_{33} =$$
$$(a_{31}a_{11} + a_{32}a_{21})x_1 + (a_{32}a_{22})x_2 + (a_{31}a_{12})x_3 + a_{31}a_{13} + a_{32}a_{23} + a_{33}$$

# GMDH → Mathematical Model II

---

- Problems:
  - real systems are too complex and...
  - equations become too confusing.
- Solution?
  - Visualize model behaviour!

# COMBI vs. MIA

---

- What's the difference between partial and full induction?
- COMBI uses supervised learning, too.
- COMBI employs different types of neurons, but:
  - same in a single layer,
  - “more complex” in following layers,
  - not connected to a previous layer, but to network inputs
  - always combine “the most fit”.

# Notes

---

- The full induction network (COMBI) works with a full input space all the time.
- It tries to apply successively complex transformations.
- The partial induction network (MIA) applies only a single transformation to most promising input subspaces.
  - It is a heuristic which speeds up the algorithm.



# 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...

# Next Lecture

---

- Recurrent ANNs = RNNs