Artificial Neural Networks – Basics of MLP, RBF and Kohonen Networks

Jerzy Stefanowski
Institute of Computing Science

Lecture 13 in Data Mining
for M.Sc. Course of SE

version for 2010

Acknowledgments

- Slides are also based on ideas coming from presentations as:
  - Rosaria Silipo: Lecture on ANN. IDA Spring School 2001
  - Prévotet Jean-Christophe (Paris VI): Tutorial on Neural Networks
  - Włodzisław Duch: Lectures on Computational Intelligence
  - Few others
- and many of my notes for a course on Machine Learning and Neural Networks (Polish Language ISWD – see my personal web page for more slides)

Outline

- Introduction
  - Inspirations
  - The biological and artificial neurons
  - Architecure of networks and basic learning rules
- Single Linear and Non-linear Perceptrons
  - Delta learning rule
- MultiLayer Perceptrons
  - MLPs and Back-Propagation
  - Tuning parameters of BP
- Radial Basis Functions
  - Architectures and learning algorithms
- Competitive Learning
  - Competitive Learning, LVQ, Kohonen self-organizing maps.
- Applications and Software Tools
- Final Remarks

Introduction

- Some definitions
  - "… a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes." - DARPA (1988)
  - A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class output of the input signals

Some properties

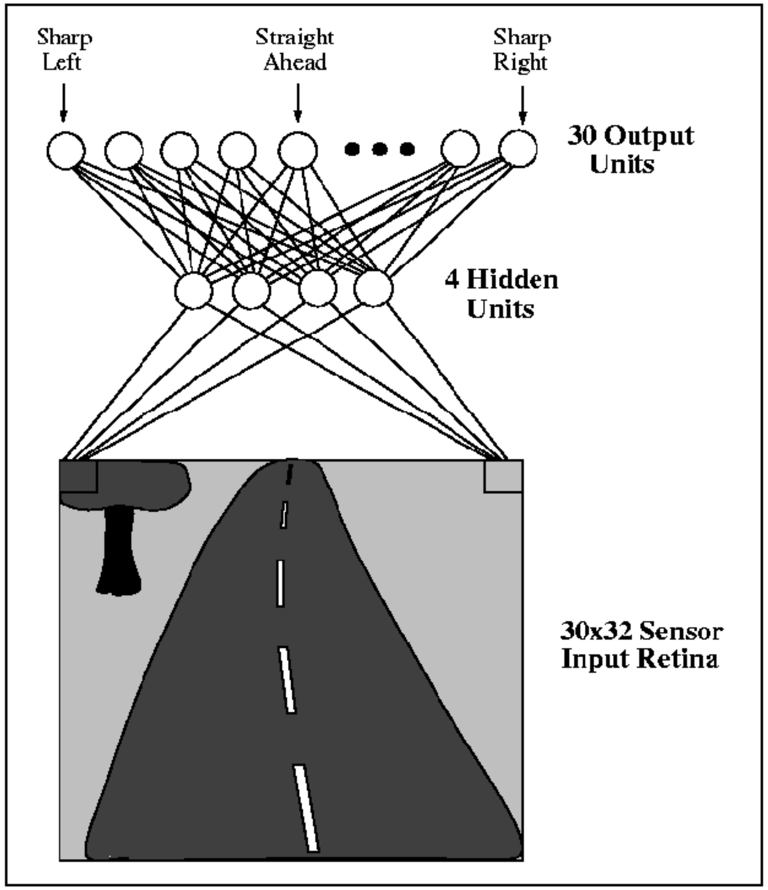
- Some points from definitions
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process
  - Emphasis on tuning weights automatically
  - …

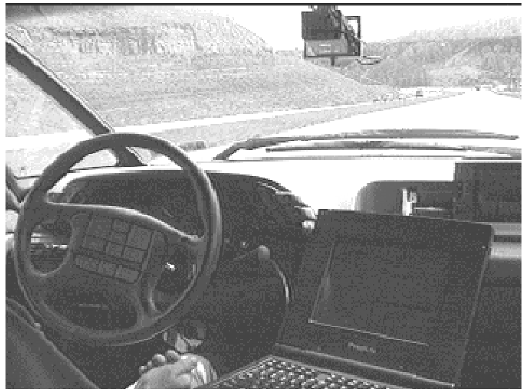
When to Consider Neural Networks

- Input: High-Dimensional and Discrete or Real-Valued
  - e.g., raw sensor input
  - Conversion of symbolic data to quantitative (numerical) representations possible
- Output: Discrete or Real Vector-Valued
  - e.g., low-level control policy for a robot actuator
  - Similar qualitative/quantitative (symbolic/numerical) conversions may apply
- Data: Possibly Noisy
- Target Function: Unknown Form
- Result: Human Readability Less Important Than Performance
  - Performance measured purely in terms of accuracy and efficiency
  - Readability: ability to explain inferences made using model; similar criteria
- Examples
  - Speech phoneme recognition
  - Image classification
  - Time signal prediction, Robotics, and many others

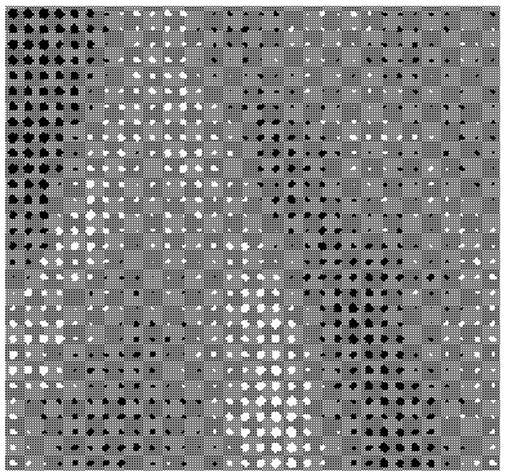
Autonomous Learning Vehicle in a Neural Net (ALVINN)

- Pomerleau *et al*
  - http://www.cs.cmu.edu/afs/cs/project/alv/member/www/projects/ALVINN.html
  - Drives 70mph on highways



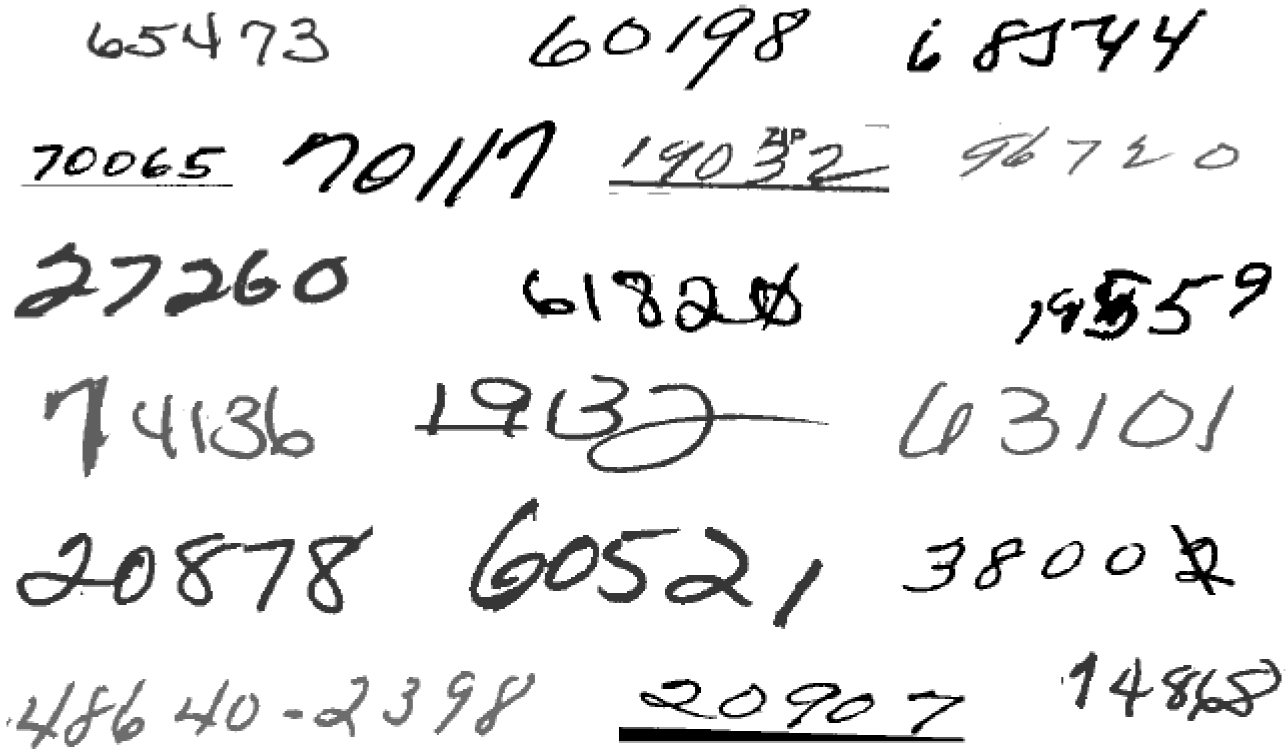


Hidden-to-Output Unit Weight Map (for one hidden unit)



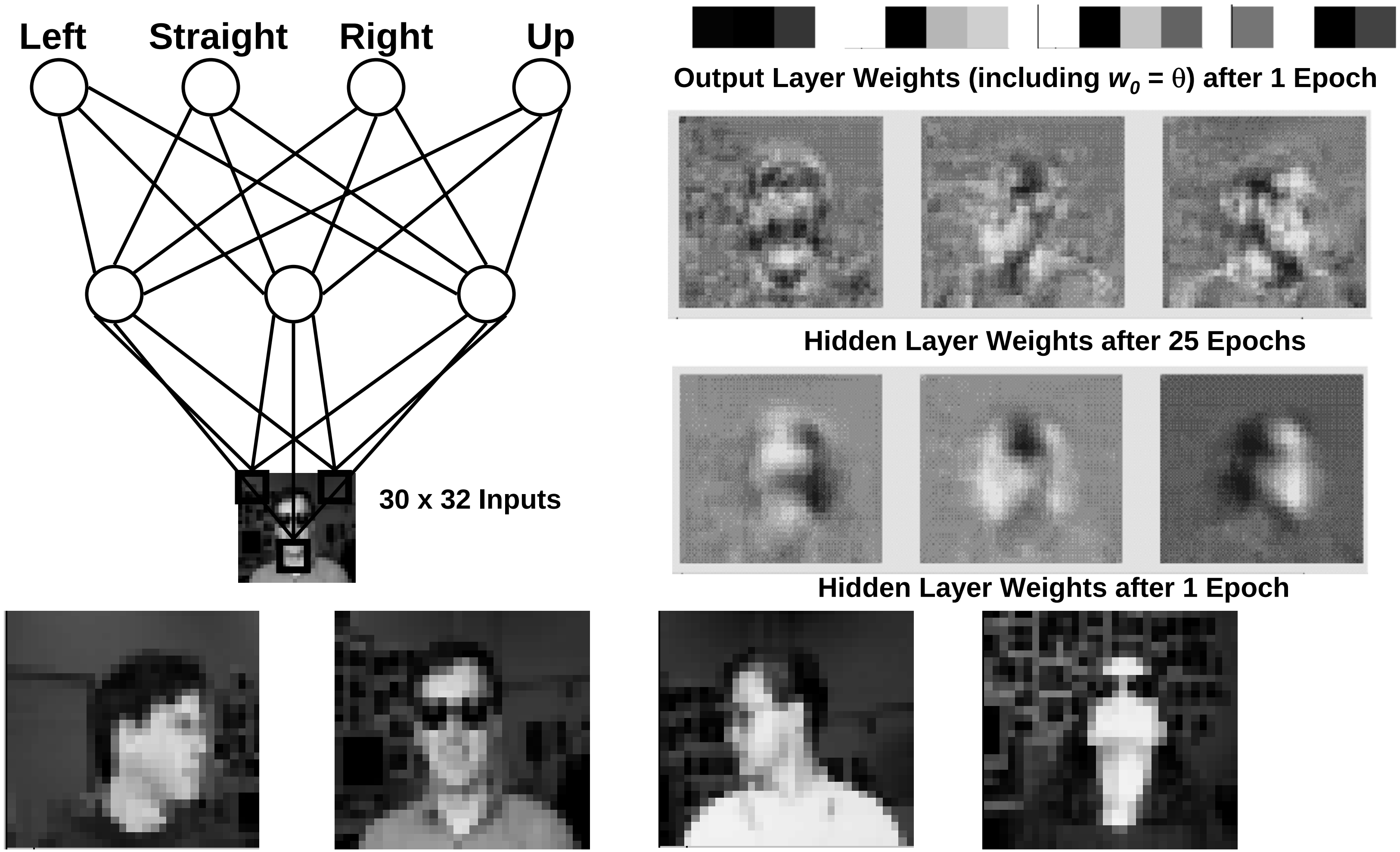
Input-to-Hidden Unit Weight Map (for one hidden unit)

Image Recognition and Classifiation of Postal Codes



Examples of handwritten postal codes
drawn from a database available from the US Postal service

Example:Neural Nets for Face Recognition



- 90% Accurate Learning Head Pose, Recognizing 1-of-20 Faces
- http://www.cs.cmu.edu/~tom/faces.html

Example: *NetTalk*

- Sejnowski and Rosenberg, 1987
- Early Large-Scale Application of Backprop
  - Learning to convert text to speech
    - Acquired model: *a mapping from letters to phonemes and stress marks*
    - Output passed to a speech synthesizer
  - Good performance after training on a vocabulary of ~1000 words
- Very Sophisticated Input-Output Encoding
  - Input: 7-letter window; determines the phoneme for the center letter and context on each side; distributed (i.e., sparse) representation: 200 bits
  - Output: units for articulatory modifiers (e.g., "voiced"), stress, closest phoneme; distributed representation
  - 40 hidden units; 10000 weights total
- Experimental Results
  - Vocabulary: trained on 1024 of 1463 (informal) and 1000 of 20000 (dictionary)
  - 78% on informal, ~60% on dictionary
- http://www.boltz.cs.cmu.edu/benchmarks/nettalk.html

ANN and Mining Data

- ANN originally comes from AI and ML
- Data Mining and Exploration of Data
  - We can meet numerical (at least partly) data, …
  - Tasks of function approximation, pattern classification, etc are also similar
    - ANN are very good approximators or classifiers
  - However, remember about time cost, parameterization, black boxes, …

Examples of Different ANN

- Perceptron
- Multi-Layer Perceptron
- Radial Basis Function (RBF)
- Kohonen Features maps
- Other architectures, e.g.
  - Hopfield networks and BAM
  - ART

Looking at ANN

- ANN could be defined by:
  - Model of artificial network (details of its component and processing)
  - Topology / architecture of the network
  - Learning

Biological Inspirations

- Humans perform complex tasks like vision, motor control, or language understanding very well

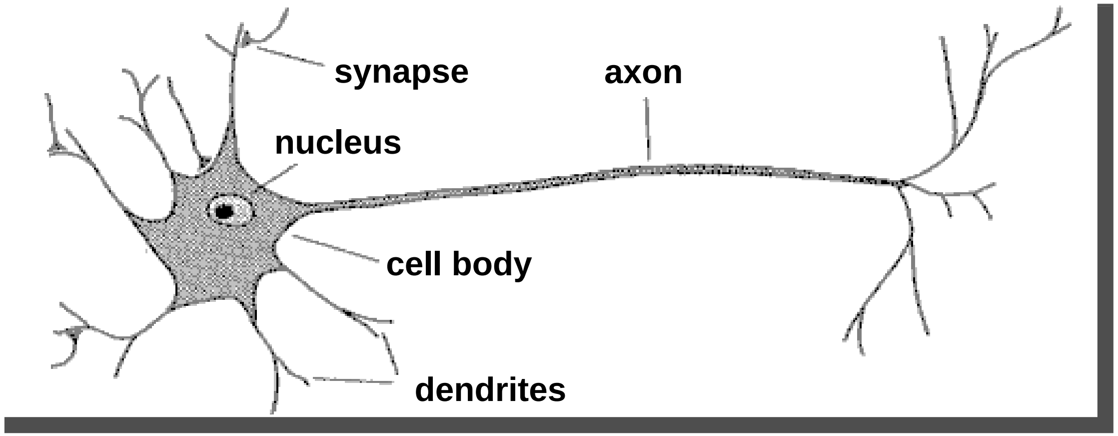
- One way to build intelligent machines is to try to imitate the (organizational principles of) human brain

Biological inspirations

- Some numbers…
  - The human brain contains about (or over) 10 billion nerve cells (neurons)
  - Each neuron is connected to the others through 10000 synapses

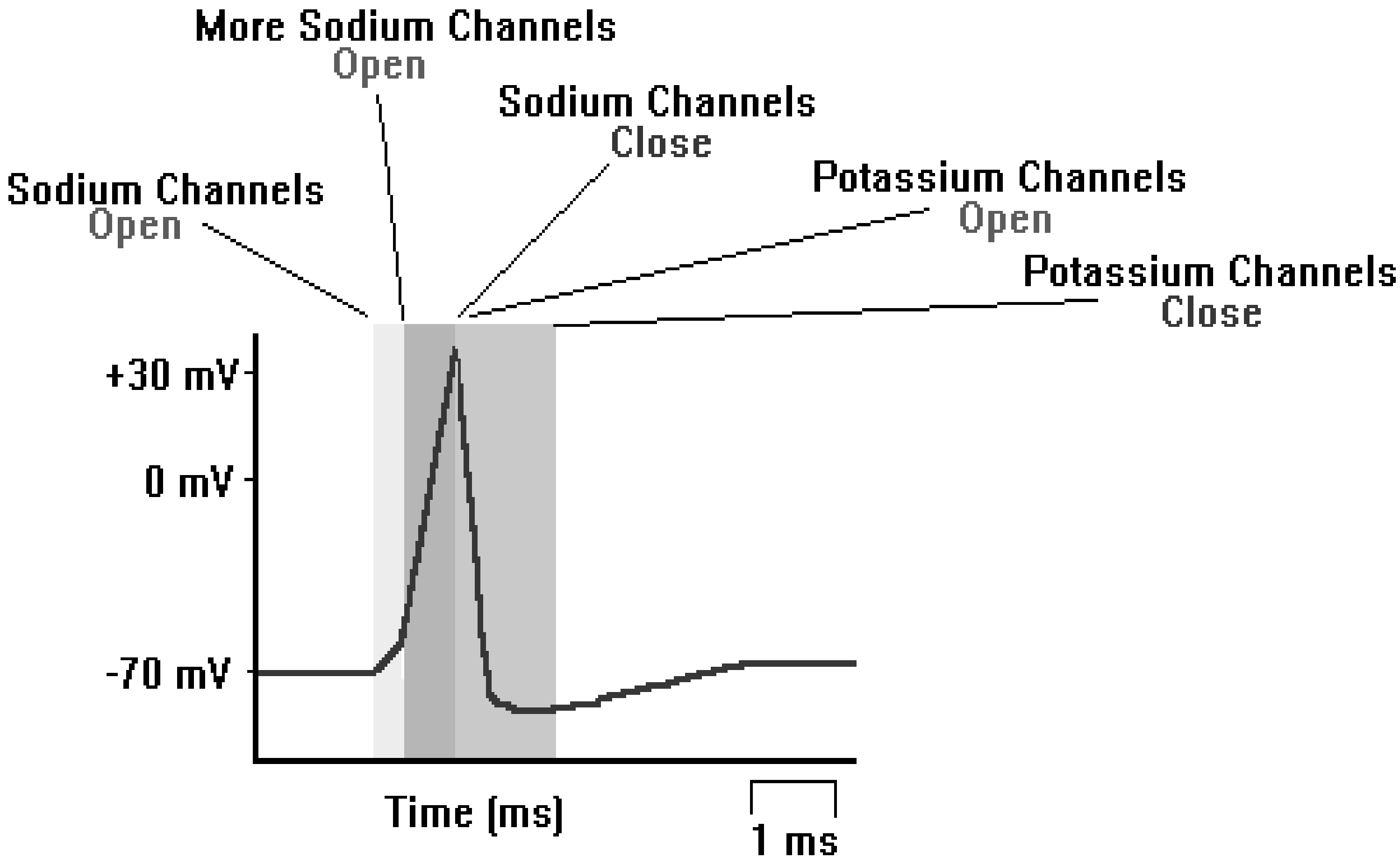
- Properties of the brain
  - It can learn, reorganize itself from experience
  - It adapts to the environment
  - It is robust and fault tolerant

Biological neuron



- A neuron has
  - A branching input (dendrites)
  - A branching output (the axon)
- The information circulates from the dendrites to the axon via the cell body
- Axon connects to dendrites via synapses
  - Synapses vary in strength
  - Synapses may be excitatory or inhibitory

The Action Potential

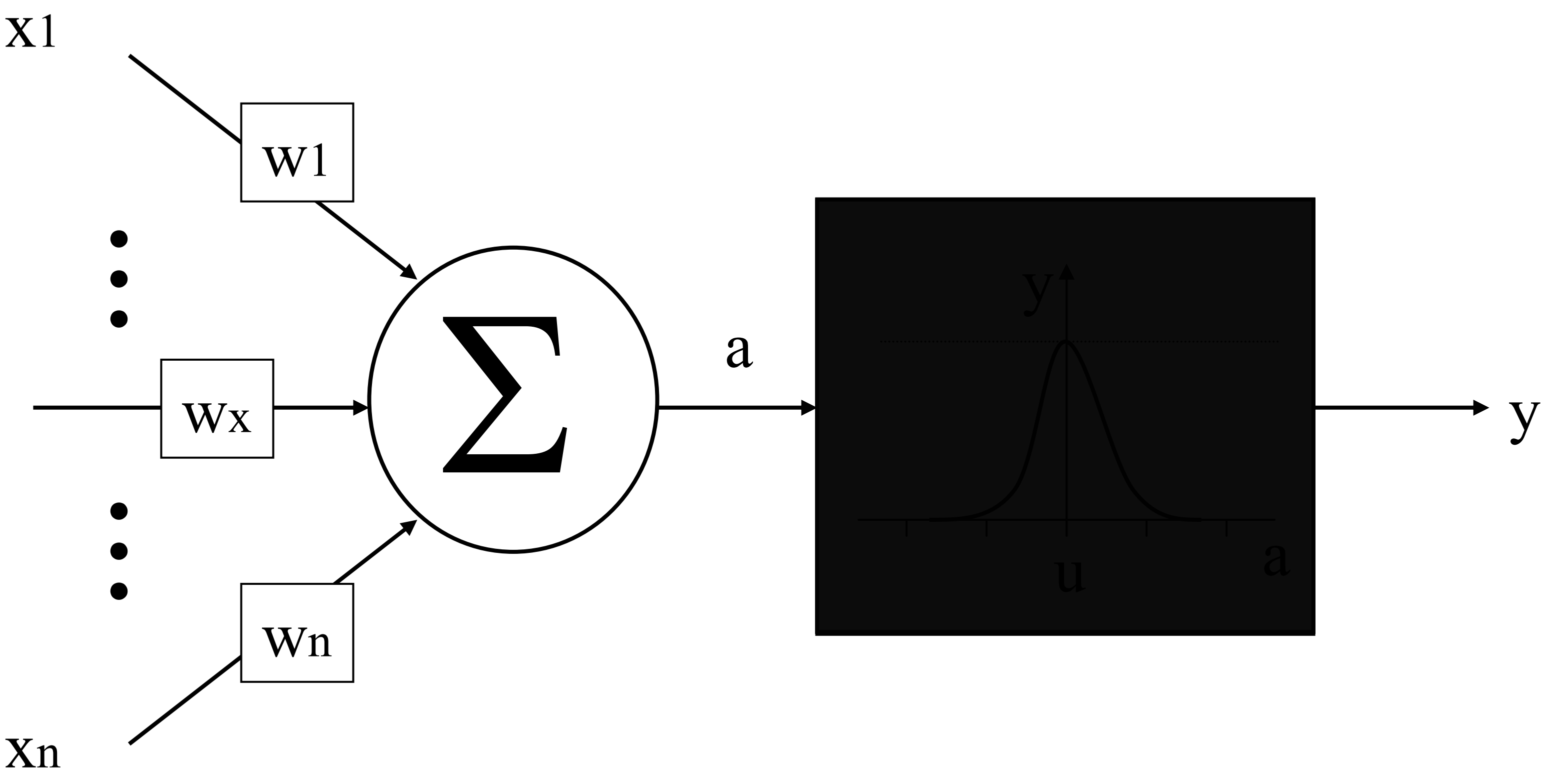


Human Brain

- The brain is a highly complex, non-linear, and parallel computer, composed of some 10^{11} neurons that are densely connected ($\sim 10^4$ connection per neuron). *We have just begun to understand how the brain works...*
- A neuron is much slower (10^{-3}sec) compared to a silicon logic gate (10^{-9}sec), however the massive interconnection between neurons make up for the comparably slow rate.
  – Complex perceptual decisions are arrived at quickly (within a few hundred milliseconds)
- 100-Steps rule: Since individual neurons operate in a few milliseconds, calculations do not involve more than about 100 serial steps and the information sent from one neuron to another is very small (a few bits)
- Plasticity: Some of the neural structure of the brain is present at birth, while other parts are developed through learning, especially in early stages of life, to adapt to the environment (new inputs).

The Artificial Neuron

(Mc Culloch and Pitt, 1943)



$$y(t+1) = f\left(\sum_{k=1}^{n} w_k\ x_k(t) - u\right) = f\left(\sum_{k=0}^{n} w_k\ x_k(t)\right)$$

Activation Functions

- Step function

$$f(a) = \begin{cases} +1 & if \quad a \geq u \\ -1 & if \quad a < u \end{cases}$$

- Linear function

$$f(a) = \begin{cases} +1 & if \quad a \geq u \\ a & if \quad -u \leq a < u \\ -1 & if \quad a < -u \end{cases}$$

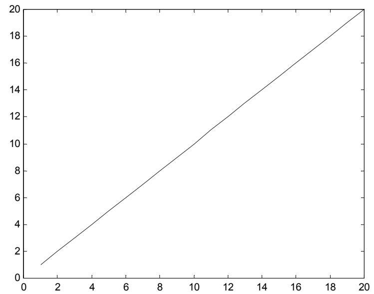
- Logistic Sigmoid

$$f(a) = \frac{1}{1 + e^{-ha}}$$

- Gaussian

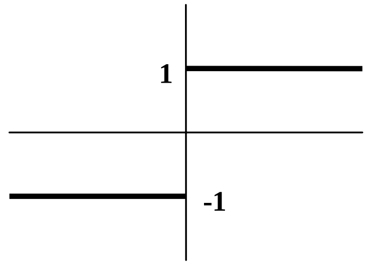
$$f(a) = e^{-\frac{|a-u|}{2\sigma^2}}$$

Activation functions

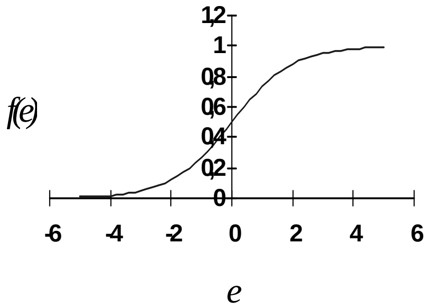


Linear

$$y = x$$

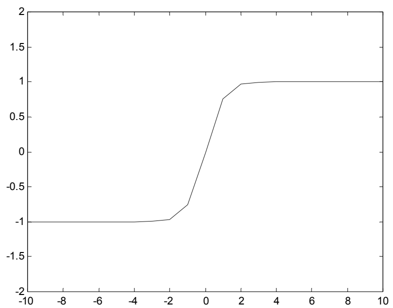


Step function



Sigmoidal (logistic)

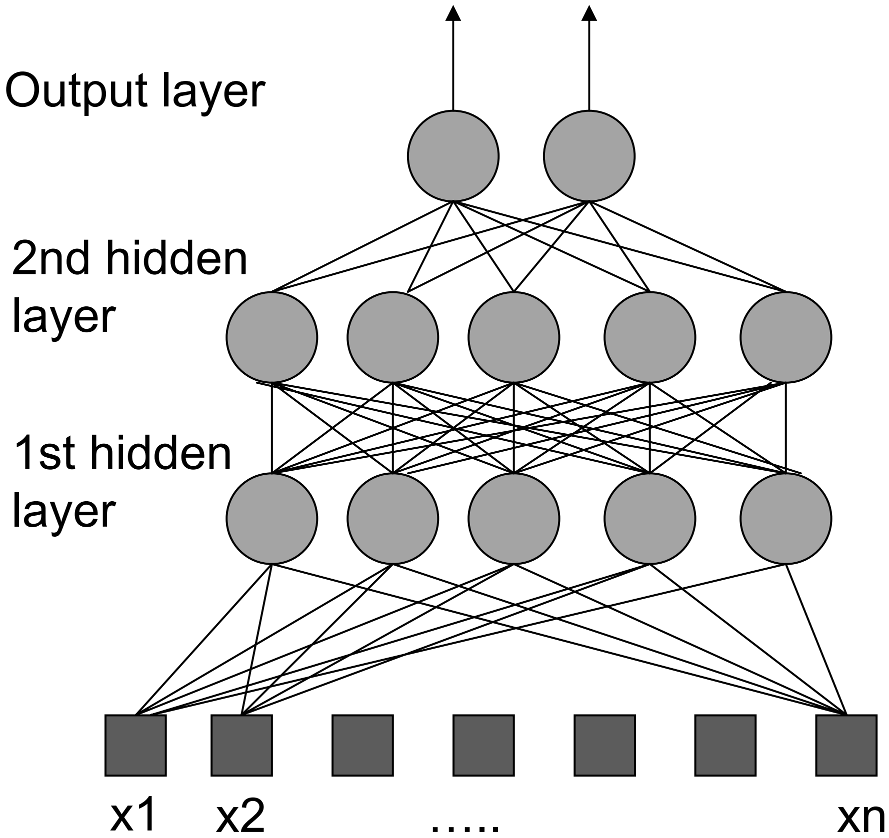
$$y = \frac{1}{1 + \exp(-\beta x)}$$



Hyperbolic tangent

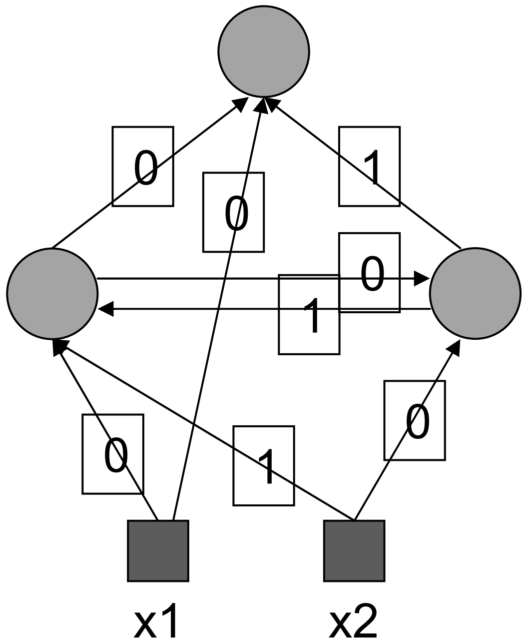
$$y = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

Network topologies
Feed Forward Neural Networks



- The information is propagated from the inputs to the outputs
- Computations of No non linear functions from n input variables by compositions of Nc algebraic functions
- Time has no role (NO cycle between outputs and inputs)

Network topologies
Recurrent Neural Networks



- Can have arbitrary topologies
- Can model systems with internal states (dynamic ones)
- Delays are associated to a specific weight
- Training is more difficult
- Performance may be problematic
  – Stable Outputs may be more difficult to evaluate
  – Unexpected behavior (oscillation, chaos, …)

Learning neural networks

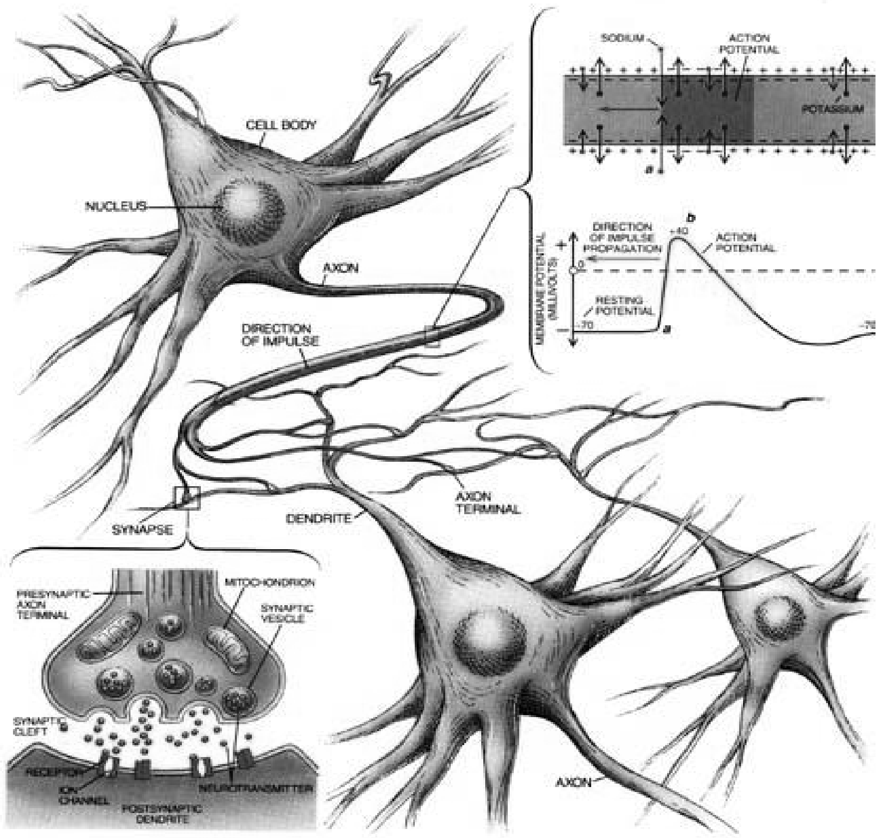
- The procedure that consists in estimating the parameters of neurons (**usually weights**) so that the whole network can perform a specific task

- Basic types of learning
  - The supervised learning
  - The unsupervised learning

- The Learning process (supervised)
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs

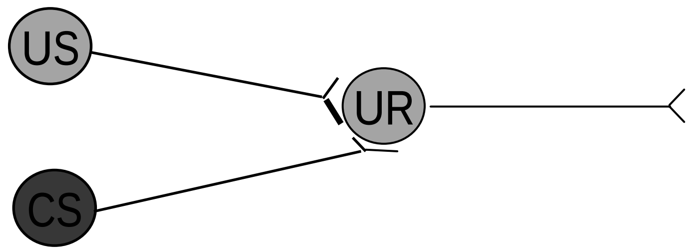
The ANN Learning Process

- Neurons can learn, (Hebb, 1949):
  - memory is stored in synapses and learning takes place by synaptic modifications;
  - neurons become organized into larger configurations to perform more complex information processing



Hebbian learning:

- When two joining cells fire simultaneously, the connection between them strengthens (Hebb, 1949)
- Discovered at a biomolecular level by Lomo (1966) (Long-term potentiation).

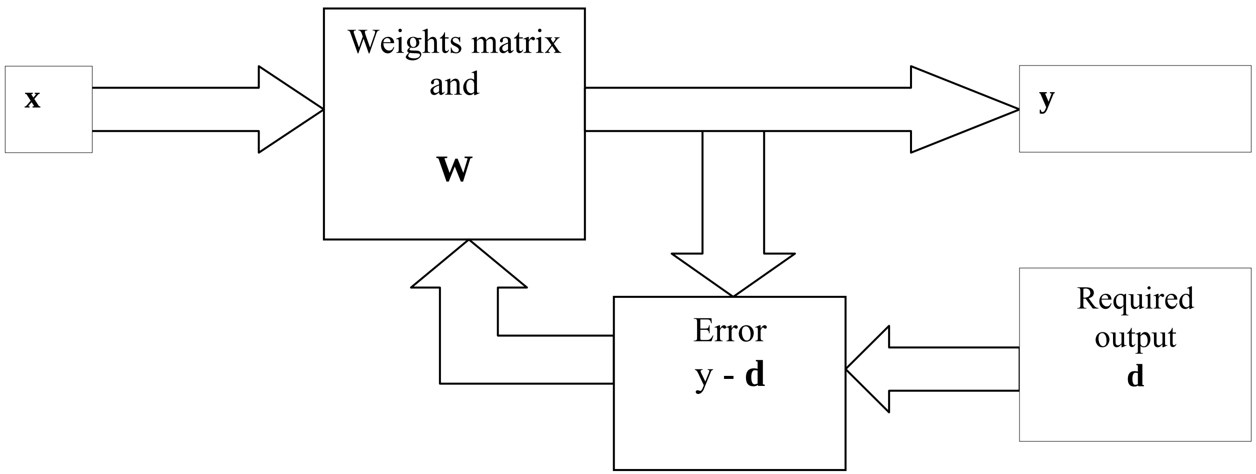


Supervised Learning of Neurons

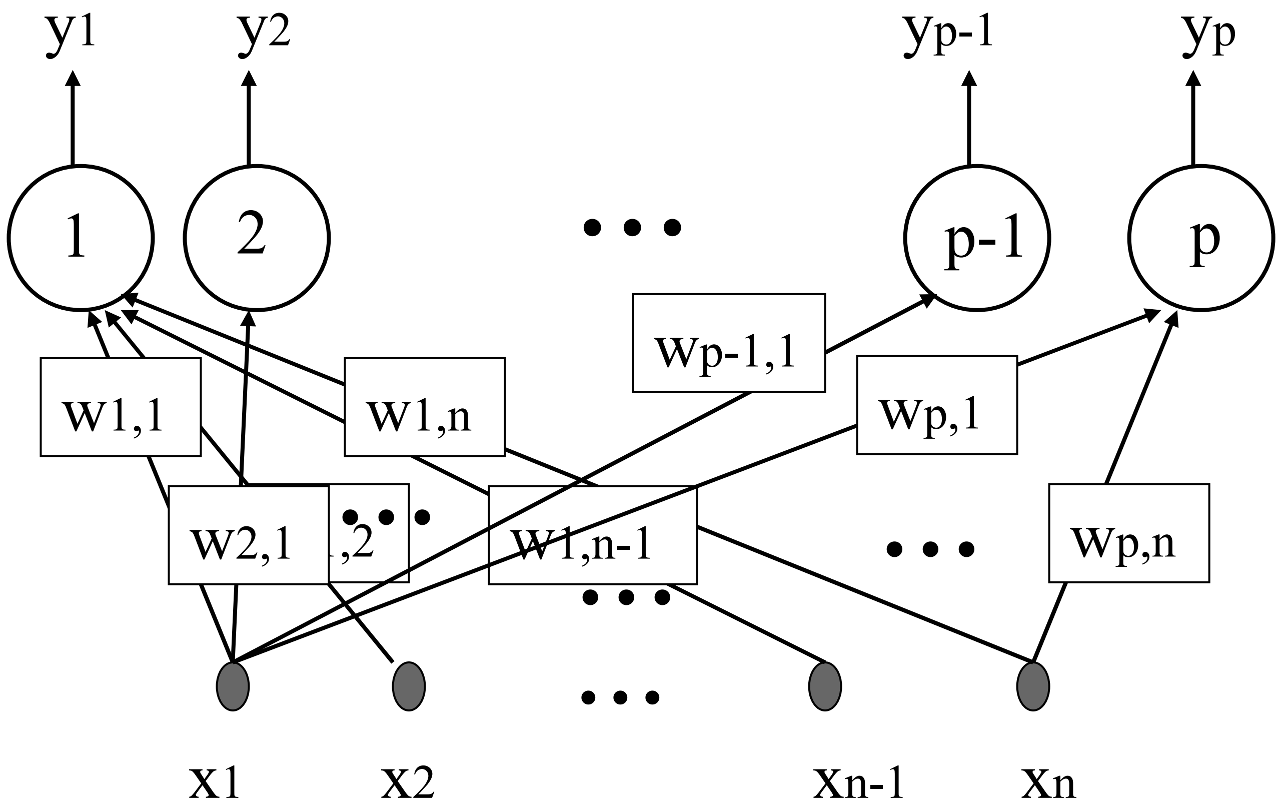
Let us suppose that a sufficiently large set of examples (training set) is available.

Supervised learning:

- The network answer to each input pattern is directly compared with the desired answer and a feedback is given to the network to correct possible errors



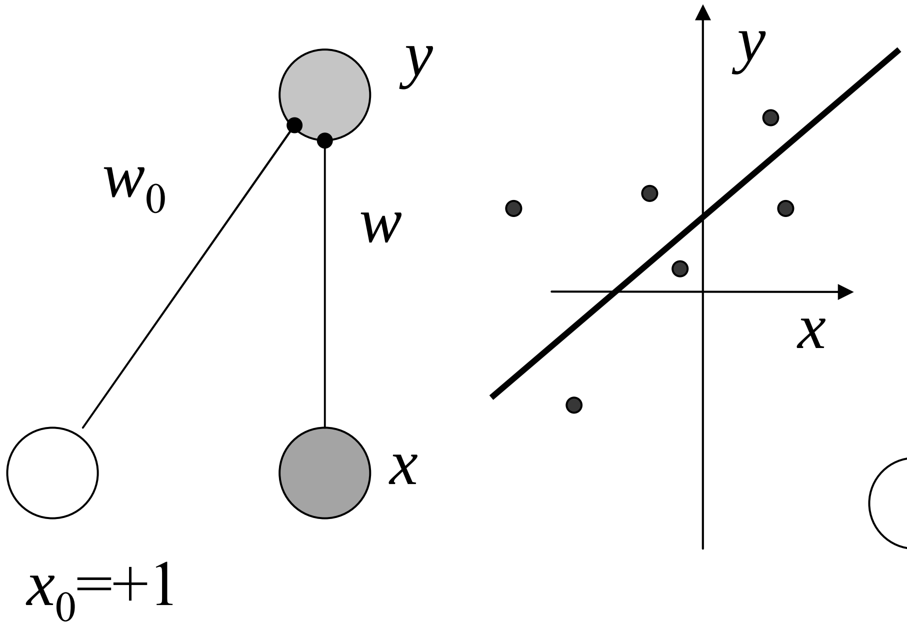
Perceptron



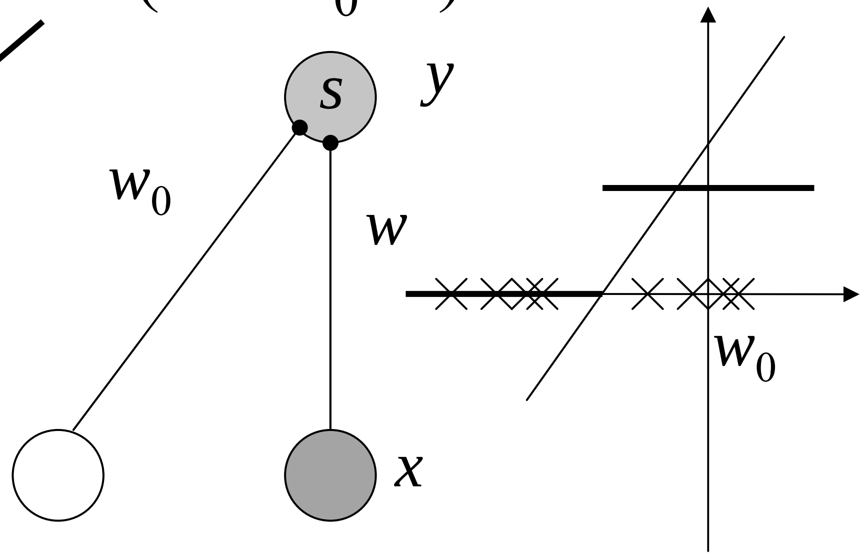
$$y_i(t+1) = f\left(\sum_{k=0}^{n} w_{ik}\ x_k(t)\right) \qquad i = 1, 2, \ldots p$$

What a Single Perceptron Does

- Regression: $y=wx+w_0$



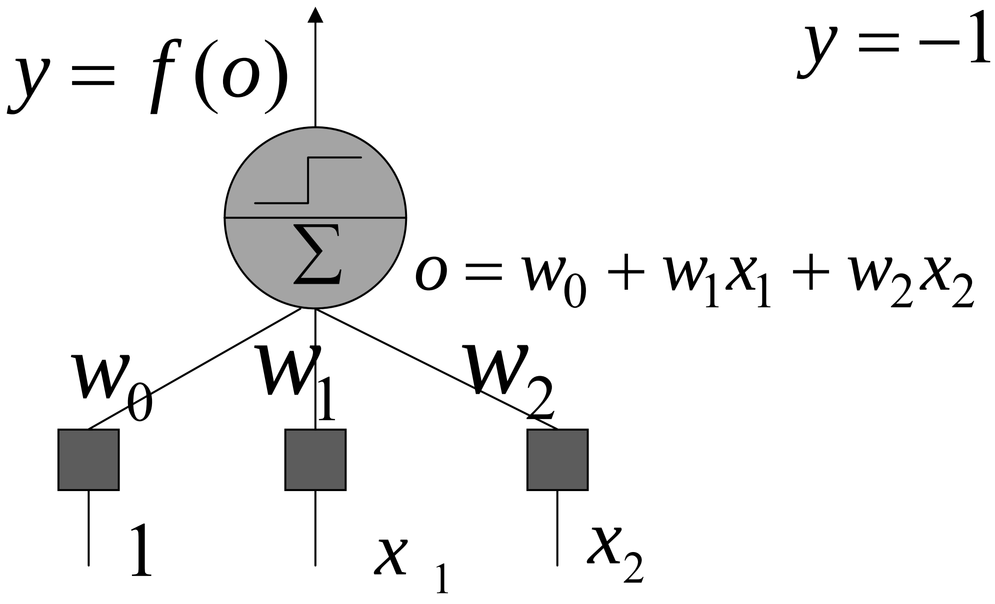
- Classification: $y=1$ if $(wx+w_0>0)$

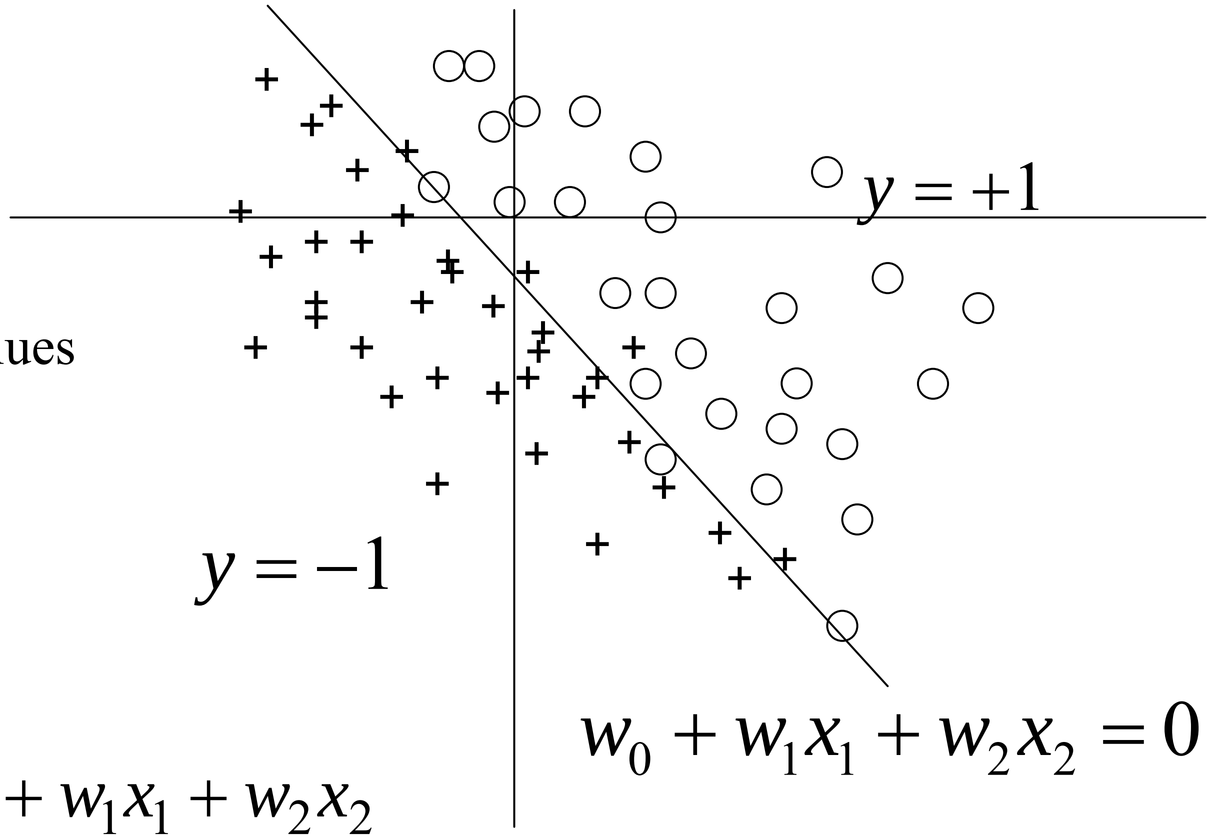


$$y = \text{sigmoid}\,(o) = \frac{1}{1+\exp\left[-\boldsymbol{w}^T\boldsymbol{x}\right]}$$

Perceptron

- Rosenblatt (1962)
- Linear separation
- Inputs :Vector of real values
- Outputs :1 or -1

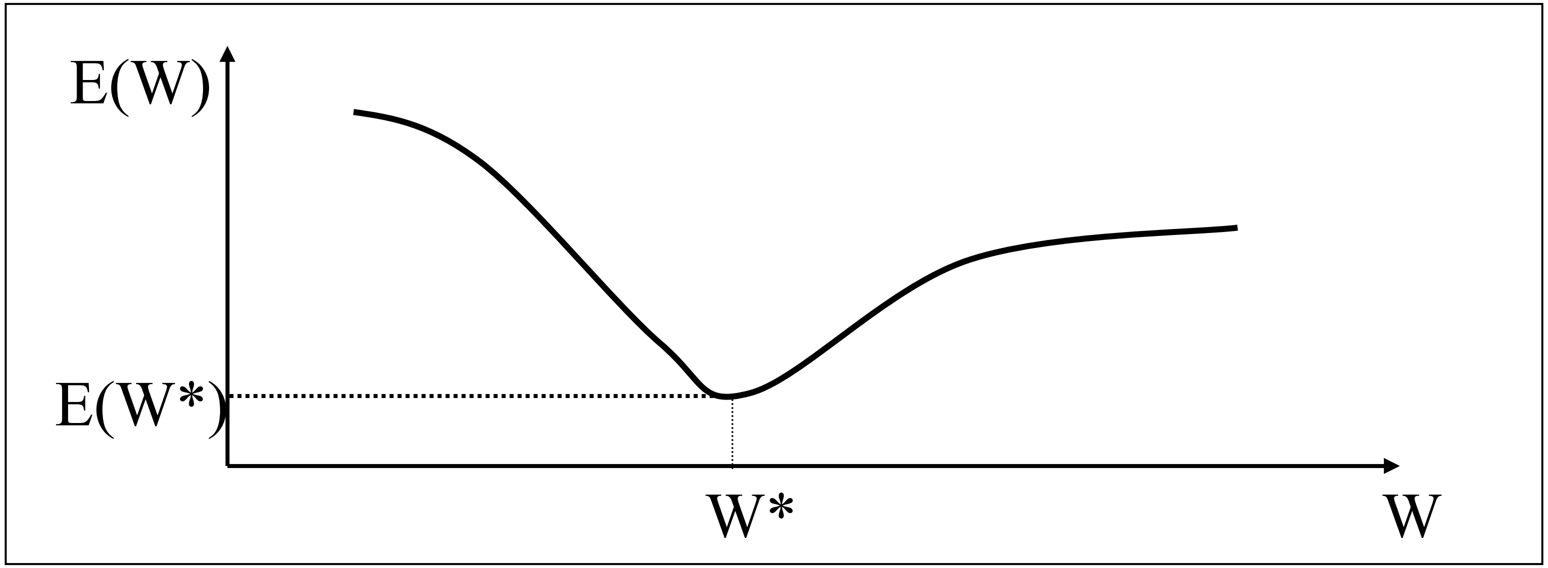




Error Function

- Training set: $T = \left\{ \left(x^q, d^q \right) \quad q = 1, 2, ..., m \right\}$
- Error Measure:

$$E\left(W\right) = f\left(o_i^q - d_i^q\right)$$



Gradient Descent algorithm

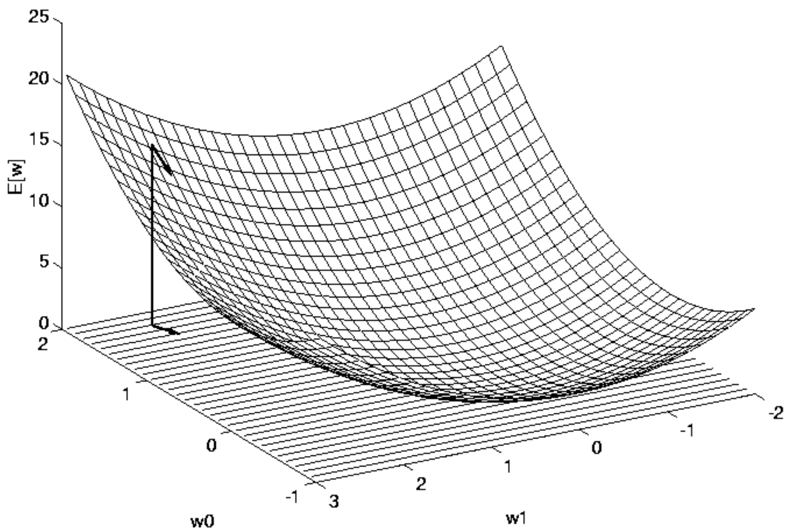
- Simple Gradient Descent Algorithm
  - Applicable to different type of learning (with proper representation)
- Algorithm *Train-Perceptron* ($D \equiv \{<x, o(x) \equiv d(x)>\}$)
  - Initialize all weights w_i to random values
  - WHILE not all examples correctly predicted DO

    FOR each training example $x \in D$

    Compute current output $o(x)$

    FOR i = 1 to n

    $w_i \leftarrow w_i + r(t - o)x_i$ **//delta perceptron learning rule**



- Definition: Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

Gradient Descent algorithm

The RMS error function:

$$E(W) = \frac{1}{2} \sum_{q=1}^{m} \sum_{i=1}^{p} \left(o_i^q - d_i^q \right)^2 = \sum_{q=1}^{m} E^q(W)$$

The learning process (stepwise looking for solution):

$$w_{ik}(t+1) = w_{ik}(t) + \Delta w_{ik}(t)$$

The gradient descent algorithm:

$$\Delta w_{ik}(t) = -\eta \frac{\partial E(t)}{\partial w_{ik}} = -\eta \sum_{q=1}^{m} \frac{\partial E^q(t)}{\partial w_{ik}} = \sum_{q=1}^{m} \Delta w_{ik}^q(t)$$

Delta Learning Rule (Widrow,Hoff)

$$\frac{\partial E^q}{\partial w_{ik}} = \frac{\partial\left(\frac{1}{2}\sum_{i=1}^{p}\left(o_i^q - d_i^q\right)^2\right)}{\partial w_{ik}} =$$

= After some computations --

$$\boxed{\Delta w_{ik}^q = -\eta\ \delta_i^q\ x_k^q}$$

In literature: Error is usually calculated as (d – o), and delta learning rule will be given in a form:

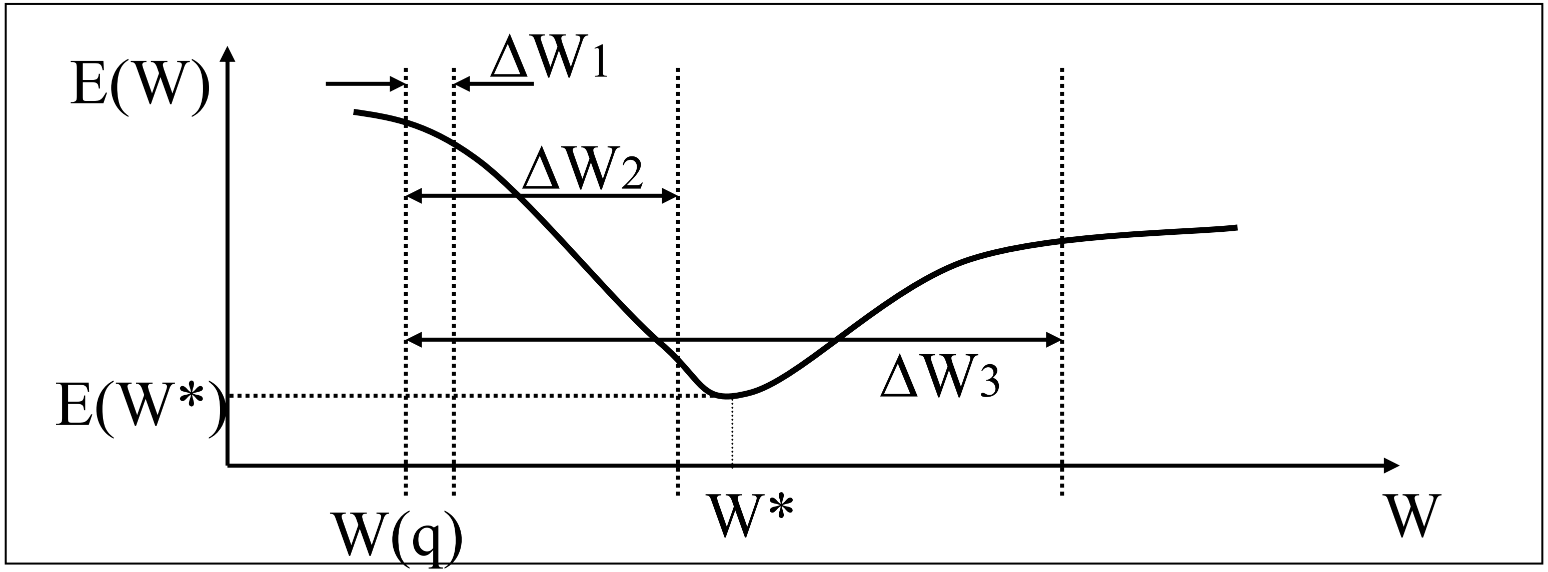
$$\boxed{\Delta w_{ik}^q = \eta\ \delta_i^q\ x_k^q}$$

Learning Rate (η)

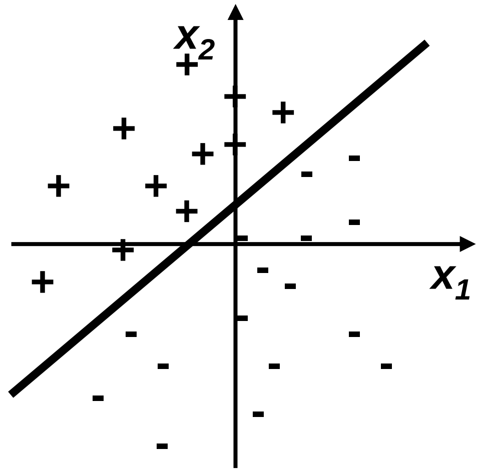
$\Delta w_1 = \eta_1 \ \delta \ x$ with η_1 too small

$\Delta w_2 = \eta_2 \ \delta \ x$ with η_2 right size

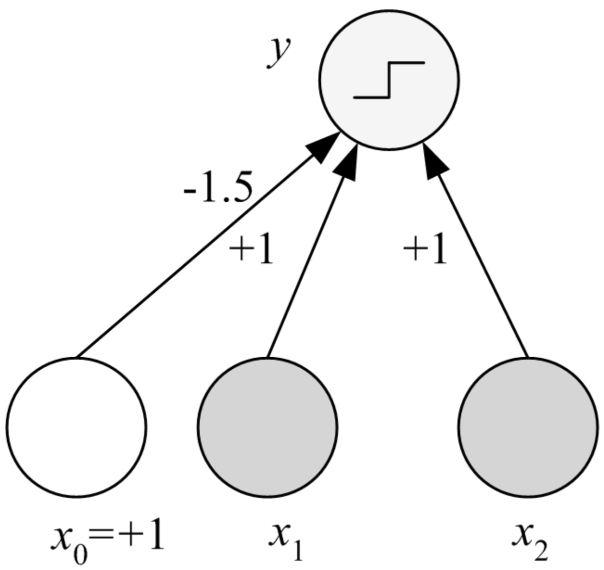
$\Delta w_3 = \eta_3 \ \delta \ x$ with η_3 too big

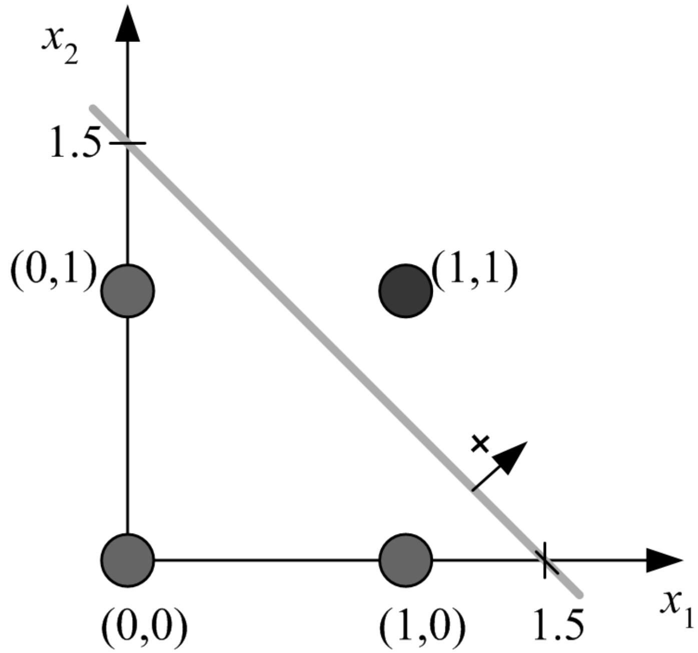


- The standard perceptron learning algorithm converges if examples are linearly separable → see
- Consider an example of a simple logical AND problem



Linearly Separable (LS) Data Set

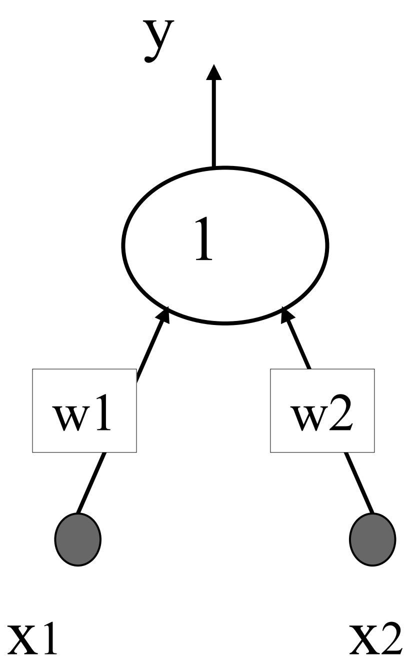




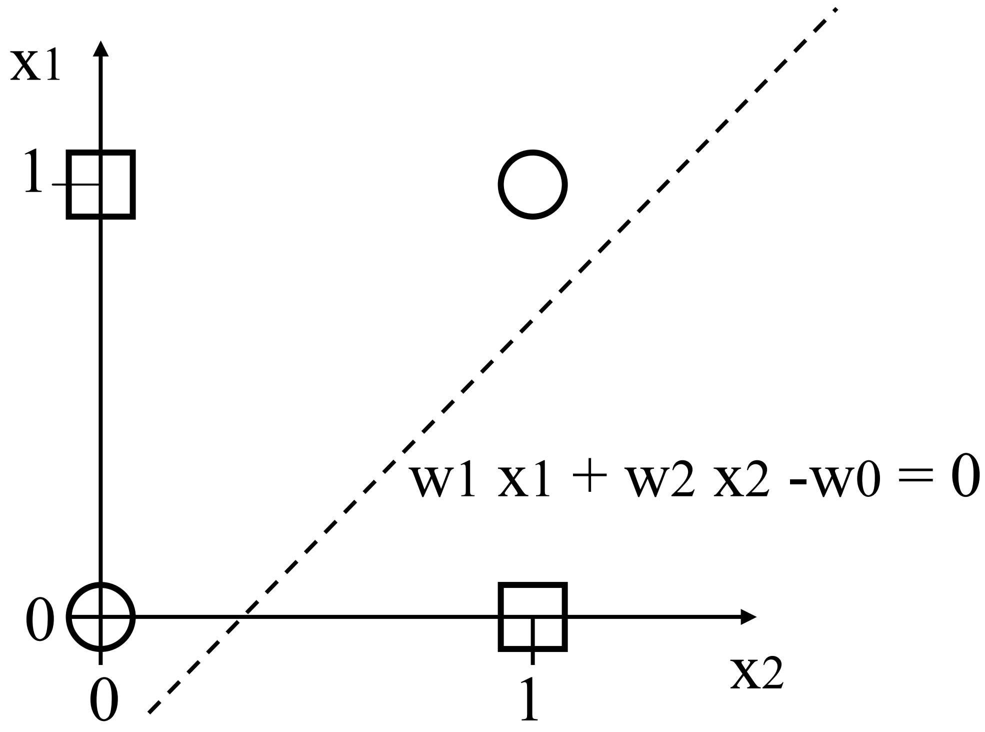
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron limitations [Minski,Papert]

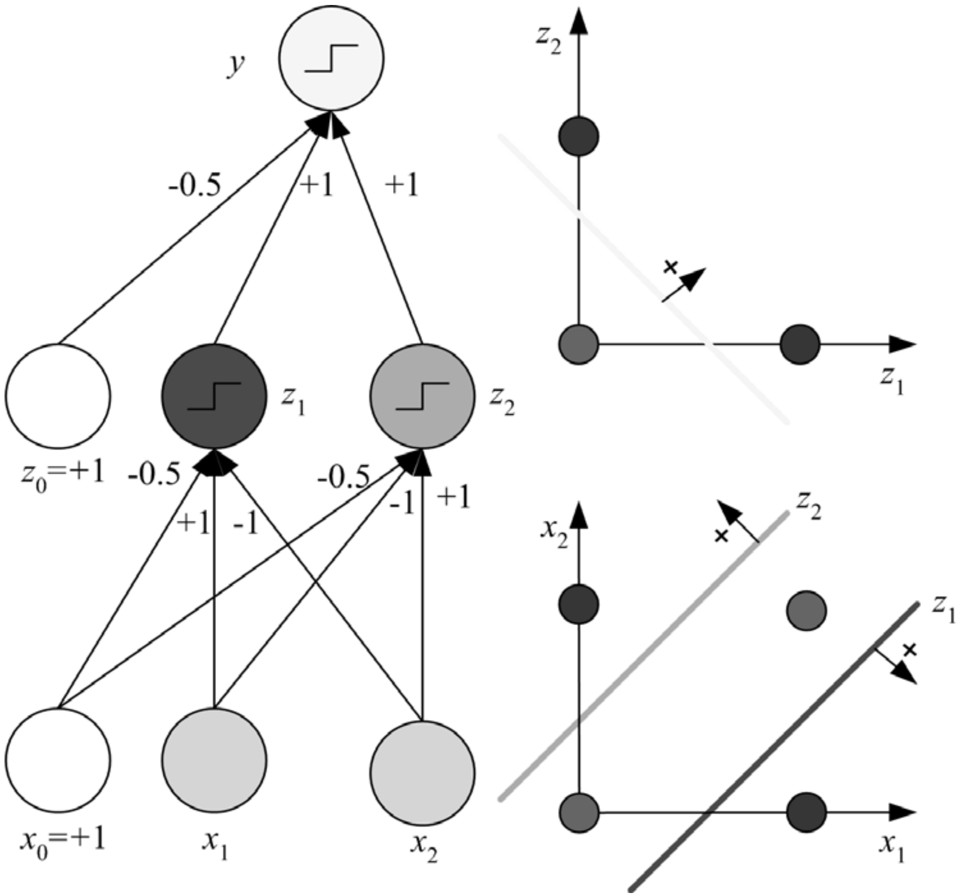
The XOR function: the non-linear separability problem



x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



Need for constructing MLP



$$o_i(t) = \frac{1}{1+e^{-(\mathrm{net}_i(t)-\theta)/\tau}}$$

The solution – 2 layered network with non-linear Functions

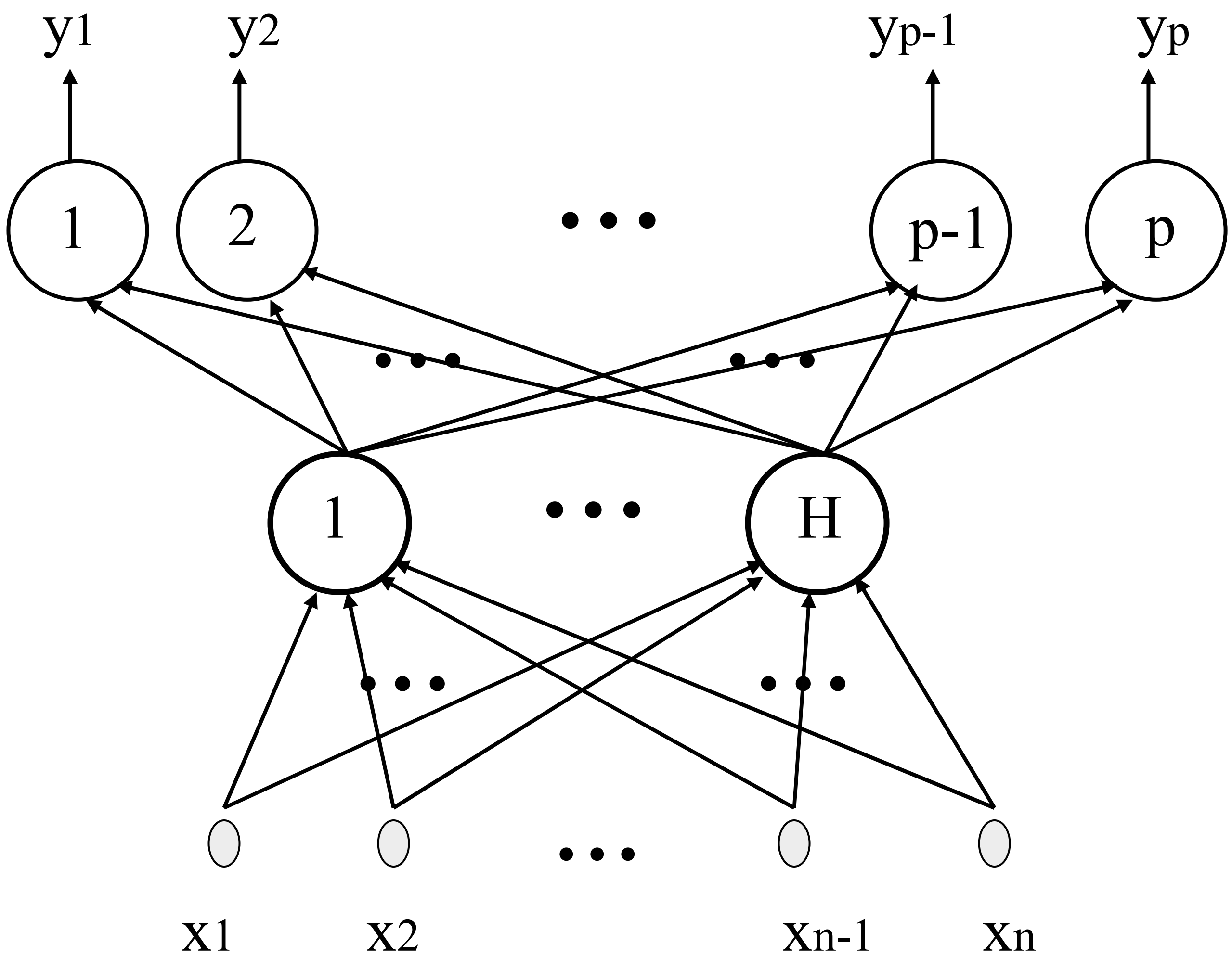
However → how to learn weights in such networks

x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

The Universality Property

- A two layer feed-forward neural network with step activation functions can implement **any** Boolean function, provided that the number of hidden neurons H is sufficiently large (Mc Culloch and Pitts, 1943) .
- If the input variables are continuous in [0,1] and the activation function is the logistic sigmoid, it can be proven that **any** continuous decision boundary can be approximated arbitrarily close by a two-layer Perceptron with a sufficient number H of hidden neurons (Cybenko, 1989) .

MultiLayer Perceptrons



Non-linear regression mapping

Output of a generic MLP neuron in layer l

$$y_i = f_i(a_i) = f_i\left(\sum_{k=0}^{n(l-1)} w_{ik}\, o_k\right) \quad i = 1,\ldots, n(l) \;\; k = 1,\ldots, n(l-1)$$

Two-layer MLP, only one output unit with linear activation function.

$$y_1 = b_1 = \sum_{k=0}^{n(1)} w_{1k}\, o_k = \sum_{k=0}^{n(1)} w_{1k}\; f_k\left(\sum_{j=0}^{n(0)} v_{kj}\; x_j\right) =$$

$$= \sum_{k=1}^{n(1)} w_{1k}\; f_k\left(\vec{v}_k^{\,T}\; \vec{x} + v_{k0}\right) + w_0$$

Back propagation (I)

(The Generalized Delta Rule)

Gradient Descent formula for a weight w_{ik} connecting units from two generic layers l and l-1 ($i \in$ layer l, $k \in$ layer l-1) after presentation of training pattern q.

$$\Delta w_{ik}^{q} = -\eta \frac{\partial E^{q}}{\partial w_{ik}}$$

Now calculations should take into account activation function.

$$\frac{\partial E^{q}}{\partial w_{ik}} = \frac{\partial E^{q}}{\partial a_{i}^{q}} \frac{\partial a_{i}^{q}}{\partial w_{ik}}$$

Back Propagation (II)

$$\frac{\partial a_i^q}{\partial w_{ik}} = o_k^q$$

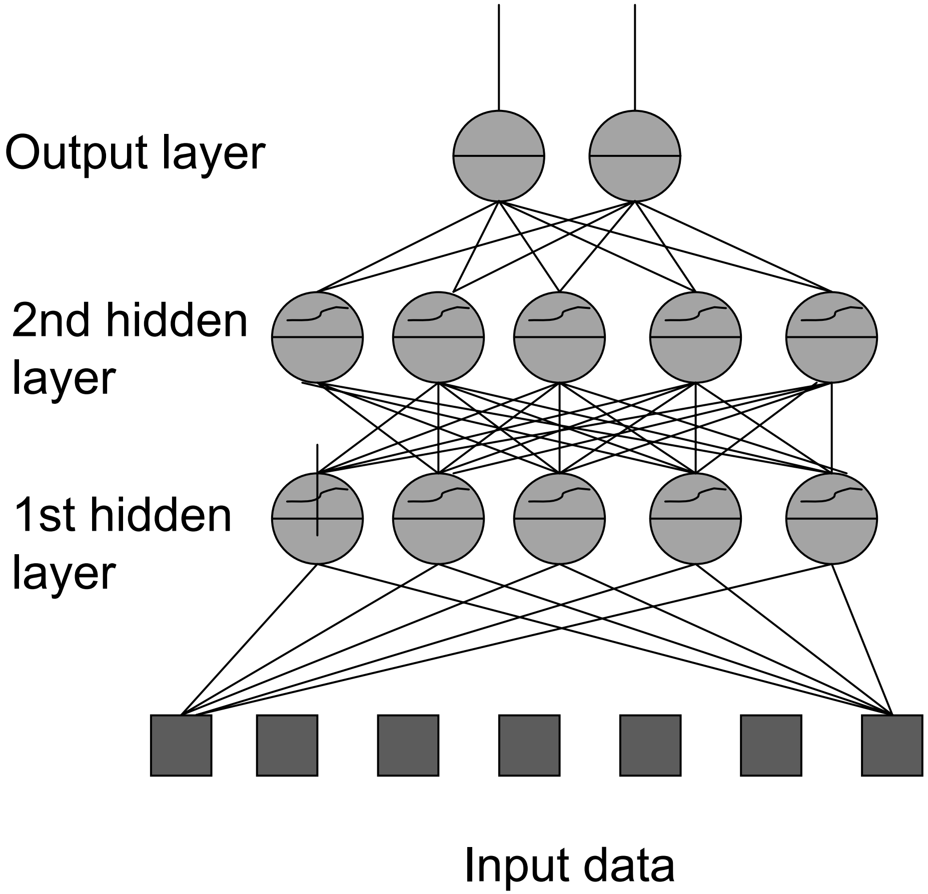
$$\delta_i^q = \frac{\partial E^q}{\partial a_i^q}$$

$$\Delta w_{ik}^q = -\eta \frac{\partial E^q}{\partial w_{ik}} = -\eta\, \delta_i^q\, o_k^q$$

For output units (i∈layer L) – generalized delta learning rule:

$$\delta_i^q = \frac{\partial E^q}{\partial a_i^q} = f'\left(a_i^q\right)\left(o_i^q - d_i^q\right)$$

Multi-Layer Perceptron



- One or more hidden layers
- Where can we use generalized delta rule?
- Where can we compute error?

We do not know the desired answers of the hidden layer and therefore we can not estimate the error function.

Back Propagation (III)

We do not know the desired answers of the hidden layer and therefore we can not estimate the error function.

For hidden units ($i \in$ layer $l < L$):

$$\delta_i^q = \frac{\partial E^q}{\partial a_i^q} = \sum_{j=1}^{n(l+1)} \frac{\partial E^q}{\partial a_j^q} \frac{\partial a_j^q}{\partial b_i^q} =$$

$$= \sum_{j=1}^{n(l+1)} \delta_j^q \frac{\partial a_j^q}{\partial a_i^q} =$$

$$= f'(a_i^q) \sum_{j=1}^{n(l+1)} \delta_j^q \, w_{ji}$$

- Create a ANN with n_{in} input, n_{out} output, and n_{hidden} hidden units.
- Initialize all network weights to small random numbers(e.g. $[-0.5, 0.5]$).
- Until the termination conditions is met, Do:
  - For each $(\vec{x}, \vec{t})$ in the training examples, Do:
    1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u inthe network.
    2. For each network output unit k, calculate its error terms σ_k

$$\sigma_k = o_k(1 - o_k)(t_k - o_k) \tag{10}$$

    3. For each hidden unit k, calculate its error term σ_h

$$\sigma_h = o_h(1 - o_h) \sum_{k \in outputs\, of\, h} (w_{kh} \cdot \sigma_k) \tag{11}$$

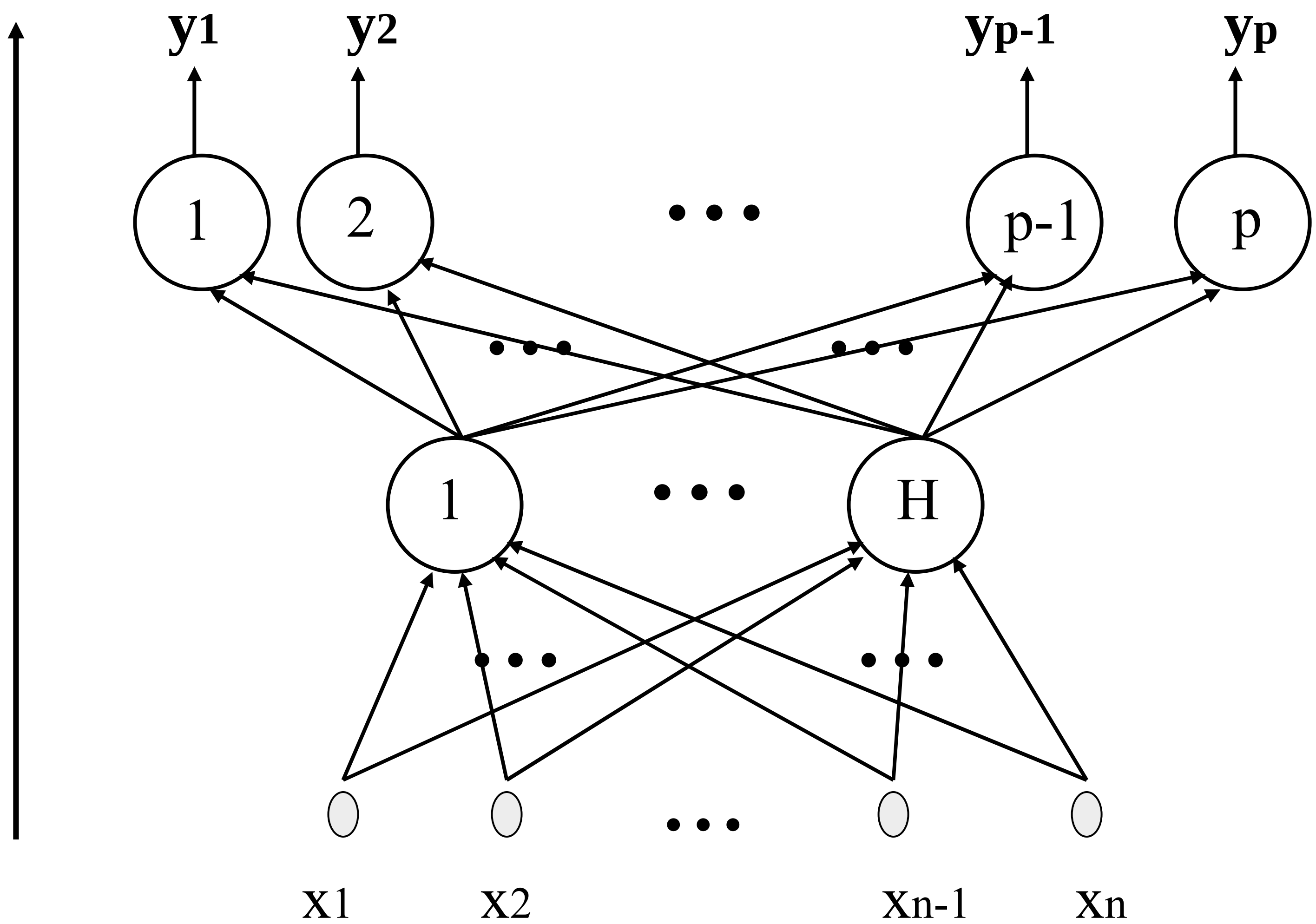
    4. update each network weight w_{ji}

$$w_{ji} = w_{ji} + \triangle w_{ji}$$

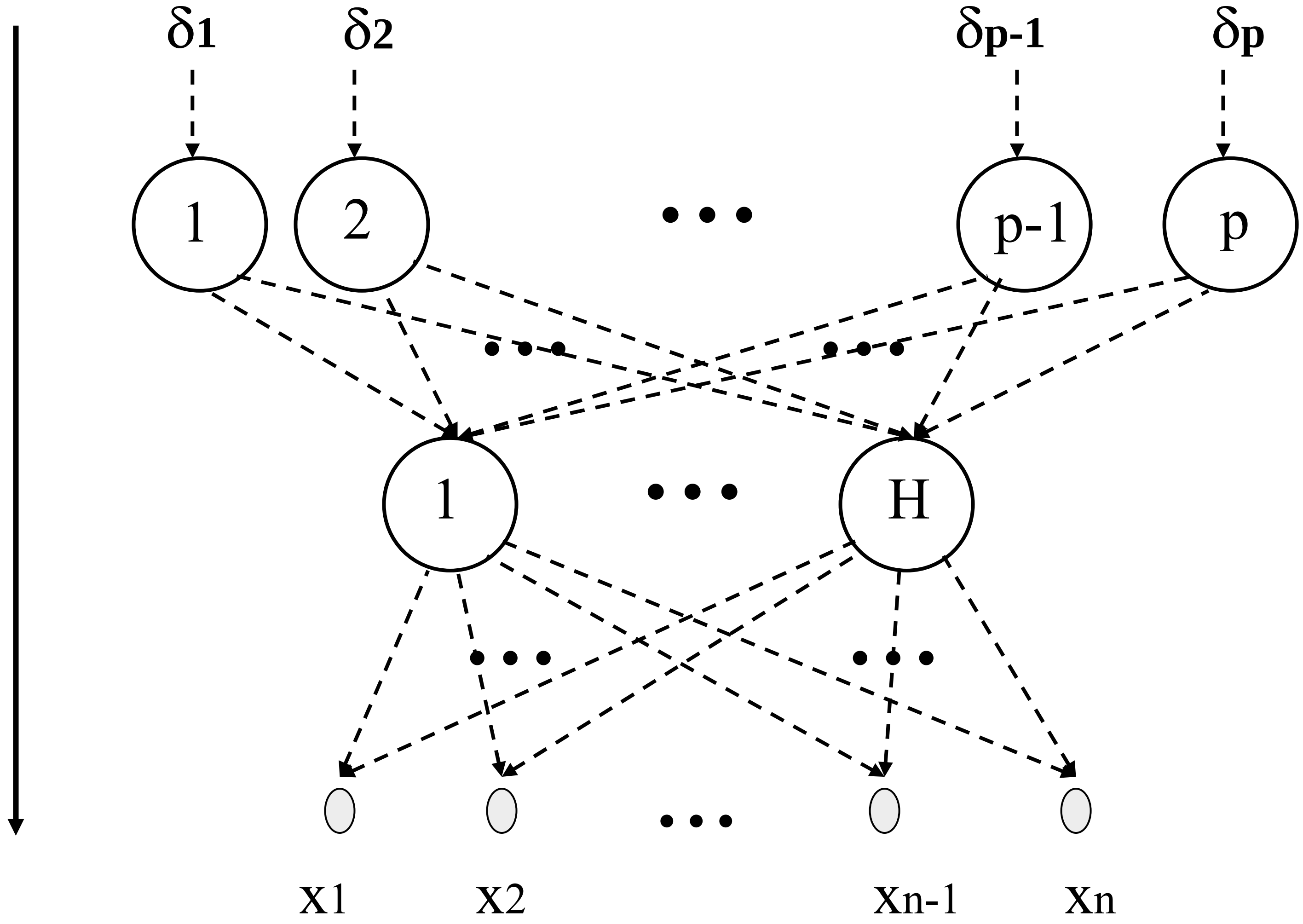
where

$$\triangle w_{ji} = \eta \sigma_j x_j i$$

Back Propagation
(forward phase)

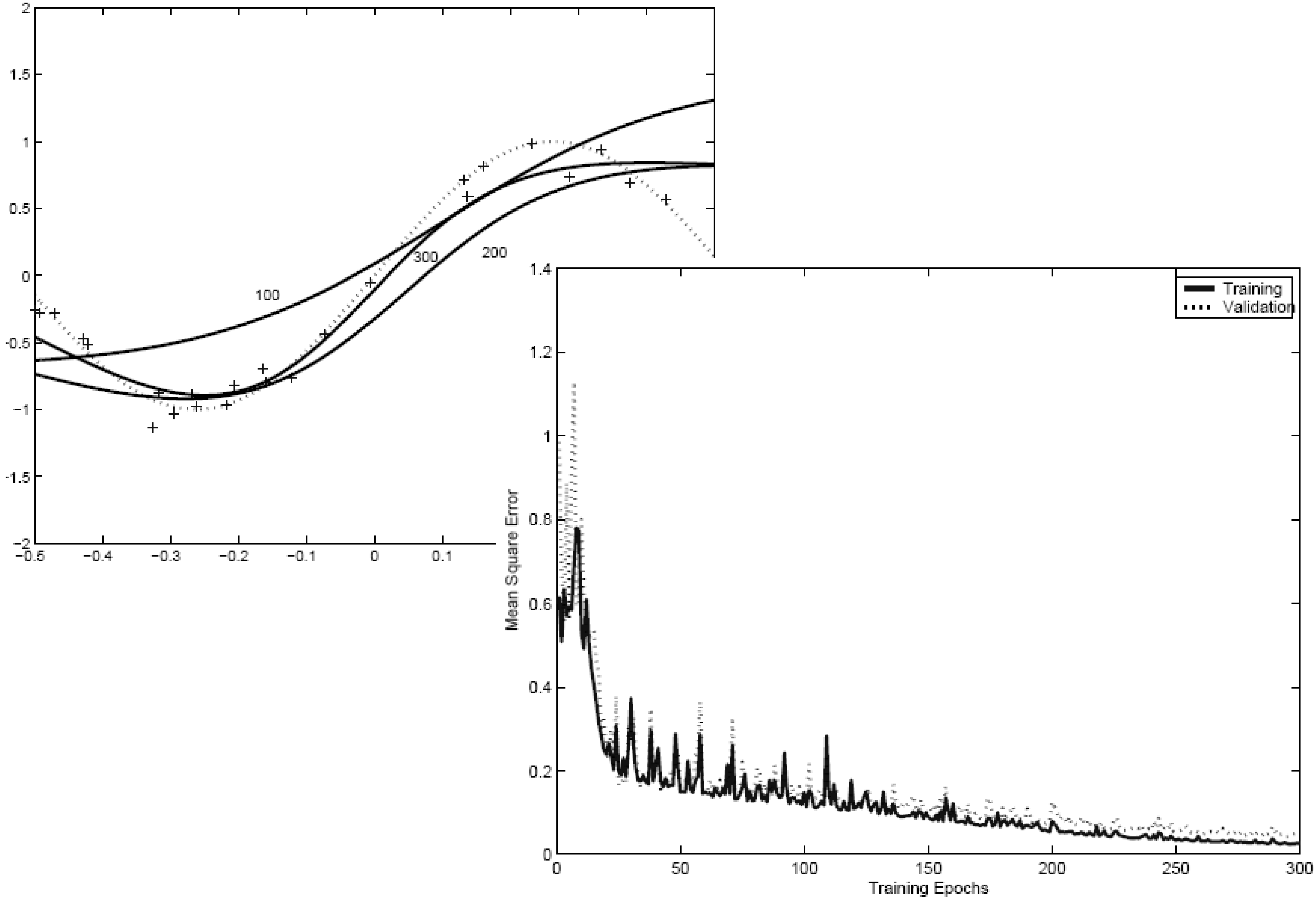


Back Propagation
(backward phase)



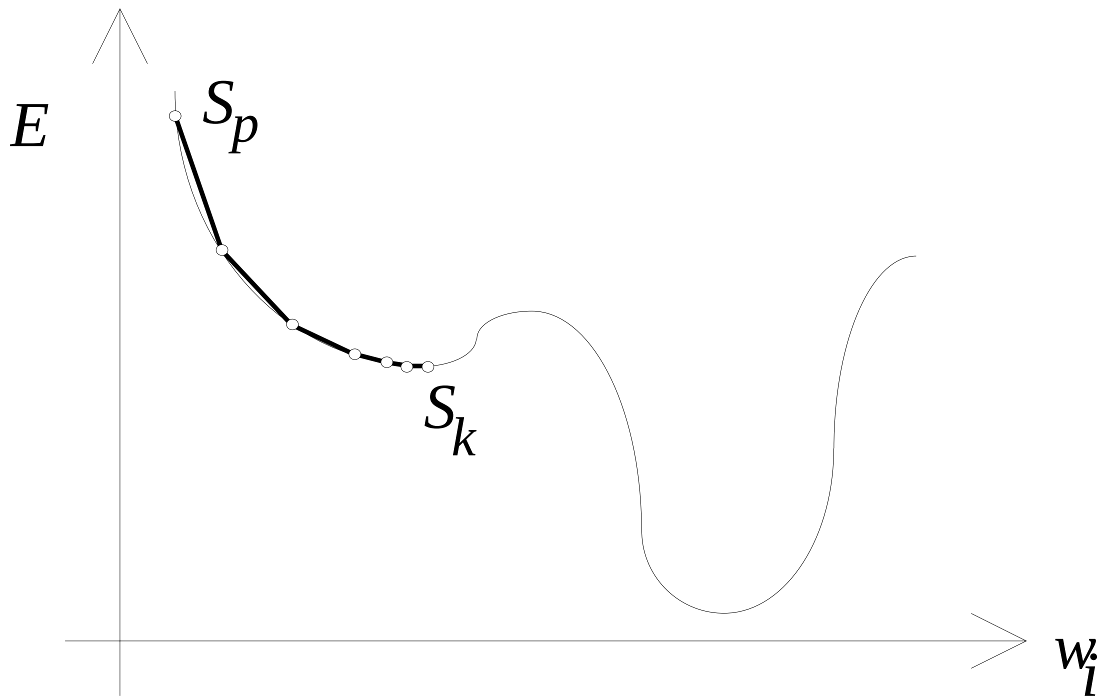
Elements of Backpropagation

- The set of learning examples is usually showed to the algorithms several times (iterations → epochs) / sometimes thousands
- The order of showing examples is randomly shuffled
- Stopping conditions
  - Threshold for RMS (should be smaller than …)
  - Max no. of iterations
  - Classification evaluations

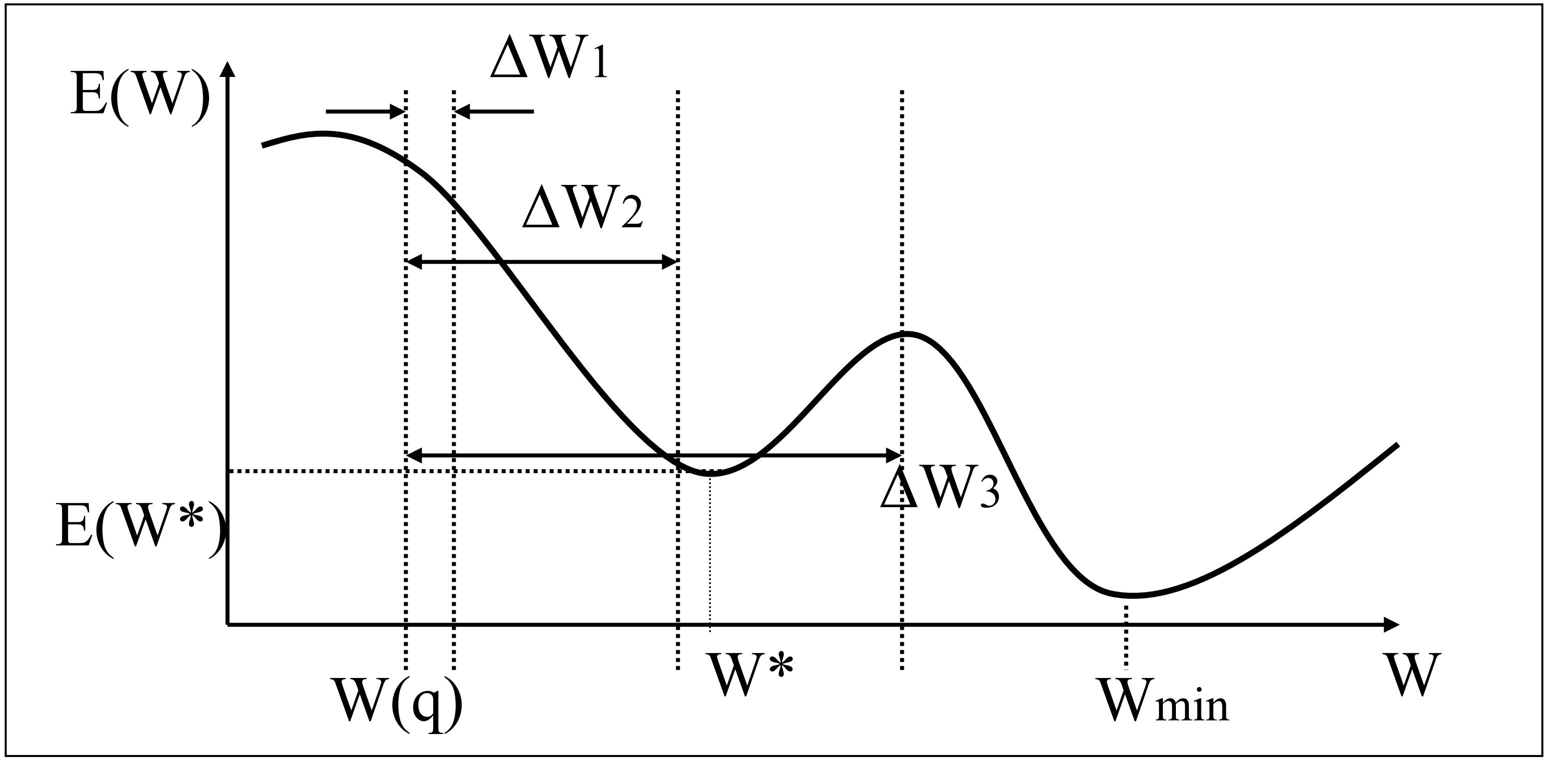

100
200
300
Training
Validation
Mean Square Error
Training Epochs

Tuning learning rate

- Too small – local minimum of error
- Too large – oscillations and unable to go inside the global minimum
- Some solutions
  - Slowly decreasing the rate with epoches (time)

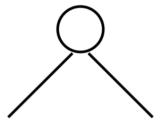
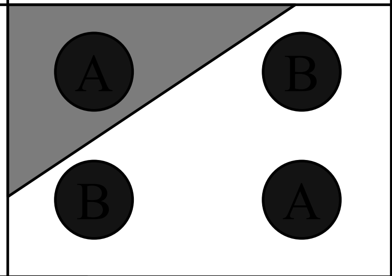
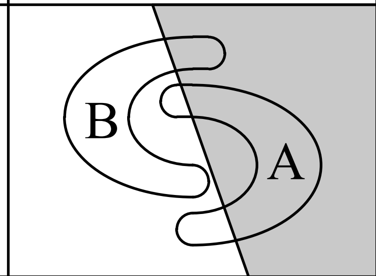
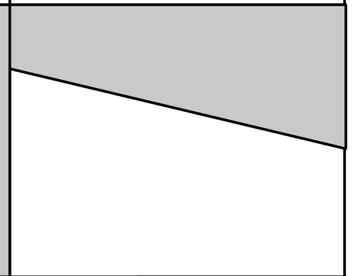
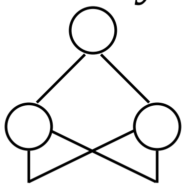
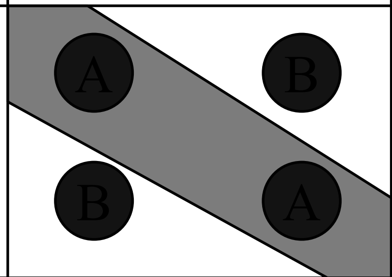
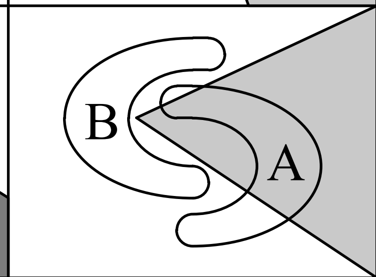
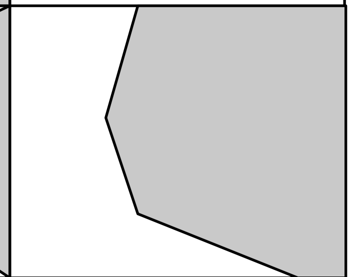
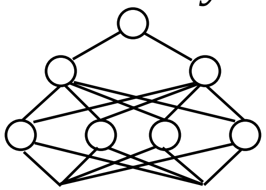
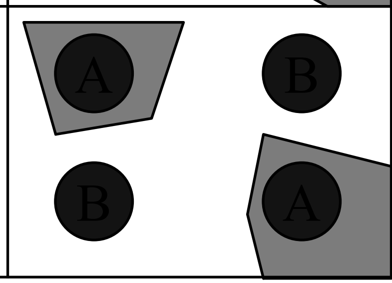
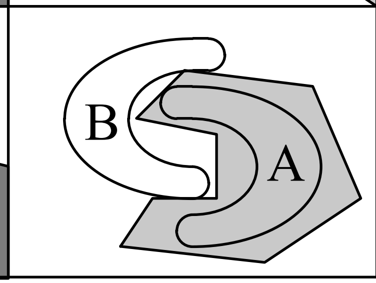
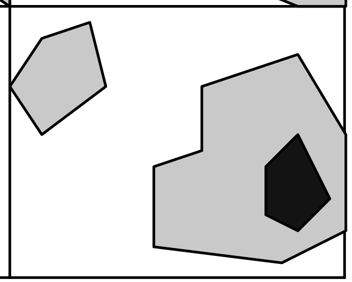


Learning Rate and Momentum Term



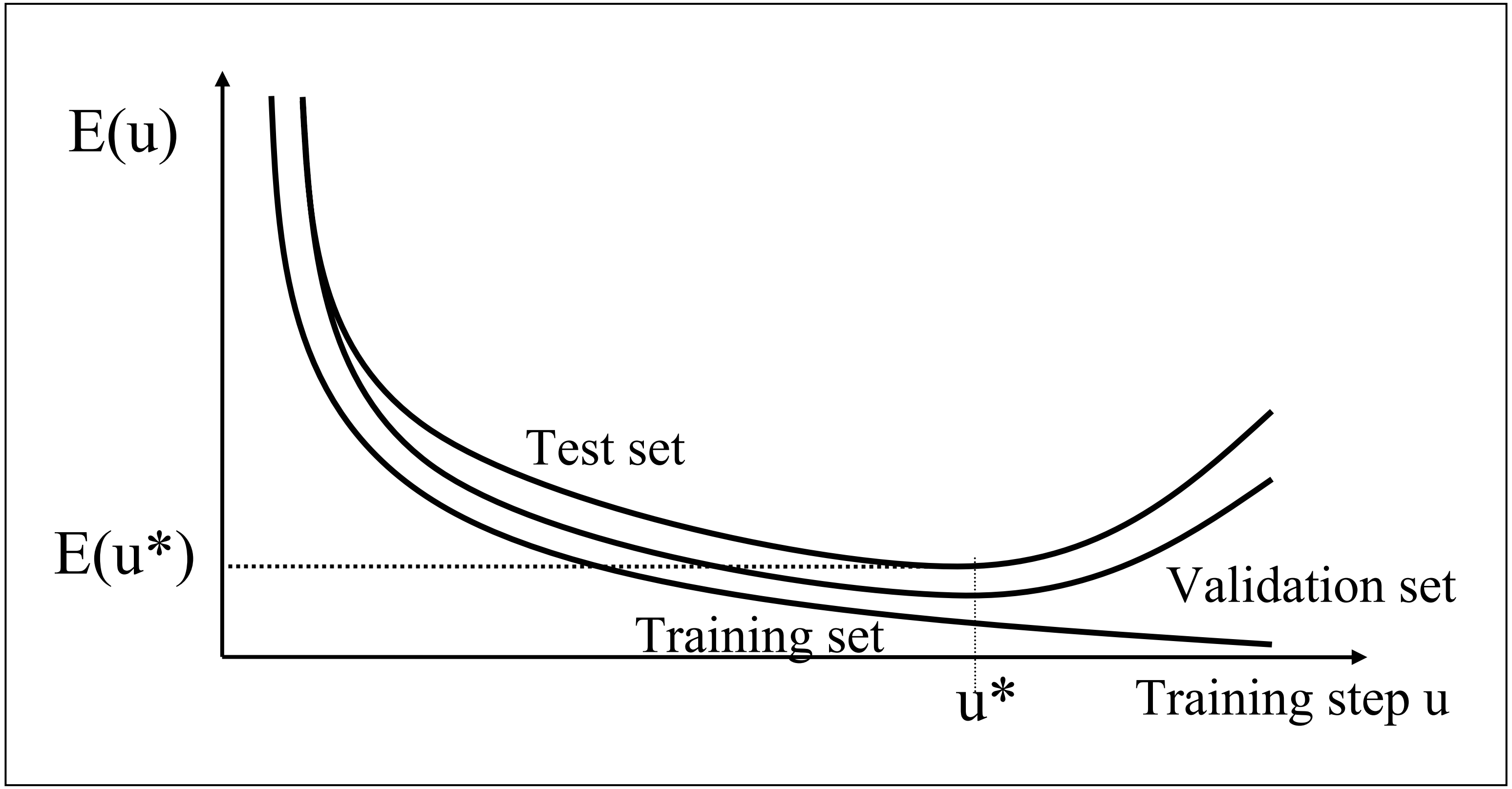
$$\Delta w_{ik}^{q} = -\eta \frac{\partial E^{q}}{\partial w_{ik}} + \alpha \Delta w_{ik}^{q-1}$$

Different non linearly separable problems and number of layers

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

Over-fitting

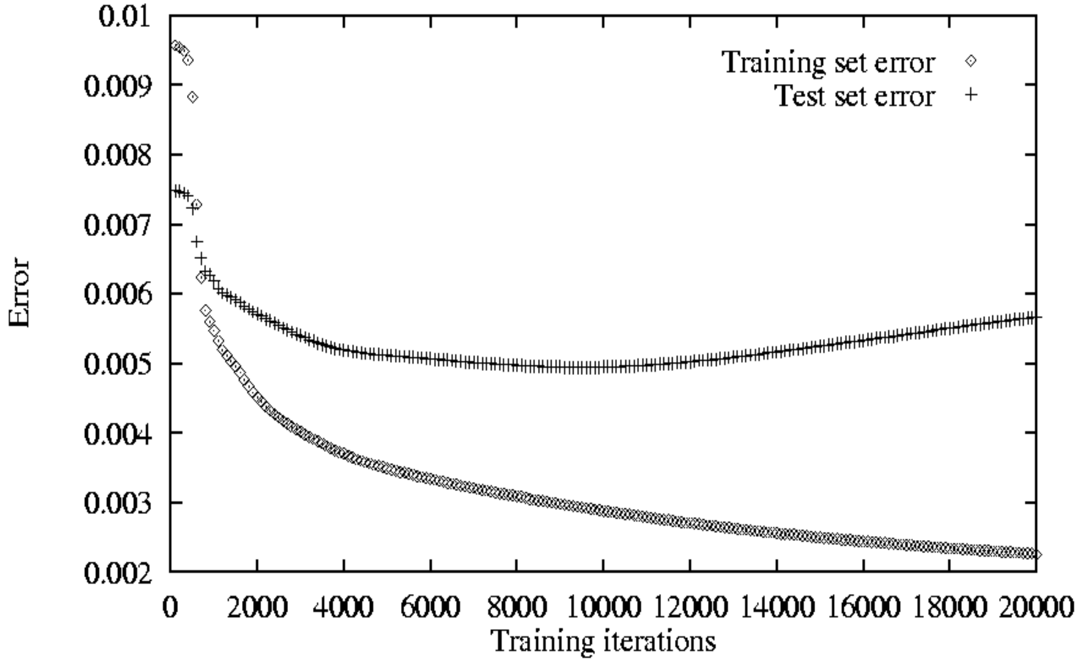
A too large number of parameters can memorize all the examples of the training set with the associated noise, errors and inconsistencies



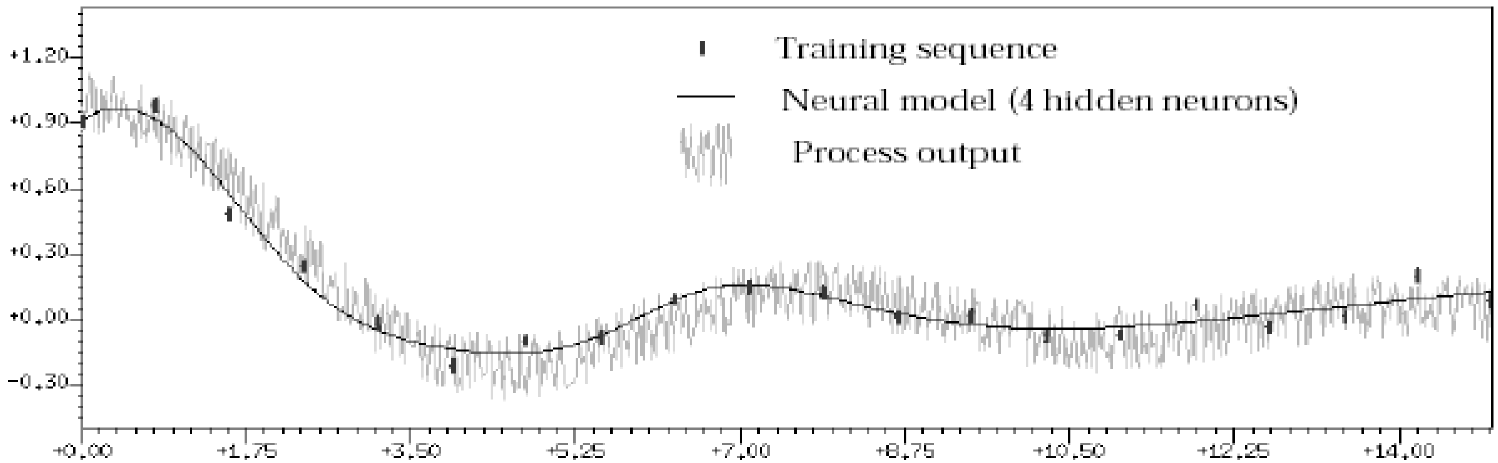
Overtraining in ANNs

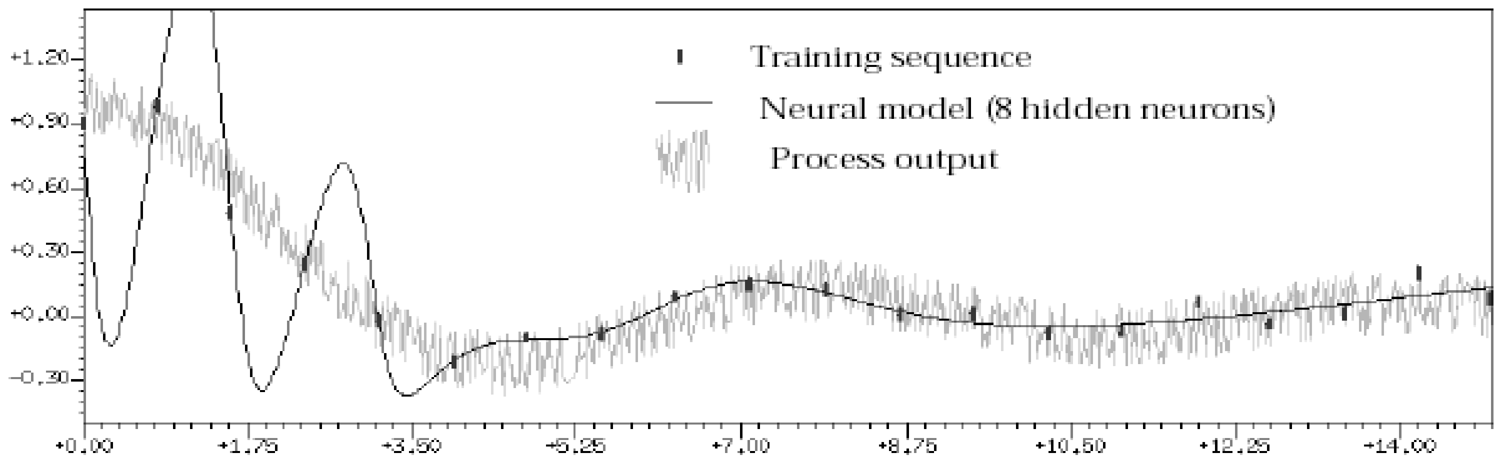
- Recall: Definition of Overfitting
  - h' worse than h on D_{train}, better on D_{test}
- Overtraining: A Type of Overfitting
  - Due to excessive iterations
  - Avoidance: stopping criterion (cross-validation: holdout, *k*-fold)
  - Avoidance: weight decay

Error versus epochs (Example 1)



Choosing the number of neurons





Network size

The universality property requires
a sufficient number of hidden neurons.

- Pruning algorithms

  Start with a large network and gradually remove weights or complete units that do not seem to be necessary

  - Sensitivity methods
  - Penalty-term methods

- Growing algorithms

  Start from a small architecture and allow new units to be added when necessary.

Neural Network as a Classifier

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or ``structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of ``hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

Knowledge Extraction

- **Global approach**

  A tree of symbolic rules is built to represent the whole network. Each rule is then tested against the network behavior until most of training space is covered.

  Disadvantage: huge trees.

- **Local approach**

  The original MLP is decomposed into a series of smaller usually single layered sub-networks. The incoming weights form the antecedent of a symbolic rule for each unit. Those rules are gradually combined together to define a more general set of rules that describes the network as a whole.

  Disadvantage: Because of the distributed knowledge in an ANN hidden units do not typically represent clear logic entities.

RBF networks

- This is becoming an increasingly popular neural network with diverse applications and is probably the main rival to the multi-layered perceptron
- Much of the inspiration for RBF networks has come from traditional statistics and pattern classification techniques (mainly local methods for non-parametric regression)
    - These include function approximation, regularization theory, density estimation and interpolation in the presence of noise [Bishop, 1995]
    - Cover Theorem on non-linear projections into new feature space where difficult decision boundaries maybe linear separable

Numerical approximation of functions

- Consider N data points characterized by p features

$$\{\mathbf{x}_i \in R^m \big| i = 1, \ldots, N\}$$

- and corresponding N outputs (real values)

$$\{d_i \in R \mid i = 1, \ldots, N\}$$

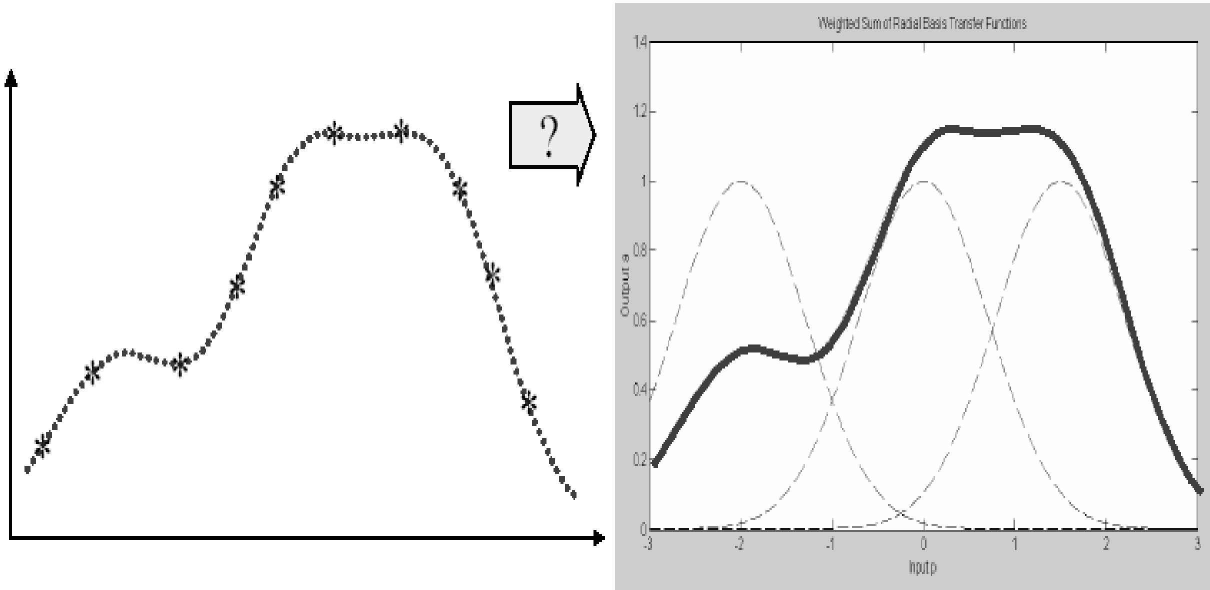
- The aim is to find an unknown function (mapping)

$$f(\mathbf{x}_i) = d_i \quad \forall i = 1, \ldots, N$$

- Complicated functions construct from simple building blocks (local approximations)

Function Approximation with Radial Basis Functions

RBF Networks approximate functions using **(radial) basis functions** as the building blocks.

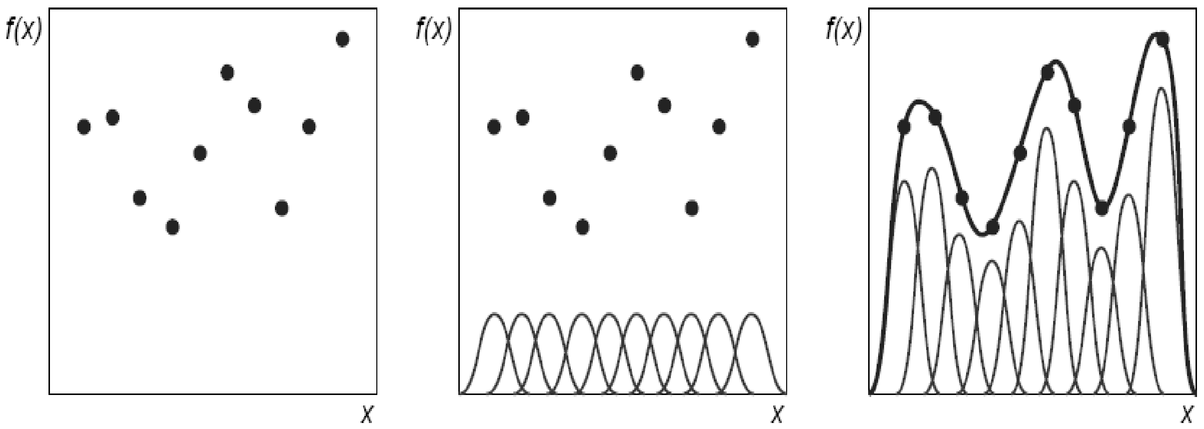


On Exact Interpolation of

- RBFs have their origins in techniques for performing **exact function interpolation** [Bishop, 1995]:
  - Find a function h(x) such that

    $$h(x^n) = t^n \quad \forall n=1, \ldots N$$

- Radial Basis Function approach (Powel 1987):
  - Use a set of N basis functions of the form $\phi(\|x-x^n\|)$, one for each point,where $\phi(.)$ is some non-linear function.
  - Output: $h(x) = \sum_n w_n \phi(\|x-x^n\|)$



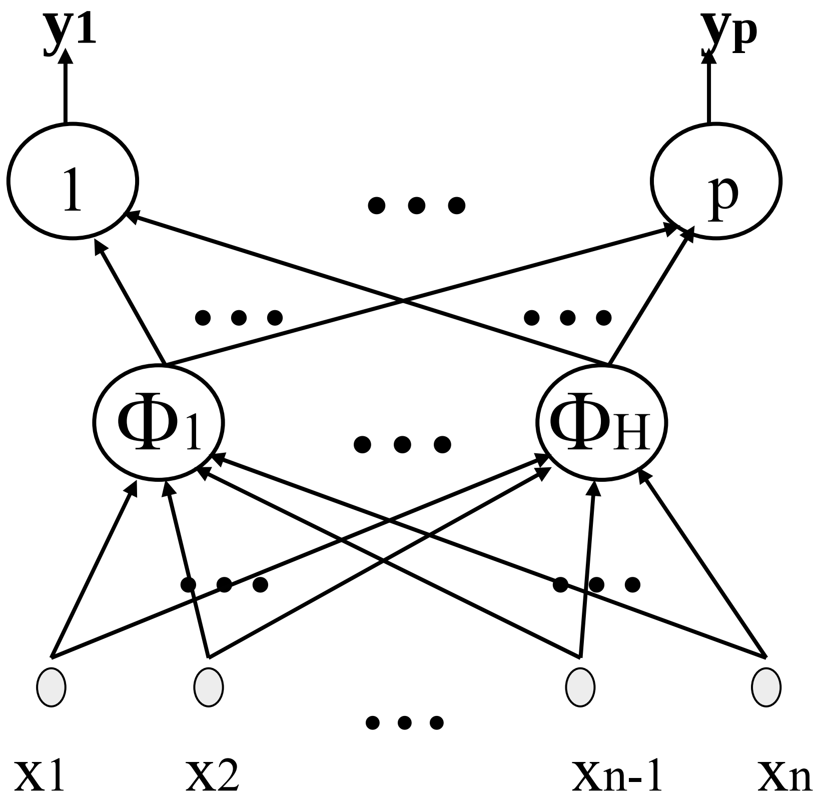
Radial Basis Function Networks

Goal: each hidden unit *k* should represent a *cluster k* in the input space, for example by containing its prototype ***x****k*.

$$y_i(\vec{x}) = \sum_{k=1}^{H} w_{ik}\,\Phi_k(\vec{x}) + w_{i0}$$

$$\Phi_k(\vec{x}) = \Phi_k\left(\left\|\vec{x} - \vec{x}_k\right\|\right)$$

$$\Phi_k(\vec{x}) = e^{-\frac{\left\|\vec{x}-\vec{\mu}_k\right\|}{2\sigma_k^2}}$$



Typical radial functions

Examples:

$$h(r) = r = \left\| X - X_i \right\|$$

$$h(r) = \left(\sigma^2 + r^2\right)^{-\alpha}, \quad \alpha > 0$$

$$h(r) = \left(\sigma^2 + r^2\right)^{\beta}, \quad 1 > \beta > 0$$

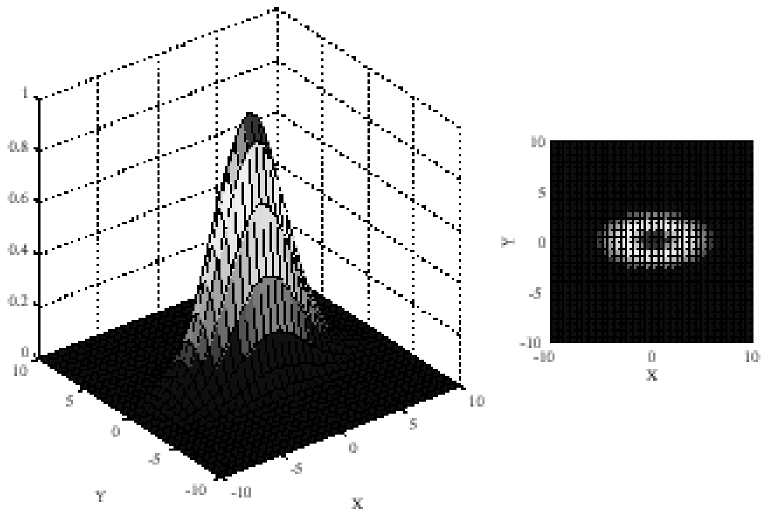
$$h(r) = e^{-(r/\sigma)^2}$$

$$h(r) = (\sigma r)^2 \ln(\sigma r)$$

Simple radial

Inverse multiquadratic
Multiquadratic
Gauss
Thin splines (cienkiej płytki)

Funkcja Gaussa wielu zmiennych

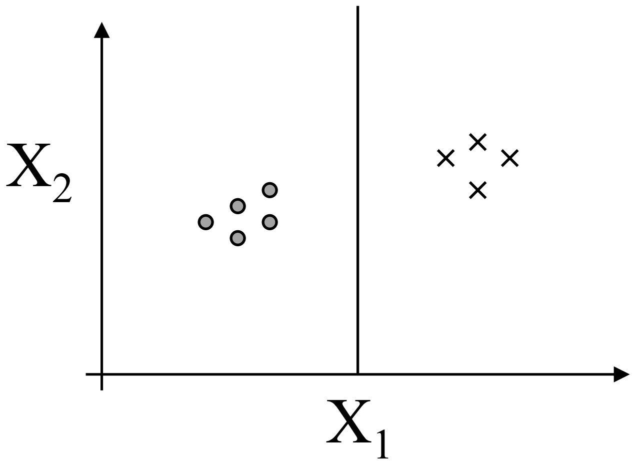


RBFNs and MLPs

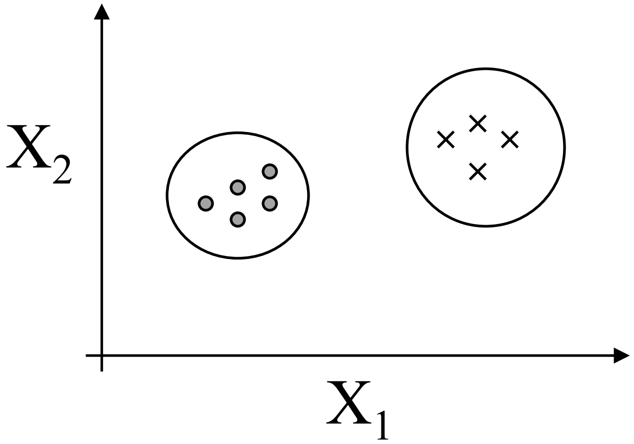
- **Locality.** In RBFNs only a small fraction of Φk is active for each input vector => more efficient training algorithms
- **Separation surfaces.** MLP produces open separation surfaces vs. RBFNs closed separation surfaces
- **Approximation capability**. The universality property still holds for RBFNs if a sufficient number of Φk is given.
- **Interpretability**. RBFNs are easier to interpret than MLPs. Φk can be interpreted as p(cluster k| **x**) and wik as p(Ci|cluster k)

MLPs versus RBFs

- **Classification**
  - MLPs separate classes via hyperplanes
  - RBFs separate classes via hyperspheres
- **Learning**
  - MLPs use distributed learning
  - RBFs use localized learning
  - RBFs train faster
- **Structure**
  - MLPs have one or more hidden layers
  - RBFs have only one layer
  - RBFs require more hidden neurons => curse of dimensionality



MLP



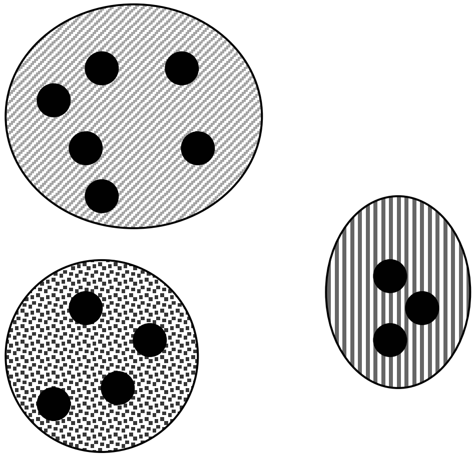
RBF

The hybrid learning strategy

1. Unsupervised training of the RBF parameters
   – **K-means clustering algorithm**
   – **Mixtures of Gaussians**
   – **Kohonen Competitive learning**
2. Supervised training of the weights connecting the hidden and the output layer
   – **Back-Propagation**
   – **Or a special mathematical approaches to solve matrix equations!**

RBF units Unsupervised Training

- K-means algorithm.



$$\vec{\mu}_k = \frac{1}{N_k} \sum_{q \in S_k} \vec{x}^q \qquad \sigma = \frac{1}{H} \sum_{i=1}^{H} \left\| \vec{\mu}_i - \vec{\mu}_j \right\|$$

$$J = \sum_{k=1}^{H} \sum_{q \in S_k} \left\| \vec{x}^q - \vec{\mu}_k \right\|^2$$

- Mixtures of Gaussians.

$$o(\vec{x}) = \sum_{j=1}^{H} \alpha_j(\vec{x}) \Phi_j(\vec{x}) \qquad \ell = \prod_{q=1}^{m} p\left(\vec{x}^q\right)$$

RBFNs Training Algorithms (I)

- Modified Back-Propagation.

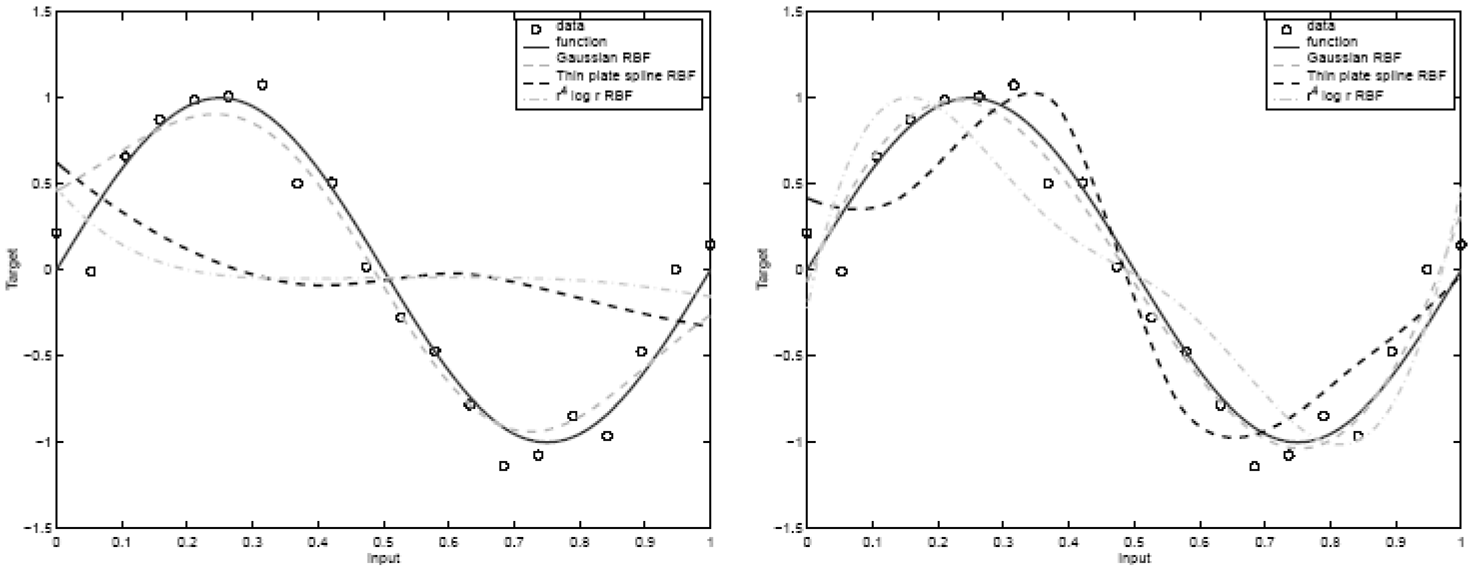
The corresponding expressions of the partial derivatives of the error function have to be evaluated and included into the gradient descent procedure.

- Orthogonal Least Square Algorithm.

RBF units are sequentially introduced. At the first step each RBF is centered on one training pattern; the RBF unit with smallest error is retained. The algorithms continues on the remaining training data.

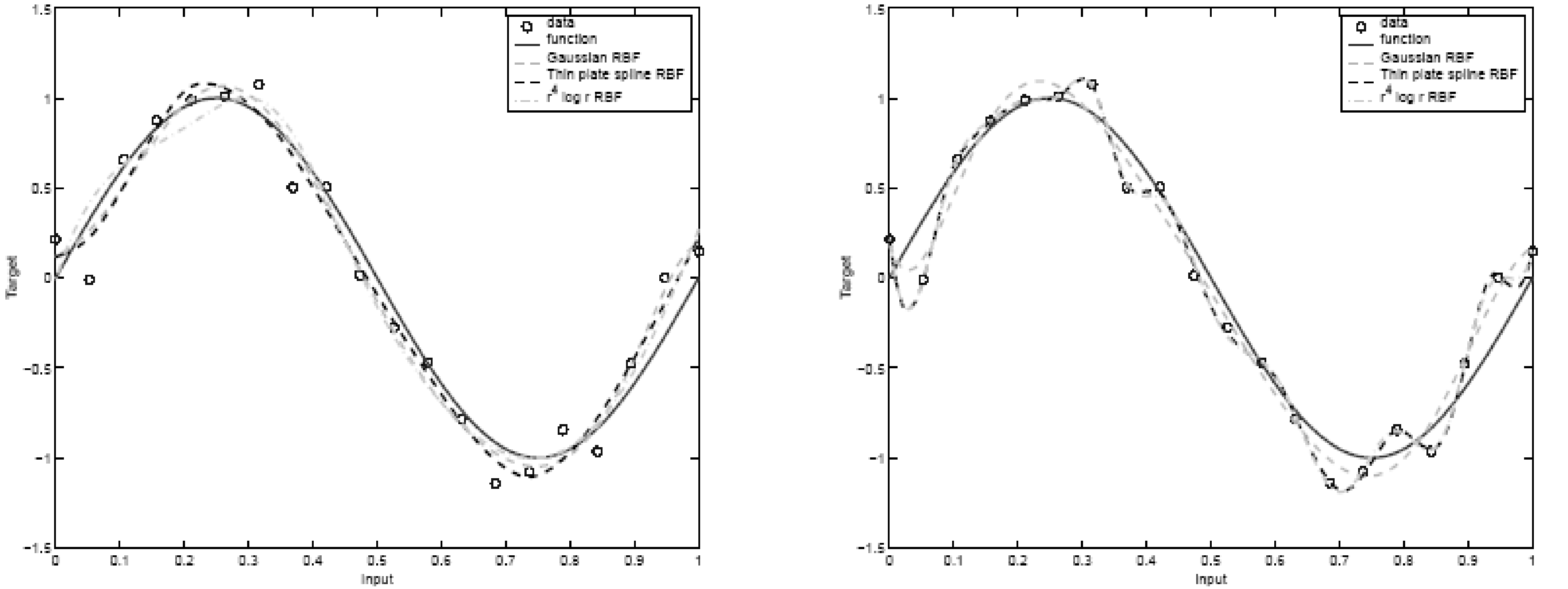
RBF analysis of sinus function

- Following lecture og prof. A.Bartkowiak Uniw. Wrocławski



Rysunek 9.3: *Aproksymacja funkcji sinus wykonana za pomocą sieci* RBF *o H=2 (lewa) i H=4 (prawa) neuronach. Sieci zostały wytrenowane na podstawie 20-elementowej zaburzonej próbki wylosowanej z sinusoidy. Pliki rr2c.eps i rr4c.eps*

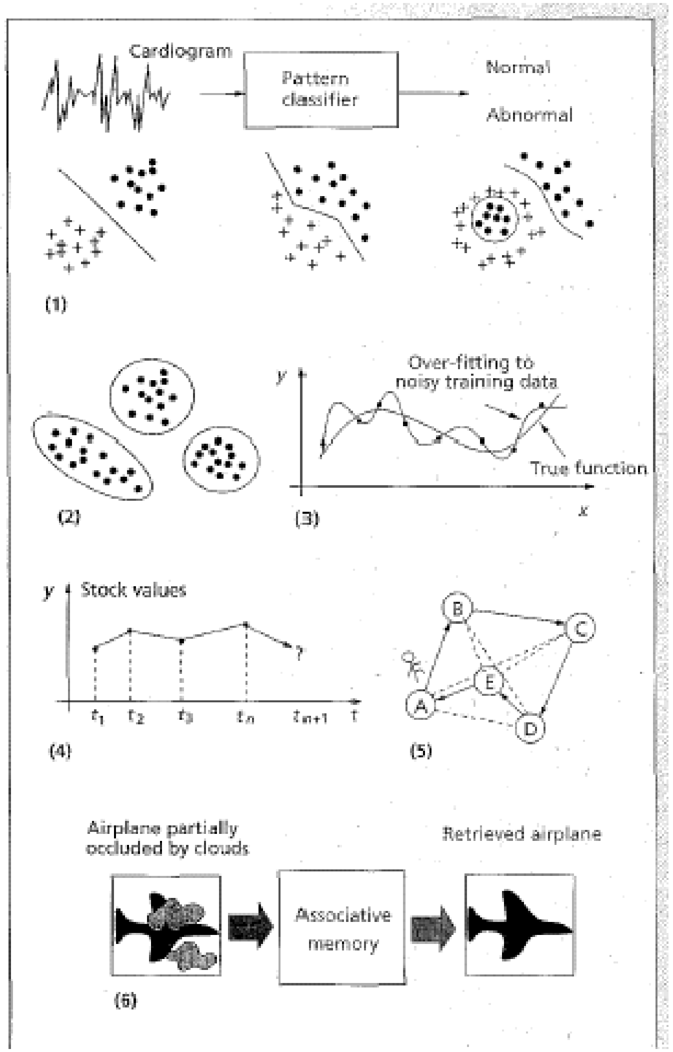
RBF analysis of sinus function (2)



Rysunek 9.4: *RBF 7 i 15 neuronow . To samo, co na rysunku 5.3, ale warstwa ukryta sieci składa się z H=7 (lewa) i H=15 (prawa) neuronów. Pliki rr7c.eps i rr15c.eps*

Tasks for ANN

- Pattern classification
- Function approximation
- Time series and forecasting
- Clustering
- Multidimensional Projections
- Association memory
- Content addressed memory
- Control strategies
- ..



Unsupervised ANN Learning

- Unsupervised Learning

  If the desired answers are not available, not even for a subset of data to use as training set, we use unsupervised learning.

- Similarity and Correlation

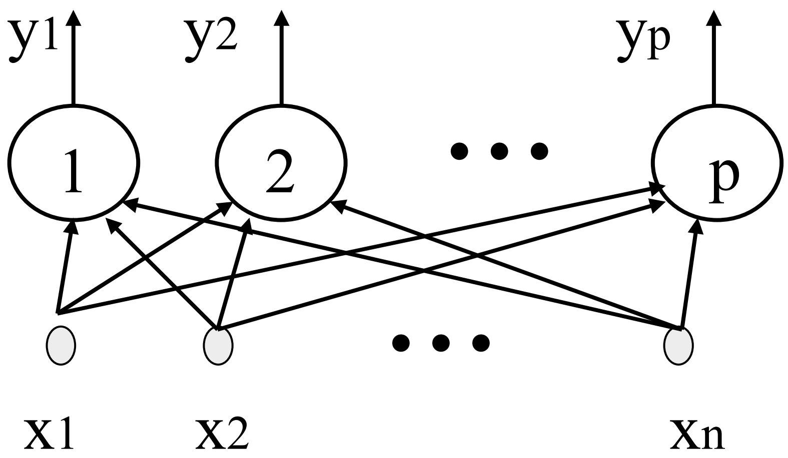
  The network should organize the training data into clusters on the basis of similarity and correlation criteria.

- Redundancy

  This can happen only if there is redundancy in the training data

- **Hebbian Learning** and **Competitive Learning**

Standard Competitive Learning
(winner-take-all)



$$w_{ij} \geq 0$$

$$a_i = \sum_{j=1}^{n} w_{ij} \; x_j = \vec{w}_i^T \; \vec{x}$$

$$y_i = \begin{cases} 1 & if \; : \; \vec{w}_i^T \; \vec{x} = \max_{k=1,\ldots,p} \left(\vec{w}_k^T \; \vec{x} \right) \\ 0 & otherwise \end{cases}$$

$$if \;\; \left\| \vec{w}_i = 1 \right\| \quad y_i = \begin{cases} 1 & if \; : \; \left\| \vec{w}_i - \vec{x} \right\| = \min_{k=1,\ldots,p} \left\| \vec{w}_k - \vec{x} \right\| \\ 0 & otherwise \end{cases}$$

SCL: Training algorithm

Goal:

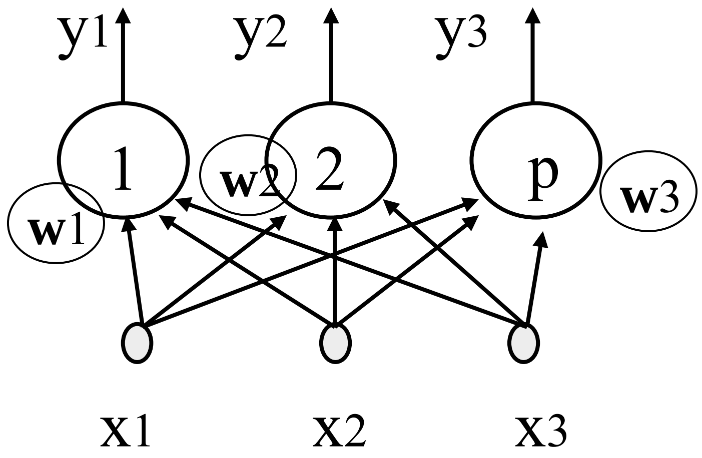
$$\left(\vec{w}_i^{\,q}(t)\right)^T \vec{x}^{\,q} \le \left(\vec{w}_i^{\,q}(t+1)\right)^T \vec{x}^{\,q} = \left(\left(\vec{w}_i^{\,q}(t)\right)^T + \left(\Delta \vec{w}_i^{\,q}(t)\right)^T\right)\vec{x}^{\,q}$$

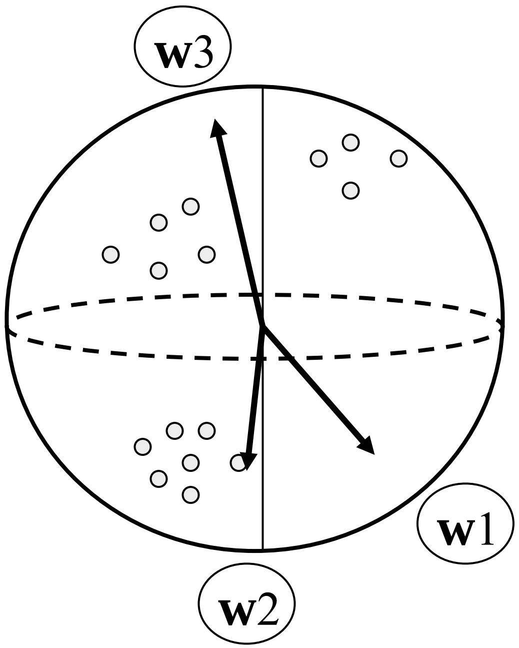
$$\Downarrow$$

$$\Delta\vec{w}_i^{\,q}(t) = \begin{cases} \eta\,\vec{x}^{\,q} \boxed{-\eta\,\vec{w}_i^{\,q}(t)} & if: \left(\vec{w}_i^{\,q}(t)\right)^T \vec{x}^{\,q} = \max_k\left(\left(\vec{w}_k^{\,q}(t)\right)^T \vec{x}^{\,q}\right) \\ 0 & otherwise \end{cases}$$

$\eta > 0$ (usually $0.1 < \eta < 0.7$)

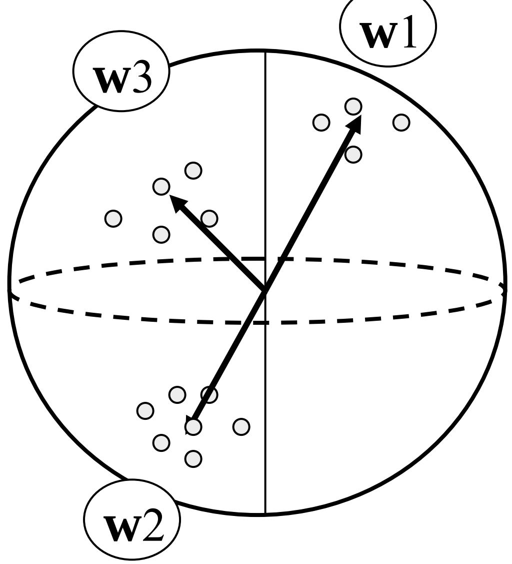
SCL Training algorithm





before
training

after
training



Learning Vector Quantization (LVQ)

LVQ is the Supervised extension of the winner-take-all learning algorithm.

$$\Delta\vec{w}_i^q(t) = \begin{cases} +\eta(t)\left(\vec{x}^q - \vec{w}_i^q(t)\right) & if: \quad class\ of\ unit\ i^q\ is\ correct \\ -\eta(t)\left(\vec{x}^q - \vec{w}_i^q(t)\right) & if: \quad class\ of\ unit\ i^q\ is\ incorrect \\ 0 \quad if: \quad i^q\ is\ not\ a\ winner \end{cases}$$

Improved LVQ

The class of the input vector q is different from the class represented by winner unit i, but it is the same as close unit j.

$$\Delta \vec{w}_i^q(t) = -\eta(t)\left(\vec{x}^q - \vec{w}_i^q(t)\right)$$
$$\Delta \vec{w}_j^q(t) = +\eta(t)\left(\vec{x}^q - \vec{w}_j^q(t)\right)$$
$$\Delta \vec{w}_k(t) = 0 \quad k \neq i, j$$

The class of the input vector q is the same as winner unit i and close unit j.

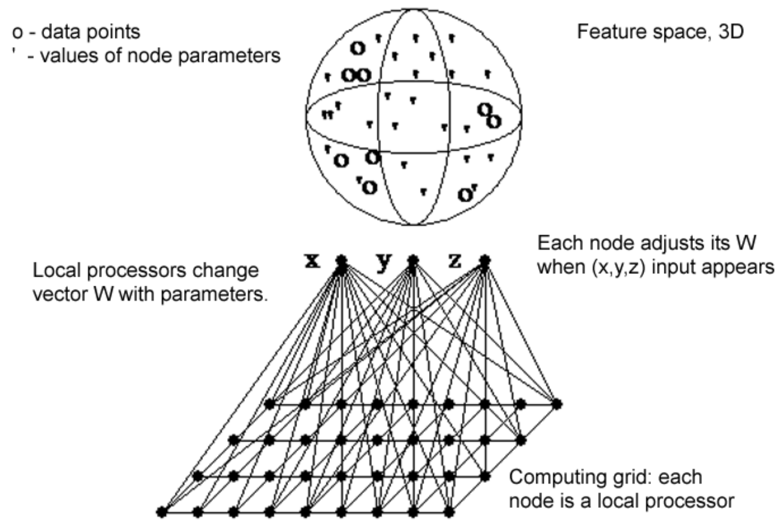
$$\Delta \vec{w}_h^q(t) = +\varepsilon\,\eta(t)\left(\vec{x}^q - \vec{w}_h^q(t)\right) \quad h = i, j$$
$$\Delta \vec{w}_k(t) = 0 \quad k \neq i, j$$

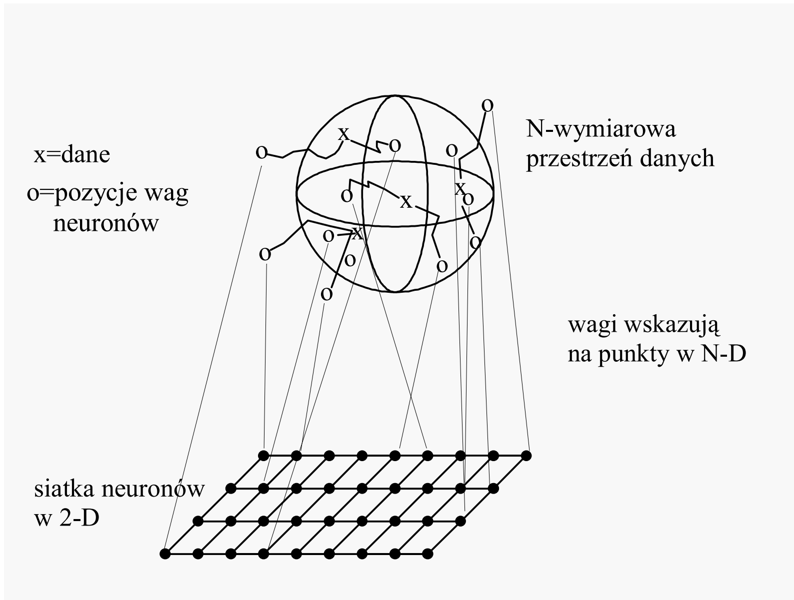
Kohonen Self-Organizing Maps

- Architecture:
  - Kohonen maps consist of a two-dimensional array of neurons, fully connected, with no lateral connections, arranged on a squared or hexagonal lattice
- Learning algorithm:
  - follows the winner-take-all strategy
  - forces close neurons to fire for similar inputs (Self-Organizing Maps)
- Properties:
  - The topology of the input space is preserved

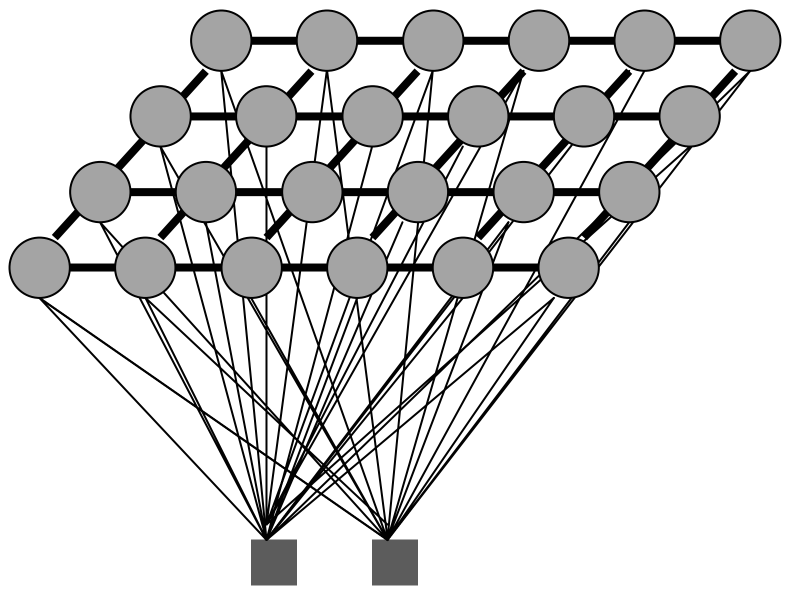
Self organizing maps

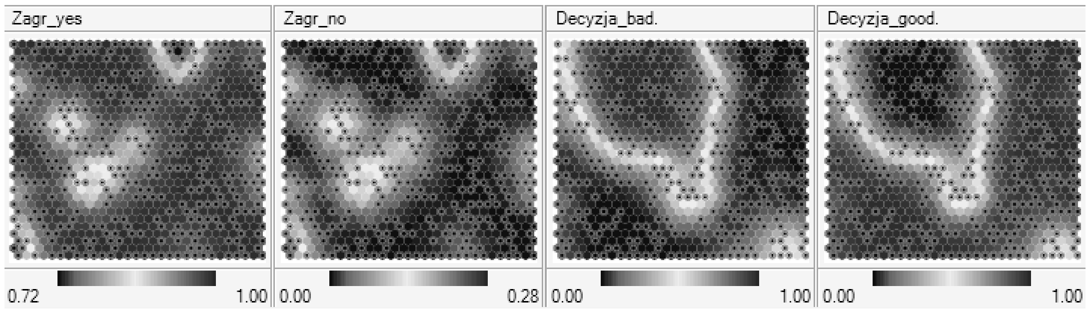
- The purpose of SOM is to map a multidimensional input space onto a topology preserving map of neurons
  - Preserve a topological so that neighboring neurons respond to « similar »input patterns
  - The topological structure is often a 2 or 3 dimensional space
  - the distance and proximity relationship (i.e., topology) are preserved as much as possible
- Similar to specific clustering: cluster centers tend to lie in a low-dimensional manifold in the feature space



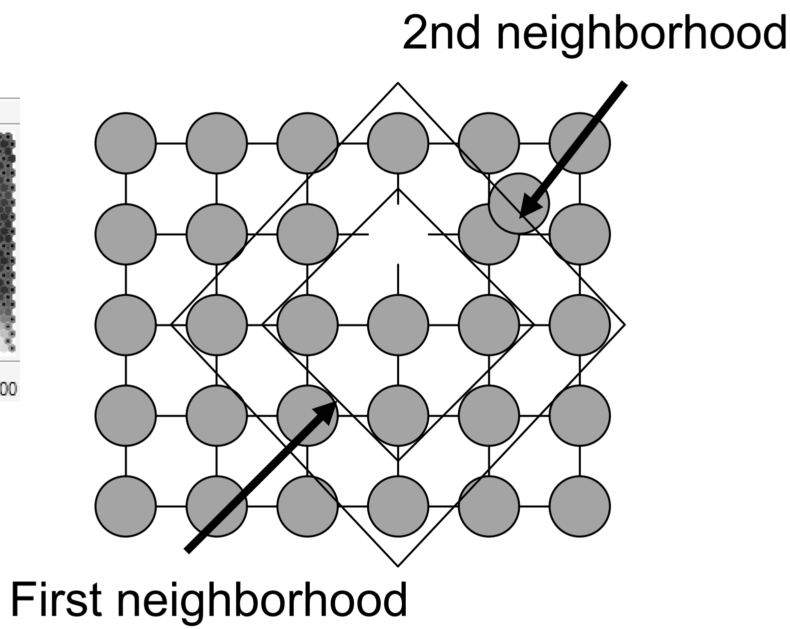


- The activation of the neuron is spread in its direct neighborhood =>neighbors become sensitive to the same input patterns
- Block distance
- The size of the neighborhood is initially large but reduce over time => Specialization of the network





Visualisation of an influence of different patterns on neuron outputs



SOM Learning Algorithm

Winner take-all learning rule

$$\Delta\vec{w}_k^q(t) = +\eta(t)\,\Lambda\left(k, i^q, t\right)\left(\vec{x}^q - \vec{w}_k^q(t)\right)$$

for all units k

Neighborhood function

$$\Lambda\left(k, i^q, t\right) = \exp\left(\frac{-\left\| r_k - r_{i^q} \right\|^2}{2\sigma(t)^2}\right)$$

Quantization Error

$$Q = \frac{1}{m}\sum_{q=1}^{m}\left\| \vec{x}^q - \vec{w}_i^q \right\|^2$$

Average Distortion

$$D = \frac{1}{m}\sum_{q=1}^{m}\Lambda\left(i^q, i^q, t\right)\left\| \vec{x}^q - \vec{w}_i^q \right\|^2$$

SOM algorithm

$\mathbf{X}^T=(X_1, X_2 .. X_d)$, samples from feature space.
Create a grid with nodes $i = 1 .. K$ in 1D, 2D or 3D,
each node with d-dimensional vector $\mathbf{W}^{(i)T} = (W_1^{(i)}\ W_2^{(i)} .. W_d^{(i)})$,
$\mathbf{W}^{(i)} = \mathbf{W}^{(i)}(t)$, changing with t – discrete time.

1. Initialize: random small $\mathbf{W}^{(i)}(0)$ for all i=1...K.
   Define parameters of neighborhood function $h(|r_i - r_c|/\sigma(t),t)$
2. Iterate: select randomly input vector $\mathbf{X}$
3. Calculate distances $d(\mathbf{X},\mathbf{W}^{(i)})$, find the winner node $\mathbf{W}^{(c)}$ most similar (closest to) $\mathbf{X}$
4. Update weights of all neurons in the neighborhood $O(r_c)$
5. Decrease the influence $h_0(t)$ and shrink neighborhood $\sigma(t)$.
6. If in the last T steps all $\mathbf{W}^{(i)}$ changed less than ε stop.

Where to use SOM

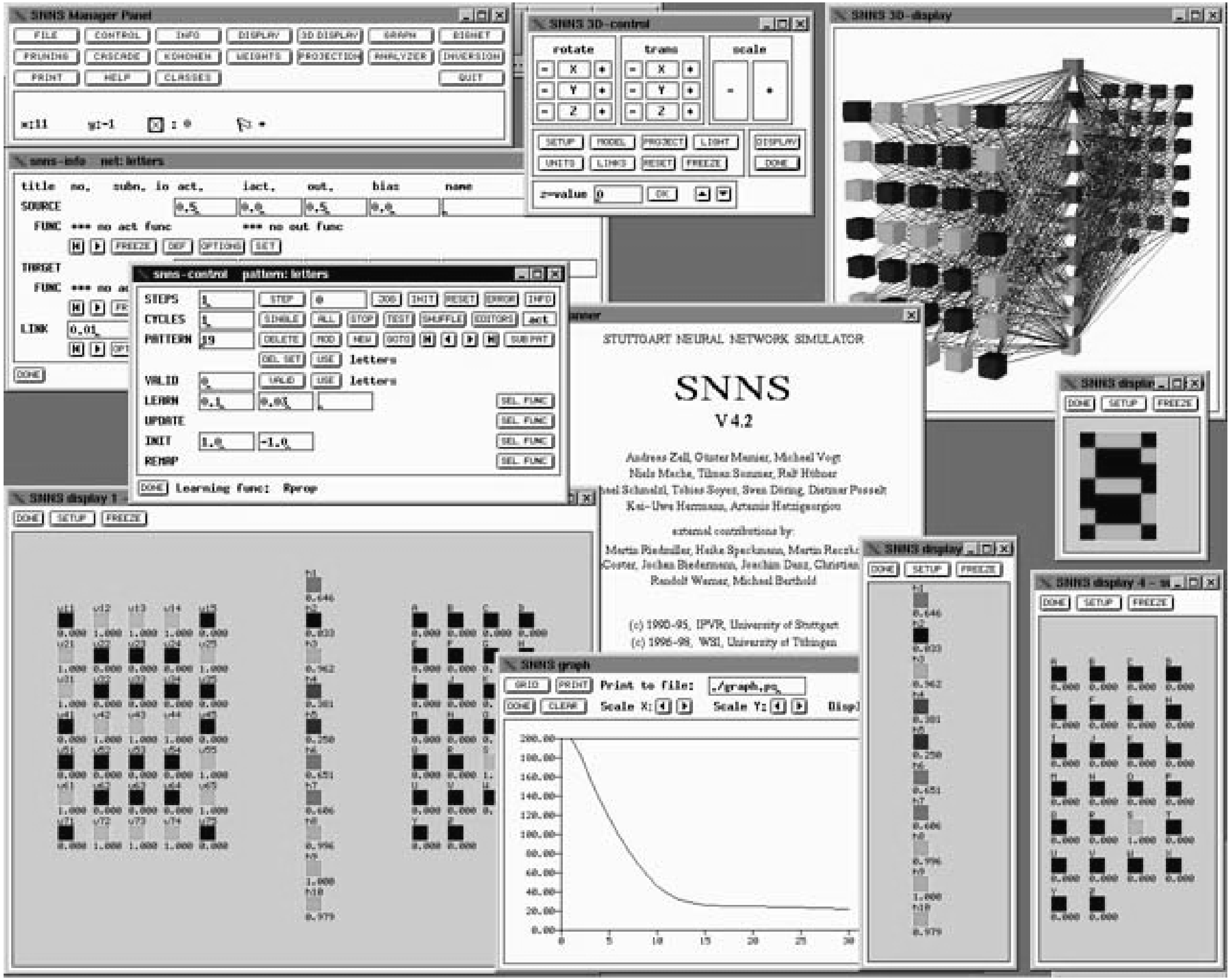
- Natural language processing: linguistic analysis, parsing, learning languages, hyphenation patterns.
- Optimization: configuration of telephone connections, VLSI design, time series prediction, scheduling algorithms.
- Signal processing: adaptive filters, real-time signal analysis, radar, sonar seismic, USG, EKG, EEG and other medical signals ...
- Image recognition and processing: segmentation, object recognition, texture recognition ...
- Content-based retrieval: examples of WebSOM, Cartia, VisierPicSom – similarity based image retrieval.

- More on SOM – see earlier lecture on clustering

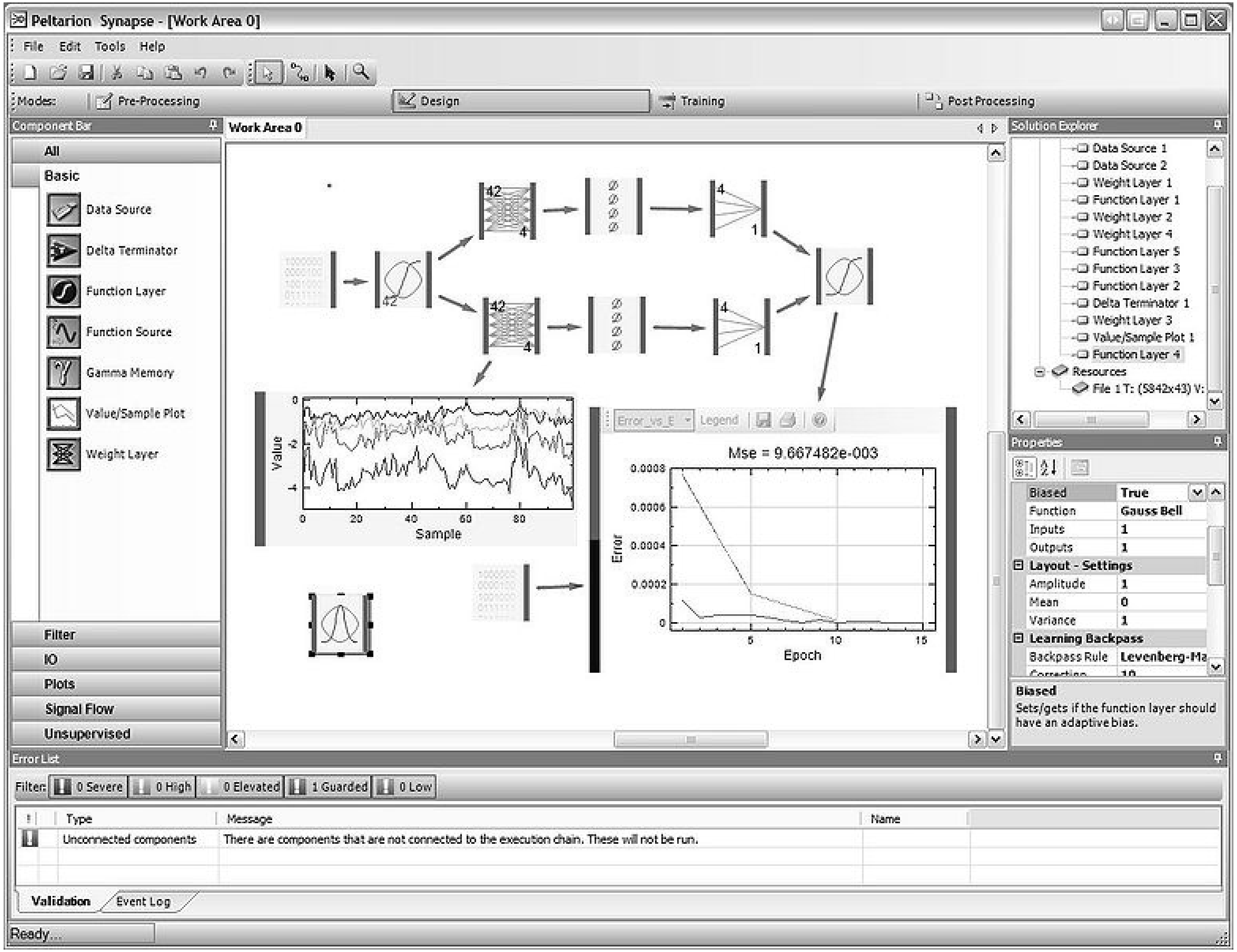
Software Tools

- Commercial products, e.g.
  - Matlab Toolbox
  - Statistica Neural Networks
  - Peltarion Synapse
  - NeuroXL
  - …
- Many others
  - Sttugart Neural Simulator
  - Limitted options WEKA, RapidMiner
  - Many university projects.e.g *NuClass7* Arlington US
  - …

SSN (Univ. Sttugart)

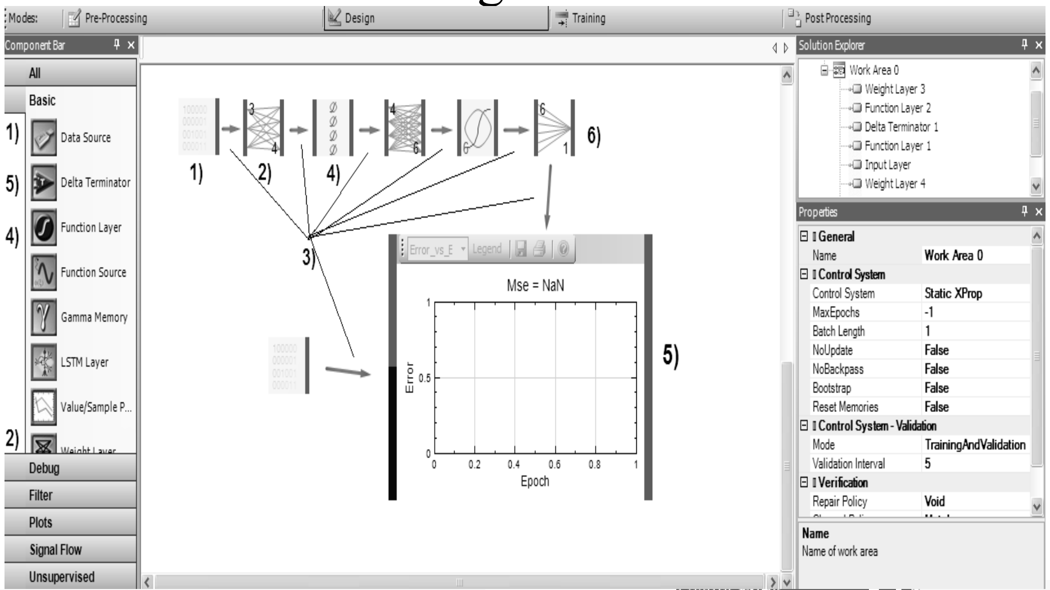


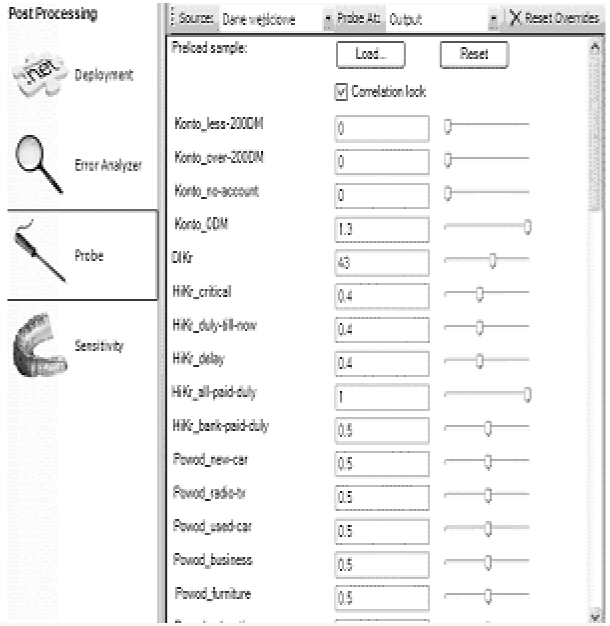
Components in Process

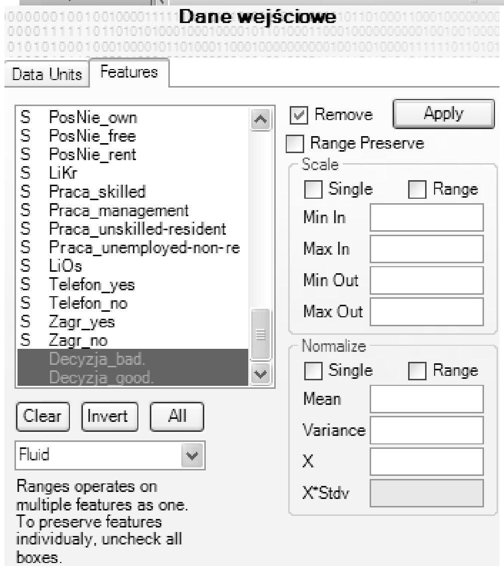


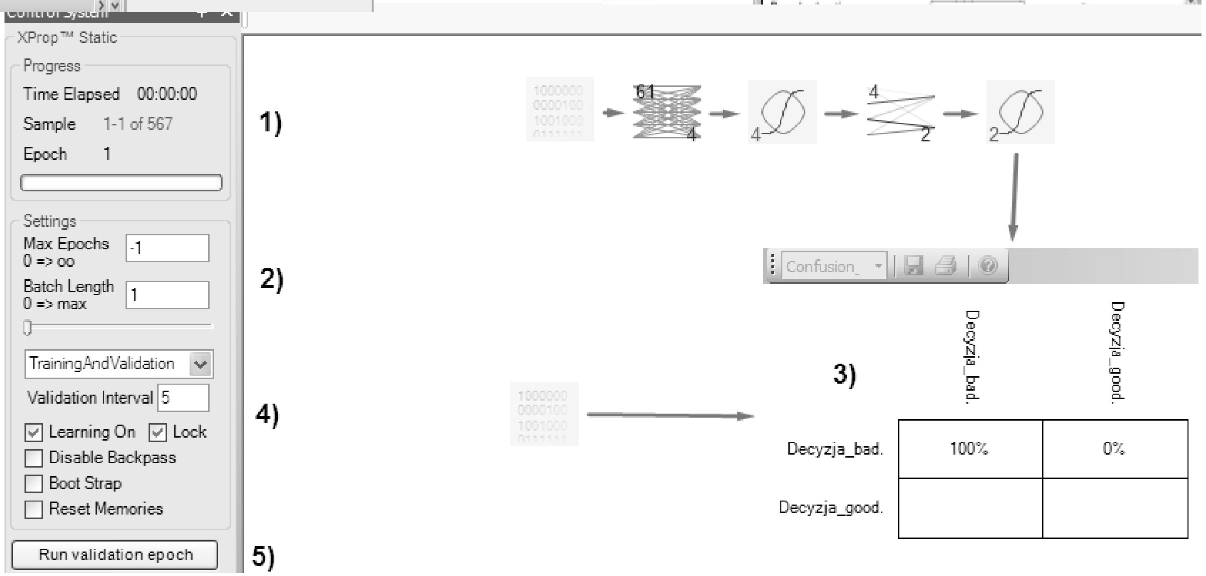
Constructing and Learning ANN in Synapse

- German credit data (UCI repository) – prediction of paying loans by bank customers / 700 good decisions and 300 bad ones



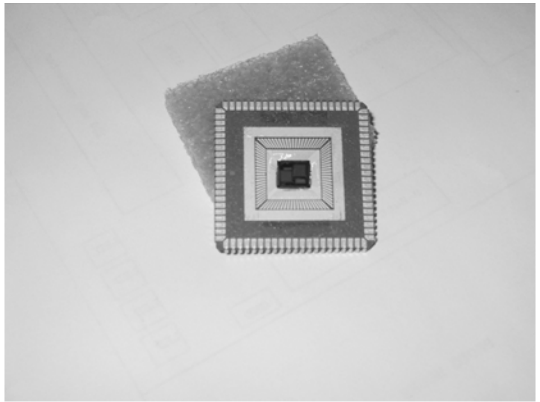


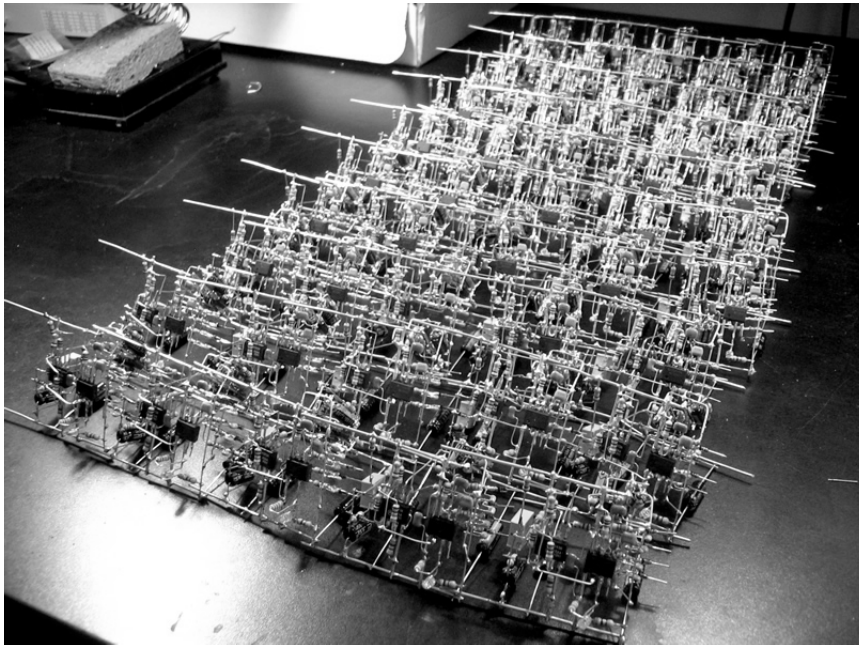


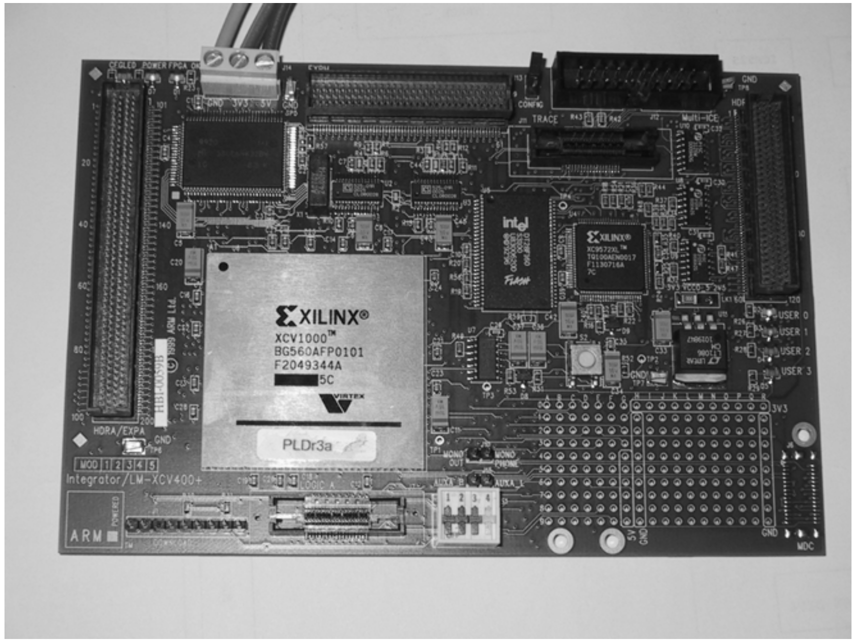


Hardware

- Usually more costly
- Specialized electronic devices
- Need for a real, popular application
- However, FPGA implementing ?







Applications

- Aerospace
  - High performance aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations, aircraft component fault detectors
- Automotive
  - Automobile automatic guidance systems, warranty activity analyzers
- Banking
  - Check and other document readers, credit application evaluators
- Defense
  - Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification
- Electronics
  - Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling

Applications

- Financial
  - Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction
- Manufacturing
  - Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process systems
- Medical
  - Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency room test advisement

Applications

- Robotics
  - Trajectory control, forklift robot, manipulator controllers, vision systems
- Speech
  - Speech recognition, speech compression, vowel classification, text to speech synthesis
- Securities
  - Market analysis, automatic bond rating, stock trading advisory systems
- Telecommunications
  - Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems
- Transportation
  - Truck brake diagnosis systems, vehicle scheduling, routing systems

Conclusions

- ANNs are roughly based on the simulation of biological nervous systems
- An equivalence can be established between many ANN paradigms and statistical analysis techniques
    - Perceptron as a non-linear regression function
    - The auto-associator projects input data onto a PC space
    - RBFNs can be interpreted as statistical classifiers
    - etc …
- ANNs drawbacks:
    - Lack of criteria to define the optimal network size => genetic algorithms?
    - Many parameters to tune
    - Hard interpretation of the ANN analysis process => fuzzy models?
    - Time and cost computational requirements

References

- J. Hertz, A. Krogh, R.G. Palmer, “Introduction to the theory of Neural Computation”, Addison-Wesley, 1991.
- C.M. Bishop, “Neural Networks for pattern recognition”, Oxford University Press, New York, 1995.
- S. Haykin, “Neural Networks, a comprehensive foundation”, IEEE Press, 1994.
- J.M. Zurada, R.J. Marks, C.J. Robonson Eds., “Computational Intelligence imitating life”, IEEE Press, New York, 1994.
- Many others

References in Polish Language

- Osowski Stanisław: Sieci Neuronowe do przetwarzania informacji. Warszawa 2000
- J.M. Zurada, Baruch: Sztuczne sieci neuronowe, PWN.
- Several books by R.Tadeusiewicz
- Krawiec K., Stefanowski J.: Uczenie maszynowe i sieci neuronowe, Wydawnictwo Politechniki Poznańskiej, Poznań 2004
- WWW teaching materials,np:
  - prof. Włodzisław Duch, UMK Toruń
  - prof. Anna Bartkowiak UWr Wrocław
  - My own slides for II part of the course Machine Learning
- Many others

Any questions, remarks?

?