Artificial neural networks and their applications

Marcin Pluciński
mplucinski@wi.zut.edu.pl

Chair of Artificial Intelligence Methods and Applied Mathematics

Faculty of Computer Science and Information Technology
West Pomeranian University of Technology

History of artificial neural networks

1943: McCulloch i Pitts – the first model of artificial neuron.

1949: Hebb – a mechanism of the information remembering by biological neurons.

1958: Rosenblatt – Perceptron neural network.

1960: Widrow – MADALINE neural network.

1968: Minsky i Papert – critical voice.

Biological neuron

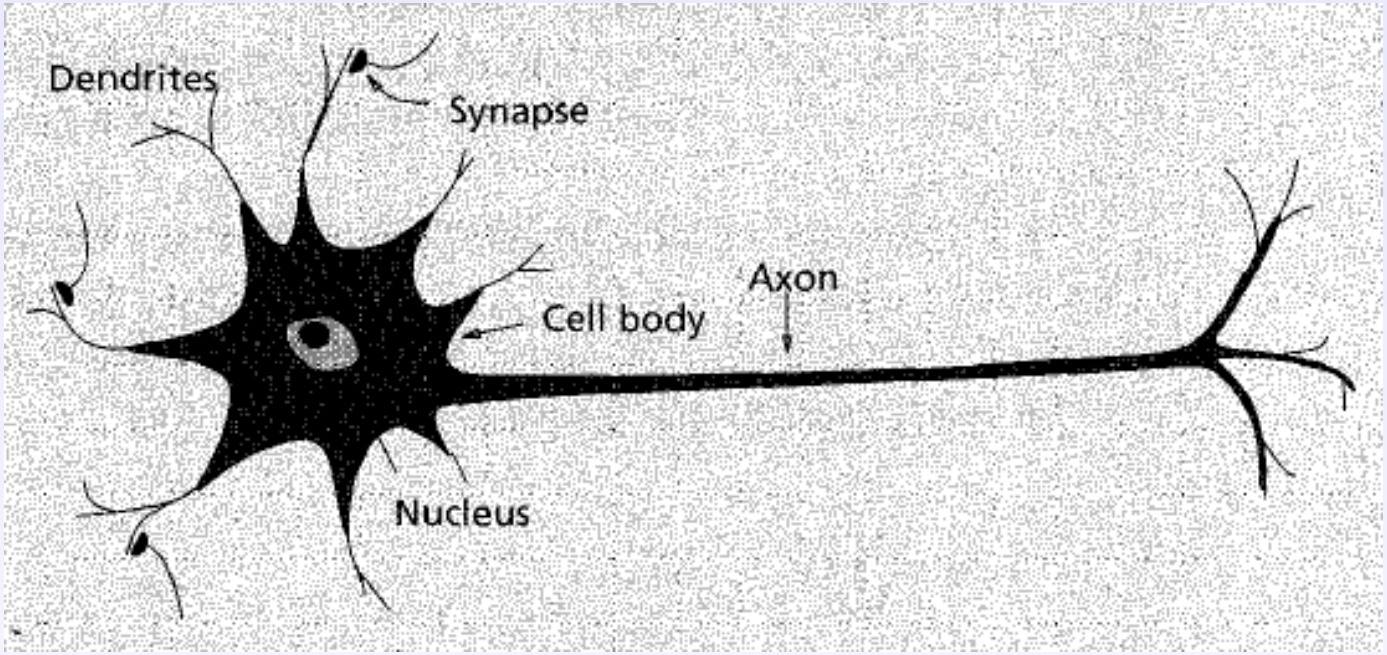
Neuron

Neuron is the basic building block of the nervous system. It is a cell, which is able to receive and transmit electrical signals.

Działanie

If the value of the electric signal put to the neuron exceeds a certain threshold, the neuron is stimulated. Stimulated neuron is discharged and the resulting signal is sent to other neurons. As long as the input signals exceeds a certain threshold, discharge volume remains the same.

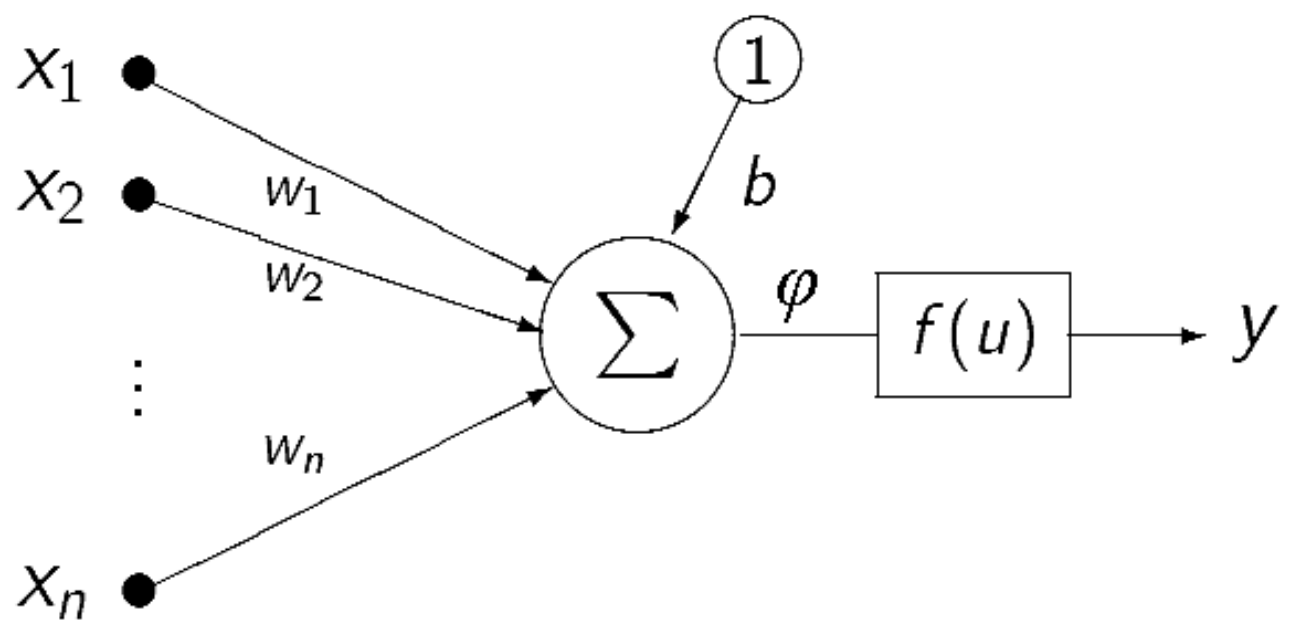
Biological neuron



The neuron consists of the following elements.

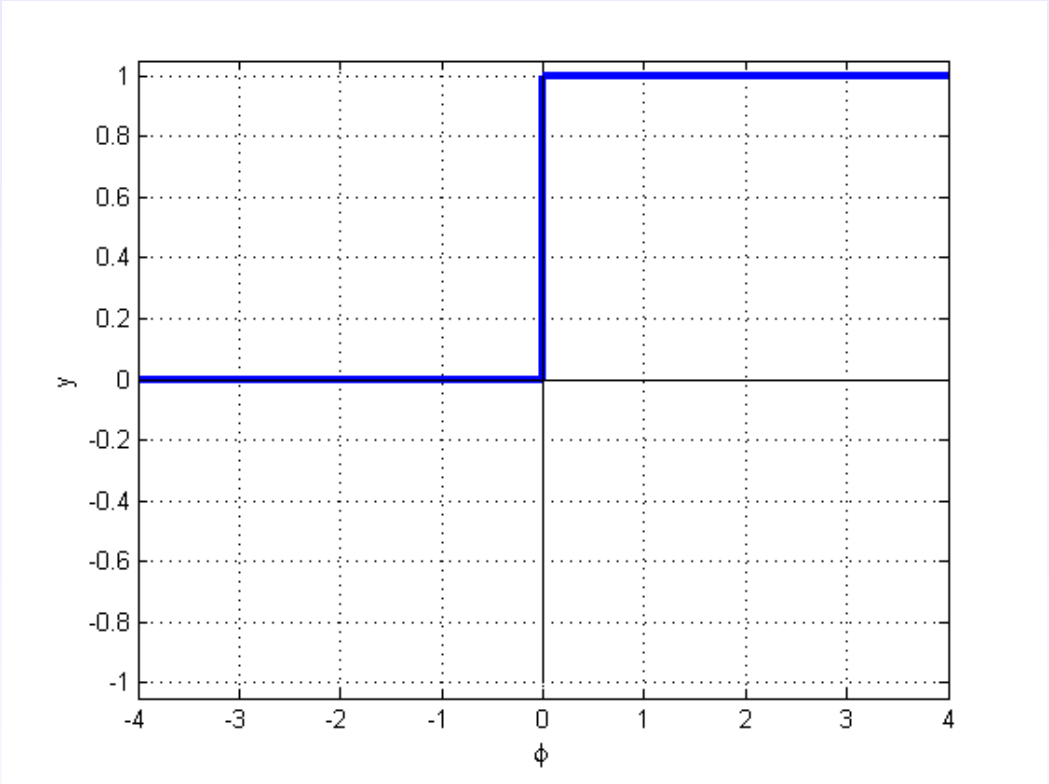
1. Many dendrites, which take impulses from other neurons.
2. The cell body with the nucleus.
3. One axon, which transmits the signal to next cells.
4. Synapses – neurotransmitters which weaken or strengthen the output signal.

Artificial neuron



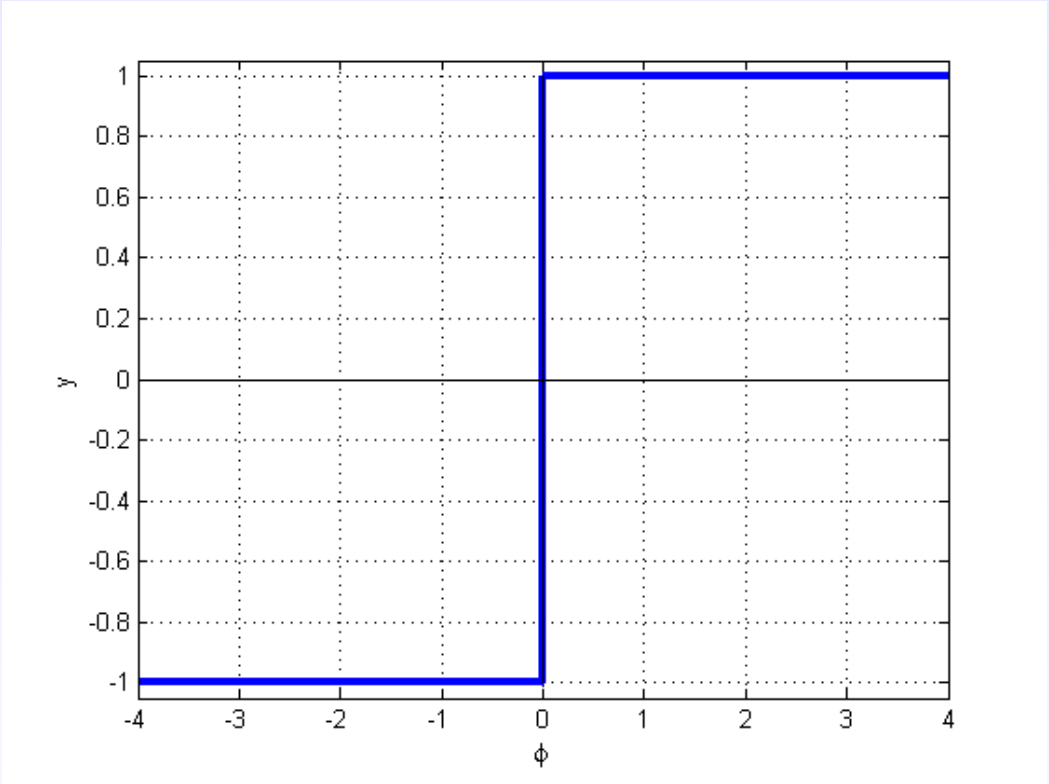
$$y(\mathbf{x}) = f(\sum_{i=1}^{n} w_i x_i + b) = f(\mathbf{w}^T \mathbf{x} + b)$$

Activation functions – unipolar threshold (step) function



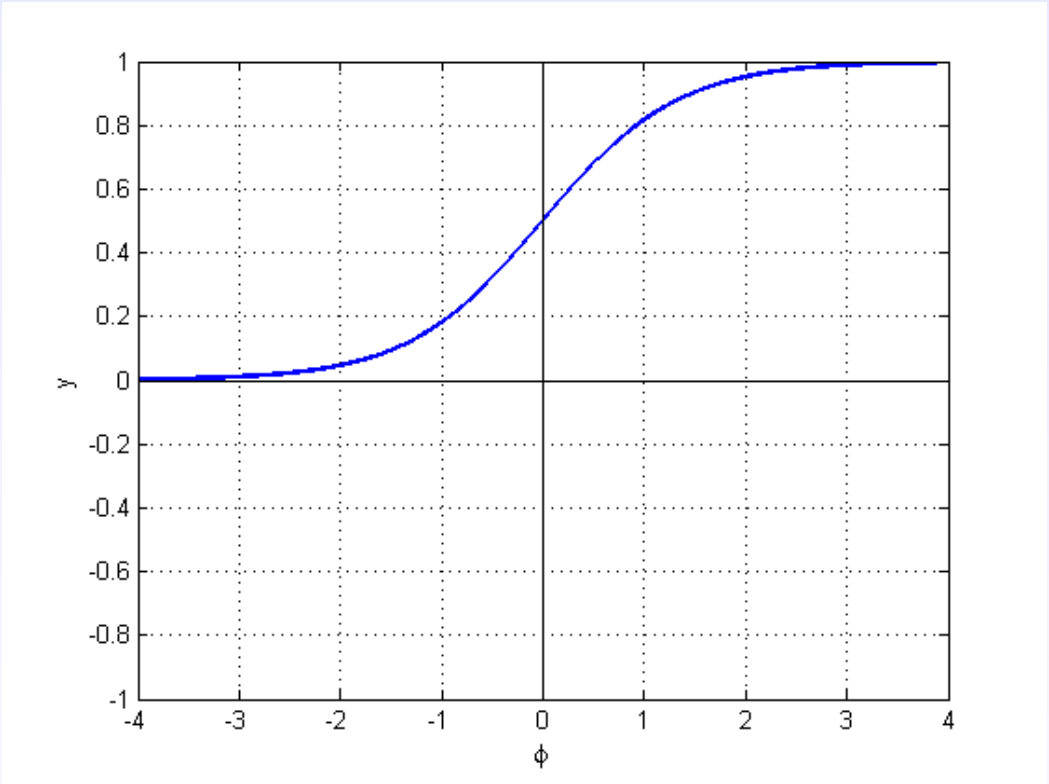
$$f(\phi) = \begin{cases} 1, & \text{for} \quad \phi > 0 \\ 0, & \text{for} \quad \phi \leq 0 \end{cases}$$

Activation functions – bipolar threshold (step) function



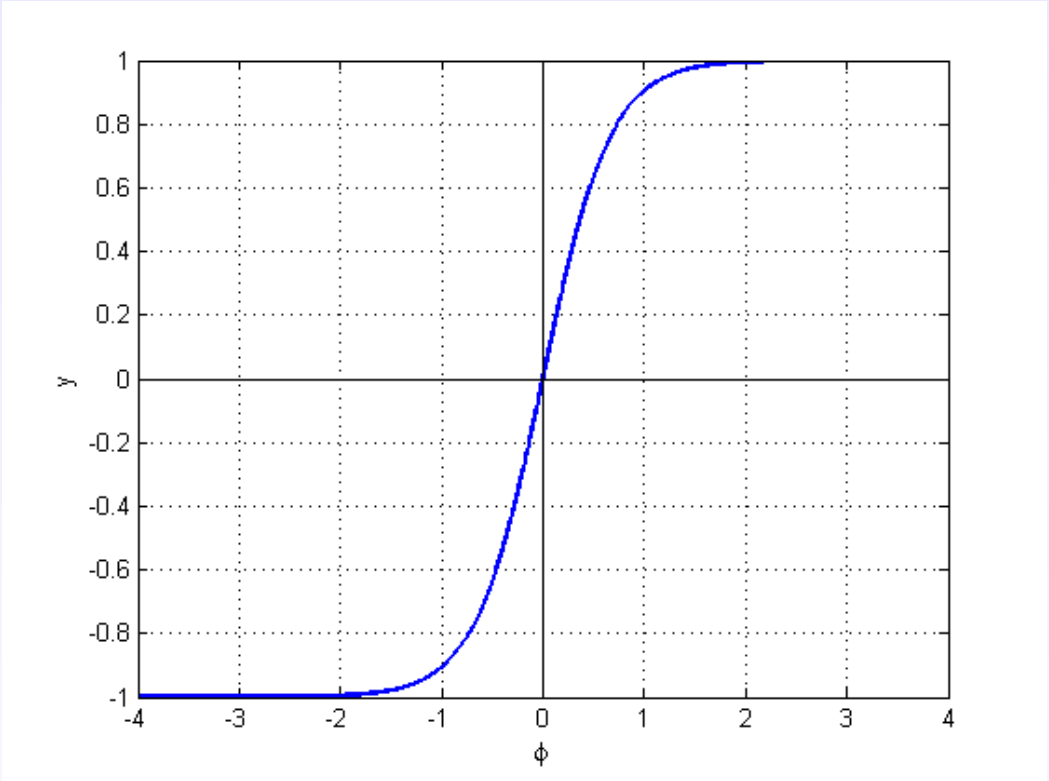
$$f(\phi) = \begin{cases} 1, & \text{for} \quad \phi > 0 \\ -1, & \text{for} \quad \phi \leq 0 \end{cases}$$

Activation functions – sigmoid (logistic) function



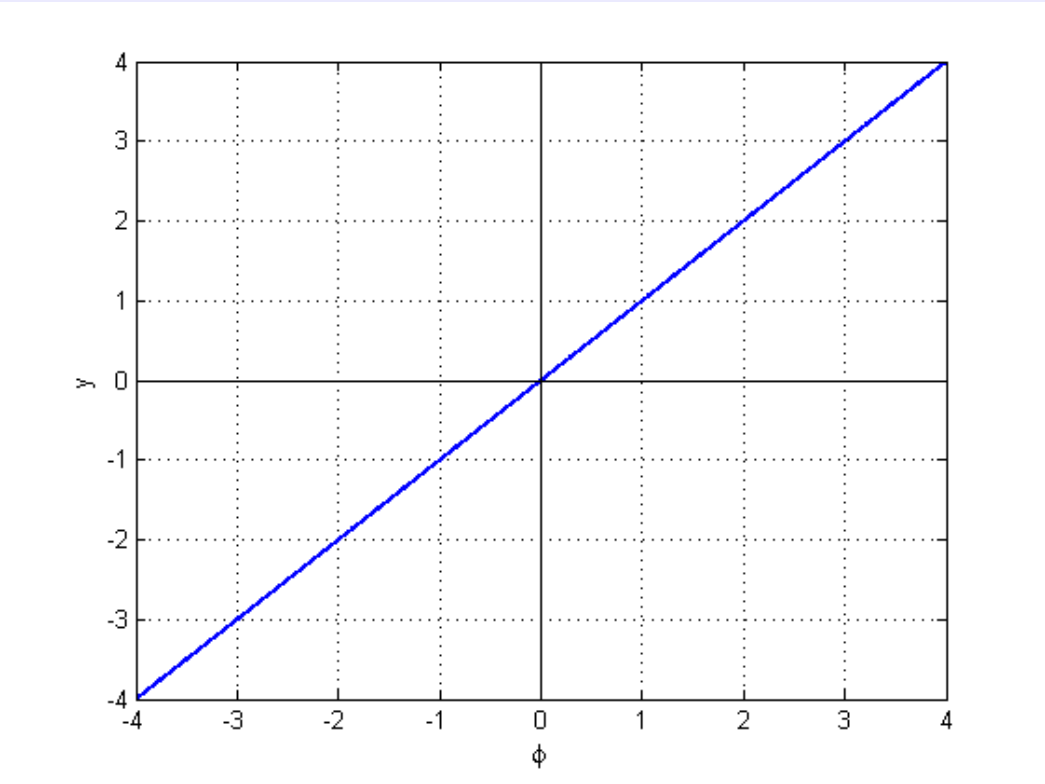
$$f(\phi) = \frac{1}{1 + e^{-\beta\phi}}$$

Activation functions – hyperbolic tangent function



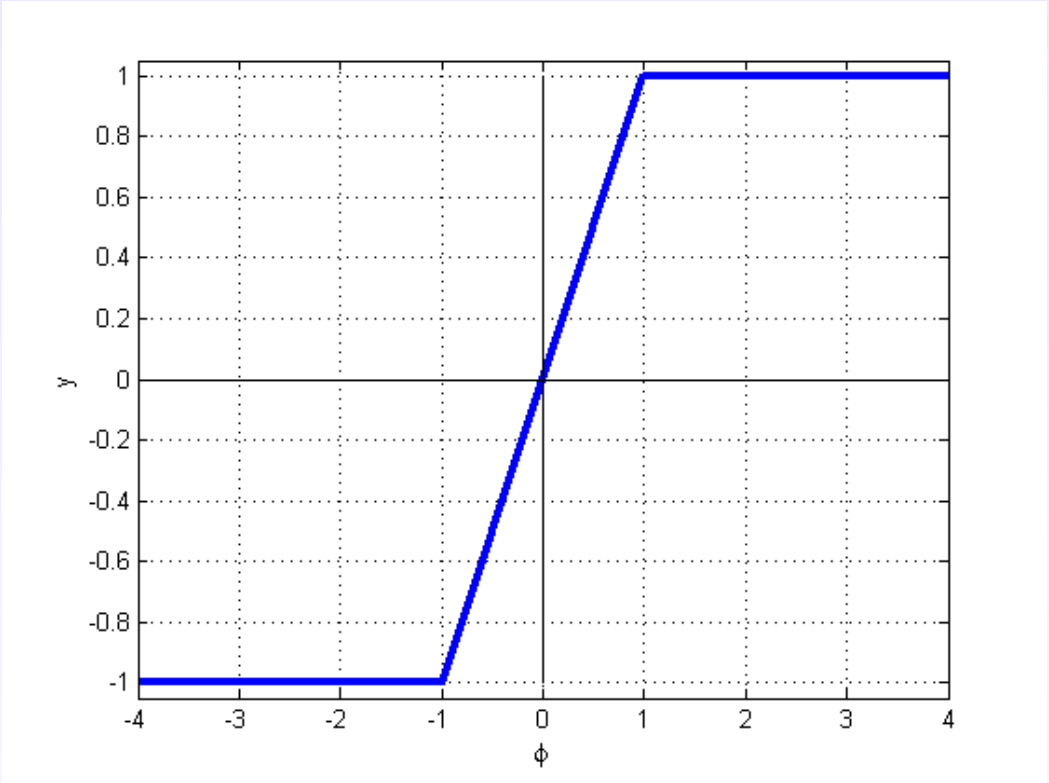
$$f(\phi) = \tanh(\phi) = \frac{e^{\beta\phi} - e^{-\beta\phi}}{e^{\beta\phi} + e^{-\beta\phi}}$$

Activation functions – linear function



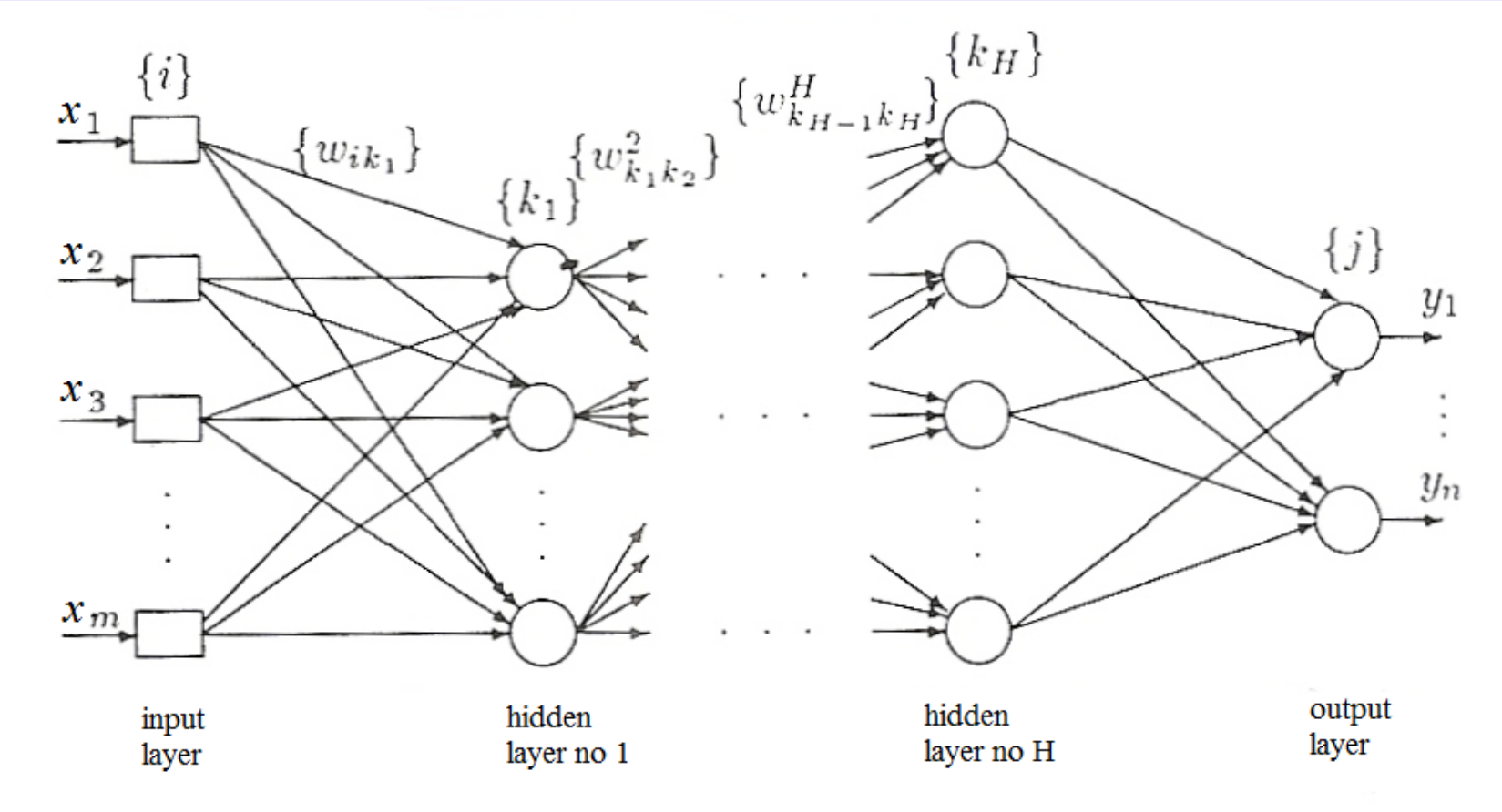
$$f(\phi) = \phi$$

Activation functions – piecewise-linear function



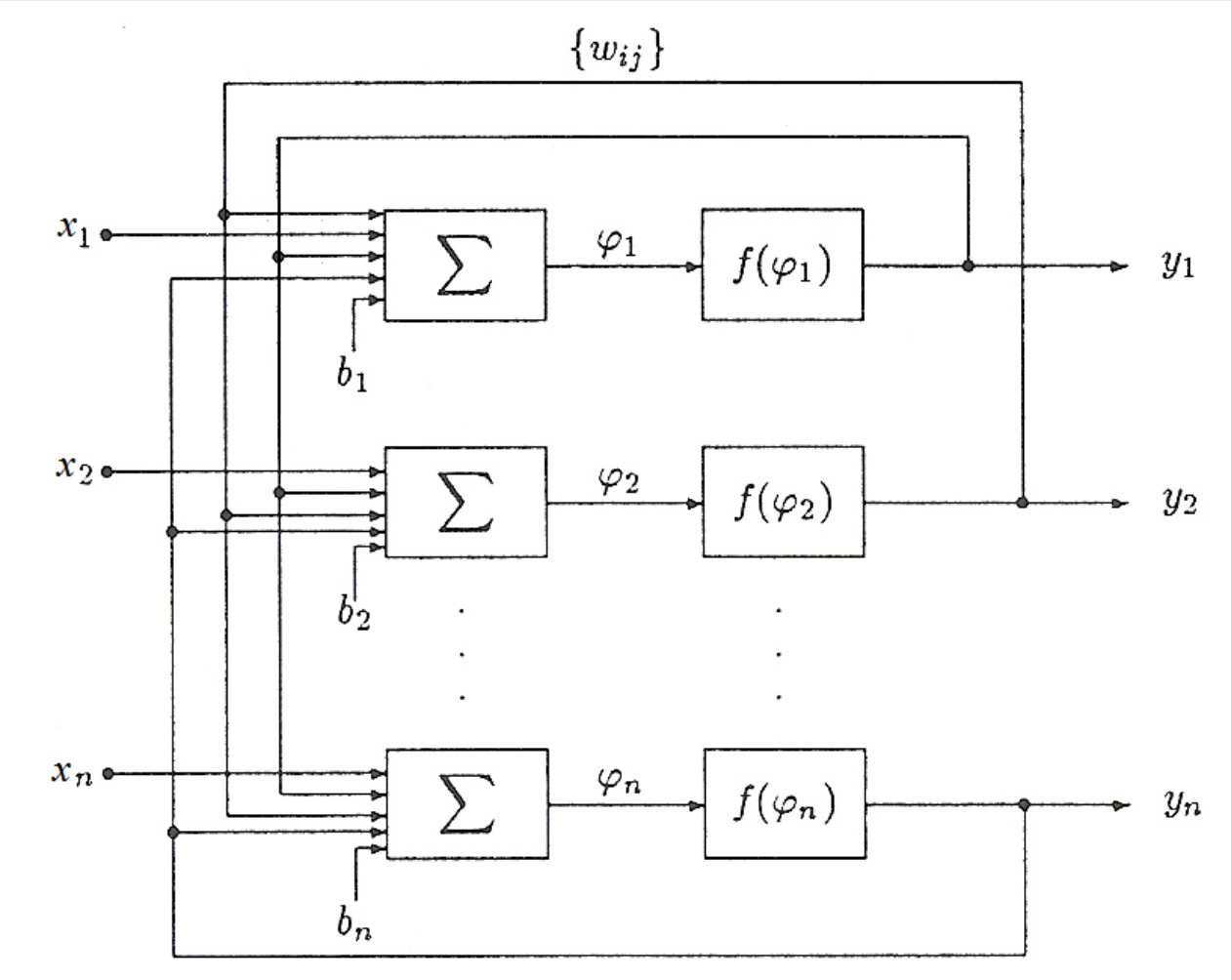
$$f(\phi) = \begin{cases} -1, & \text{for} \quad \phi < -1 \\ \phi, & \text{for} \quad -1 \leq \phi < 1 \\ 1, & \text{for} \quad \phi \geq 1 \end{cases}$$

Types of neural networks



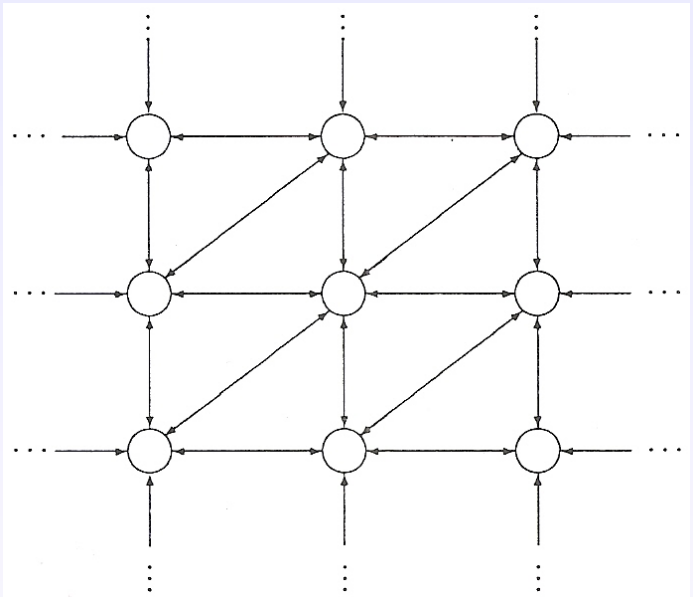
Multilayer, feedforward neural network.

Types of neural networks



Recurrent neural network.

Types of neural networks



Cellular neural network.

Applications

Neural networks can perform the following tasks:

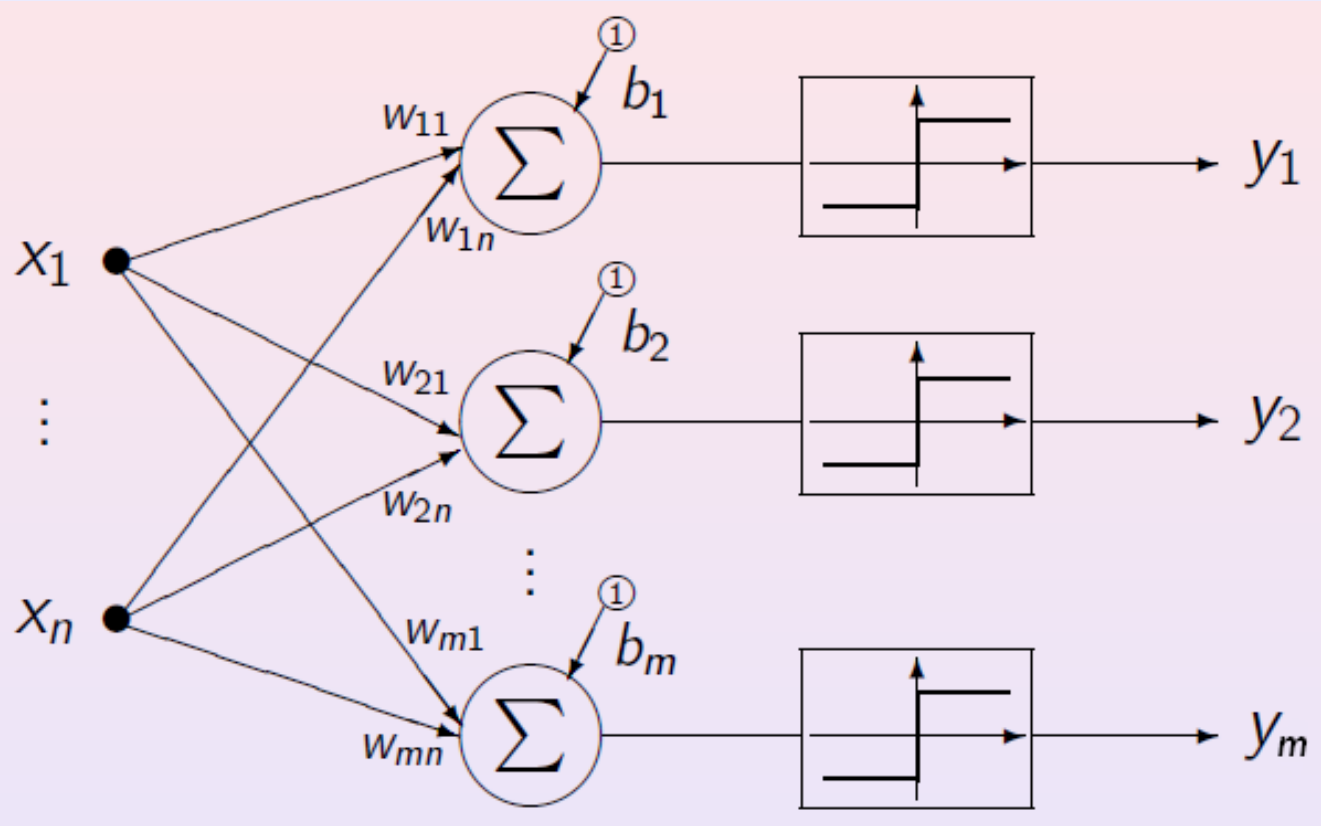
1. classification,
2. approximation (modelling),
3. prediction,
4. signals filtration,
5. data analysis (e.g. clustering, PCA),
6. optimisation,
7. other (e.g. data/image compression).

Perceptron

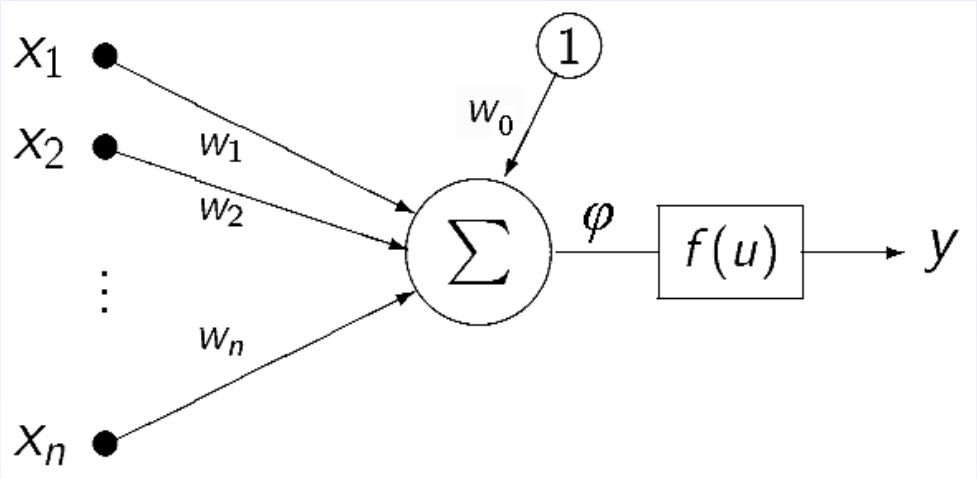
General information

- A single layer of neurons
- Unipolar of bipolar threshold activation function
- Supervised learning
- Learning algorithm – delta rule
- Application – classification

Perceptron



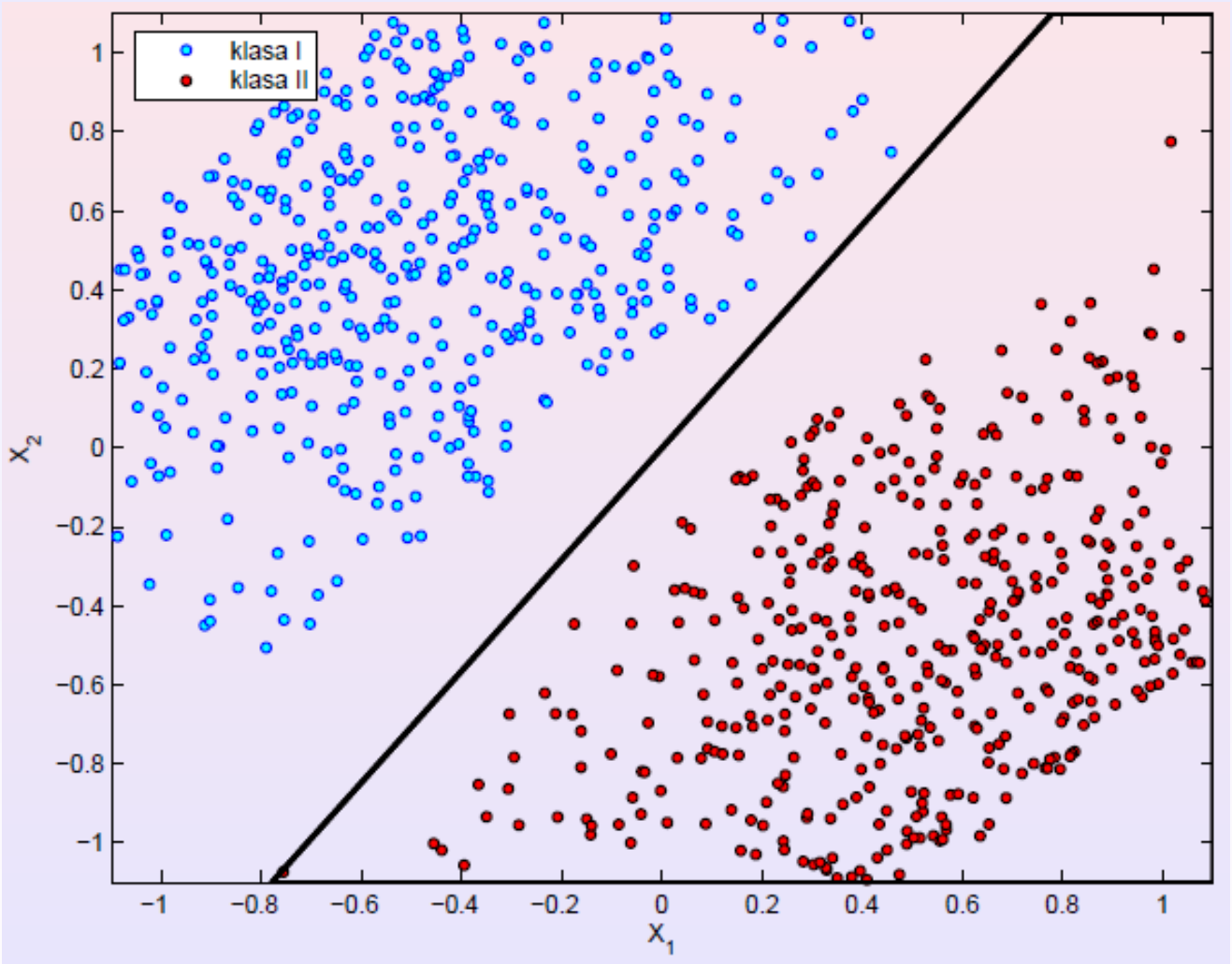
Perceptron



$$y(\mathbf{x}) = f(\sum_{i=0}^{n} w_i x_i) = f(\mathbf{w}^T \mathbf{x})$$

Perceptron

A single neuron separates the input space into two parts. Its basic task is a binary classification.



Perceptron

Binary classification

The aim of the classification is assigning an object to a class at the basis of its attributes values (input data). In the binary classification case, there are possible two classes at the output.

Let's consider a training set (a set of pairs):

$$(\mathbf{x}_i, d_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data and $d_i \in \{0, 1\}$ are given output data.

Perceptron

Hyper-plane equation in the space $\mathbb{R}^n$:

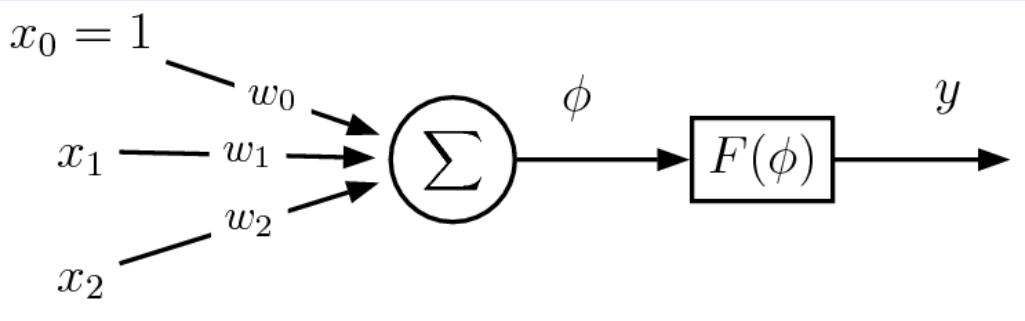
$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n = 0$$

$$\mathbf{w}^T \mathbf{x} = 0$$

Learning of the neuron is based on the selection of the weights in such a way that:

- $\mathbf{w}^T \mathbf{x} > 0$ is satisfied for samples with given output d equal 1,
- $\mathbf{w}^T \mathbf{x} \leq 0$ is satisfied for samples with given output d equal 0 or -1.

Perceptron



For example, for the neuron with two inputs:

$$\phi = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2$$

separating line is described by the equation:

$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0$$

or:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

Learning of the network

Purpose of learning

- The network is learnt on the base of learning data.
- Learning data – represent information about the desired network behavior.
- Purpose of learning – a selection of weights to realise the desired task.

Learning methods

Supervised learning

Learning data consist of pairs:

$$(\mathbf{x}_i, d_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data and $d_i \in \{0, 1\}$ or $d_i \in \mathbb{R}$ are connected with them, given output data.

Learning methods

Supervised learning

Learning data consist of pairs:

$$(\mathbf{x}_i, d_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data and $d_i \in \{0, 1\}$ or $d_i \in \mathbb{R}$ are connected with them, given output data.

Unsupervised learning

Learning data has a form of the set:

$$(\mathbf{x}_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data.

Delta rule

After presentation of the sample number p, we modify the weights according to the formula:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$$

where:

$$\delta_p = d_p - y_p = \begin{cases} -1 \\ 0 \\ 1 \end{cases}$$

d_p – given output for the sample no p
y_p – output of the neuron calculated for the sample no p
$\mathbf{x}_p$ – input for the sample no p
η – rate of learning
k – learning step

Learning algorithm

1. Initiation of weights
2. $n = 1$ (set the counter)
3. **while** $n > 0$ (check if all samples are classified correctly)
   - $n = 0$ (reset the counter)
   - Mix samples randomly in the training set
   - For successive samples:
     - Calculate output y_p of the neuron for sample p
     - Calculate error $\delta_p = d_p - y_p$
     - **if** $\delta_p != 0$

       $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$

       $n++$ (increment the counter)

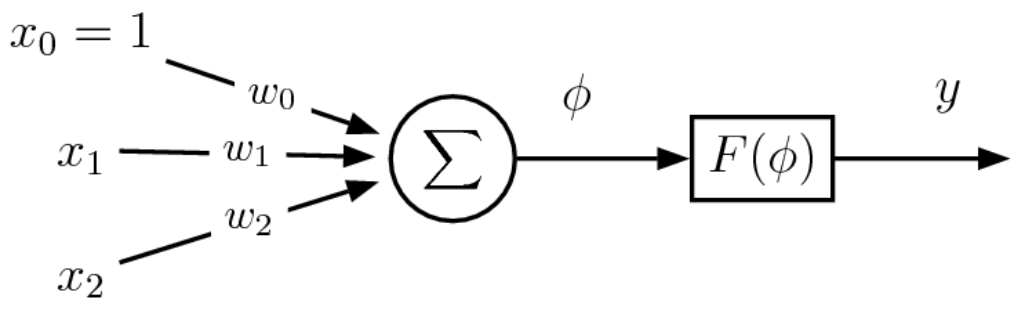
Features of the learning algorithm

- It always finishes its operation with a success if the set of patterns is linearly separable
- It doesn't stop when the patterns are not linearly separable.
- The margin of separation is always greater than or equal to zero.

In practice, we want the margin of separation to be the greatest.

Example

We have a neuron with a unipolar threshold (step) activation function.

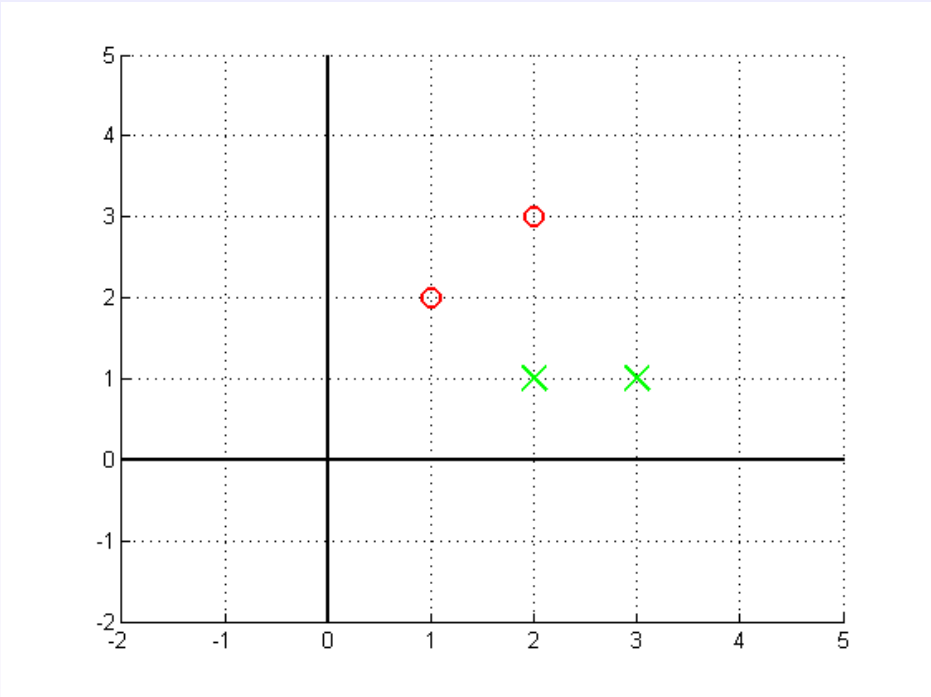


Using the delta rule, choose the weights of the neuron in order to properly classify learning samples shown in the table below.

x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

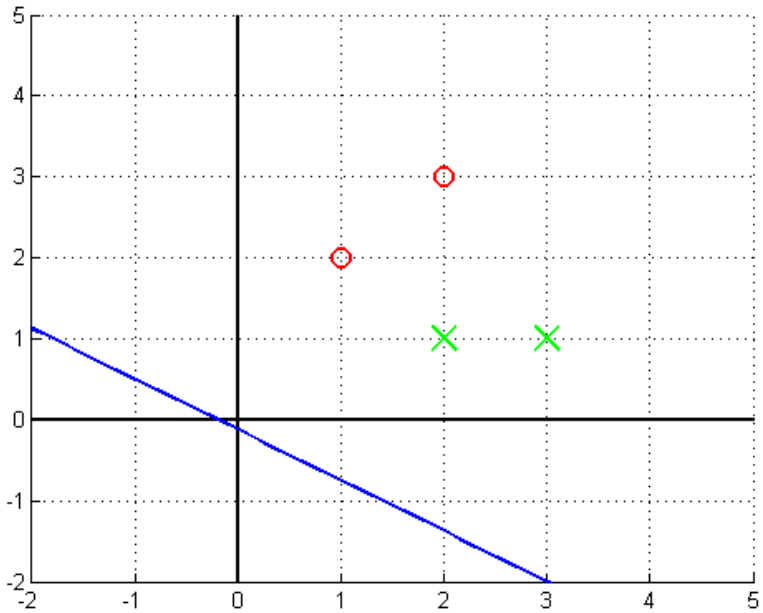
Rate of learning: $\eta = 0.2$.
Initial weights: $\mathbf{w}(0) = [w_0 \ w_1 \ w_2]^T = [0.1 \ 0.5 \ 0.8]^T$.

Example



x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

Example



x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

Weights:

$$\mathbf{w}(0) = [w_0\ w_1\ w_2]^T = [0.1\ 0.5\ 0.8]^T$$

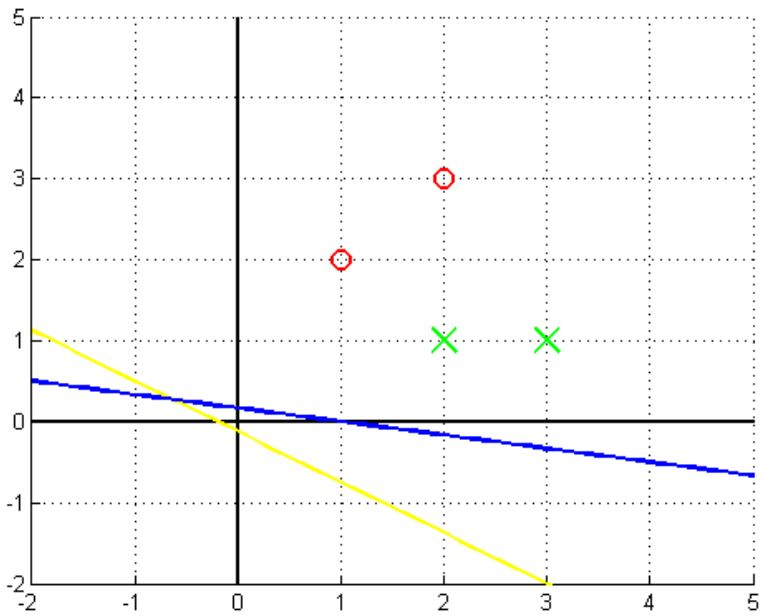
Separating line:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

or:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} = -\frac{5}{8} x_1 - \frac{1}{8}$$

Example



x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

Weights:

$$\mathbf{w} = [w_0\ w_1\ w_2]^T = [-0.1\ 0.1\ 0.6]^T$$

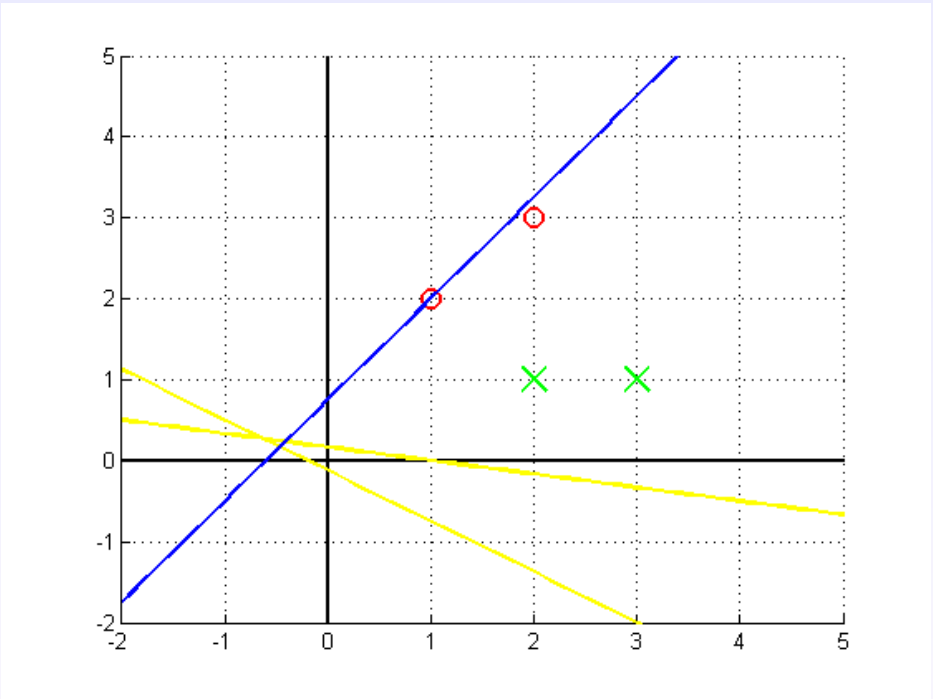
Separating line:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

or:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} = -\frac{1}{6} x_1 + \frac{1}{6}$$

Example



x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

Weights:

$$\mathbf{w} = [w_0\ w_1\ w_2]^T = [-0.3\ -0.5\ 0.4]^T$$

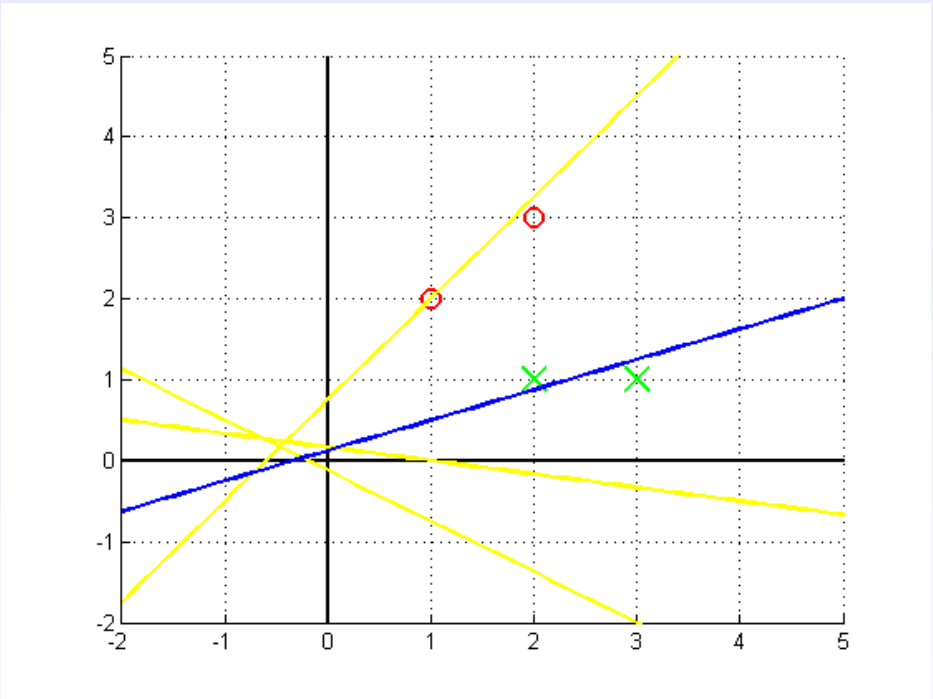
Separating line:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

or:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} = \frac{5}{4} x_1 + \frac{3}{4}$$

Example



x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

Weights:

$$\mathbf{w} = [w_0\ w_1\ w_2]^T = [-0.1\ -0.3\ 0.8]^T$$

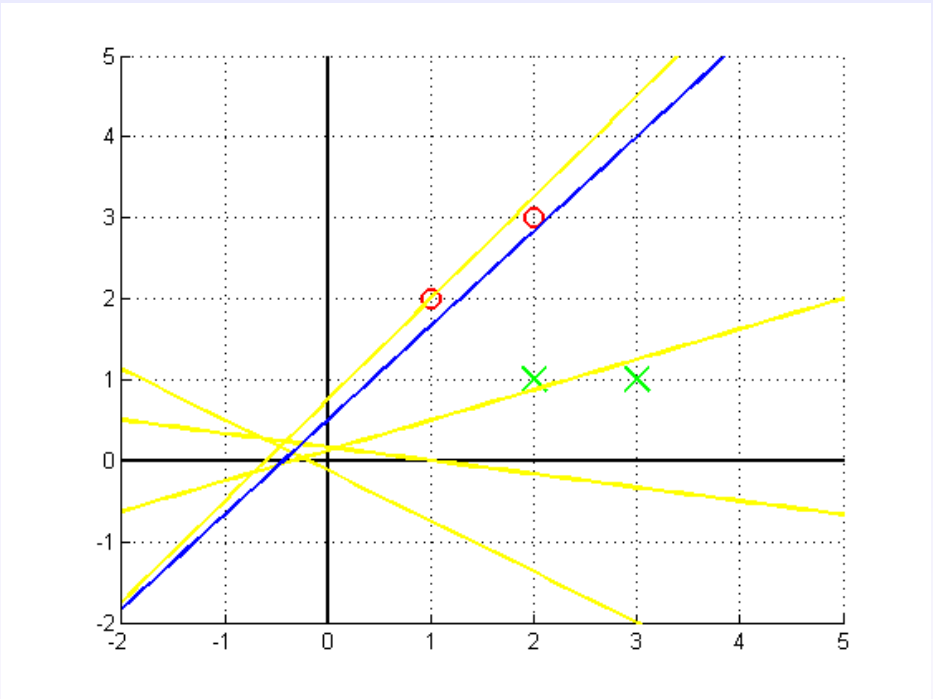
Separating line:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

or:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} = \frac{3}{8} x_1 + \frac{1}{8}$$

Example



x_1	x_2	d
1	2	1
2	3	1
2	1	0
3	1	0

Weights:

$$\mathbf{w} = [w_0\ w_1\ w_2]^T = [-0.3\ -0.7\ 0.6]^T$$

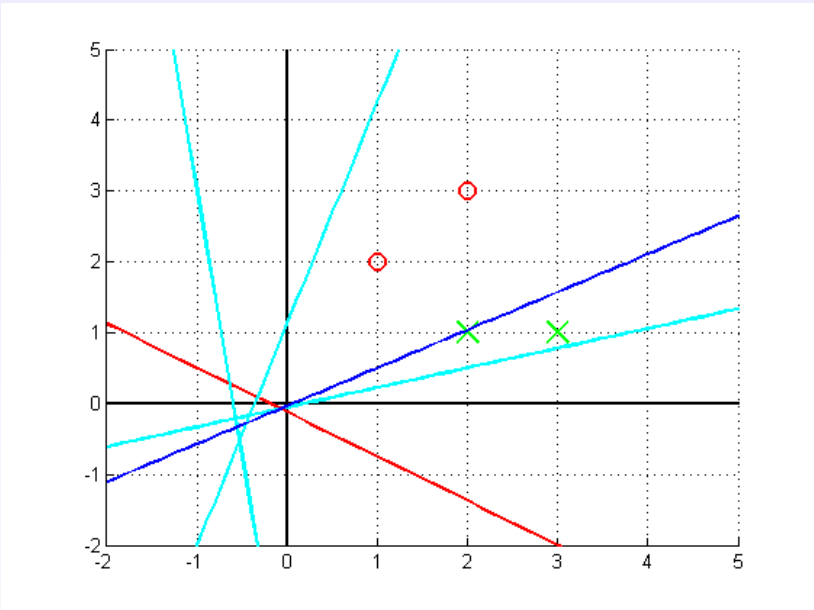
Separating line:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

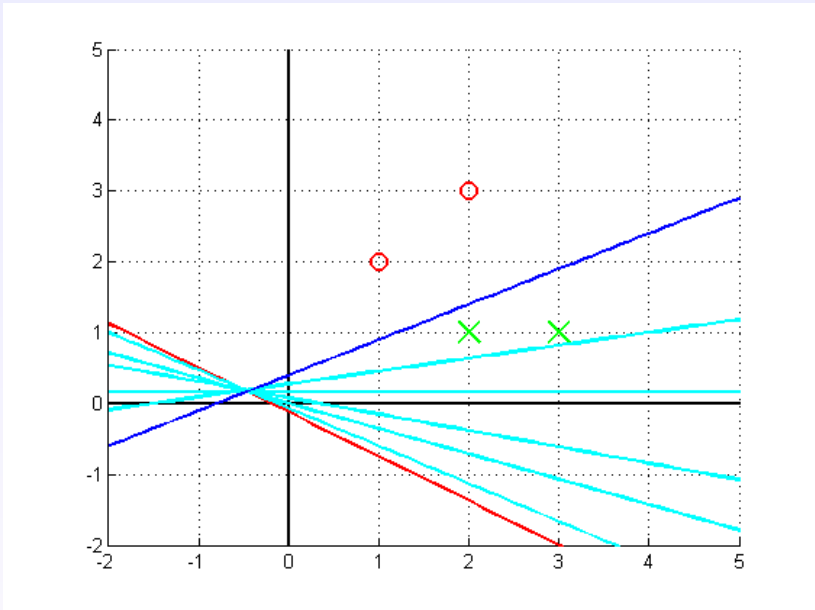
or:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} = \frac{7}{6} x_1 + \frac{3}{6}$$

Example – learning with different η value

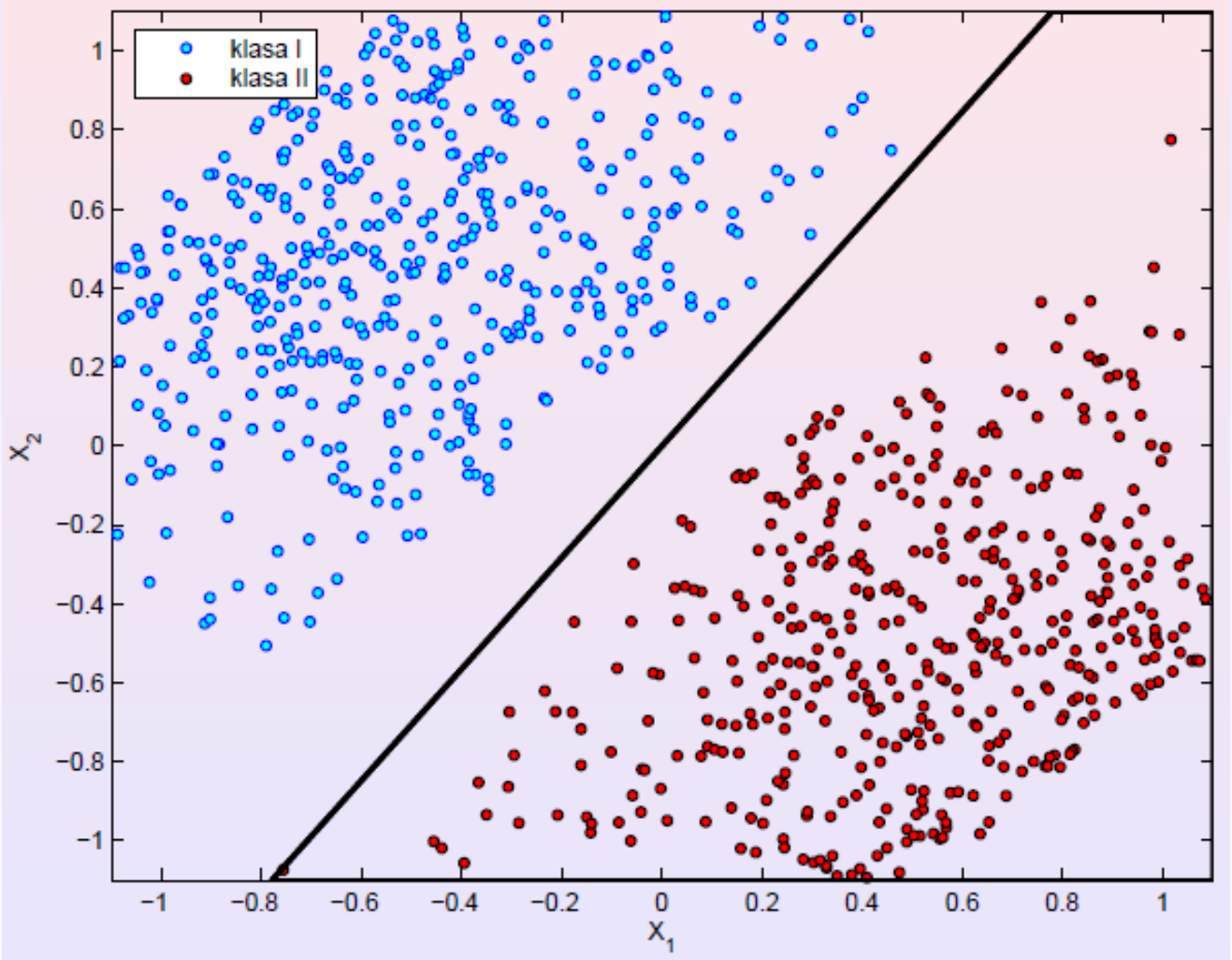


$\eta = 1$

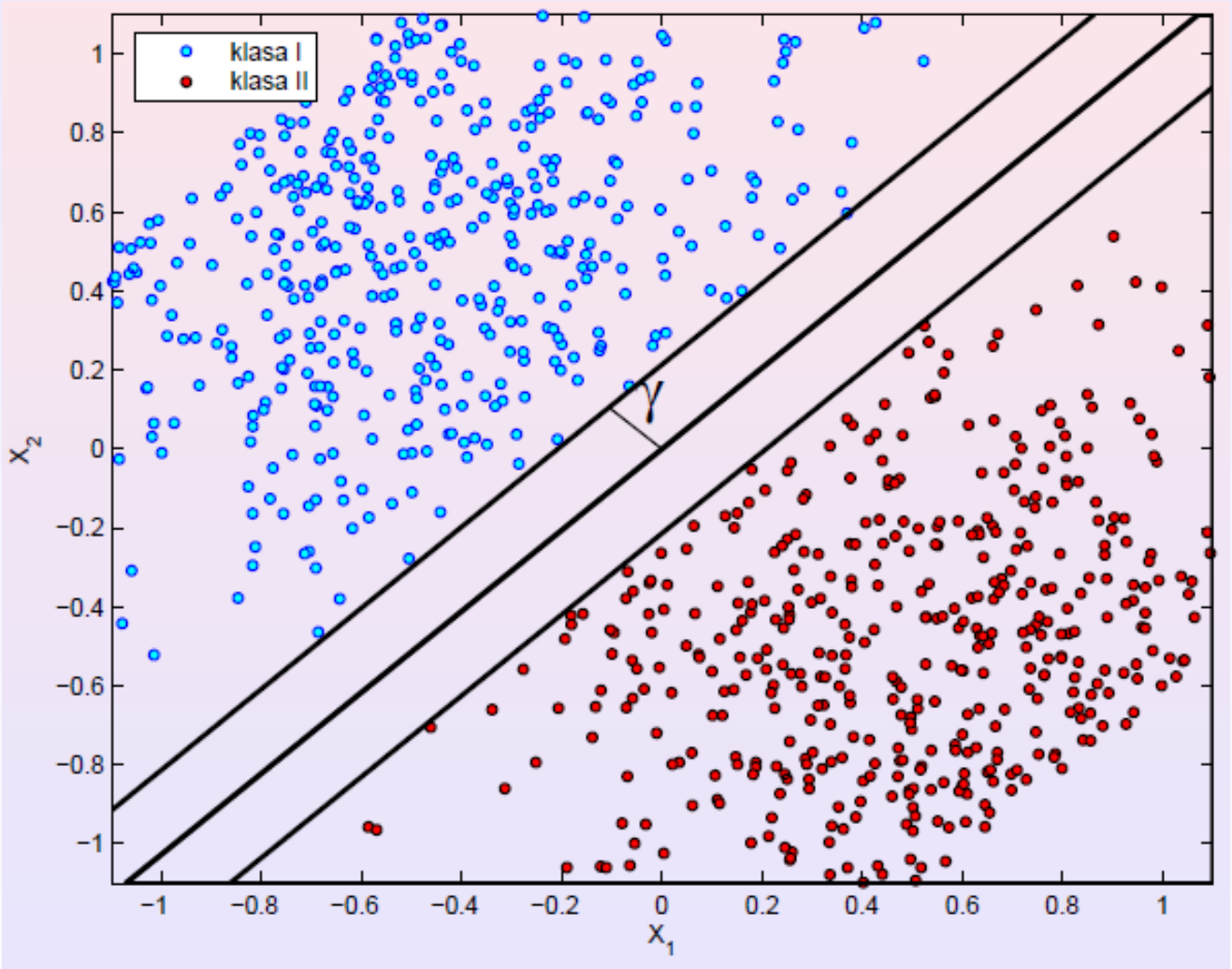


$\eta = 0.05$

Learning with a given separation margin γ



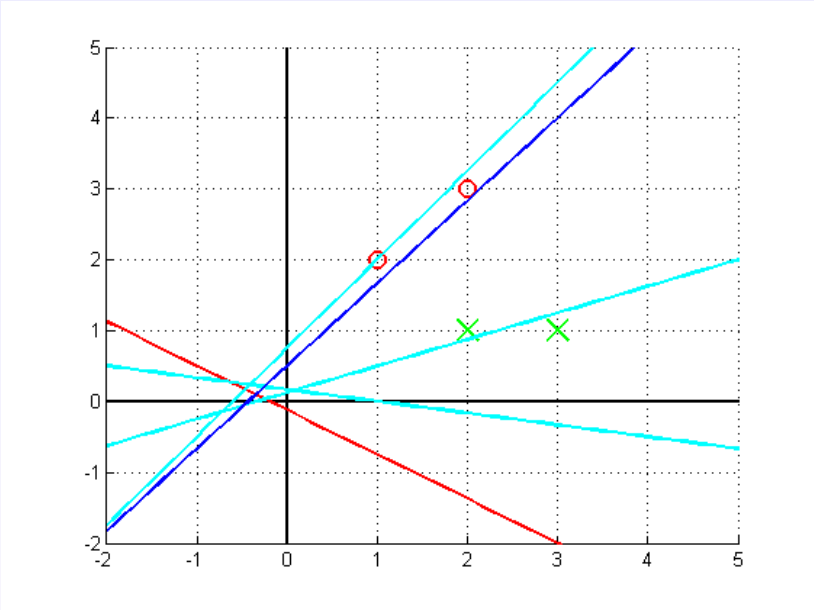
Learning with a given separation margin γ



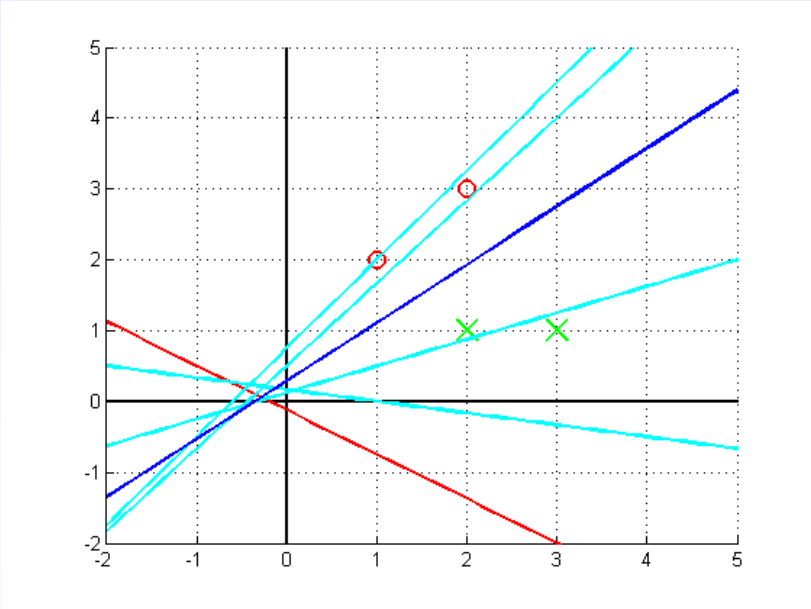
Learning algorithm with a given separation margin

1. Initiation of weights
2. $n = 1$ (set the counter)
3. **while** $n > 0$ (check if all samples are classified correctly)
   - $n = 0$ (reset the counter)
   - Mix samples randomly in the training set
   - For successive samples:
     - Calculate output y_p of the neuron for sample p
     - Calculate error $\delta_p = d_p - y_p$
     - Calculate a distance between sample and separation line $l_p = \frac{\phi_p}{||w||}$
     - **if** $\delta_p != 0$
       $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$
       $n++$ (increment the counter)
     - **elseif** $\mathrm{abs}(l_p) < \gamma_{min} - tol$ (tol – small value, e.g. 0.01)
       $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathrm{sgn}(l_p) \cdot \gamma_{min} - l_p) \cdot \mathbf{x}_p$
       $n++$ (increment the counter)

Example – learning with a given separation margin γ

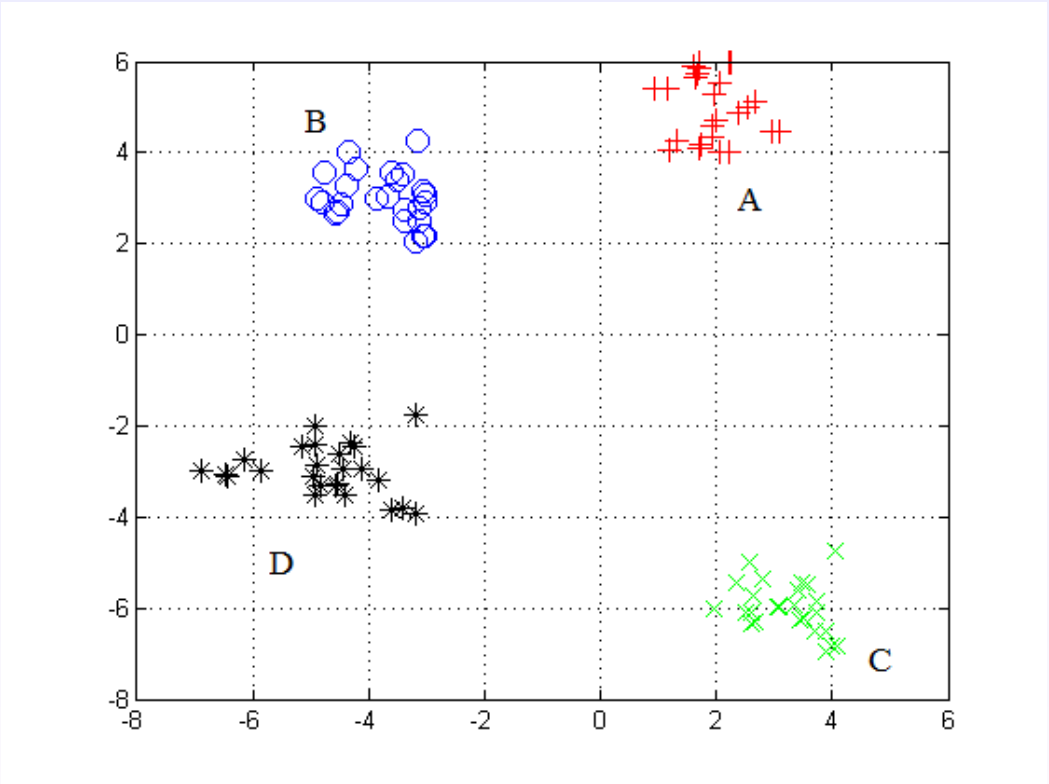


Learning without given separation margin



$\gamma_{min} = 0.6$

Example – multi-value classification

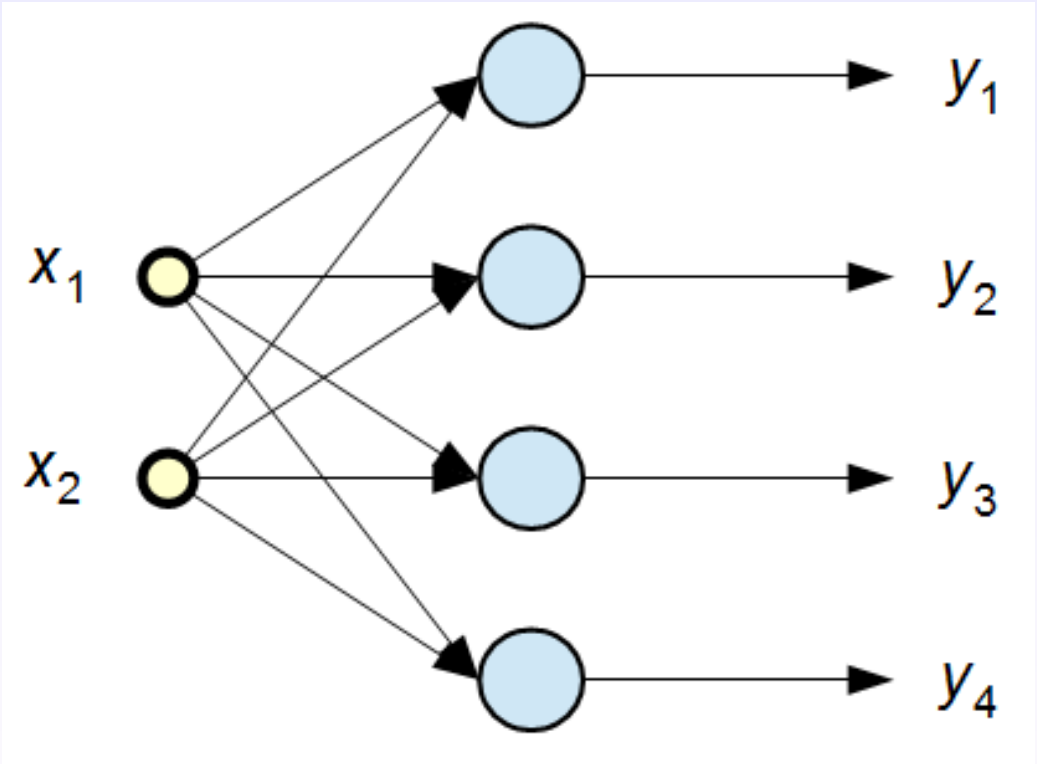


Samples belong to 4 classes:

2.04	5.54	A
1.92	4.58	A
⋮		
−3.67	3.04	B
−3.02	3.05	B
⋮		
3.08	−5.95	C
3.34	−5.94	C
⋮		
−4.13	−2.95	D
−4.44	−2.94	D
⋮		

Example – multi-value classification

We create one layer network with 4 neurons.



Example – multi-value classification

We can code the given output in the following way:

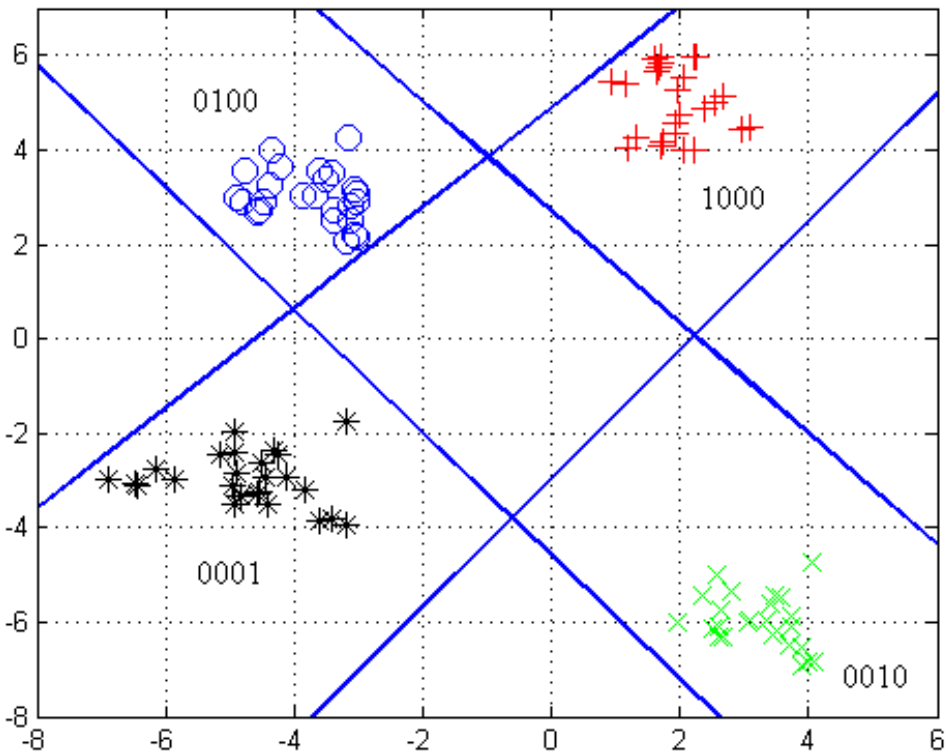
$$\begin{aligned} A &\rightarrow 1000 \\ B &\rightarrow 0100 \\ C &\rightarrow 0010 \\ D &\rightarrow 0001 \end{aligned}$$

Data after coding:

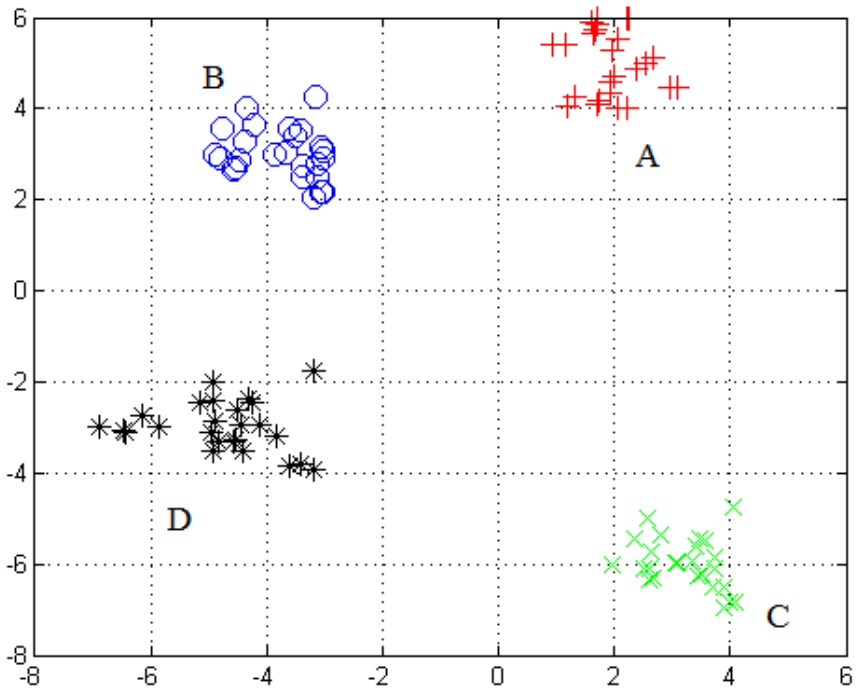
x_1	x_2	Class	d_1	d_2	d_3	d_4
2.04	5.54	A	1	0	0	0
1.92	4.58	A	1	0	0	0
⋮						
−3.67	3.04	B	0	1	0	0
−3.02	3.05	B	0	1	0	0
⋮						
3.08	−5.95	C	0	0	1	0
3.34	−5.94	C	0	0	1	0
⋮						
−4.13	−2.95	D	0	0	0	1
−4.44	−2.94	D	0	0	0	1
⋮						

Example – multi-value classification

Each neuron can be learnt separately with given output: d_1, d_2, d_3 and d_4. After learning (with the given separation margin):



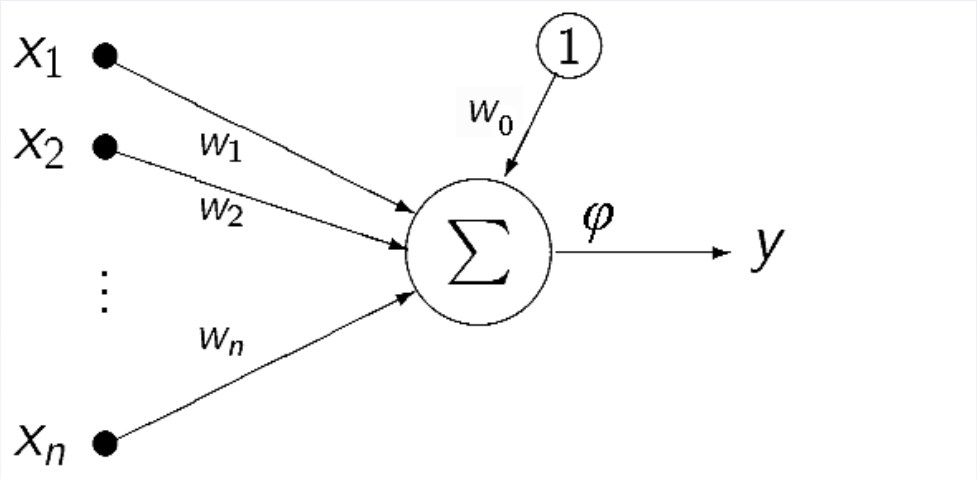
Example – multi-value classification



Is it possible to create the network with only 2 outputs and coding below?

$$\begin{aligned} A &\rightarrow 00 \\ B &\rightarrow 10 \\ C &\rightarrow 11 \\ D &\rightarrow 01 \end{aligned}$$

ADALINE neuron (Adaptive Linear Neuron)



$$y(\mathbf{x}) = \phi = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}$$

Delta rule

After presentation of the sample number p, we modify the weights according to the formula:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$$

where:

$$\delta_p = d_p - y_p$$

d_p – given output for the sample no p
y_p – output of the neuron calculated for the sample no p
$\mathbf{x}_p$ – input for the sample no p
η – rate of learning
k – learning step

Delta rule

After presentation of the sample number p, we modify the weights according to the formula:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$$

where:

$$\delta_p = d_p - y_p$$

d_p – given output for the sample no p
y_p – output of the neuron calculated for the sample no p
$\mathbf{x}_p$ – input for the sample no p
η – rate of learning
k – learning step

Cumulative correction

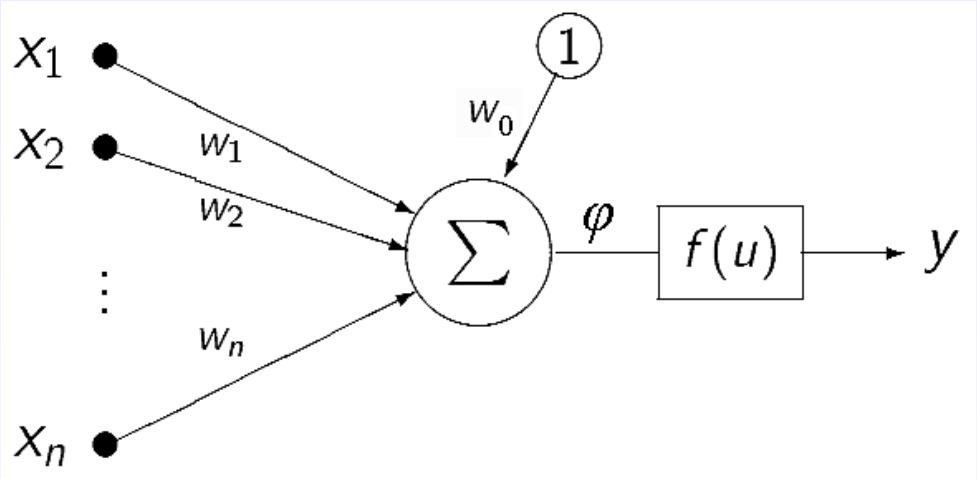
After presentation of all samples from the learning data set we can calculate:

$$\Delta\mathbf{w} = \frac{1}{L}\eta \sum_{p=1}^{L} \delta_p \cdot \mathbf{x}_p$$

Learning algorithm

1. Initiation of weights
2. $Q = \text{Real\_max\_value}$ (set large value of the error in the beginning)
3. **while** $Q > Q_{min}$ (check if the actual error is smaller than the given error)
   - $Q = 0$ (reset the error)
   - Mix samples randomly in the training set
   - For successive samples:
     - Calculate output y_p of the neuron for sample p
     - Calculate error $\delta_p = d_p - y_p$
     - Calculate new weights $\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot \mathbf{x}_p$
     - $Q = Q + \delta_p^2$
   - Calculate mean square error $Q = Q/L$ (L – number of samples)

Neuron with a nonlinear activation function



$$y(\mathbf{x}) = f(\sum_{i=0}^{n} w_i x_i) = f(\mathbf{w}^T \mathbf{x})$$

Generalized delta rule

After presentation of the sample number p, we modify the weights according to the formula:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot f'(\phi_p) \cdot \mathbf{x}_p$$

where:

$$\delta_p = d_p - y_p$$

d_p – given output for the sample no p
y_p – output of the neuron calculated for the sample no p
$\mathbf{x}_p$ – input for the sample no p
η – rate of learning
k – learning step

Generalized delta rule

After presentation of the sample number p, we modify the weights according to the formula:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot \delta_p \cdot f'(\phi_p) \cdot \mathbf{x}_p$$

where:

$$\delta_p = d_p - y_p$$

d_p – given output for the sample no p
y_p – output of the neuron calculated for the sample no p
$\mathbf{x}_p$ – input for the sample no p
η – rate of learning
k – learning step

Generalized error

$$\delta'_p = \delta_p \cdot f'(\phi_p)$$

Derivatives of activation functions

Derivative of the sigmoid activation function:

$$f(\phi) = \frac{1}{1 + e^{-\beta\phi}}$$

can be calculated with formula:

$$f'(\phi) = \beta \cdot f(\phi) \cdot (1 - f(\phi))$$

Derivatives of activation functions

Derivative of the sigmoid activation function:

$$f(\phi) = \frac{1}{1 + e^{-\beta\phi}}$$

can be calculated with formula:

$$f'(\phi) = \beta \cdot f(\phi) \cdot (1 - f(\phi))$$

Derivative of the hyperbolic tangent activation function:

$$f(\phi) = \tanh(\phi) = \frac{e^{\beta\phi} - e^{-\beta\phi}}{e^{\beta\phi} + e^{-\beta\phi}}$$

can be calculated with formula:

$$f'(\phi) = \beta \cdot (1 - f^2(\phi))$$

Generalized delta rule

- The algorithm is convergent to nearest local minimum of the error function $Q(\mathbf{w})$.
- The algorithm does not guarantee finding of the global minimum!
- There is necessity of learning with different, random initial weights.

Momentum component

After presentation of the sample number p, we modify the weights according to the formula:

$$\Delta \mathbf{w}(k+1) = \eta \cdot \delta_p \cdot f'(\phi_p) \cdot \mathbf{x}_p + \alpha \cdot \Delta \mathbf{w}(k)$$

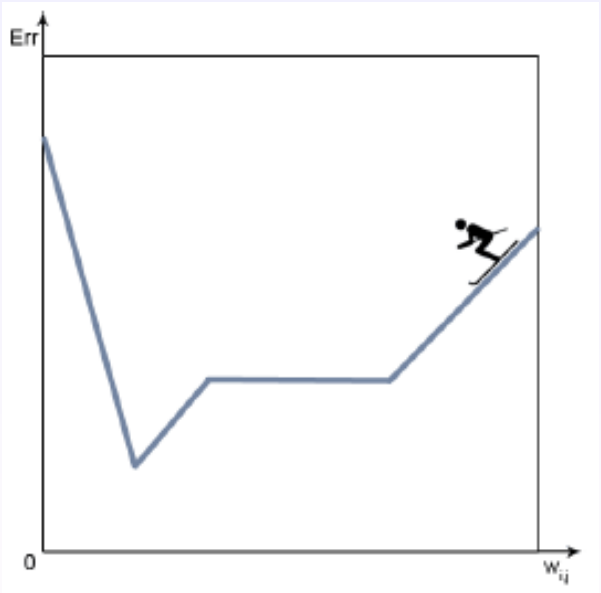
$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta \mathbf{w}(k+1)$$

where:
α – momentum coefficient,
k – learning step.

Momentum component

Momentum component:

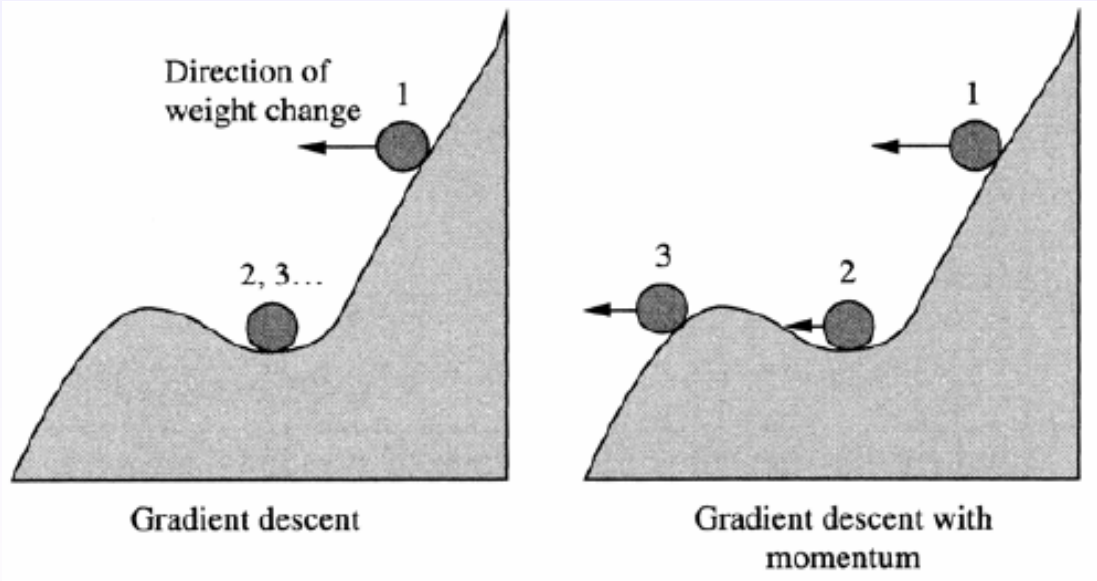
- accelerates the process of the network learning,
- allows the use of larger values of the rate of learning η,
- allows to skip local minima of the error,
- eliminates oscillations of weights during learning.



Momentum component

Momentum component:

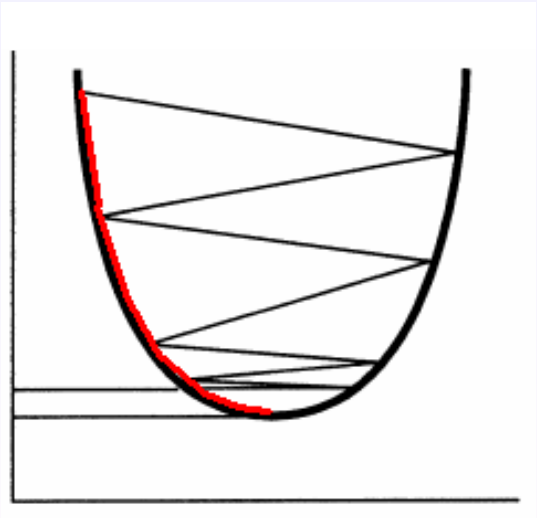
- accelerates the process of the network learning,
- allows the use of larger values of the rate of learning η,
- allows to skip local minima of the error,
- eliminates oscillations of weights during learning.



Momentum component

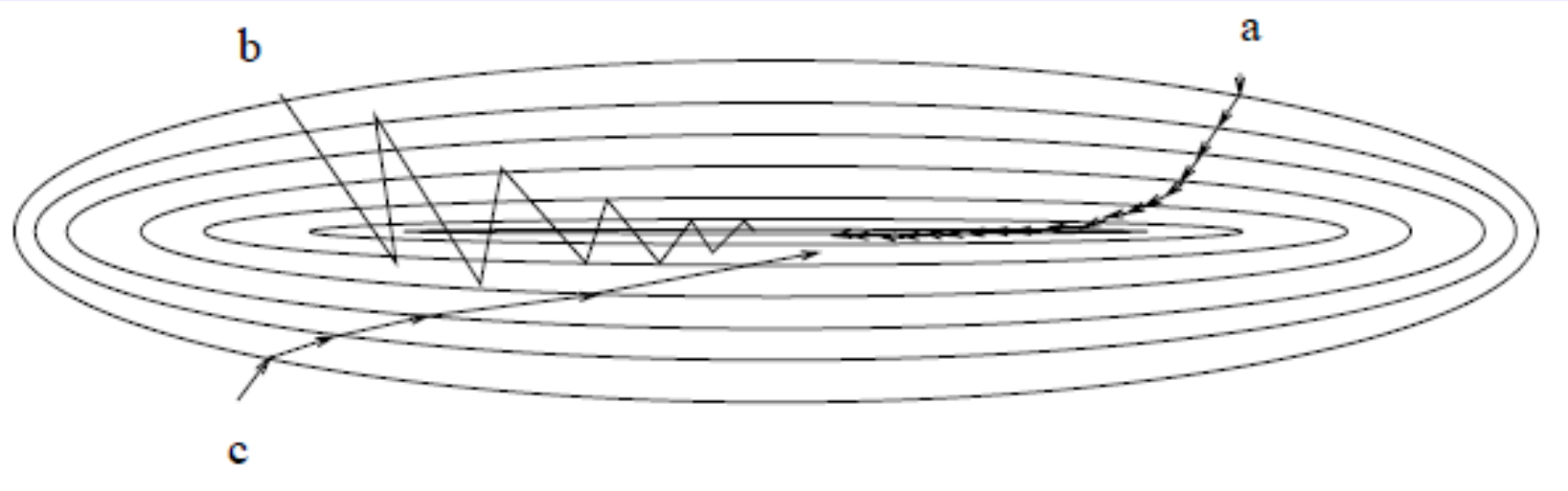
Momentum component:

- accelerates the process of the network learning,
- allows the use of larger values of the rate of learning η,
- allows to skip local minima of the error,
- eliminates oscillations of weights during learning.



Momentum component

An exemplary process of learning:

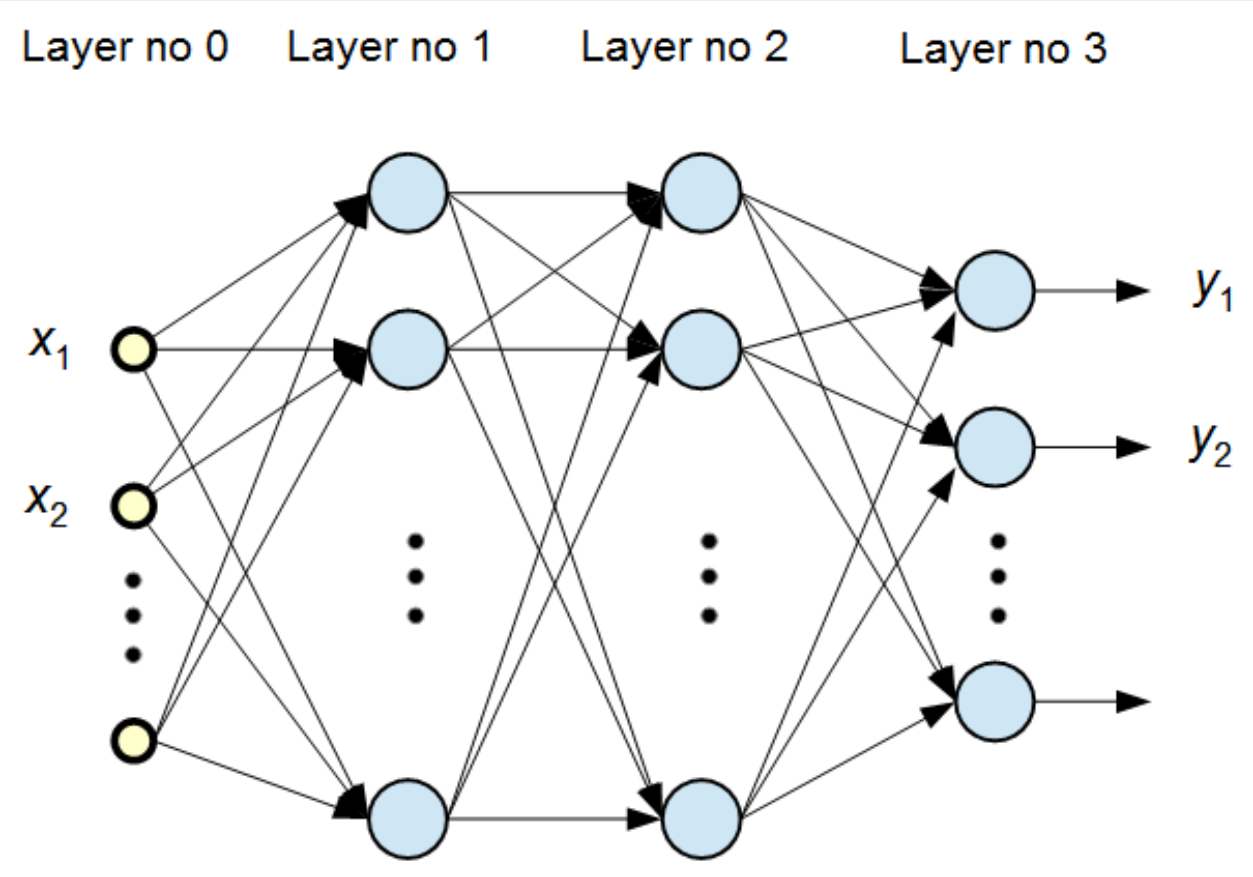


a) small η coefficient,

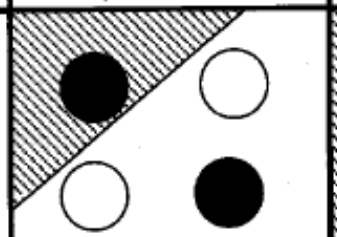
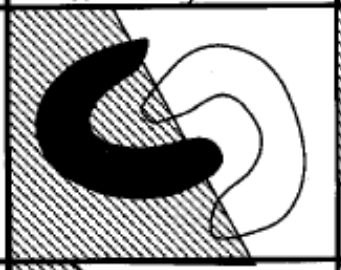
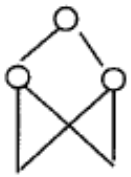
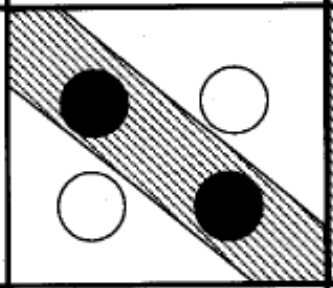
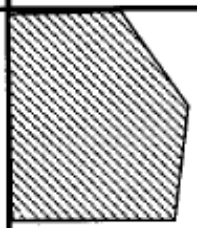
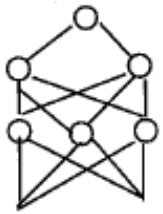
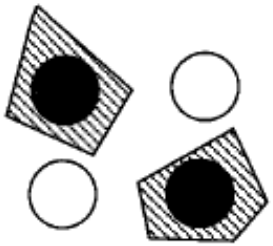
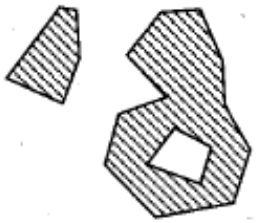
b) large η coefficient,

c) large η coefficient with momentum component.

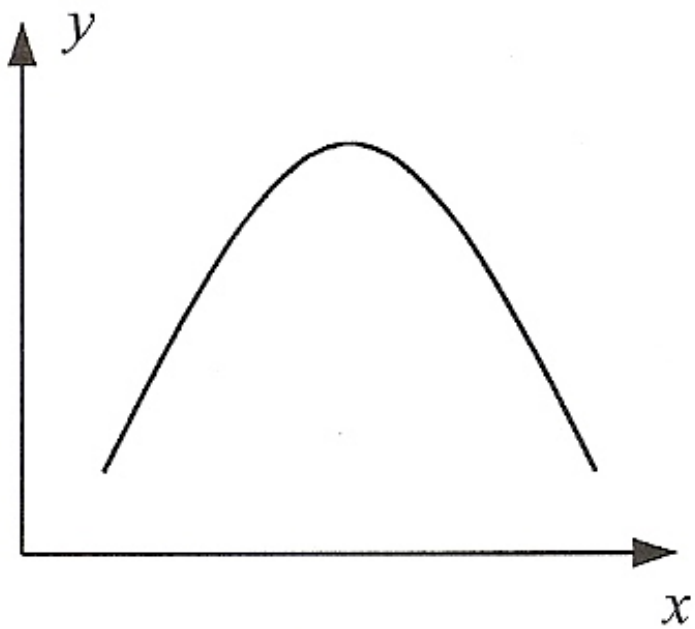
Multilayer, feedforward neural network



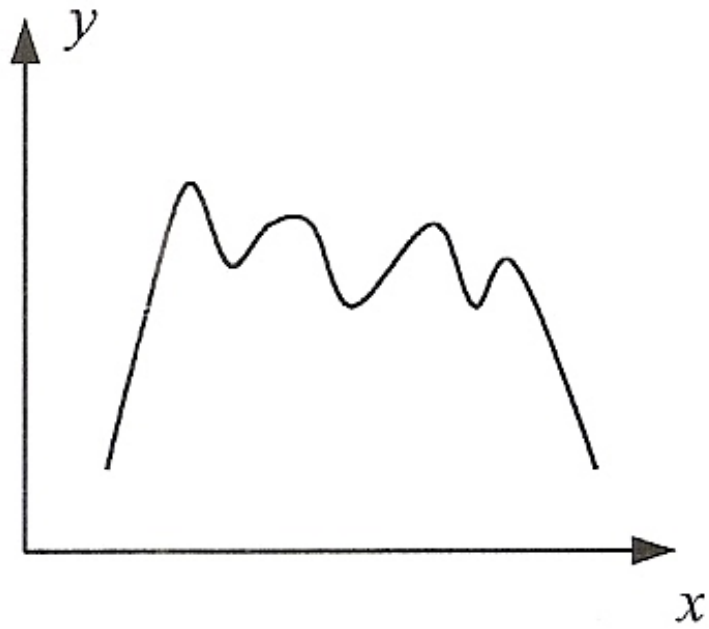
Choice of the network structure

Structure	Description of decision regions	Exclusive-OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

Choice of the network structure

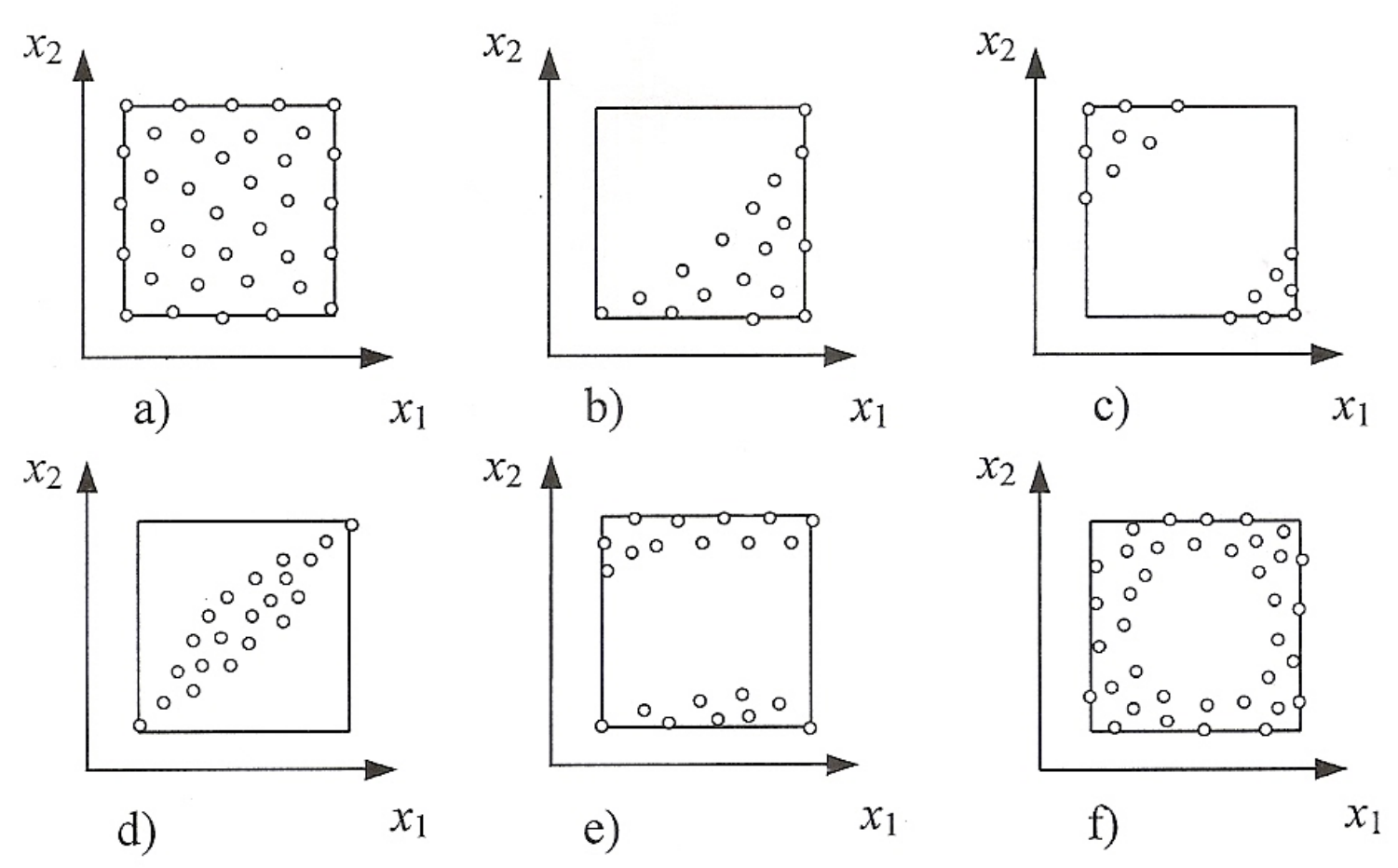


a) simple characterisitic: small number of neurons on the hidden layer

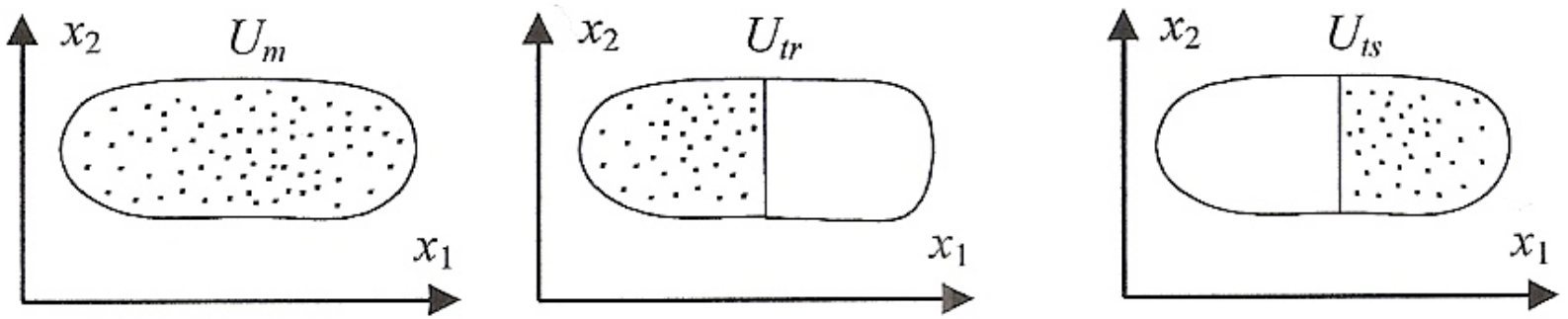


b) complicated characteristic: large number of neurons on the hidden layer

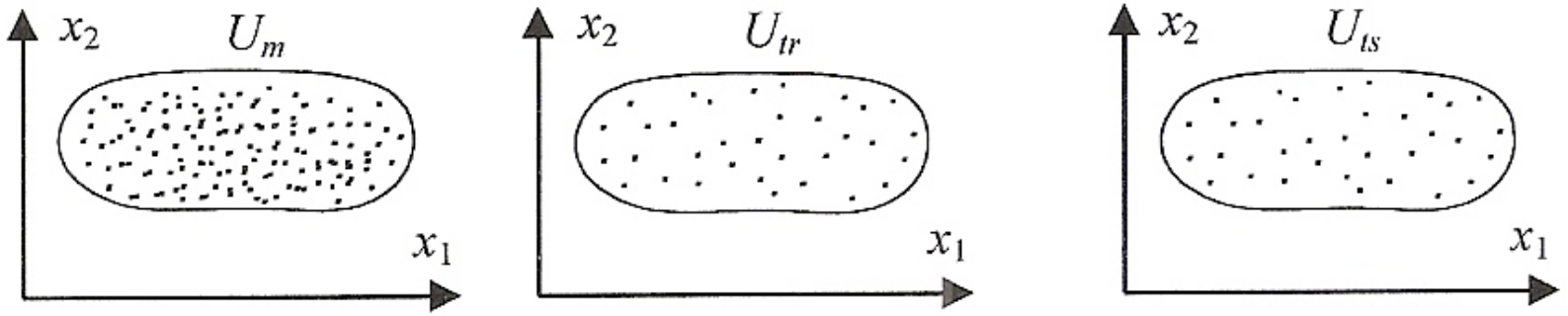
Learning data



Learning data

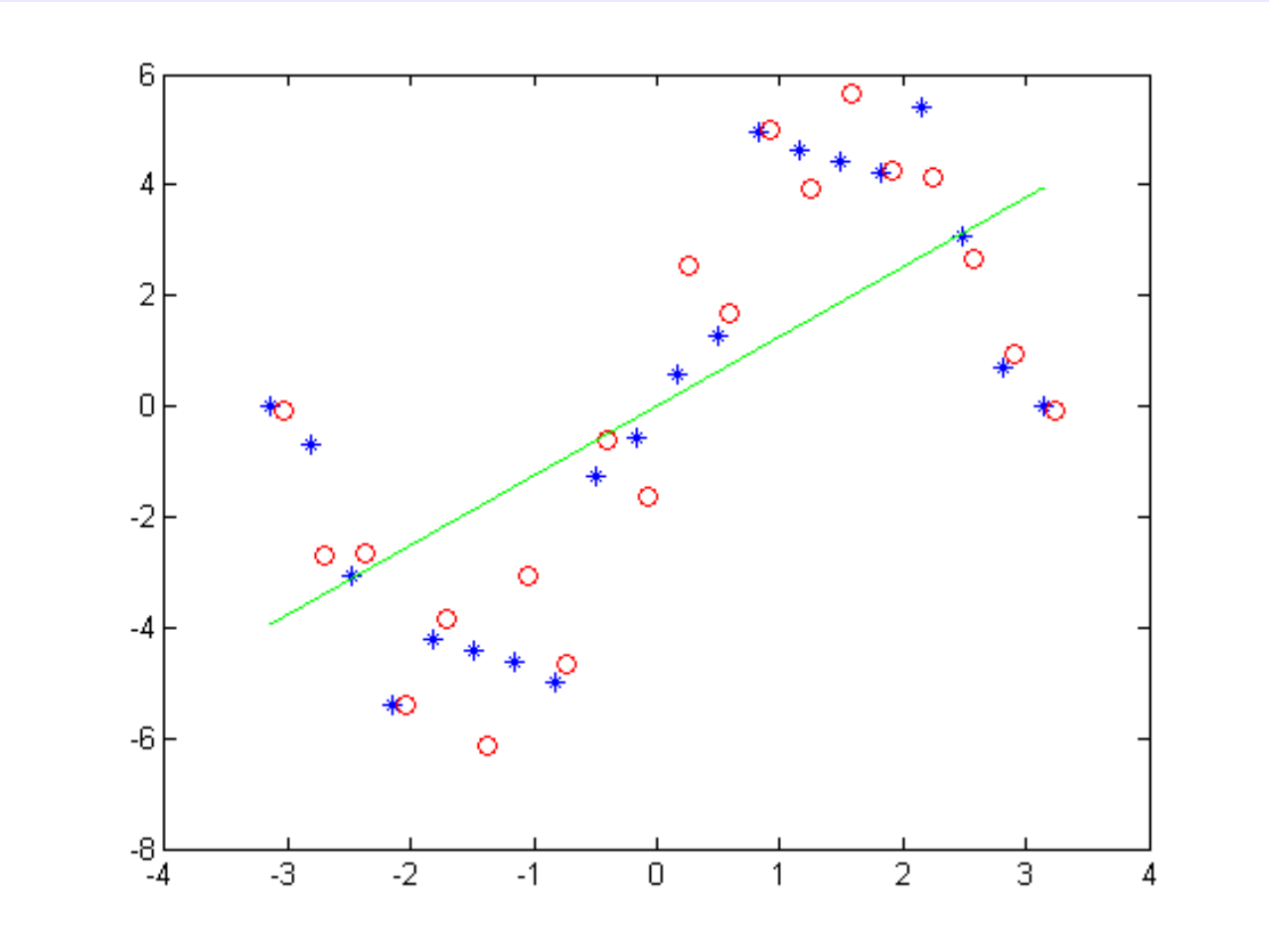


a) Wrong division of the data set into training and testing part



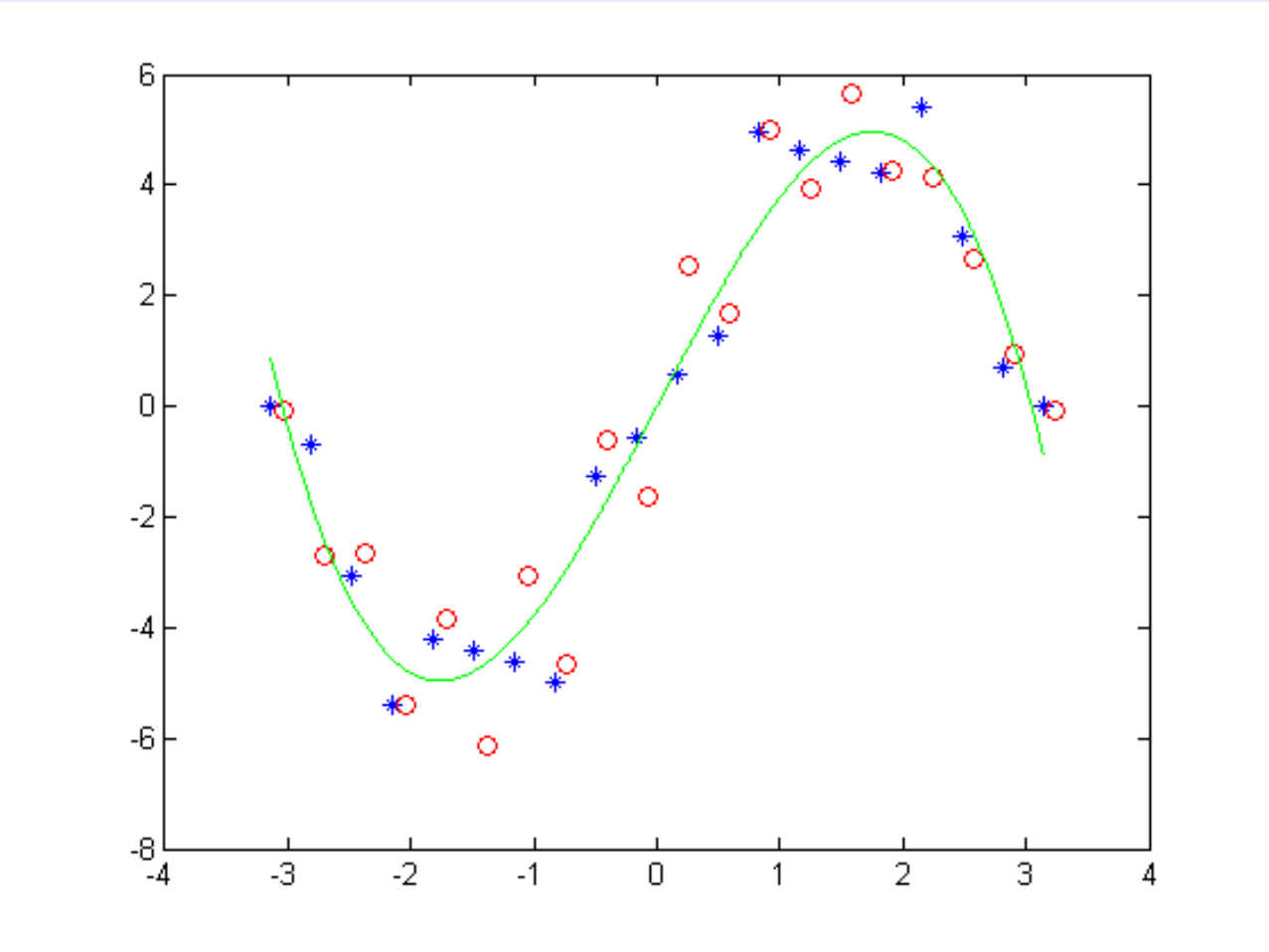
b) Correct division of the data set into training and testing part

Approximation with a polynomial of the 1-st order



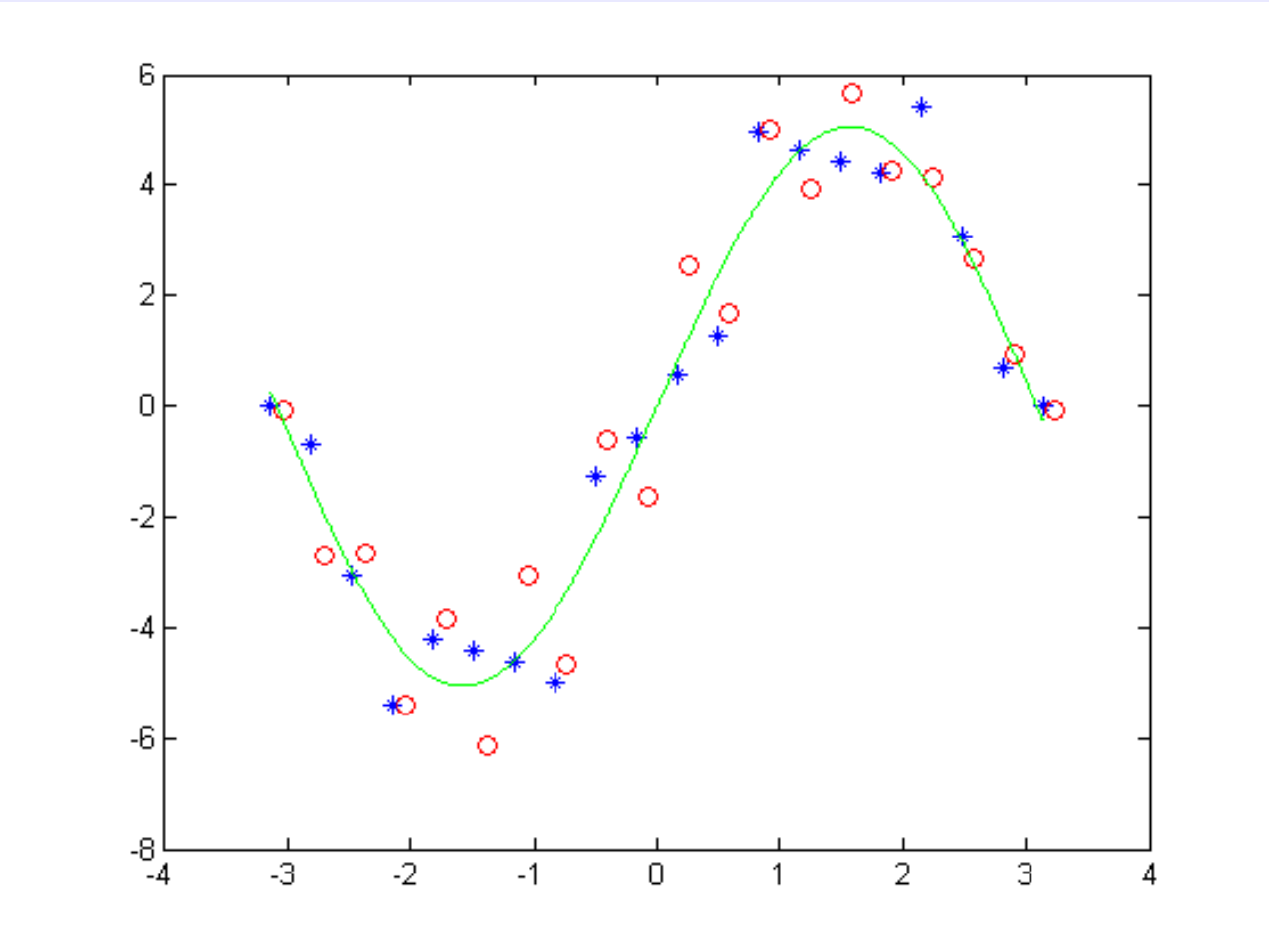
Learning data error (blue) = 44.30;
Testing data error (red) = 44.37

Approximation with a polynomial of the 3-rd order



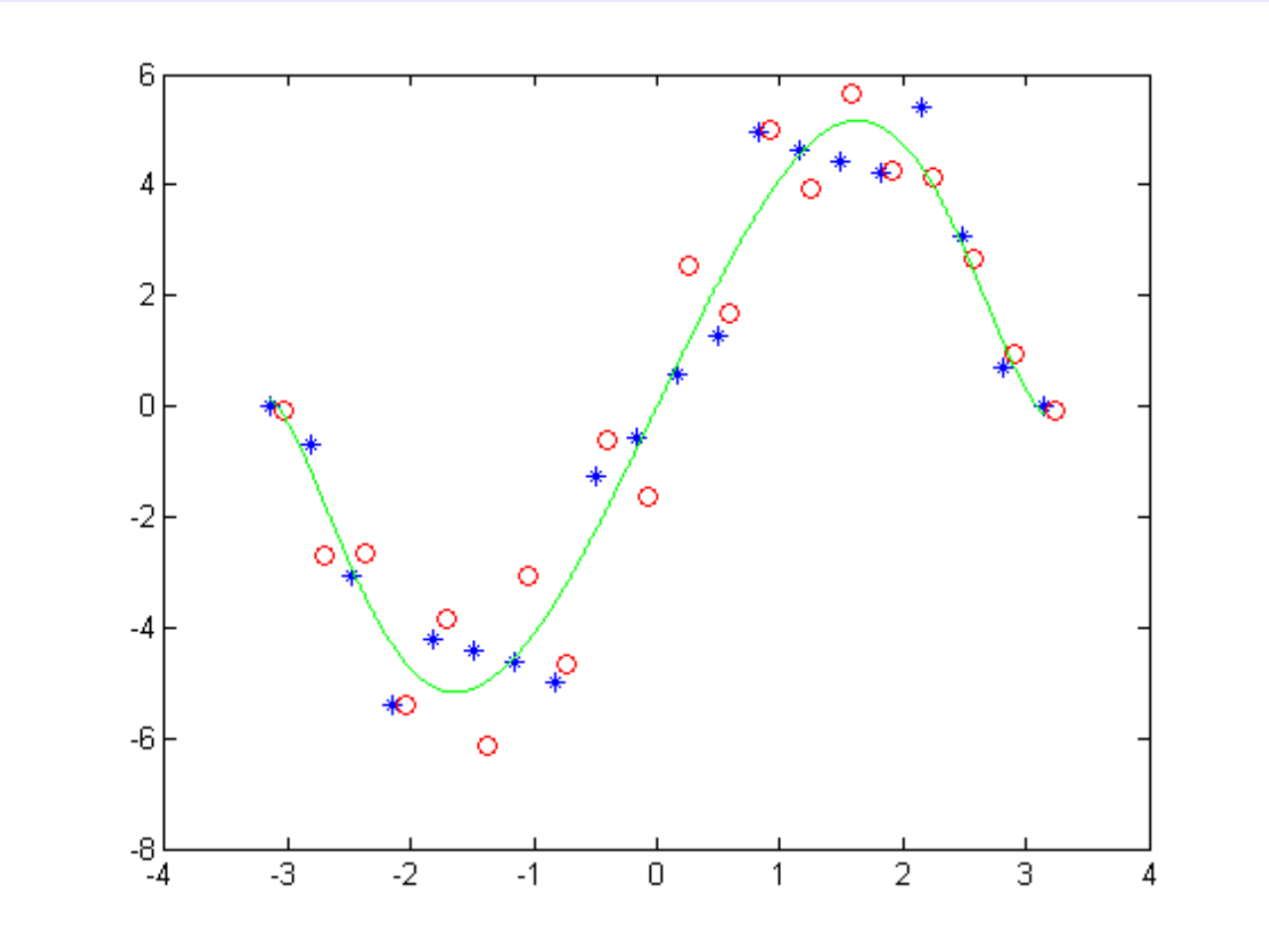
Learning data error (blue) = 14.69;
Testing data error (red) = 17.96

Approximation with a polynomial of the 5-th order



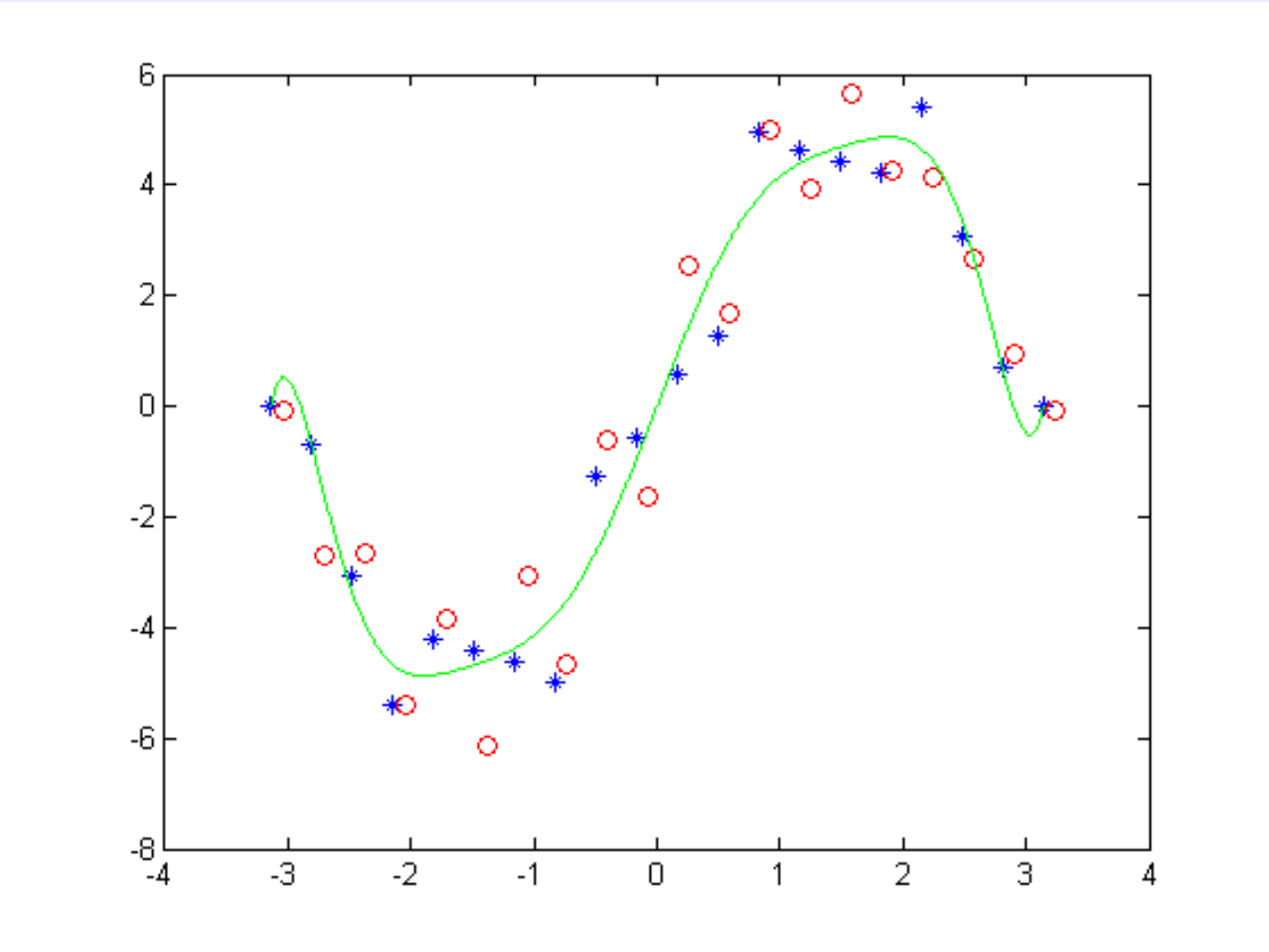
Learning data error (blue) = 12.37;
Testing data error (red) = 16.61

Approximation with a polynomial of the 7-th order



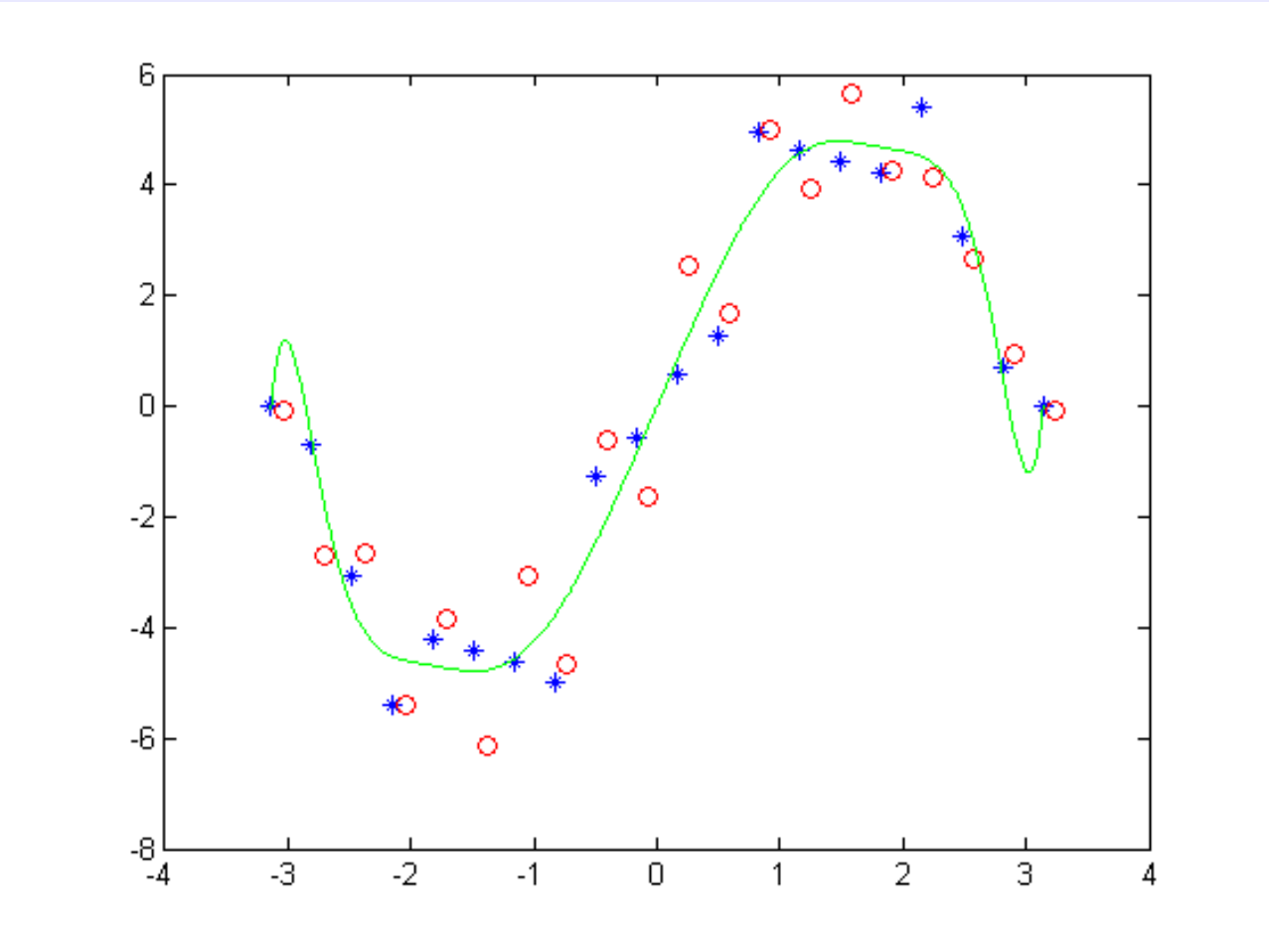
Learning data error (blue) = 11.99;
Testing data error (red) = 16.54

Approximation with a polynomial of the 9-th order



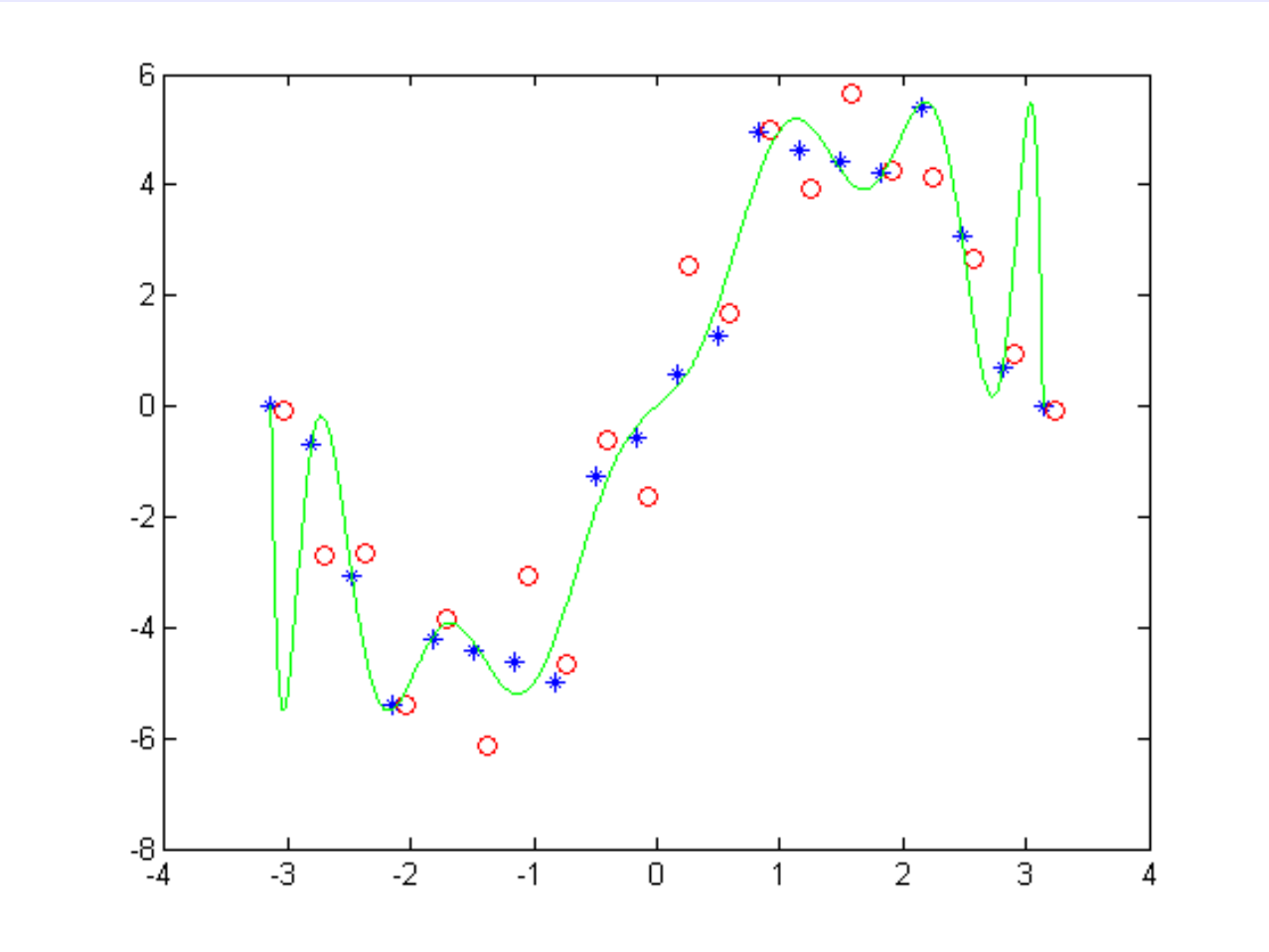
Learning data error (blue) = 10.21;
Testing data error (red) = 20.01

Approximation with a polynomial of the 11-th order



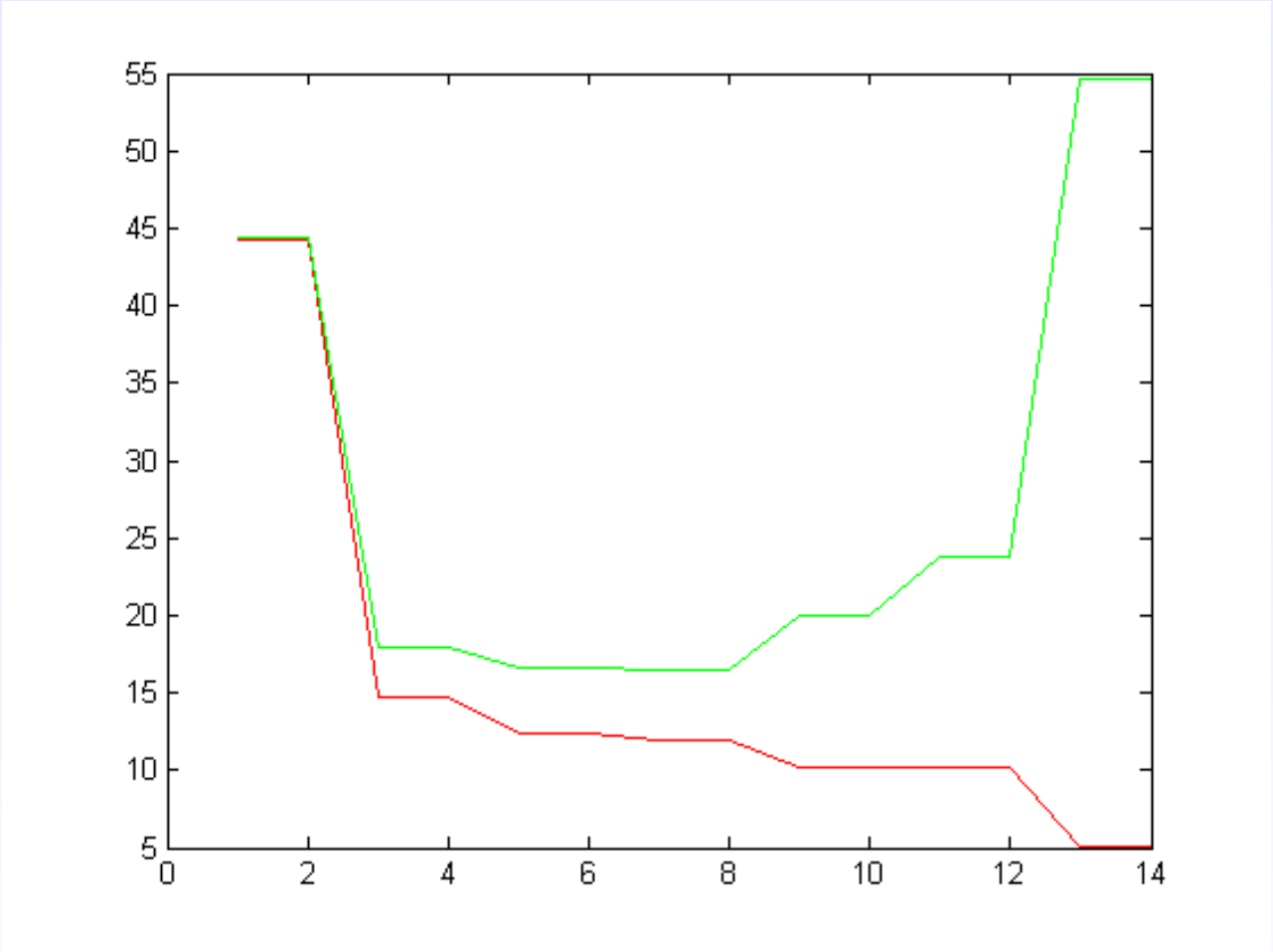
Learning data error (blue) = 10.19;
Testing data error (red) = 23.72

Approximation with a polynomial of the 13-th order



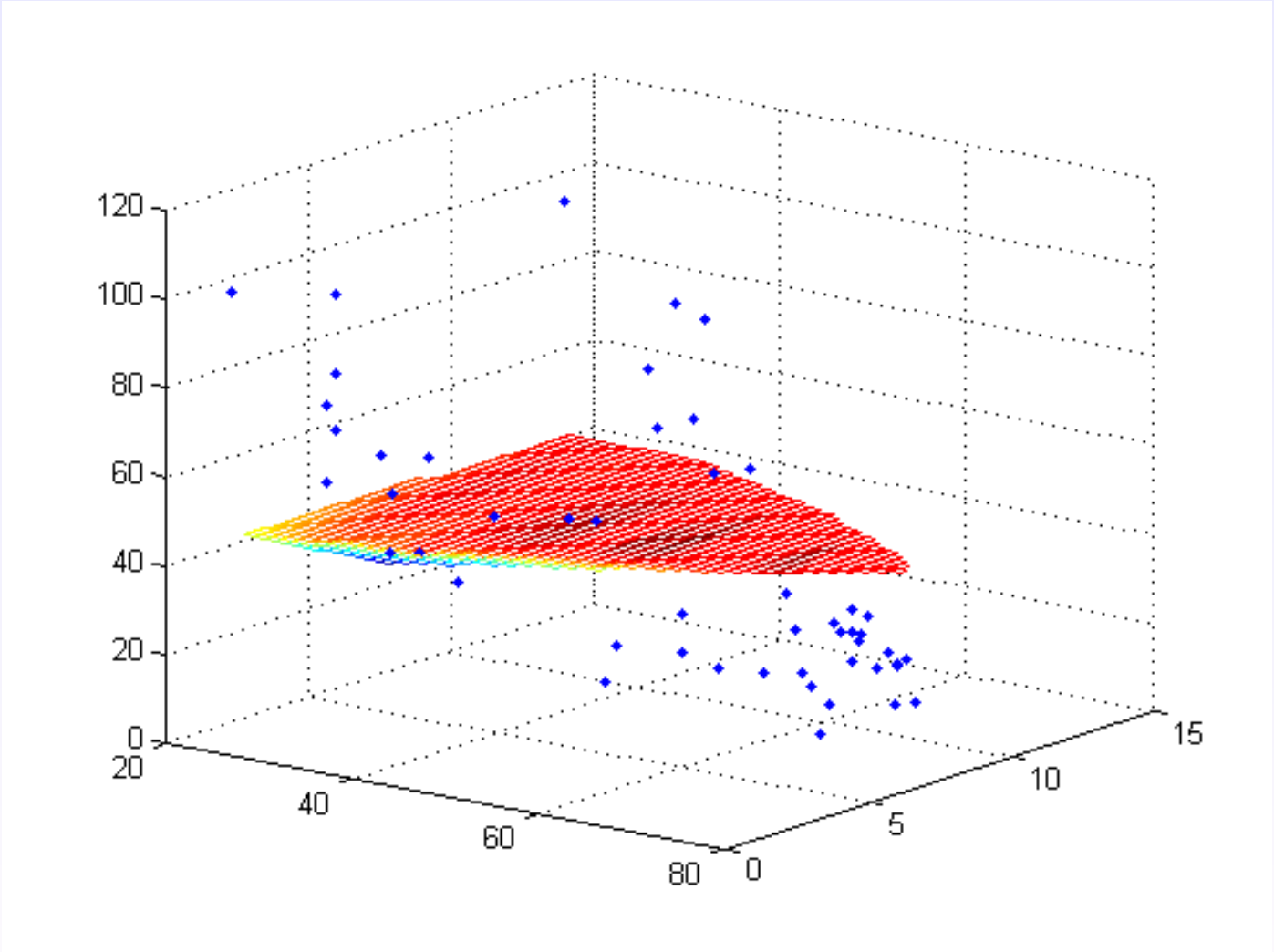
Learning data error (blue) = 5.02;
Testing data error (red) = 54.61

Overfitting and underfitting of the network



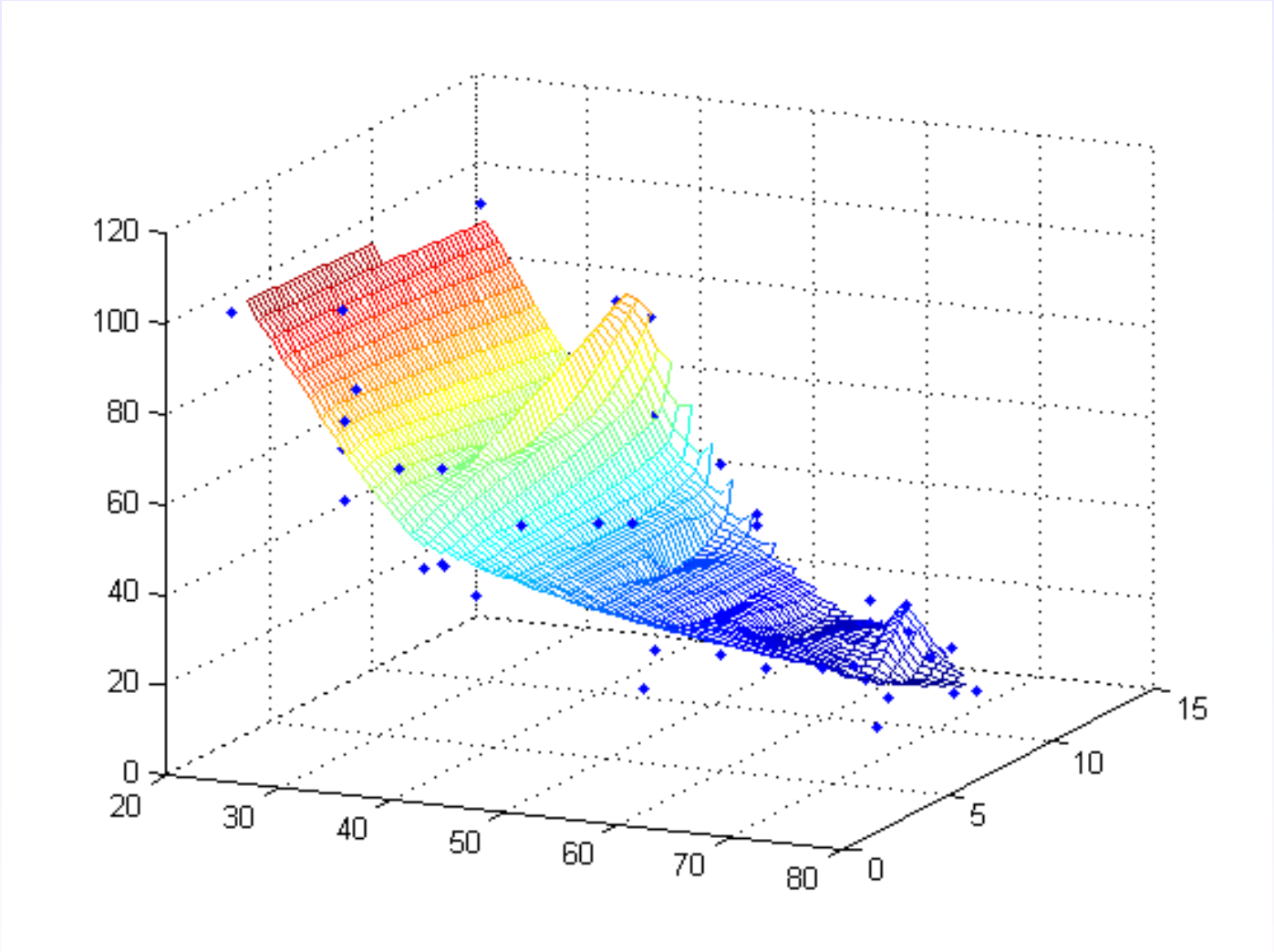
Learning data error (red) and testing data error (green) as a function of polynomial order.

Overfitting and underfitting of the network



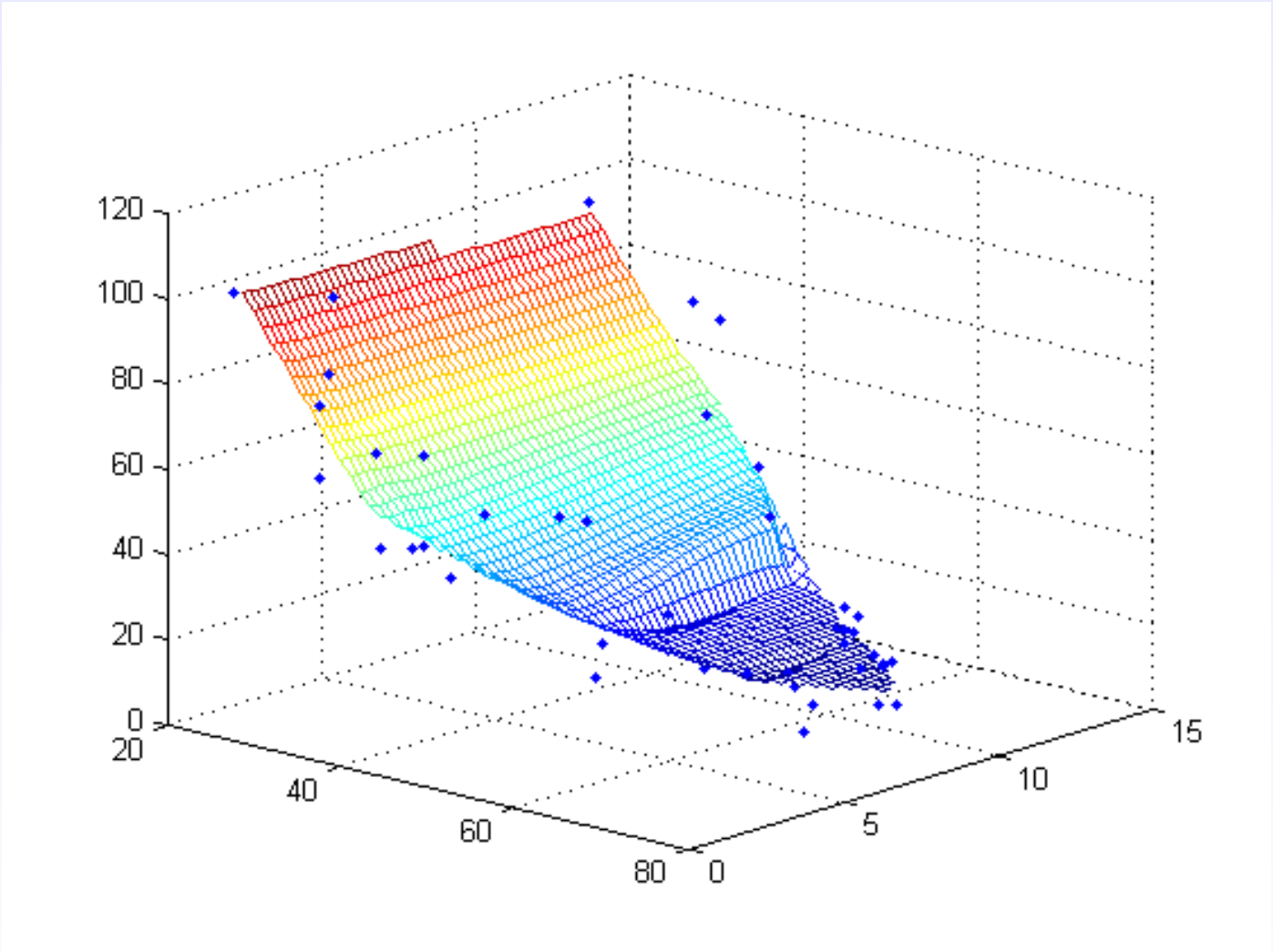
The network is underfitted (hidden layer with 1 neuron).

Overfitting and underfitting of the network



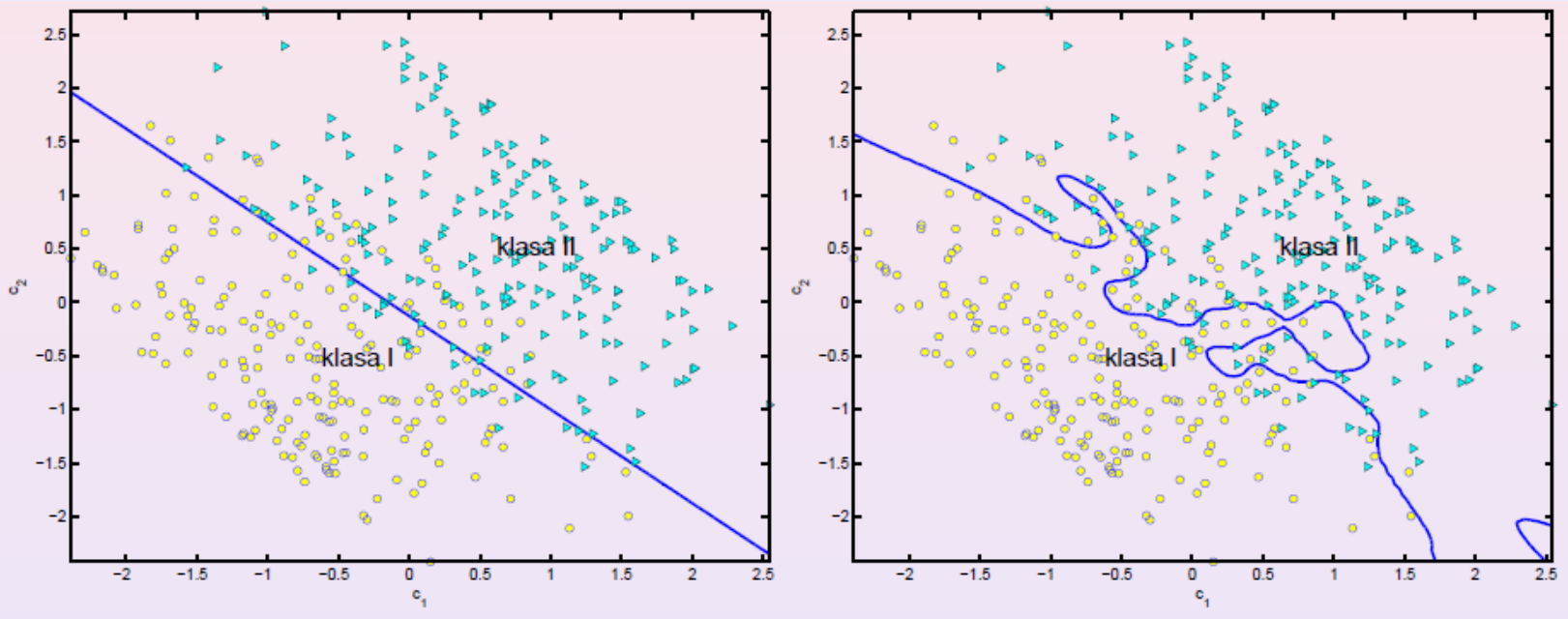
The network is overfitted (hidden layer with 10 neurons).

Overfitting and underfitting of the network



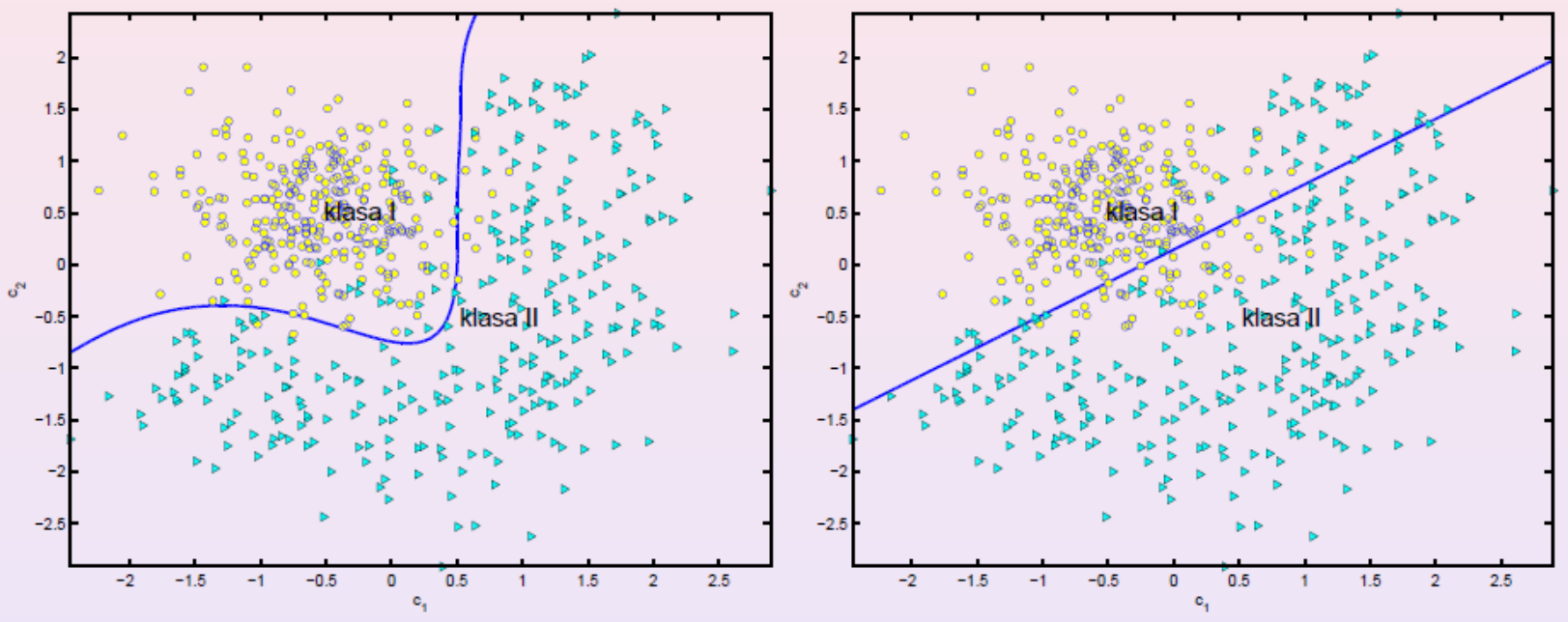
The network is learnt correctly (hidden layer with 3 neurons).

Overfitting and underfitting of the network



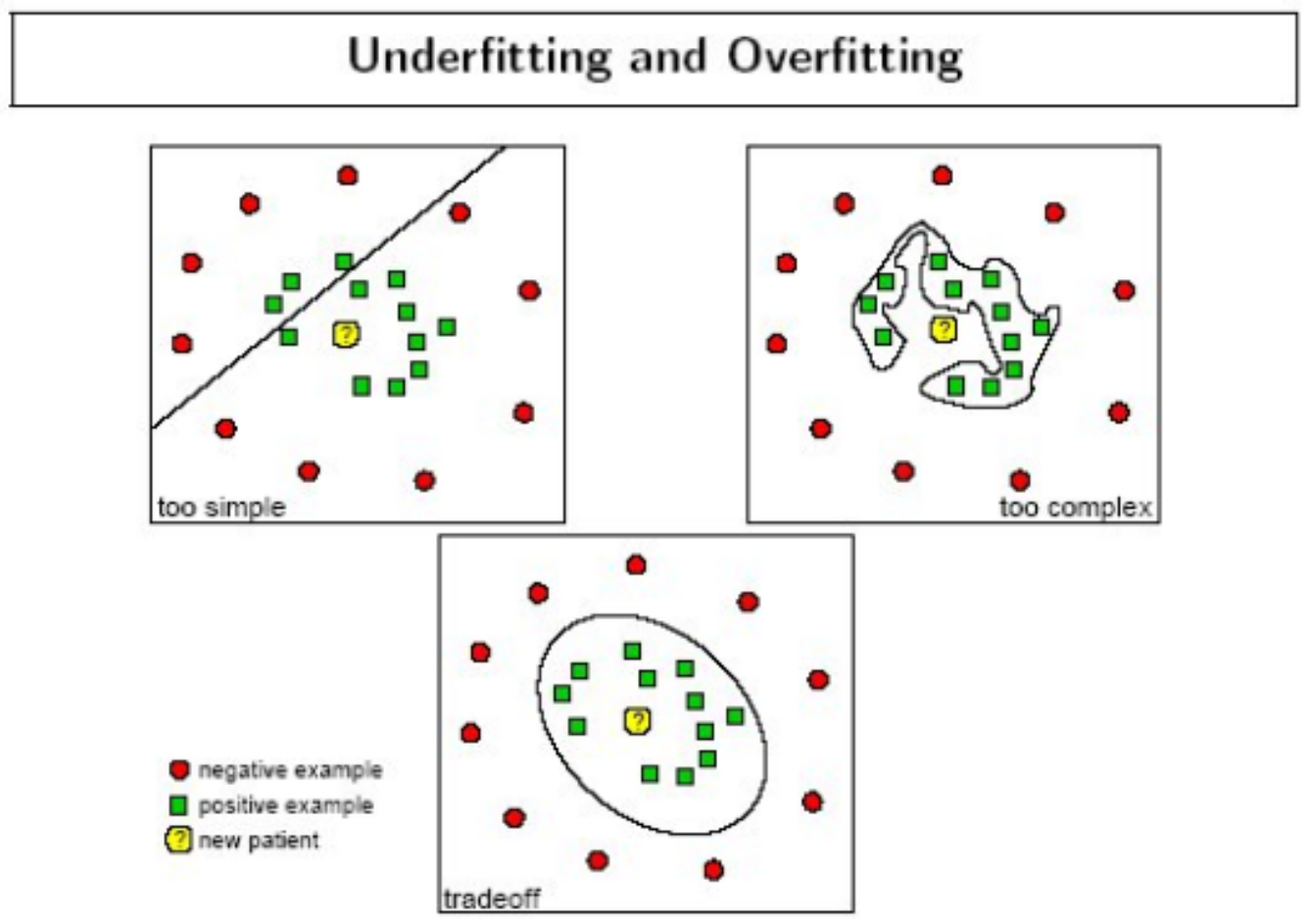
Correctly learnt network (left) and overfitted network (right).

Overfitting and underfitting of the network

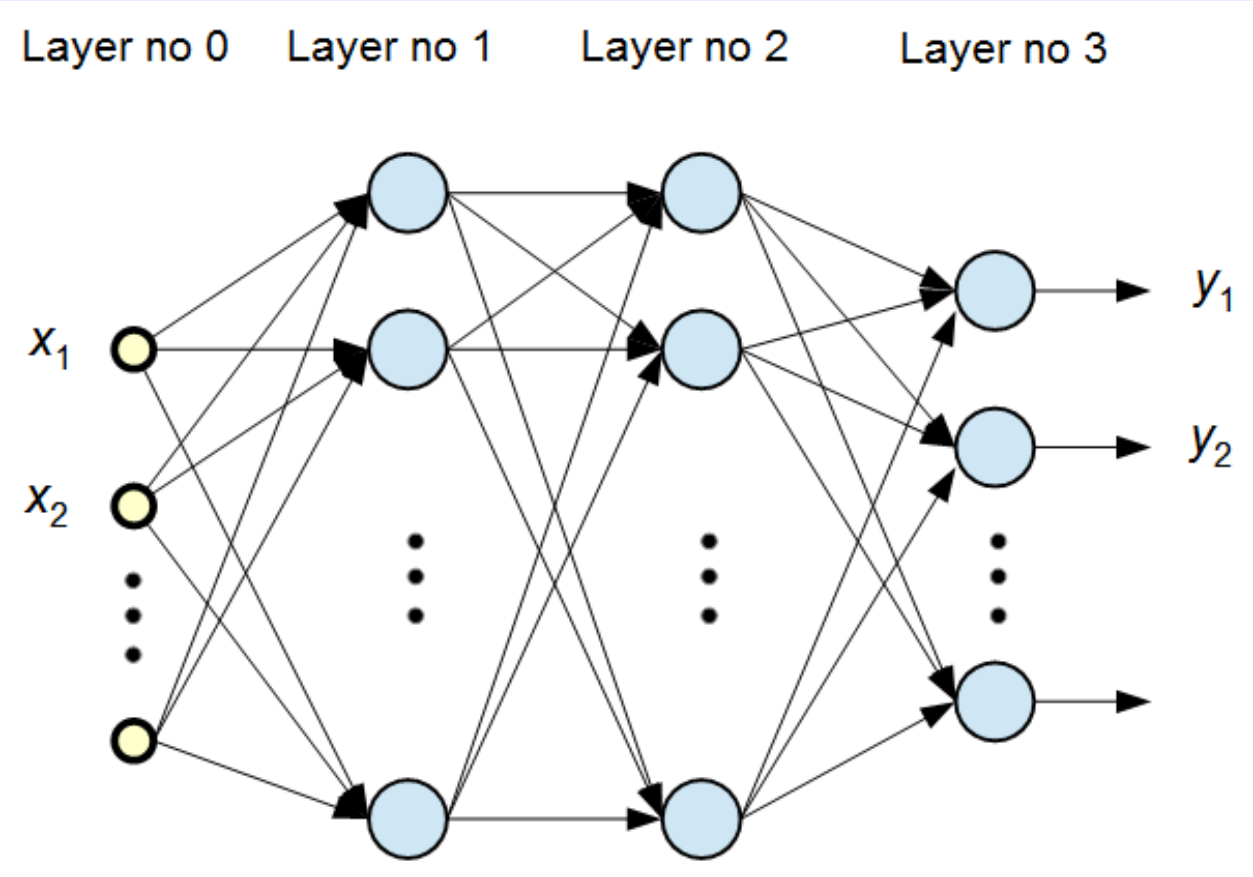


Correctly learnt network (left) and underfitted network (right).

Overfitting and underfitting of the network



Multilayer, feedforward neural network



Multilayer, feedforward neural network

We can consider a N-layer network with the same activation functions $f(\phi)$. Let's introduce the following notation:

- $n = 0, \ldots, N$ – number of the layer
- $p = 1, \ldots, L$ – number of the sample (L – no of samples in the learning set)
- $i, j = 1, \ldots, t_n$ – no of the neuron on the layer n (t_n – no of neurons on the layer n)

Multilayer, feedforward neural network

Output of the neuron j on the layer no n can be calculated as:

$$v_j^{np} = f(\phi_j^{np}) = f\left(\sum_{i=0}^{t_{n-1}} w_{ji}^n \cdot v_i^{(n-1)p}\right) = f\left(\mathbf{w}_j^{nT} \cdot \mathbf{v}^{(n-1)p}\right) \qquad (1)$$

where: w_{ji}^n – weight on input no i.

$$y_j^p = v_j^{Np} \qquad x_i^p = v_i^{0p}$$

Multilayer, feedforward neural network

Output of the neuron j on the layer no n can be calculated as:

$$v_j^{np} = f(\phi_j^{np}) = f\left(\sum_{i=0}^{t_{n-1}} w_{ji}^n \cdot v_i^{(n-1)p}\right) = f\left(\mathbf{w}_j^{nT} \cdot \mathbf{v}^{(n-1)p}\right) \qquad (1)$$

where: w_{ji}^n – weight on input no i.

$$y_j^p = v_j^{Np} \qquad x_i^p = v_i^{0p}$$

We can calculate **generalized** error for the output layer:

$$\delta'^{Np}_j = f'(\phi_j^{Np}) \cdot \delta_j^p = f'(\phi_j^{Np}) \cdot (d_j^p - y_j^p) \qquad (2)$$

Multilayer, feedforward neural network

The error is backpropagated onto hidden layers:

$$\delta'^{np}_j = f'(\phi^{np}_j) \sum_{k=1}^{t_{n+1}} \delta'^{(n+1)p}_k \cdot w^{(n+1)}_{kj} \tag{3}$$

Multilayer, feedforward neural network

The error is backpropagated onto hidden layers:

$$\delta'^{np}_j = f'(\phi^{np}_j) \sum_{k=1}^{t_{n+1}} \delta'^{(n+1)p}_k \cdot w^{(n+1)}_{kj} \tag{3}$$

After calculating of generalized errors for all network neurons, we can calculate weight corrections with the generalized delta rule:

$$\Delta w^{np}_{ji} = \eta \cdot \delta'^{np}_j \cdot v^{(n-1)p}_i \tag{4}$$

$$\Delta \mathbf{w}^{np}_j = \eta \cdot \delta'^{np}_j \cdot \mathbf{v}^{(n-1)p}$$

Backpropagation error algorithm

For the sample no p:

1. Put the vector $\mathbf{x}^p$ into the network input.
2. Calculate the output v_j^{np} for each neuron on successive network layers, from the first hidden layer to the output (formula 1).
3. Calculate generalized errors for the output layer (formula 2).
4. Backpropagate output generalized error onto hidden layers neurons (formula 3).
5. Calculate correction of weights (formula 4) and modify network weights.

Backpropagation error algorithm

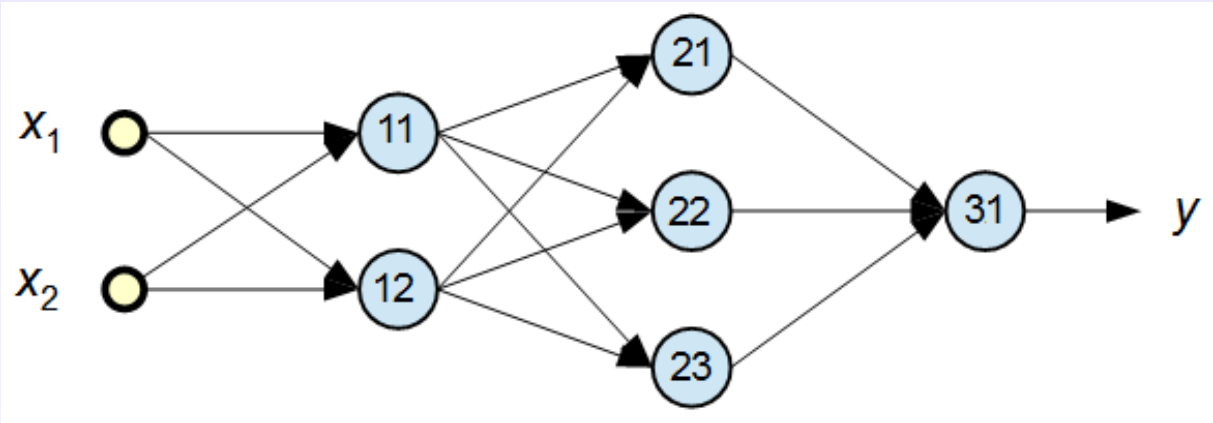
For the sample no p:

1. Put the vector $\mathbf{x}^p$ into the network input.
2. Calculate the output v_j^{np} for each neuron on successive network layers, from the first hidden layer to the output (formula 1).
3. Calculate generalized errors for the output layer (formula 2).
4. Backpropagate output generalized error onto hidden layers neurons (formula 3).
5. Calculate correction of weights (formula 4) and modify network weights.

3A: After step no 3, calculate the sum of squared errors $Q^p = \sum_{j=1}^{N} (\delta_j^{Np})^2$ and add it to the total error of learning data Q.

3B: If this was the last sample in the set, we must check whether the error Q is smaller than the given threshold. If so, abort learning.

Network of linear neurons – example



x_1	x_2	d
1	2	1
0	1	2
2	0	3
⋮	⋮	⋮

$$\mathbf{w}^{11}(0) = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.1 \end{bmatrix} \quad \mathbf{w}^{12}(0) = \begin{bmatrix} 0.2 \\ -0.2 \\ 0.2 \end{bmatrix}$$

$$\mathbf{w}^{21}(0) = \begin{bmatrix} -0.1 \\ 0.2 \\ 0.3 \end{bmatrix} \quad \mathbf{w}^{22}(0) = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.2 \end{bmatrix} \quad \mathbf{w}^{23}(0) = \begin{bmatrix} -0.2 \\ 0.1 \\ 0.2 \end{bmatrix} \quad \mathbf{w}^{31}(0) = \begin{bmatrix} -0.1 \\ 0.3 \\ -0.2 \\ 0.1 \end{bmatrix}$$

RBF Neural Networks

RBF – Radial Basis Function

Neurons in the hidden layer implements the function changing its value radially around a certain point $\mathbf{c}$ known as the center of the neuron.

The RBF function has the general form:

$$f(\mathbf{x}) = \phi(||\mathbf{x} - \mathbf{c}||)$$

where: $||\mathbf{x} - \mathbf{c}||$ – the distance between a sample $\mathbf{x}$ and the center $\mathbf{c}$.

RBF Neural Networks

One of the most popular RBF functions is the Gauss function:

$$\phi(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{c}||^2}{2\sigma^2}\right)$$

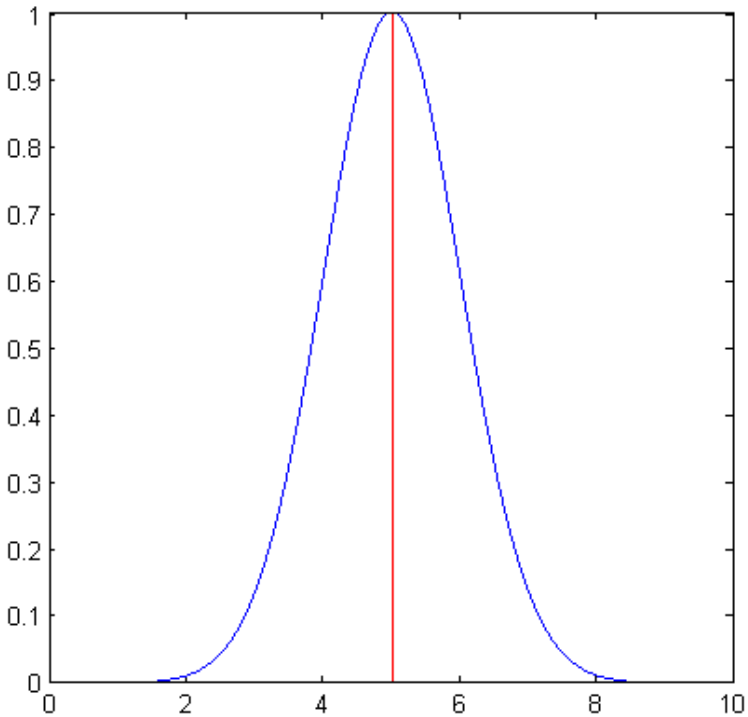
where:

$$||\mathbf{x}-\mathbf{c}|| = \sqrt{\sum_{i=1}^{n}(x_i - c_i)^2}\,, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

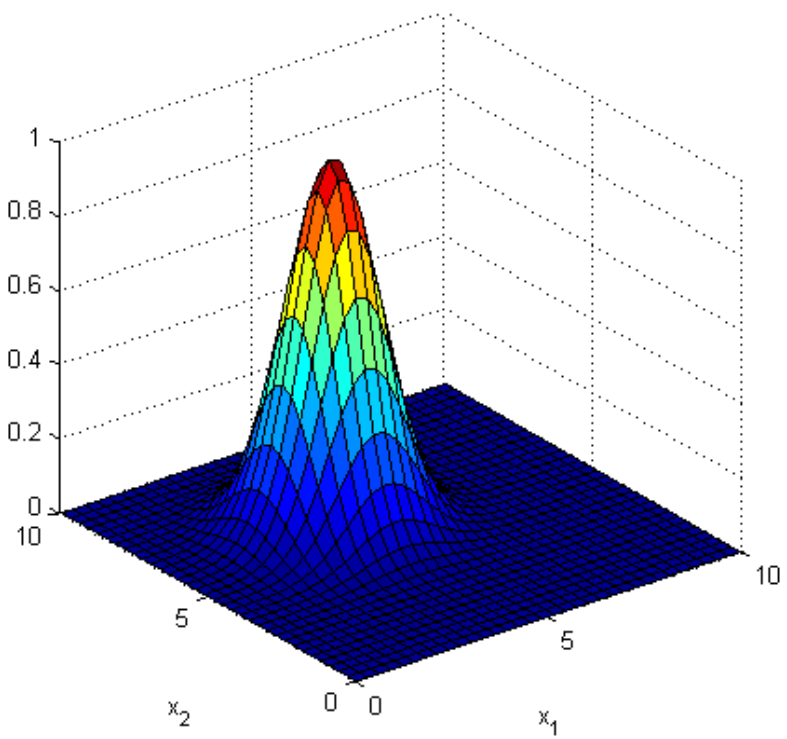
The parameter σ defines the spread of the RBF function.

Gauss function

$$\phi(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{c}||^2}{2\sigma^2}\right)$$



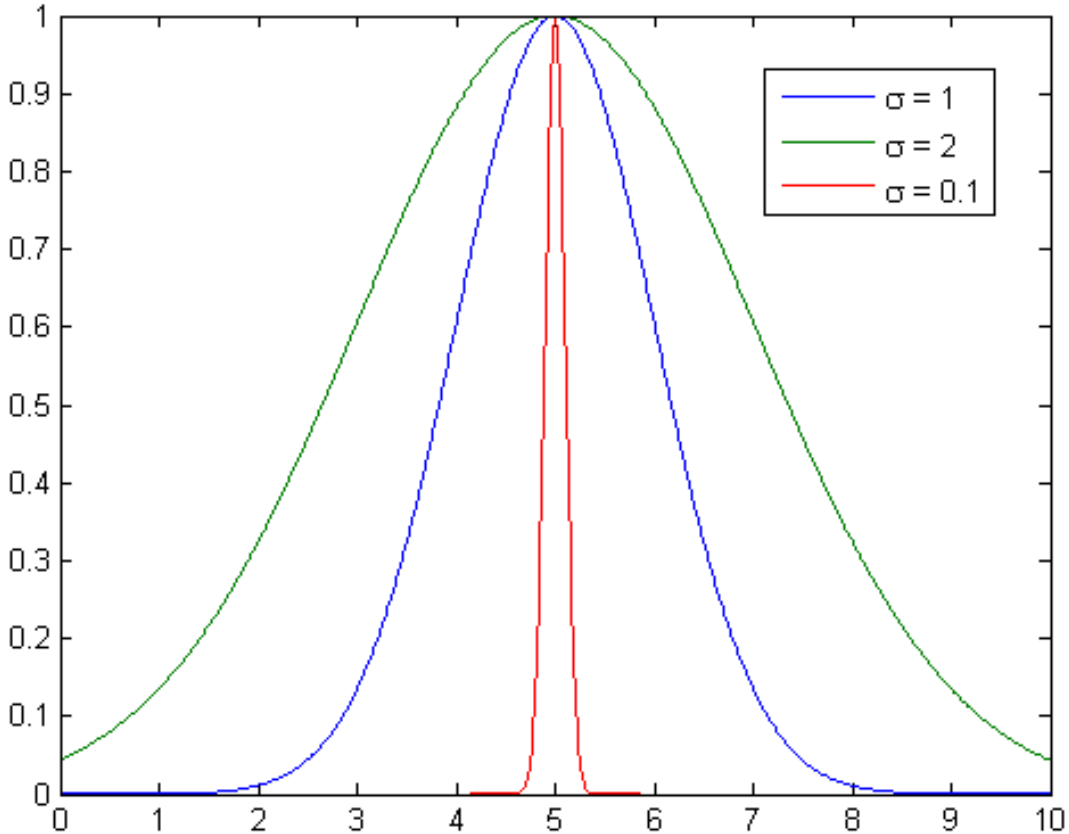
$c = 5, \quad \sigma = 1$



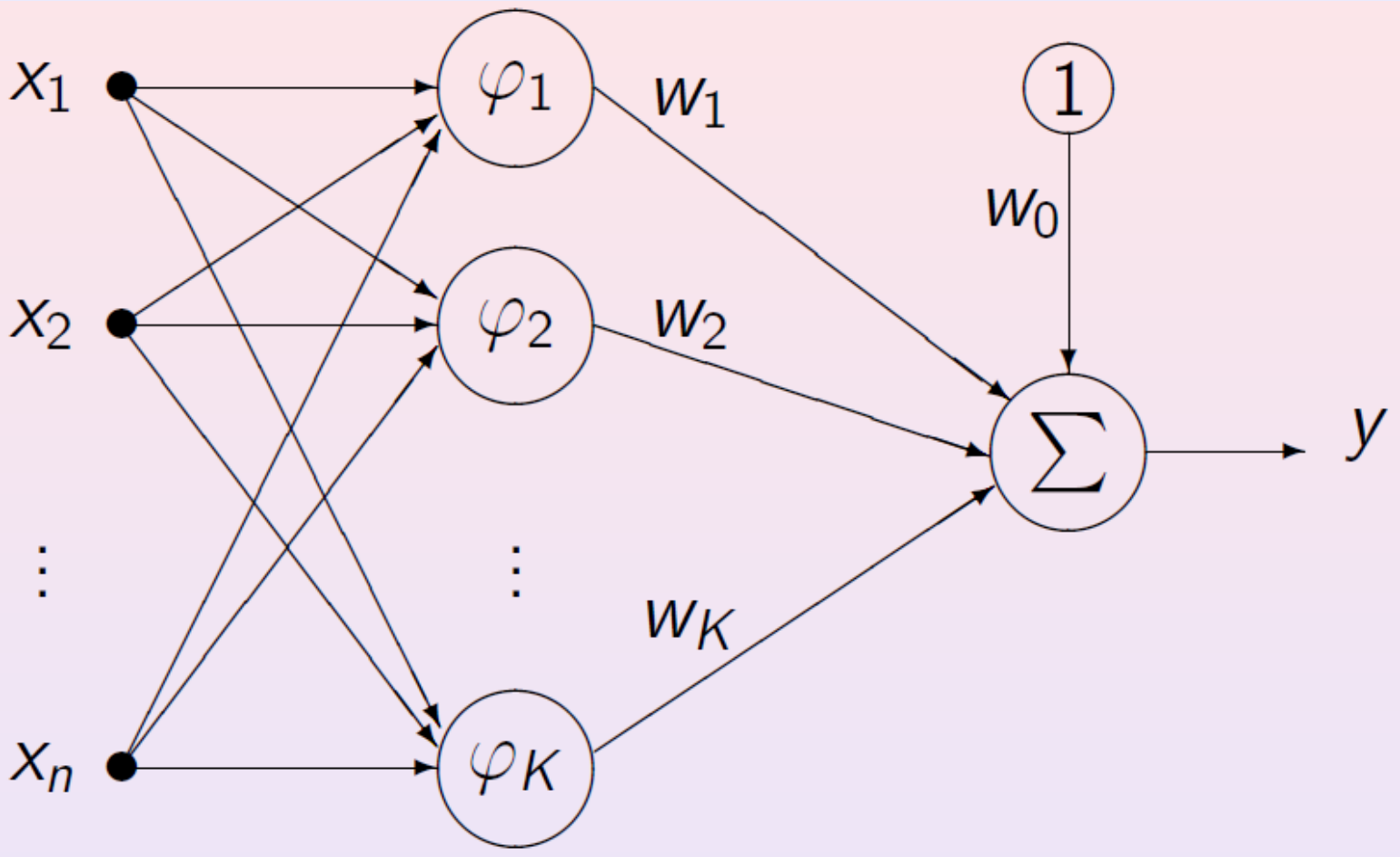
$\mathbf{c} = [4\ 6]^T, \quad \sigma = 1$

Gauss function

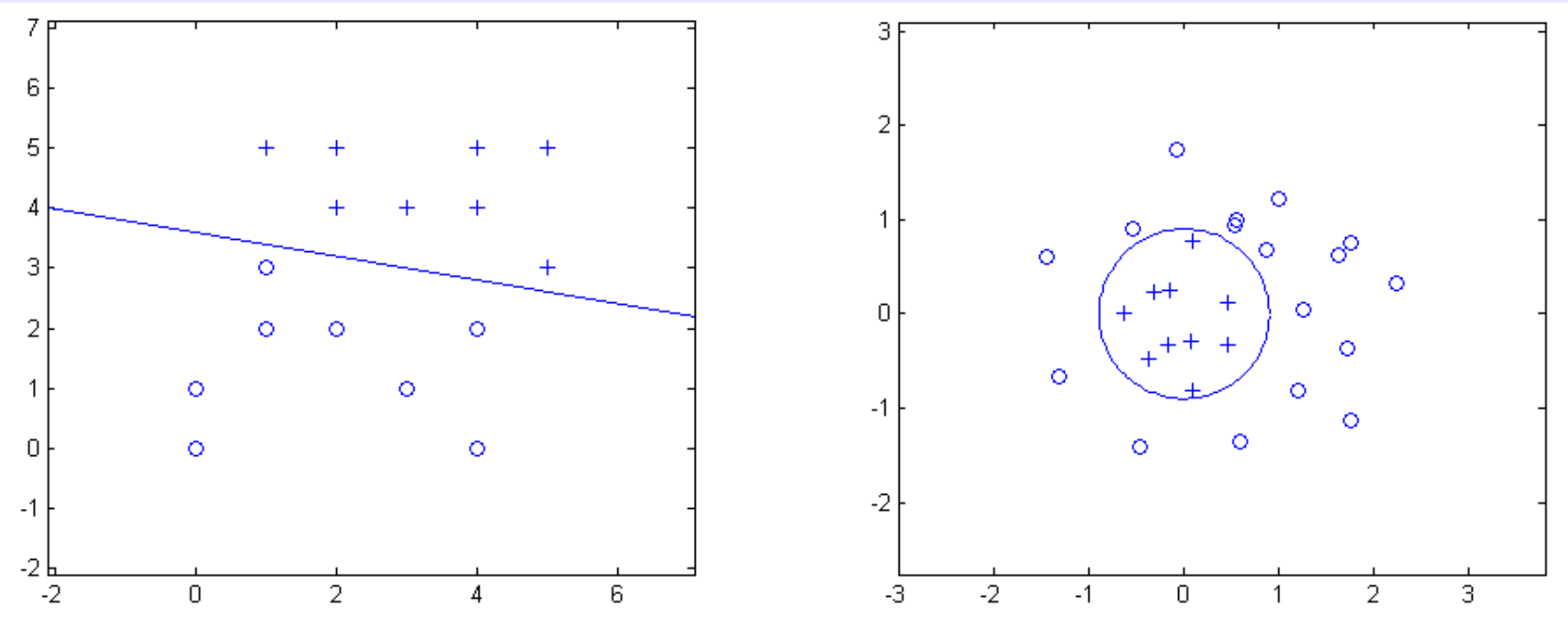
$$\phi(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{c}||^2}{2\sigma^2}\right)$$



Structure of the RBF network



Performance of the RBF network



In classification tasks, the RBF neuron divides the input space into 2 parts with a boundary which has the shape of a circle (for 2 inputs), a sphere (for 3 inputs) or hyper-sphere.

Learning of the RBF network

Learning of the RBF network is supervised. Before learning, the input part of data samples should be normalized.

1. Choose the number K of RBF neurons on the hidden layer.
2. Determine centres $\mathbf{c}_i$ and spreads σ_i of RBF neurons.
3. Determine weights w_j of output layer neurons.

Determination of centres $\mathbf{c}_i$ and spreads σ_i

In the simplest case, centres of RBF neurons can be selected randomly (uniform distribution of centres in the input space is very advantageous).

- The spread can be the same for all neurons and equal:

$$\sigma = \frac{d}{\sqrt{2K}}$$

  where: d – the maximum distance between the centres.

- Another way is to take σ_i equal to mean standard deviation of the distance of samples from the centre they belongs to.
- σ_i can be also equal to the distance between the centre $\mathbf{c}_i$ from the closest neighbour centre.
- The best result is obtained by dividing the data into learning and validating parts and the application of the validation.

Determination of centres $\mathbf{c}_i$ – clusterization

K-means algorithm:

1. Generate randomly K points in the input space (uniform distribution of points in the input space is very advantageous). These points are the initial cluster centres $\mathbf{c}_i$.
2. Assign each sample to the nearest center $\mathbf{c}_i$.
3. Calculate new cluster centres:

$$\mathbf{c}_i^{new} = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{x}_j^{(i)}$$

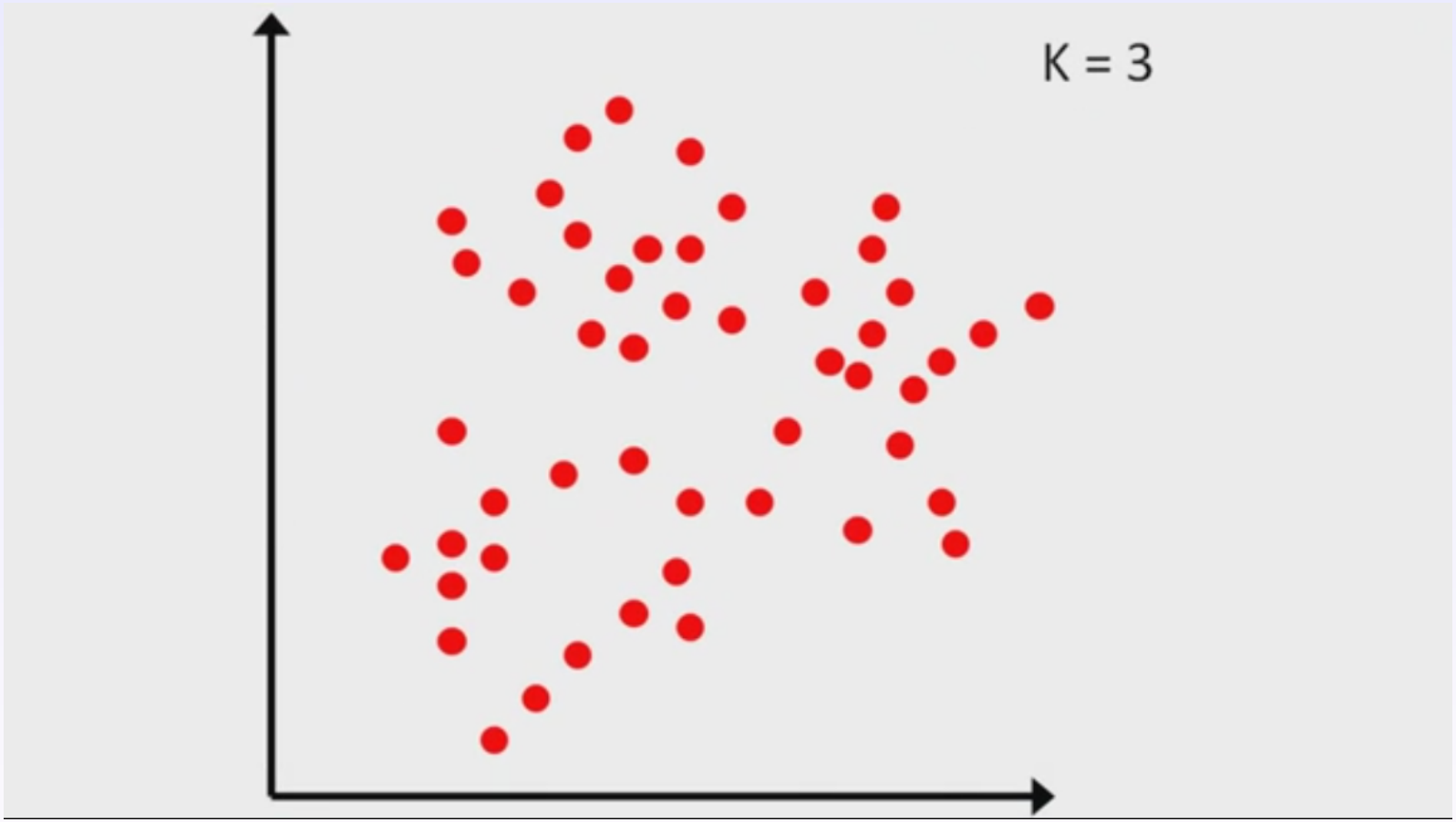
   where: $\mathbf{x}_j^{(i)}$ – sample no j which belongs to cluster i, N_i – no of samples that belongs to cluster i.

4. Check the displacement of centres:

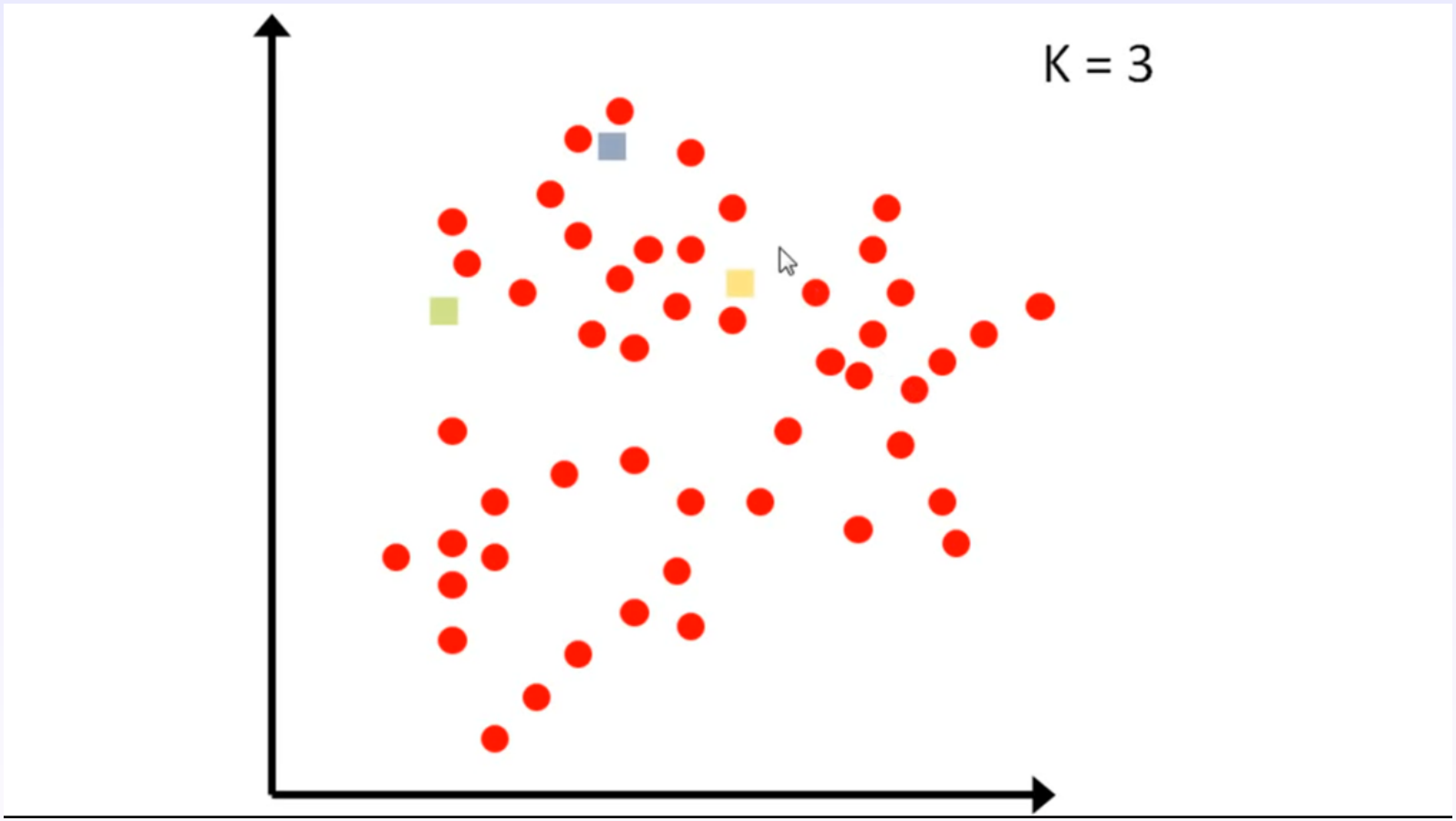
$$\Delta_i = ||\mathbf{c}_i^{new} - \mathbf{c}_i||$$

5. If $\max \Delta_i > \epsilon$ go back to point 2, otherwise the algorithm is finished.

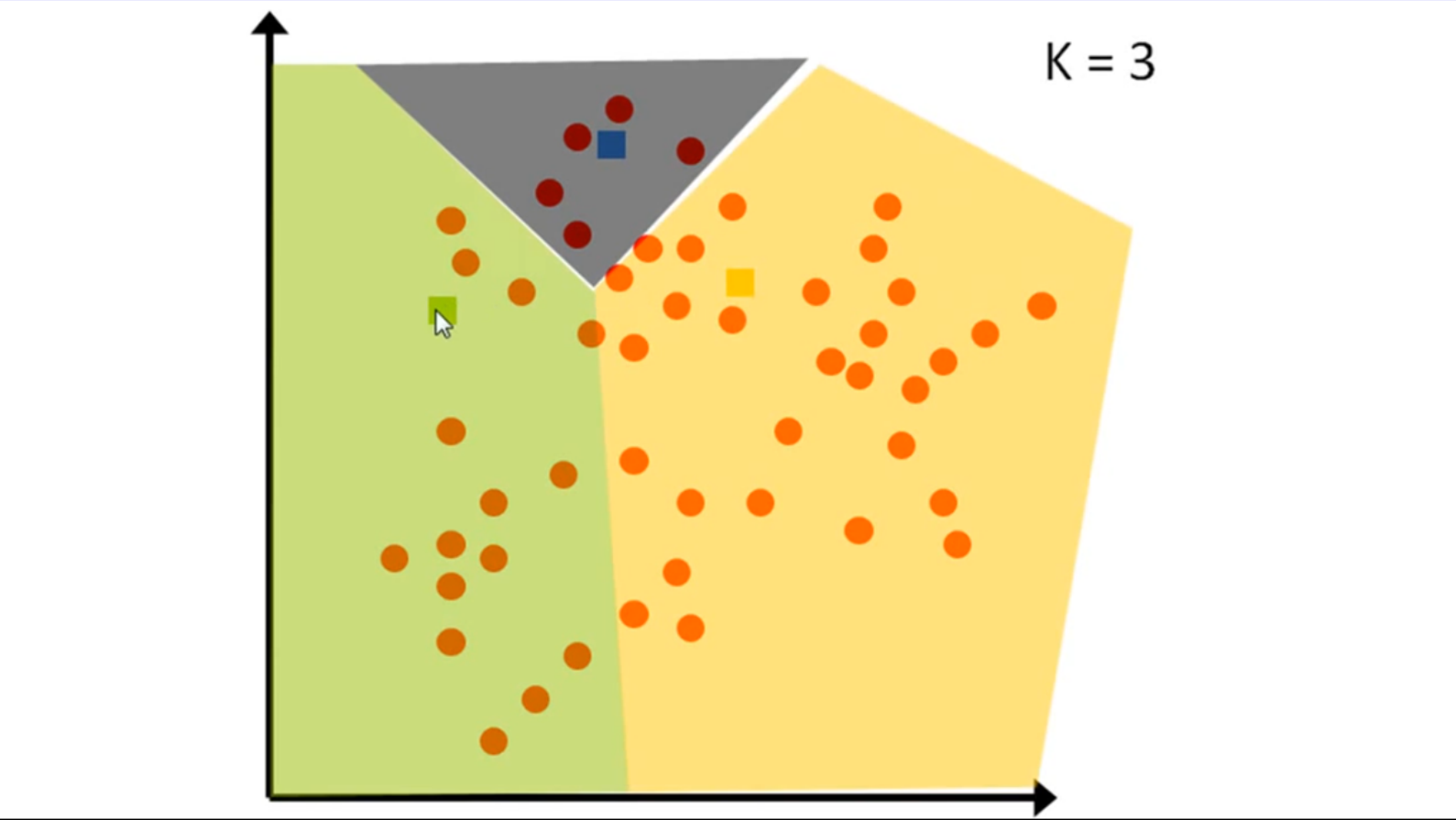
K-means algorithm – example



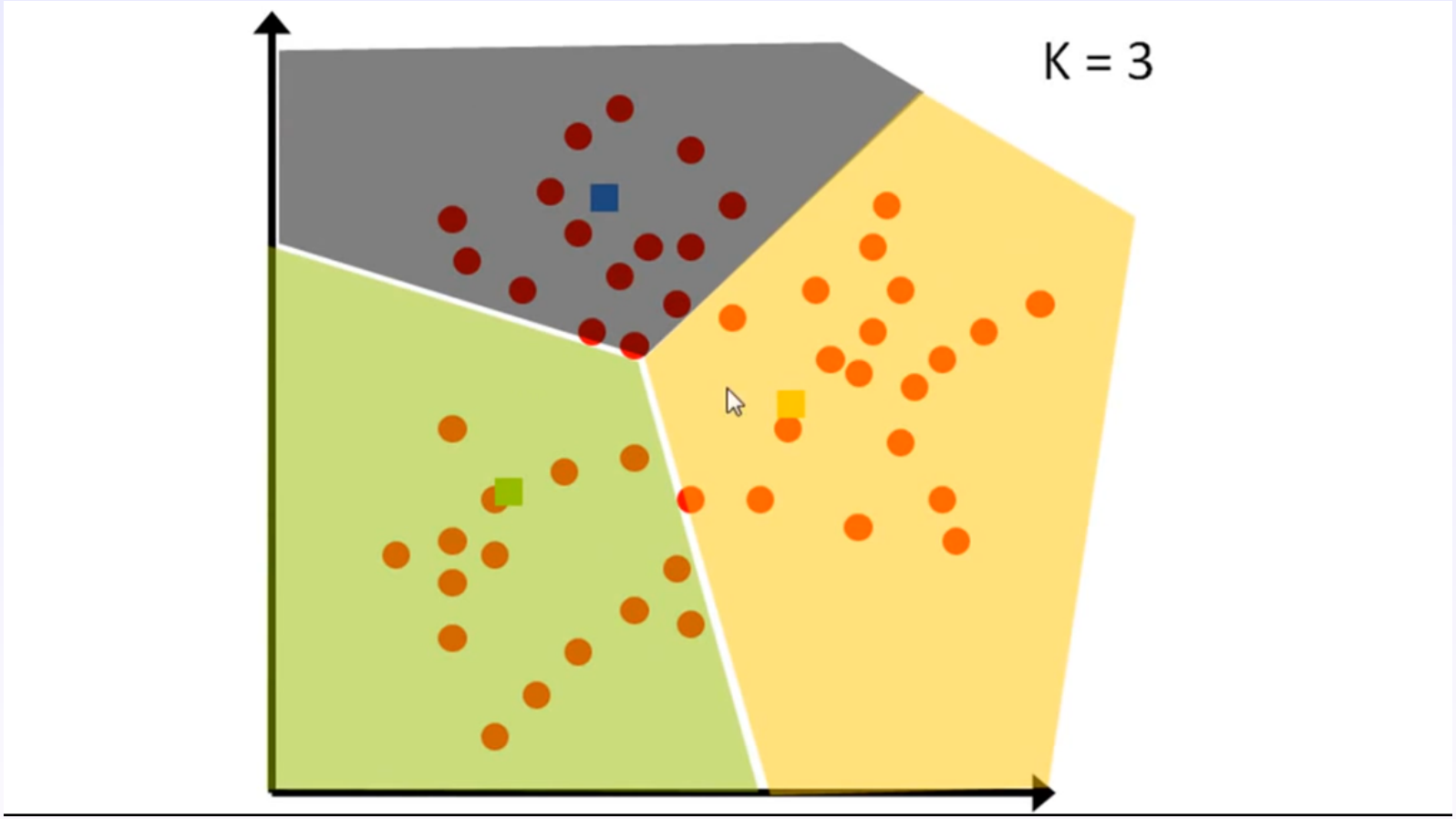
K-means algorithm – example



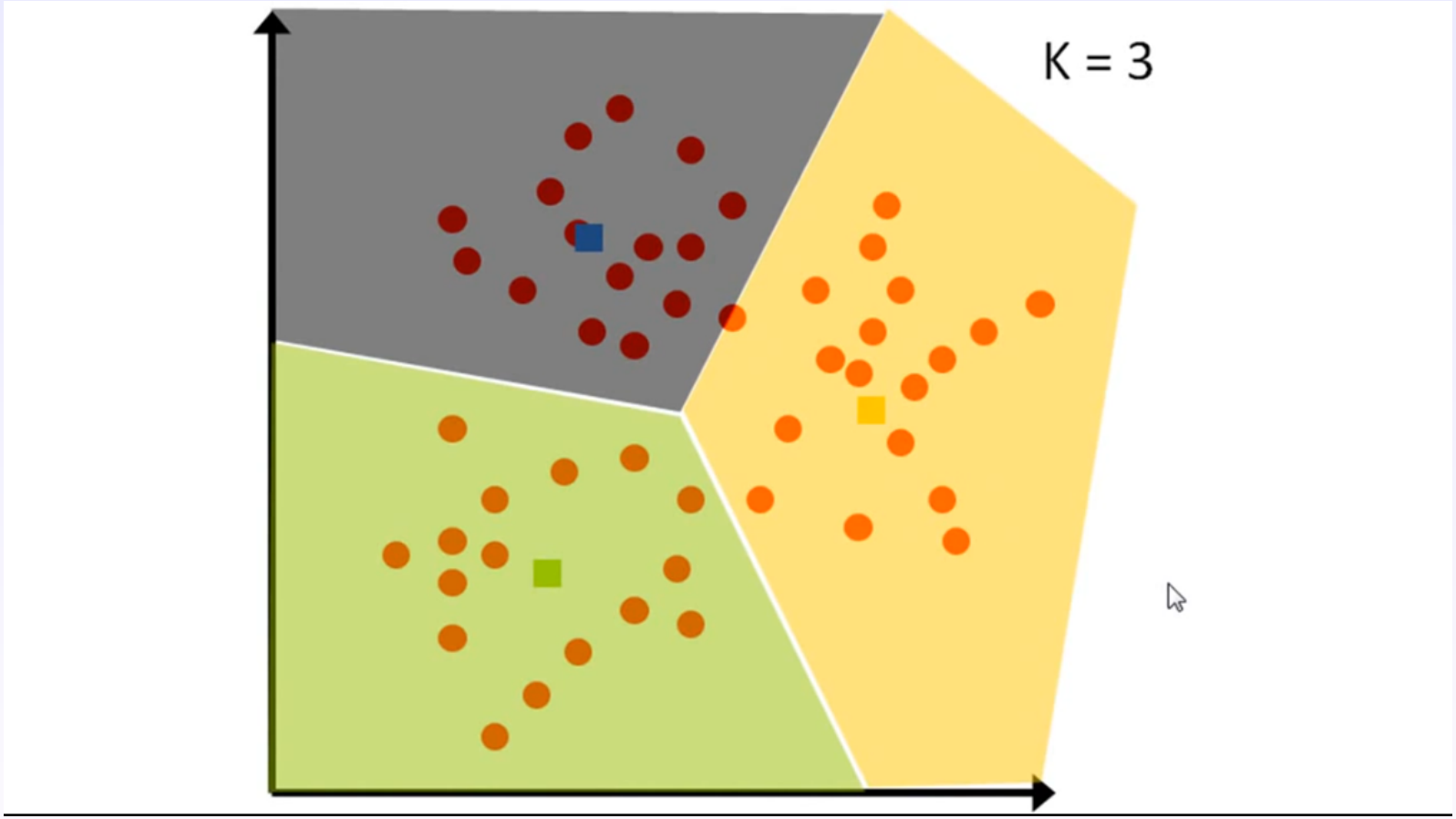
K-means algorithm – example



K-means algorithm – example



K-means algorithm – example



Determination of output layer weights

1. We have determined centres and spreads of RBF neurons, so output layer weights can be learnt on the base of the delta rule.
2. Weights can be also calculated in such a way that the mean square error of the model is minimised.

Calculation of weights

For each sample p we can create the equation:

$$w_0 + w_1 \cdot \phi(||\mathbf{x}^{(p)} - \mathbf{c}_1||) + \ldots + w_K \cdot \phi(||\mathbf{x}^{(p)} - \mathbf{c}_K||) = d^{(p)}$$

Assuming that we have L samples, we obtain the system of L equations, with $K+1$ unknown weights w_i.

This system can be written as:

$$\mathbf{G} \cdot \mathbf{w} = \mathbf{d}$$

where:

$$\mathbf{G} = \begin{bmatrix} 1 & \phi(||\mathbf{x}^{(1)} - \mathbf{c}_1||) & \ldots & \phi(||\mathbf{x}^{(1)} - \mathbf{c}_K||) \\ \vdots & & & \\ 1 & \phi(||\mathbf{x}^{(p)} - \mathbf{c}_1||) & \ldots & \phi(||\mathbf{x}^{(p)} - \mathbf{c}_K||) \\ \vdots & & & \\ 1 & \phi(||\mathbf{x}^{(L)} - \mathbf{c}_1||) & \ldots & \phi(||\mathbf{x}^{(L)} - \mathbf{c}_K||) \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_K \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} d^{(1)} \\ \vdots \\ d^{(p)} \\ \vdots \\ d^{(L)} \end{bmatrix}$$

Calculation of weights

The solution which minimises the mean square error of the model can be calculated as:

$$\mathbf{w} = \mathbf{G}^{+} \cdot \mathbf{d} = (\mathbf{G}^{T} \cdot \mathbf{G})^{-1} \cdot \mathbf{G}^{T} \cdot \mathbf{d}$$

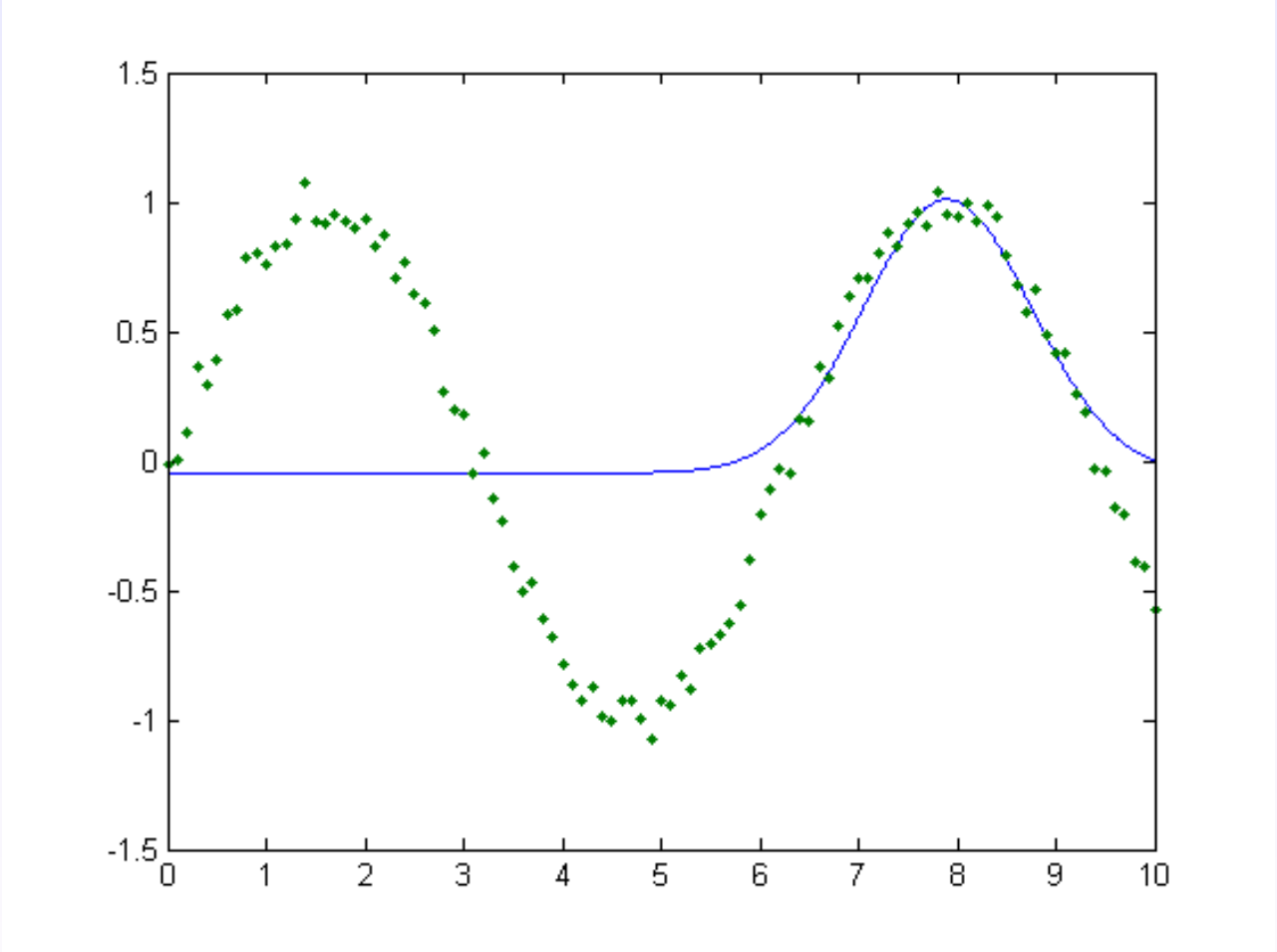
Determination of the RBF neurons number

1. Divide data into learning and validation parts.
2. Create the network with 1 RBF neuron, next with 2 neurons and so on.
3. Observe the error of the learning and validating data set.
4. If the error of the validating set begins to grow, stop the addition of neurons.

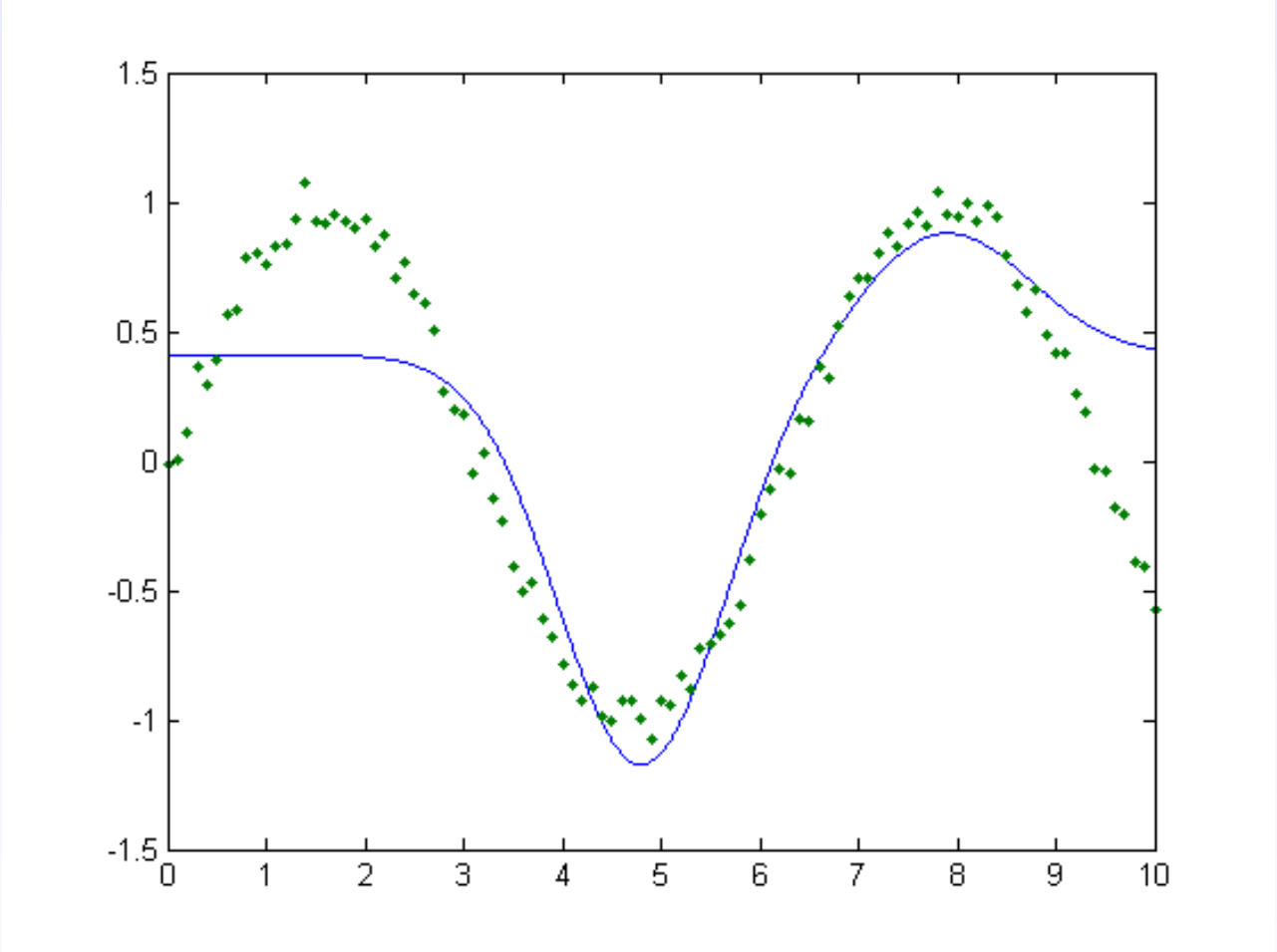
newrb method

1. Assume the spread of RBF neurons (the same for all).
2. Assume the minimum mean square error of the model.
3. Create the network with 1 RBF neuron.
4. Calculate the error for each sample.
5. Calculate the mean square error (MSE) of the model.
6. If MSE is greater than assumed minimum, add next RBF neuron and locate its center in the sample which cause the greatest error. Calculate output layer weights and go back to point 4. Otherwise stop learning.

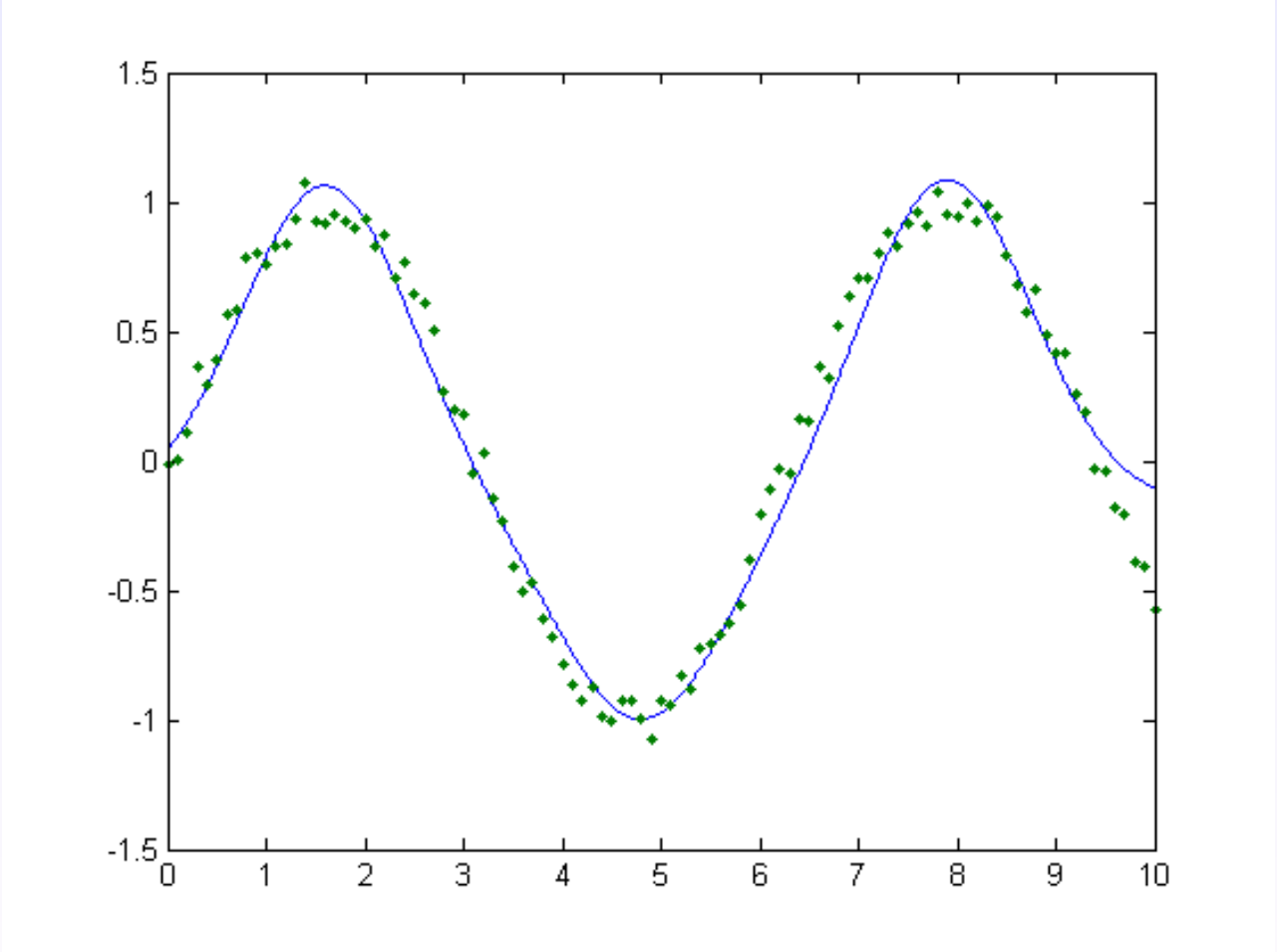
`newrb` method – example: $\sigma = 1$



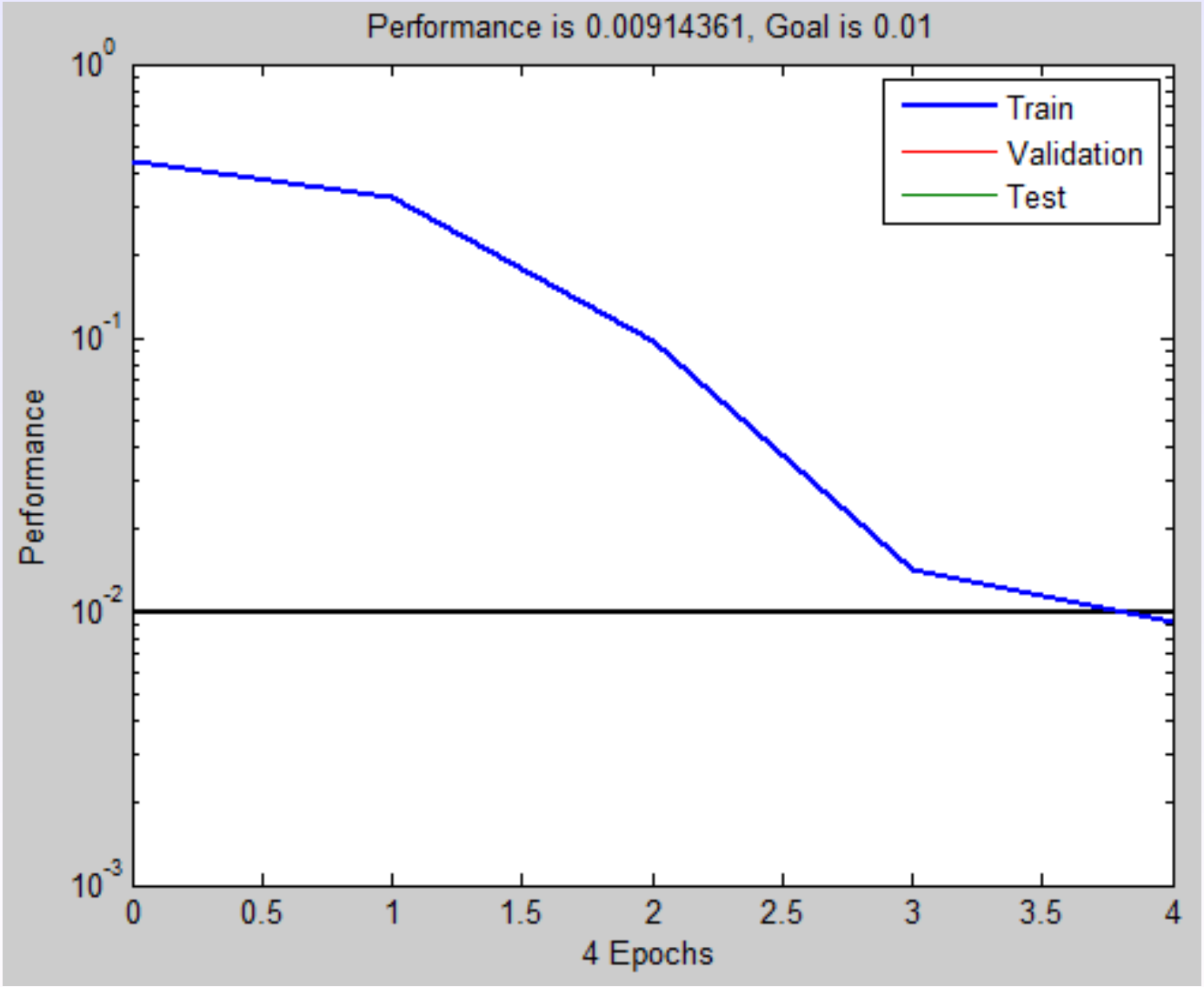
`newrb` method – example: $\sigma = 1$



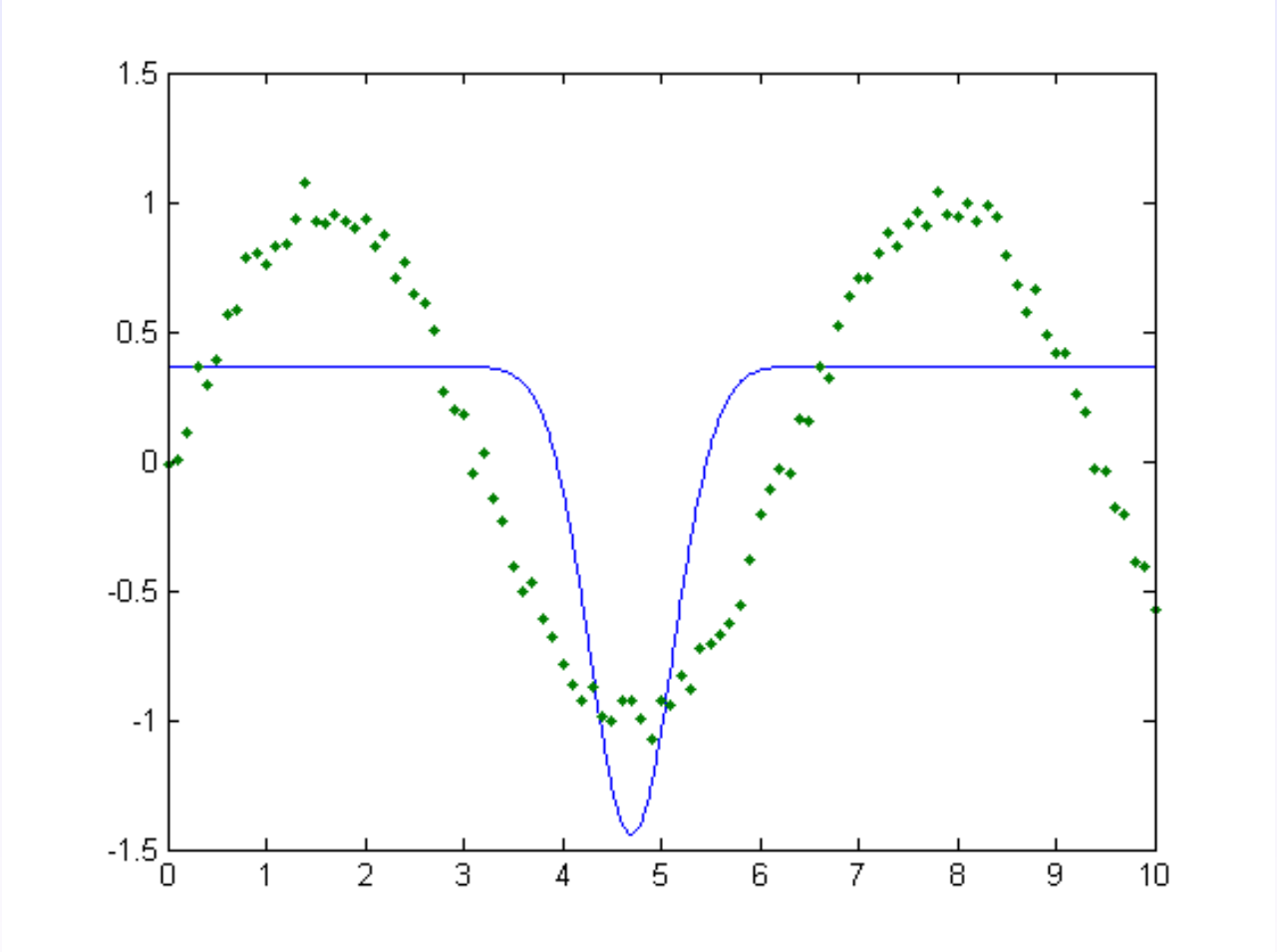
`newrb` method – example: $\sigma = 1$



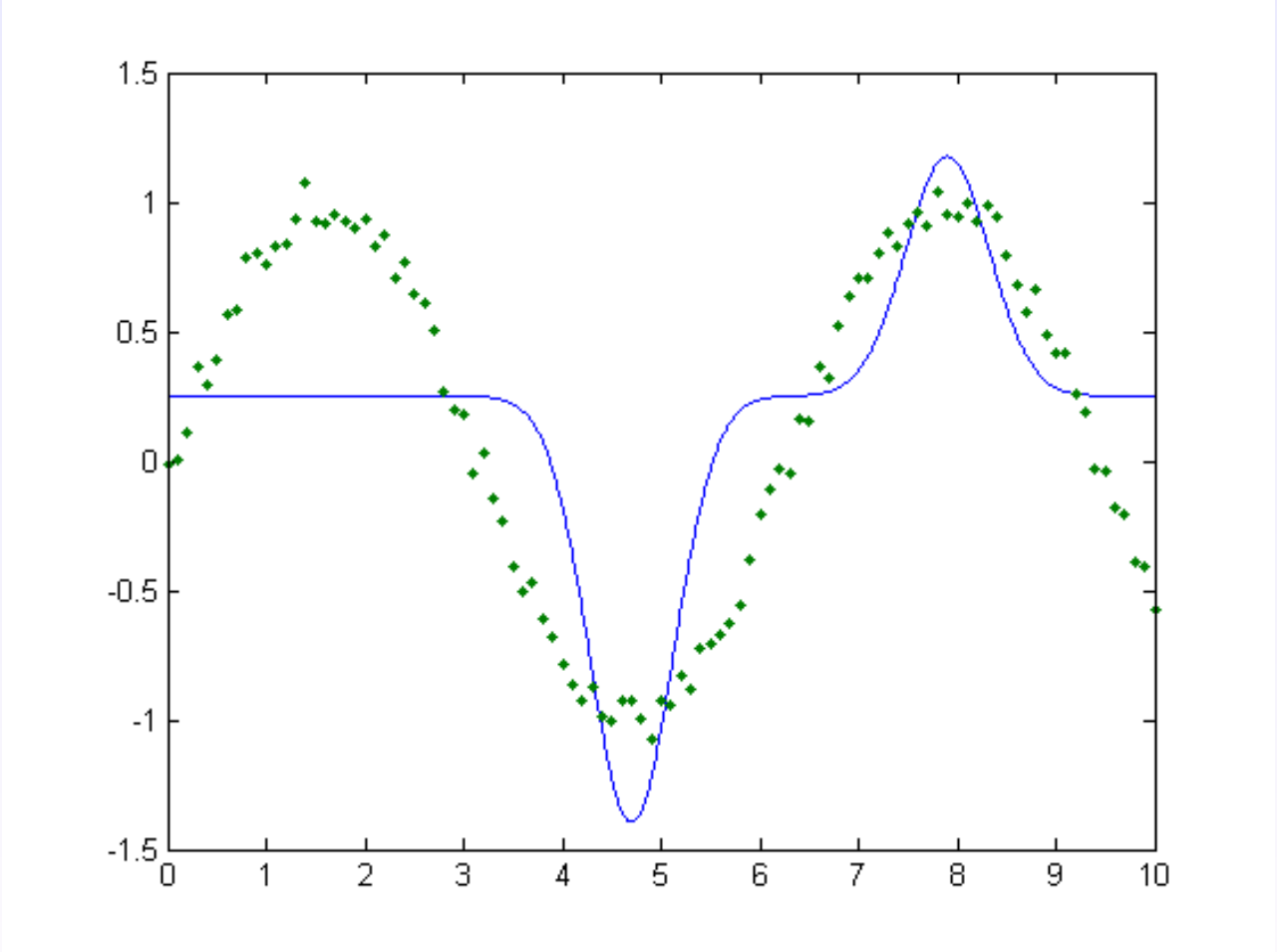
newrb method – example: $\sigma = 1$



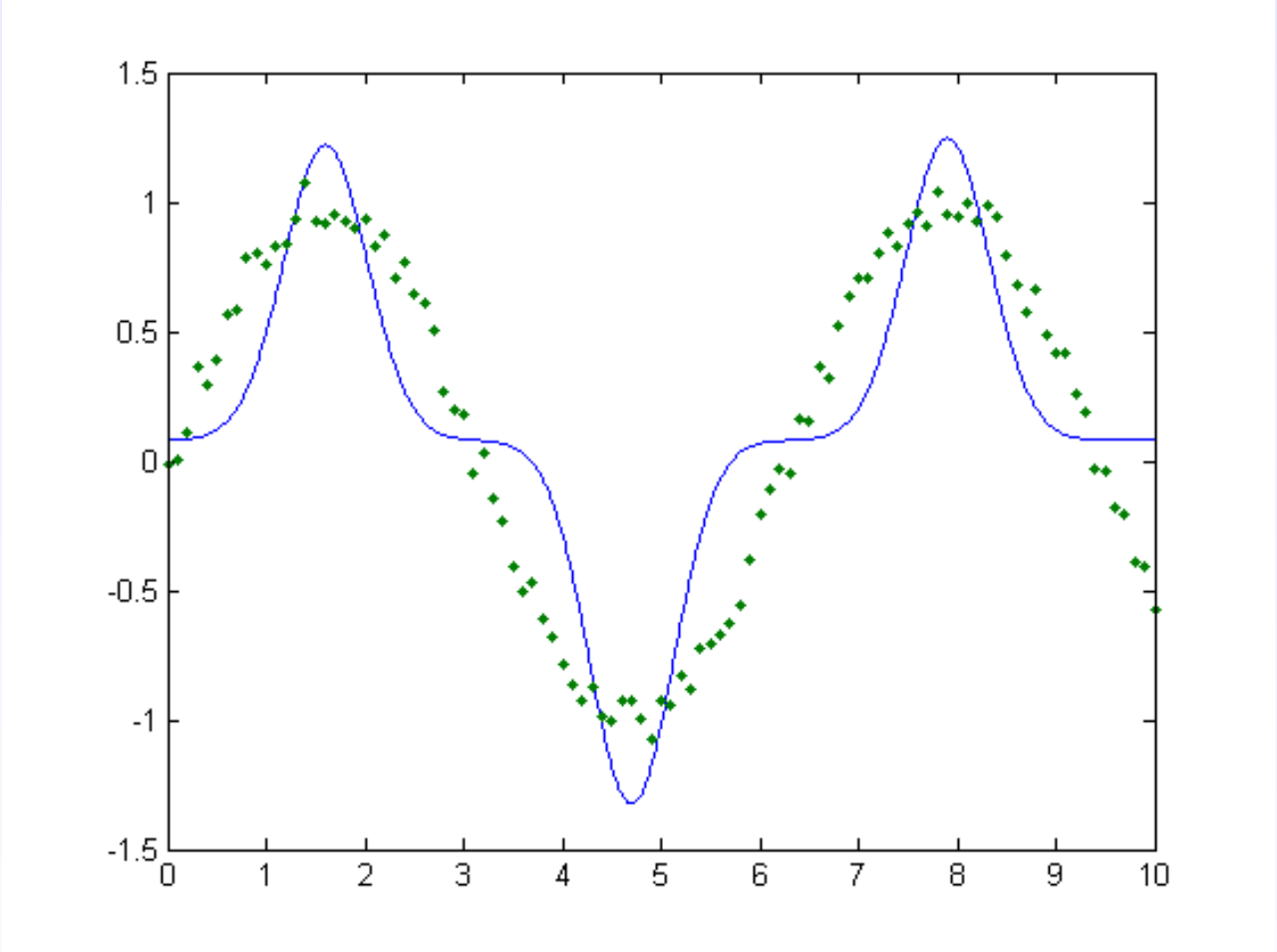
newrb method – example: $\sigma = 0.5$



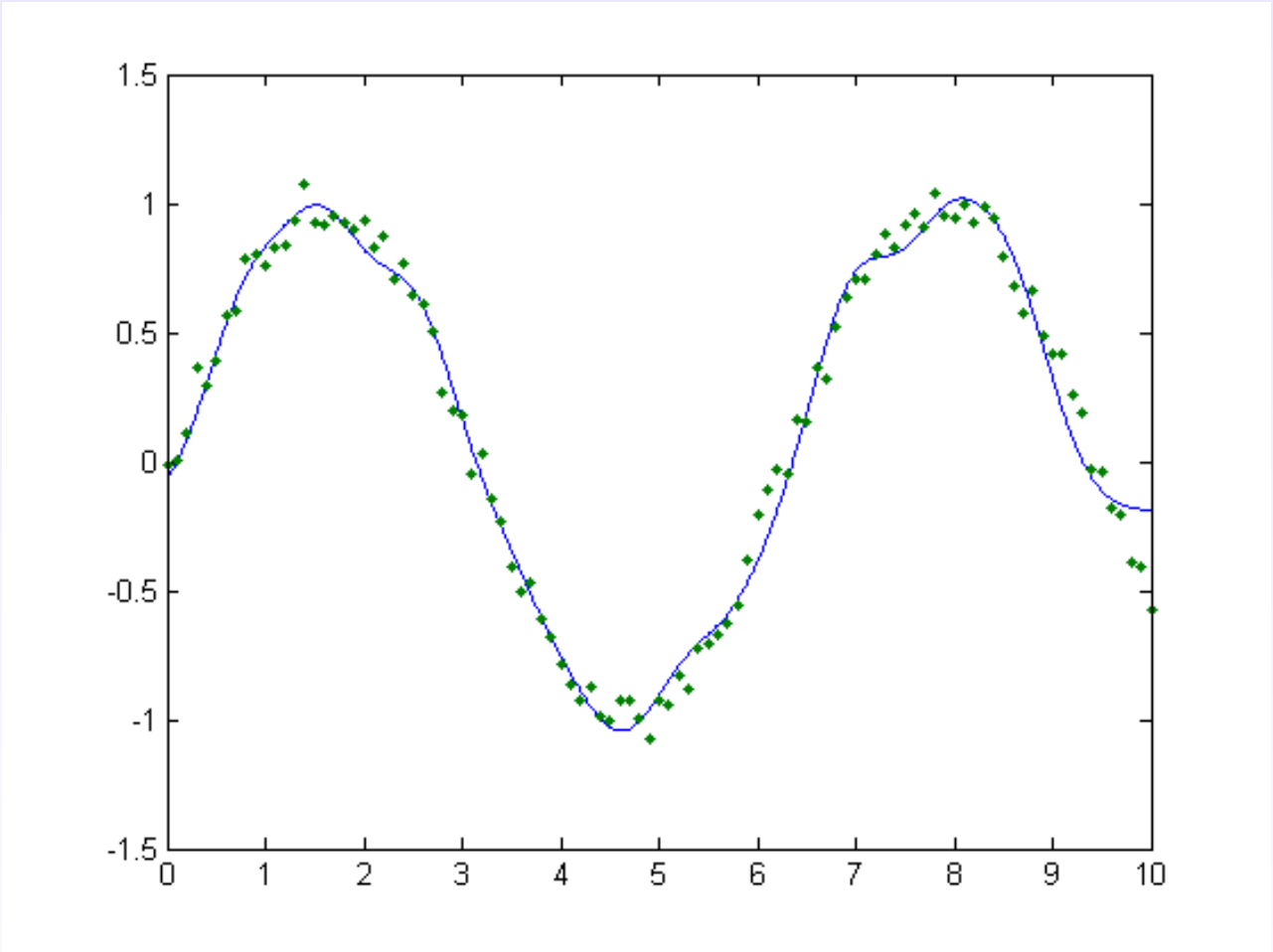
`newrb` method – example: $\sigma = 0.5$



`newrb` method – example: $\sigma = 0.5$

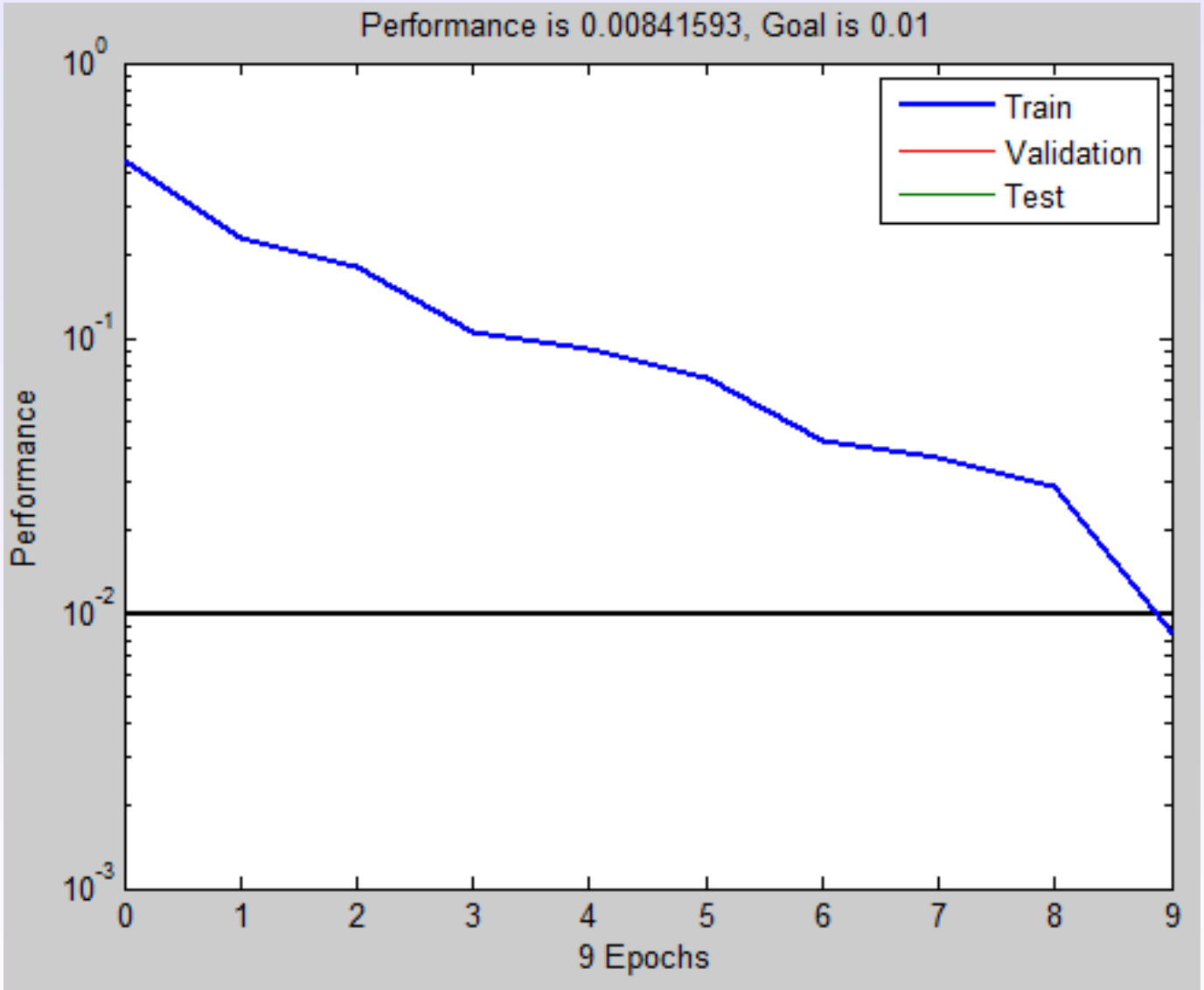


`newrb` method – example: $\sigma = 0.5$



Characteristic for the network with 9 RBF neurons.

`newrb` method – example: $\sigma = 0.5$



newrbe method

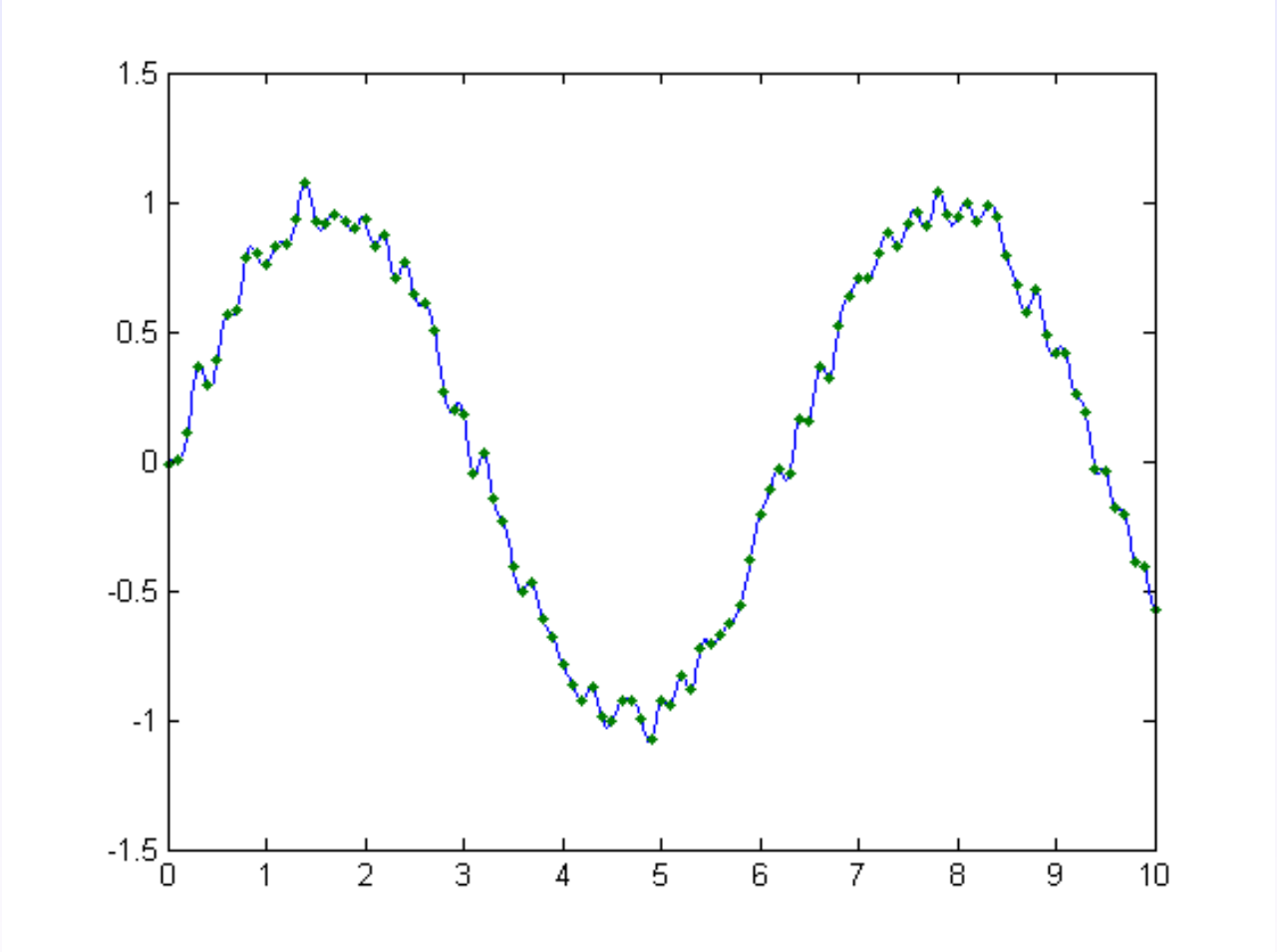
1. Assume the spread of RBF neurons (the same for all).
2. Assume the number of RBF neurons equal to the number of samples. Locate centres of neurons in learning samples.
3. Calculate output layer weights.

newrbe method

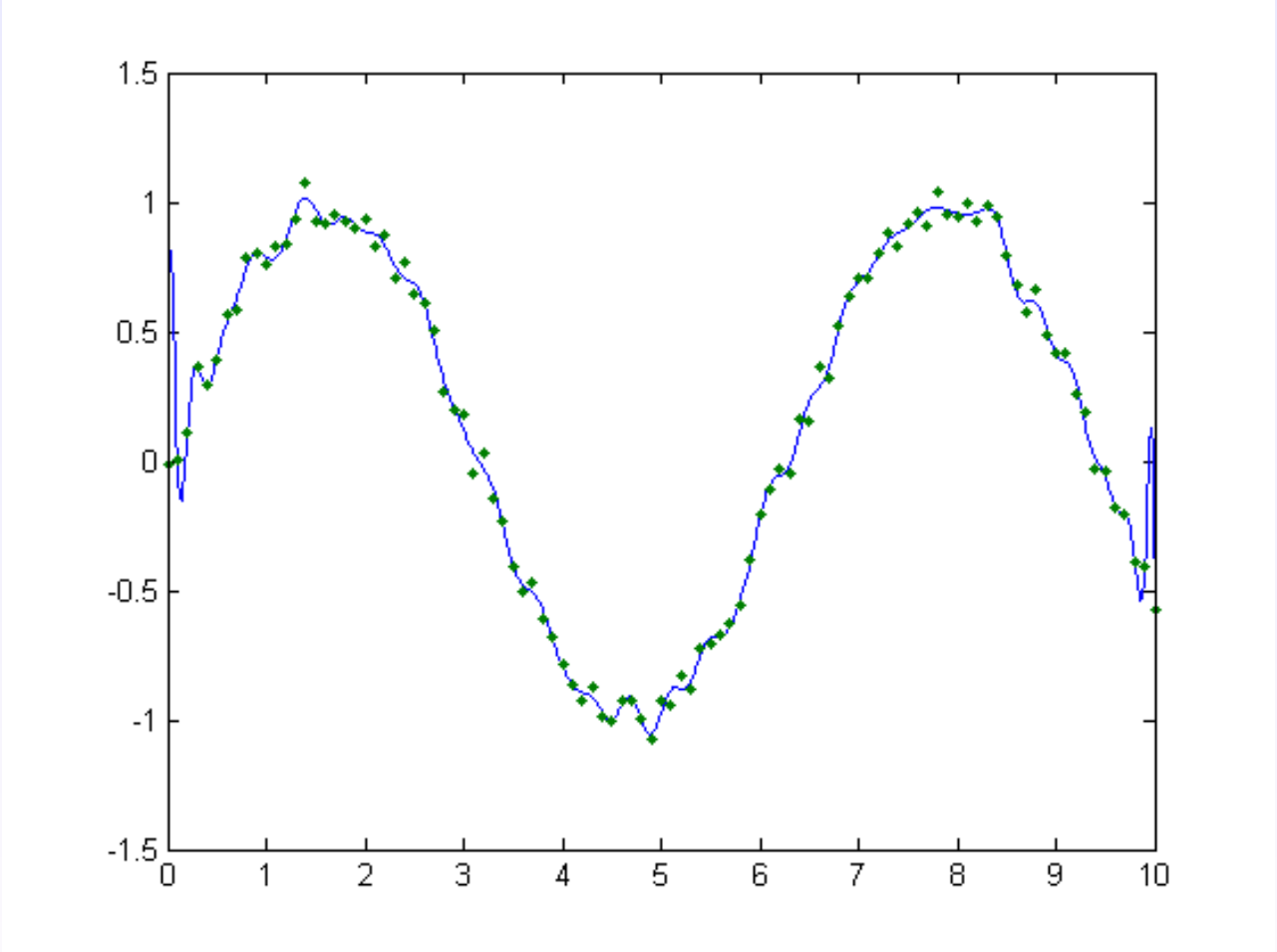
1. Assume the spread of RBF neurons (the same for all).
2. Assume the number of RBF neurons equal to the number of samples. Locate centres of neurons in learning samples.
3. Calculate output layer weights.

- There is **only one (!)** parameter to tune in the network. The rest is defined by data (centres) or calculated (weights).
- The network can be easily overfitted. For very small σ values the learning data error decreases to almost zero.
- Value of σ **must** be determined with the application of validation. If the amount of data is large, we can divide it into training and validating parts. As the σ value we must take the value that minimises validating data error.
- If the amount of data is small – we must apply crossvalidation.

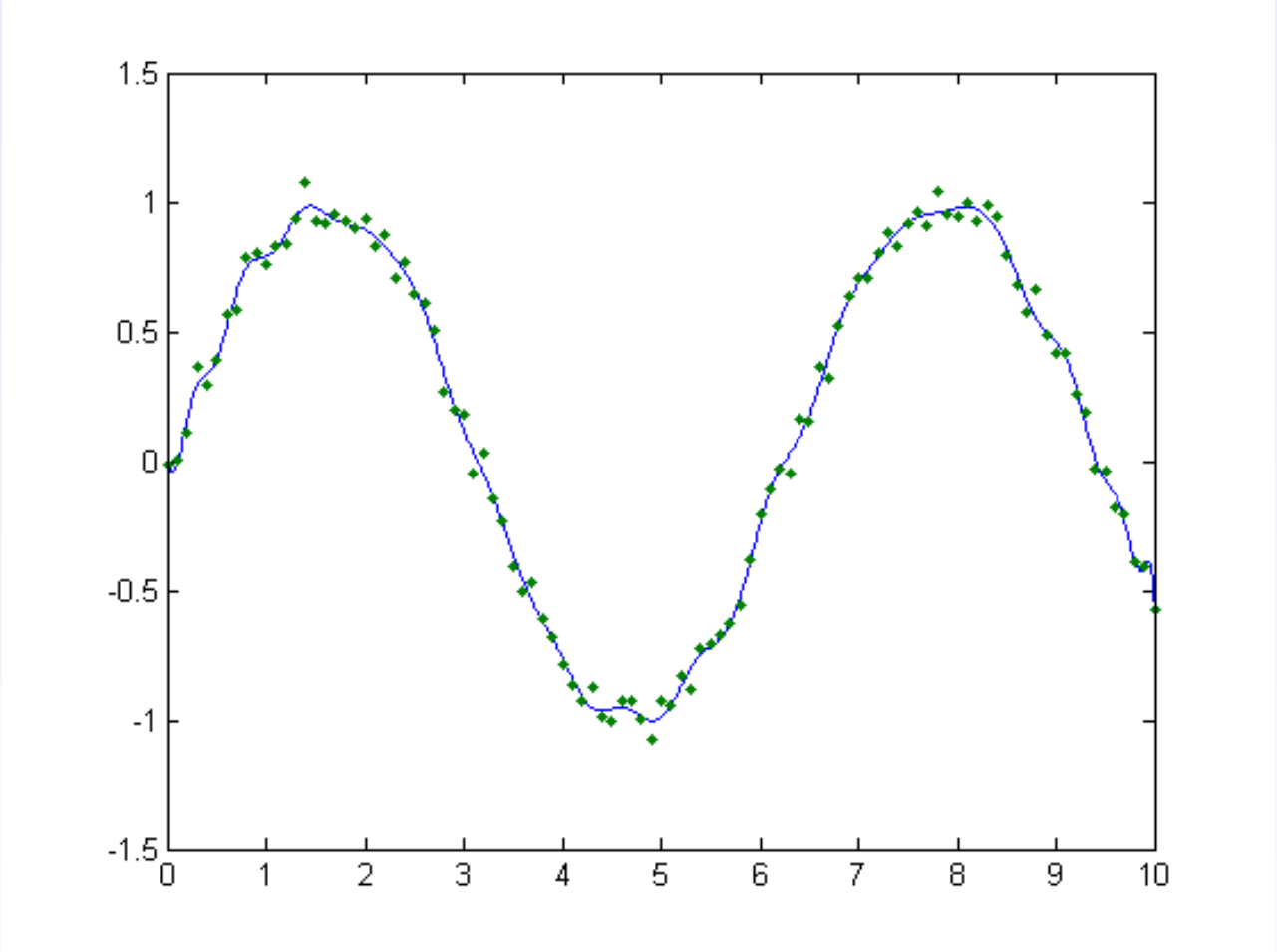
newrbe method– example: $\sigma = 0.1$



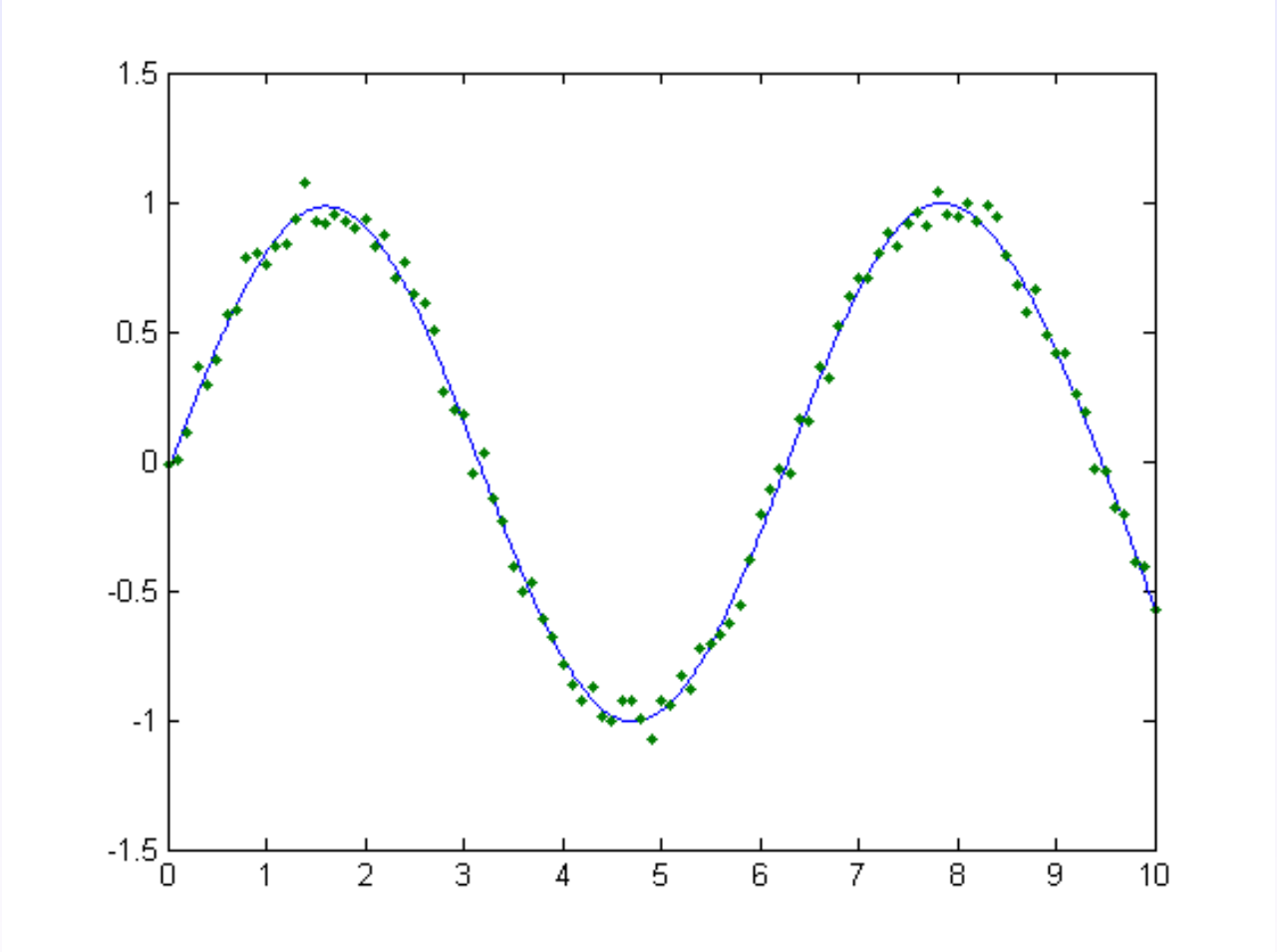
newrbe method– example: $\sigma = 0.5$



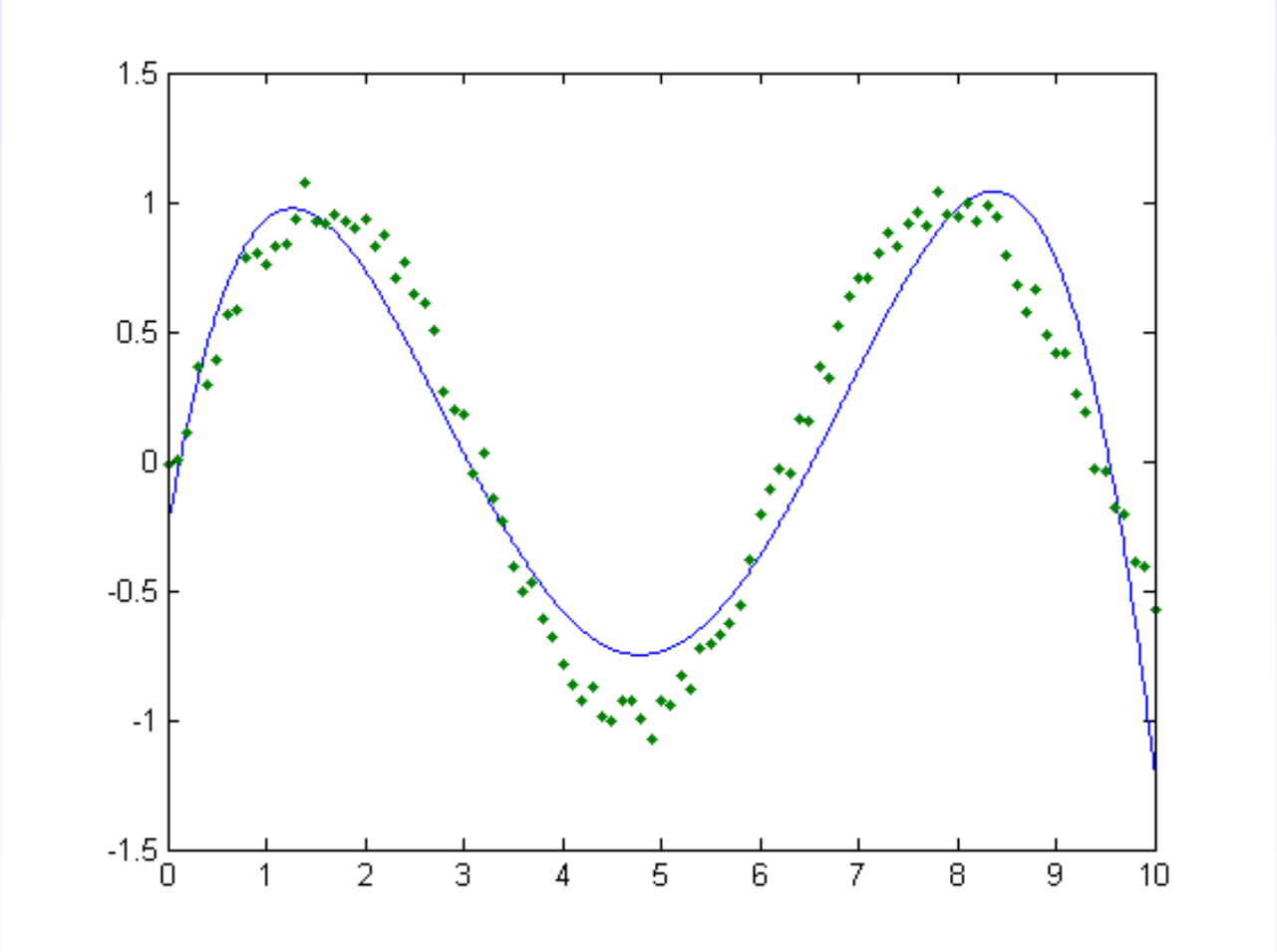
newrbe method– example: $\sigma = 1$



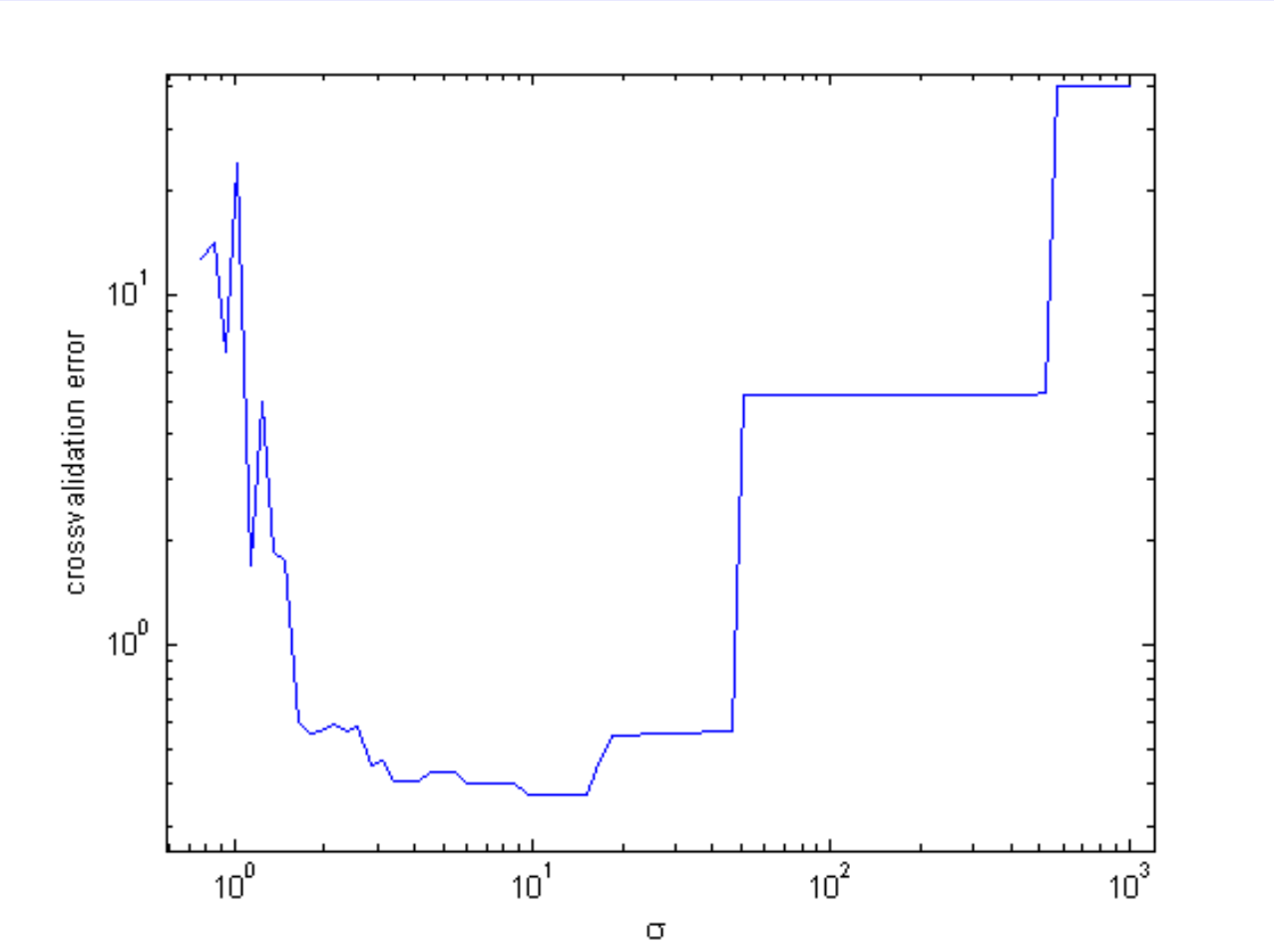
newrbe method– example: $\sigma = 10$



newrbe method– example: $\sigma = 100$



`newrbe` method – crossvalidation error



Crossvalidation error determined on the base of 'leave one out' method for different σ values.

Learning methods

Supervised learning

Learning data consist of pairs:

$$(\mathbf{x}_i, d_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data and $d_i \in \{0, 1\}$ or $d_i \in \mathbb{R}$ are connected with them, given output data.

Learning methods

Supervised learning

Learning data consist of pairs:

$$(\mathbf{x}_i, d_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data and $d_i \in \{0, 1\}$ or $d_i \in \mathbb{R}$ are connected with them, given output data.

Unsupervised learning

Learning data has a form of the set:

$$(\mathbf{x}_i), \qquad i = 1 \ldots L,$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots x_{in} \in \mathbb{R}^n)$ are input data.

Unsupervised learning

- The given output is not required in samples.
- The network should 'discover' (without external help) patterns, features, interdependencies, arrangement of the input data, and then provide this information in a properly encoded form in the output.
- Redundancy of the data is required for the effective unsupervised learning.
- During learning, the network usually tries to divide the learning set into classes, according to certain common features of samples. The network should be able to identify such features in any presented input vector.

Unsupervised learning – tasks

The kind of task salved by the network depends on the network structure and its learning method.

- Similarity determination – realised by the network with a single output neuron whose value shows how much an input sample is similar to the pattern memorised during the learning process.
- Classification – the network has a number of output neurons equal to the amount of recognised classes. The input sample is assigned to a specific class. The task of the learning process is to divide samples similar to each other into classes and to assign each class to the one output neuron.
- The search for the archetyp – the network works in a similar way to the classification case, but on the output we get a pattern typical to the class.

Unsupervised learning – tasks

The kind of task salved by the network depends on the network structure and its learning method.

- Coding – the output vector is the coded version of the pattern typical to the class.
- Principal component analysis (PCA) – the network has a certain number of output neurons, and each neuron specifies the similarity of the input sample with respect to the principal components (most important features).
- Creating maps of features – output layer neurons are geometrically arranged (e.g. in the form of a 2-dimensional array). During the presentation of the input sample only 1 output is activated. The idea of its operation assumes that similar input samples generate the activity of geometrically close neurons. So, the output layer is a kind of the topographic map of the input data features.

Hebb rule

One of the first rules of unsupervised neuron learning was described by Donald Hebb. This rule was based on the phenomenon of the formation of conditioned reflexes which are acquired by humans and animals.

If a neuron A is cyclically stimulated by a neuron B, then it becomes more sensitive to stimulation.

Hebb rule

One of the first rules of unsupervised neuron learning was described by Donald Hebb. This rule was based on the phenomenon of the formation of conditioned reflexes which are acquired by humans and animals.

If a neuron A is cyclically stimulated by a neuron B, then it becomes more sensitive to stimulation.

If we denote by ϕ_A and ϕ_B the states of activity of the neurons A and B, and by w_{AB} – weight of its connection, then the above rule can be described in the form:

$$w_{AB}(k+1) = w_{AB}(k) + \eta \cdot \phi_A \cdot \phi_B$$

Data normalisation

Normalisation

Scaling of the each input in such a way that it takes the value from the same and assumed interval $[l, u]$. Most commonly, the normalisation scales values to the range $[0, 1]$ or $[-1, 1]$.

$$x_{norm} = (x - x_{min}) \cdot \frac{u - l}{x_{max} - x_{min}} + l$$

where:

- x – original value,
- x_{norm} – value after normalisation,
- x_{min}, x_{max} – minimum and maximum value of the data.

Data normalisation

```
% normalisation of the vector x
% new_min, new_max - new range of the vector values (by default [0,1])
% x_norm - vector after normalisation
% min_x, max_x - minimum and maximum value in the original vector

function [x_norm, min_x, max_x] = normalisation(x, new_min, new_max)

if nargin < 2
   new_min = 0;
   new_max = 1;
end
if nargout > 1
   min_x = min(x);
   max_x = max(x);
end

x_norm = (x - min(x))*(new_max-new_min)/(max(x)-min(x)) + new_min;
```

```
>> x = 0:10
x =
     0     1     2     3     4     5     6     7     8     9    10

>> x1 = normalisation(x)
x1 =
     0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.0

>> x2 = normalisation(x,-10,10)
x2 =
   -10    -8    -6    -4    -2     0     2     4     6     8    10

>> [x2, x_low, x_high] = normalisation(x,-100,100)
x2 =
  -100   -80   -60   -40   -20     0    20    40    60    80   100
x_low =
     0
x_high =
    10
```

Normalisation of the Z axis

In unsupervised learning, a frequent requirement is the same length of training vectors.

Normalisation of the Z axis

In unsupervised learning, a frequent requirement is the same length of training vectors.

1. In the first step, all inputs are scaled to the interval $[-1, 1]$.
2. In the second step, we add an additional variable whose value is a function of the real input values. This value is calculated in such a way that after normalisation, the vector length is equal to 1.

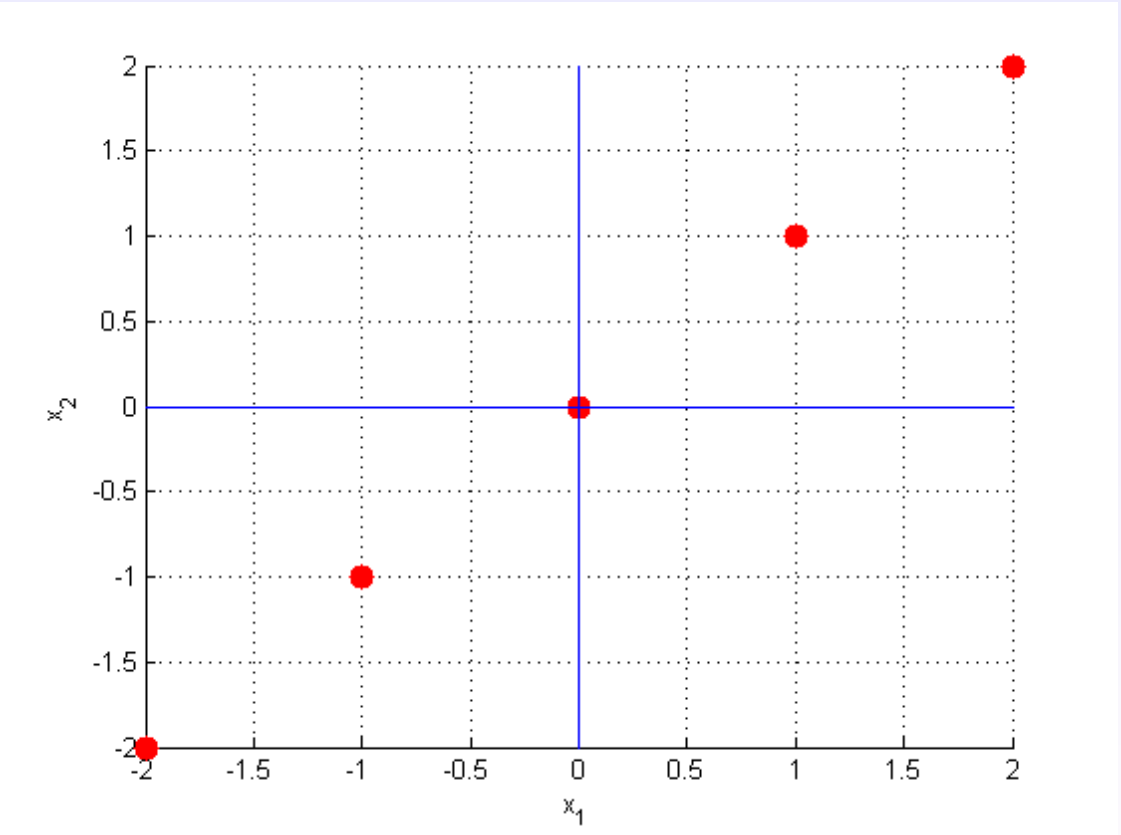
We calculate:

$$\hat{x}_i = f \cdot x_i, \qquad i = 1 \ldots n, \qquad f = \frac{1}{\sqrt{n}}$$

$$\hat{x}_{n+1} = f \cdot \sqrt{n - d^2}, \qquad d = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Normalisation of the Z axis – example

x_1	x_2
-2	-2
-1	-1
0	0
1	1
2	2



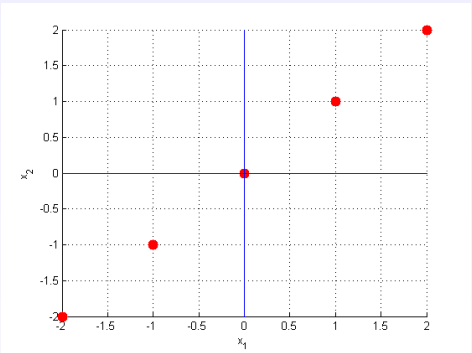
Normalisation of the Z axis – example

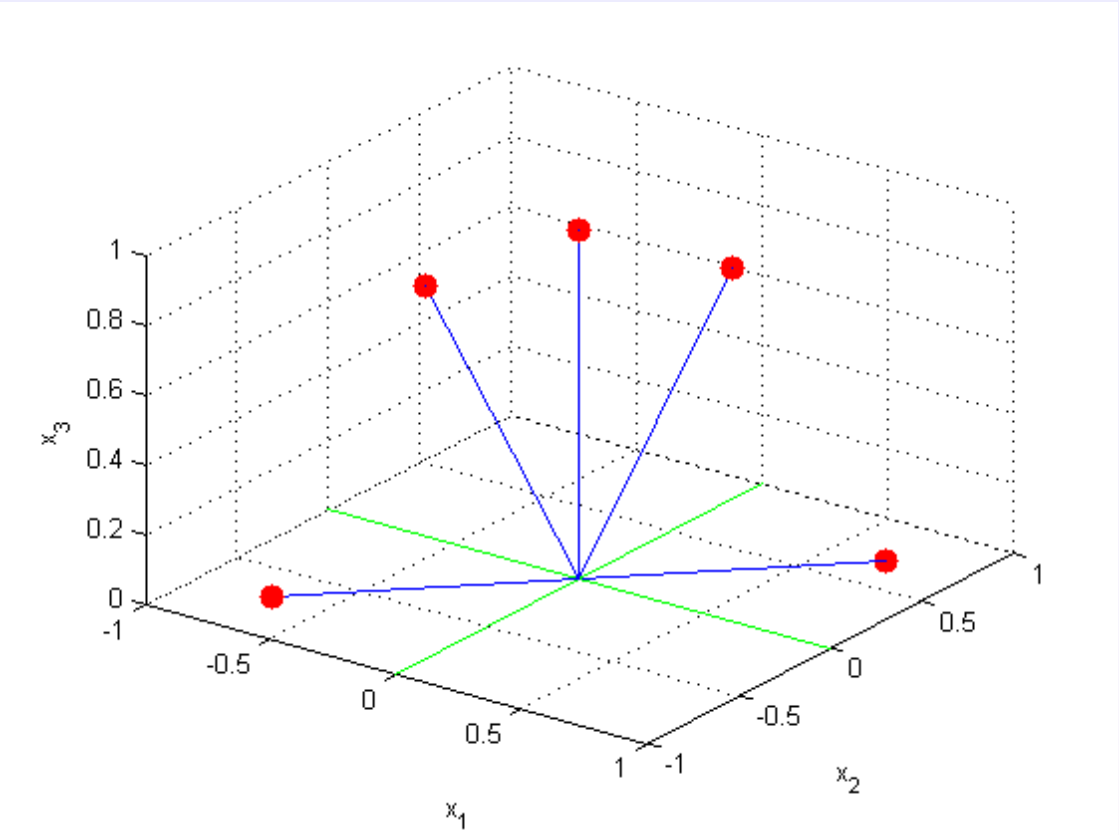
$$\hat{x}_i = f \cdot x_i, \qquad i = 1 \ldots n, \qquad f = \frac{1}{\sqrt{n}}$$

$$\hat{x}_{n+1} = f \cdot \sqrt{n - d^2}, \qquad d = \sqrt{\sum_{i=1}^{n} x_i^2}$$

x_1	x_2	x_1^{norm}	x_2^{norm}	d^2	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$
-2	-2	-1	-1	2	$-\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$	0
-1	-1	-0.5	-0.5	$\frac{1}{2}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{\sqrt{3}}{2}$
0	0	0	0	0	0	0	1
1	1	0.5	0.5	$\frac{1}{2}$	$\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	$\frac{\sqrt{3}}{2}$
2	2	1	1	2	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	0

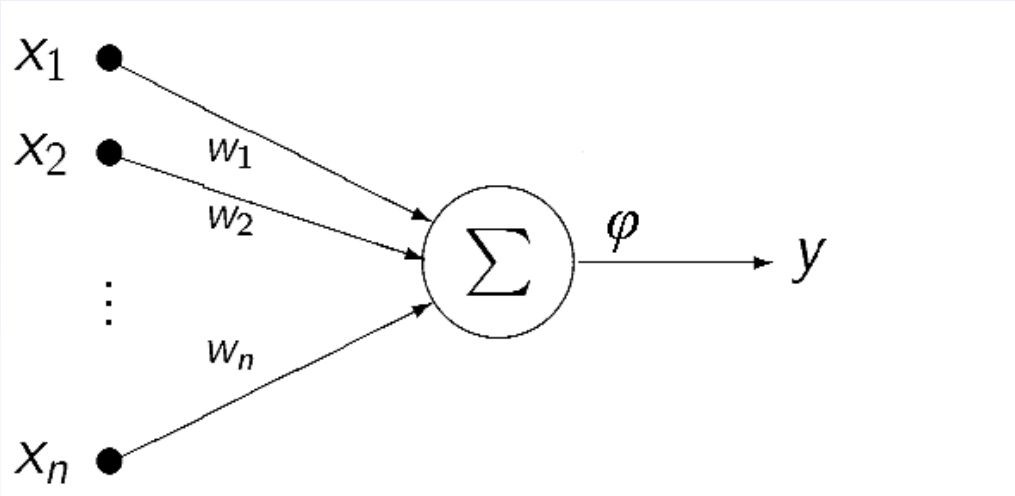
Normalisation of the Z axis – example





Grossberg rule

- Let's assume that we have only one linear neuron.
- Next, let's assume that we have only one learning sample **x**.

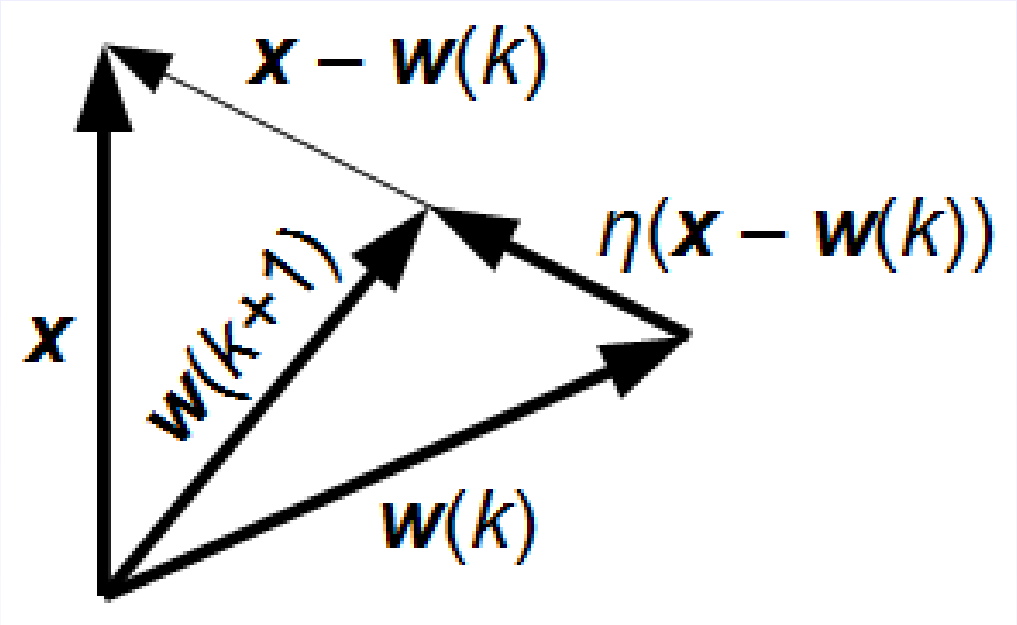


Weights will be modified according to the rule:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$

Grossberg rule

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$



Grossberg rule

The neuron output is the scalar product of the input **x** and the vector of weights **w**.

$$y = \mathbf{w}^T \cdot \mathbf{x} = |\mathbf{w}| \cdot |\mathbf{x}| \cdot \cos(\alpha)$$

where: α is the angle between vectors **x** and **w**.

For normalised vectors **x** and **w**, an output depends only on the angle α.

Grossberg rule

The neuron output is the scalar product of the input **x** and the vector of weights **w**.

$$y = \mathbf{w}^T \cdot \mathbf{x} = |\mathbf{w}| \cdot |\mathbf{x}| \cdot \cos(\alpha)$$

where: α is the angle between vectors **x** and **w**.

For normalised vectors **x** and **w**, an output depends only on the angle α.

- The neuron is most stimulated, when the input vector **x** is similar to the weights vector **w**.
- During learning, weights become similar to the presented sample.

Grossberg rule

- Let's assume that we have only one linear neuron.
- We have a lot of similar learning samples $\mathbf{x}$.

Weights will be modified according to the rule:

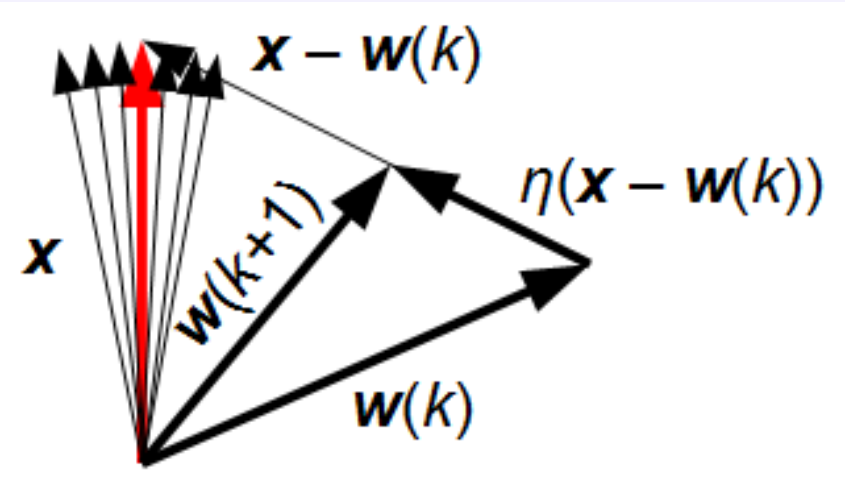
$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$

Grossberg rule

- Let's assume that we have only one linear neuron.
- We have a lot of similar learning samples **x**.

Weights will be modified according to the rule:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \cdot (\mathbf{x} - \mathbf{w}(k))$$

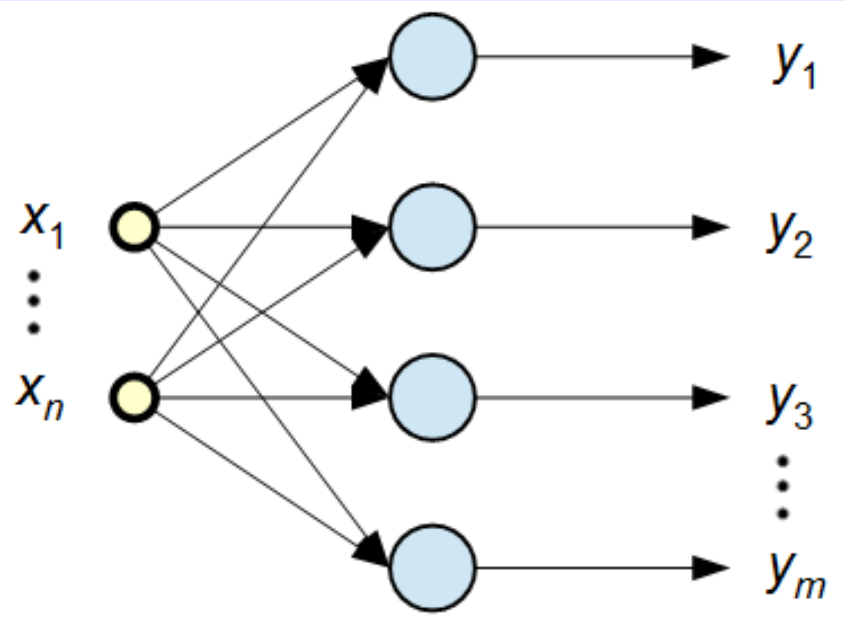


Grossberg rule

- During learning, weights become similar to the average presented sample.
- The neuron is most stimulated, when the input vector $\mathbf{x}$ is similar to the weights vector $\mathbf{w}$.

Trained neuron is able to determine the similarity of the input sample to the memorised pattern.

Self-Organising Networks



- The network consists of a single layer of linear neurons.
- The weights vector has the same size as the input vector.
- Each neuron represents one class (cluster) of data.
- During learning, neurons compete with each other for the modification of weights.

Competitive learning – WTA

Weights of the most stimulated neuron will be modified according to Grossberg rule:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta(k) \cdot (\mathbf{x} - \mathbf{w}(k))$$

In this way, each neuron will be learnt only with these samples whose values are similar to its weights, and it means that they belong to the class it represents.

Such strategy will be called Winner Takes All (WTA) learning strategy.

Competitive learning – WTA

- In each learning step, only weights of the winner neuron $\mathbf{w}^{j^*}$ are modified.
- The winner neuron is determined as:

$$j^* = \arg \min_{j=1 \ldots m} d(\mathbf{x}, \mathbf{w}^j)$$

where: d – is a measure of the distance between vectors $\mathbf{x}$ and $\mathbf{w}^j$.

Distance measures

- Scalar product:

$$d(\mathbf{x}, \mathbf{w}) = 1 - \mathbf{x} \cdot \mathbf{w} = 1 - ||\mathbf{x}|| \cdot ||\mathbf{w}|| \cdot \cos(\mathbf{x}, \mathbf{w})$$

- Euclidean measure:

$$d(\mathbf{x}, \mathbf{w}) = ||\mathbf{x} - \mathbf{w}|| = \sqrt{\sum_{i=1}^{n} (x_i - w_i)^2}$$

- L_1 (Manhattan) distance measure:

$$d(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{n} |x_i - w_i|$$

- L_∞ distance measure:

$$d(\mathbf{x}, \mathbf{w}) = \max_i |x_i - w_i|$$

Rate of learning coefficient η

Rate of learning coefficient η should be decreased during learning.

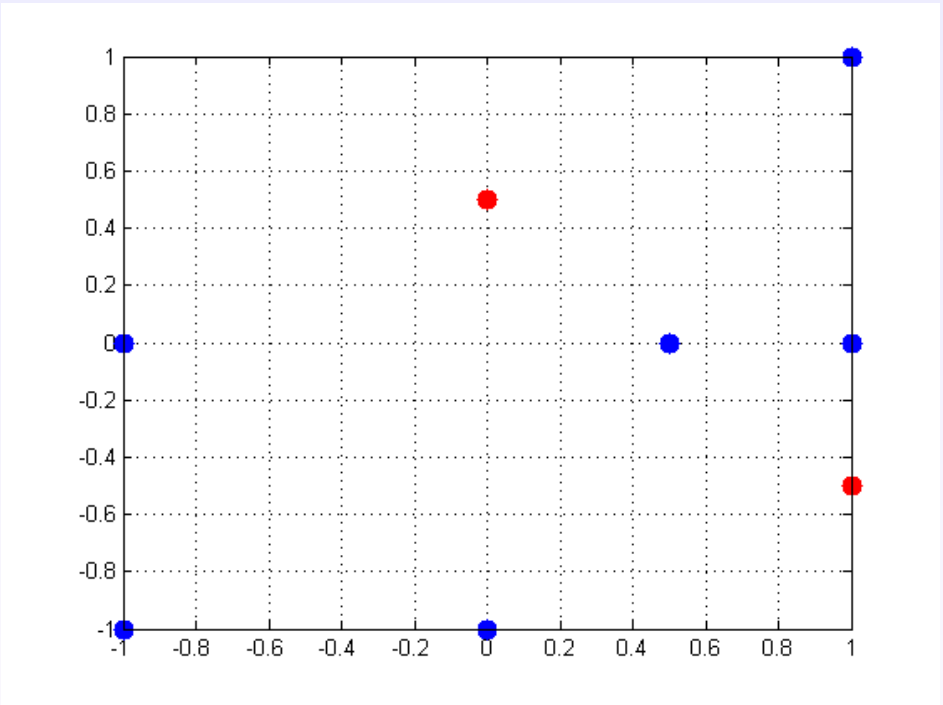
Linearly:

$$\eta(k) = \eta_{min} + (\eta_{max} - \eta_{min}) \frac{k_{max} - k}{k_{max}}$$

Exponentially:

$$\eta(k) = \eta_{max} \left(\frac{\eta_{min}}{\eta_{max}} \right)^{\frac{k}{k_{max}}}$$

Competitive learning – example



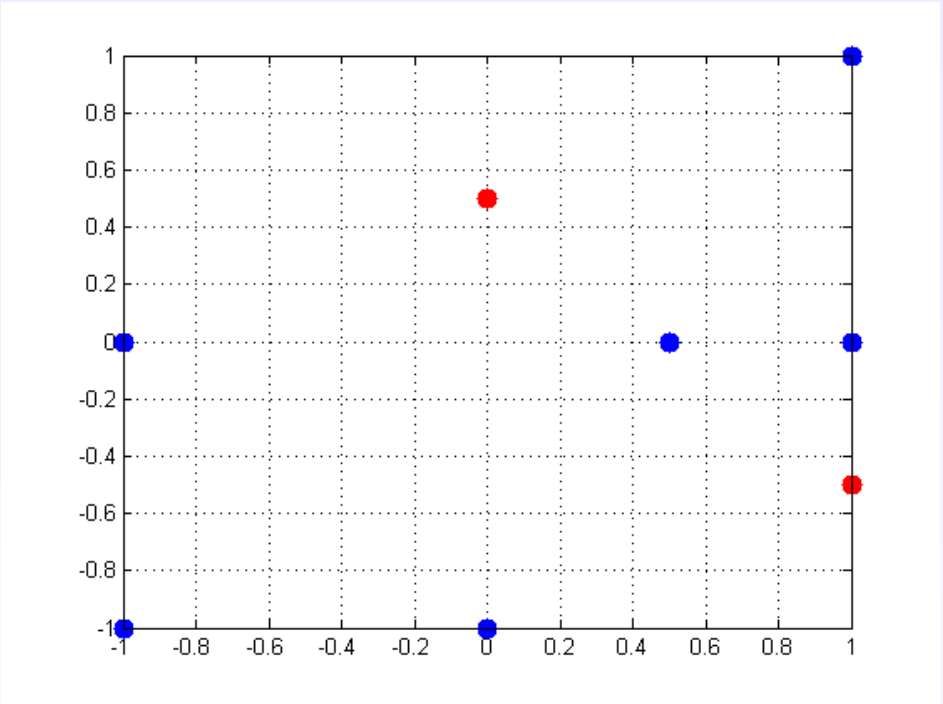
Learning data:

x_1	x_2
1	1
3	3
1	2
2.5	2
2	1
3	2

After normalisation:

x_1	x_2
-1	-1
1	1
-1	0
0.5	0
0	-1
1	0

Competitive learning – example



Data after normalisation:

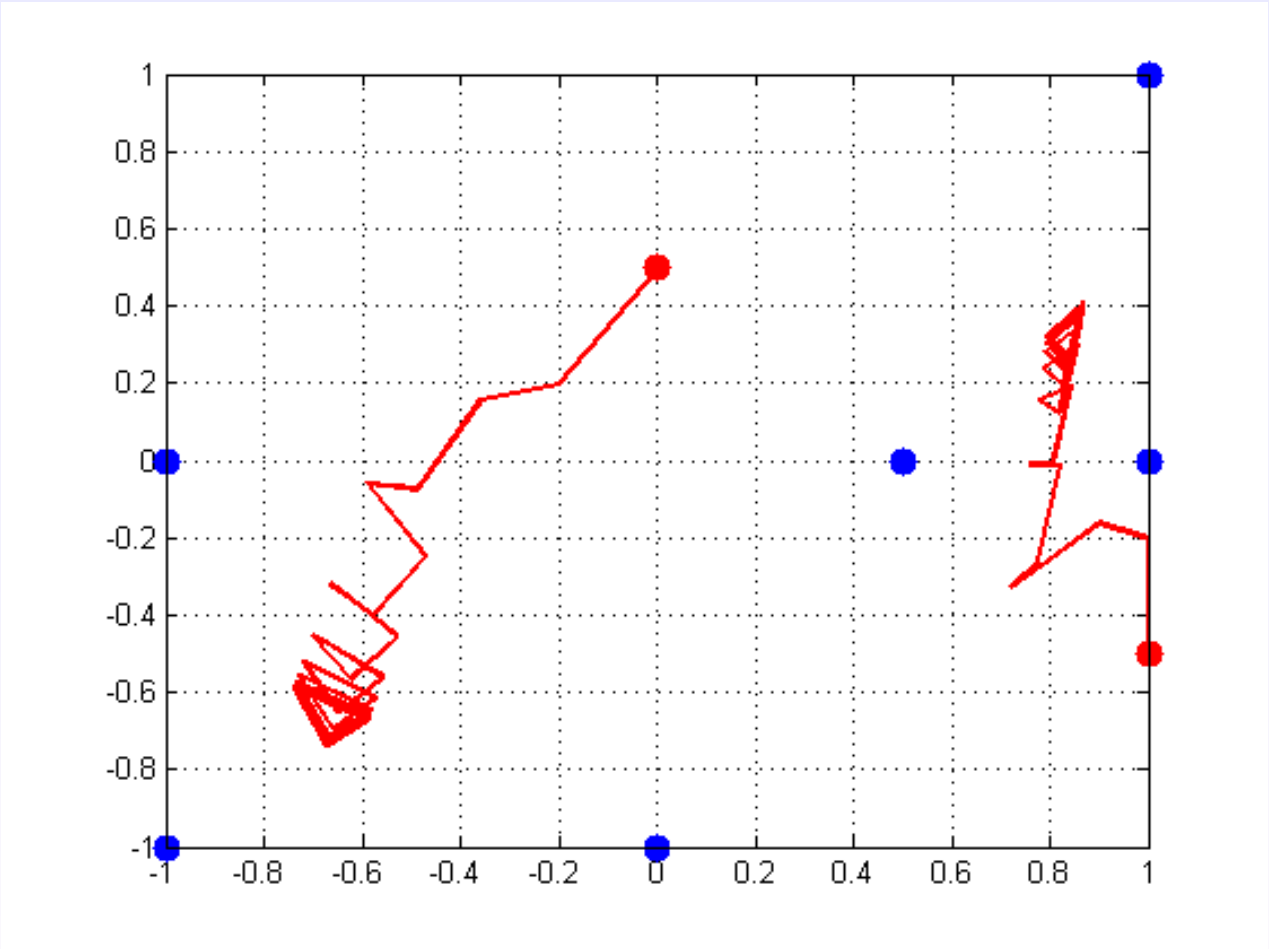
x_1	x_2
-1	-1
1	1
-1	0
0.5	0
0	-1
1	0

Initial weights:

$$\mathbf{w}^1(0) = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} \quad \mathbf{w}^2(0) = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

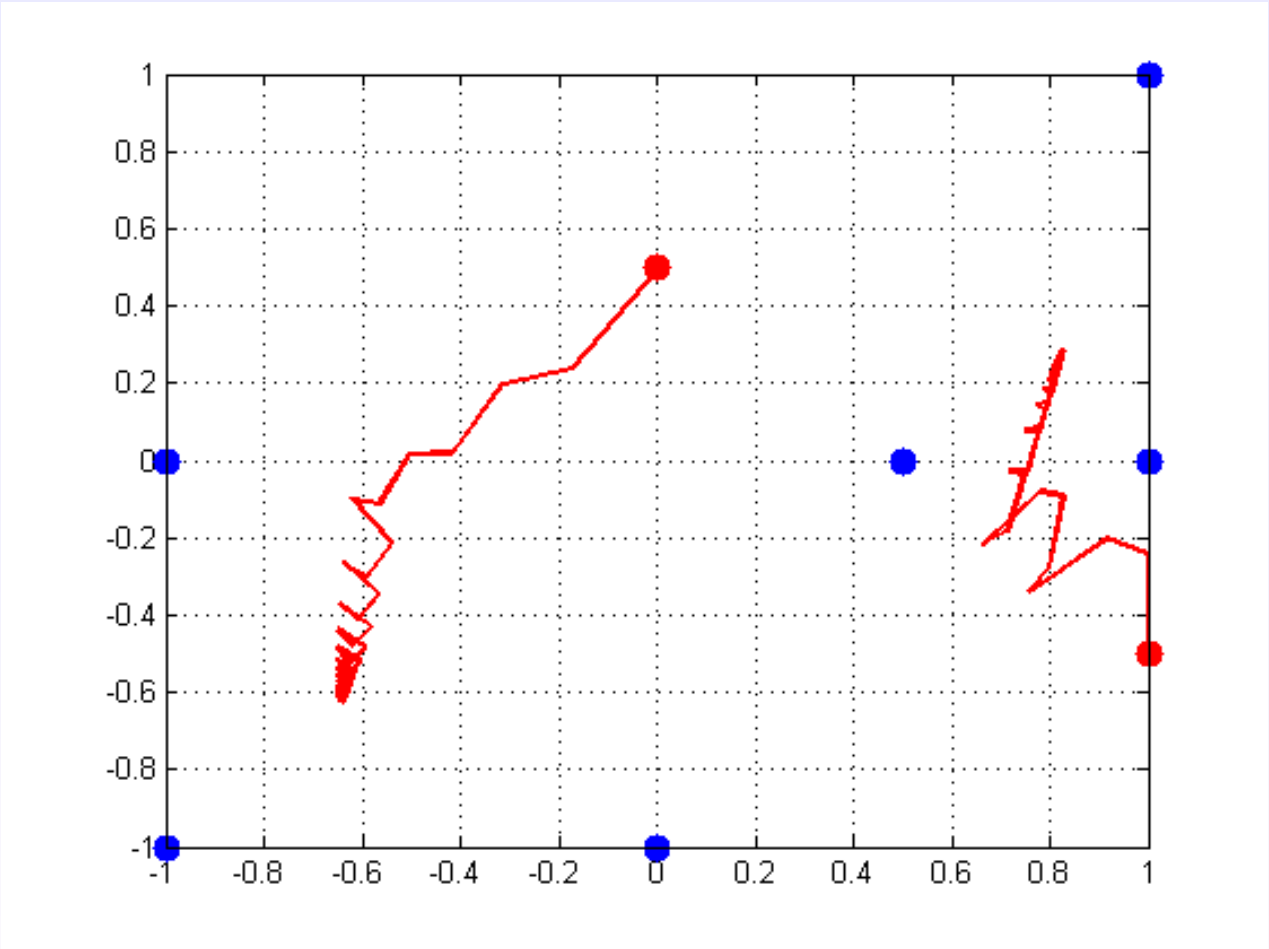
$$\eta = 0.2$$

Competitive learning – example



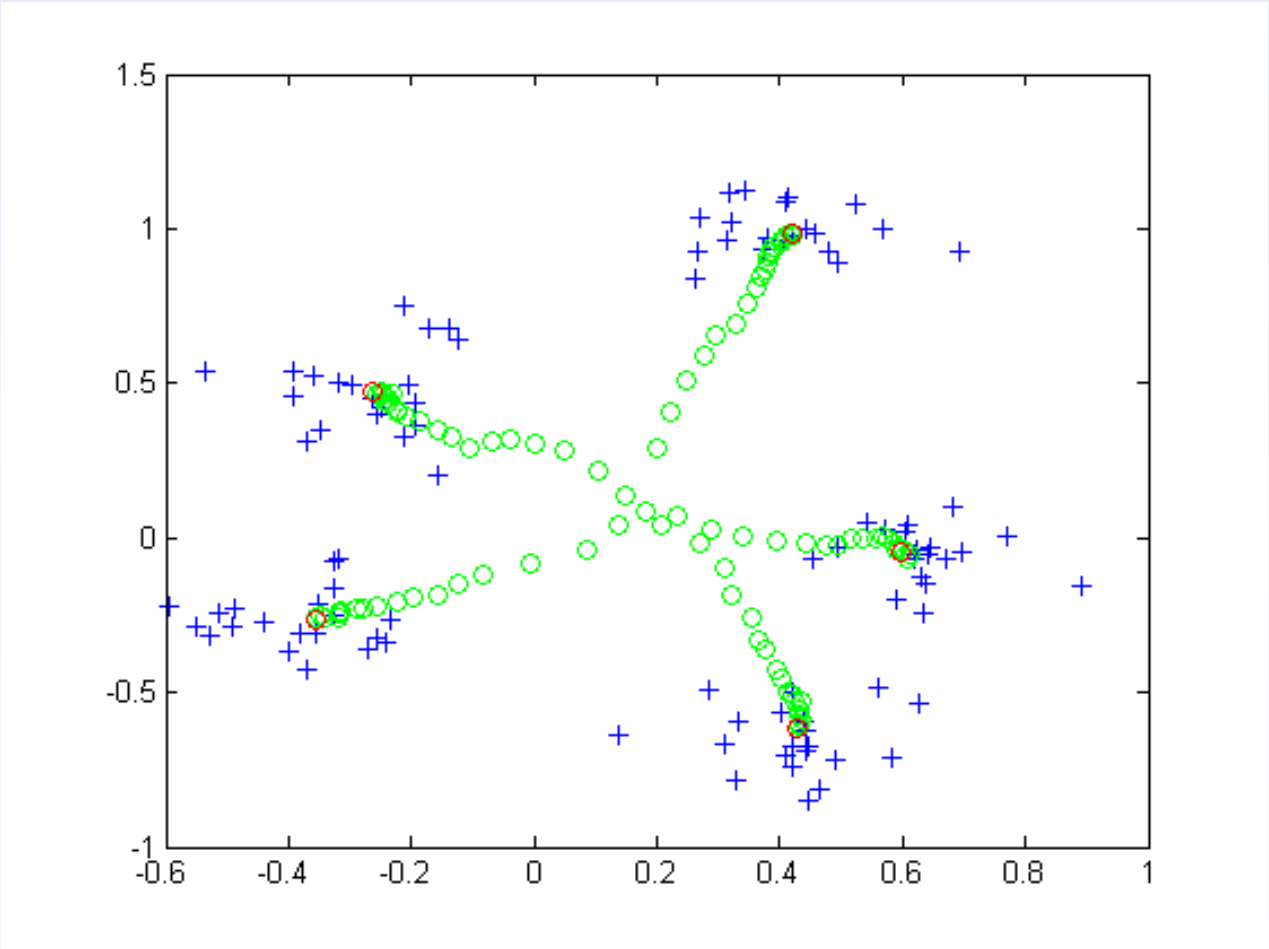
Learning with constant value of $\eta = 0.2$.

Competitive learning – example



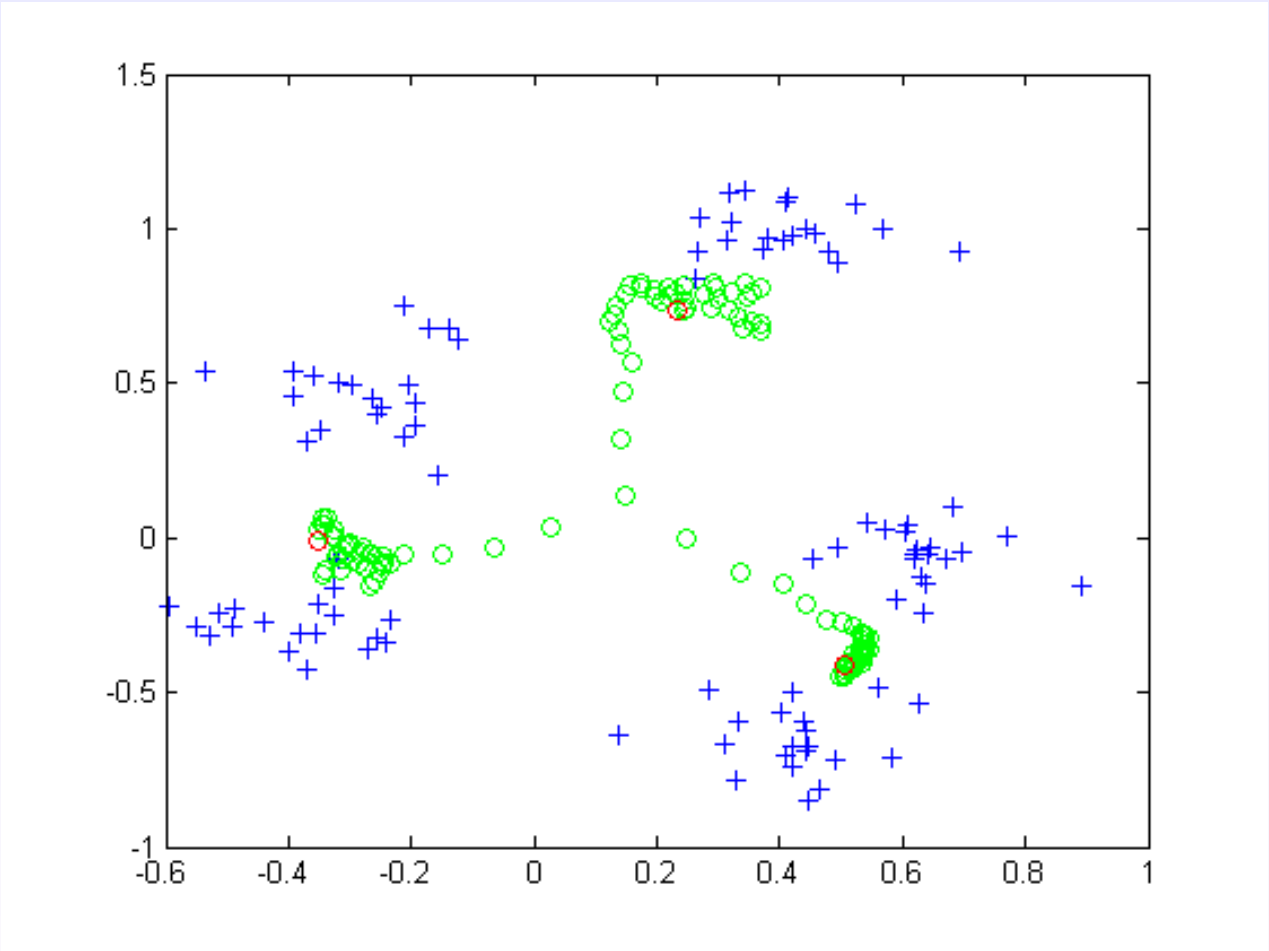
Learning with value of η decreasing from 0.2 to 0 during learning.

Competitive learning – example



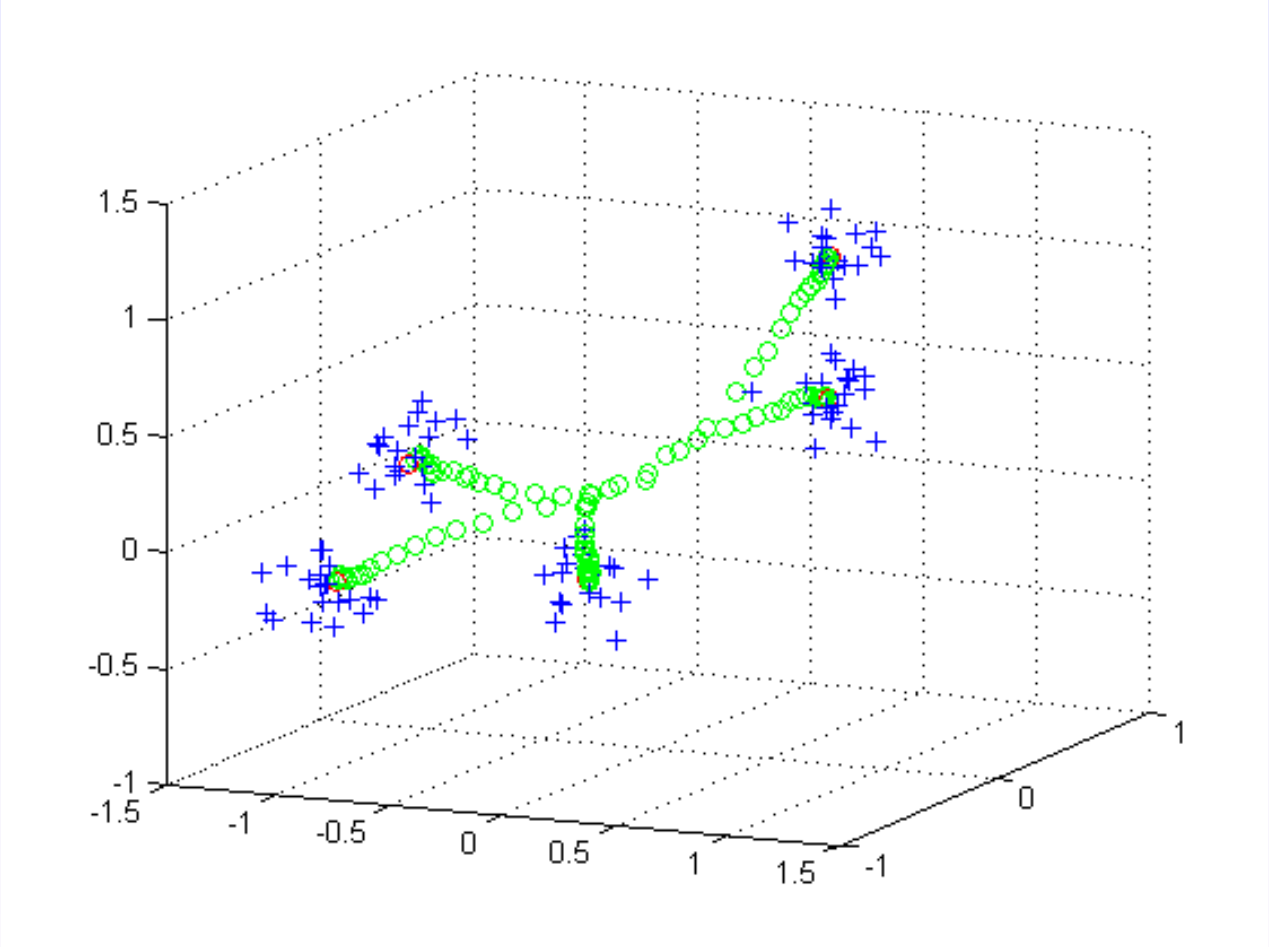
Learning of the network with 5 neurons.

Competitive learning – example



Learning of the network with 3 neurons.

Competitive learning – example



Learning of the network with 5 neurons.

Competitive learning – WTM

Using Kohonen rule, we modify the weights of all neurons:

$$\mathbf{w}^j(k+1) = \mathbf{w}^j(k) + \eta(k) \cdot G(\mathbf{w}^j, \mathbf{w}^{j^*}) \cdot (\mathbf{x} - \mathbf{w}(k))$$

Such strategy will be called Winner Takes Most (WTM) learning strategy.

$G(\mathbf{w}^j, \mathbf{w}^{j^*})$ – the neighbourhood function which determines the degree of similarity between the neuron no j to the winner neuron no j^*.

Neighbourhood functions

WTA

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \begin{cases} 1 & \text{for} \quad j = j^* \\ 0 & \text{for} \quad \text{other} \end{cases}$$

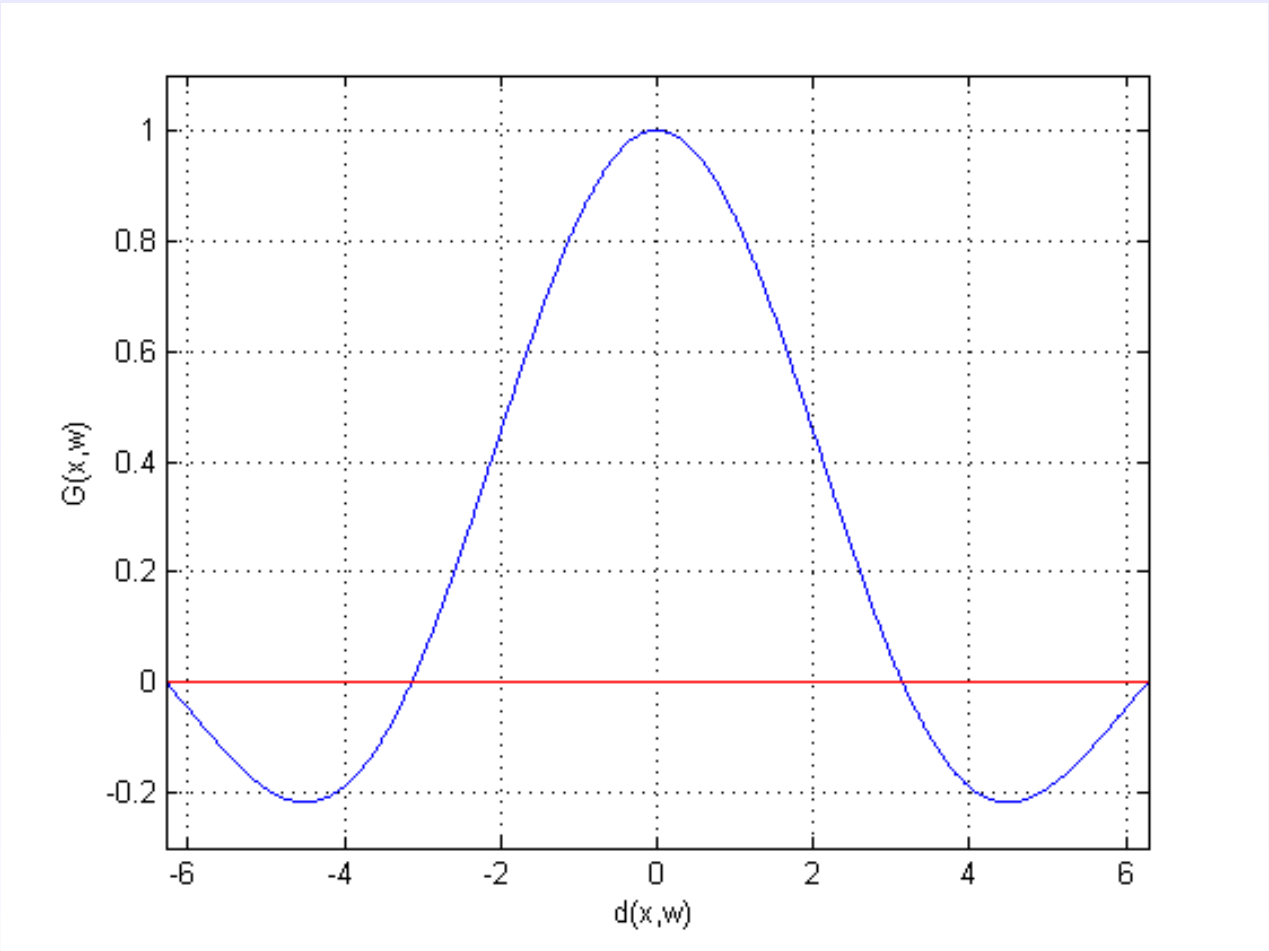
Gauss neighbourhood

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \exp\left(-\frac{d^2(\mathbf{w}^j, \mathbf{w}^{j^*})}{2\lambda^2}\right)$$

'Mexican hat' function

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \begin{cases} 1 & \text{for} \quad d(\mathbf{w}^j, \mathbf{w}^{j^*}) = 0 \\ \frac{\sin(\lambda \cdot d(\mathbf{w}^j, \mathbf{w}^{j^*}))}{\lambda \cdot d(\mathbf{w}^j, \mathbf{w}^{j^*}} & \text{for} \quad |d(\mathbf{w}^j, \mathbf{w}^{j^*})| \in (0, 2\pi/\lambda) \\ 0 & \text{for} \quad \text{other} \end{cases}$$

Neighbourhood functions



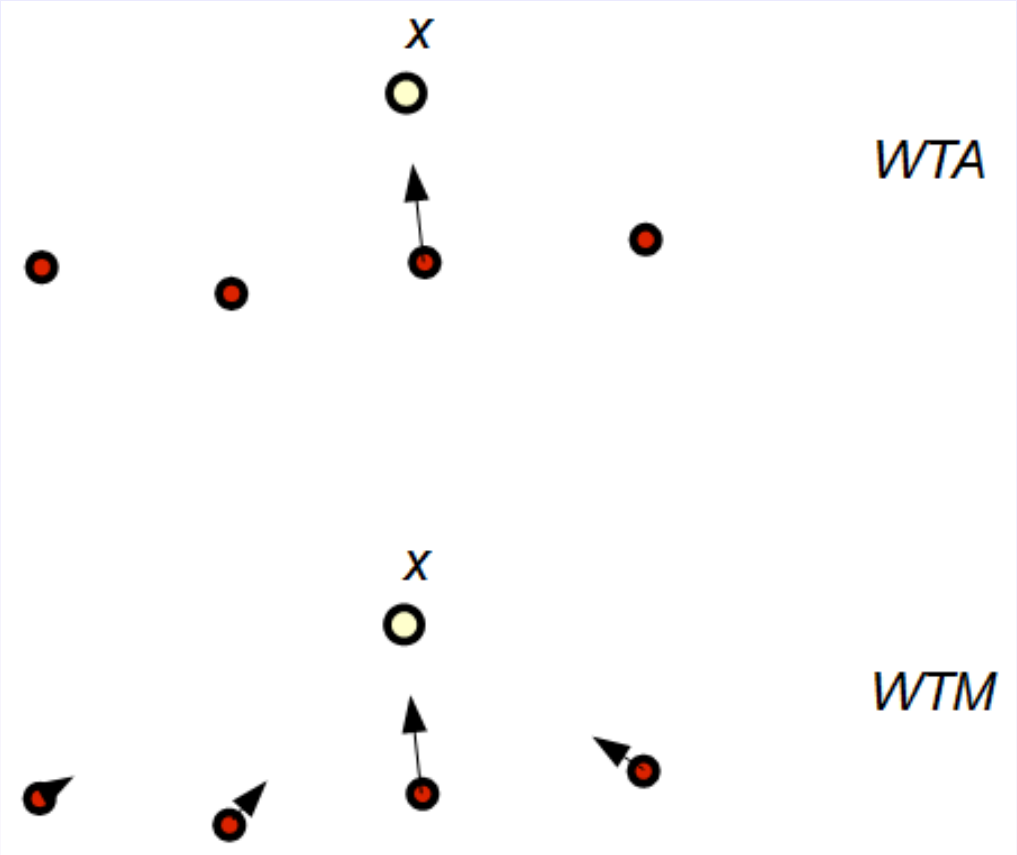
'Mexican hat' function.

Neighbourhood functions

- The width of the neighbourhood function should decrease during learning.
- Decreasing of the λ value can be realised e.g. exponentially:

$$\lambda(k) = \lambda_{max} \left(\frac{\lambda_{min}}{\lambda_{max}} \right)^{\frac{k}{k_{max}}}$$

WTA – WTM



Changing of the position of neurons in the WTA and WTM strategy.

Kohonen networks

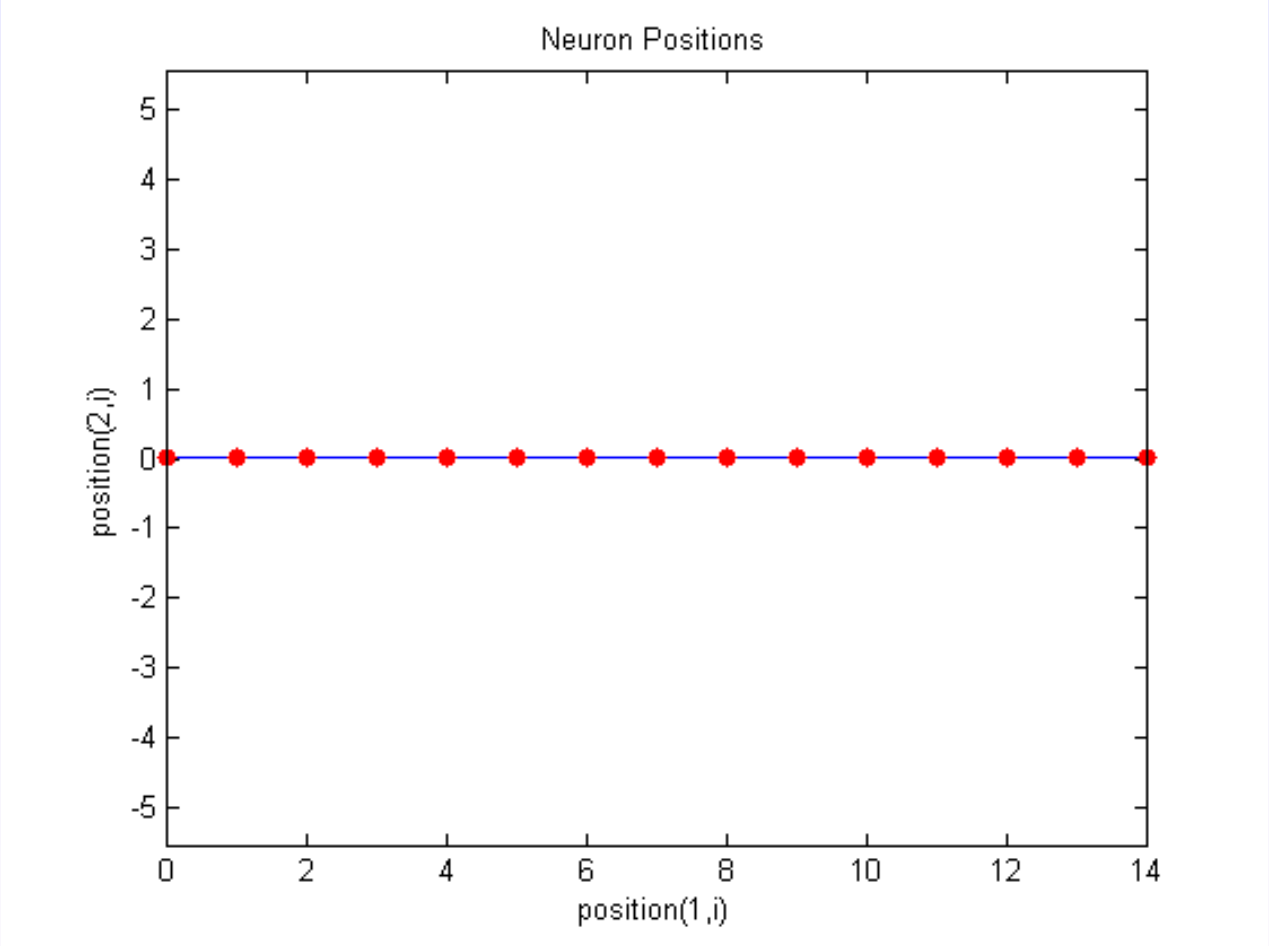
Topology

Graph structure defining arrangement and neighbourhood of neurons.

Similarity measures

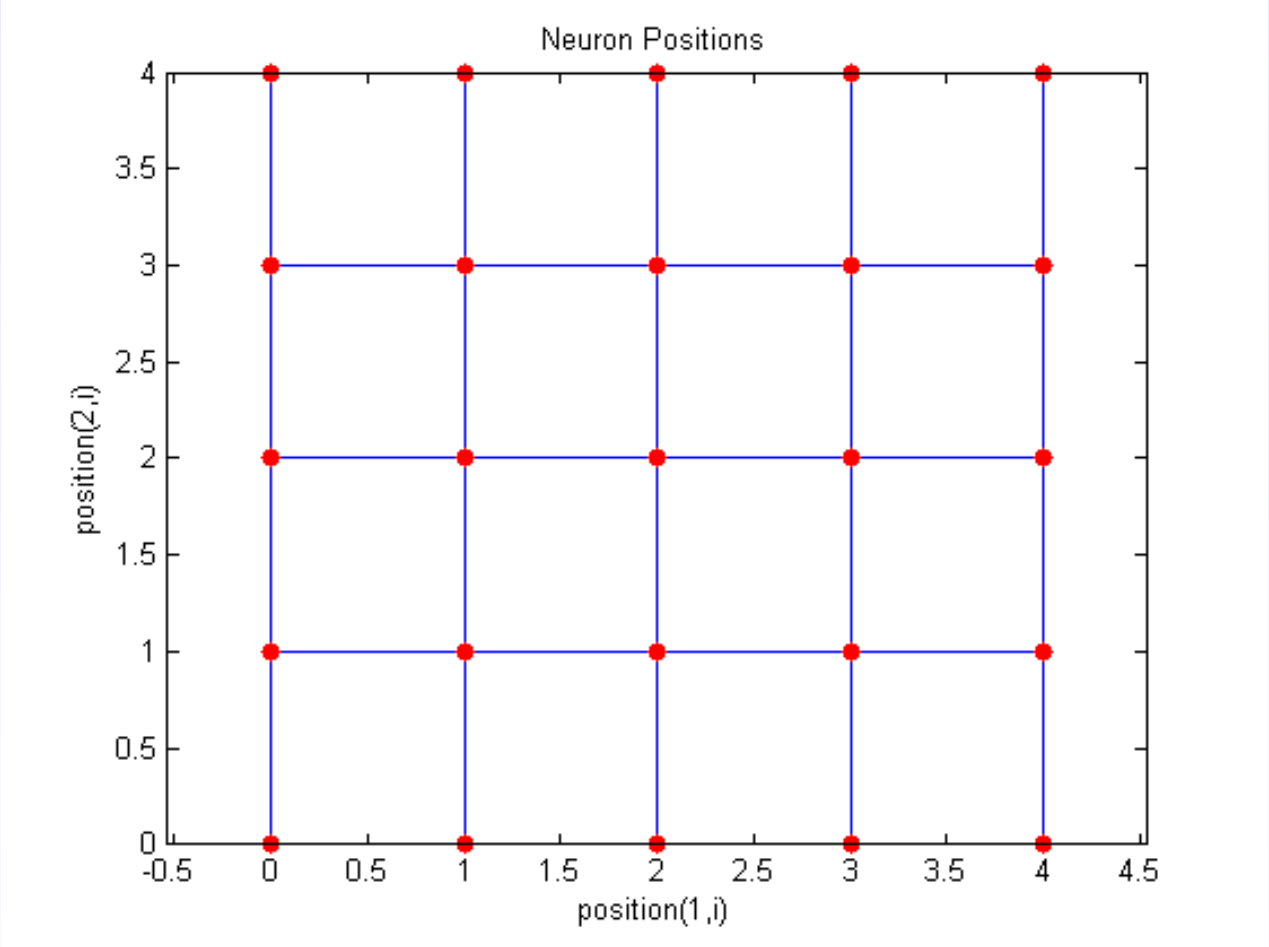
- Metrical – similarity defines the geometry of the input space.
- Topological – similarity of vectors is defined by the network topology.

Topology examples



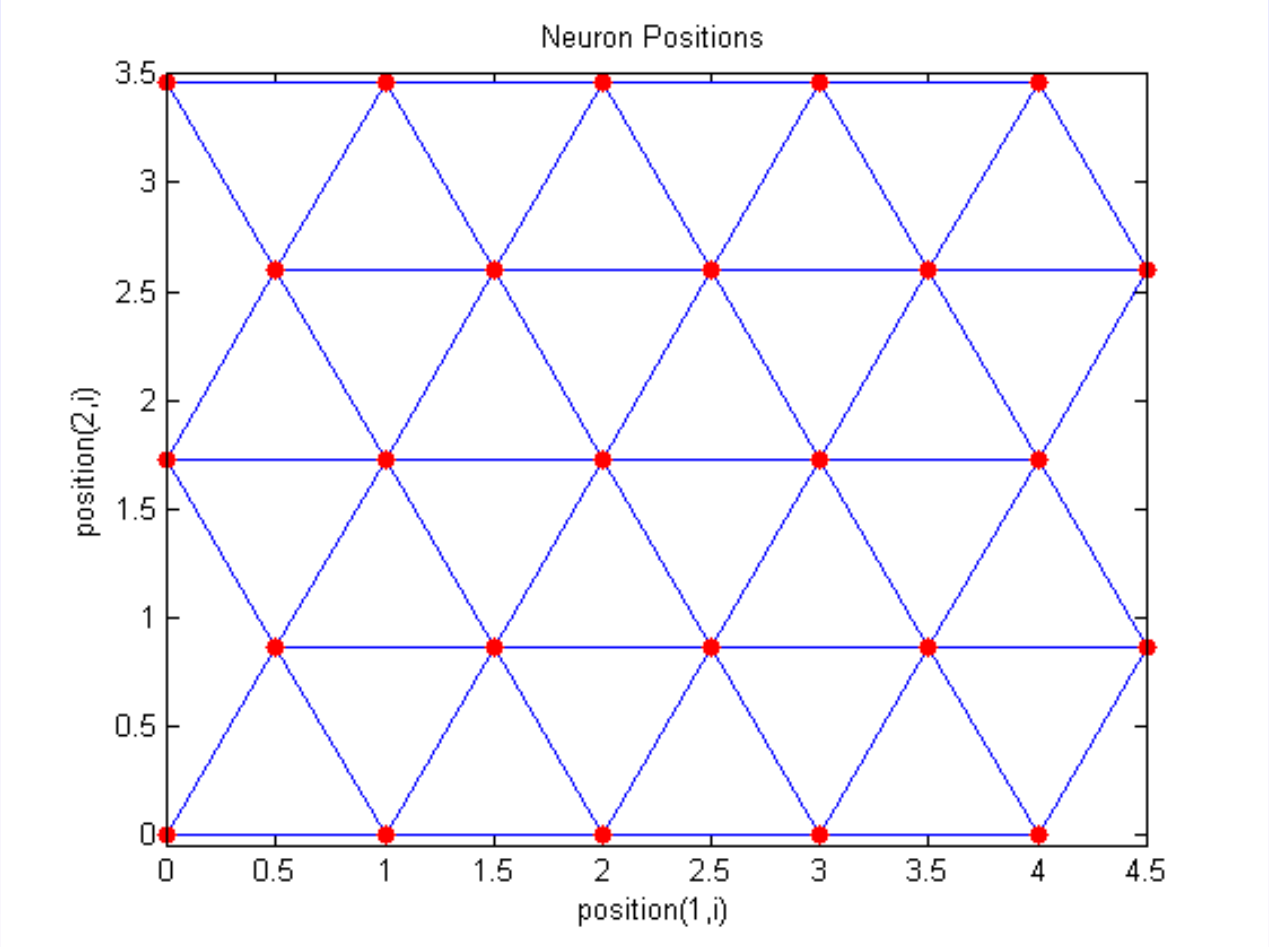
One dimensional – linear.

Topology examples



Two dimensional – rectangular.

Topology examples



Two dimensional – hexagonal.

Kohonen networks

When the topology is imposed onto the network, we consider the distance between neurons in the graph defining the topology, during determination of the neighbourhood.

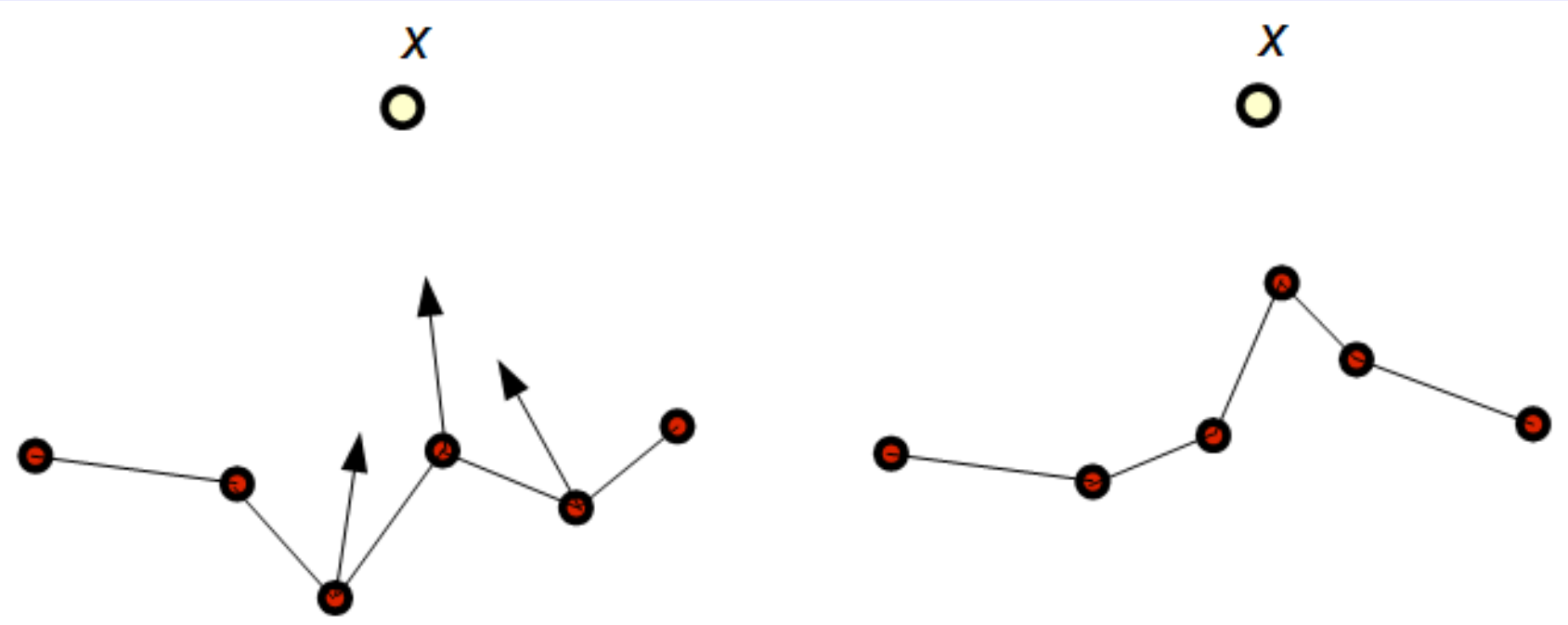
Rectangular neighbourhood

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \begin{cases} 1 & \text{for} \quad d(j, j^*) \leq \lambda \\ 0 & \text{for} \quad \text{other} \end{cases}$$

Gauss neighbourhood

$$G(\mathbf{w}^j, \mathbf{w}^{j^*}) = \exp\left(-\frac{d^2(j, j^*)}{2\lambda^2}\right)$$

Kohonen networks



Changing of neurons positions for the network with linear topology and width of a rectangular neighbourhood equal to $\lambda = 1$.

Kohonen networks

- The network with topology create so-called feature maps.
- After presentation of the input sample only 1 output is activated.
- Similar input samples should generate activity of close (in the graph) neurons.
- The output layer is thus a kind of topographic map of input data features.

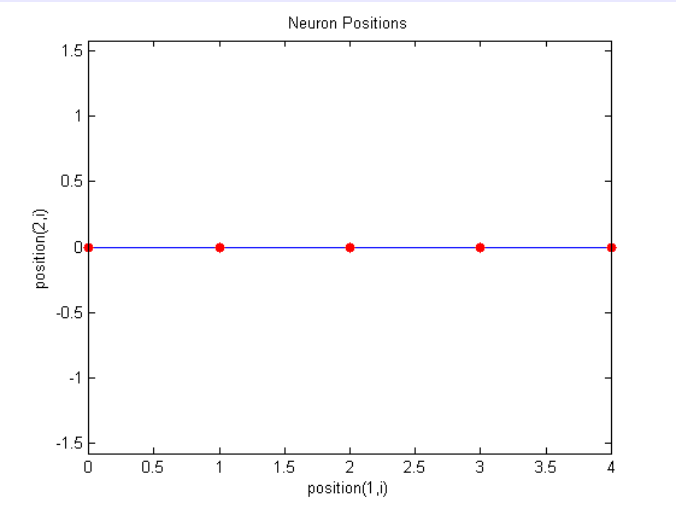
The quality of self-organising networks

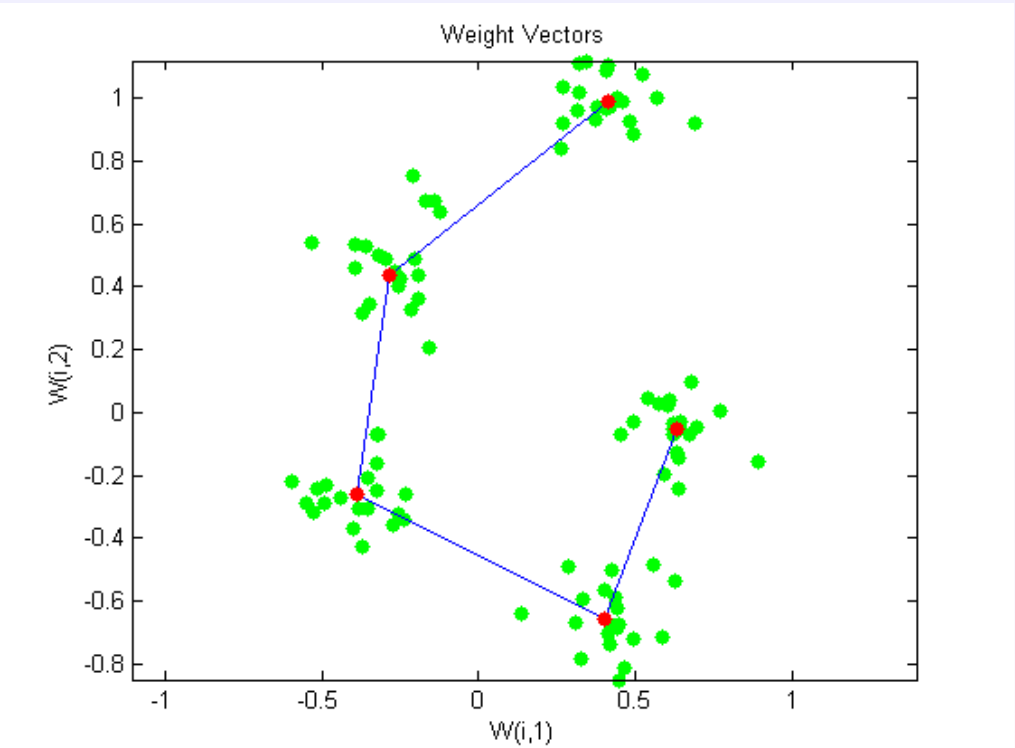
The quality of the self-organising network can be determined by formula:

$$q = \sum_{p=1}^{L} ||\mathbf{x}^p - \mathbf{w}^{j^*}||$$

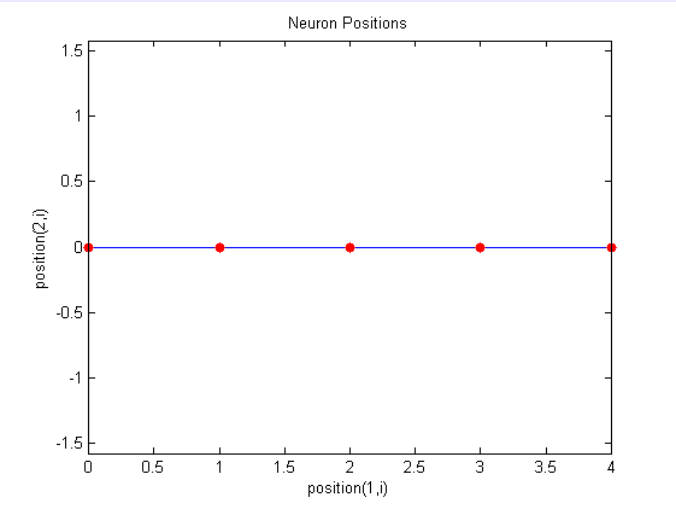
where: $\mathbf{w}^{j^*}$ – weights of the winner neuron for the sample $\mathbf{x}$.

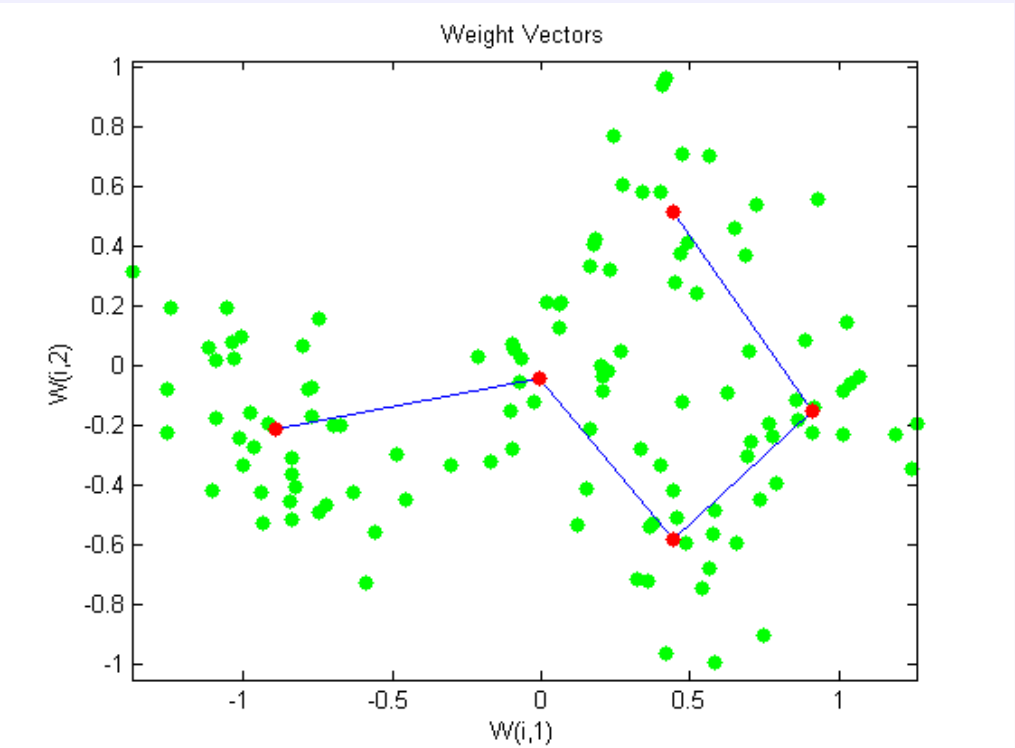
Kohonen networks – examples



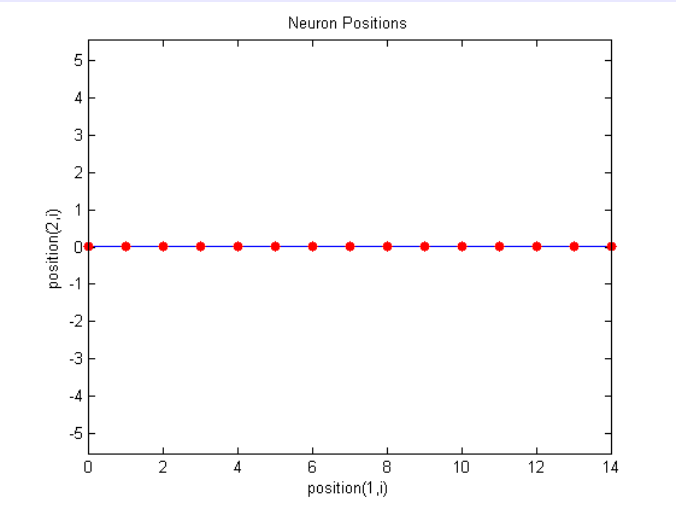


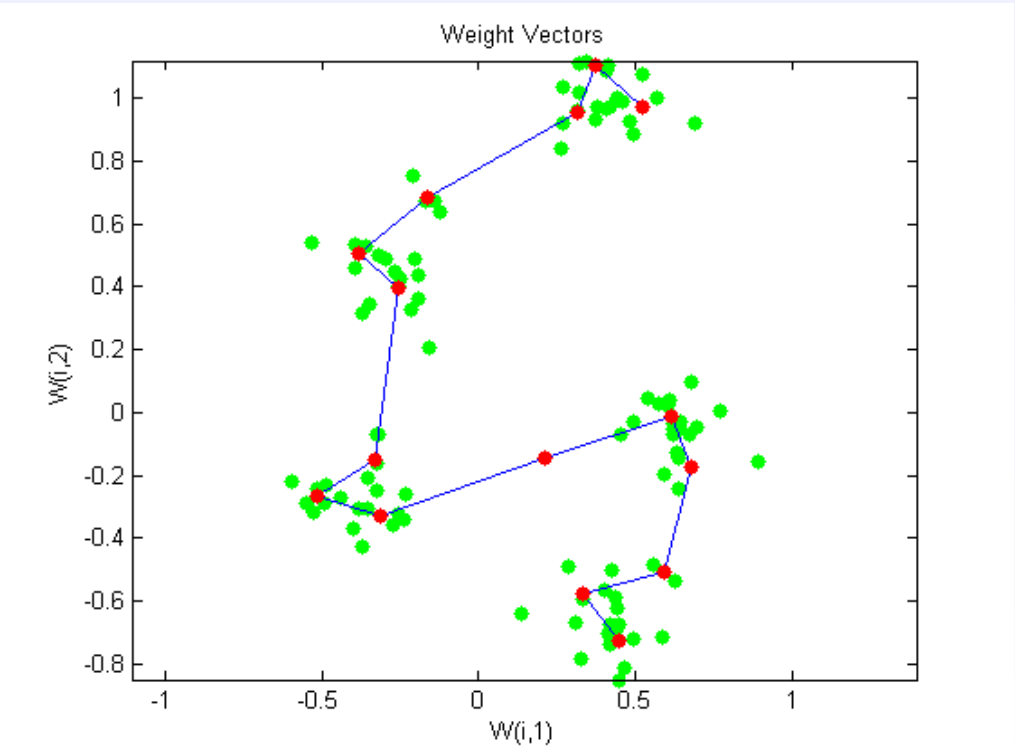
Kohonen networks – examples



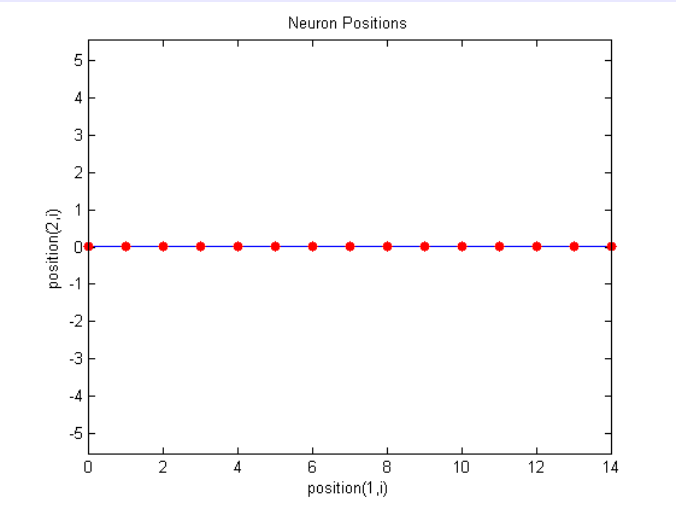


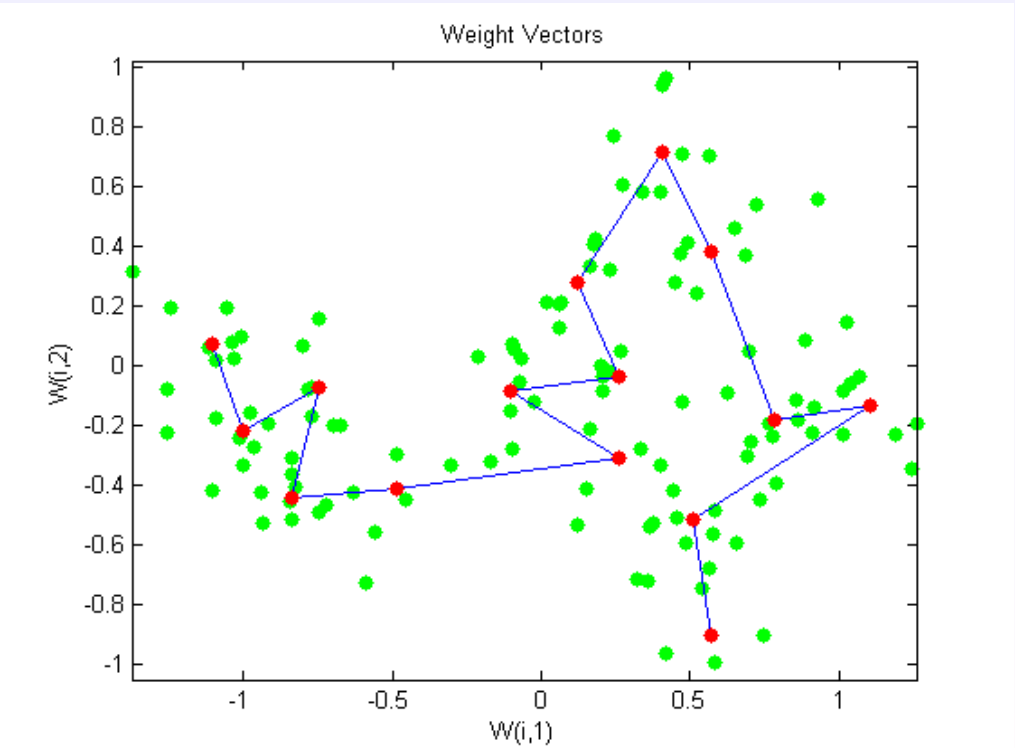
Kohonen networks – examples



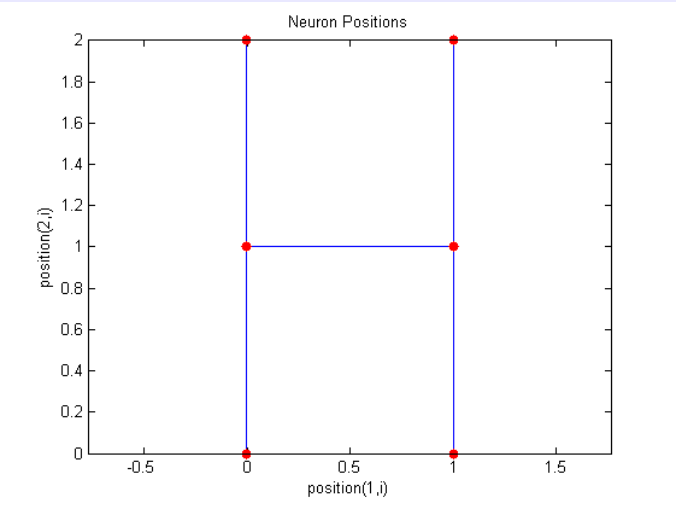


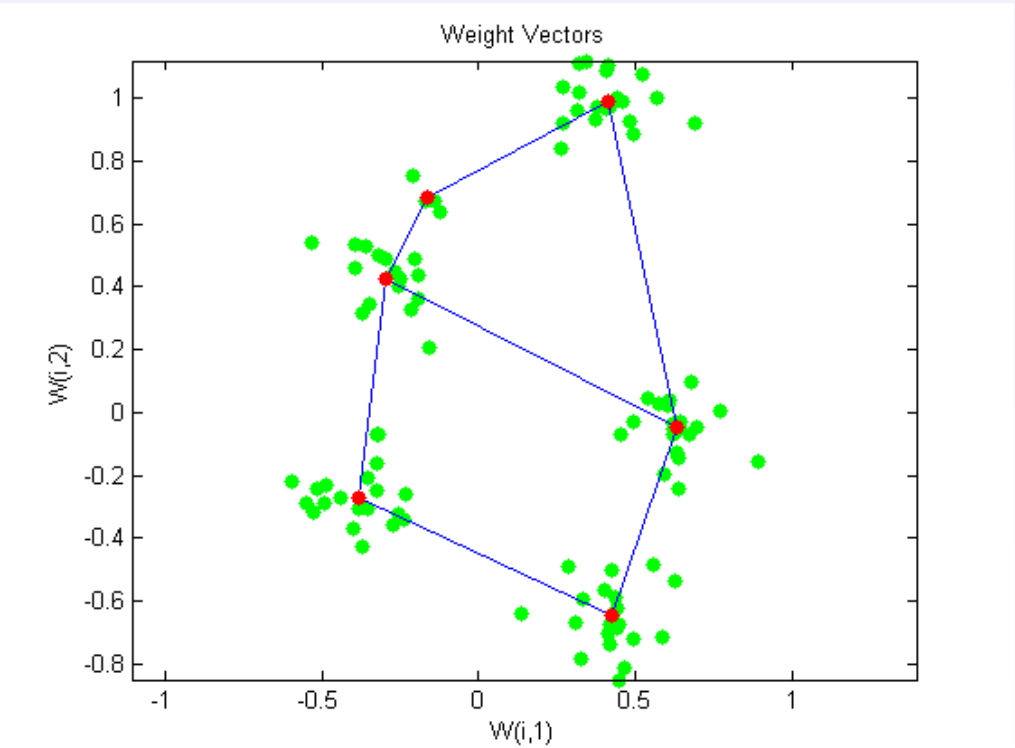
Kohonen networks – examples



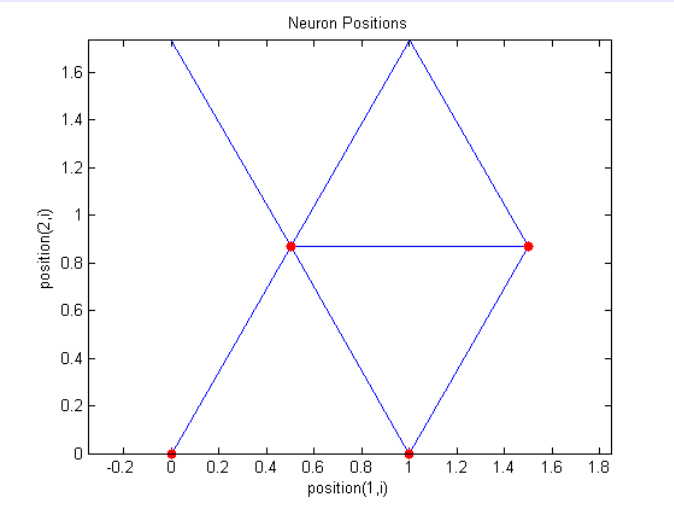


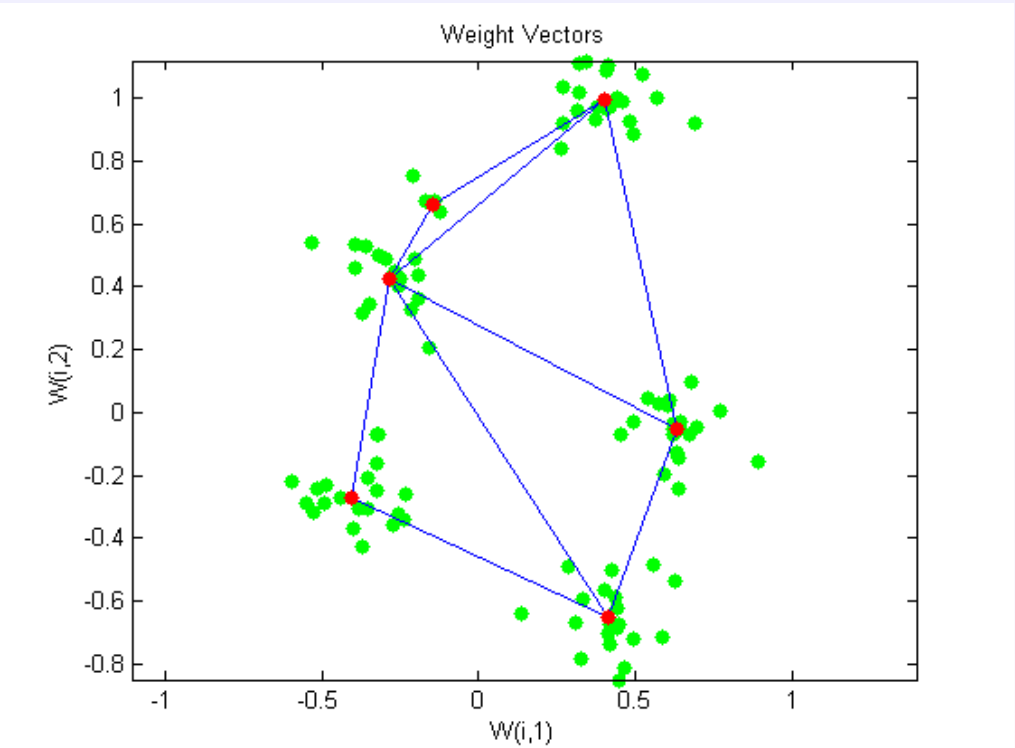
Kohonen networks – examples



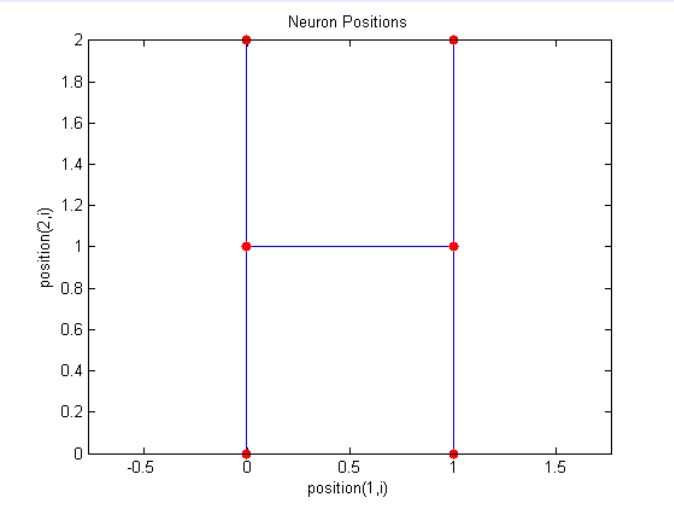


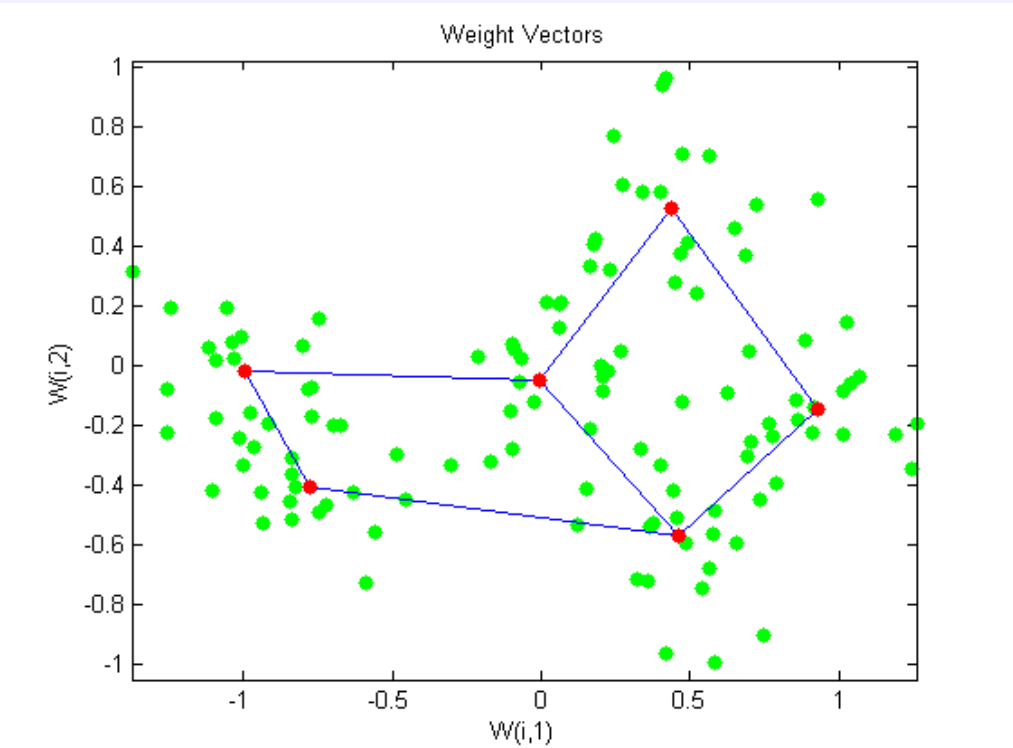
Kohonen networks – examples



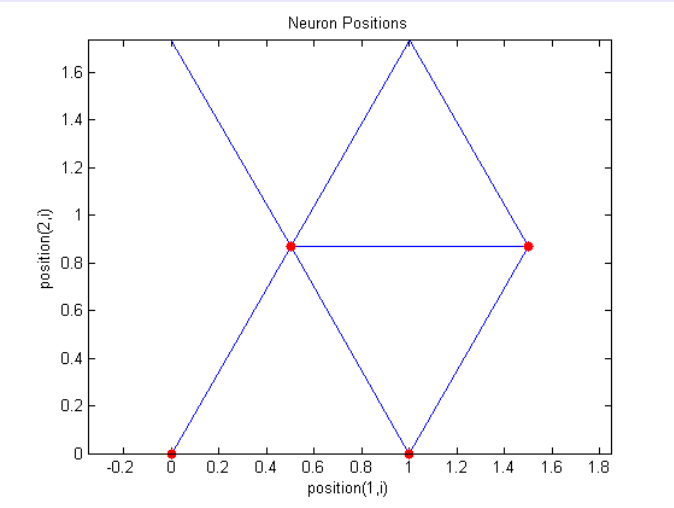


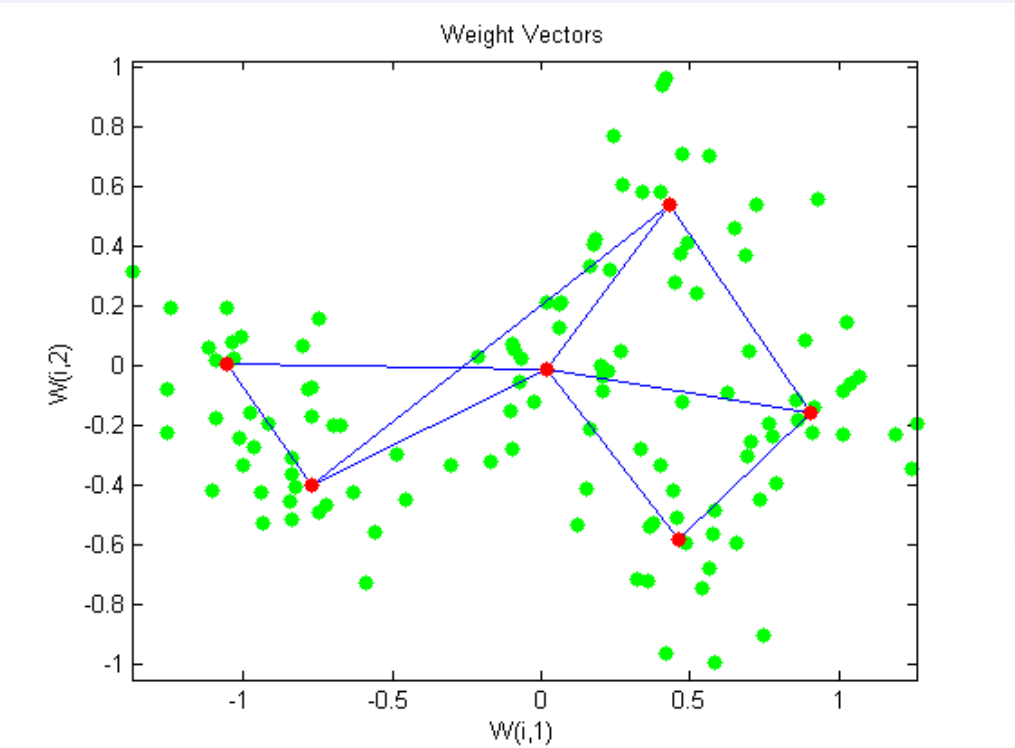
Kohonen networks – examples



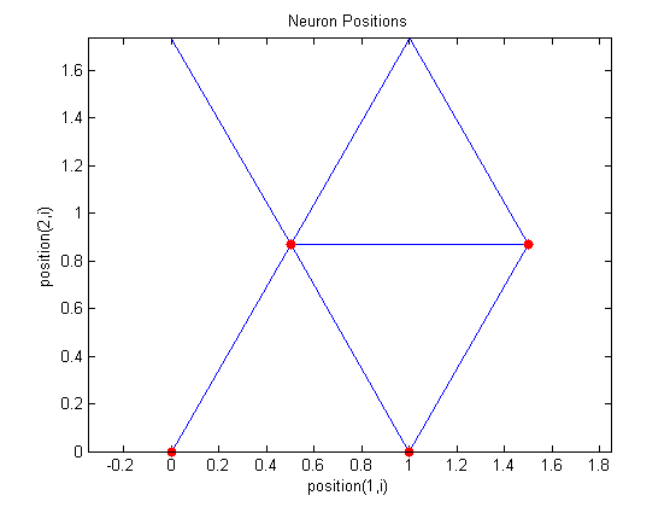


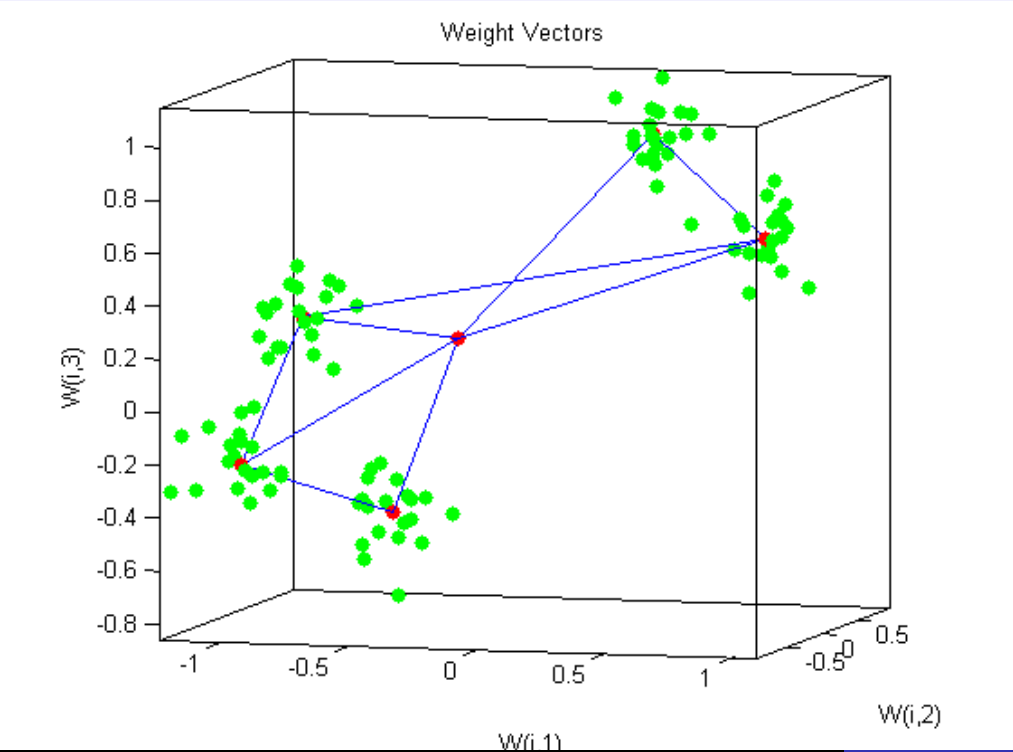
Kohonen networks – examples



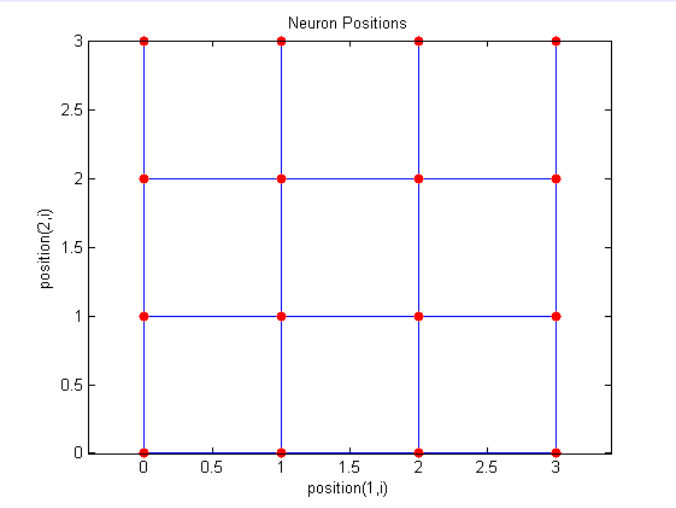


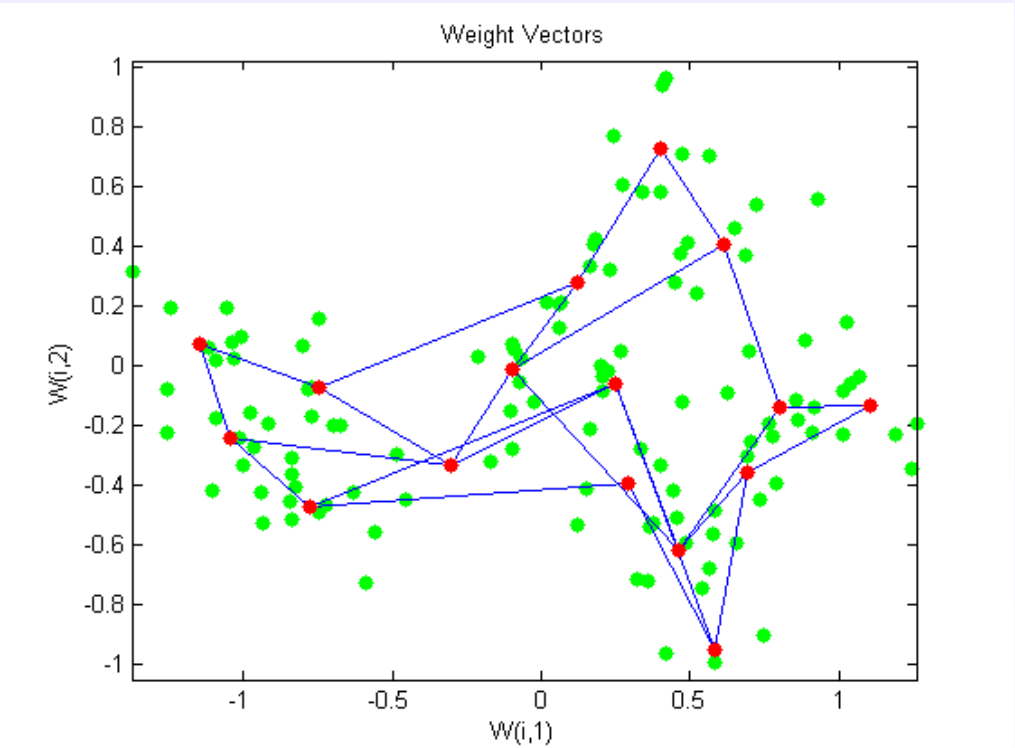
Kohonen networks – examples



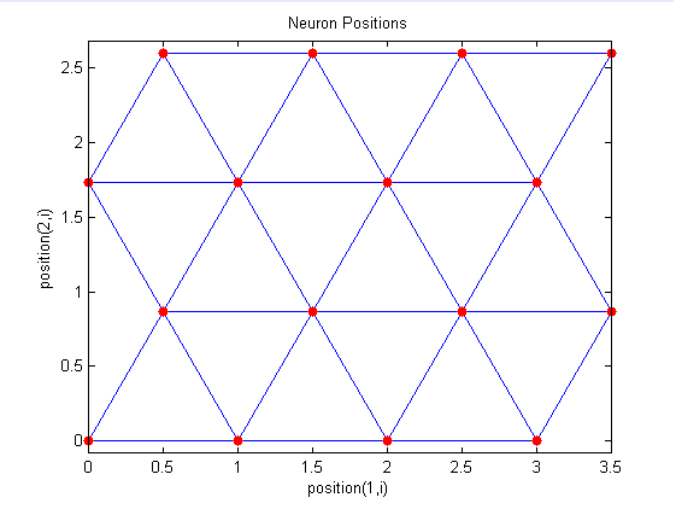


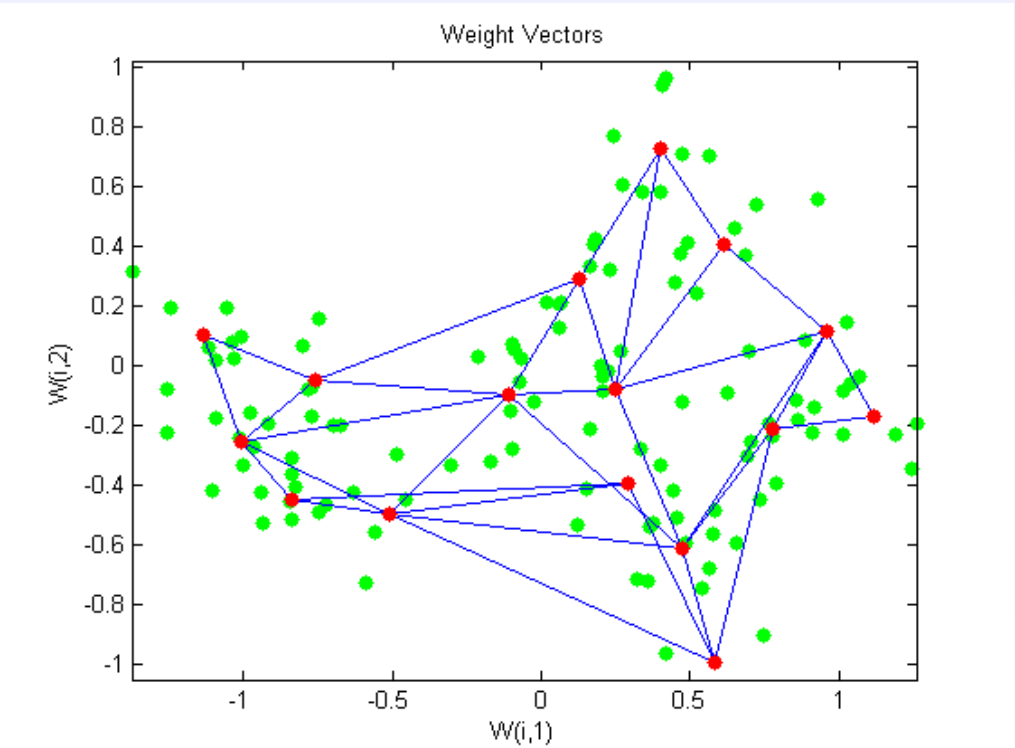
Kohonen networks – examples



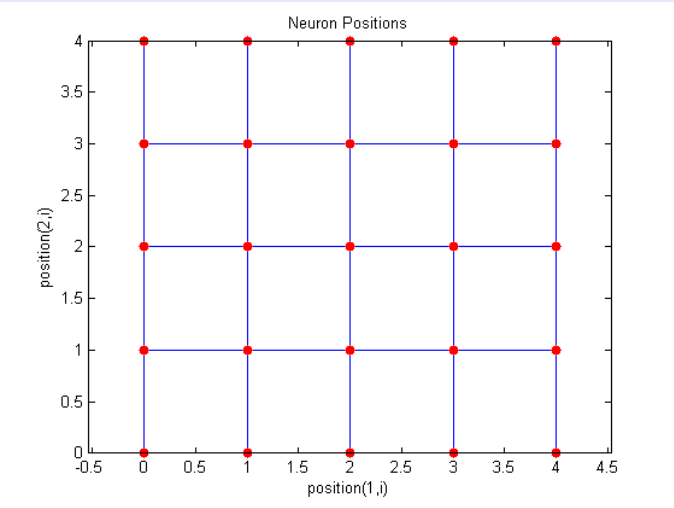


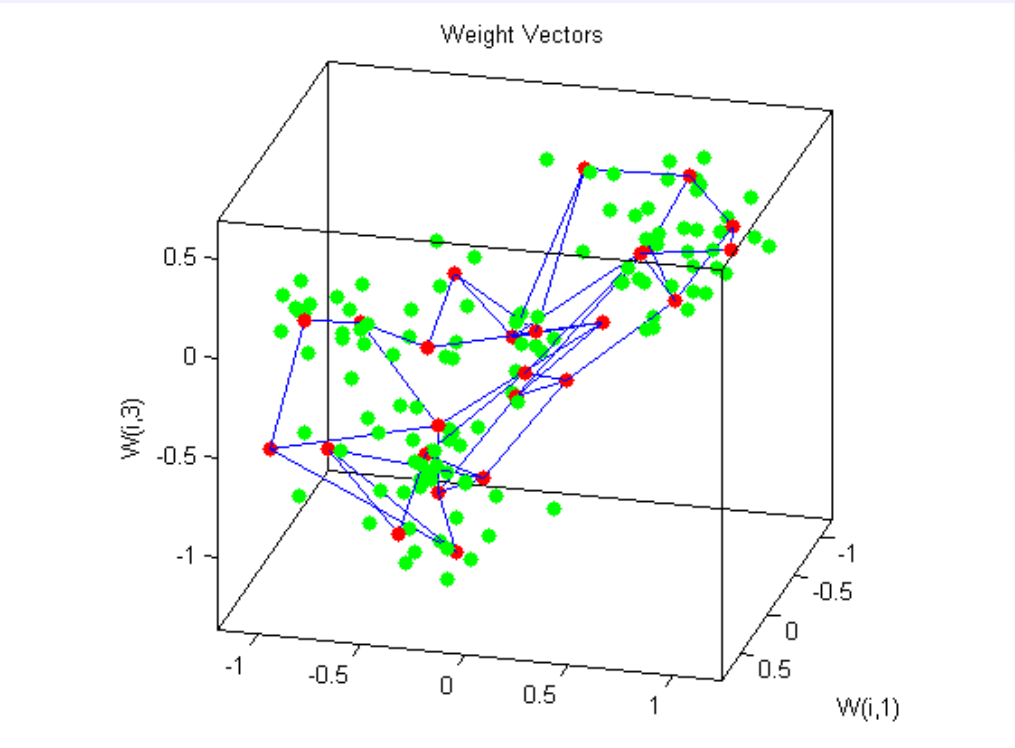
Kohonen networks – examples



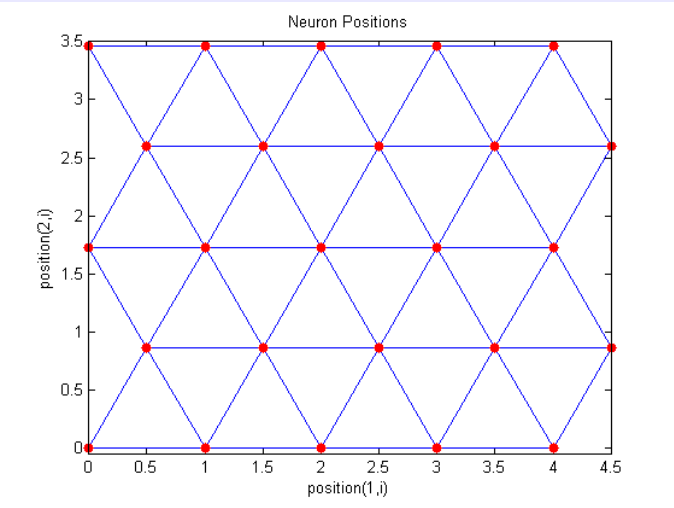


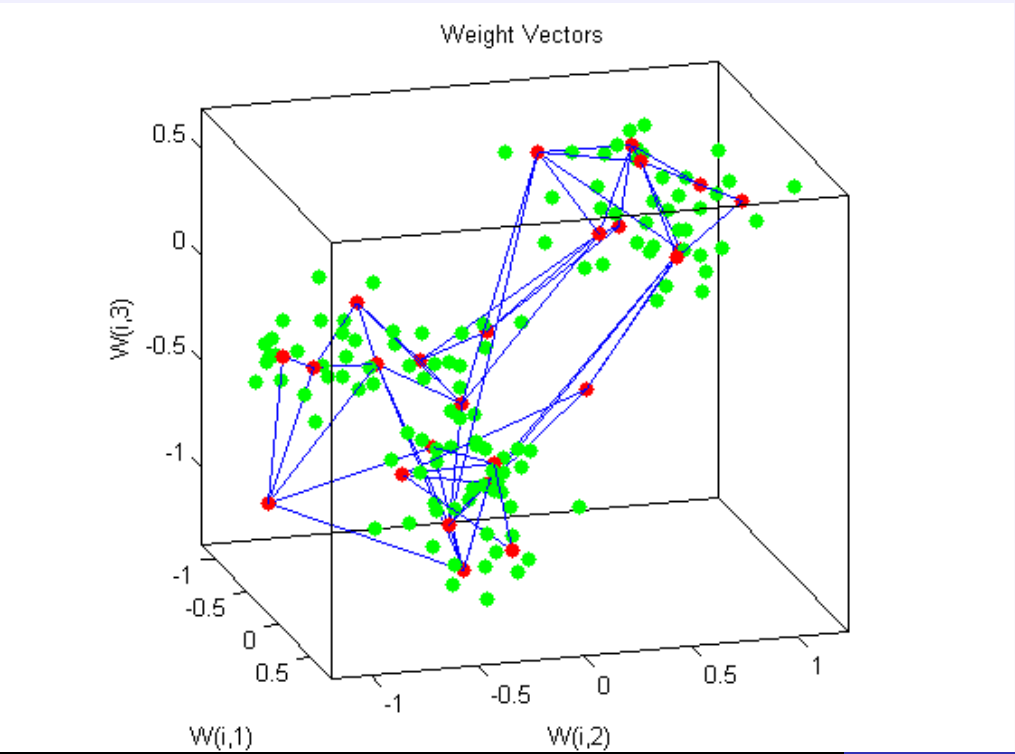
Kohonen networks – examples



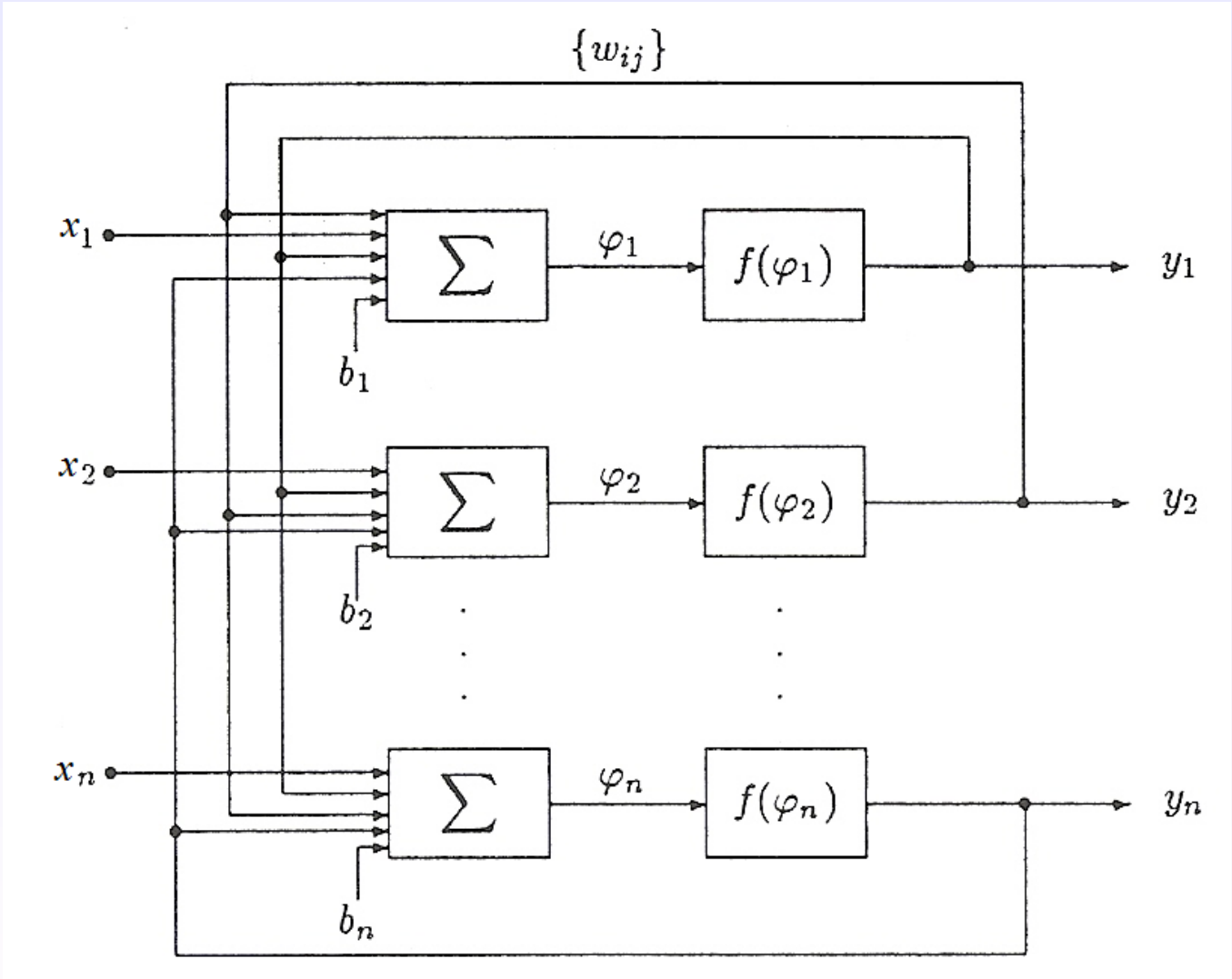


Kohonen networks – examples





Recurrent neural networks



Recurrent neural networks

General informations

- There are feedback connections in networks – outputs of neurons are connected to inputs.
- In the network, signal 'oscillates' between the output and the input to achieve a certain convergence criterion – and then it is given to the output.
- Recurrent neural networks are nonlinear dynamic systems from the point of view of systems theory.

Recurrent neural networks

General informations

- There are feedback connections in networks – outputs of neurons are connected to inputs.
- In the network, signal 'oscillates' between the output and the input to achieve a certain convergence criterion – and then it is given to the output.
- Recurrent neural networks are nonlinear dynamic systems from the point of view of systems theory.

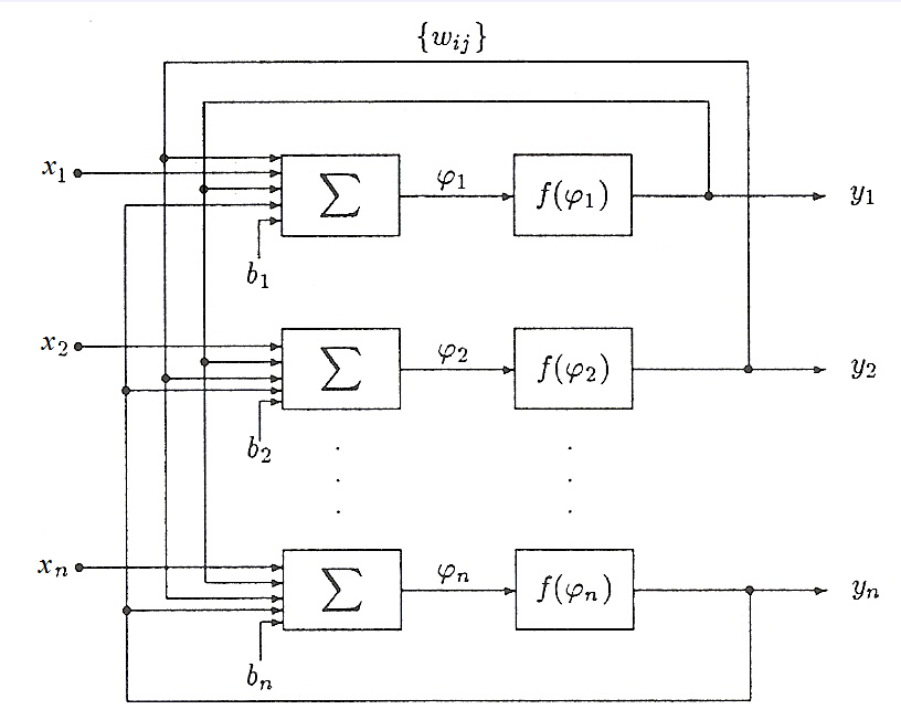
Tasks

- Autoassociative memory.
- Optimization.

Discrete Hopfield network

Main features

- One layer (virtual) of neurons with feedback connections.
- No of neurons = no of inputs = no of outputs = n.
- Square weight matrix of size $n \times n$.



Discrete Hopfield network

Each neuron determines its output signal on the base of formula:

$$\phi_i(k) = \sum_{j=1}^{n} w_{ij} \cdot y_j + b_i$$

$$y_i(k+1) = \begin{cases} 1 & \text{for} \quad \phi_i(k) > 0 \\ y_i(k) & \text{for} \quad \phi_i(k) = 0 \\ 0 & \text{for} \quad \phi_i(k) < 0 \end{cases}$$

where:

- $y_i(k)$ – output of the neuron no i in moment k,
- w_{ij} – weight of the connection between output of the neuron no j and input of the neuron no i,
- b_i – threshold of the neuron no i (often not used!).

Discrete Hopfield network

- There are feedback connections in the network. Output of each neuron is connected to inputs of all other neurons. There is no connection between the output and the input of the same neuron:

$$w_{ii} = 0\,.$$

- The weight matrix is symmetric:

$$w_{ij} = w_{ji}\,.$$

- Since each neuron is connected with each other, the network has no layers.

Discrete Hopfield network – behavior

- At the beginning moment $k = 0$, we connect input signals $x_i \in \{0, 1\}$ to the neuron, and in that way we define the beginning state of the network.

$$y_i(0) = x_i$$

- In this moment, inputs are disconnected and an iterative process of state updating begins in the network, according to formulas:

$$\phi_i(k) = \sum_{j=1}^{n} w_{ij} \cdot y_j + b_i$$

$$y_i(k+1) = \begin{cases} 1 & \text{for} \quad \phi_i(k) > 0 \\ y_i(k) & \text{for} \quad \phi_i(k) = 0 \\ 0 & \text{for} \quad \phi_i(k) < 0 \end{cases}$$

Discrete Hopfield network – behavior

- Changes of the state are realised in discrete moments of time.
- The network works asynchronously – in one moment of time only one output is actualised (usually chosen at random).
- After **finished** number of iterations the network reaches a stable state:

$$y_i(k+1) = y_i(k)\,, \quad \forall i$$

- At this moment, the (so called) recovery process ends and the state of the network is transmitted to the output.

Discrete Hopfield network – energy function

- The Hopfield network has defined, so called, energy function.
- There is defined a certain value of this function for each network state (defined by output vector $\mathbf{y}$).
- The energy function is bounded on the bottom and non-growing during state changes – it means that during the recovery process the energy function value decreases or remains unchanged.
- Stable state, achieved at the end of the recovery process, corresponds to a local minimum of the energy function.

Discrete Hopfield network – energy function

- The Hopfield network has defined, so called, energy function.
- There is defined a certain value of this function for each network state (defined by output vector $\mathbf{y}$).
- The energy function is bounded on the bottom and non-growing during state changes – it means that during the recovery process the energy function value decreases or remains unchanged.
- Stable state, achieved at the end of the recovery process, corresponds to a local minimum of the energy function.

Energy function can be described by formula:

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j + \sum_{i=1}^{n} b_i y_i$$

Discrete Hopfield network – energy function

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j + \sum_{i=1}^{n} b_i y_i$$

Let's assume that the state of the neuron i changes in moment k according to formula:

$$y_i(k+1) = y_i(k) + \Delta y_i(k)$$

The network works asynchronously, so the state of other neurons is unchanged, i.e.:

$$y_j(k+1) = y_j(k)\,, \quad \text{for } j \neq i$$

The change of energy can be calculated as:

$$\Delta E(k) = E(\mathbf{y}(k+1)) - E(\mathbf{y}(k)) = -\Delta y_i(k) \cdot \left(\sum_{j=1}^{n} w_{ij}y_j + b_i \right) = -\Delta y_i(k) \cdot \phi_i(k)$$

Discrete Hopfield network – energy function

$$\Delta E(k) = E(\mathbf{y}(k+1)) - E(\mathbf{y}(k)) = -\Delta y_i(k) \cdot \left(\sum_{j=1}^{n} w_{ij} y_j + b_i \right) = -\Delta y_i(k) \cdot \phi_i(k)$$

If $\phi_i(k) = 0$ than the change of energy equals 0.

Other possible cases are considered in the table.

$y_i(k+1)$	$y_i(k)$	$\Delta y_i(k)$	$\phi_i(k)$	$\Delta E(k)$
0	0	0	–	0
0	1	−1	–	–
1	0	1	+	–
1	1	0	+	0

Therefore, there is always:

$$\Delta E(k) \leq 0$$

$$E(\mathbf{y}(k+1)) \leq E(\mathbf{y}(k))$$

Discrete Hopfield network – energy function

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j + \sum_{i=1}^{n} b_i y_i$$

It can be noticed that the energy function is bounded on the bottom, because:

$$|E(\mathbf{y})| \leq \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} |w_{ij}| + \sum_{i=1}^{n} |b_i|$$

Discrete Hopfield network – energy function

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j + \sum_{i=1}^{n} b_i y_i$$

It can be noticed that the energy function is bounded on the bottom, because:

$$|E(\mathbf{y})| \leq \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} |w_{ij}| + \sum_{i=1}^{n} |b_i|$$

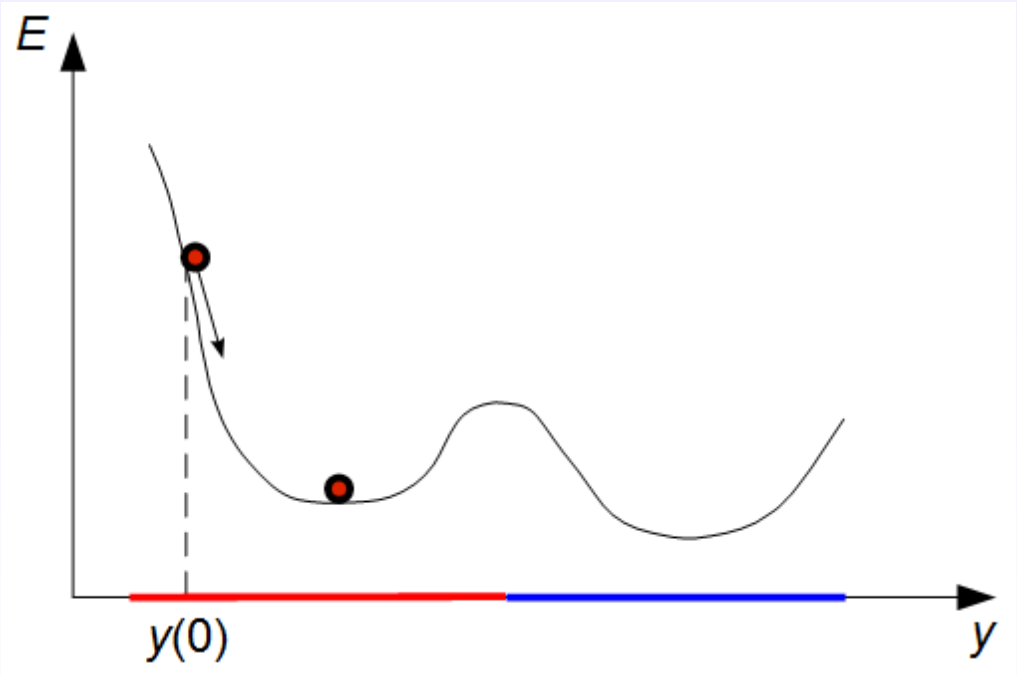
Because a monotonic and limited sequence must be convergent, the value $E(\mathbf{y})$ will tend towards a certain finite value E_{min}.

Discrete Hopfield network – energy function

- The set of energy function values is finite – it results from the fact, that its domain is finite because $y_i \in \{0, 1\}$.
- It means that the set of possible energy changes ΔE is also finite.
- Possible energy changes can not be infinitely small (what could result in an infinitely long setting up of the state).
- Finally, the energy reaches a steady state E_{min} in finite number of steps k_{max}.
- The state $\mathbf{y}$ in which $E(\mathbf{y}) = E_{min}$ (local minimum) is the stable state.

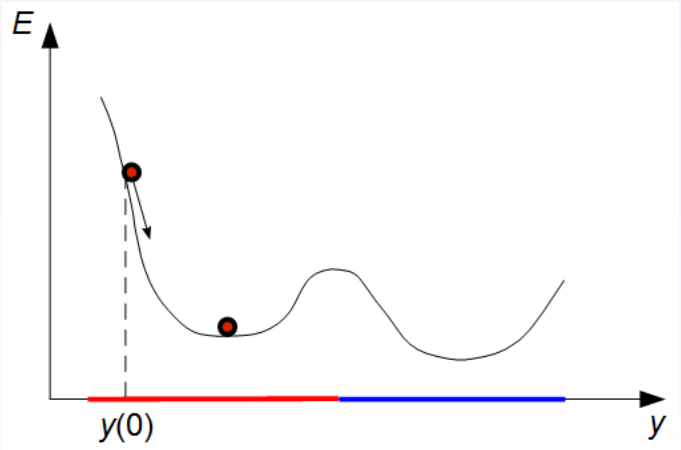
Discrete Hopfield network – energy function

- The beginning state of the network is defined by the chose of $\mathbf{y}(0)$.
- If it is not a stable state, in the course of subsequent iterations (recovery phase), $\mathbf{y}$ changes in such a way that the energy function value decreases until reaching the local minimum.

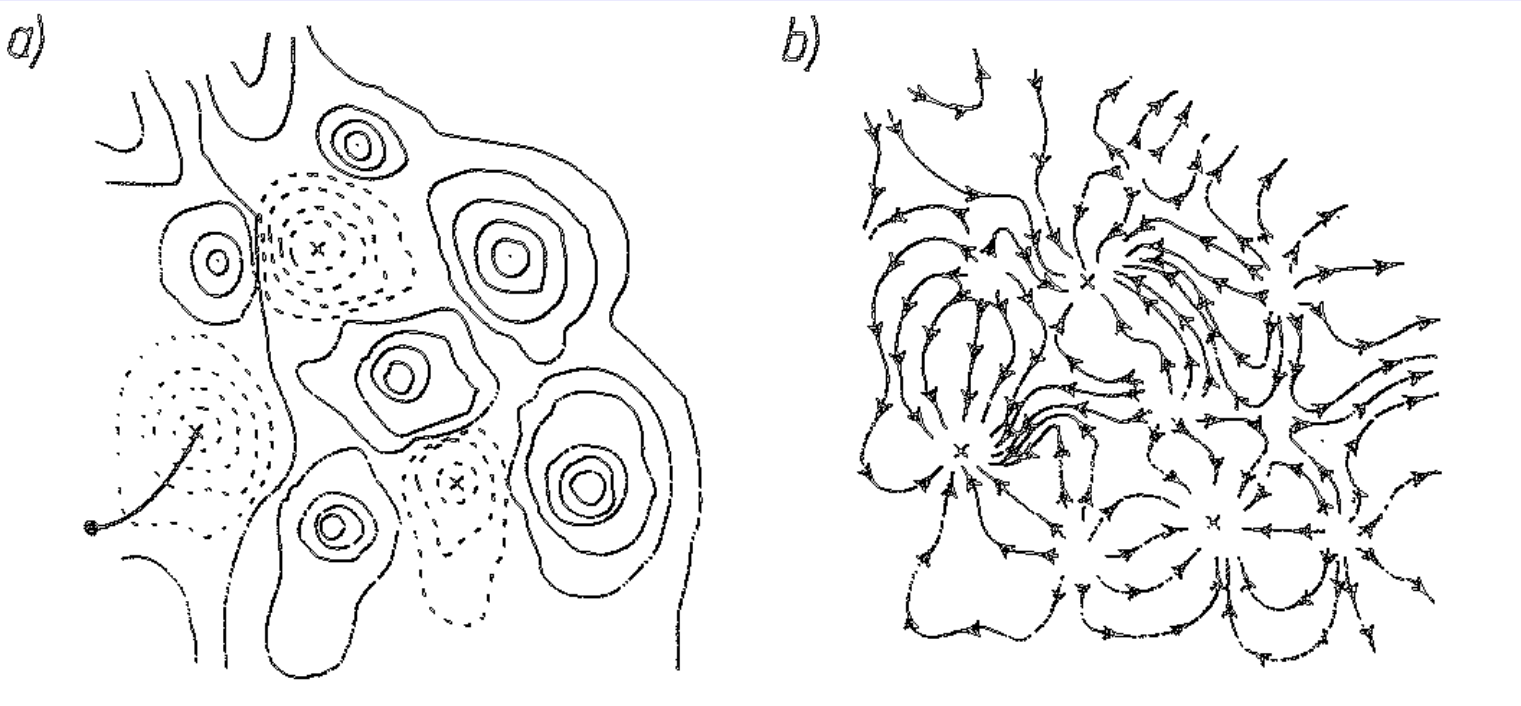


Discrete Hopfield network – energy function

- Stable states are called **attractors**.
- To each attractor, we can assign a set of initial states $\mathbf{y}(0)$, which initiate an evolution of the network state ending in it. Such set is called **a basin of attraction** of the attractor.
- The choice of connection weights between neurons has a decisive impact for the number of attractors, their mutual distance and the corresponding value of the energy function (depth of basins).



Discrete Hopfield network – energy function



The shape of attractions areas of the recurrent network: a) contour plot, b) mapping of directions of energy function changes during the recovery phase.

Autoassociative memory

- The concept of such a memory is associated with one of the basic functions of the brain. For example, straining attention we can recognize unclear speech, read unreadable handwriting, guess the entire word in a crossword, seeing only some of letters.
- Such process is called association. This is done by recovering the entire available information on the base of fragments or information that is distorted.

Autoassociative memory

Let's assume that we have a set of M different patterns:

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\} \subset R^n.$$

Autoassociative memory connected with this set is the system implementing a mapping:

$$F: \ R^n \to R^n \,,$$

such that:

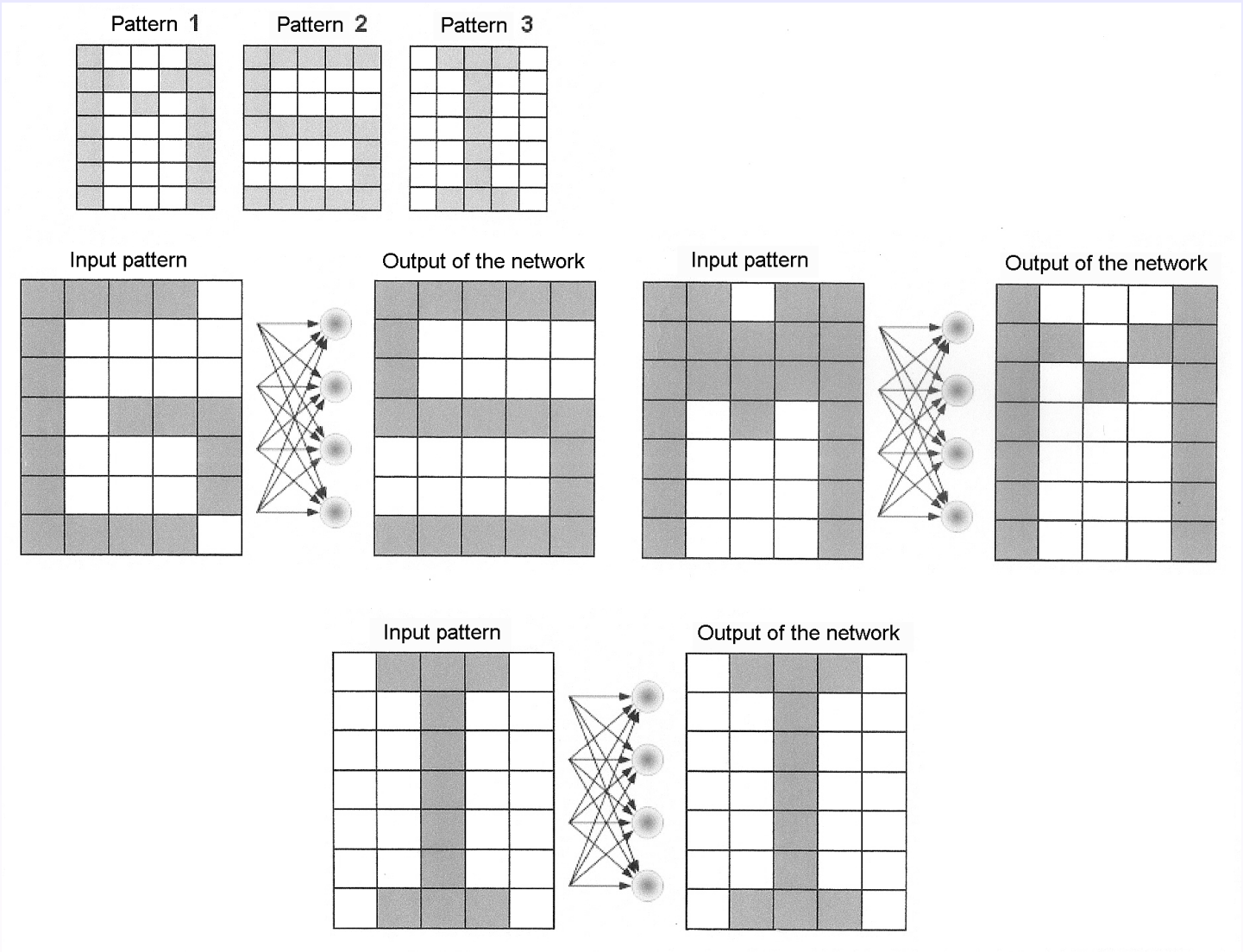
$$F(\mathbf{x}_i) = \mathbf{x}_i \,, \quad \text{for } i = 1 \ldots M$$

and:

$$F(\mathbf{x}) = \mathbf{x}_S \,,$$

where: $\mathbf{x}_S$ is the most similar to $\mathbf{x}$ from all M patterns.

Autoassociative memory



Similarity degree

Similarity degree for two vectors **x** and **y** can be defined on the base of Euclidean measure:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad \text{for } \mathbf{x}, \mathbf{y} \in R^n$$

Similarity degree

Similarity degree for two vectors **x** and **y** can be defined on the base of Euclidean measure:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad \text{for } \mathbf{x}, \mathbf{y} \in R^n$$

As, analysed here vectors belong to the Hamming space:

$$H^n = \{\mathbf{x} \in R^n : \quad x_i \in \{0, 1\}\},$$

i.e. set of n-dimensional vectors with elements 0 and 1, we can apply, so called, Hamming distance measure d_H. For vectors **x** and **y**, it is equal to the number of different elements.

Similarity degree

Similarity degree for two vectors **x** and **y** can be defined on the base of Euclidean measure:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad \text{for } \mathbf{x}, \mathbf{y} \in R^n$$

As, analysed here vectors belong to the Hamming space:

$$H^n = \{\mathbf{x} \in R^n : \quad x_i \in \{0, 1\}\},$$

i.e. set of n-dimensional vectors with elements 0 and 1, we can apply, so called, Hamming distance measure d_H. For vectors **x** and **y**, it is equal to the number of different elements.

For:

$$\mathbf{x} = [1, 0, 0, 1, 0] \quad \text{i} \quad \mathbf{y} = [1, 0, 1, 1, 1]$$

we have:

$$d_H(\mathbf{x}, \mathbf{y}) = 2$$

Hopfield network as an autoassociative memory

- Described property of the Hopfield network (existence of attractors to which the state of the network is evolving from the given initial state) allows its application as an autoassociative memory.
- The shape of the energy function plot (and the location of attractors) depends on network weights.
- So, it is enough to choose weights in such a way that each pattern is one of attractors, and its basin of attraction is as wide and as deep as possible to assure associations between initial and final states.

Learning of the Hopfield network

For originally described network, Hopfield proposed the following method of weights calculation:

$$w_{ij} = \begin{cases} \sum_{m=1}^{M} (2x_i^{(m)} - 1)(2x_j^{(m)} - 1) & \text{for} \quad i \neq j \\ 0 & \text{for} \quad i = j \end{cases}$$

where:

- M – number of patterns,
- $x_i^{(m)}$ – input no i for pattern no m.

Learning of the Hopfield network

For discrete bipolar network (state described with -1 and 1) weights can be calculated with other formulas.

If we have only one pattern:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \{-1, 1\}$$

weights can be calculated as (Hebbian rule):

$$w_{ij} = \begin{cases} \frac{1}{n} \cdot x_i \cdot x_j & \text{for} \quad i \neq j \\ 0 & \text{for} \quad i = j \end{cases}$$

or in a matrix form:

$$\mathbf{W} = \frac{1}{n}(\mathbf{x} \cdot \mathbf{x}^T - \mathbf{1})$$

Learning of the Hopfield network

If we have M patterns:

$$\mathbf{x} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(M)} \\ \vdots & \dots & & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(M)} \end{bmatrix}, \quad x_i \in \{-1, 1\}$$

weights can be calculated as (Hebbian rule):

$$w_{ij} = \begin{cases} \frac{1}{n} \cdot \sum_{m=1}^{M} x_i^{(m)} \cdot x_j^{(m)} & \text{for} \quad i \neq j \\ 0 & \text{for} \quad i = j \end{cases}$$

or in a matrix form:

$$\mathbf{W} = \frac{1}{n}(\mathbf{X} \cdot \mathbf{X}^T - \mathbf{1}M)$$

Learning of the Hopfield network

If we have M patterns:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(M)} \\ \vdots & \dots & & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(M)} \end{bmatrix}, \quad x_i \in \{-1, 1\}$$

weights can be also calculated with the application of pseudo-inverse matrix:

$$\mathbf{W} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$$

Capacity of the Hopfield network

- Capacity of the network learnt with Heebian rule equals $0.138 \cdot n$ (where n – number of neurons).
- Capacity of the network learnt with pseudo-inverse method equals $n - 1$.

Capacity of the Hopfield network

Caution!

- If the network remembers patterns $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ it also remembers (and can restore) their negations $\{\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \ldots, \overline{\mathbf{x}}_M\}$.
- So, if the set of patterns consists of only one pattern, the network has 2 attractors connected with the pattern and its negation.
- For greater number of patterns, we have attractors located in patterns, their negations, and conjunctions of patterns and negations: ($\mathbf{x}_1$ AND $\mathbf{x}_2$, $\mathbf{x}_1$ AND $\overline{\mathbf{x}}_2$, ...).
- If we have many patterns, there will be also many spurious (false) attractors which are not connected with any pattern. The disadvantage of the network is also the fact that you can not guarantee that all patterns become attractors.

Hopfield network – example

Let's assume that we have only 1 pattern ($M = 1$):

$$\mathbf{x} = [1\ 0\ 1\ 0]^T$$

Weights are calculated with formula:

$$w_{ij} = \begin{cases} \sum_{m=1}^{M} (2x_i^{(m)} - 1)(2x_j^{(m)-1}) & \text{for} \quad i \neq j \\ 0 & \text{for} \quad i = j \end{cases}$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Hopfield network – example

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Let's check the network behavior for input:

$$\mathbf{x}_t = [1\ 0\ 1\ 0]^T = \mathbf{y}(0)$$

Activity of neurons:

$$\phi_i(k) = \sum_{j=1}^{n} w_{ij} \cdot y_j(k)$$

As the result we get:

$$\begin{array}{l} \phi_1(0) = 1 \\ \phi_2(0) = -2 \\ \phi_3(0) = 1 \\ \phi_4(0) = -2 \end{array} \Rightarrow \begin{array}{l} y_1(1) = 1 \\ y_2(1) = 0 \\ y_3(1) = 1 \\ y_4(1) = 0 \end{array}$$

Hopfield network – example

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Let's check the network behavior for input:

$$\mathbf{x}_t = [1\ 0\ 0\ 0]^T = \mathbf{y}(0)$$

Activity of neurons:

$$\phi_i(k) = \sum_{j=1}^{n} w_{ij} \cdot y_j(k)$$

As the result we get:

$$\begin{array}{l} \phi_1(0) = 0 \\ \phi_2(0) = -1 \\ \phi_3(0) = 1 \\ \phi_4(0) = -1 \end{array} \Rightarrow \begin{array}{l} y_1(1) = 1 \\ y_2(1) = 0 \\ y_3(1) = 1 \\ y_4(1) = 0 \end{array}$$

Hopfield network – example

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}$$

Let's check the network behavior for input:

$$\mathbf{x}_t = [0\ 1\ 0\ 1]^T = \mathbf{y}(0)$$

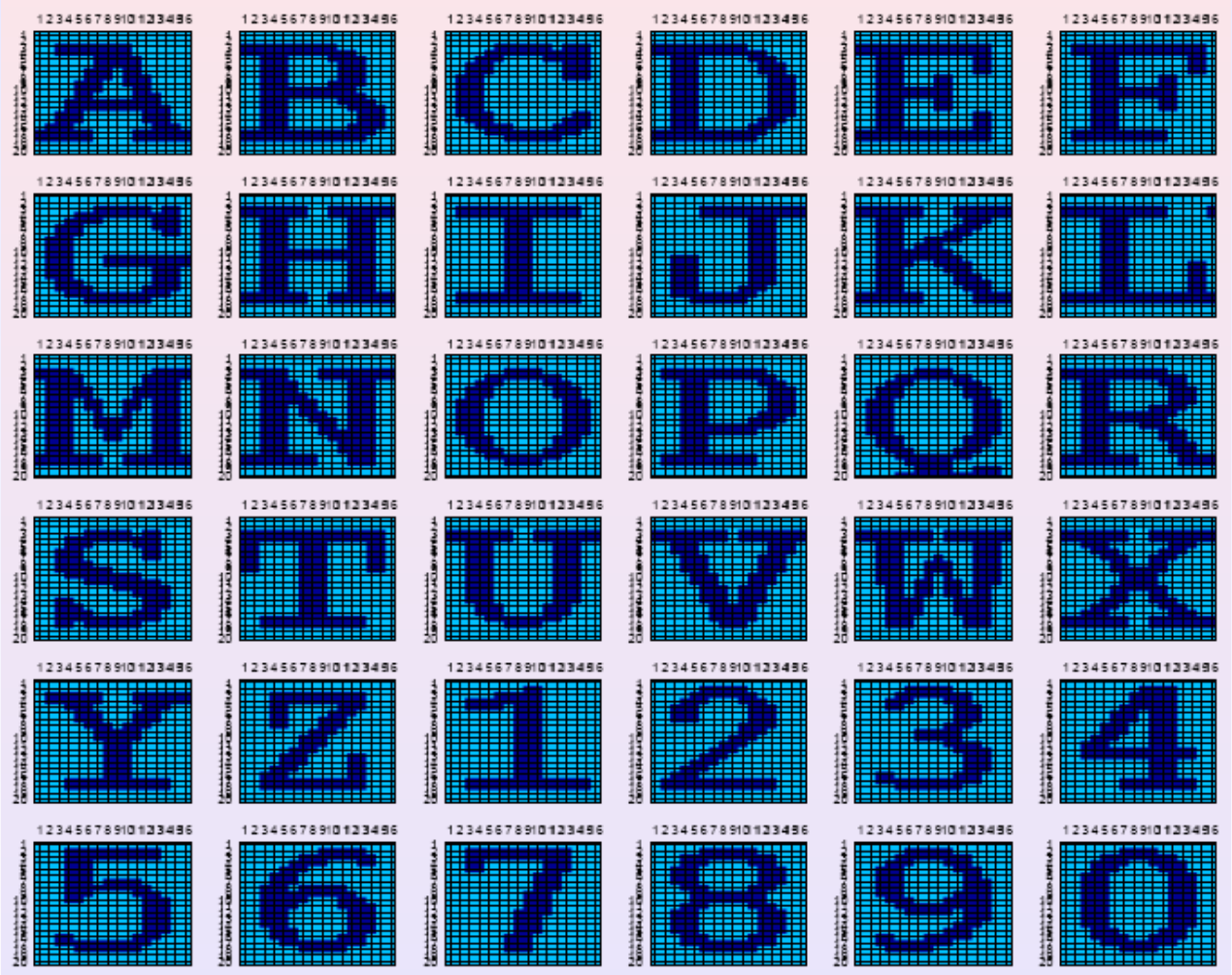
Activity of neurons:

$$\phi_i(k) = \sum_{j=1}^{n} w_{ij} \cdot y_j(k)$$

As the result we get:

$$\begin{array}{l} \phi_1(0) = -2 \\ \phi_2(0) = 1 \\ \phi_3(0) = -2 \\ \phi_4(0) = 1 \end{array} \Rightarrow \begin{array}{l} y_1(1) = 0 \\ y_2(1) = 1 \\ y_3(1) = 0 \\ y_4(1) = 1 \end{array}$$

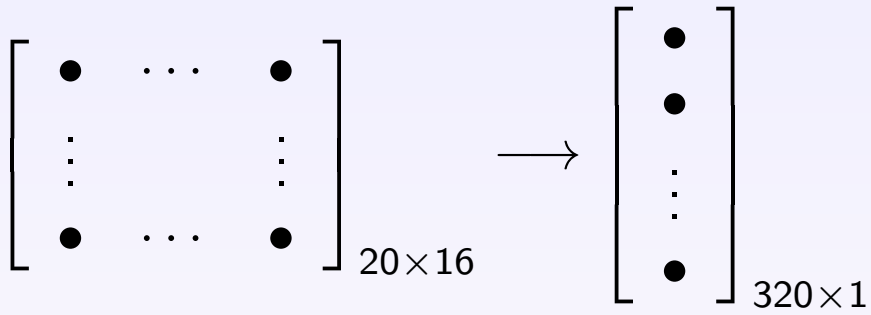
Hopfield network – character recognition



36 learning patterns – matrix 20×16.

Hopfield network – character recognition

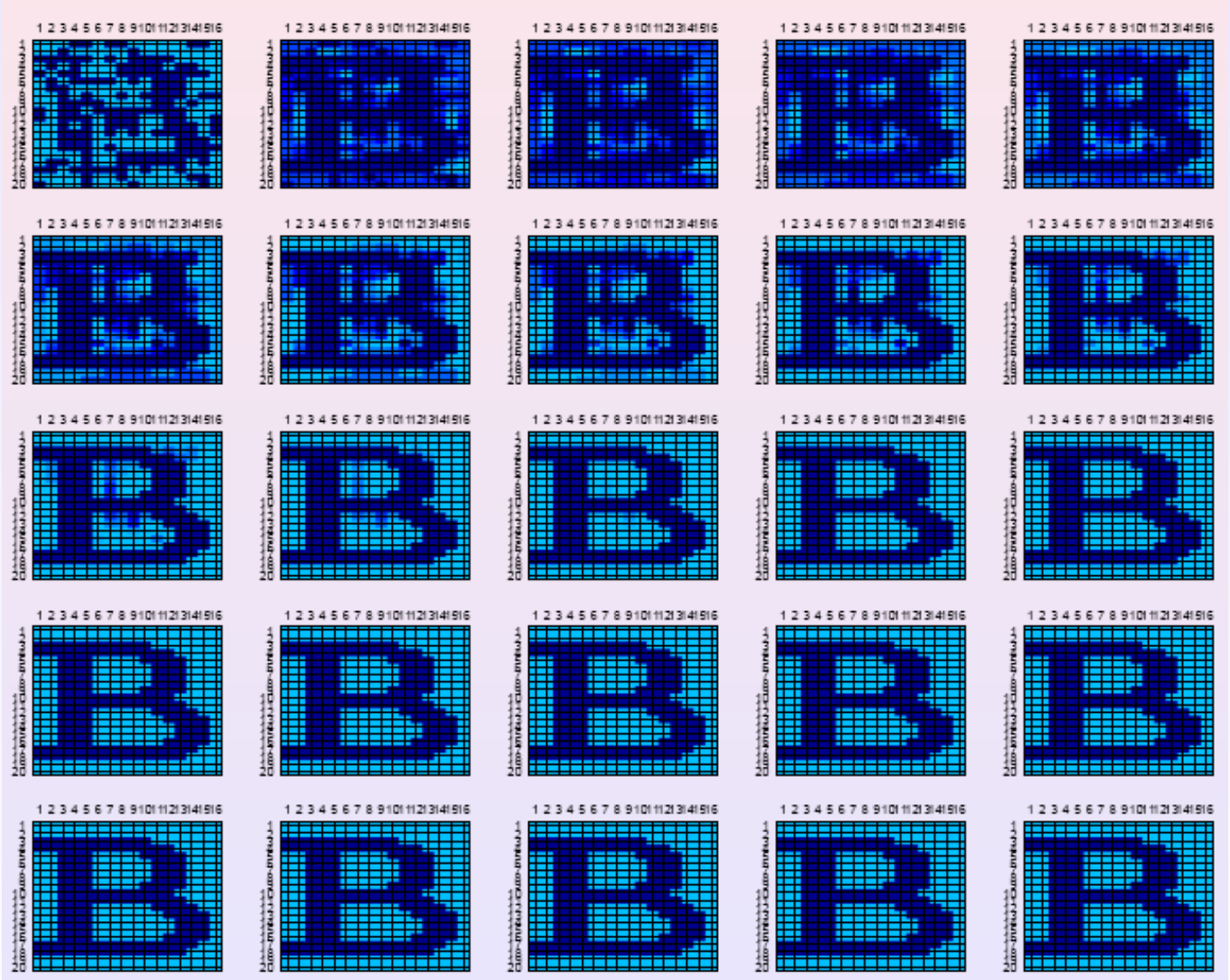
- Each character is represented by a matrix of size 20×16 filled with numbers -1 and 1.
- Each character matrix is converted into vector (function `reshape(M,new_row,new_col)` in MATLAB).



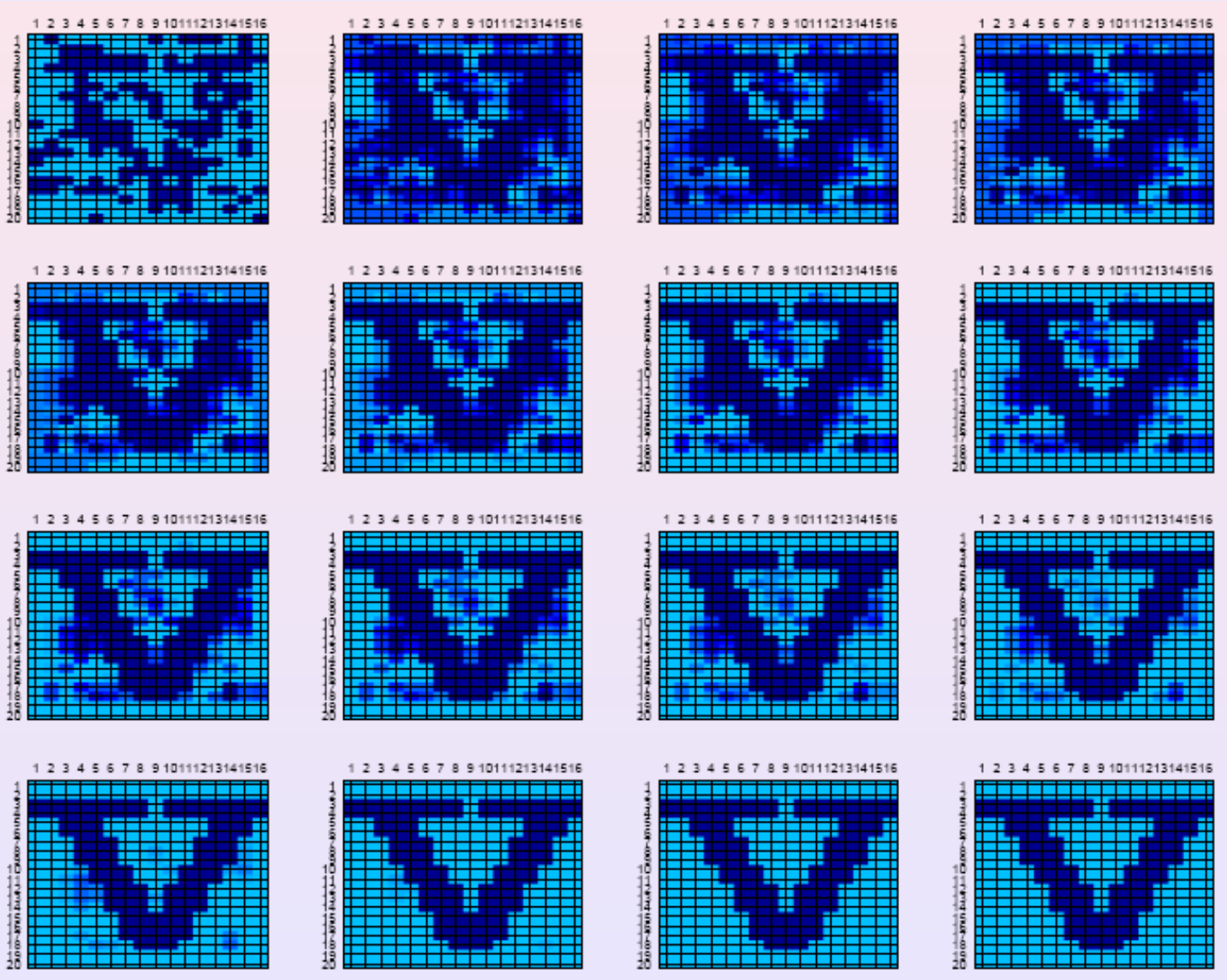
- In that way, we get the matrix of patterns:

$$\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M)}]$$

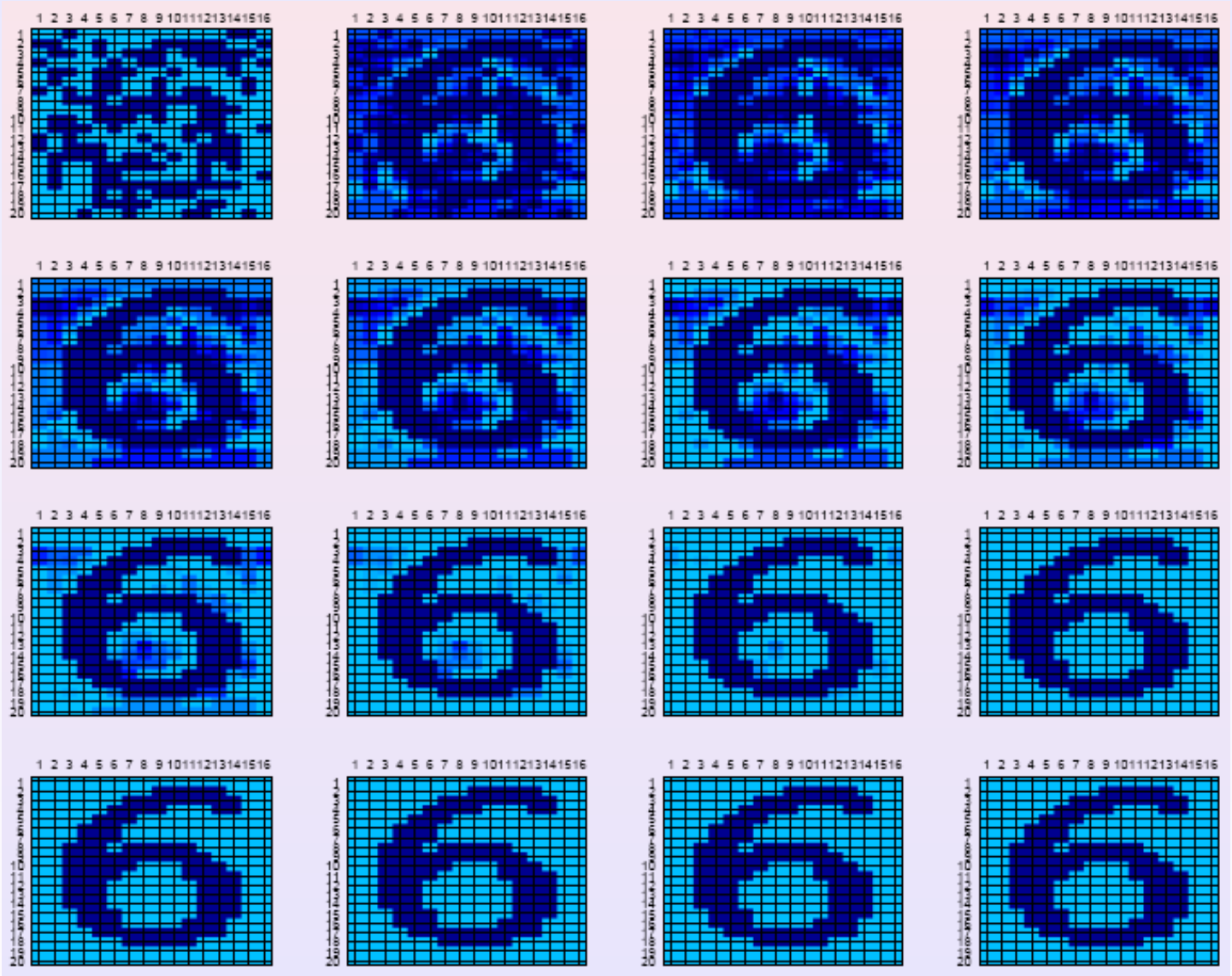
Hopfield network – character recognition



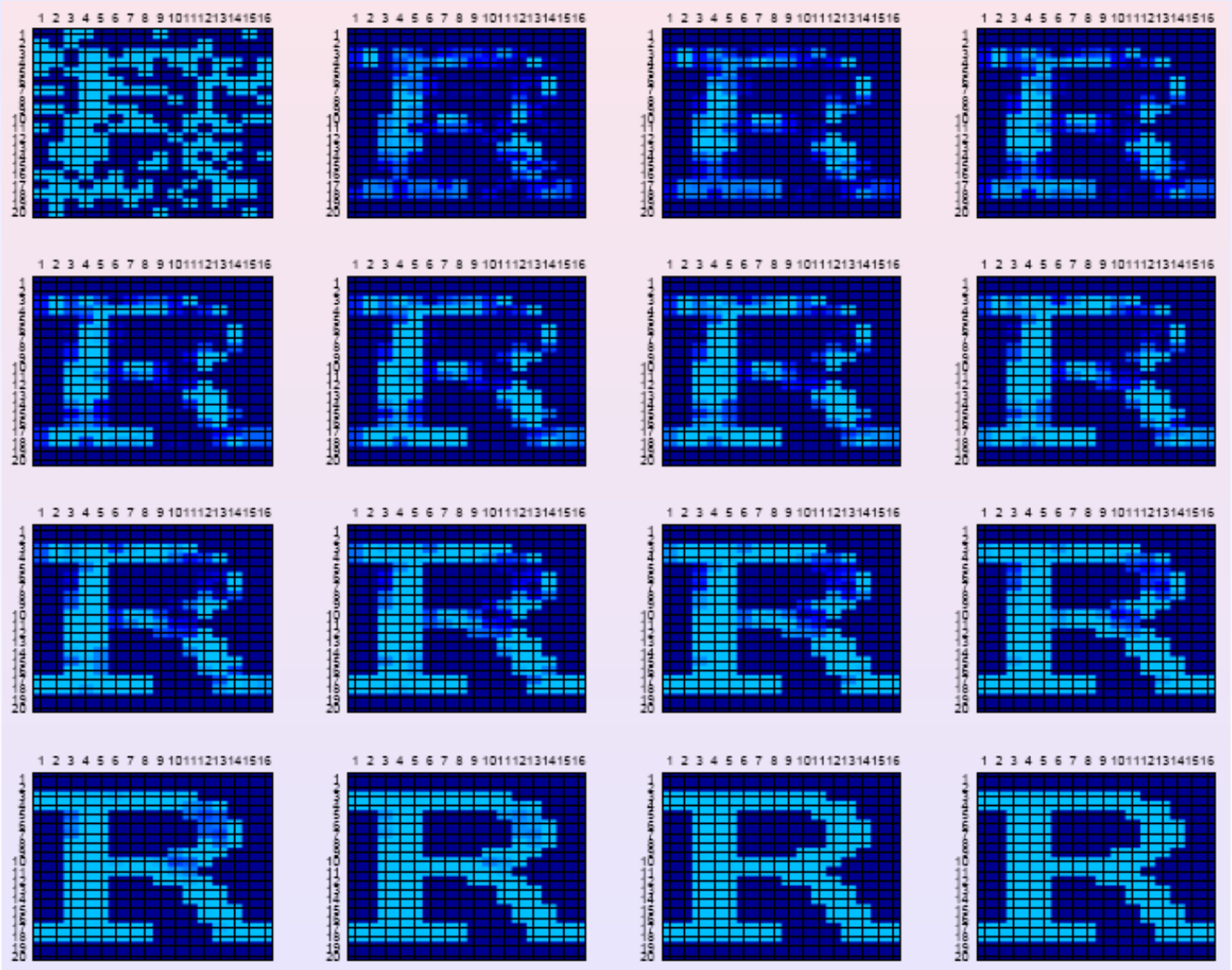
Hopfield network – character recognition



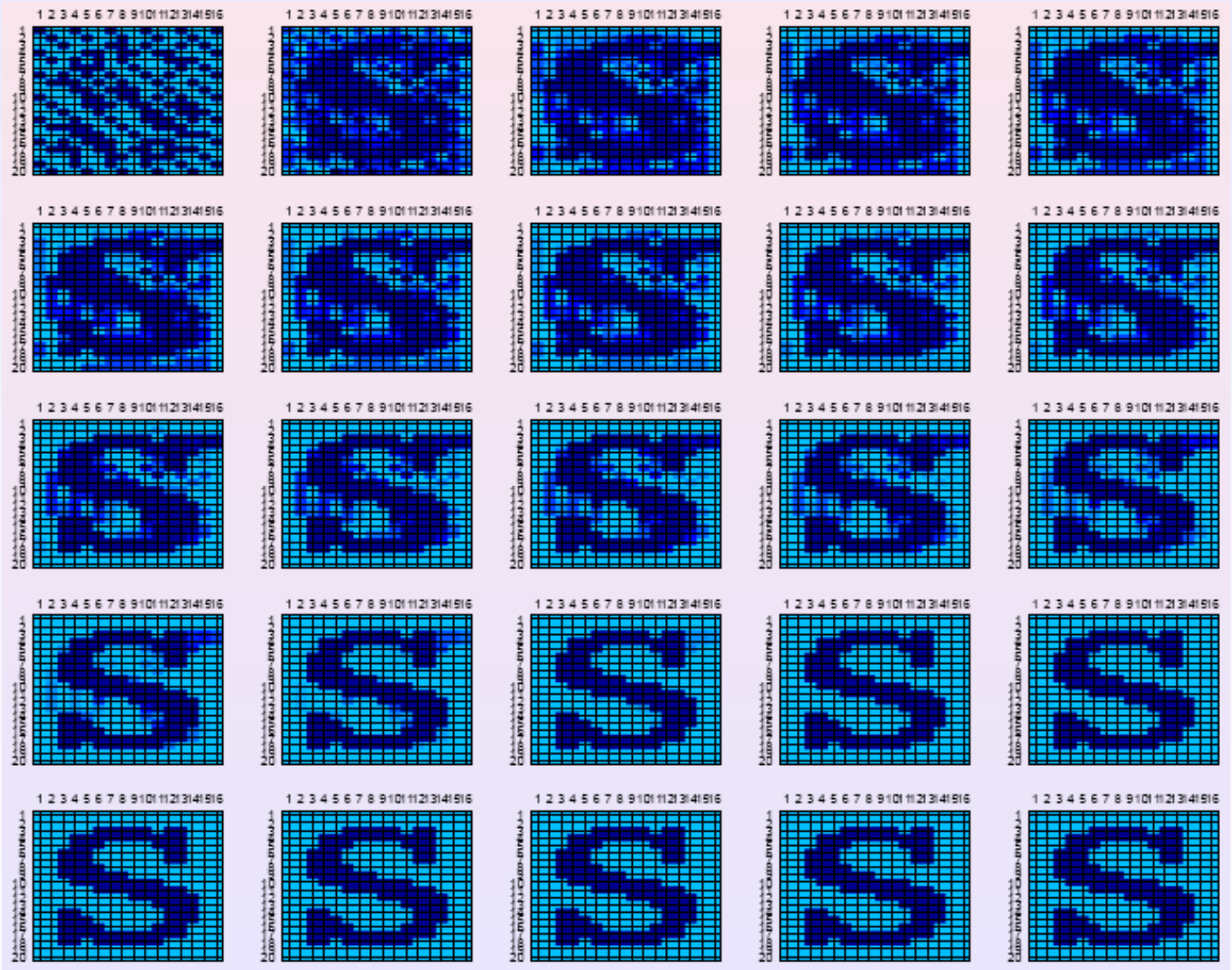
Hopfield network – character recognition



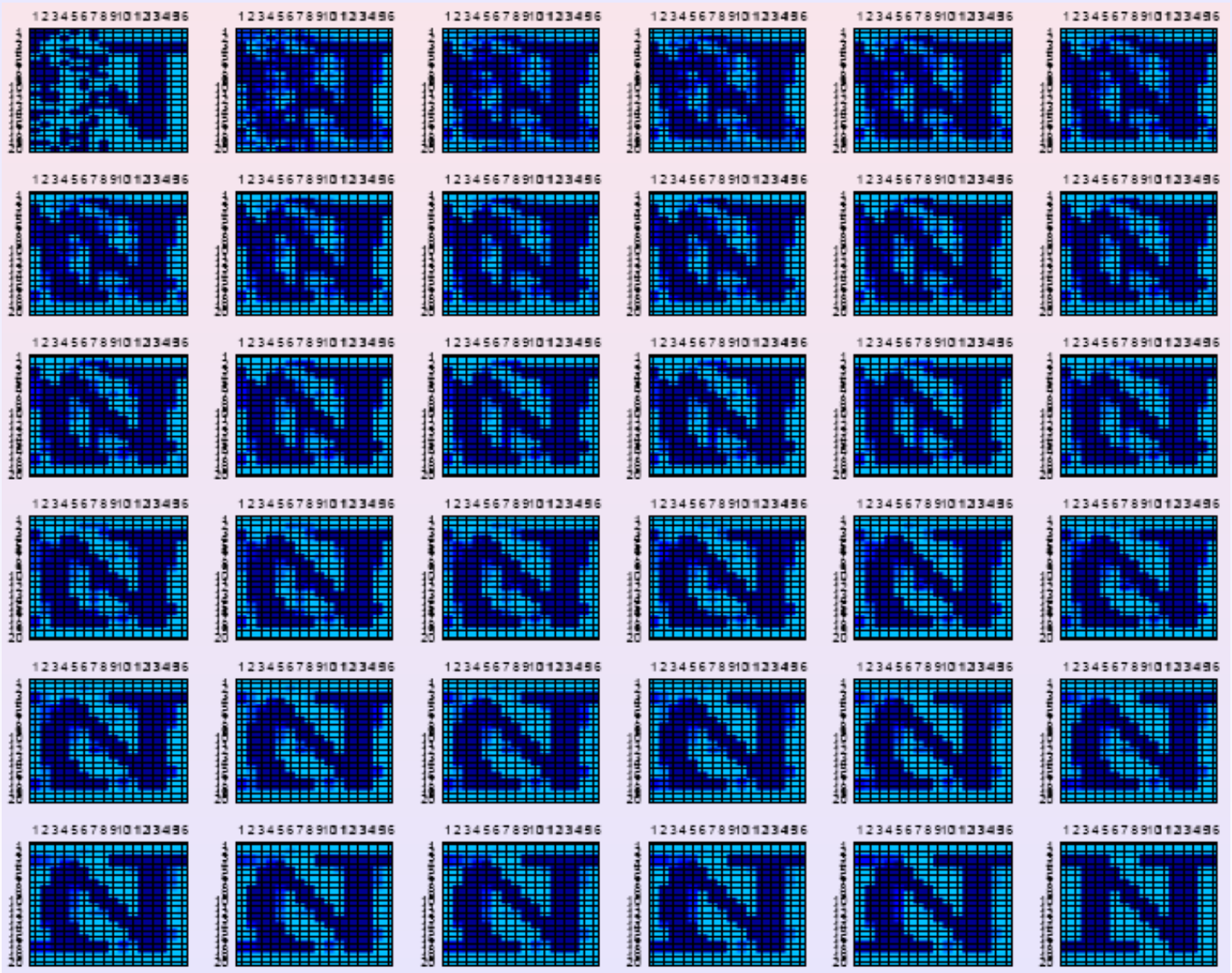
Hopfield network – character recognition



Hopfield network – character recognition



Hopfield network – character recognition



Interpolation of time series

Interpolation of time series

Tasks:

1. prediction,
2. signal filtration.

Interpolation of time series

1. We assume that the signal can be described by a deterministic model.
2. We also accept the 'naive' assumption that the signal value in a given, discrete moment of time k depends only on signal values in earlier moments: $k-1$, $k-2$, e.t.c.
3. The basis for this assumption is the application of the signal model in the form of a difference equation.

Signal model – example

Suppose we have a homogeneous, linear differential equation in the form:

$$\ddot{y} + 2\dot{y} + 2y = 0$$

Signal model – example

Suppose we have a homogeneous, linear differential equation in the form:

$$\ddot{y} + 2\dot{y} + 2y = 0$$

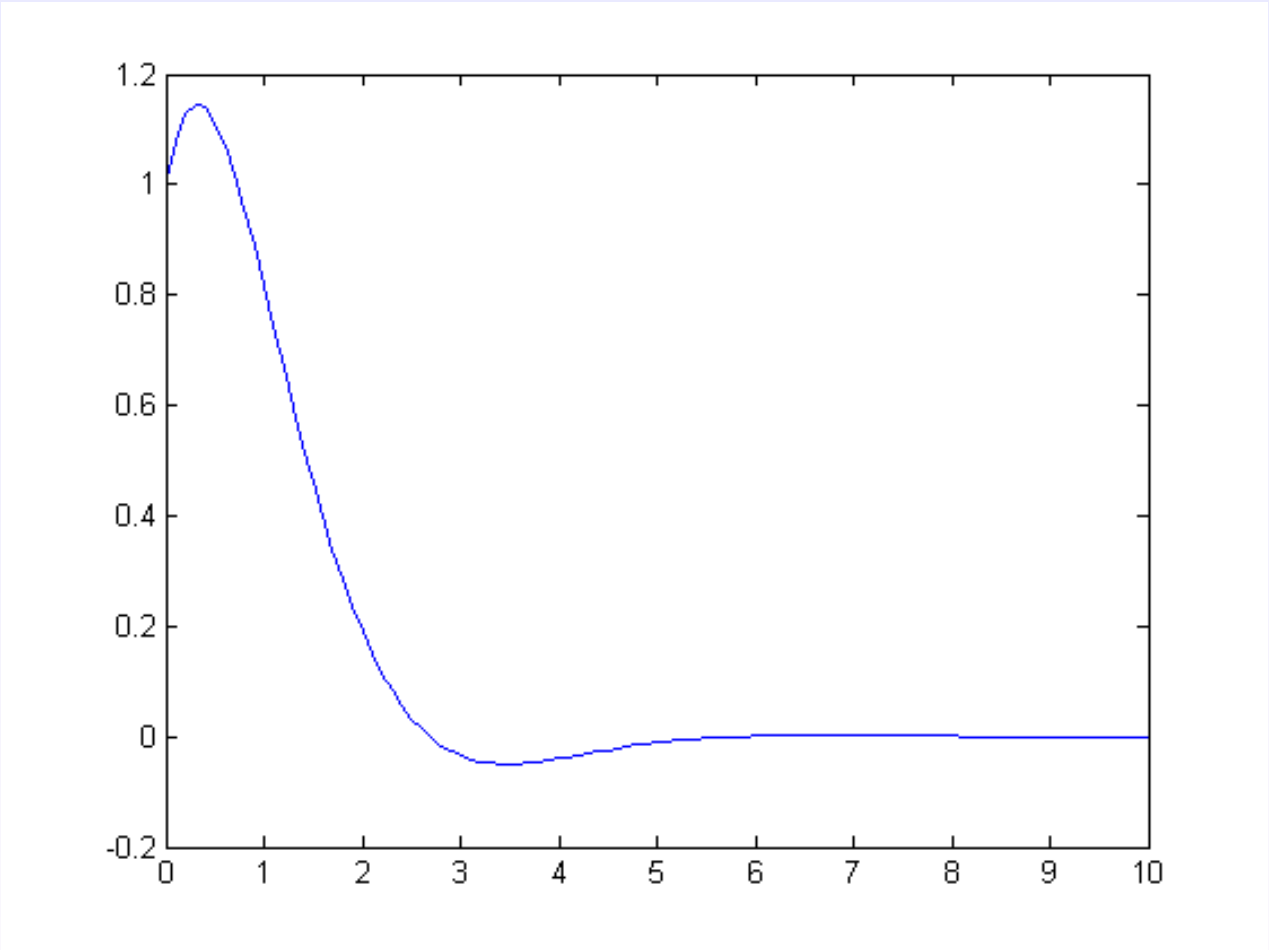
The equation has a general solution:

$$y = e^{-t} \cdot (C_1 \cos(t) + C_2 \sin(t))$$

For example, for initial conditions: $y(0) = 1$ i $\dot{y}(0) = 1$, we can find $C_1 = 1$ i $C_2 = 2$, and we have:

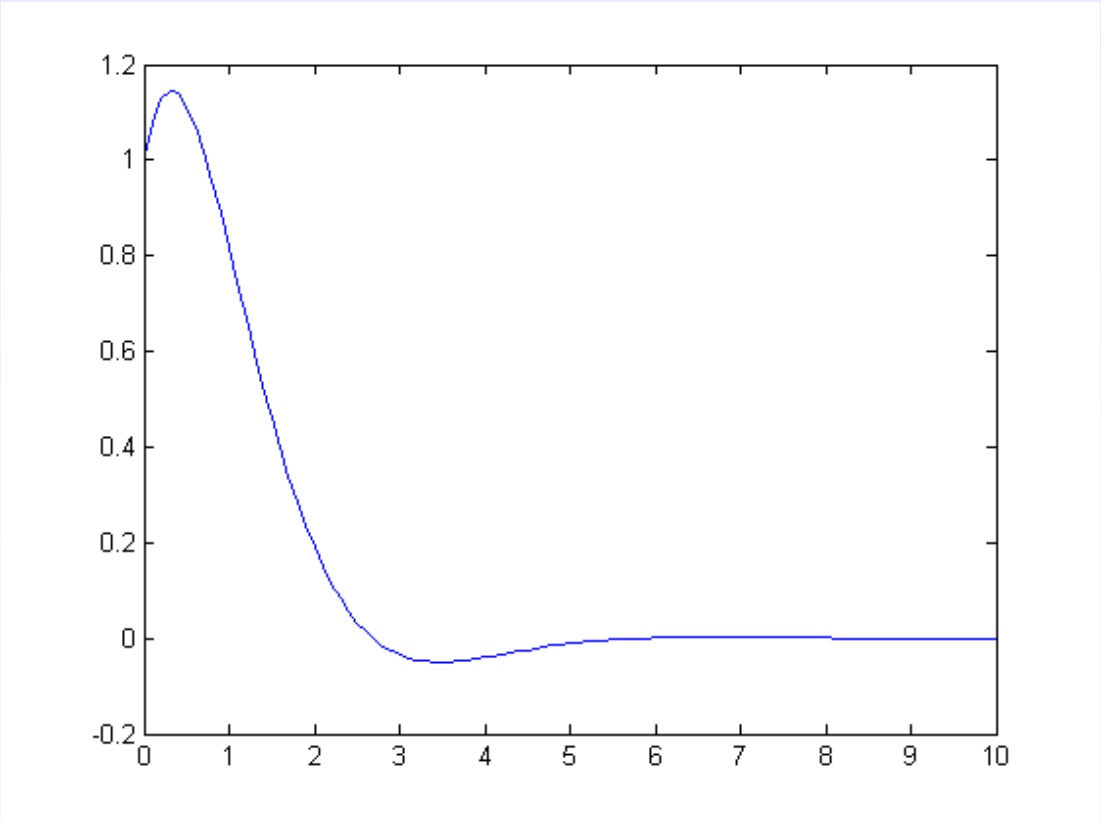
$$y = e^{-t} \cdot (\cos(t) + 2\sin(t))$$

Signal model – example



$$y = e^{-t} \cdot (\cos(t) + 2\sin(t))$$

Signal model – example



Differential equation $\ddot{y} + 2\dot{y} + 2y = 0$ can therefore be treated as a continuous model of the signal that we can see in the plot.

Signal model – example

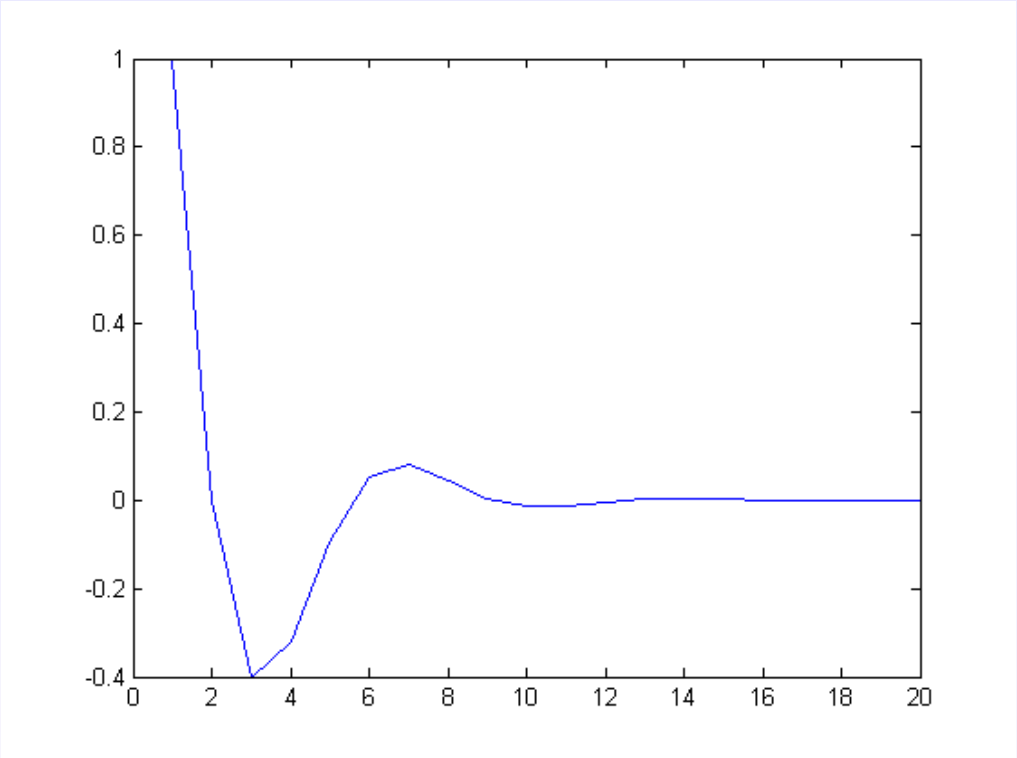
In computer modeling, the differential equation $\ddot{y} + 2\dot{y} + 2y = 0$ can be discretized.

Then, we will get a difference equation:

$$y(k) = y(k-1) \cdot \frac{2+2T}{1+2T+2T^2} - y(k-2) \cdot \frac{1}{1+2T+2T^2}$$

The form and parameters of the equation depend on the applied discretization method. Parameters also depend on the used sampling step T.

Signal model – example

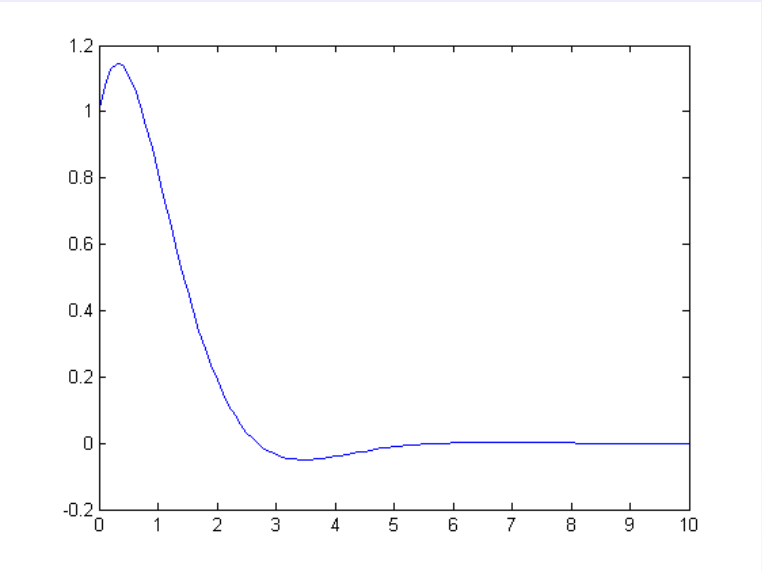


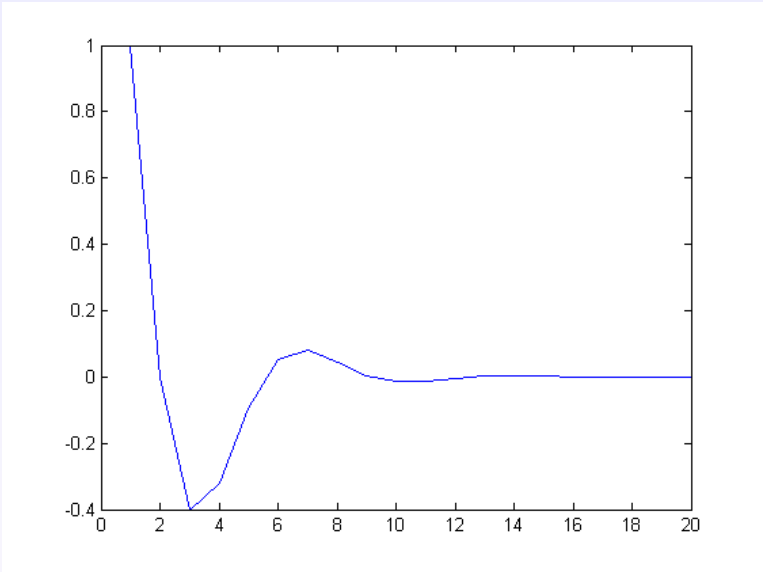
Simulation of the model described by the difference equation:

$$y(k) = y(k-1) \cdot \frac{2+2T}{1+2T+2T^2} - y(k-2) \cdot \frac{1}{1+2T+2T^2}$$

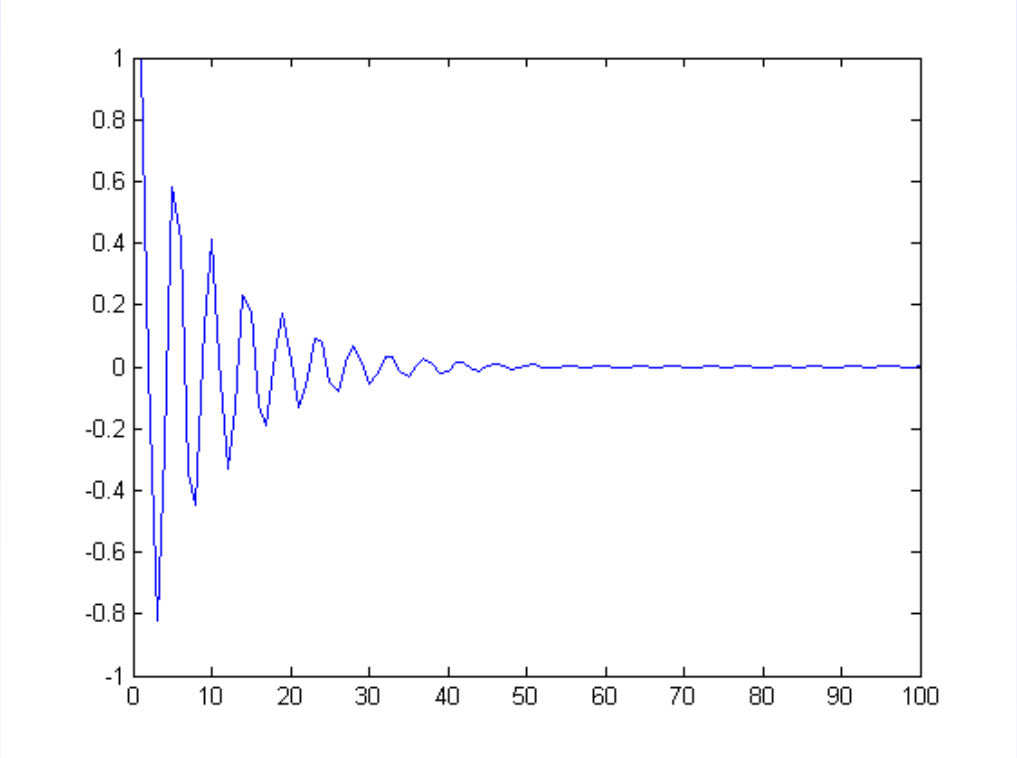
for the sampling step $T = 0.5$ and initial conditions $y(0) = 1$, $y(1) = 0$.

Signal model – example





Signal model – example



Discretization can significantly change the behavior of the model.The plot shows the simulation of the model described by the difference equation:

$$y(k) = y(k-1) \cdot \frac{2+2T}{1+2T+2T^2} - y(k-2) \cdot \frac{1}{1+2T+2T^2}$$

for the sampling step $T = 0.1$ and initial conditions $y(0) = 1$, $y(1) = 0$.

Signal model – example

Difference equation:

$$y(k) = y(k-1) \cdot \frac{2+2T}{1+2T+2T^2} - y(k-2) \cdot \frac{1}{1+2T+2T^2}$$

can be presented more generally in the form:

$$y(k) = w_1 \cdot y(k-1) + w_2 \cdot y(k-2)$$

where: $w_1 = \frac{2+2T}{1+2T+2T^2}$ i $w_2 = \frac{1}{1+2T+2T^2}$

Such an equation can be easily modeled by one linear neuron!

Signal model

Using more complex networks, we can model any (linear or non-linear) difference equation with a general form:

$$y(k) = F(y(k-1), y(k-2), \ldots y(k-n))$$

n – determines the size of the time window, i.e. we will use the n previous signal values to predict the current value.

The neural network will have n inputs (previous signal values) and one output (predicted actual value).

Data preparation

Let us assume that we know the stock prices of a certain company:

Day 1	56
Day 2	58
Day 3	55
Day 4	53
Day 5	52
Day 6	50

Data preparation

Day 1	56
Day 2	58
Day 3	55
Day 4	53
Day 5	52
Day 6	50

If we assume a size of the window equal 3, training samples will take the form:

Sample 1:	56 58 55	53
Sample 2:	58 55 53	52
Sample 3:	55 53 52	50

Data preparation

Day 1	56
Day 2	58
Day 3	55
Day 4	53
Day 5	52
Day 6	50

If we assume a size of the window equal 3, training samples will take the form:

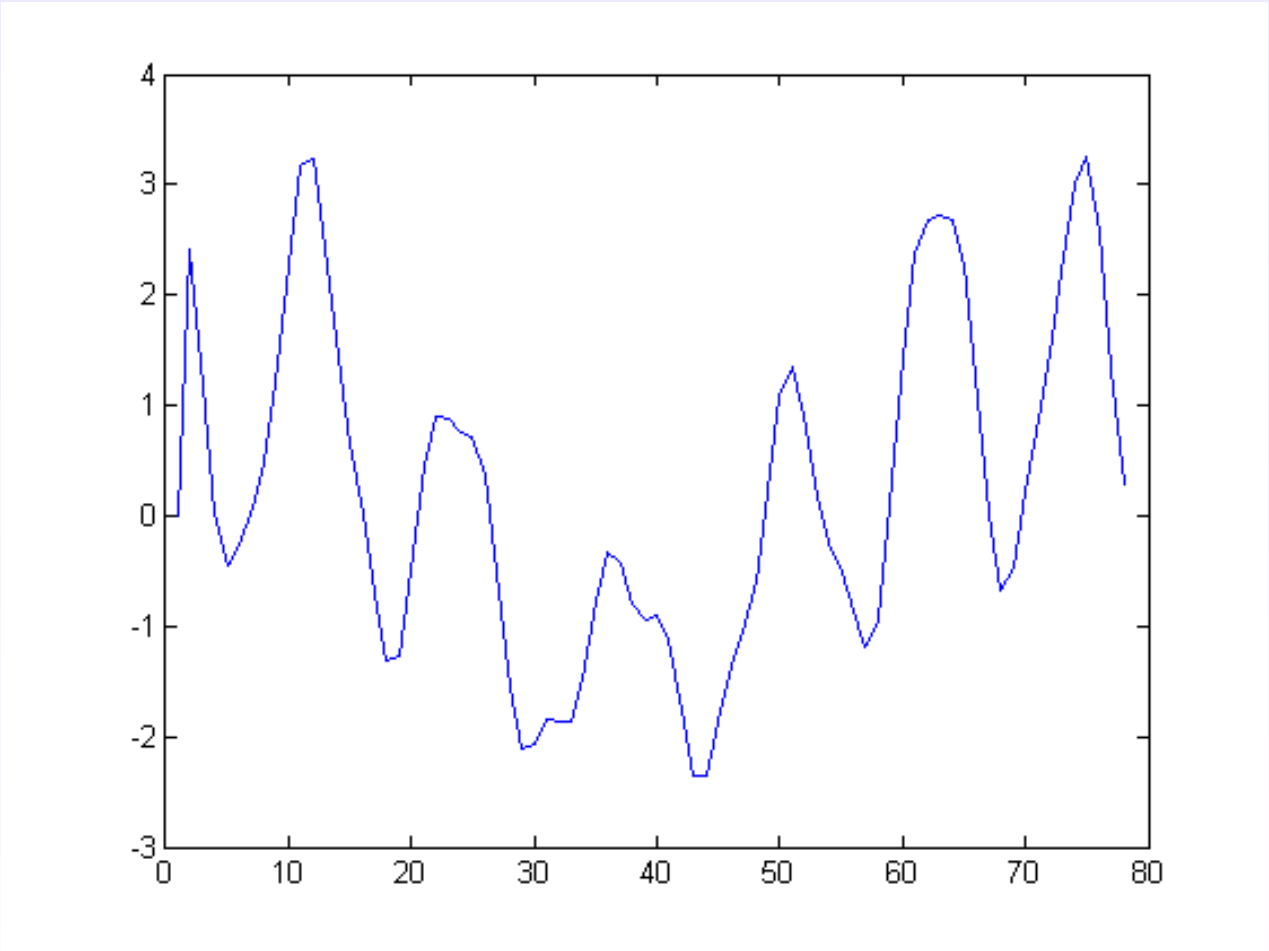
Sample 1:	56 58 55	53
Sample 2:	58 55 53	52
Sample 3:	55 53 52	50

The network learned with their help will model the difference equation:

$$y(k) = F(y(k-1), y(k-2), y(k-3))$$

Examples

The signal is given:
$y = \sin(t) + \cos(t) + \sin(3t) \cdot \cos(3t) + \sin(5t) + \cos(5t) + 0.3\sin(50t)$



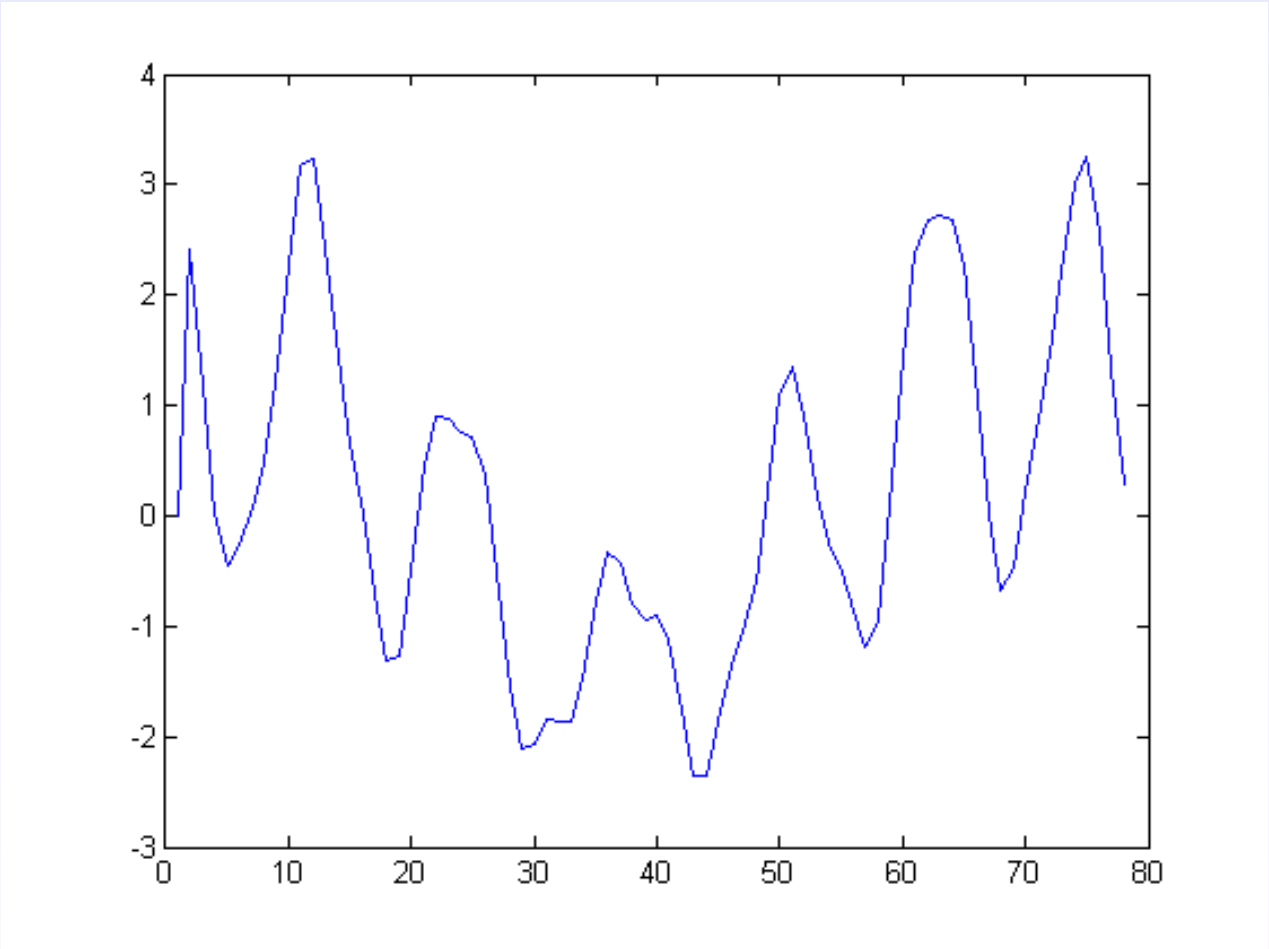
Examples

- We choose the size of the time window $n = 4$.
- The network will model the difference equation:

$$y(k) = F(y(k-1), y(k-2), y(k-3), y(k-4))$$

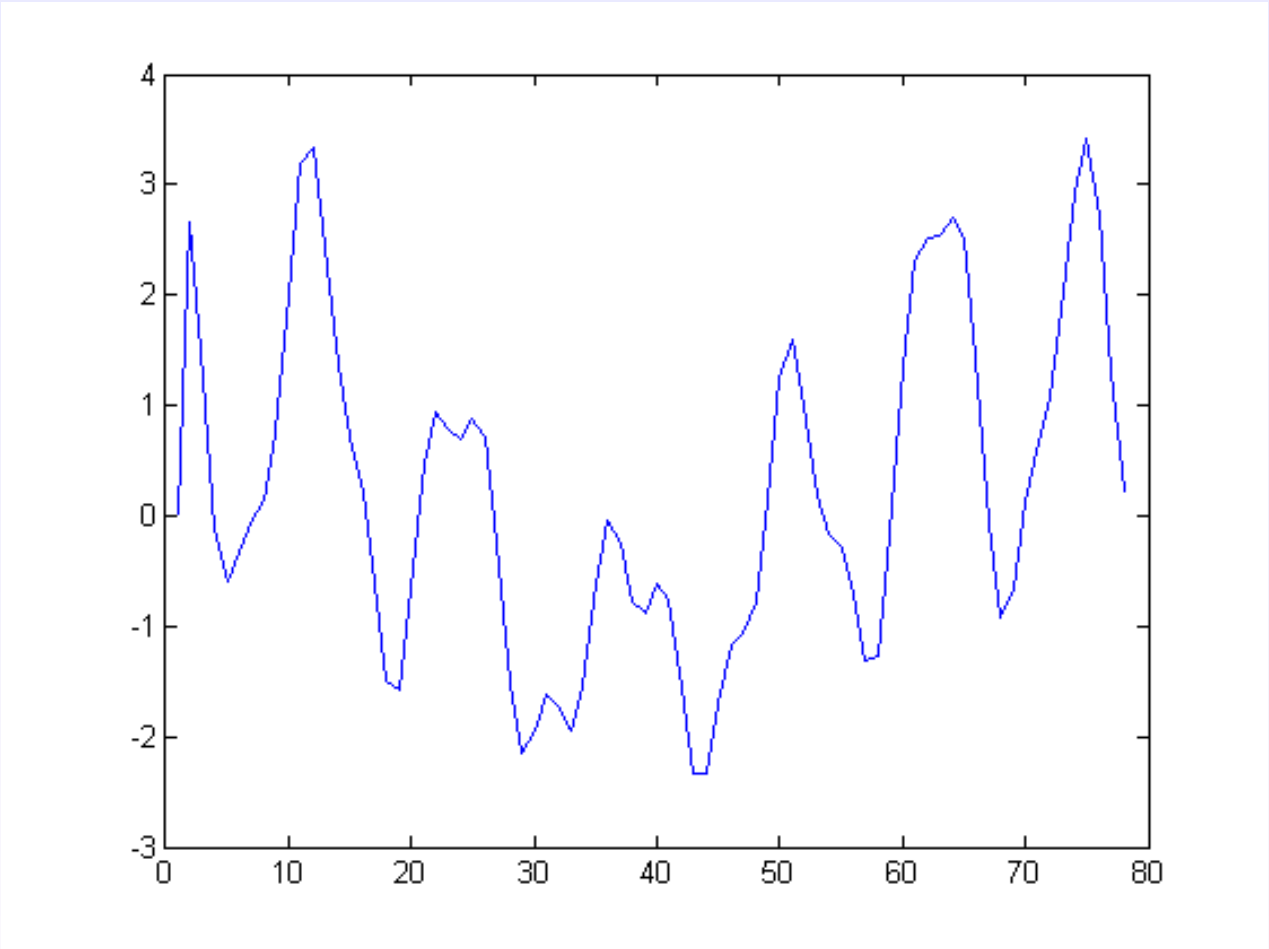
- We take 6 neurons on the hidden layer.
- The hidden layer has hyperbolic tangent activation function, the output layer is linear.
- We prepare learning data as shown before.

Examples



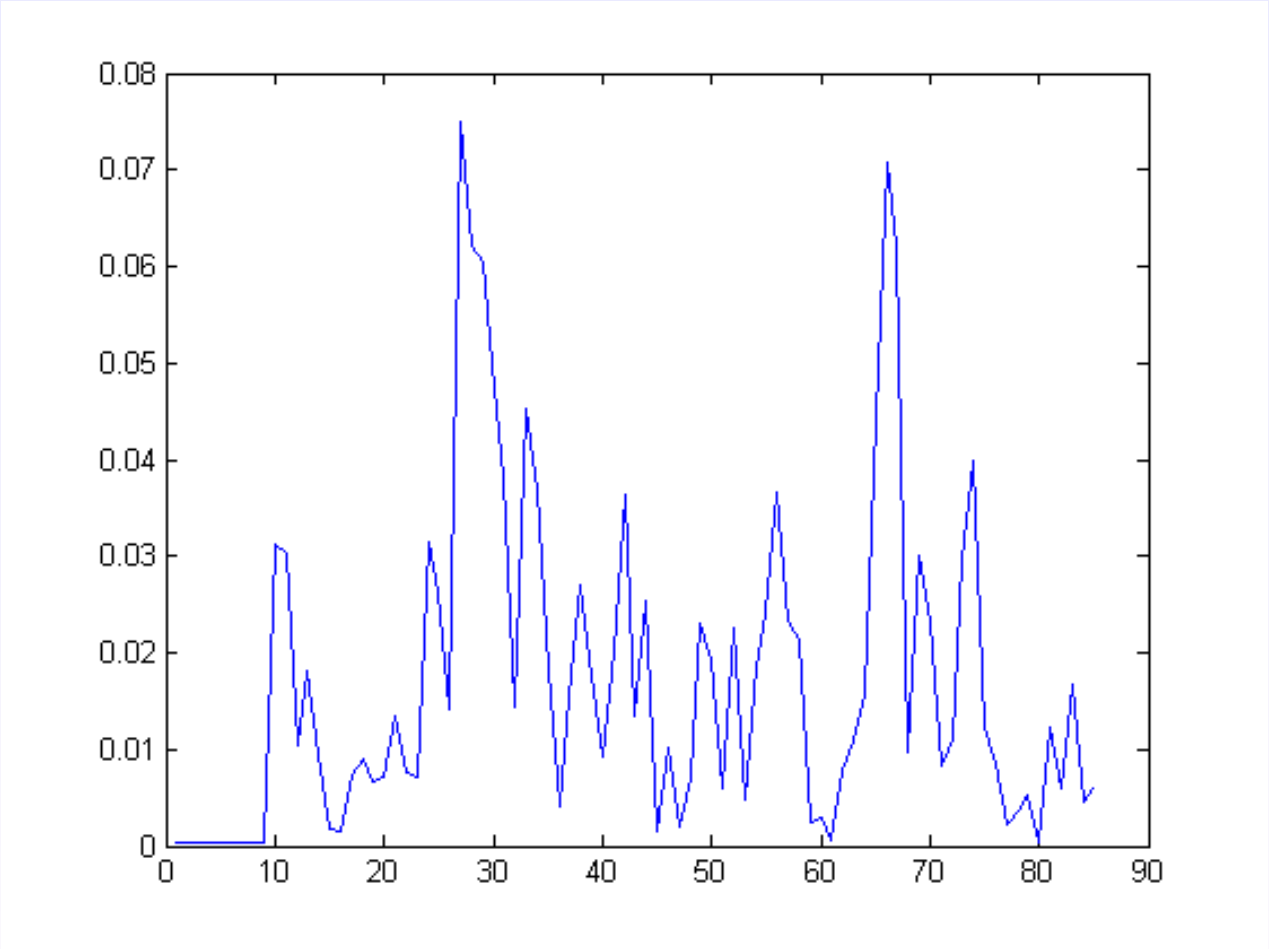
Learning signal.

Examples



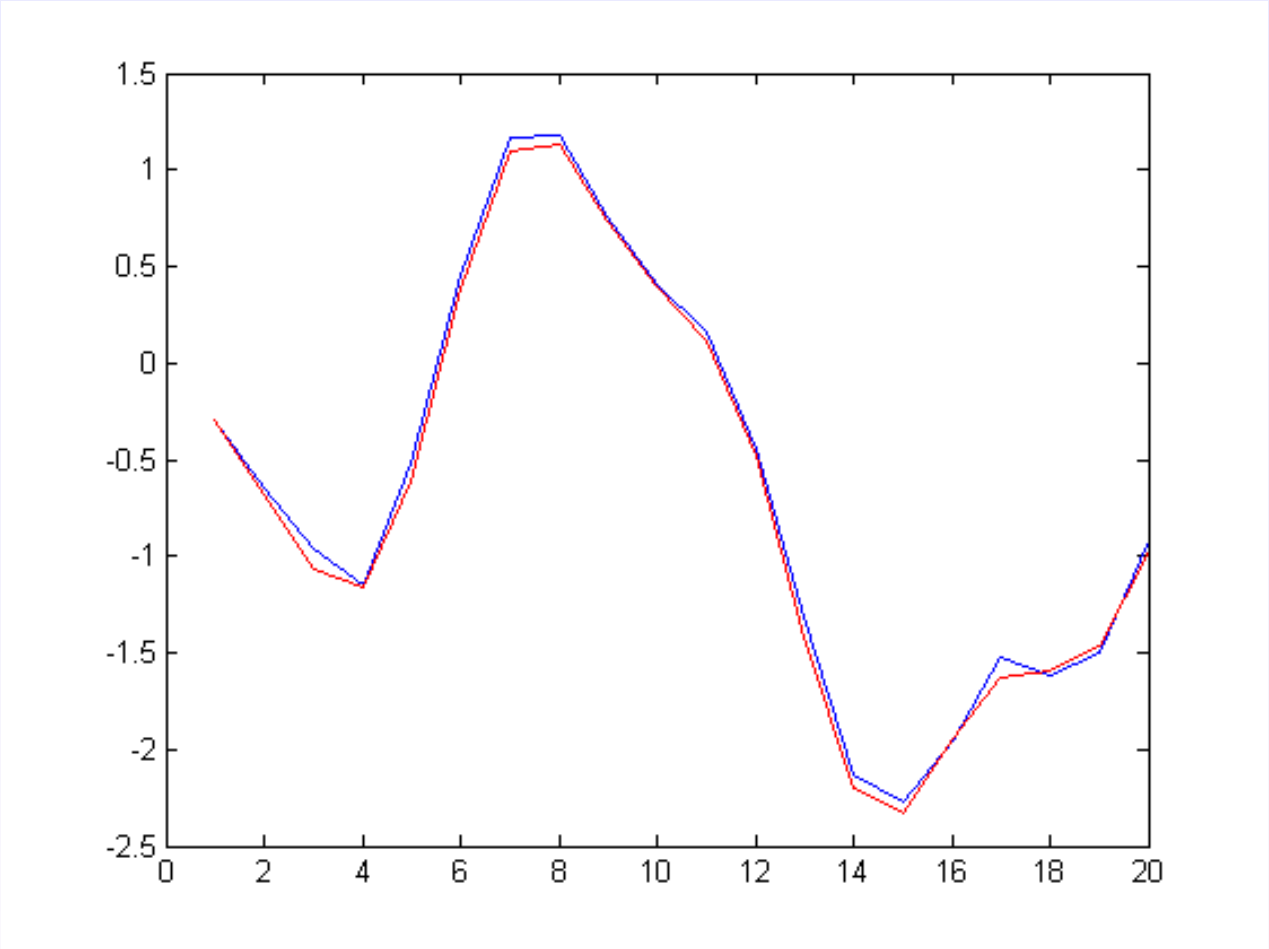
Signal modeled by the neural network.

Examples



Error of the network.

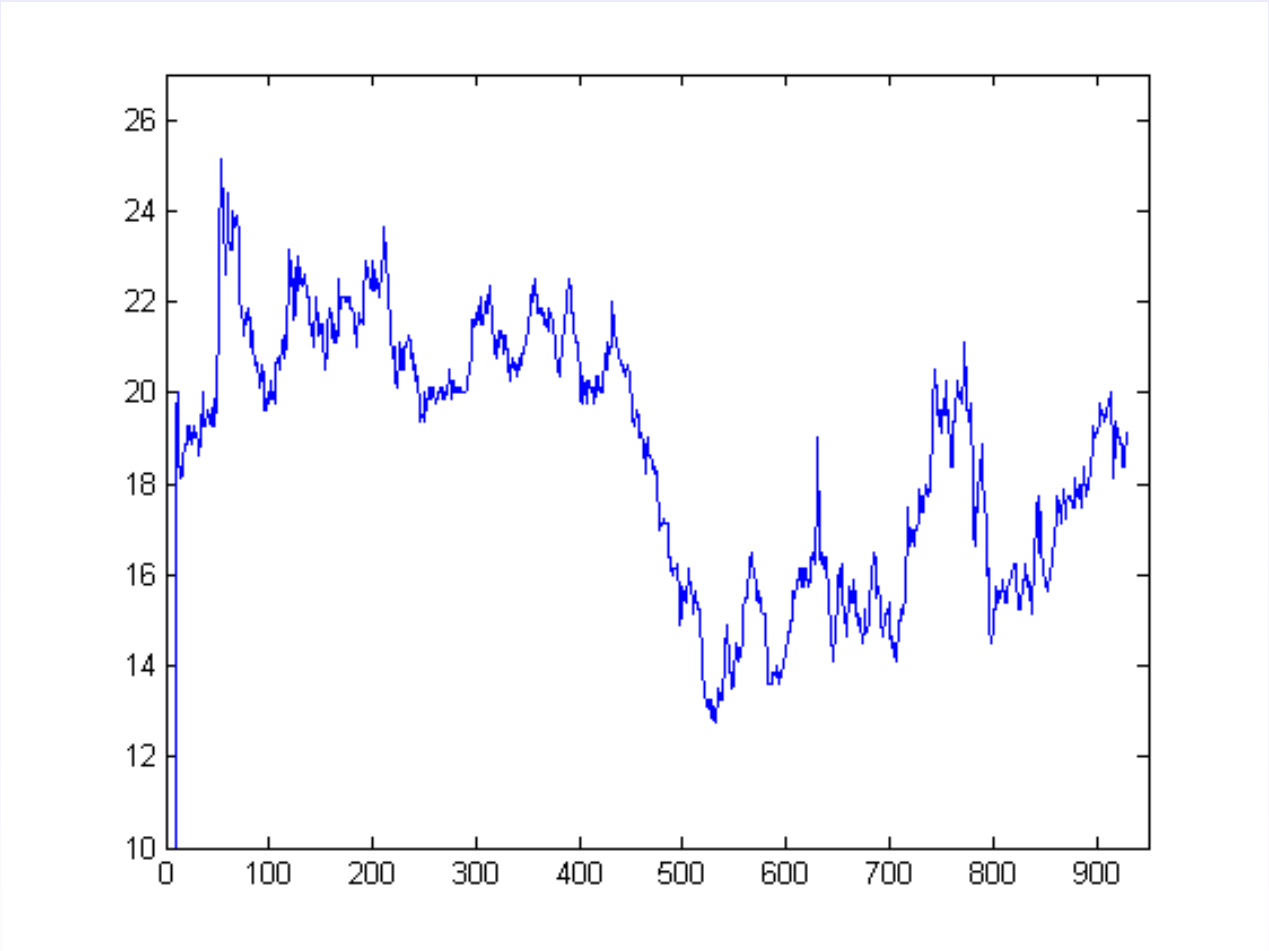
Examples



Prediction for data not used in learning. Blue line – prediction, red line – real value.

Examples

We have data of the stock value of a certain company:



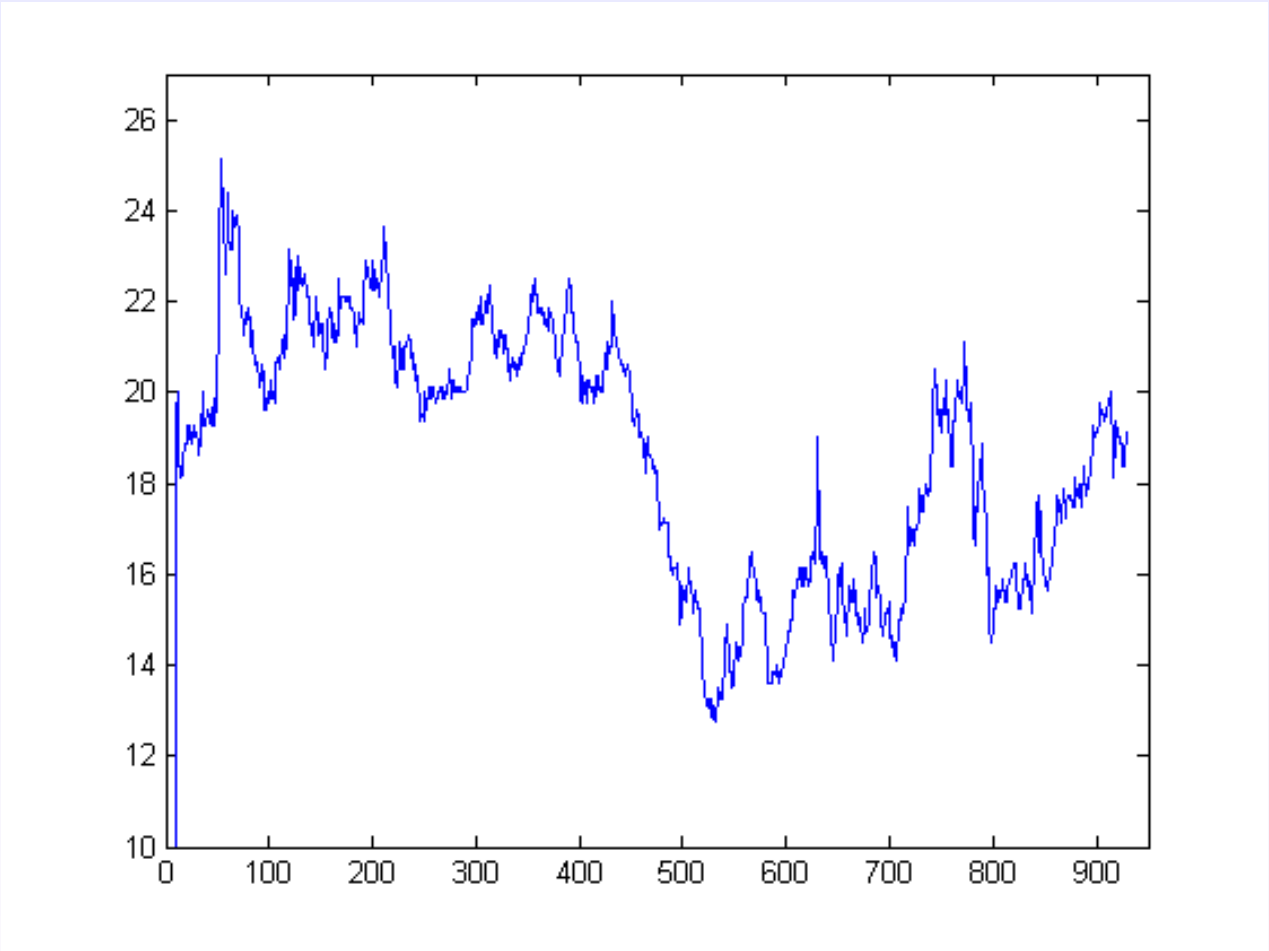
Examples

- We choose the size of the time window $n = 5$.
- The network will model the difference equation:

$$y(k) = F(y(k-1), y(k-2), y(k-3), y(k-4), y(k-5))$$

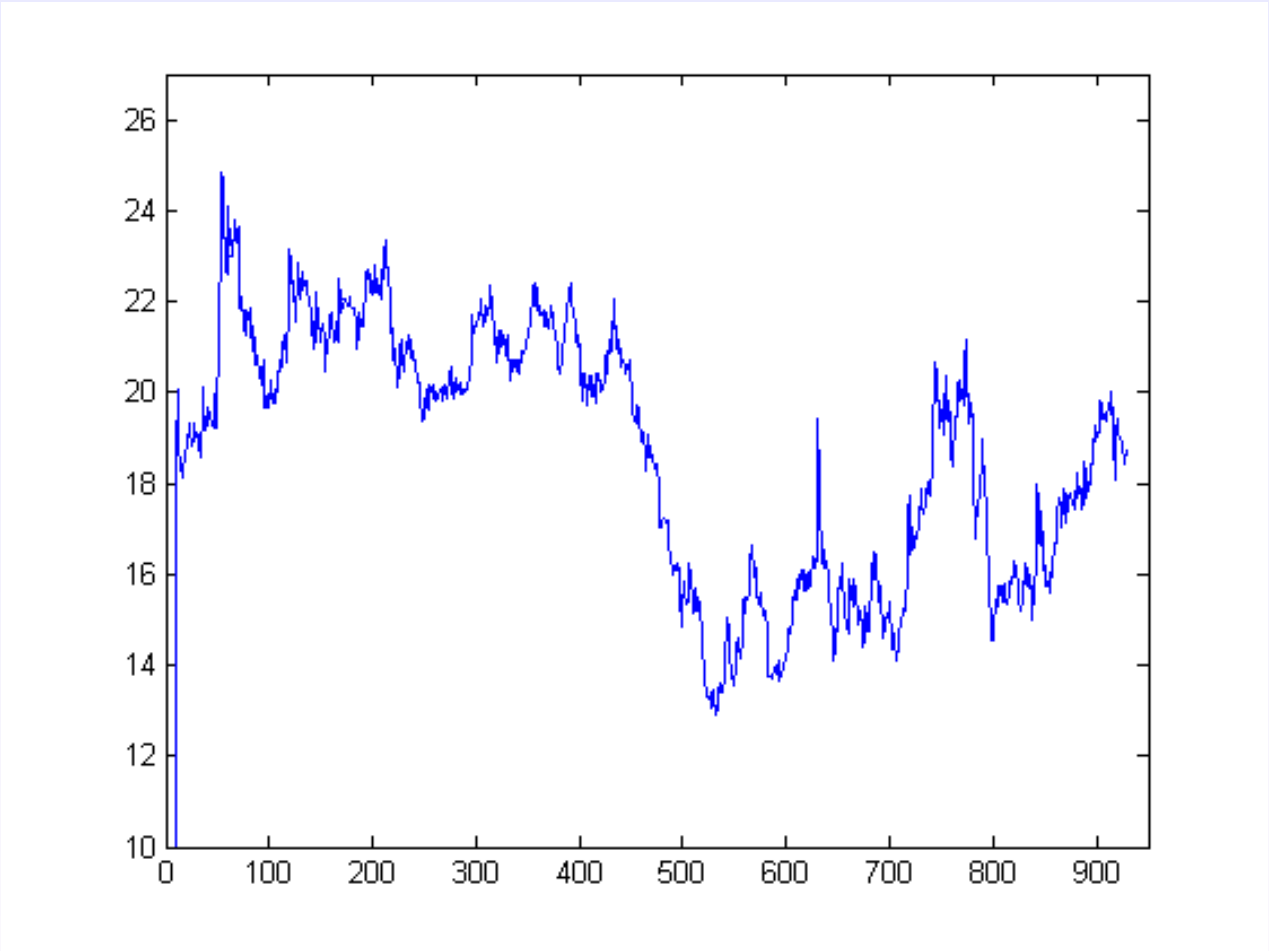
- We take 6 neurons on the hidden layer.
- The hidden layer has hyperbolic tangent activation function, the output layer is linear.
- We prepare learning data as shown before.

Examples



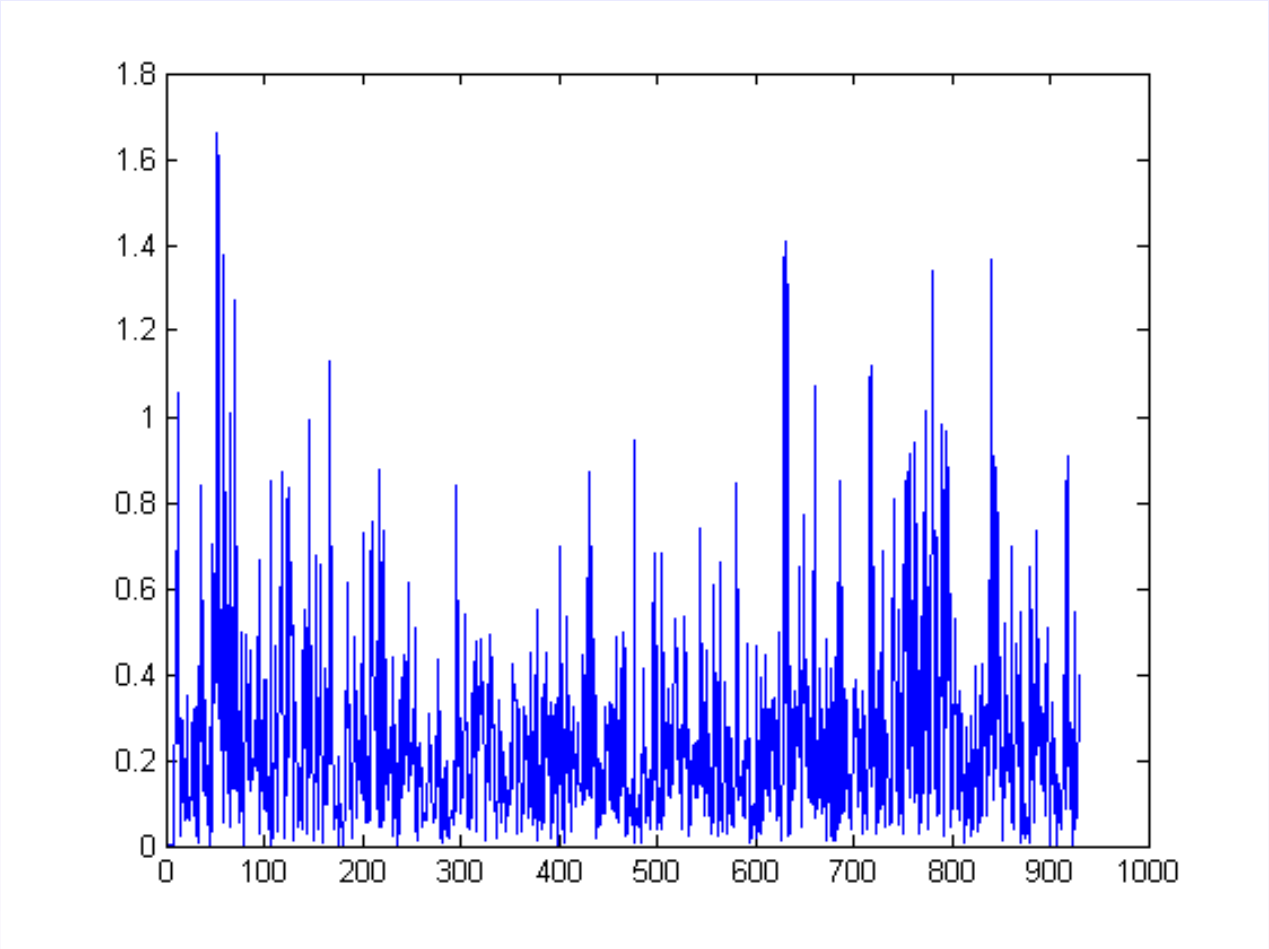
Learning signal.

Examples



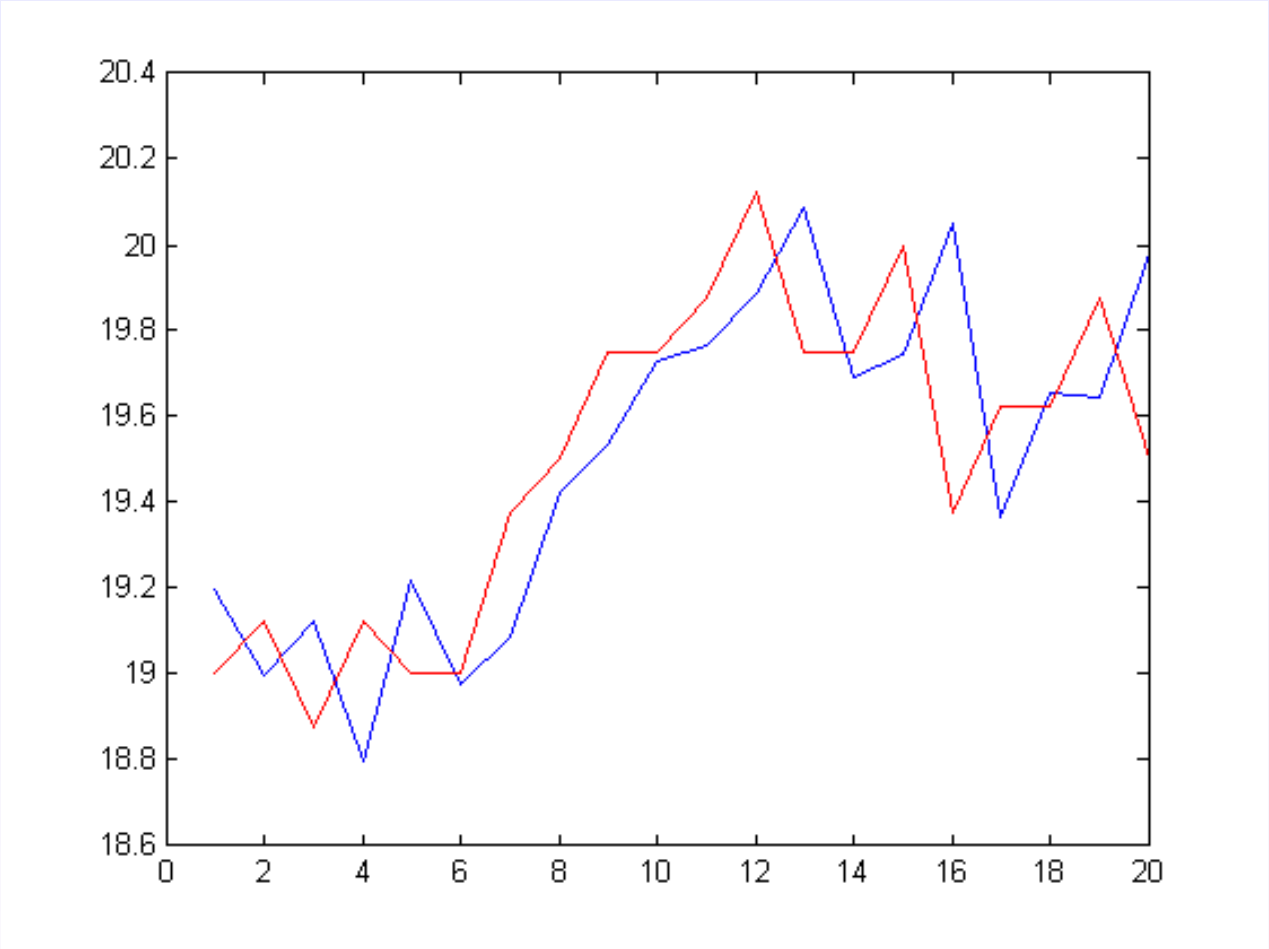
Signal modeled by the neural network.

Examples



Error of the network.

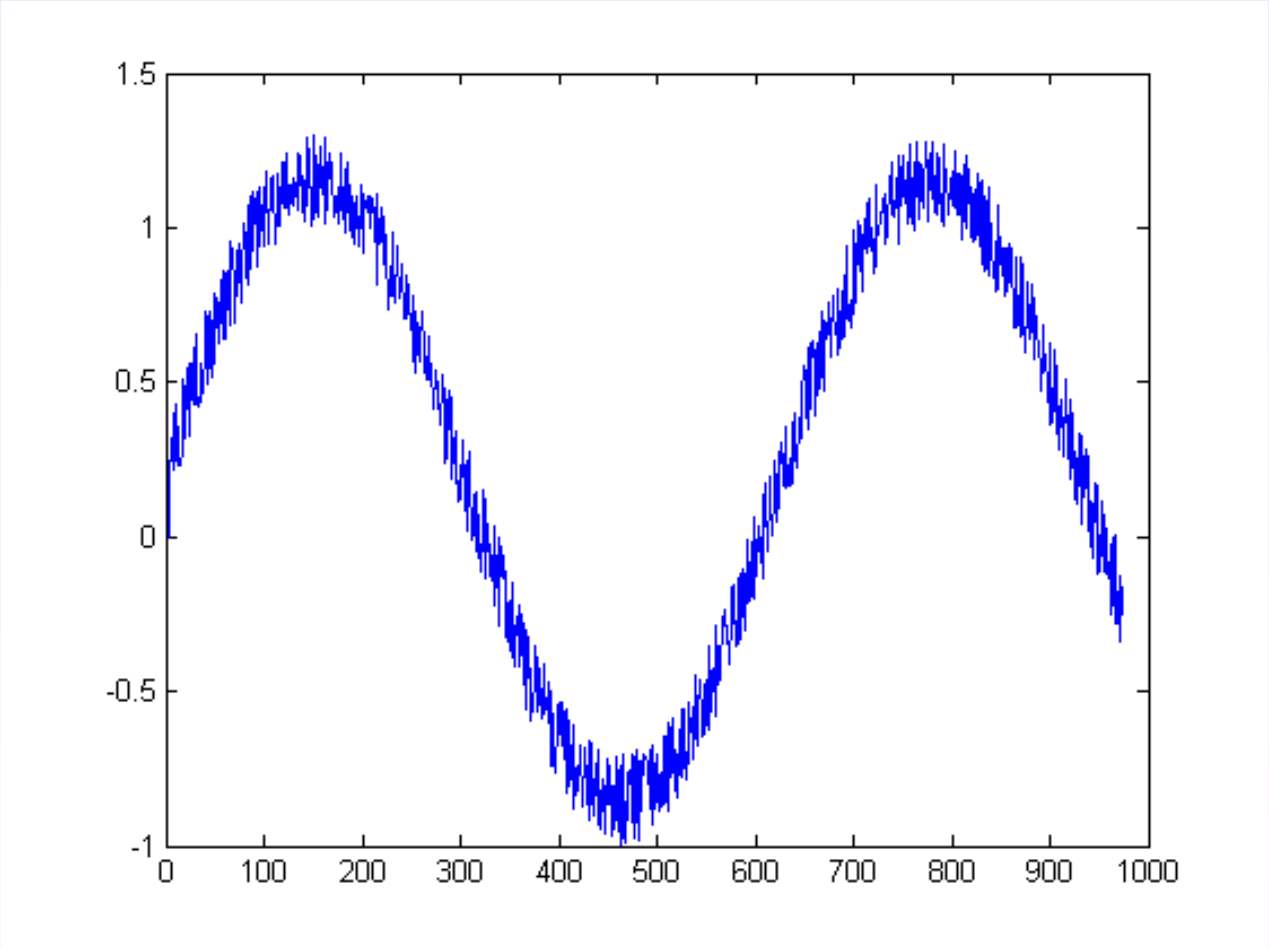
Examples



Prediction for data not used in learning. Blue line – prediction, red line – real value.

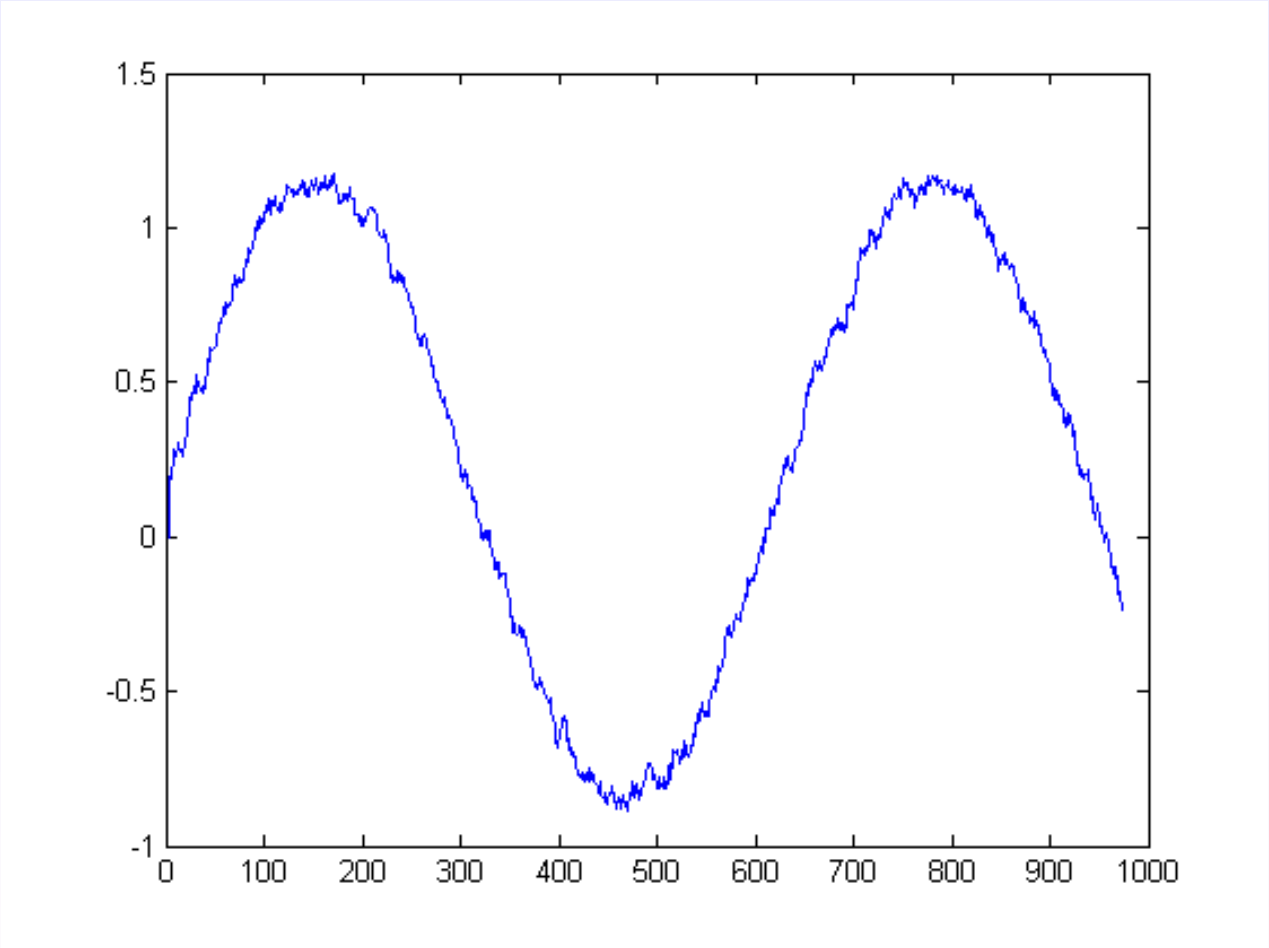
Examples

Learning signal – noisy sinusoid.



Examples

The signal modeled by the network. The noise filtering effect is obtained using a simple network composed of one linear neuron.



Modeling of dynamic objects

Tasks:

1. identification,
2. control.

Modeling of dynamic objects

The general model of a dynamic object has the form of a differential equation:

$$F(y^{(n)}(t), y^{(n-1)}(t), \ldots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \ldots, \dot{x}(t), x(t)) = 0$$

For real objects we always have: $n \geq m$.

Modeling of dynamic objects

The general model of a dynamic object has the form of a differential equation:

$$F(y^{(n)}(t), y^{(n-1)}(t), \ldots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \ldots, \dot{x}(t), x(t)) = 0$$

For real objects we always have: $n \geq m$.

If the object is linear, it can be described by a linear differential equation:

$$a_n y^{(n)}(t) + a_{n-1} y^{(n-1)}(t) + \ldots + a_1 \dot{y}(t) + a_0 y(t) = \\ = b_m x^{(m)}(t) + b_{m-1} x^{(m-1)}(t) + \ldots + b_1 \dot{x}(t) + b_0 x(t)$$

Modeling of dynamic objects

Linear differential equation:

$$a_n y^{(n)}(t) + a_{n-1} y^{(n-1)}(t) + \ldots + a_1 \dot{y}(t) + a_0 y(t) = \\ = b_m x^{(m)}(t) + b_{m-1} x^{(m-1)}(t) + \ldots + b_1 \dot{x}(t) + b_0 x(t)$$

has the form of a linear difference equation after discretization:

$$y(k) = A_1 y(k-1) + A_2 y(k-2) + \ldots + A_n y(k-n) + \\ + B_0 x(k) + B_1 x(k-1) + \ldots + B_m x(k-m)$$

Such an equation can be modeled using one linear neuron.

Modeling of dynamic objects

Non-linear differential equation:

$$F(y^{(n)}(t), y^{(n-1)}(t), \ldots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \ldots, \dot{x}(t), x(t)) = 0$$

has the form of a non-linear difference equation after discretization:

$$y(k) = F(y(k-1), y(k-2), \ldots, y(k-n),\ x(k), x(k-1), \ldots, x(k-m))$$

Modeling of dynamic objects

Non-linear differential equation:

$$F(y^{(n)}(t), y^{(n-1)}(t), \ldots, \dot{y}(t), y(t), x^{(m)}(t), x^{(m-1)}(t), \ldots, \dot{x}(t), x(t)) = 0$$

has the form of a non-linear difference equation after discretization:

$$y(k) = F(y(k-1), y(k-2), \ldots, y(k-n),\ x(k), x(k-1), \ldots, x(k-m))$$

- Such equation can be modeled using a non-linear multi-layer network.
- n – determines the size of the time window for the output signal, a m – determines the size of the time window for the input signal.
- The neural network will have $n + m + 1$ inputs and 1 output.

Data preparation

Let's assume that we want to create a model of dynamics for changes in fuel prices depending on the dollar exchange rate. Every week we calculate the average dollar exchange rate and the average price of fuel.

	Dollar	Fuel
Week 1	3.20	4.70
Week 2	3.25	4.88
Week 3	3.15	4.73
Week 4	3.10	4.65
Week 5	3.12	4.60
Week 6	3.21	4.68

Data preparation

	Dollar	Fuel
Week 1	3.20	4.70
Week 2	3.25	4.88
Week 3	3.15	4.73
Week 4	3.10	4.65
Week 5	3.12	4.60
Week 6	3.21	4.68

If we choose the size of the time window for output $n = 2$ and for input $m = 1$, learning sample will have the form:

Sample 1:	4.70 4.88 3.25 3.15	4.73
Sample 2:	4.88 4.73 3.15 3.10	4.65
Sample 3:	4.73 4.65 3.10 3.12	4.60
Sample 3:	4.65 4.60 3.12 3.21	4.68

Data preparation

	Dollar	Fuel
Week 1	3.20	4.70
Week 2	3.25	4.88
Week 3	3.15	4.73
Week 4	3.10	4.65
Week 5	3.12	4.60
Week 6	3.21	4.68

If we choose the size of the time window for output $n = 2$ and for input $m = 1$, learning sample will have the form:

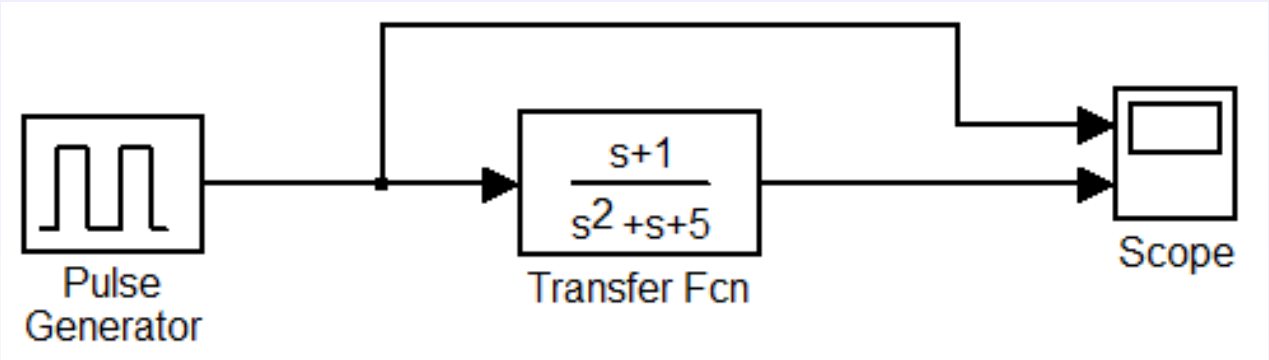
Sample 1:	4.70 4.88 3.25 3.15	4.73
Sample 2:	4.88 4.73 3.15 3.10	4.65
Sample 3:	4.73 4.65 3.10 3.12	4.60
Sample 3:	4.65 4.60 3.12 3.21	4.68

The network learned with their help will model the difference equation:

$$y(k) = F(y(k-1), y(k-2), x(k), x(k-1))$$

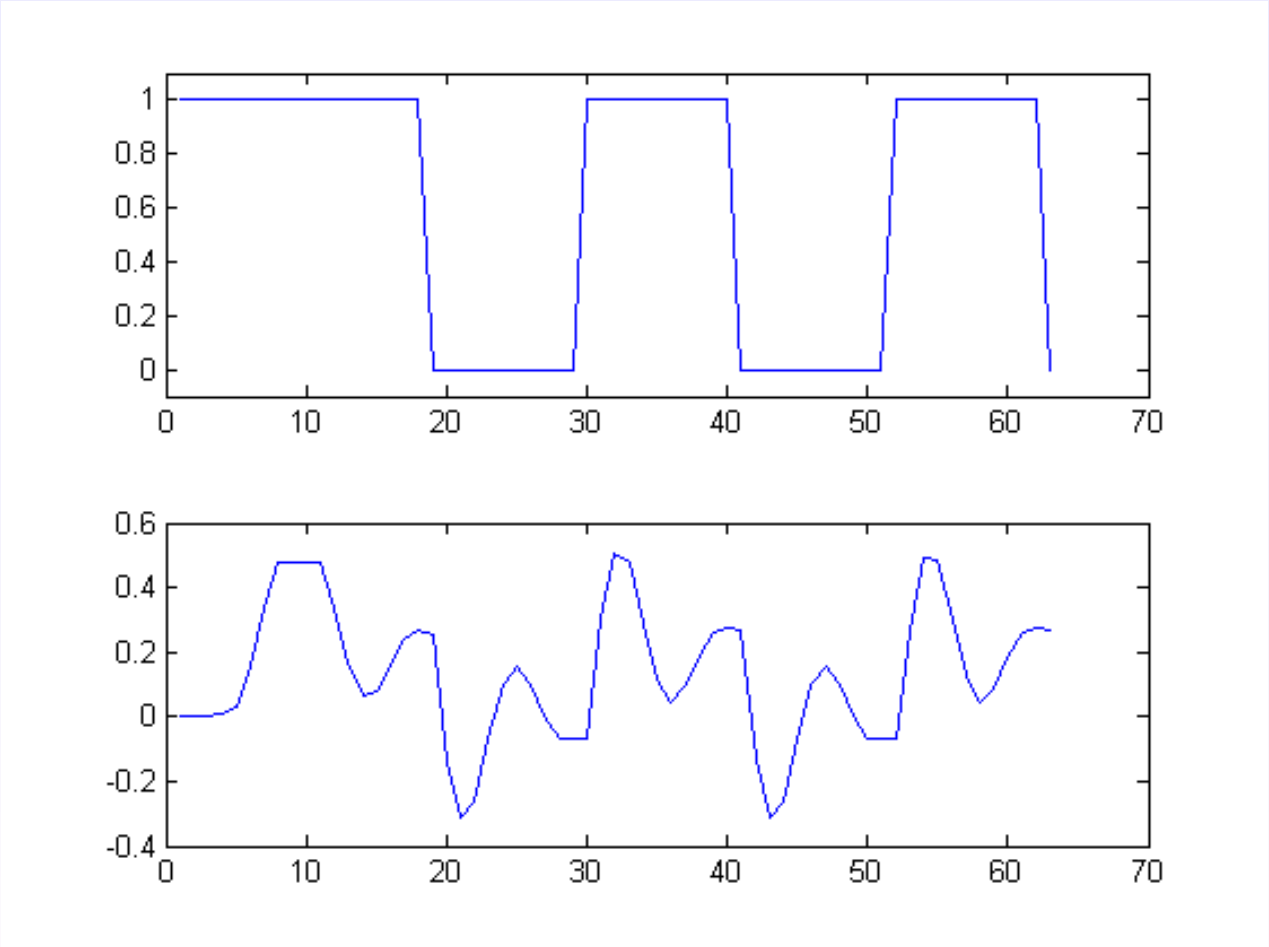
Example

The behavior of the dynamic object has been modeled:



Example

The input and output signal of the object was saved:



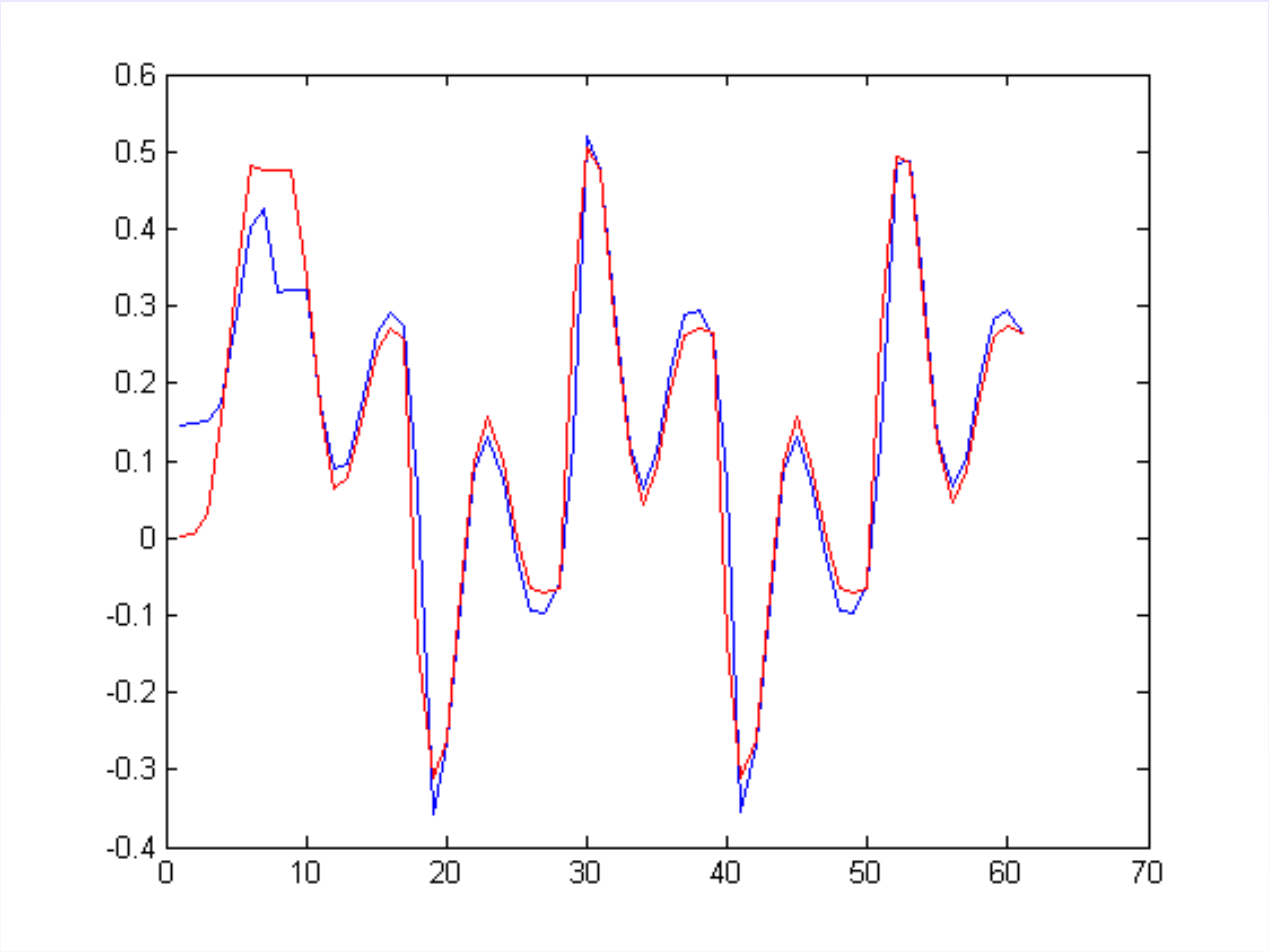
Example

- We choose the size of the time window for output $n = 2$ and input $m = 1$.
- The network will model the difference equation:

$$y(k) = F(y(k-1), y(k-2), x(k), x(k-1))$$

- We choose a network with one linear neuron.
- We prepare learning data as before.

Example



Red line – output training signal, blue line – signal generated by the network.

Neural network as a controller

Preparation of training data

```
@relation labor
@attribute 'duration' real
@attribute 'wage-increase-first-year' real
@attribute 'wage-increase-second-year' real
@attribute 'wage-increase-third-year' real
@attribute 'cost-of-living-adjustment' {'none','tcf','tc'}
@attribute 'working-hours' real
@attribute 'pension' {'none','ret_allw','empl_contr'}
@attribute 'standby-pay' real
@attribute 'shift-differential' real
@attribute 'education-allowance' {'yes','no'}
@attribute 'statutory-holidays' real
@attribute 'vacation' {'below_average','average','generous'}
@attribute 'longterm-disability-assistance' {'yes','no'}
@attribute 'contribution-to-dental-plan' {'none','half','full'}
@attribute 'bereavement-assistance' {'yes','no'}
@attribute 'contribution-to-health-plan' {'none','half','full'}
@attribute 'class' {'bad','good'}
@data
1,5,?,?,?,40,?,?,2,?,11,'average',?,?,'yes',?,'good'
2,4.5,5.8,?,?,35,'ret_allw',?,?,'yes',11,'below_average',?,'full',?,'full','good'
?,?,?,?,?,38,'empl_contr',?,5,?,11,'generous','yes','half','yes','half','good'
3,3.7,4,5,'tc',?,?,?,?,'yes',?,?,?,?,'yes',?,'good'
3,4.5,4.5,5,?,40,?,?,?,?,12,'average',?,'half','yes','half','good'
2,2,2.5,?,?,35,?,?,6,'yes',12,'average',?,?,?,?,'good'
3,4,5,5,'tc',?,'empl_contr',?,?,?,12,'generous','yes','none','yes','half','good'
3,6.9,4.8,2.3,?,40,?,?,3,?,12,'below_average',?,?,?,?,'good'
2,3,7,?,?,38,?,12,25,'yes',11,'below_average','yes','half','yes',?,'good'
1,5.7,?,?,'none',40,'empl_contr',?,4,?,11,'generous','yes','full',?,?,'good'
3,3.5,4,4.6,'none',36,?,?,3,?,13,'generous',?,?,'yes','full','good'
2,6.4,6.4,?,?,38,?,?,4,?,15,?,?,'full',?,?,'good'
```

Attribute types

- Nominal attributes: they have no numerical value, the only relations are 'equality' and 'inequality'.
- Order attributes: a relation of order is defined for them.
- Interval attributes: a measure of the distance between attributes is defined. The zero position in the interval scale is arbitrary.
- Real attributes.

Data normalization

- Normalization is not always necessary, but almost always recommended.
- Unification of the importance and significance of attributes.
- Normalization of outputs (in the case of multi-output models) – important due to minimization of error for each output.
- Easier interpretation of the weight values.
- Some networks (eg: RBF, Kohonen) require normalization for correct work.

Missing attributes

- Removing of samples.
- Completing of attributes.
- Sample splitting.
- For nominal attributes - treating a missing attribute as another possible value.