

Feedforward Neural Networks. Classification and Approximation

- ❑ Classification and Approximation Problems
- ❑ BackPropagation (BP) Neural Networks
- ❑ Radial Basis Function (RBF) Networks
- ❑ Support Vector Machines

Classification problems

Example 1: identifying the type of an iris flower



- **Attributes:** sepal/petal lengths, sepal/petal width
- **Classes:** Iris setosa, Iris versicolor, Iris virginica



Example 2: handwritten character recognition

- **Attributes:** various statistical and geometrical characteristics of the corresponding image
- **Classes:** set of characters to be recognized

⇒ **Classification = find the relationship between some vectors with attribute values and classes labels**

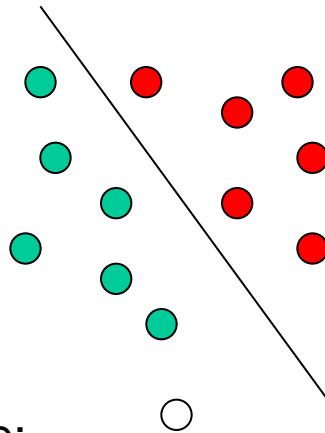
(Du Trier et al; Feature extraction methods for character Recognition. A Survey. Pattern Recognition, 1996)



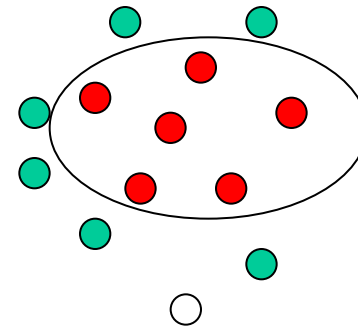
Classification problems

Classification:

- Problem: identify the class to which a given data (described by a set of attributes) belongs
- Prior knowledge: examples of data belonging to each class



Simple example:
linearly separable case



A more difficult example:
nonlinearly separable case

Approximation problems

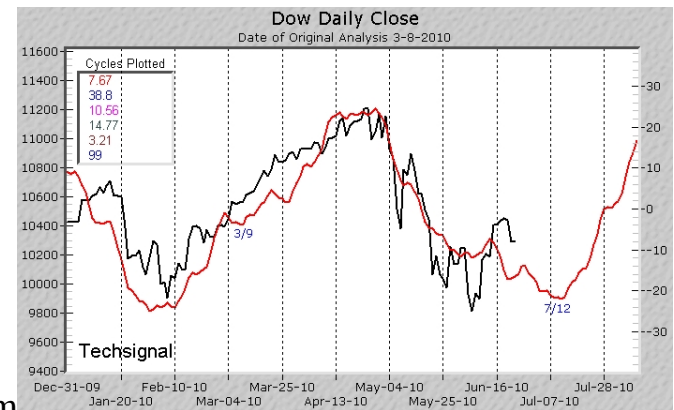
- Estimation of a house price knowing:

- Total surface
- Number of rooms
- Size of the back yard
- Location

=> approximation problem = find a numerical relationship between some output and input value(s)

- Estimating the amount of resources required by a software application or the number of users of a web service or a stock price knowing historical values

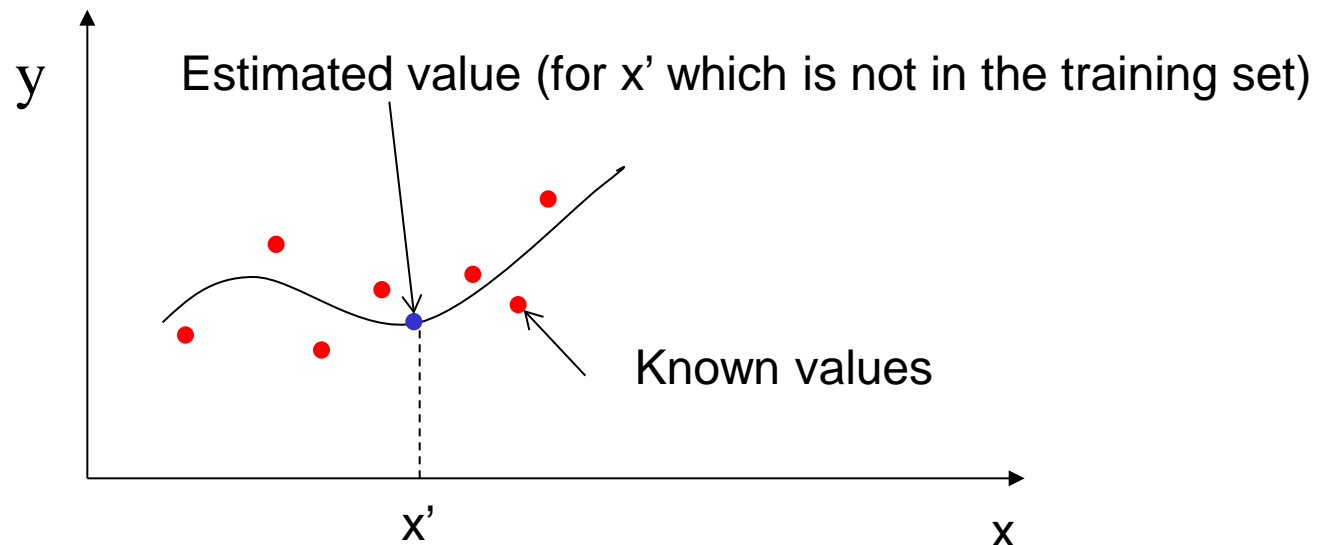
=> prediction problem=
find a relationship between future values and previous values



Approximation problems

Regression (fitting, prediction):

- **Problem:** estimate the value of a characteristic depending on the values of some predicting characteristics
- **Prior knowledge:** pairs of corresponding values (training set)



Approximation problems

All approximation (mapping) problems can be stated as follows:

Starting from a set of data (X_i, Y_i) , X_i in \mathbb{R}^N and Y_i in \mathbb{R}^M find a function $F: \mathbb{R}^N \rightarrow \mathbb{R}^M$ which minimizes the distance between the data and the corresponding points on its graph: $\|Y_i - F(X_i)\|^2$

Questions:

- What structure (shape) should have F ?
- How can we find the parameters defining the properties of F ?

Approximation problems

Can be such a problem be solved by using neural networks ?

Yes, at least in theory, the neural networks are proven “**universal approximators**” [Hornik, 1985]:

“ Any continuous function can be approximated by a feedforward neural network having at least one hidden layer. The accuracy of the approximation depends on the number of hidden units.”

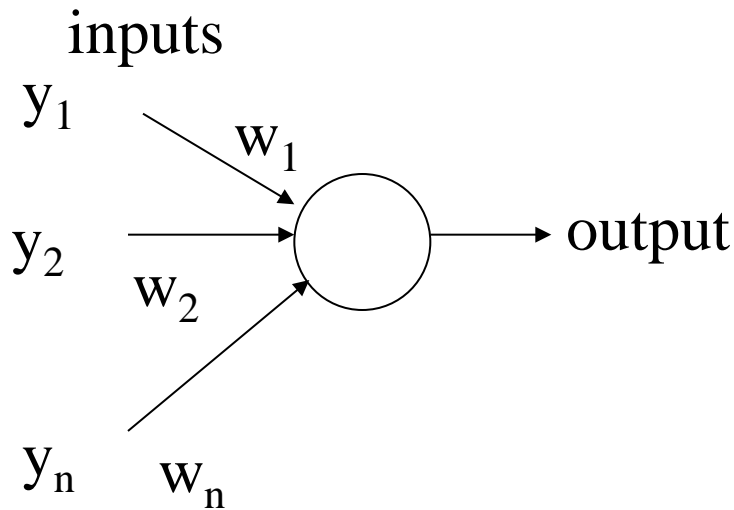
- The shape of the function is influenced by the **architecture** of the network and by the properties of the **activation functions**.
- The function parameters are in fact the **weights** corresponding to the connections between neurons.

Neural Networks Design

Steps to follow in designing a neural network:

- **Choose the architecture:** number of layers, number of units on each layer, activation functions, interconnection style
- **Train the network:** compute the values of the weights using the training set and a learning algorithm.
- **Validate/test the network:** analyze the network behavior for data which do not belong to the training set.

Functional units (neurons)



Weights assigned to the connections

Functional unit: several inputs, one output

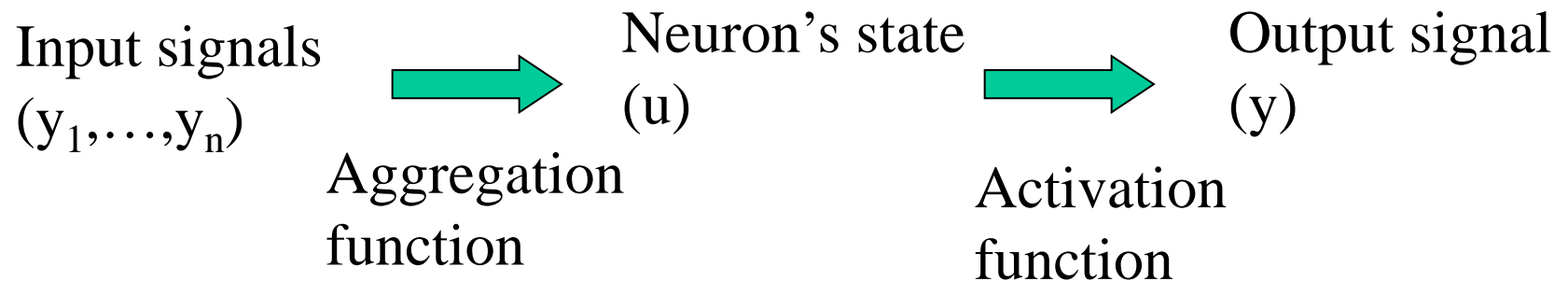
Notations:

- **input signals:** y_1, y_2, \dots, y_n
- **synaptic weights:** w_1, w_2, \dots, w_n
(they model the synaptic permeability)
- **threshold (bias):** b (or θ)
(it models the activation threshold of the neuron)
- **Output:** y
- All these values are usually real numbers

Functional units (neurons)

Output signal generation:

- The input signals are “combined” by using the connection weights and the threshold
 - The obtained value corresponds to the local potential of the neuron
 - This “combination” is obtained by applying a so-called **aggregation function**
- The output signal is constructed by applying an **activation function**
 - It corresponds to the pulse signals propagated along the axon



Functional units (neurons)

Aggregation functions:

Weighted sum

$$u = \sum_{j=1}^n w_j y_j - w_0$$

$$u = \prod_{j=1}^n y_j^{w_j}$$

Euclidean distance

$$u = \sqrt{\sum_{j=1}^n (w_j - y_j)^2}$$

$$u = \sum_{j=1}^n w_j y_j + \sum_{i,j=1}^n w_{ij} y_i y_j + \dots$$

Multiplicative neuron

High order connections

Remark: in the case of the weighted sum the threshold can be interpreted as a synaptic weight which corresponds to a virtual unit which always produces the value -1

$$u = \sum_{j=0}^n w_j y_j$$

Functional units (neurons)

Activation functions:

$$f(u) = \text{sgn}(u) = \begin{cases} -1 & u \leq 0 \\ 1 & u > 0 \end{cases} \quad \text{signum}$$

$$f(u) = H(u) = \begin{cases} 0 & u \leq 0 \\ 1 & u > 0 \end{cases} \quad \text{Heaviside}$$

$$f(u) = \begin{cases} -1 & u < -1 \\ u & -1 \leq u \leq 1 \\ 1 & u > 1 \end{cases} \quad \text{Saturated linear}$$

$$f(u) = u \quad \text{linear}$$

Functional units (neurons)

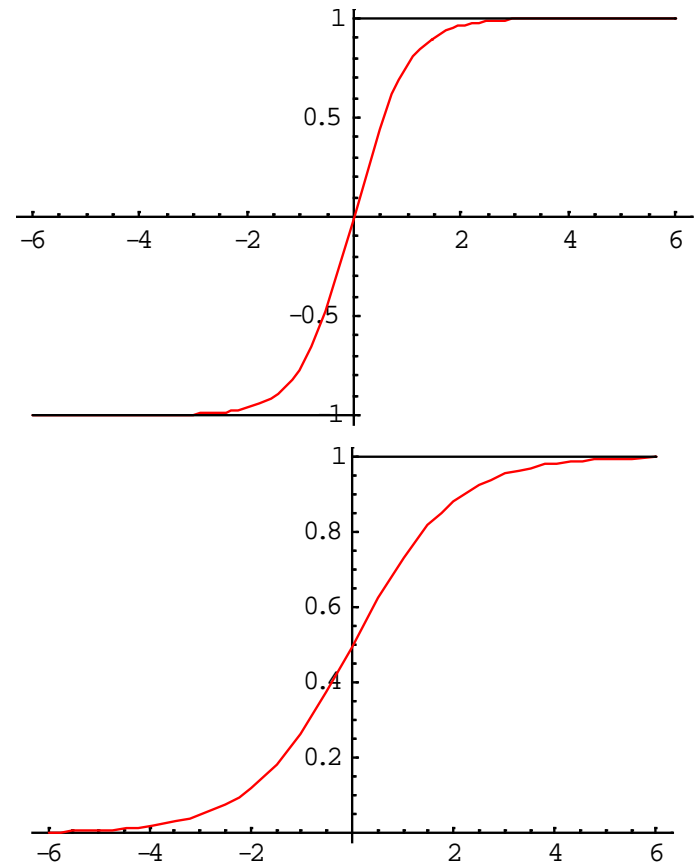
Sigmoidal aggregation functions

(Hyperbolic tangent)

$$f(u) = \tanh(u) = \frac{\exp(2u) - 1}{\exp(2u) + 1}$$

$$f(u) = \frac{1}{1 + \exp(-u)}$$

(Logistic)

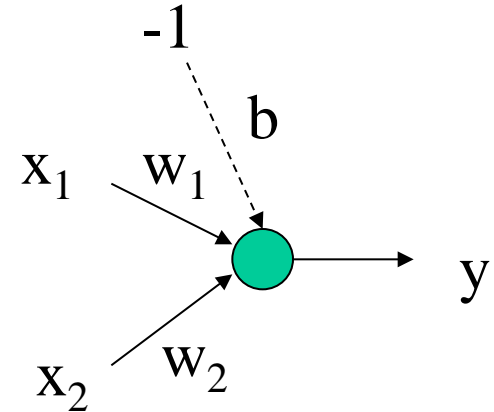
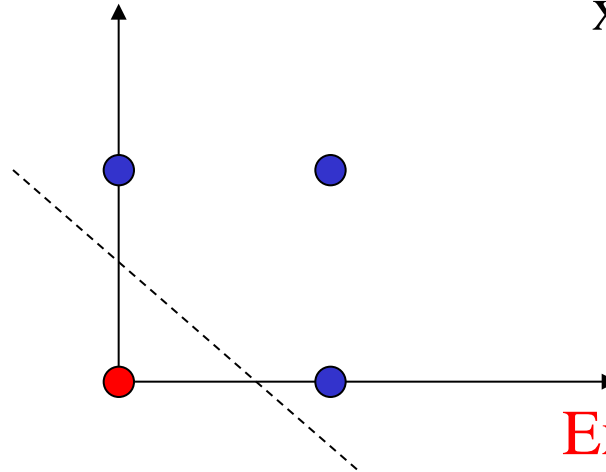


Functional units (neurons)

- What can do a single neuron ?
- It can solve simple problems (linearly separable problems)

	0	1
0	0	1
1	1	1

OR



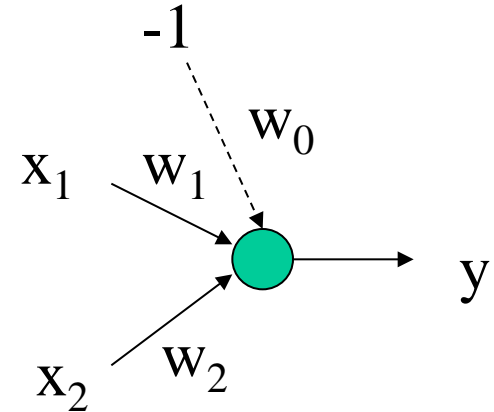
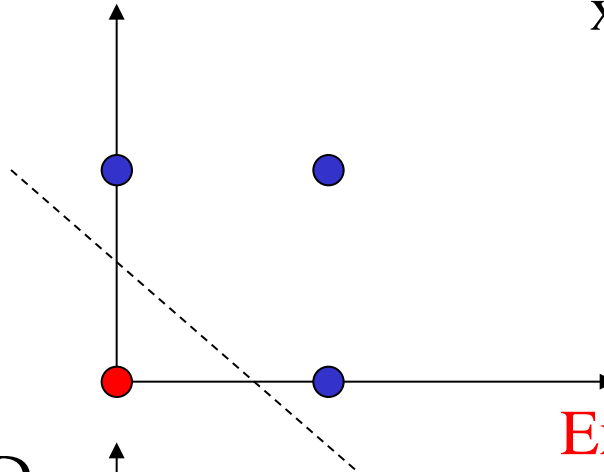
Ex: $y = H(w_1x_1 + w_2x_2 - b)$
 $w_1 = w_2 = 1, w_0 = 0.5$

Functional units (neurons)

- What can do a single neuron ?
- It can solve simple problems (linearly separable problems)

	0	1
0	0	1
1	1	1

OR

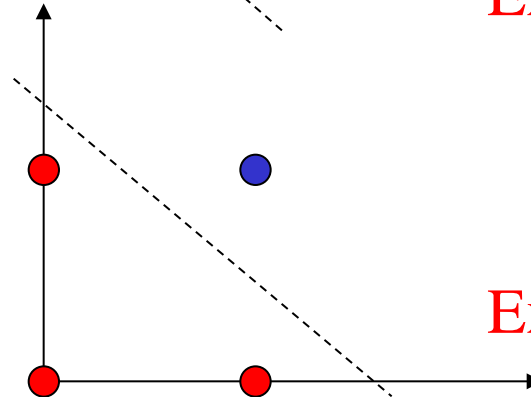


$$y = H(w_1x_1 + w_2x_2 - w_0)$$

Ex: $w_1 = w_2 = 1, w_0 = 0.5$

	0	1
0	0	0
1	0	1

AND

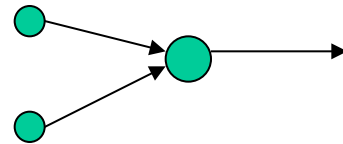
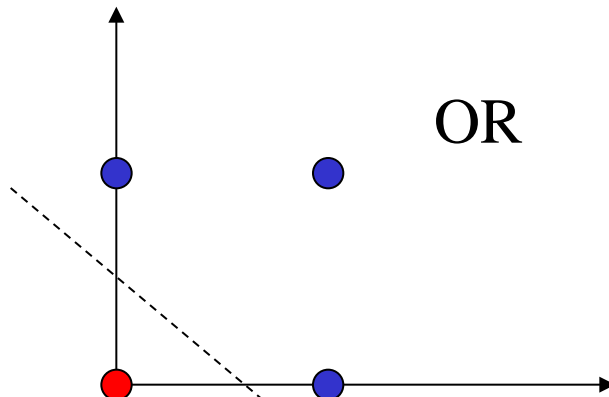


$$y = H(w_1x_1 + w_2x_2 - w_0)$$

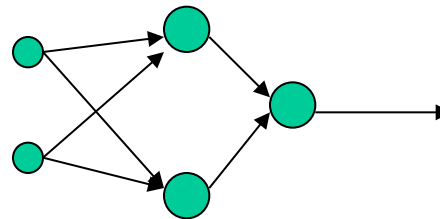
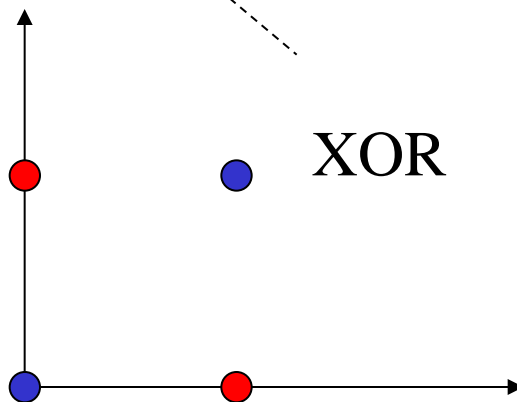
Ex: $w_1 = w_2 = 1, w_0 = 1.5$

Functional units (neurons)

Representation of boolean functions: $f:\{0,1\}^2 \rightarrow \{0,1\}$



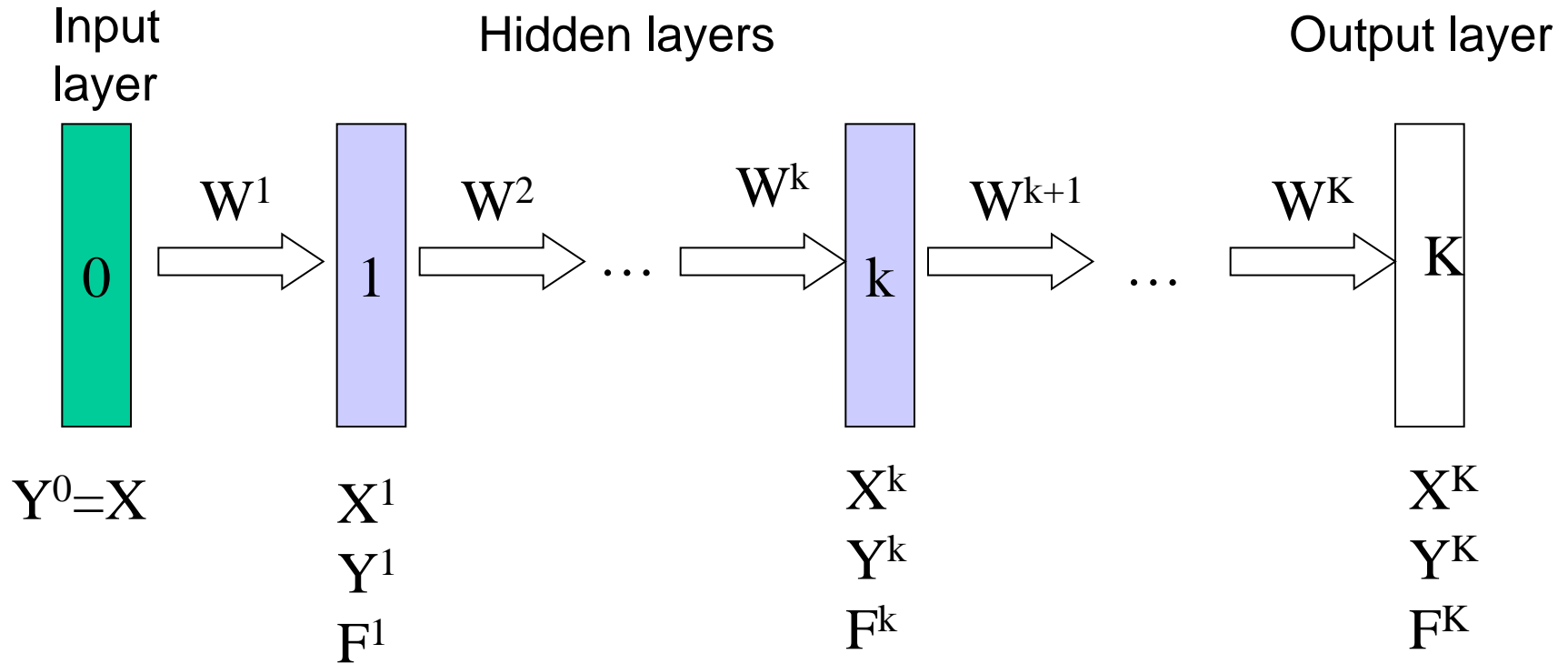
Linearly separable
problem: one layer
network



Nonlinearly separable
problem: multilayer
network

Architecture and notations

Feedforward network with K layers



X = input vector, Y = output vector, F = vectorial activation function

Functioning

Computation of the output vector

$$Y^K = F^K (W^K F^{K-1} (W^{K-1} \dots F^1 (W^1 X)))$$

$$Y^k = F^k (X^k) = F(W^k Y^{k-1})$$

FORWARD Algorithm (propagation of the input signal toward the output layer)

Y[0]:=X (X is the input signal)

FOR k:=1,K DO

 X[k]:=W[k]Y[k-1]

 Y[k]:=F(X[k])

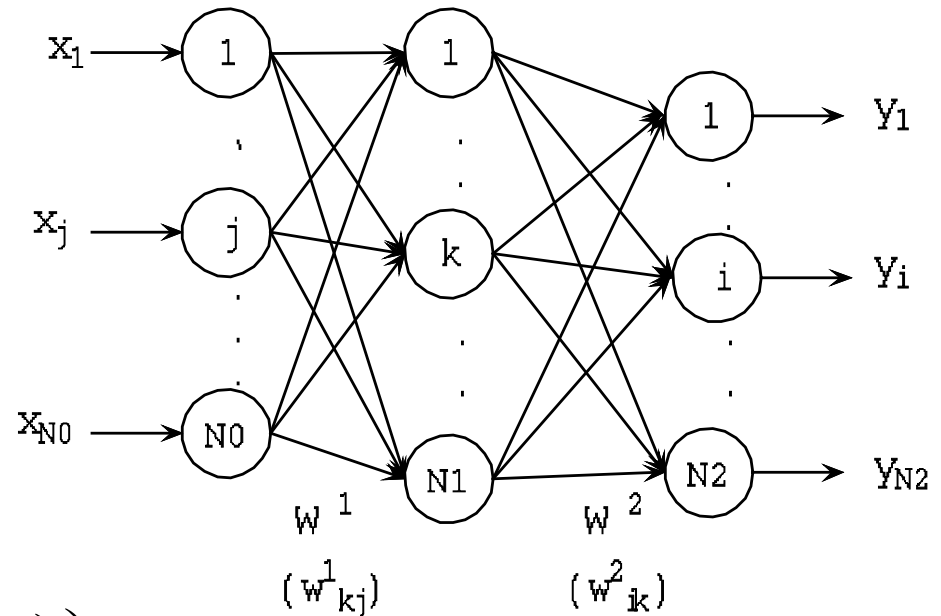
ENDFOR

Rmk: Y[K] is the output of the network

A particular case

One hidden layer

Adaptive parameters: $W1, W2$



$$y_i = f_2 \left(\sum_{k=0}^{N1} w^{(2)}_{ik} f_1 \left(\sum_{j=0}^{N0} w^{(1)}_{kj} x_j \right) \right)$$

A simpler notation : $w^{(2)}_{ik} = w_{ik}$; $w^{(1)}_{kj} = w_{kj}$

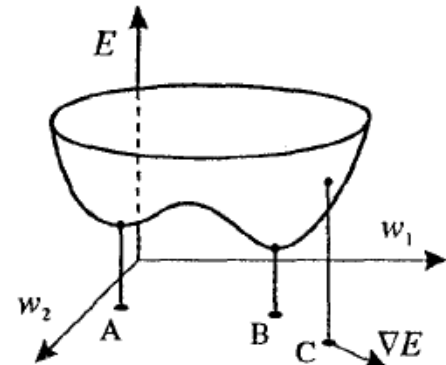
Learning process

Learning based on minimizing a error function

- Training set: $\{(x^1, d^1), \dots, (x^L, d^L)\}$
- Error function (mean squared error):

$$E(W) = \frac{1}{2L} \sum_{l=1}^L \sum_{i=1}^{N2} \left(d_i^l - f_2 \left(\sum_{k=0}^{N1} w_{ik} f_1 \left(\sum_{j=0}^{N0} w_{kj} x_j \right) \right) \right)^2$$

- Aim of learning process: find W which minimizes the error function
- Minimization method: gradient method



Learning process

Gradient based adjustment $w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E(w(t))}{\partial w_{ij}}$

Learning rate

$$E(W) = \frac{1}{2L} \sum_{l=1}^L \sum_{i=1}^{N2} \left(d_i^l - f_2 \left(\underbrace{\sum_{k=0}^{N1} w_{ik} f_1 \left(\underbrace{\sum_{j=0}^{N0} w_{kj} x_j}_{x_k} \right)}_{y_k} \right) \right)^2$$

x_i

y_i

$E_1(W)$

Learning process

- Partial derivatives computation

$$E(W) = \frac{1}{2L} \sum_{l=1}^L \sum_{i=1}^{N2} \left(d_i^l - f_2 \left(\underbrace{\sum_{k=0}^{N1} w_{ik} f_1 \left(\underbrace{\sum_{j=0}^{N0} w_{kj} x_j}_{x_k} \right)}_{y_k} \right) \right)^2$$

$\underbrace{\hspace{10em}}_{x_i}$
 $\underbrace{\hspace{10em}}_{y_i}$

$$\frac{\partial E_l(W)}{\partial w_{ik}} = -(d_i^l - y_i) f_2'(x_i) y_k = -\delta_i^l y_k$$

$$\frac{\partial E_l(W)}{\partial w_{kj}} = -\sum_{i=1}^{N2} w_{ik} (d_i^l - y_i) f_2'(x_i) f_1'(x_k) x_j = -\left(f_1'(x_k) \sum_{i=1}^{N2} w_{ik} \delta_i^l \right) x_j = -\delta_k^l x_j$$

$$E_l(W) = \frac{1}{2} \sum_{i=1}^{N2} \left(d_i^l - f_2 \left(\sum_{k=0}^{N1} w_{ik} f_1 \left(\sum_{j=0}^{N0} w_{kj} x_j \right) \right) \right)^2$$

Learning process

- Partial derivatives computation

$$\frac{\partial E_l(W)}{\partial w_{ik}} = -(d_i^l - y_i) f_2'(x_i) y_k = -\delta_i^l y_k$$

$$\frac{\partial E_l(W)}{\partial w_{kj}} = -\sum_{i=1}^{N2} w_{ik} (d_i^l - y_i) f_2'(x_i) f_1'(x_k) x_j = -\left(f_1'(x_k) \sum_{i=1}^{N2} w_{ik} \delta_i^l \right) x_j = -\delta_k^l x_j$$

$$E_l(W) = \frac{1}{2} \sum_{i=1}^{N2} \left(d_i^l - f_2 \left(\sum_{k=0}^{N1} w_{ik} f_1 \left(\sum_{j=0}^{N0} w_{kj} x_j \right) \right) \right)^2$$

Remark:

The derivatives of sigmoidal activation functions have particular properties:

Logistic: $f'(x) = f(x)(1-f(x)) = y(1-y)$

Tanh: $f'(x) = 1-f^2(x) = 1-y^2$

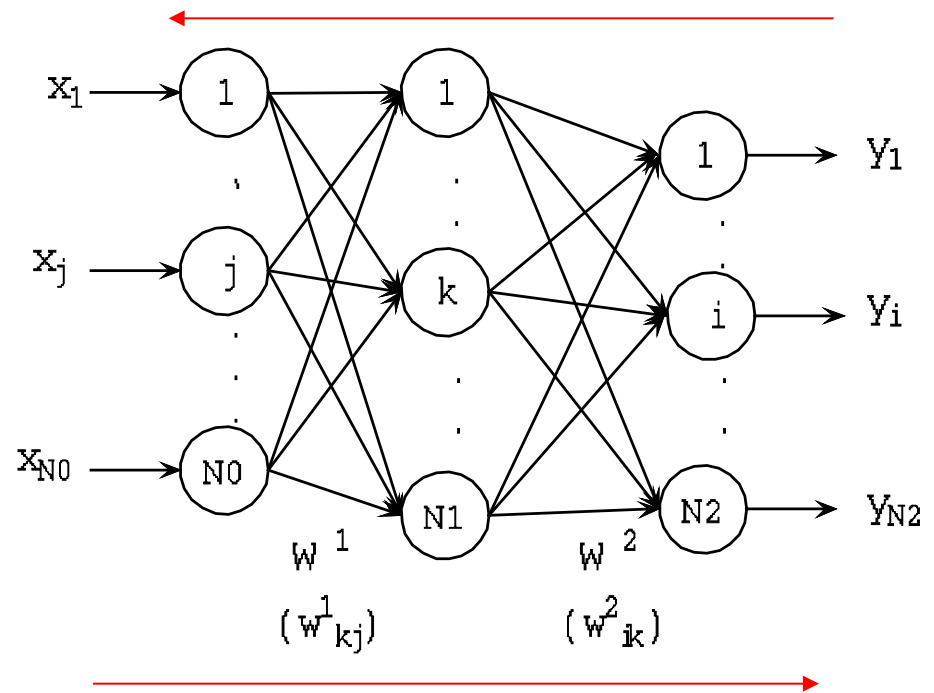
The BackPropagation Algorithm

Main idea:

For each example in the training set:

- compute the output signal
- compute the error corresponding to the output level
- propagate the error back into the network and store the corresponding delta values for each layer
- adjust each weight by using the error signal and input signal for each layer

Computation of the error signal (BACKWARD)



Computation of the output signal (FORWARD)

The BackPropagation Algorithm

General structure

Random initialization of weights

epoch {
REPEAT
 FOR I=1,L DO
 FORWARD stage
 BACKWARD stage
 weights adjustment
 ENDFOR
 Error (re)computation
UNTIL <stopping condition>

Rmk.

- The weights adjustment depends on the learning rate
- The error computation needs the recomputation of the output signal for the new values of the weights
- The stopping condition depends on the value of the error and on the number of epochs
- This is a so-called serial (incremental) variant: the adjustment is applied separately for each example from the training set

The BackPropagation Algorithm

Details (serial variant)

$$w_{kj} := \text{rand}(-1,1), w_{ik} := \text{rand}(-1,1)$$

$$p := 0$$

REPEAT

FOR $l := 1, L$ DO

/* FORWARD Step */

$$x_k^l := \sum_{j=0}^{N0} w_{kj} x_j^l, y_k^l := f_1(x_k^l), x_i^l := \sum_{k=0}^{N1} w_{ik} y_k^l, y_i^l := f_2(x_i^l)$$

/* BACKWARD Step */

$$\delta_i^l := f_2'(x_i^l)(d_i^l - y_i^l), \delta_k^l := f_1'(x_k^l) \sum_{i=1}^{N2} w_{ik} \delta_i^l$$

/* Adjustment Step */

$$w_{kj} := w_{kj} + \eta \delta_k^l x_j^l, w_{ik} := w_{ik} + \eta \delta_i^l y_k^l$$

ENDFOR

The BackPropagation Algorithm

Details (serial variant)

```
/* Error computation */
E := 0
FOR l := 1, L DO
  /* FORWARD Step */
  
$$x_k^l := \sum_{j=0}^{N0} w_{kj} x_j^l, y_k^l := f_1(x_k^l), x_i^l := \sum_{k=0}^{N1} w_{ik} y_k^l, y_i^l := f_2(x_i^l)$$

  /* Error summation */
  
$$E := E + \sum_{l=1}^L (d_i^l - y_i^l)^2$$

ENDFOR
E := E / (2L)
p := p + 1
UNTIL p > pmax OR E < E*
```

E^* denotes the expected training accuracy
 p_{\max} denotes the maximal number of epochs

The BackPropagation Algorithm

Batch variant

Random initialization of weights

REPEAT

initialize the variables which will contain the adjustments

FOR I=1,L DO

FORWARD stage

BACKWARD stage

cumulate the adjustments

ENDFOR

Apply the cumulated adjustments

Error (re)computation

UNTIL <stopping condition>

Rmk.

- The incremental variant can be sensitive to the presentation order of the training examples
- The batch variant is not sensitive to this order and is more robust to the errors in the training examples
- It is the starting algorithm for more elaborated variants, e.g. momentum variant

The BackPropagation Algorithm

Details (batch variant) $w_{kj} := rand(-1,1), w_{ik} := rand(-1,1), i = 1..N2, k = 0..N1, j = 0..N0$

$p := 0$

REPEAT

$$\Delta_{kj}^1 := 0, \Delta_{ik}^2 := 0$$

FOR $l := 1, L$ DO

/* FORWARD step */

$$x_k^l := \sum_{j=0}^{N0} w_{kj} x_j^l, y_k^l := f_1(x_k^l), x_i^l := \sum_{k=0}^{N1} w_{ik} y_k^l, y_i^l := f_2(x_i^l)$$

/* BACKWARD step */

$$\delta_i^l := f_2'(x_i^l)(d_i^l - y_i^l), \delta_k^l := f_1'(x_k^l) \sum_{i=1}^{N2} w_{ik} \delta_i^l$$

/* Adjustment step */

$$\Delta_{kj}^1 := \Delta_{kj}^1 + \eta \delta_k^l x_j^l, \Delta_{ik}^2 := \Delta_{ik}^2 + \eta \delta_i^l y_k^l$$

ENDFOR

$$w_{kj} := w_{kj} + \Delta_{kj}^1, w_{ik} := w_{ik} + \Delta_{ik}^2$$

The BackPropagation Algorithm

```
/* Error computation */  
 $E := 0$   
FOR  $l := 1, L$  DO  
  /* FORWARD Step */  
   $x_k^l := \sum_{j=0}^{N0} w_{kj} x_j^l, y_k^l := f_1(x_k^l), x_i^l := \sum_{k=0}^{N1} w_{ik} y_k^l, y_i^l := f_2(x_i^l)$   
  /* Error summation */  
   $E := E + \sum_{l=1}^L (d_i^l - y_i^l)^2$   
ENDFOR  
 $E := E / (2L)$   
 $p := p + 1$   
UNTIL  $p > p_{\max}$  OR  $E < E^*$ 
```

Variants

Different variants of BackPropagation can be designed by changing:

- Error function
- Minimization method
- Learning rate choice
- Weights initialization

Variants

Error function:

- ❑ MSE (mean squared error function) is appropriate in the case of approximation problems
- ❑ For classification problems a better error function is the cross-entropy error:
- ❑ Particular case: two classes (one output neuron):
 - d_l is from $\{0,1\}$ (0 corresponds to class 0 and 1 corresponds to class 1)
 - y_l is from $(0,1)$ and can be interpreted as the probability of class 1

$$CE(W) = - \sum_{l=1}^L (d_l \ln y_l + (1 - d_l) \ln(1 - y_l))$$

Rmk: the partial derivatives change, thus the adjustment terms will be different

Variants

Entropy based error:

- ❑ Different values of the partial derivatives
- ❑ In the case of logistic activation functions the error signal will be:

$$\begin{aligned}\delta_l &= \left(\frac{d_l}{y_l} - \frac{1-d_l}{1-y_l} \right) f_2'(x^{(2)}) = \frac{d_l(1-y_l) - y_l(1-d_l)}{y_l(1-y_l)} \cdot y_l(1-y_l) \\ &= d_l(1-y_l) - y_l(1-d_l)\end{aligned}$$

Variants

Minimization method:

- ❑ The gradient method is a simple but not very efficient method
- ❑ More sophisticated and faster methods can be used instead:
 - ❑ Conjugate gradient methods
 - ❑ Newton's method and its variants
- ❑ Particularities of these methods:
 - ❑ Faster convergence (e.g. the conjugate gradient converges in n steps for a quadratic error function)
 - ❑ Needs the computation of the hessian matrix (matrix with second order derivatives) : second order methods

Variants

Example: Newton's method

$E : R^n \rightarrow R$, $w \in R^n$ is the vector of all weights

By Taylor's expansion in $w(p)$ (estimation corresponding to epoch p)

$$E(w) \cong E(w(p)) + (\nabla E(w(p)))^T (w - w(p)) + \frac{1}{2} (w - w(p))^T H(w(p))(w - w(p))$$

$$H(w(p))_{ij} = \frac{\partial E(w(p))}{\partial w_i \partial w_j}$$

By derivating the Taylor's expansion with respect to w the minimum will be the solution of :

$$H(w(p))w - H(w(p))w(p) + \nabla E(w(p)) = 0$$

Thus the new estimation of w is :

$$w(p+1) = w(p) - H^{-1}(w(p)) \cdot \nabla E(w(p))$$

Variants

Particular case: Levenberg-Marquardt

- This is the Newton method adapted for the case when the objective function is a sum of squares (as MSE is)

$$E(w) = \sum_{l=1}^L E_l(w), \quad e(w) = (E_1(w), \dots, E_L(w))^T$$

$$w(p+1) = w(p) - (J^T(w(p)) \cdot J(w(p)) + \mu_p I)^{-1} J^T(w(p)) e(w(p))$$

$J(w)$ = jacobian of $e(w)$

$$J_{ij}(w) = \frac{\partial E_i(w)}{\partial w_j}$$



Used in order to deal with singular matrices

Advantage:

- Does not need the computation of the hessian

Problems in BackPropagation

- ❑ Low convergence rate (the error decreases too slow)
- ❑ Oscillations (the error value oscillates instead of continuously decreasing)
- ❑ Local minima problem (the learning process is stuck in a local minima of the error function)
- ❑ Stagnation (the learning process stagnates even if it is not a local minima)
- ❑ Overtraining and limited generalization

Problems in BackPropagation

Problem 1: The error decreases too slow or the error value oscillates instead of continuously decreasing

Causes:

- Inappropriate value of the learning rate (too small values lead to slow convergence while too large values lead to oscillations)
 - **Solution:** adaptive learning rate
- Slow minimization method (the gradient method needs small learning rates in order to converge)

Solutions:

- heuristic modification of the standard BP (e.g. momentum)
- other minimization methods (Newton, conjugate gradient)

Problems in BackPropagation

Adaptive learning rate:

- If the error is increasing then the learning rate should be decreased
- If the error significantly decreases then the learning rate can be increased
- In all other situations the learning rate is kept unchanged

$$E(p) > (1 + \gamma)E(p-1) \Rightarrow \eta(p) = a\eta(p-1), 0 < a < 1$$

$$E(p) < (1 - \gamma)E(p-1) \Rightarrow \eta(p) = b\eta(p-1), 1 < b < 2$$

$$(1 - \gamma)E(p-1) \leq E(p) \leq (1 + \gamma)E(p-1) \Rightarrow \eta(p) = \eta(p-1)$$

Example: $\gamma=0.05$

Problems in BackPropagation

Momentum variant:

- Increase the convergence speed by introducing some kind of “inertia” in the weights adjustment: the weight changes corresponding to the current epoch includes the adjustments from the previous epoch

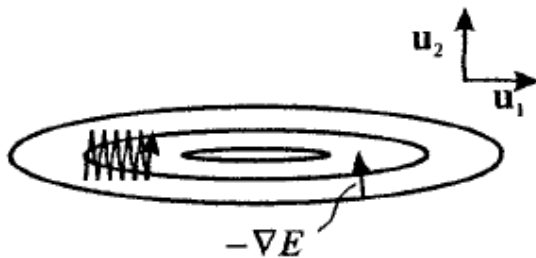
$$\Delta w_{ij}(p+1) = \eta(1-\alpha)\delta_i y_j + \alpha\Delta w_{ij}(p)$$

Momentum coefficient: α in $[0.1,0.9]$

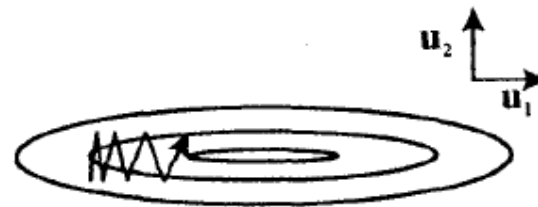
Problems in BackPropagation

Momentum variant:

- The effect of these enhancements is that flat spots of the error surface are traversed relatively rapidly with a few big steps, while the step size is decreased as the surface gets rougher. This implicit adaptation of the step size increases the learning speed significantly.



Simple gradient descent



Use of inertia term

Problems in BackPropagation

Problem 2: Local minima problem (the learning process is stuck in a local minima of the error function)

Cause: the gradient based methods are local optimization methods

Solutions:

- Restart the training process using other randomly initialized weights
- Introduce random perturbations into the values of weights:

$$w_{ij} := w_{ij} + \xi_{ij}, \quad \xi_{ij} = \text{random variables}$$

- Use a global optimization method

Problems in BackPropagation

Solution:

- Replacing the gradient method with a stochastic optimization method
- This means using a random perturbation instead of an adjustment based on the gradient computation
- Adjustment step:

Δ_{ij} = random values

IF $E(W + \Delta) < E(W)$ THEN accept the adjustment ($W := W + \Delta$)

Rmk:

- The adjustments are usually based on normally distributed random variables
- If the adjustment does not lead to a decrease of the error then it is not accepted

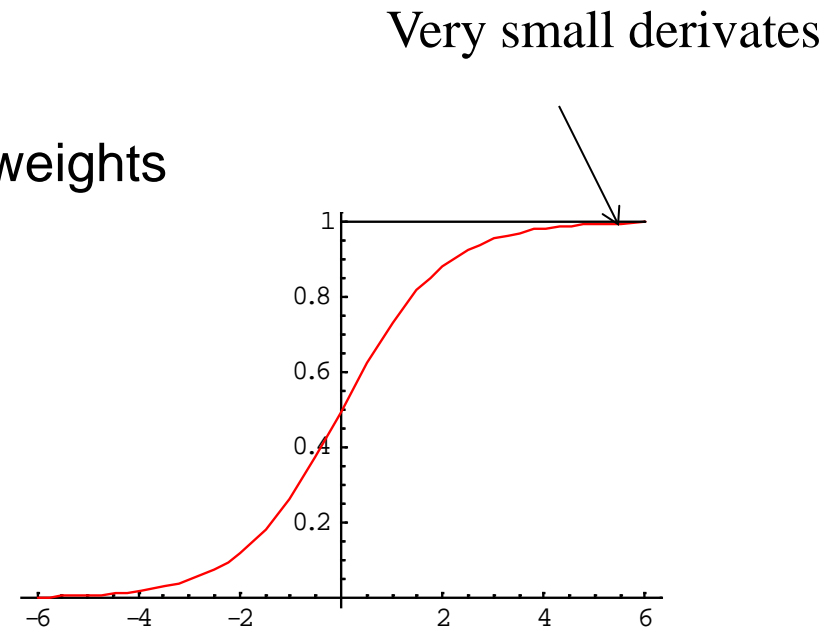
Problems in BackPropagation

Problem 3: Stagnation (the learning process stagnates even if it is not a local minima)

Cause: the adjustments are too small because the arguments of the sigmoidal functions are too large

Solutions:

- Penalize the large values of the weights (weights-decay)
- Use only the signs of derivatives values



Problems in BackPropagation

Penalization of large values of the weights: add a regularization term to the error function

$$E_{(r)}(W) = E(W) + \lambda \sum_{i,j} w_{ij}^2$$

The adjustment will be:

$$\Delta_{ij}^{(r)} = \Delta_{ij} - 2\lambda w_{ij}$$

Problems in BackPropagation

Resilient BackPropagation (use only the sign of the derivative not its value)

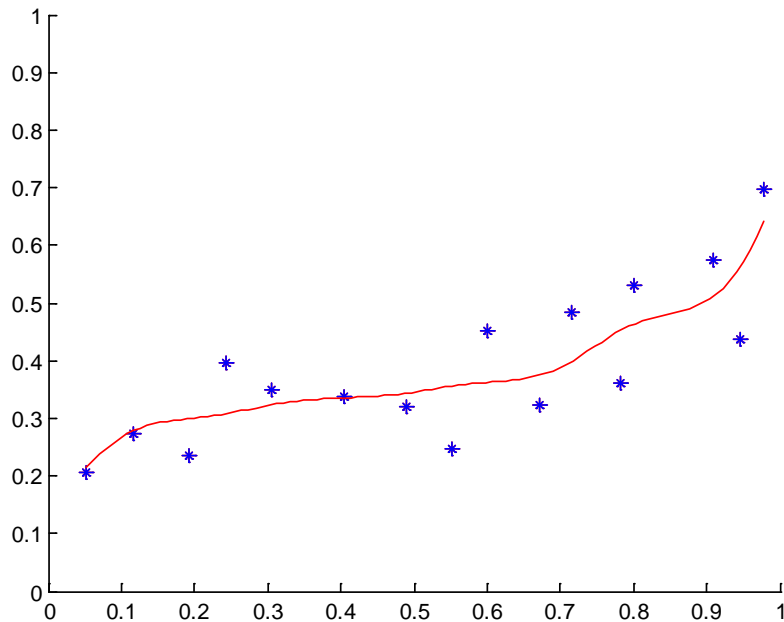
$$\Delta w_{ij}(p) = \begin{cases} -\Delta_{ij}(p) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} > 0 \\ \Delta_{ij}(p) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} < 0 \end{cases}$$

$$\Delta_{ij}(p) = \begin{cases} a\Delta_{ij}(p-1) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} \cdot \frac{\partial E(W(p-2))}{\partial w_{ij}} > 0 \\ b\Delta_{ij}(p-1) & \text{if } \frac{\partial E(W(p-1))}{\partial w_{ij}} \cdot \frac{\partial E(W(p-2))}{\partial w_{ij}} < 0 \end{cases}$$

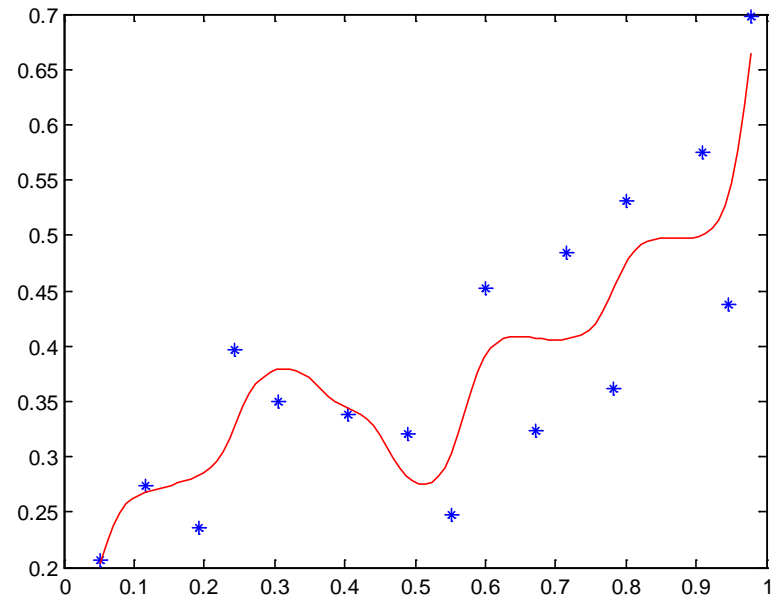
$$0 < b < 1 < a$$

Problems in BackPropagation

Problem 4: Overtraining and limited generalization ability



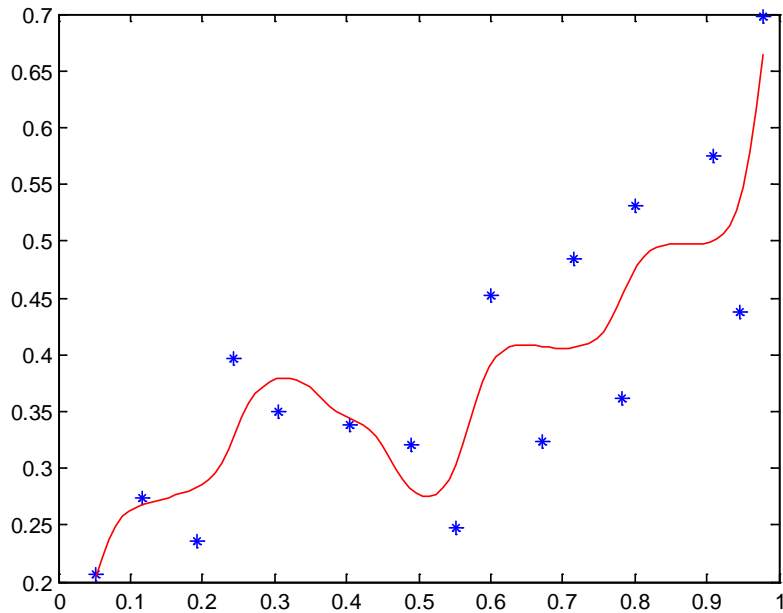
5 hidden units



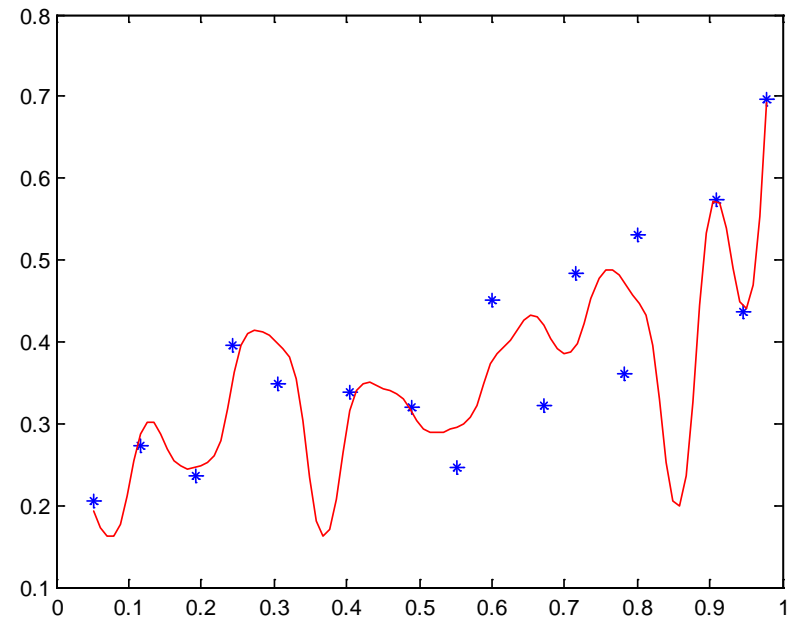
10 hidden units

Problems in BackPropagation

Problem 4: Overtraining and limited generalization ability



10 hidden units



20 hidden units

Problems in BackPropagation

Problem 4: Overtraining and limited generalization ability

Causes:

- **Network architecture** (e.g. number of hidden units)
 - A large number of hidden units can lead to overtraining (the network extracts not only the useful knowledge but also the noise in data)
- **The size of the training set**
 - Too few examples are not enough to train the network
- **The number of epochs** (accuracy on the training set)
 - Too many epochs could lead to overtraining

Solutions:

- Dynamic adaptation of the architecture
- Stopping criterion based on validation error; cross-validation

Problems in BackPropagation

Dynamic adaptation of the architectures:

- **Incremental strategy:**
 - Start with a small number of hidden neurons
 - If the learning does not progress new neurons are introduced
- **Decremental strategy:**
 - Start with a large number of hidden neurons
 - If there are neurons with small weights (small contribution to the output signal) they can be eliminated

Problems in BackPropagation

Stopping criterion based on validation error :

- Divide the learning set in m parts: $(m-1)$ are for training and another one for validation
- Repeat the weights adjustment as long as the error on the validation subset is decreasing (the learning is stopped when the error on the validation subset start increasing)

Cross-validation:

- Applies for m times the learning algorithm by successively changing the learning and validation steps

1: $S=(S_1, S_2, \dots, S_m)$

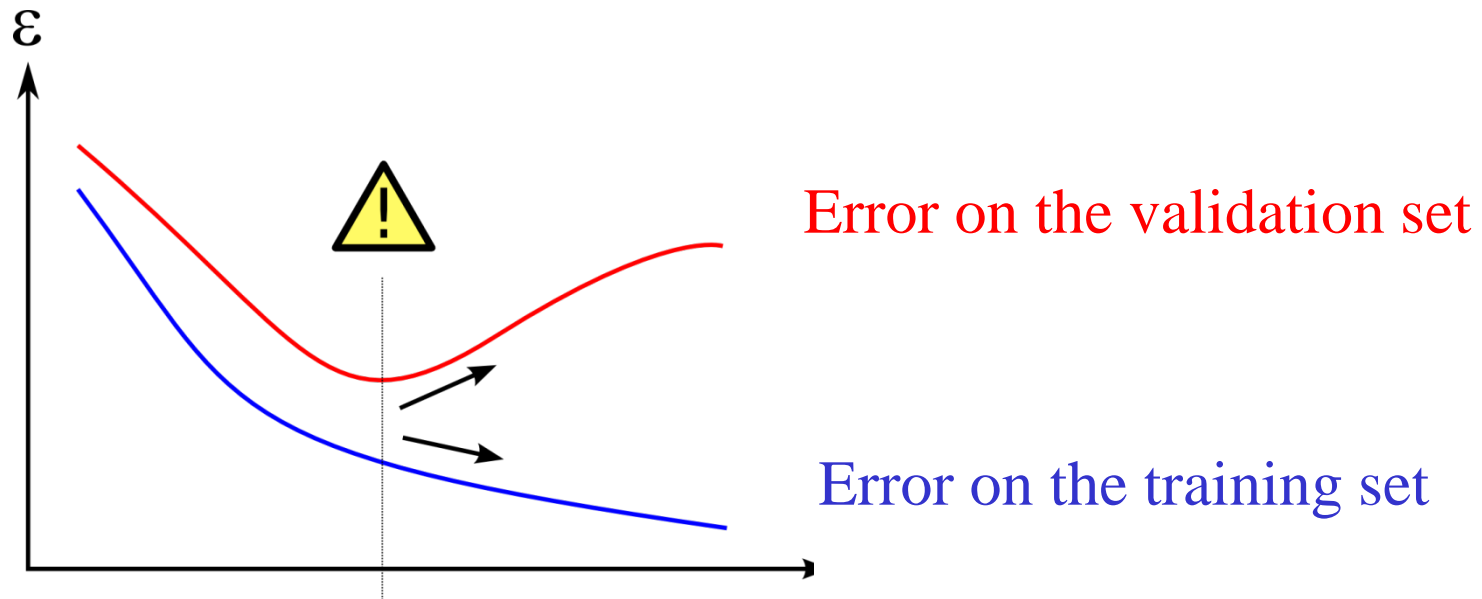
2: $S=(S_1, S_2, \dots, S_m)$

....

m : $S=(S_1, S_2, \dots, S_m)$

Problems in BackPropagation

Stop the learning process when the error on the validation set start to increase (even if the error on the training set is still decreasing) :



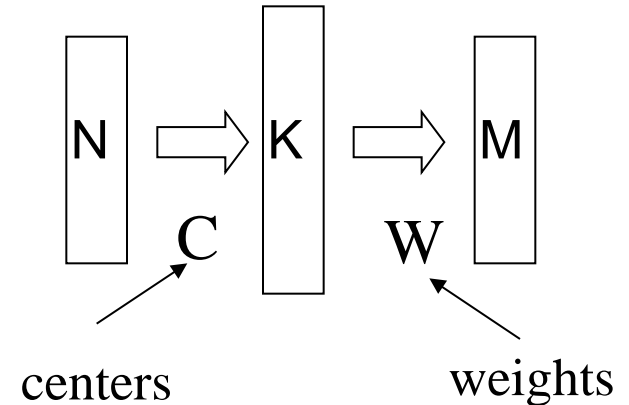
RBF networks

RBF - “Radial Basis Function”:

Architecture:

- Two levels of functional units
- Aggregation functions:

- Hidden units: distance between the input vector and the corresponding center vector
- Output units: weighted sum



$$G(X, C^k) = \|X - C^k\| = \sum_{i=1}^N (x_i - c_i^k)^2$$

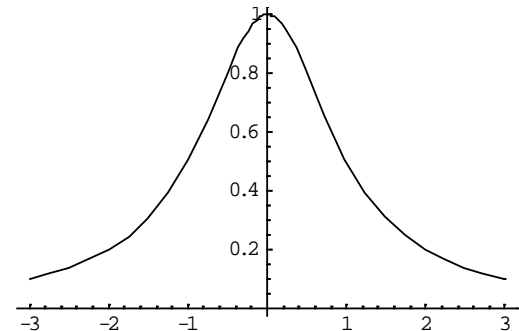
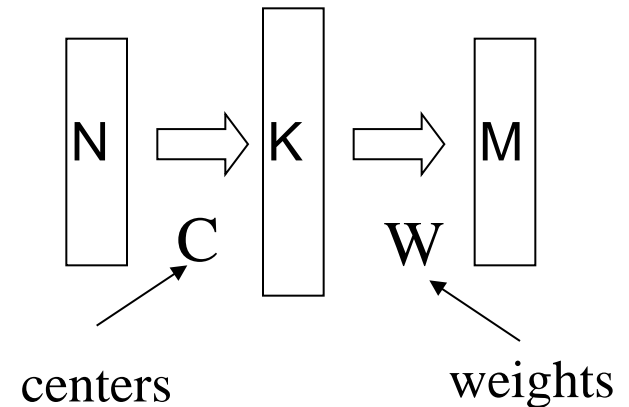
Rmk: hidden units do not have bias values (activation thresholds)

RBF networks

The **activation functions** for the hidden neurons are functions with radial symmetry

- Hidden units generates a significant output signal only for input vectors which are close enough to the corresponding center vector

The activation functions for the output units are usually linear functions



RBF networks

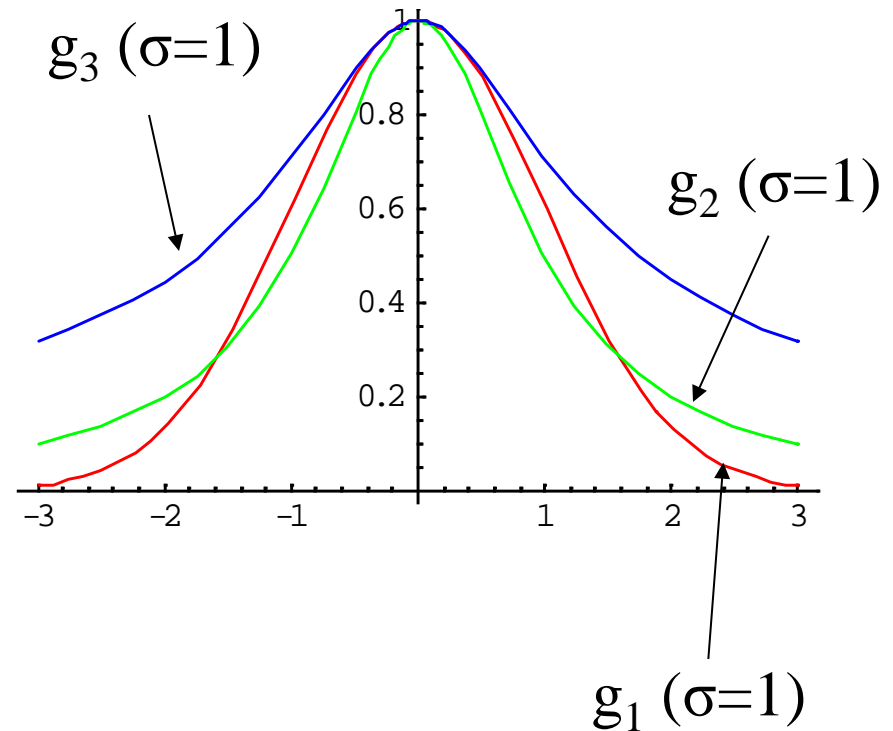
Examples of functions with radial symmetry:

$$g_1(u) = \exp(-u^2 / (2\sigma^2))$$

$$g_2(u) = 1 / (u^2 + \sigma^2)$$

$$g_3(u) = 1 / \sqrt{u^2 + \sigma^2}$$

Rmk: the parameter σ controls the width of the graph

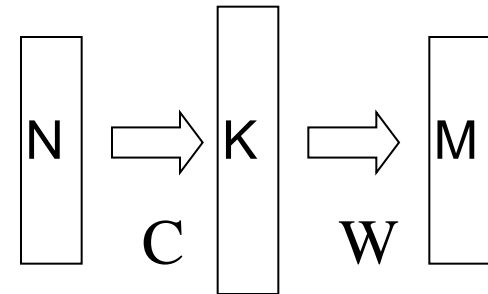


RBF networks

Computation of the output signal:

$$y_i = \sum_{k=1}^K w_{ik} g(\|X - C^k\|) - w_{i0}, \quad i = \overline{1, M}$$

$$y_i = \sum_{k=1}^K w_{ik} z_k - w_{i0}, \quad z_k = g(\|X - C^k\|)$$



Centers matrix

Weight matrix

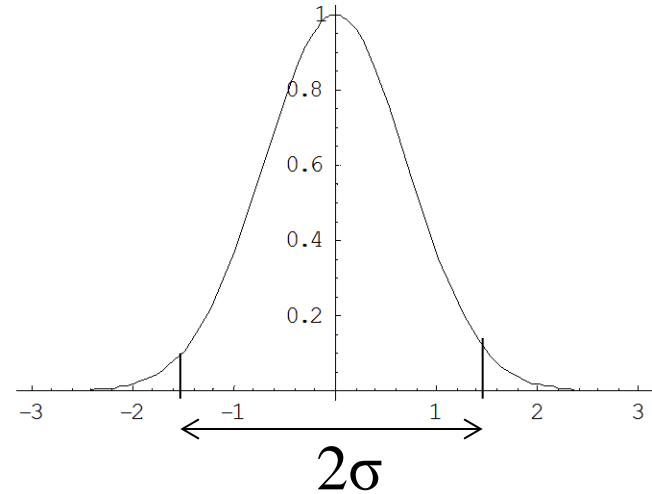
The vectors C^k can be interpreted as prototypes;

- only input vectors similar to the prototype of the hidden unit “activate” that unit
- the output of the network for a given input vector will be influenced only by the output of the hidden units having centers close enough to the input vector

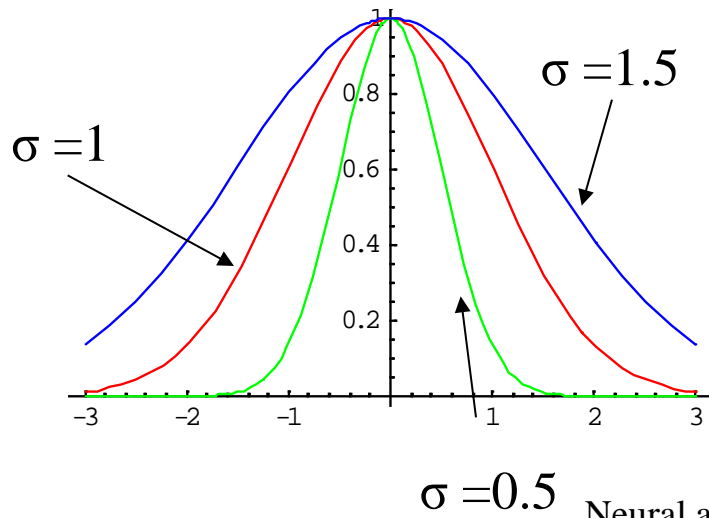
RBF networks

Each hidden unit is “sensitive” to a region in the input space corresponding to a neighborhood of its center. This region is called **receptive field**

The size of the receptive field depends on the parameter σ



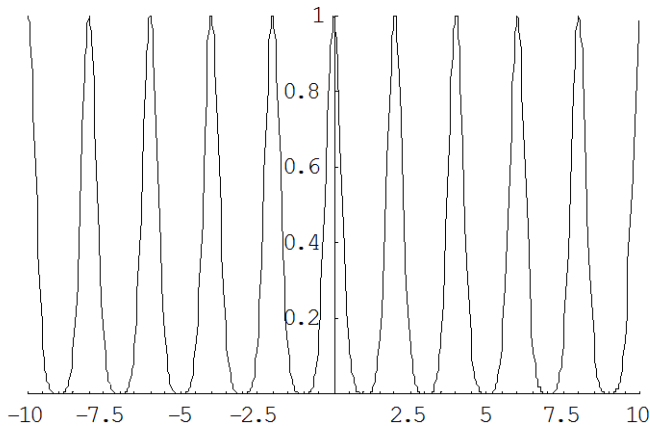
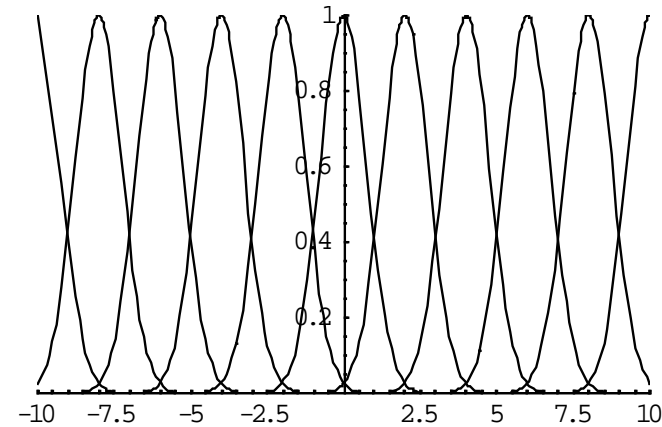
$$g(u) = \exp\left(-\frac{u^2}{2\sigma^2}\right)$$



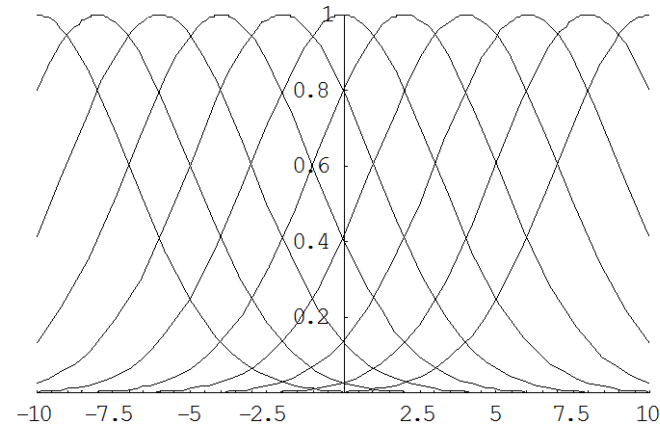
RBF networks

- The receptive fields of all hidden units covers the input space
- A good covering of the input space is essential for the approximation power of the network
- Too small or too large values of the width of the radial basis function lead to inappropriate covering of the input space

appropriate covering



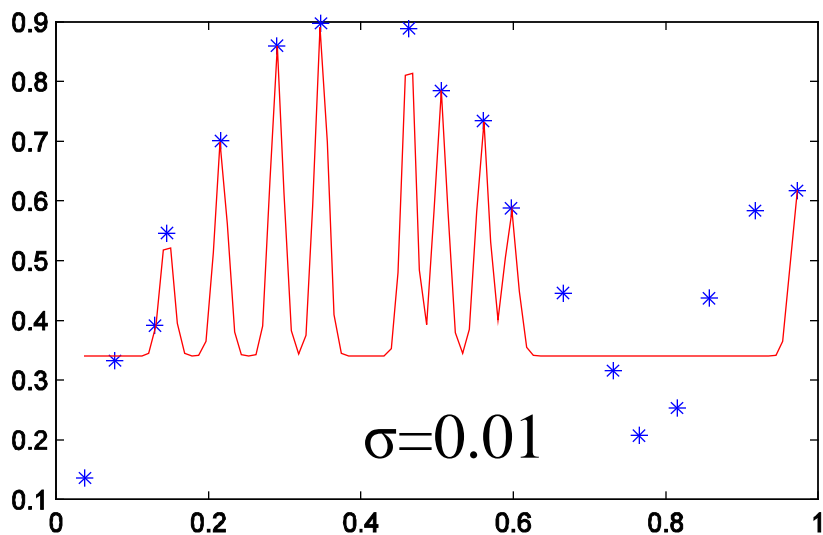
subcovering



overcovering

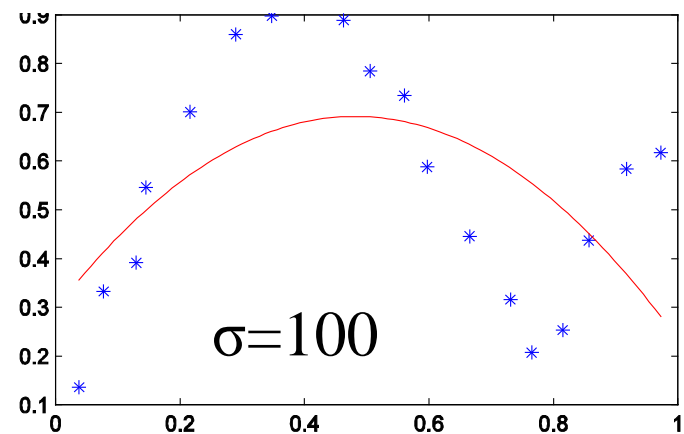
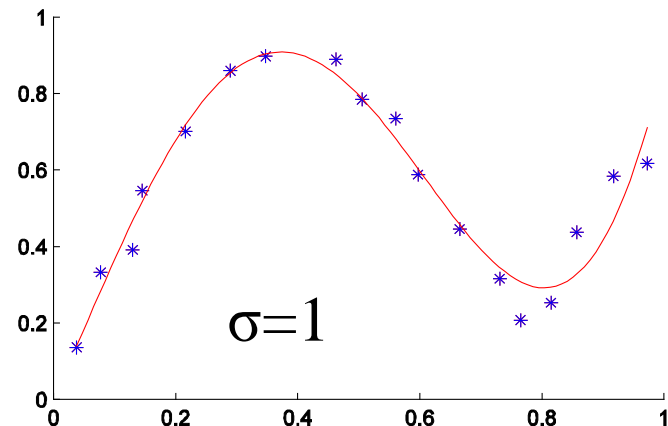
RBF networks

- The receptive fields of all hidden units covers the input space
- A good covering of the input space is essential for the approximation power of the network
- Too small or too large values of the width of the radial basis function lead to inappropriate covering of the input space



subcovering

appropriate covering



overcovering

RBF networks

RBF networks are universal approximators:

a network with N inputs and M outputs can approximate any function defined on \mathbb{R}^N , taking values in \mathbb{R}^M , as long as there are enough hidden units

The theoretical foundations of RBF networks are:

- Theory of approximation
- Theory of regularization

RBF networks

Adaptive parameters:

- Centers (prototypes) corresponding to hidden units
- Receptive field widths (parameters of the radial symmetry activation functions)
- Weights associated to connections between the hidden and output layers

Learning variants:

- **Simultaneous learning of all parameters** (similar to BackPropagation)
 - **Rmk:** same drawbacks as multilayer perceptron's BackPropagation
- **Separate learning of parameters:** centers, widths, weights

RBF networks

Separate learning :

Training set: $\{(x^1, d^1), \dots, (x^L, d^L)\}$

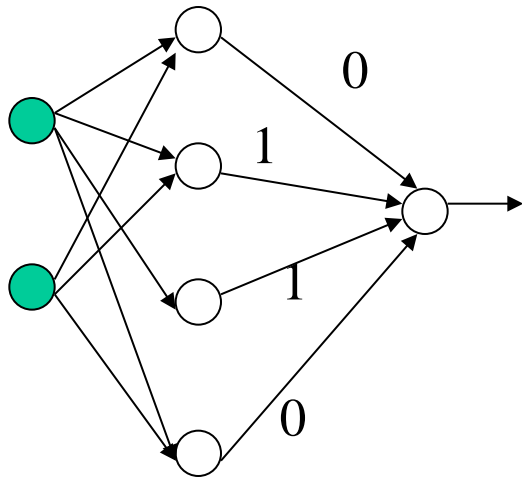
1. Estimating of the centers: simplest variant

- $K=L$ (nr of centers = nr of examples),
 - $C^k=x^k$ (this corresponds to the case of exact interpolation: see the example for XOR)

RBF networks

Example (particular case) : RBF network to represent XOR

- 2 input units
- 4 hidden units
- 1 output unit



Centers:

Hidden unit 1: (0,0)

Hidden unit 2: (1,0)

Hidden unit 3: (0,1)

Hidden unit 4: (1,1)

Weights:

w1: 0

w2: 1

w3: 1

w4: 0

Activation function:

$g(u)=1$ if $u=0$

$g(u)=0$ if $u \neq 0$

This approach cannot be applied for general approximation problems

RBF networks

Separate learning :

Training set: $\{(x^1, d^1), \dots, (x^L, d^L)\}$

1. Estimating of the centers

- $K < L$: the centers are established
 - by random selection from the training set
 - simple but not very effective
 - by systematic selection from the training set (Orthogonal Least Squares)
 - by using a clustering method

RBF networks

Orthogonal Least Squares:

- Incremental selection of centers such that the error on the training set is minimized
- The new center is chosen such that it is orthogonal on the space generated by the previously chosen centers (this process is based on the Gram-Schmidt orthogonalization method)
- This approach is related with regularization theory and ridge regression

RBF networks

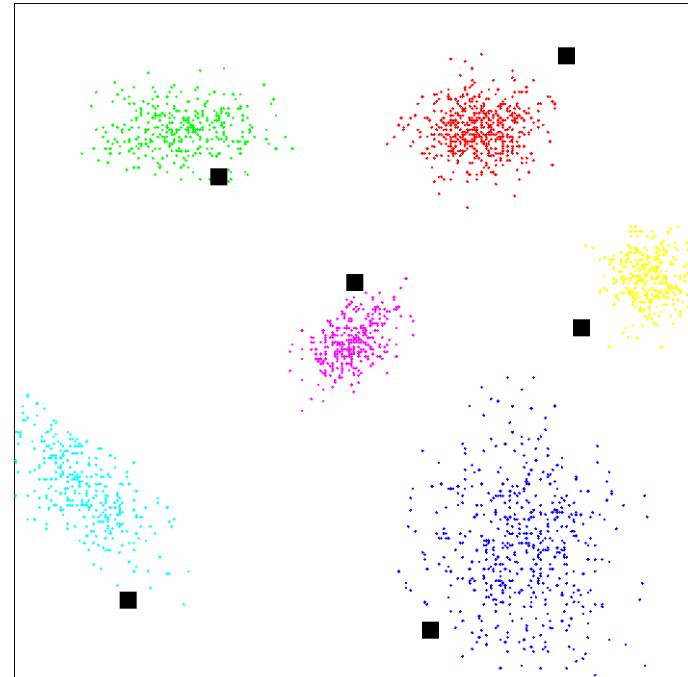
Clustering:

- Identify K groups in the input data $\{X^1, \dots, X^L\}$ such that data in a group are sufficiently similar and data in different groups are sufficiently dissimilar
- Each group has a representative (e.g. the mean of data in the group) which can be considered the center
- The algorithms for estimating the representatives of data belong to the class of partitional clustering methods
- Classical algorithm: K-means

RBF networks

K-means:

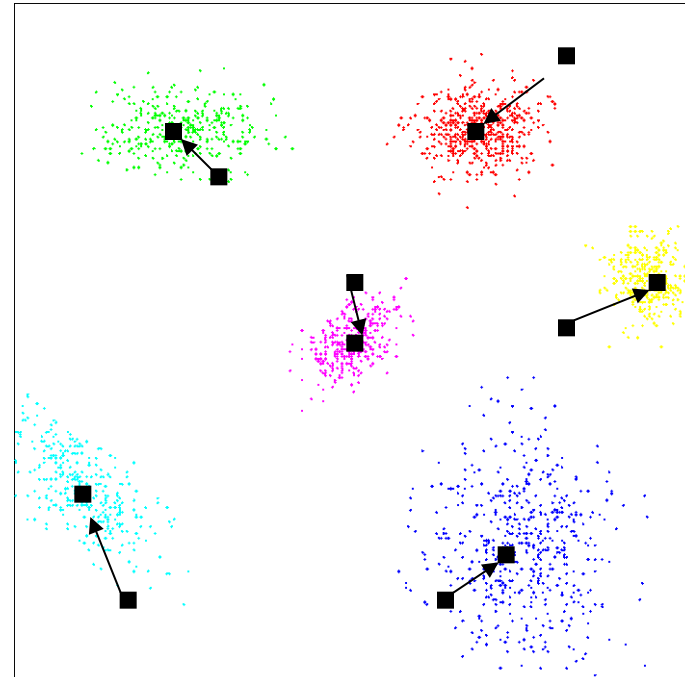
- Start with randomly initialized centers
- Iteratively:
 - Assign data to clusters based on the nearest center criterion
 - Recompute the centers as mean values of elements in each cluster



RBF networks

K-means:

- Start with randomly initialized centers
- Iteratively:
 - Assign data to clusters based on the nearest center criterion
 - Recompute the centers as mean values of elements in each cluster



RBF networks

K-means:

- $C^k := (\text{rand}(\text{min}, \text{max}), \dots, \text{rand}(\text{min}, \text{max}))$, $k=1..K$ or C^k is a randomly selected input data
- REPEAT
 - FOR $l:=1, L$
Find $k(l)$ such that $d(X^l, C^{k(l)}) \leq d(X^l, C^k)$
Assign X^l to class $k(l)$
 - Compute
 $C^k := \text{mean of elements which were assigned to class } k$
- UNTIL “no modification in the centers of the classes”

Remarks:

- usually the centers are not from the set of data
- the number of clusters should be known in advance

RBF networks

Incremental variant:

- Start with a small number of centers, randomly initialized
- Scan the set of input data:
 - If there is a center close enough to the data then this center is slightly adjusted in order to become even closer to the data
 - if the data is dissimilar enough with respect to all centers then a new center is added (the new center will be initialized with the data vector)

RBF networks

Incremental variant:

$K := K_0$

$C_i^k := \text{rand}(\text{min}, \text{max}), i = 1..N; k = 1..K$

$t := 0$

REPEAT

FOR $l := 1, L$ DO

find $k^* \in \{1, \dots, K\}$ such that $d(X^l, C^{k^*}) \leq d(X^l, C^k)$

IF $d(X^l, C^{k^*}) < \delta$ THEN $C^{k^*} := C^{k^*} + \eta \cdot (X^l - C^{k^*})$

ELSE $K := K + 1; C^K := X^l$

$t := t + 1$

$\eta := \eta_0 t^{-\alpha}$

UNTIL $t > t_{\max}$ OR $\eta < \varepsilon$

δ is a dissimilarity threshold

α controls the decrease of the learning rates

RBF networks

2. Estimating the receptive fields widths.

Heuristic rules:

$$\sigma = \frac{d_{\max}}{\sqrt{2K}}, \quad d_{\max} = \text{maximal distance between centers}$$

$$\sigma_k = \gamma d(C^k, C^j), \quad C^j = \text{the closest center to } C^k, \gamma \in [0.5, 1]$$

$$\sigma_k = \frac{1}{m} \sum_{j=1}^m d(C^k, C^j), \quad C^1, \dots, C^m : \text{the closest } m \text{ centers to } C^k$$

$$\sigma_k = \frac{1}{q_k} \sum_{j=1}^{q_k} d(C^k, X^j), \quad X^1, \dots, X^{q_k} : \text{input vectors represented by unit } k$$

RBF networks

3. Estimating the weights of connections between hidden and output layers:

- This is equivalent with the problem of training one layer linear network
- Variants:
 - Apply linear algebra tools (pseudo-inverse computation)
 - Apply Widrow-Hoff learning (training based on the gradient method applied to one layer neural networks)

□ Initialization:

$w_{ij}(0) := \text{rand}(-1, 1)$ (the weights are randomly initialized in $[-1, 1]$),
 $k := 0$ (iteration counter)

□ Iterative process

REPEAT

FOR $l := 1, L$ DO

Compute $y_i(l)$ and

$\delta_i(l) = d_i(l) - y_i(l)$, $i = 1, M$

Adjust the weights:

$w_{ij} := w_{ij} + \text{eta} * \delta_i(l) * x_j(l)$

Compute the $E(W)$ for the new values of the weights

$k := k + 1$

UNTIL $E(W) < E^*$ OR $k > k_{\text{max}}$

RBF vs. BP networks

RBF networks:

- 1 hidden layer
- Distance based aggregation function for the hidden units
- Activation functions with radial symmetry for hidden units
- Linear output units
- Separate training of adaptive parameters
- Similar with local approximation approaches

BP networks:

- many hidden layers
- Weighted sum as aggregation function for the hidden units
- Sigmoidal activation functions for hidden neurons
- Linear/nonlinear output units
- Simultaneous training of adaptive parameters
- Similar with global approximation approaches

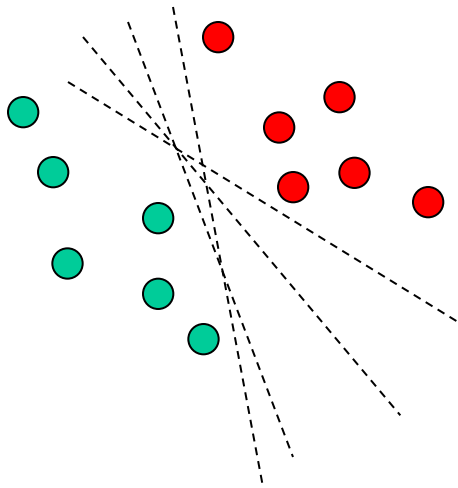
Support Vector Machines

Support Vector Machine (SVM) = machine learning technique characterized by

- The learning process is based on solving a quadratic optimization problem
- Ensures a good generalization power
- It relies on the statistical learning theory (main contributors: Vapnik and Chervonenkis)
- applications: handwritten recognition, speaker identification, object recognition
- Bibliografie: C.Burges – A Tutorial on SVM for Pattern Recognition, Data Mining and Knowledge Discovery, 2, 121–167 (1998)

Support Vector Machines

Let us consider a simple linearly separable classification problem



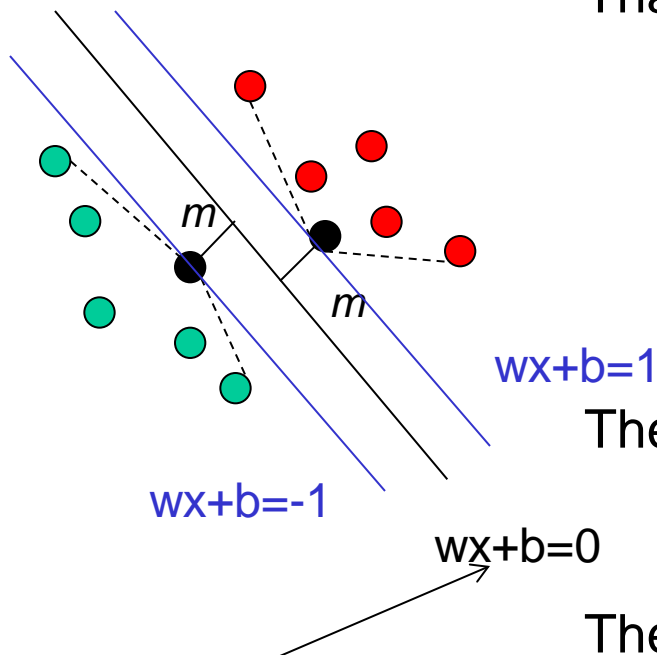
There is an infinity of lines (hyperplanes, in the general case) which ensure the separation in the two classes

Which separating hyperplane is the best?

That which leads to the best generalization ability = correct classification for data which do not belong to the training set

Support Vector Machines

Which is the best separating line (hyperplane) ?



That for which the minimal distance to the convex hulls corresponding to the two classes is maximal

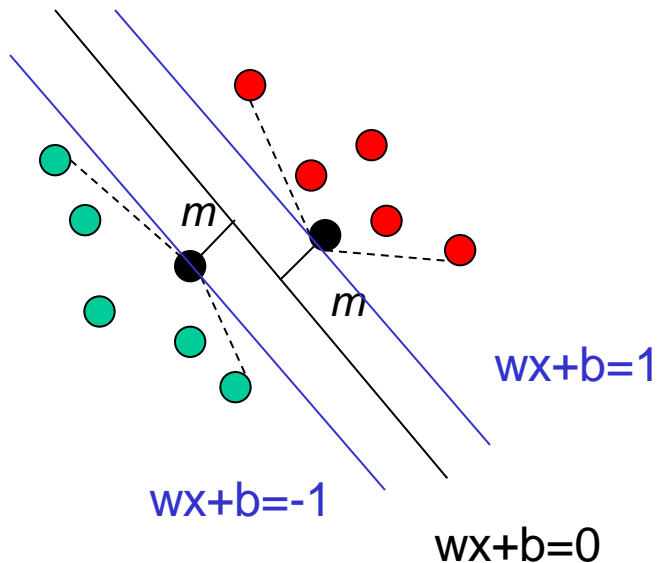
The lines (hyperplanes) going through the marginal points are called canonical lines (hyperplanes)

The distance between these lines is $2/\|w\|$,
Thus maximizing the width of the separating regions means minimizing the norm of w

Eq. of the separating hyperplane

Support Vector Machines

How can we find the separating hyperplane?



Find w and b which

minimize $\|w\|^2$

(maximize the separating region)

and satisfy

$(wx_i+b)y_i-1 \geq 0$

For all examples in the training set

$\{(x_1, y_1), (x_2, y_2), \dots, (x_L, y_L)\}$

$y_i = -1$ for the green class

$y_i = 1$ for the red class

(classify correctly all examples from the training set)

Support Vector Machines

The constrained minimization problem can be solved by using the Lagrange multipliers method:

Initial problem:

minimize $\|w\|^2$ such that $(wx_i + b)y_i - 1 \geq 0$ for all $i=1..L$

Introducing the Lagrange multipliers, the initial optimization problem is transformed in a problem of finding the saddle point of V :

$$V(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^L \alpha_i (y_i (w \cdot x_i + b) - 1), \quad \alpha_i \geq 0$$

(w^*, b^*, α^*) is saddle point if : $V(w^*, b^*, \alpha^*) = \max_{\alpha} \min_{w, b} V(w, b, \alpha)$

To solve this problem the dual function should be constructed:

$$W(\alpha) = \min_{w, b} V(w, b, \alpha)$$

$$\frac{\partial V(w, b, \alpha)}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^L \alpha_i y_i x_i \quad \frac{\partial V(w, b, \alpha)}{\partial b} = 0 \Rightarrow 0 = \sum_{i=1}^L \alpha_i y_i$$

Support Vector Machines

Thus we arrived to the problem of maximizing the dual function (with respect to α):

$$W(\alpha) = \sum_{i=1}^L \alpha_i - \frac{1}{2} \sum_{i,j=1}^L \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

such that the following constraints are satisfied:

$$\alpha_i \geq 0, \quad \sum_{i=1}^L \alpha_i y_i = 0$$

By solving the above problem (with respect to the multipliers α) the coefficients of the separating hyperplane can be computed as follows:

$$w^* = \sum_{i=1}^L \alpha_i y_i x_i, \quad b^* = 1 - w \cdot x_k$$

where k is the index of a non-zero multiplier and x_k is the corresponding training example (belonging to class +1)

Support Vector Machines

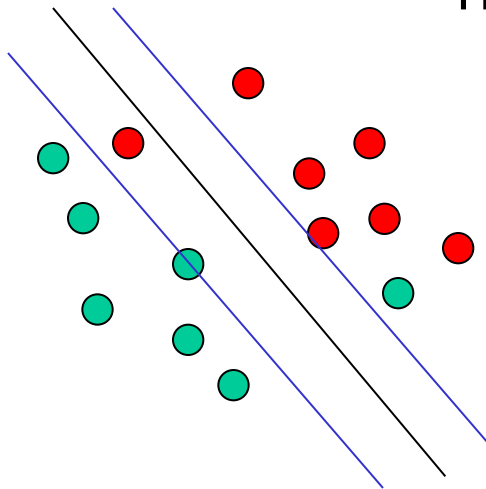
Remarks:

- The nonzero multipliers correspond to the examples for which the constraints are active ($w \cdot x + b = 1$ or $w \cdot x + b = -1$). These examples are called support vectors and they are the only examples which have an influence on the equation of the separating hyperplane
- the other examples from the training set (those corresponding to zero multipliers) can be modified without influencing the separating hyperplane)
- The decision function obtained by solving the quadratic optimization problem is:

$$D(z) = \text{sgn}\left(\sum_{i=1}^L \alpha_i y_i (x_i \cdot z) + b^*\right)$$

Support Vector Machines

What happens when the data are not very well separated?



The condition corresponding to each class is relaxed:

$$w \cdot x_i + b \geq 1 - \xi_i, \quad \text{daca } y_i = 1$$

$$w \cdot x_i + b \leq 1 + \xi_i, \quad \text{daca } y_i = -1$$

The function to be minimized becomes:

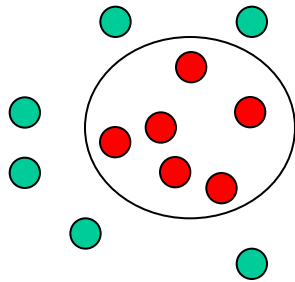
$$V(w, b, \alpha, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^L \xi_i - \sum_{i=1}^L \alpha_i (y_i (w \cdot x_i + b) - 1)$$

Thus the constraints in the dual problem are also changed:

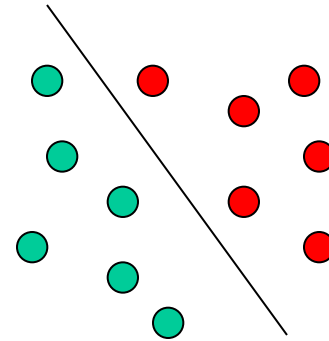
instead of $\alpha_i \geq 0$ it is used $0 \leq \alpha_i \leq C$

Support Vector Machines

What happens if the problem is nonlinearily separable?



$$x_1^2 + x_2^2 - R^2 = 0$$



$$w \cdot z + b = 0, \quad z_1 = x_1^2, z_2 = x_2^2$$

$$w_1 = w_2 = 1, \quad b = -R^2$$

$$x_1 \rightarrow \theta(x_1) = x_1^2$$

$$x_2 \rightarrow \theta(x_2) = x_2^2$$

Support Vector Machines

In the general case a transformation is applied:

$x \rightarrow \theta(x)$ and the scalar product of the transformed vectors becomes :

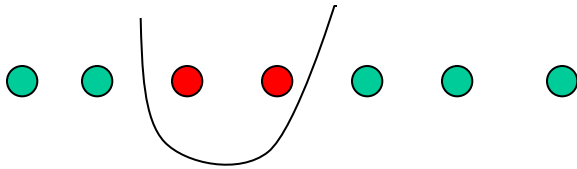
$$\theta(x) \cdot \theta(x') = K(x, x')$$

Since the optimization problem contains only scalar products it is not necessary to know explicitly the transformation θ but it is enough to know the **kernel function K**

Support Vector Machines

Example 1: Transforming a nonlinearly separable problem in a linearly separable one by going to a higher dimension

$$(x - \alpha)(x - \beta) = x^2 - (\alpha + \beta)x + \alpha\beta$$



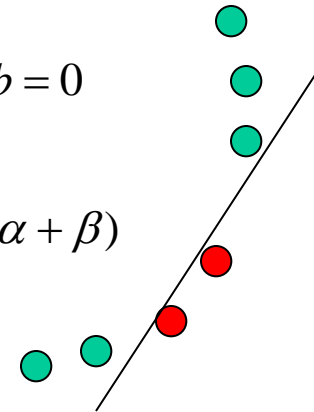
1-dimensional nonlinearly separable pb

$$w_1 z_1 + w_2 z_2 + b = 0$$

$$z_1 = x^2, z_2 = x$$

$$w_1 = 1, w_2 = -(\alpha + \beta)$$

$$b = \alpha\beta$$



2-dimensional linearly separable pb

Example 2: Constructing a kernel function when the decision surface corresponds to an arbitrary quadratic function (from dimension 2 the pb. is transferred in dimension 5).

$$\theta(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

$$K(x, x') = \theta(x_1, x_2) \cdot \theta(x'_1, x'_2) = (x \cdot x' + 1)^2$$

Support Vector Machines

Examples of kernel functions:

$$K(x, x') = (x \cdot x' + 1)^d$$

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

$$K(x, x') = \tanh(kx \cdot x' + b)$$

The decision function becomes:

$$D(z) = \text{sgn}\left(\sum_{i=1}^L \alpha_i y_i K(x_i, z) + b^*\right)$$

Support Vector Machines

Implementations

LibSVM [<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>]: (+ links to implementations in Java, Matlab, R, C#, Python, Ruby)

SVM-Light [http://www.cs.cornell.edu/People/tj/svm_light/]:
implementation in C

Spider [<http://www.kyb.tue.mpg.de/bs/people/spider/tutorial.html>]:
implementation in Matlab

SciLab interface for LibSVM
(<http://atoms.scilab.org/toolboxes/libsvm>)