بازشناسی الگو

درس ۹

طبقه‌بندی‌کننده‌های غیرخطی

Nonlinear Classifiers

کاظم فولادی قلعه

دانشکده مهندسی، پردیس فارابی

دانشگاه تهران

http://courses.fouladi.ir/pr

طبقه‌بندی‌کننده‌های غیرخطی

۱

مقدمه

In the previous chapter we dealt with the design of **linear classifiers** described by linear discriminant functions (hyperplanes) g(x).

In the simple two-class case,we saw that the perceptron algorithm computes the weights of the linear function g(x), provided that the classes are linearly separable .

For nonlinearly separable classes, linear classifiers were optimally designed, for example, by minimizing the squared error .

In this chapter we will deal with problems that are not linearly separable and for which the design of a linear classifier, even in an optimal way, does not lead to satisfactory performance. The design of nonlinear classifiers emerges now as an inescapable necessity.
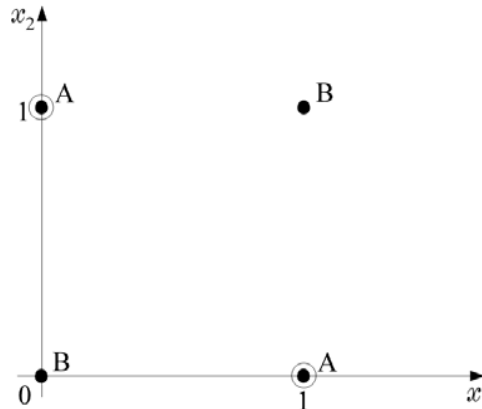
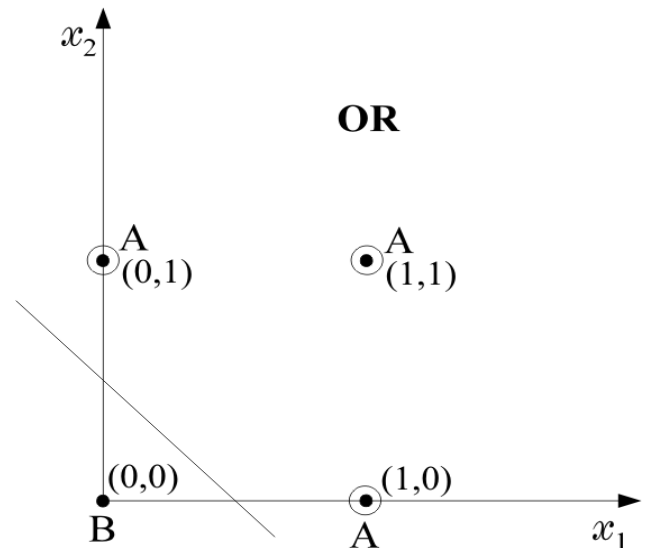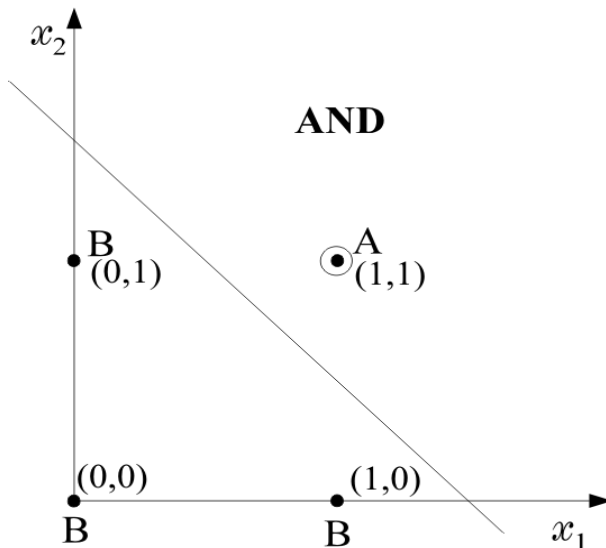طبقه‌بندی‌کننده‌های غیرخطی

۲

مسئله‌ی
XOR

NONLINEAR CLASSIFIERS

❖ The XOR problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

❖ There is no single line (hyperplane) that separates class A from class B. On the contrary, AND and OR operations are linearly separable problems
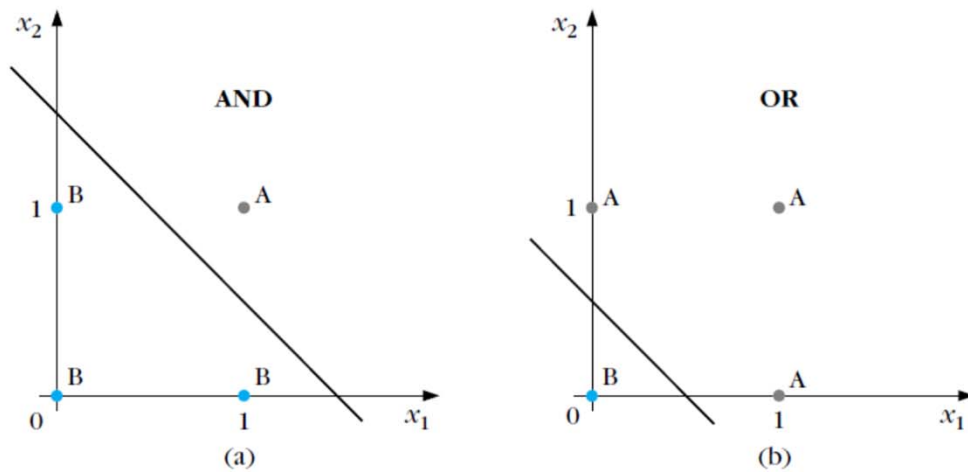
FIGURE 4.2

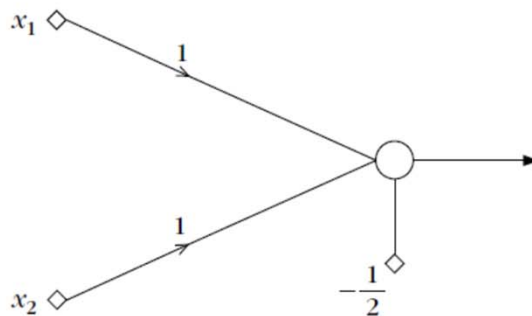Classes A and B for (a) the AND and (b) OR problems.

FIGURE 4.3
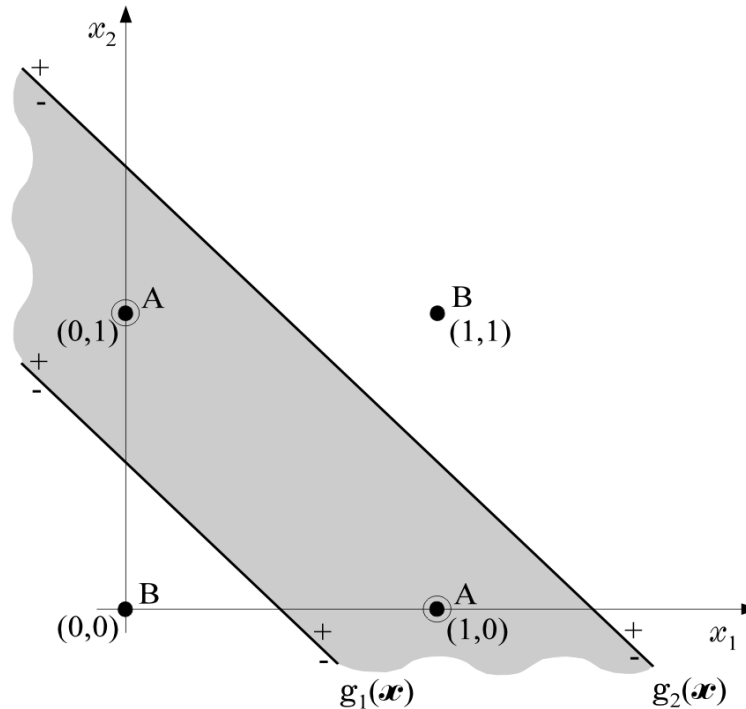
A perceptron realizing an OR gate.

طبقه‌بندی‌کننده‌های غیرخطی

۳

پرسپترون دو لایه

❖ The Two-Layer Perceptron

➢ For the XOR problem, draw two, instead, of one lines

9

➢ Then class B is located outside the shaded area and class A inside. This is a two-phase design.

- **Phase 1:** Draw two lines (hyperplanes)

$$g_1(\underline{x}) = g_2(\underline{x}) = 0$$

Each of them is realized by a <u>perceptron</u>.
The outputs of the perceptrons will be

$$y_i = f(g_i(\underline{x})) = \begin{cases} 0 \\ 1 \end{cases} \quad i = 1, 2$$

depending on the position of <u>x</u>.

- **Phase 2:** Find the position of <u>x</u> *w.r.t.* both lines, based on the values of y_1, y_2.

10

1st phase				2nd phase
x_1	x_2	y_1	y_2	
0	0	0(-)	0(-)	B(0)
0	1	1(+)	0(-)	A(1)
1	0	1(+)	0(-)	A(1)
1	1	1(+)	1(+)	B(0)

- Equivalently:

The computations of the first phase perform a mapping

$$\underline{x} \rightarrow \underline{y} = [y_1, y_2]^T$$

11

The decision is now performed on the transformed $\underline{y}$ data.
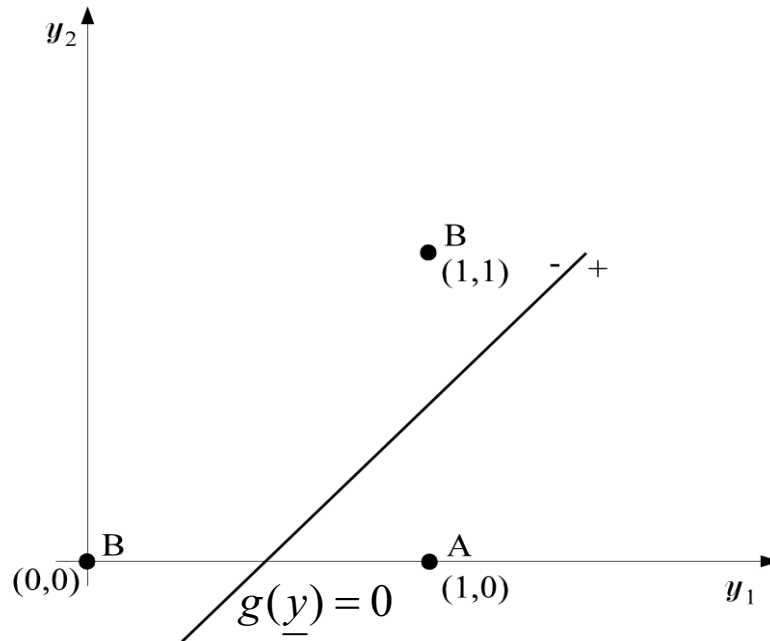
This can be performed via a second line,
which can also be realized by a <u>perceptron</u>.

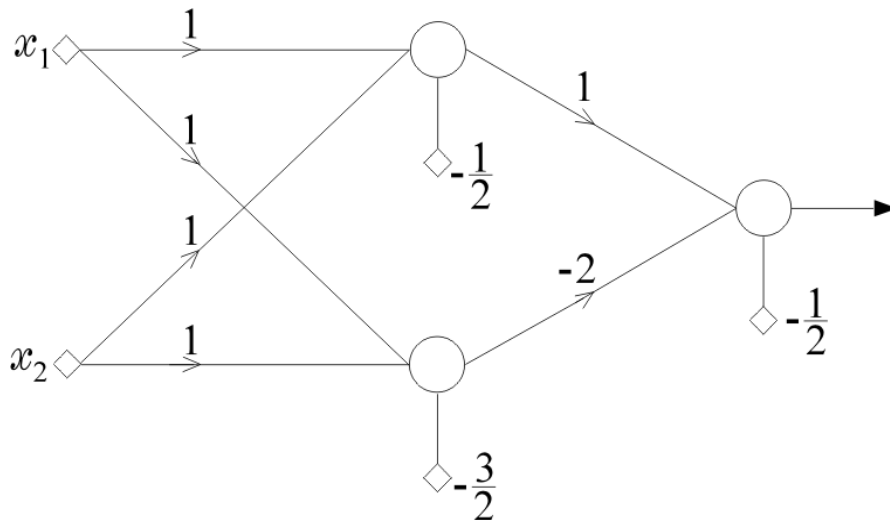➢ Computations of the first phase perform a **mapping** that transforms the nonlinearly separable problem to a linearly separable one.

➢ The architecture

- This is known as the two layer perceptron with one hidden and one output layer. The activation functions are

$$f(.) = \begin{cases} 0 \\ 1 \end{cases}$$

- The neurons (nodes) of the figure realize the following lines (hyperplanes)

$$g_1(\underline{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\underline{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\underline{y}) = y_1 - 2y_2 - \frac{1}{2} = 0$$

❖ Classification capabilities of the two-layer perceptron

➢ The mapping performed by the first layer neurons is onto the vertices of the unit side square, e.g., (0, 0), (0, 1), (1, 0), (1, 1).

➢ The more general case,

$$\underline{x} \in R^l$$

$$\underline{x} \to \underline{y} = [y_1, ..., y_p]^T, \ y_i \in \{0, \ 1\} \ \ i = 1, \ 2, ..., p$$

15

$$\underline{x} \in R^l$$
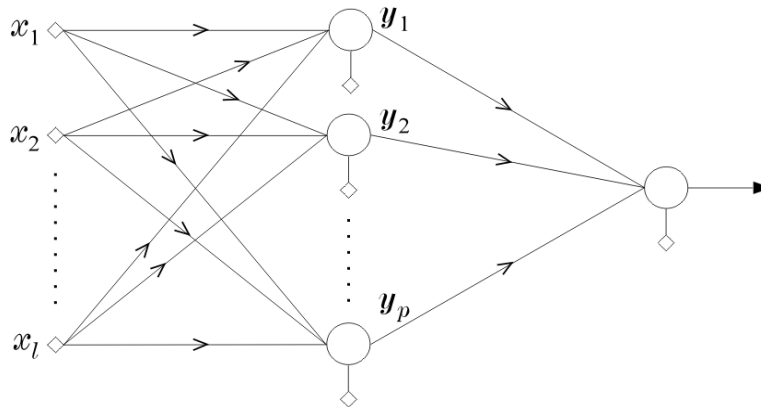
$$\underline{x} \rightarrow \underline{y} = [y_1, ..., y_p]^T , \; y_i \in \{0, \; 1\} \;\; i = 1, \; 2, ..., p$$

performs a mapping of a vector

onto the vertices of the unit side H_p hypercube

➢ The mapping is achieved with p neurons each realizing a hyperplane.

➢ The output of each of these neurons is 0 or 1 depending on the relative position of $\underline{x}$ w.r.t. the hyperplane.

➢ Intersections of these hyperplanes form regions in the
l-dimensional space. Each region corresponds to a vertex
of the H_p unit hypercube.
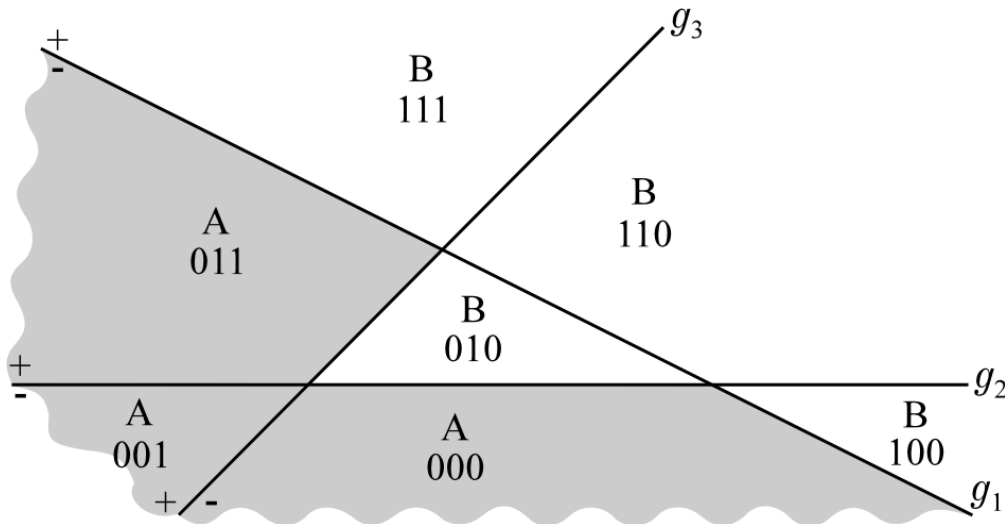
For example,
the 001 vertex corresponds to the region which is located

to the (-) side of $g_1(\underline{x}) = 0$

to the (-) side of $g_2(\underline{x}) = 0$

to the (+) side of $g_3(\underline{x}) = 0$

18
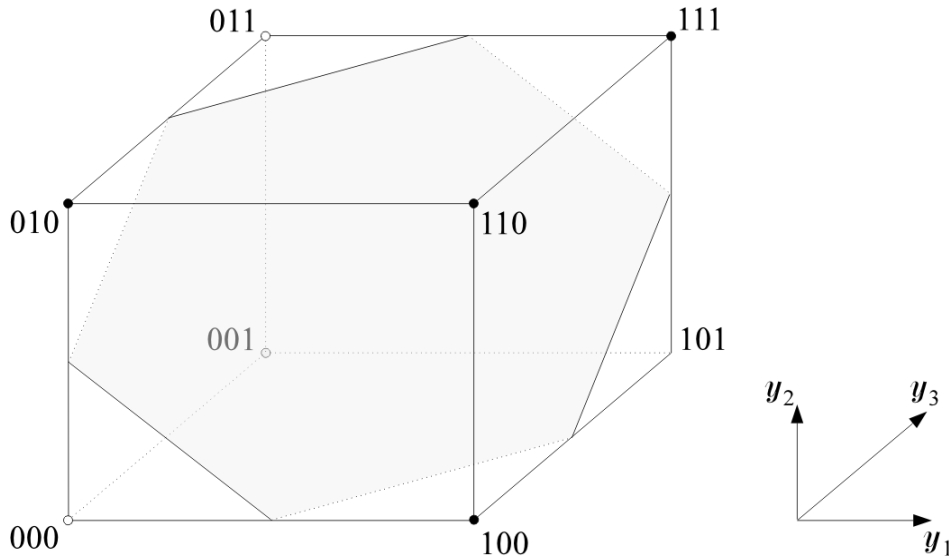
➢ The output neuron realizes a hyperplane in the transformed $\underline{y}$ space, that separates some of the vertices from the others.

➢ Thus, the two layer perceptron has the capability to classify vectors into **classes that consist of unions of polyhedral regions**: But **NOT ANY** union. It depends on the relative position of the corresponding vertices.
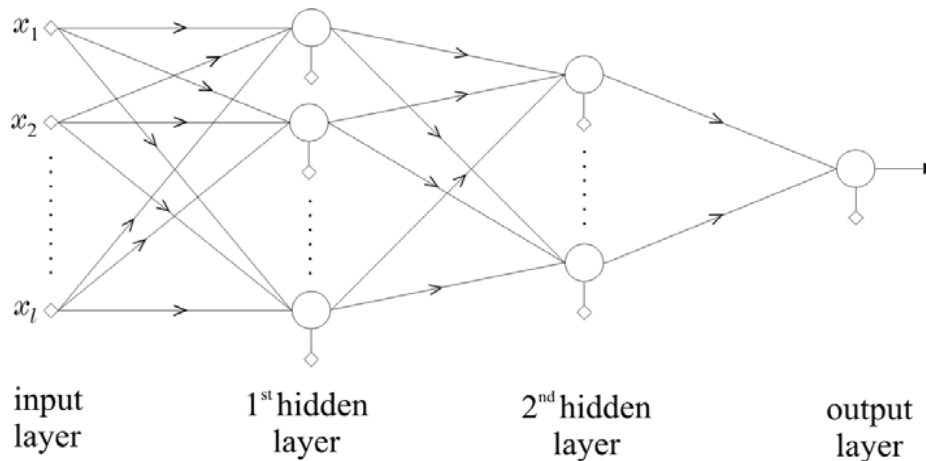
طبقه‌بندی‌کننده‌های غیرخطی

۴

پرسپترون سه لایه

❖ Three-layer perceptrons

➢ The architecture

➢ This is capable to classify vectors into classes consisting of **ANY** union of polyhedral regions.

➢ The idea is similar to the XOR problem. It realizes more than one planes in the $\underline{y} \in R^p$ space.

21

➢ The reasoning

- For each vertex, corresponding to class, say A, construct a hyperplane which leaves THIS vertex on one side (+) and ALL the others to the other side (-).
- The output neuron realizes an OR gate

➢ Overall:

> The first layer of the network forms the hyperplanes,
> The second layer forms the regions, and
> The output neuron forms the classes.

❖ Designing Multilayer Perceptrons
  ➢ One direction is to adopt the above rationale and develop a structure that classifies **correctly all** the training patterns.
  ➢ The other direction is to choose a structure and compute the synaptic weights to optimize a cost function.

طبقه‌بندی‌کننده‌های غیرخطی

۵

الگوریتم پس‌انتشار

❖ The Backpropagation Algorithm

➢ This is an algorithmic procedure that computes the synaptic weights iteratively, so that an adopted cost function is minimized (optimized)

➢ In a large number of optimizing procedures, computation of derivatives are involved. Hence, discontinuous activation functions pose a problem, i.e.,

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

➢ There is always an escape path!!! The logistic function

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

is an example. Other functions are also possible and in some cases more desirable.

24

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

$a_3 > a_2 > a_1$
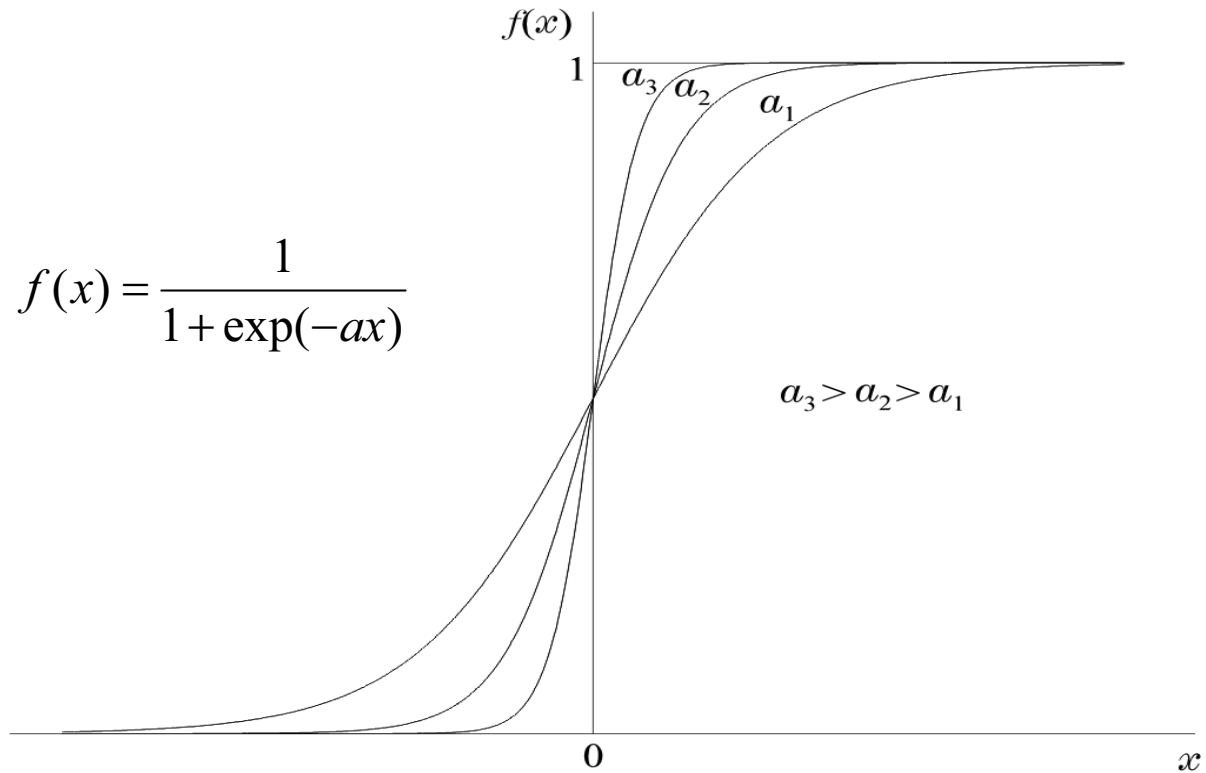
➢ The steps:

- Adopt an optimizing cost function, e.g.,
  - – Least Squares Error
  - – Relative Entropy

  between desired responses and actual responses of the network for the available training patterns. That is, from now on we have to live with errors. We only try to minimize them, using certain criteria.

- Adopt an algorithmic procedure for the optimization of the cost function with respect to the synaptic weights, e.g.,
  - – Gradient descent
  - – Newton's algorithm
  - – Conjugate gradient

26

- The task is a **nonlinear** optimization one. For the gradient descent method

$$\underline{w}_1^r(\text{new}) = \underline{w}_1^r(\text{old}) + \Delta\underline{w}_1^r$$

$$\Delta\underline{w}_1^r = -\mu\frac{\partial J}{\partial w_1^r}$$

➤ The Procedure:

- Initialize unknown weights randomly with small values.
- Compute the gradient terms backwards, starting with the weights of the last (3rd) layer and then moving towards the first
- Update the weights
- Repeat the procedure until a termination procedure is met

➤ Two major philosophies:

- Batch mode: The gradients of the last layer are computed once ALL training data have appeared to the algorithm, i.e., by summing up all error terms.
- Pattern mode: The gradients are computed every time a new training data pair appears. Thus gradients are based on successive individual errors.

28

Batch

Computational flow

1st 2nd 3rd

The Algorithm

Pattern

Input

$\left(\begin{smallmatrix} x \\ . \end{smallmatrix}\right)$

Desired output

29
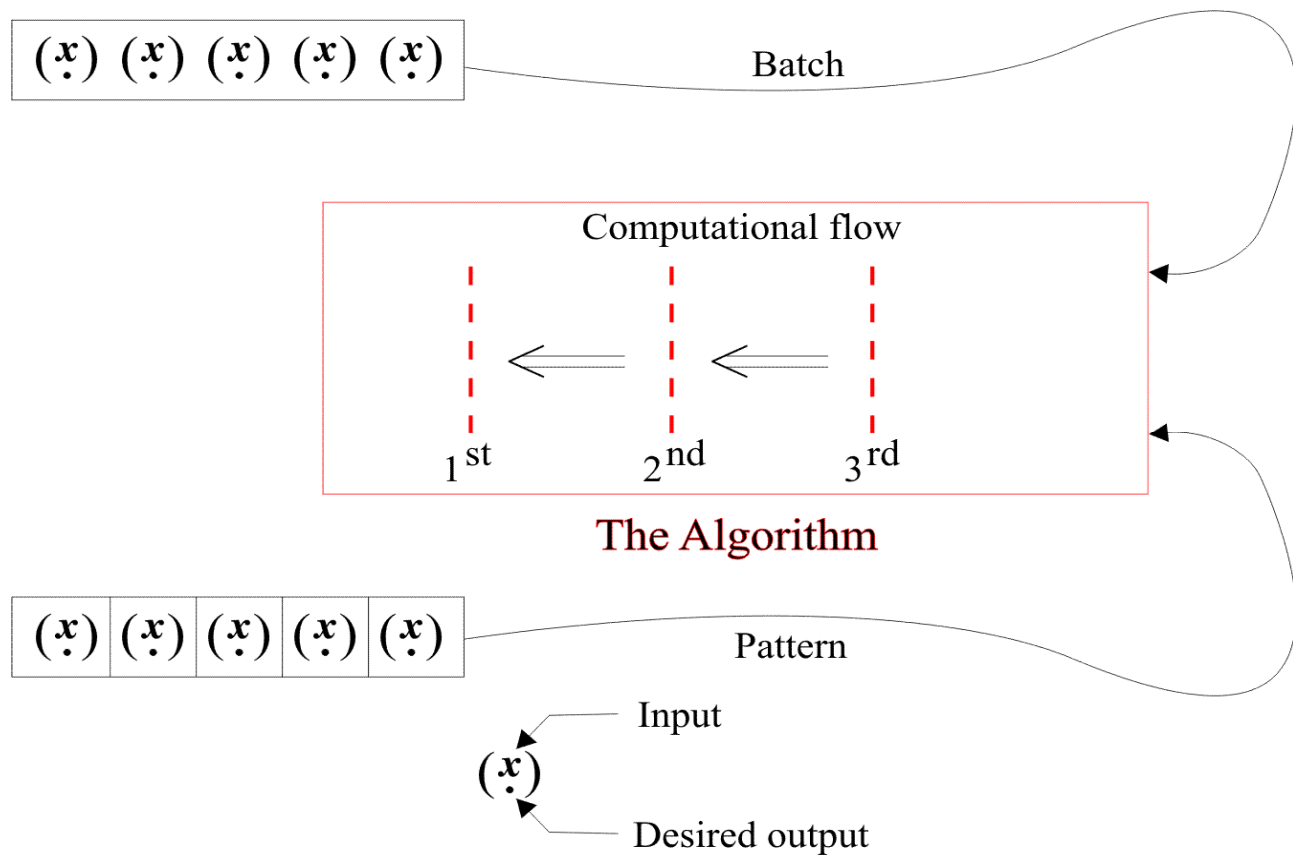
➢ A major problem:
The algorithm may converge to a local minimum

○ Initial point

● Final point

✳ Optimum
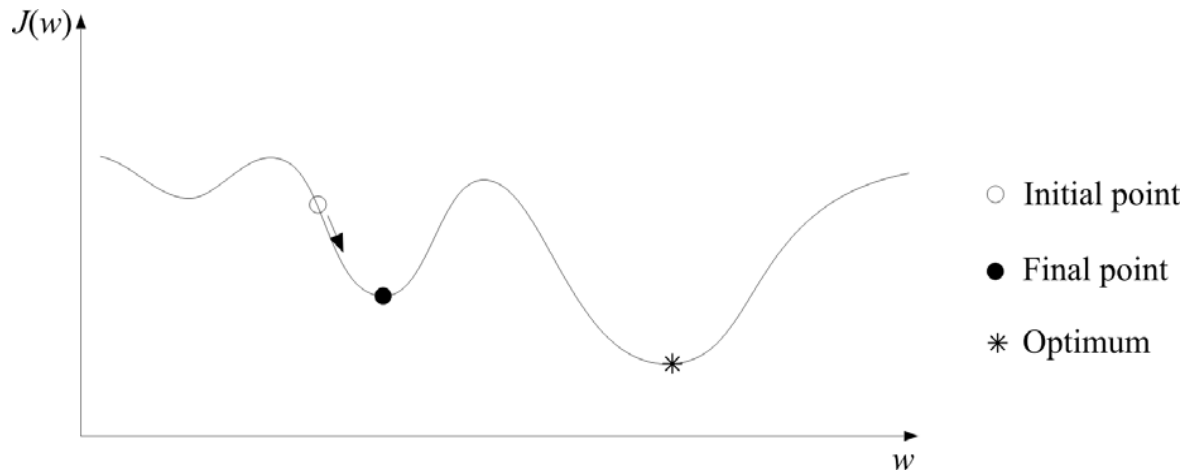
طبقه‌بندی‌کننده‌های غیرخطی

۶

انتخاب
تابع هزینه

➢ The Cost function choice

Examples:

- The Least Squares

$$J = \sum_{i=1}^{N} E(i)$$

$$E(i) = \sum_{m=1}^{k} e_m^2(i) = \sum_{m=1}^{k} (y_m(i) - \hat{y}_m(i))^2$$

$$i = 1, 2, ..., N$$

$y_m(i) \rightarrow$ Desired response of the mth output neuron (1 or 0) for $\underline{x}(i)$

$\hat{y}_m(i) \rightarrow$ Actual response of the mth output neuron, in the interval [0, 1], for input $\underline{x}(i)$

- The cross-entropy

$$J = \sum_{i=1}^{N} E(i)$$

$$E(i) = \sum_{m=1}^{k} \left\{ y_m(i) \ln \hat{y}_m(i) + (1 - y_m(i)) \ln(1 - \hat{y}_m(i)) \right\}$$

This presupposes an interpretation of *y* and *ŷ* as **probabilities**

➢ **Classification error rate.** This is also known as discriminative learning. Most of these techniques use a smoothed version of the classification error.

➢ Remark 1:

A common feature of all the above is the danger of local minimum convergence.

"**Well formed**" cost functions guarantee convergence to a "good" solution, that is one that classifies correctly ALL training patterns, provided such a solution exists.

The cross-entropy cost function **is** a well formed one. The Least Squares **is not**.

➢ Remark 2:

➢ Both, the <u>Least Squares</u> and the <u>cross entropy</u> lead to output values $\hat{y}_m(i)$ that approximate optimally class a-posteriori probabilities!!!

$$\hat{y}_m(i) \cong P(\omega_m | \underline{x}(i))$$

That is, the probability of class ω_m given $\underline{x}(i)$.

This is a very interesting result. It **does not** depend on the underlying distributions. It is a characteristic of **certain** cost functions. How good or bad is the approximation, depends on the underlying model. Furthermore, it is only valid at the global minimum.

35

طبقه‌بندی‌کننده‌های غیرخطی

۷

انتخاب اندازه‌ی شبکه

➢ Choice of the network size.

How big a network can be?
How many layers and how many neurons per layer??

There are two major directions

- Pruning Techniques**:** These techniques start from a large network and then weights and/or neurons are removed iteratively, according to a criterion.

- Constructive techniques:
  They start with a small network and keep increasing it, according to a predetermined procedure and criterion.

─Methods based on parameter sensitivity

$$\delta J = \sum_i g_i \delta w_i + \frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} \sum_i \sum_j h_{ij} \delta w_i \delta w_j$$

+ higher order terms where

$$g_i = \frac{\partial J}{\partial w_i}, \ h_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

Near a minimum and assuming that

$$\delta J \cong \frac{1}{2} \sum_i h_{ii} \delta w_i^2$$

Pruning is now achieved in the following procedure:

✓ Train the network using Backpropagation for a number of steps

✓ Compute the saliencies

$$s_i = \frac{h_{ii} w_i^2}{2}$$

✓ Remove weights with small s_i.

✓ Repeat the process

— Methods based on function regularization

$$J = \sum_{i=1}^{N} E(i) + a E_p(\underline{w})$$

39

The second term favours small values for the weights, e.g.,

$$E_p(\underline{\omega}) = \sum_k h(w_k^2)$$

$$h(w_k^2) = \frac{w_k^2}{w_0^2 + w_k^2}$$

where $w_0 \cong 1$
After some training steps, weights with small values are removed.

- Constructive techniques:
  They start with a small network and keep increasing it, according to a predetermined procedure and criterion.

➤ Remark:

Why not start with a large network and leave the algorithm to decide which weights are small??

This approach is just naïve. It overlooks that classifiers must have good **generalization** properties. A large network can result in small errors for the training set, since it can learn the particular details of the training set. On the other hand, it will not be able to perform well when presented with data unknown to it. The size of the network must be:

- Large enough to learn what makes data of the same class similar and data from different classes dissimilar

- Small enough not to be able to learn underlying differences between data of the same class.

This leads to the so called overfitting.

طبقه‌بندی‌کننده‌های غیرخطی
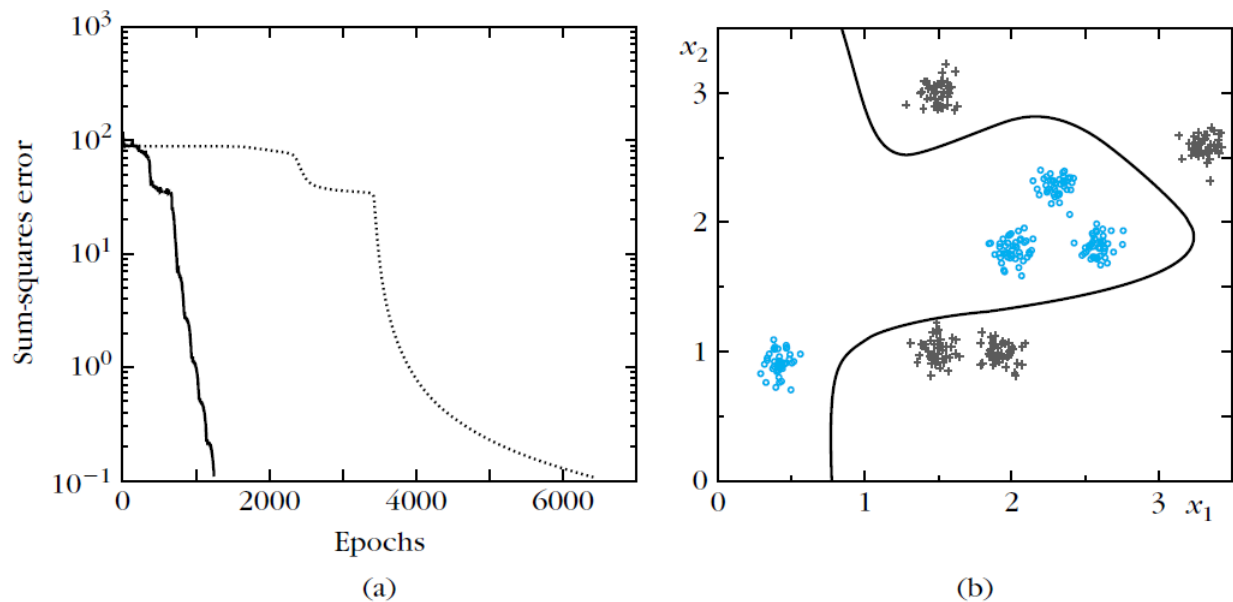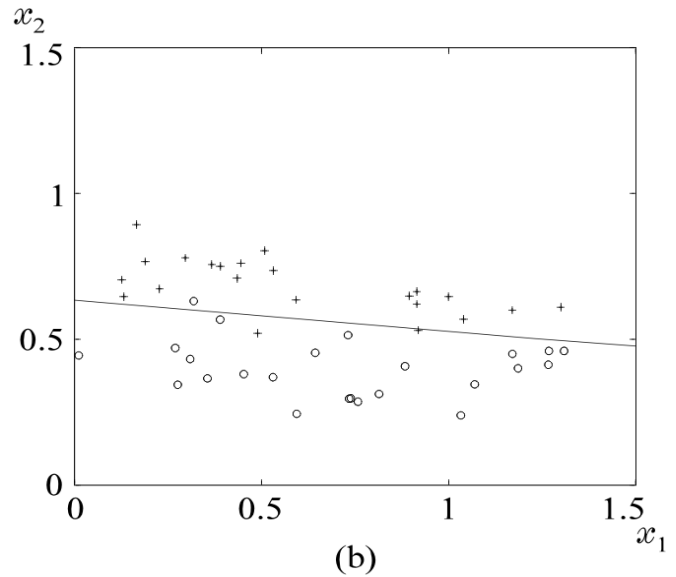
۸

یک مثال
شبیه‌سازی

FIGURE 4.15

(a) Error convergence curves for the adaptive momentum (dark line) and the momentum algorithms. Note that the adaptive momentum leads to faster convergence. (b) The decision curve formed by the multilayer perceptron.

Example:

(a)

(b)
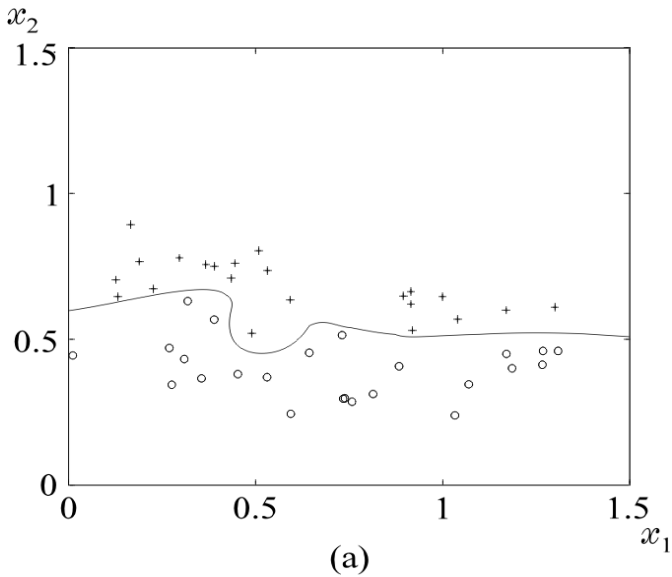
➢ Overtraining is another side of the same coin, i.e.,
the network adapts to the peculiarities of the training set.
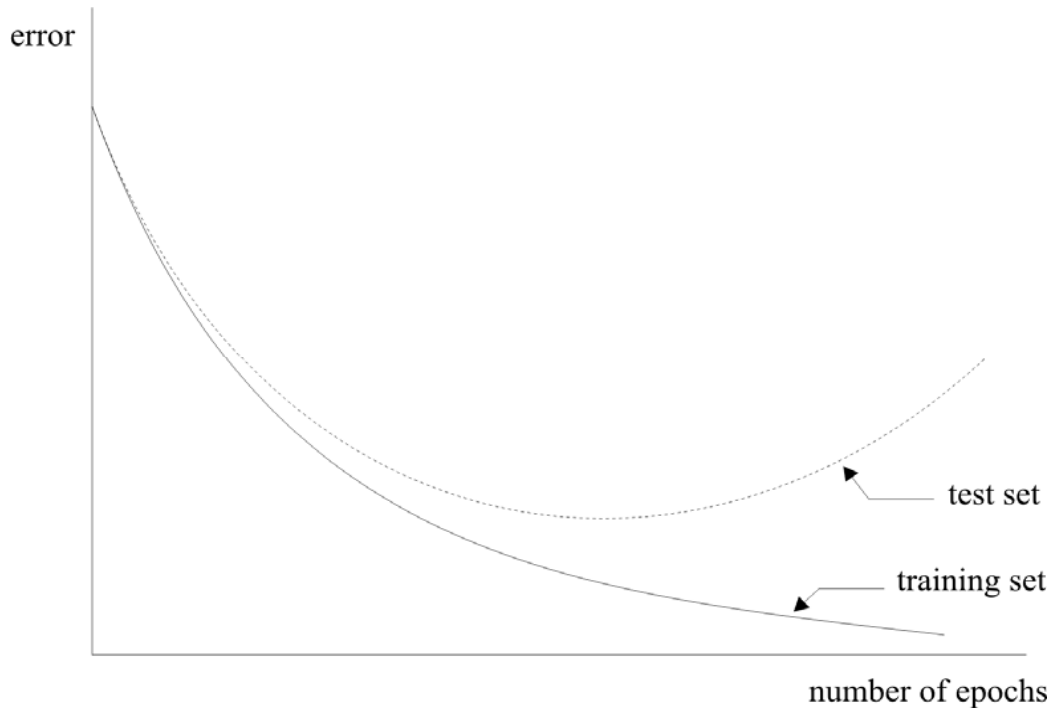
طبقه‌بندی‌کننده‌های غیرخطی

۹

طبقه‌بندی کننده‌های خطی تعمیم‌یافته

❖ Generalized Linear Classifiers

➢ Remember the XOR problem. The mapping

$$\underline{x} \rightarrow \underline{y} = \begin{bmatrix} f(g_1(\underline{x})) \\ f(g_2(\underline{x})) \end{bmatrix}$$

$f(.) \rightarrow$ The activation function transforms the nonlinear task into a linear one.

➢ In the more general case:
- Let $\underline{x} \in R^l$ and a nonlinear classification task.

$$f_i(.), i = 1,2,...,k$$

47

- Are there any functions and an appropriate k, so that the mapping

$$\underline{x} \rightarrow \underline{y} = \begin{bmatrix} f_1(\underline{x}) \\ ... \\ f_k(\underline{x}) \end{bmatrix}$$

  transforms the task into a linear one, in the $\underline{y} \in R^k$ space?

- If this is true, then there exists a hyperplane $\underline{w} \in R^k$ so that

$$\text{If } w_0 + \underline{w}^T \underline{y} > 0, \quad \underline{x} \in \omega_1$$

$$w_0 + \underline{w}^T \underline{y} < 0, \quad \underline{x} \in \omega_2$$

48

➤ In such a case this is equivalent with approximating the nonlinear discriminant function $g(\underline{x})$, in terms of i.e., $f_i(\underline{x})$

$$g(\underline{x}) \cong w_0 + \sum_{i=1}^{k} w_i f_i(\underline{x}) \gtrless 0$$

➤ Given $f_i(\underline{x})$, the task of computing the weights is a linear one.

➤ How sensible is this??

- From the numerical analysis point of view, this is justified if $f_i(\underline{x})$ are interpolation functions.
- From the Pattern Recognition point of view, this is justified by Cover's theorem
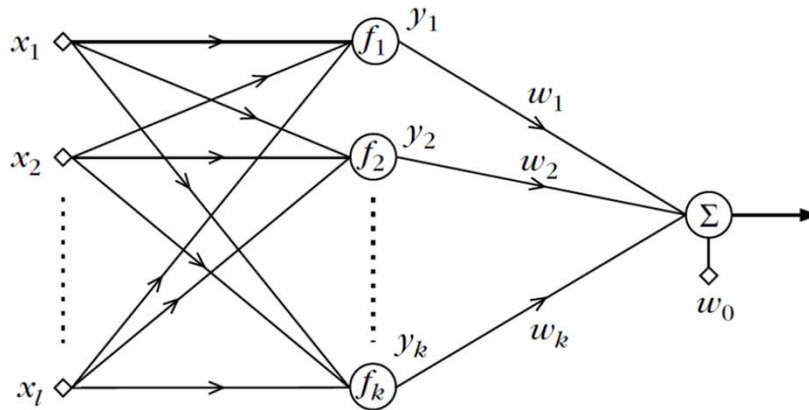
49

FIGURE 4.17

Generalized linear classifier.

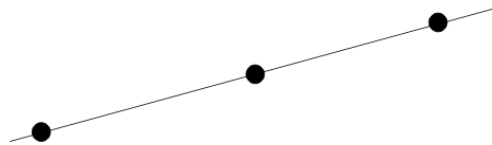طبقه‌بندی‌کننده‌های غیرخطی

۱۰

ظرفیت فضای n-بعدی در دایکوتومی‌های خطی

❖ Capacity of the *l*-dimensional space in Linear Dichotomies

    ➤ Assume N points in $\mathbb{R}^l$ assumed to be in general position, that is:

      Not $\ell + 1$ of these lie on a $\ell - 1$ dimensional space

not in general
position
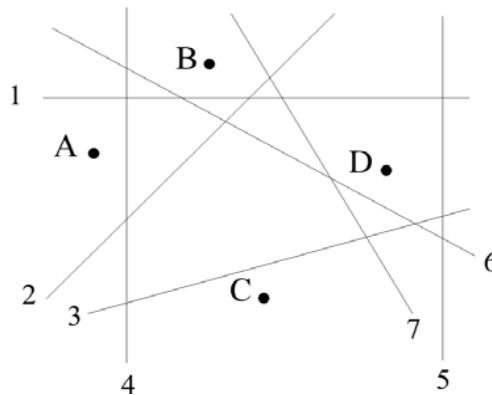
general
position

➤ Cover's theorem states: The number of groupings that can be formed by (l-1)-dimensional hyperplanes to separate N points in two classes is

$$O(N,l) = 2\sum_{i=0}^{l}\binom{N-1}{i}, \qquad \binom{N-1}{i} = \frac{(N-1)!}{(N-1-i)!\,i!}$$

Example: $N = 4$, $l = 2$, $O(4,2) = $ **14**

Notice: The total number of possible groupings is $2^4 = 16$

Figure 4.18 shows two examples of such hyperplanes resulting in $O(4, 2) = 14$ and $O(3, 2) = 8$ two-class groupings, respectively. The seven lines of Figure 4.18a form the following groupings. [(ABCD)], [A,(BCD)], [B,(ACD)], [C,(ABD)], [D,(ABC)], [(AB),(CD)], and [(AC),(BD)]. Each grouping corresponds to two possibilities. For example, (ABCD) can belong to either class ω_1 or ω_2.

$$\text{for } N \leq l+1, O(N, l) = 2^N$$
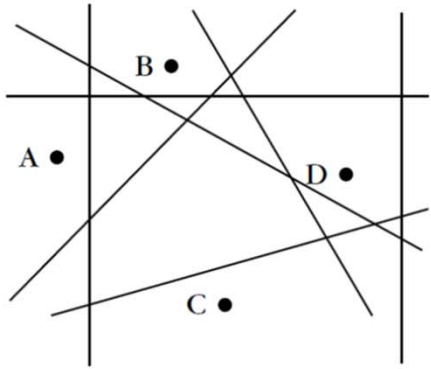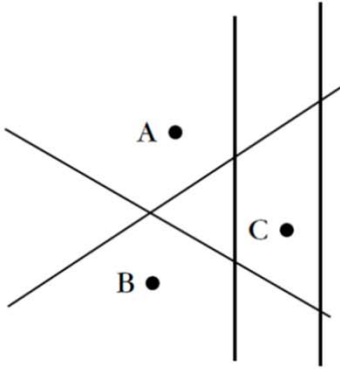
(a)   (b)

FIGURE 4.18 $O(4, 2) = 14$ $O(3, 2) = 8$

Number of linear dichotomies (a) for four and (b) for three points.

54

➢ Probability of grouping N points in two linearly separable classes is

$$\frac{O(N,l)}{2^N} = P_N^l$$
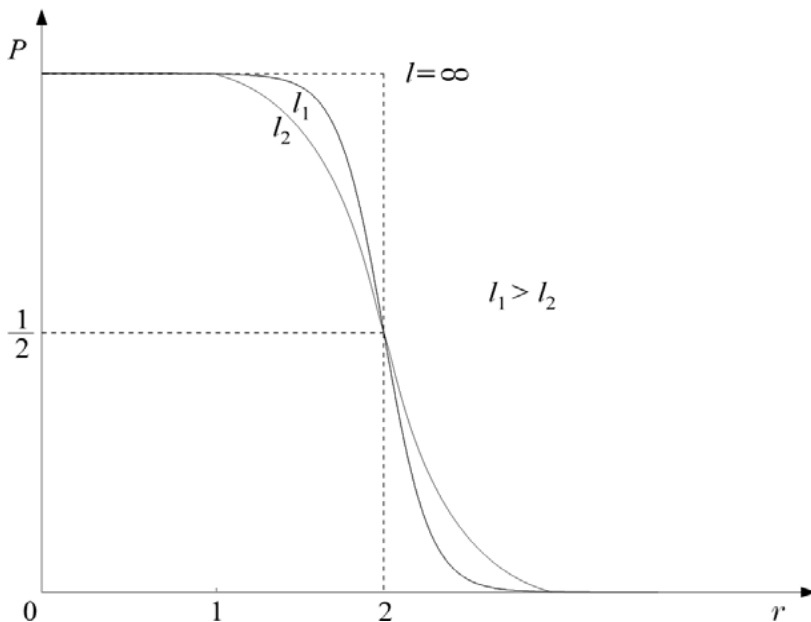
$$N = r(l+1)$$

Thus, the probability of having N points in linearly separable classes tends to 1, for large l, provided $N < 2(l+1)$

Hence, by mapping to a higher dimensional space, we increase the probability of **linear separability,** provided the space is not too densely populated.

طبقه‌بندی‌کننده‌های غیرخطی

۱۱

شبکه‌های توابع پایه‌ی شعاعی (RBF)

❖ Radial Basis Function Networks (RBF)

➢ Choose
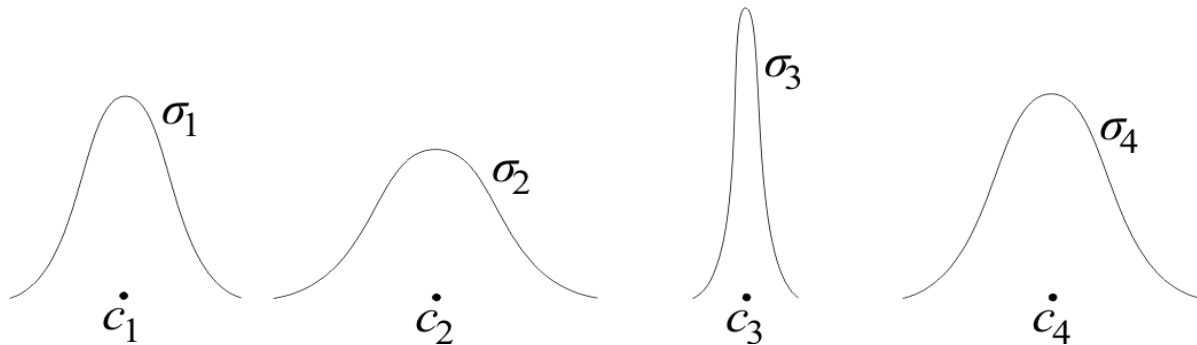
$$f_i(\underline{x}) = \exp\left(-\frac{\|x - \underline{c}_i\|^2}{2\sigma_i^2} \right)$$
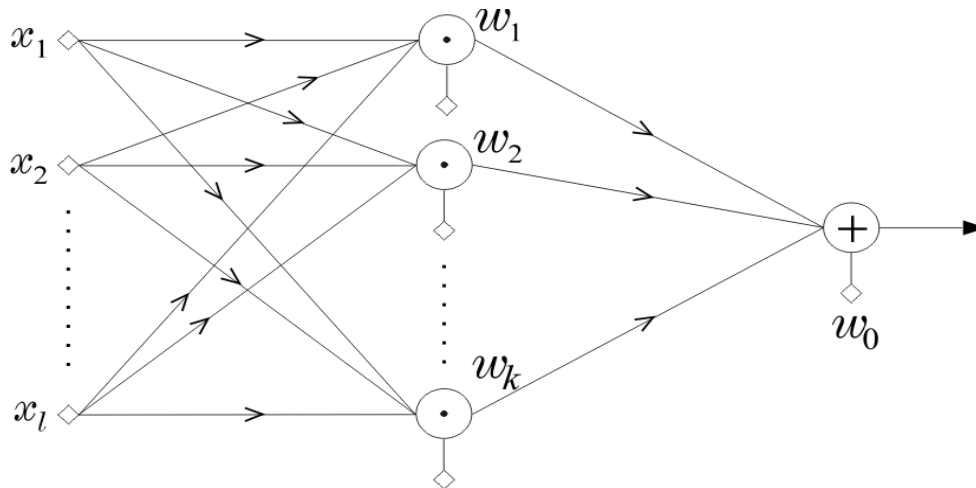
Equivalent to a single layer network, with RBF activations and linear output node.

➢ Example: The XOR problem

- Define:

$$\underline{c}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \underline{c}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \sigma_1 = \sigma_2 = \frac{1}{\sqrt{2}}$$

$$\underline{y} = \begin{bmatrix} \exp(-\|\underline{x} - \underline{c}_1\|^2) \\ \exp(-\|\underline{x} - \underline{c}_2\|^2) \end{bmatrix}$$

- $\begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.135 \\ 1 \end{bmatrix}, \ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0.135 \end{bmatrix}$

  $\begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.368 \\ 0.368 \end{bmatrix}, \ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0.368 \\ 0.368 \end{bmatrix}$

60
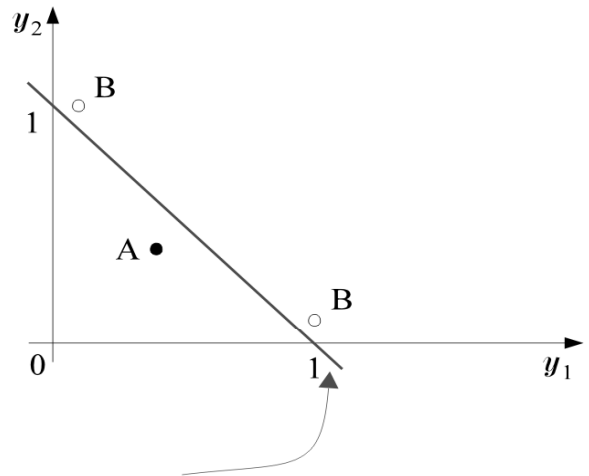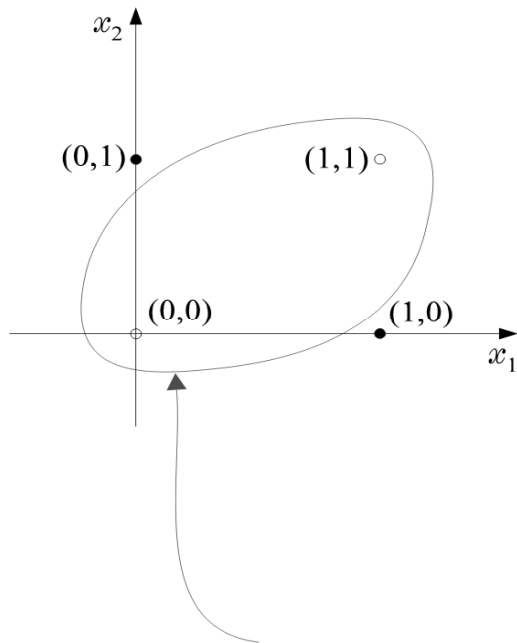
$$g(\underline{y}) = y_1 + y_2 - 1 = 0$$

$$g(\underline{x}) = \exp(-\|\underline{x} - \underline{c}_1\|^2) + \exp(-\|\underline{x} - \underline{c}_2\|^2) - 1 = 0$$

➢ Training of the RBF networks

- **Fixed centers:** Choose centers randomly among the data points. Also fix σ_i's. Then

$$g(\underline{x}) = w_0 + \underline{w}^T \underline{y}$$

   is a typical linear classifier design.

- **Training of the centers:** This is a nonlinear optimization task

- Combine supervised and unsupervised learning procedures.

- The unsupervised part reveals <u>clustering tendencies</u> of the data and assigns the centers at the <u>cluster representatives</u>.

62

طبقه‌بندی‌کننده‌های غیرخطی

۱۲

تخمین‌گرهای جهانی

❖ Universal Approximators

It has been shown that any nonlinear continuous function can be approximated arbitrarily close, both, by a two layer perceptron, with sigmoid activations, and an RBF network, provided a large enough number of nodes is used.

❖ **Multilayer Perceptrons** vs. **RBF networks**

➤ MLP's involve activations of global nature. All points on a plane $w^T \underline{x} = c$ give the same response.

➤ RBF networks have activations of a local nature, due to the exponential decrease as one moves away from the centers.

➤ MLP's learn slower but have better generalization properties

طبقه‌بندی‌کننده‌های غیرخطی

۱۳

ماشین‌های بردار پشتیبان

(حالت غیرخطی)

❖ Support Vector Machines: The non-linear case

➢ Recall that the probability of having linearly separable classes increases as the **dimensionality** of the feature vectors **increases**. Assume the mapping:

$$\underline{x} \in \mathbb{R}^l \to \underline{y} \in \mathbb{R}^k, \ \ k > l$$

Then use SVM in $\mathbb{R}^k$

➢ Recall that in this case the dual problem formulation will be

$$\underset{\underline{\lambda}}{\text{maximize}} \ (\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \underline{y}_i^T \underline{y}_j)$$

where $\underline{y}_i \in \mathbb{R}^k$

Also, the classifier will be

$$g(\underline{y}) = \underline{w}^T \underline{y} + w_0$$

$$= \sum_{i=1}^{N_s} \lambda_i y_i \underline{y}_i \underline{y}$$

where $\underline{x} \rightarrow \underline{y} \in \mathbb{R}^k$

Thus, inner products in a high dimensional space are involved, hence

- High complexity

➢ Something clever: Compute the inner products in the **high** dimensional space as functions of inner products performed in the **low** dimensional space!!!

➢ Is this POSSIBLE?? Yes. Here is an example

Let $\underline{x} = \left[x_1, \ x_2 \right]^T \in \mathbb{R}^2$

Let $\underline{x} \rightarrow \underline{y} = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix} \in \mathbb{R}^3$

Then, it is easy to show that

$$ \underline{y}_i^T \underline{y}_j = (\underline{x}_i^T \underline{x}_j)^2. $$

➢ Mercer's Theorem

Let $\underline{x} \rightarrow \underline{\Phi}(\underline{x}) \in H$

Then, the inner product in H

$$\sum_r \Phi_r(\underline{x}) \Phi_r(\underline{y}) = K(\underline{x}, \underline{y})$$

where

$$\int K(\underline{x}, \underline{y}) g(\underline{x}) g(\underline{y}) d\underline{x} d\underline{y} \geq 0$$

for **any** $g(\underline{x})$, $\underline{x}$:

$$\int g^2(\underline{x}) d\underline{x} < +\infty$$

$K(\underline{x}, \underline{y})$ symmetric function known as **kernel**.

69

➤ The opposite is also true. Any kernel, with the above properties, corresponds to an inner product in SOME space!!!

➤ Examples of kernels

- Radial Basis Functions:

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

- Polynomial:

$$K(\underline{x}, \underline{z}) = (\underline{x}^T \underline{z} + 1)^q, \ q > 0$$

- Hyperbolic Tangent:

$$K(\underline{x}, \underline{z}) = \tanh(\beta \underline{x}^T \underline{z} + \gamma)$$

for appropriate values of β, γ.

➢ SVM Formulation

- **Step 1:** Choose appropriate kernel. This implicitly assumes a mapping to a higher dimensional (yet, not known) space.

- **Step 2:**

$$\max_{\underline{\lambda}}(\sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j K(\underline{x}_i, \underline{x}_j))$$

subject to: $0 \le \lambda_i \le C, \quad i = 1, 2, ..., N$

$$\sum_i \lambda_i y_i = 0$$

This results to an implicit combination

$$\underline{w} = \sum_{i=1}^{N_s} \lambda_i y_i \underline{\varphi}(\underline{x}_i)$$

71

- **Step 3:** Assign $\underline{x}$ to

$$g(\underline{x}) = \sum_{i=1}^{N_s} \lambda_i y_i \, K(\underline{x}_i, \underline{x}) + w_0 \overset{\omega_1}{\underset{\omega_2}{\gtrless}} 0$$

- The SVM Architecture
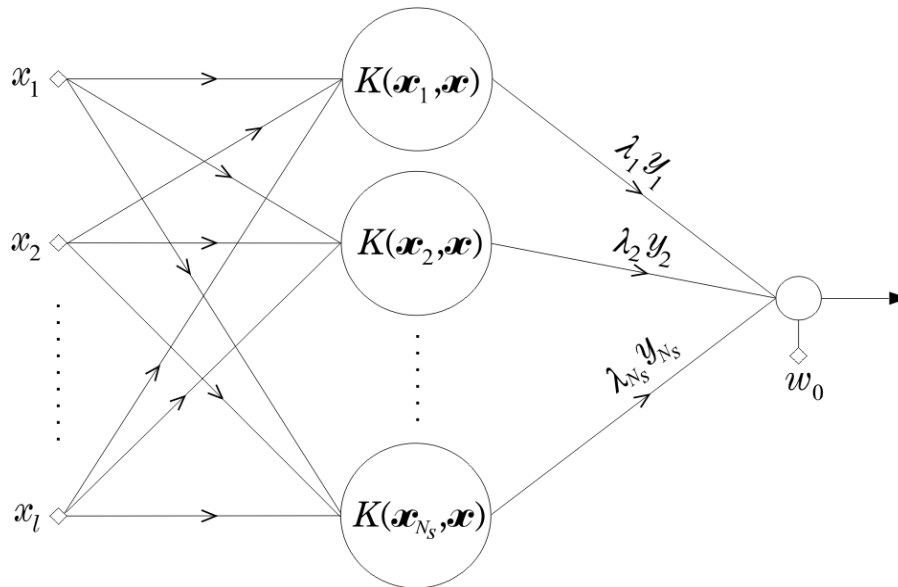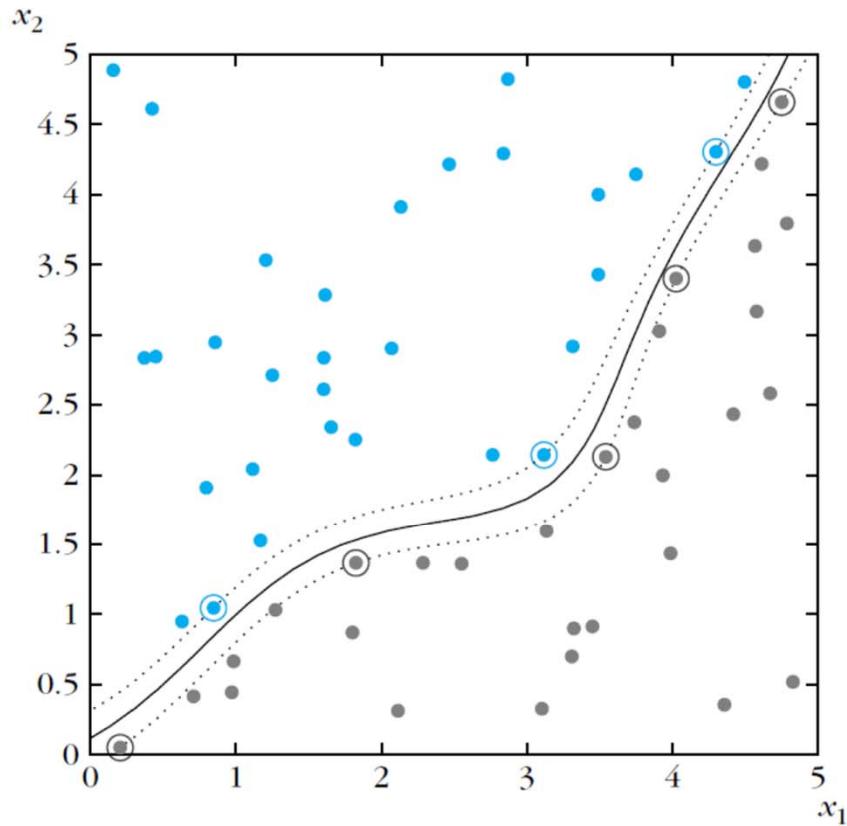
FIGURE 4.24

Example of a nonlinear SVM classifier for the case of two nonlinearly separable classes. The Gaussian RBF kernel was used. Dotted lines mark the margin and circled points the support vectors.

طبقه‌بندی‌کننده‌های غیرخطی

۱۴

درخت‌های تصمیم

❖ Decision Trees

This is a family of non-linear classifiers. They are multistage decision systems, in which classes are sequentially rejected, until a finally accepted class is reached. To this end:

➤ The feature space is split into **unique** regions in a sequential manner.

➤ Upon the arrival of a feature vector, sequential decisions, assigning features to specific regions, are performed along a path of nodes of an appropriately constructed tree.

➤ The sequence of decisions is applied to individual features, and the queries performed in each node are of the type:

$$\text{is feature} \quad x_i \leq \alpha$$

where α is a pre-chosen (during training) threshold.

75

➤ The figures below are such examples. This type of trees is known as Ordinary Binary Classification Trees (OBCT). The decision hyperplanes, splitting the space into regions, are parallel to the axis of the spaces. Other types of partition are also possible, yet less popular.
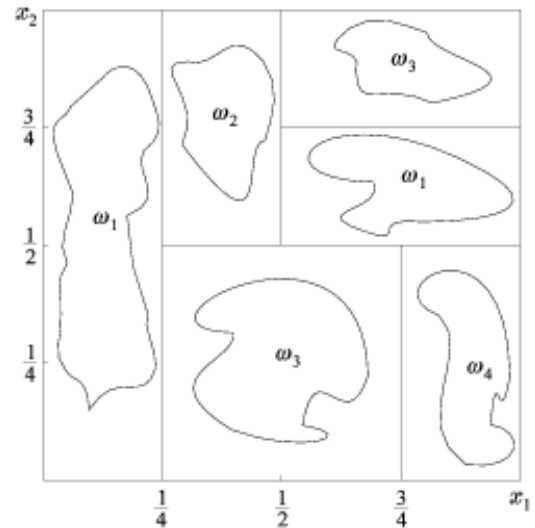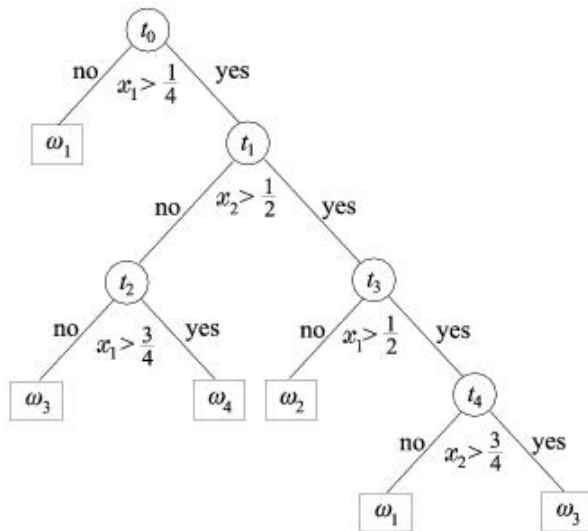
➢ Design Elements that define a decision tree.

- Each node, *t,* is associated with a subset $X_t \subseteq X$, where X is the training set. At each node, X_t is split into two (binary splits) disjoint descendant subsets $X_{t,Y}$ and $X_{t,N}$, where

$$X_{t,Y} \cap X_{t,N} = \varnothing$$
$$X_{t,Y} \cup X_{t,N} = X_t$$

$X_{t,Y}$ is the subset of X_t for which the answer to the query at node *t* is YES. $X_{t,N}$ is the subset corresponding to NO. The split is decided according to an adopted question (query).

77

- A splitting criterion must be adopted for the best split of X_t into $X_{t,Y}$ and $X_{t,N}$.

- A stop-splitting criterion must be adopted that controls the growth of the tree and a node is declared as terminal (leaf).

- A rule is required that assigns each (terminal) leaf to a class.

➢ Set of Questions: In OBCT trees the set of questions is of the type

$$\text{is} \quad x_i \le \alpha \quad ?$$

The choice of the specific x_i and the value of the threshold α, for each node t, are the results of searching, during training, among the features and a set of possible threshold values. The final combination is the one that results to the best value of a criterion.

➢ Splitting Criterion: The main idea behind splitting at each node is the resulting descendant subsets $X_{t,Y}$ and $X_{t,N}$ to be more class homogeneous compared to X_t. Thus the criterion must be in harmony with such a goal. A commonly used criterion is the node impurity:

$$I(t) = -\sum_{i=1}^{M} P(\omega_i \mid t) \log_2 P(\omega_t \mid t)$$

and
$$P(\omega_i \mid t) \approx \frac{N_t^i}{N_t}$$

where N_t^i is the number of data points in X_t that belong to class ω_i. The decrease in node impurity is defined as:

$$\Delta I(t) = I(t) - \frac{N_{t,Y}}{N_t} I(t_Y) - \frac{N_{t,N}}{N_t} I(t_N)$$

Example 4.2

In a tree classification task, the set X_t, associated with node t, contains $N_t = 10$ vectors. Four of these belong to class ω_1, four to class ω_2, and two to class ω_3, in a three-class classification task. The node splitting results into two new subsets X_{tY}, with three vectors from ω_1, and one from ω_2, and X_{tN} with one vector from ω_1, three from ω_2, and two from ω_3. The goal is to compute the decrease in node impurity after splitting.

We have that

$$I(t) = -\frac{4}{10} \log_2 \frac{4}{10} - \frac{4}{10} \log_2 \frac{4}{10} - \frac{2}{10} \log_2 \frac{2}{10} = 1.521$$

$$I(t_Y) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} = 0.815$$

$$I(t_N) = -\frac{1}{6} \log_2 \frac{1}{6} - \frac{3}{6} \log_2 \frac{3}{6} - \frac{2}{6} \log_2 \frac{2}{6} = 1.472$$

Hence, the impurity decrease after splitting is

$$\Delta I(t) = 1.521 - \frac{4}{10}(0.815) - \frac{6}{10}(1.472) = 0.315$$

- The goal is to choose the parameters in each node (feature and threshold) that result in a split with the highest decrease in impurity.

- Why highest decrease? Observe that the highest value of *I(t)* is achieved if all classes are equiprobable, i.e., X_t is the least homogenous.

➤ **Stop - splitting rule**. Adopt a threshold *T* and stop splitting a node (i.e., assign it as a leaf), if the impurity decrease is less than *T*. That is, node *t* is "pure enough".

➤ Class Assignment Rule: Assign a leaf to a class ω_j, where:

$$j = \arg \max_i P(\omega_i \mid t)$$

➢ Summary of an OBCT algorithmic scheme:

- Begin with the root node, i.e., $X_t = X$

- For each new node t

  * For every feature $x_k, k = 1, 2, \ldots, l$

    . For every value $\alpha_{kn}, n = 1, 2, \ldots, N_{tk}$
      . Generate X_{tY} and X_{tN} according to the answer in the question: is $x_k(i) \leq \alpha_{kn}, \ i = 1, 2, \ldots, N_t$
      . Compute the impurity decrease
    . End
    . Choose α_{kn_0} leading to the maximum decrease w.r. to x_k

  * End

  * Choose x_{k_0} and associated $\alpha_{k_0 n_0}$ leading to the overall maximum decrease of impurity

  * If stop-splitting rule is met declare node t as a leaf and designate it with a class label

  * If not, generate two descendant nodes t_Y and t_N with associated subsets X_{tY} and X_{tN}, depending on the answer to the question: is $x_{k_0} \leq \alpha_{k_0 n_0}$

- End

83

➤ Remarks:

- A critical factor in the design is the size of the tree. Usually one grows a tree to a large size and then applies various pruning techniques.

- Decision trees belong to the class of unstable classifiers. This can be overcome by a number of "averaging" techniques. Bagging is a popular technique. Using bootstrap techniques in *X,* various trees are constructed, T_i, *i=1, 2, …, B.* The decision is taken according to a majority voting rule.

84

طبقه‌بندی‌کننده‌های غیرخطی

۱۵

منابع

منبع اصلی

Pattern Recognition

FOURTH EDITION

Sergios Theodoridis
Konstantinos Koutroumbas
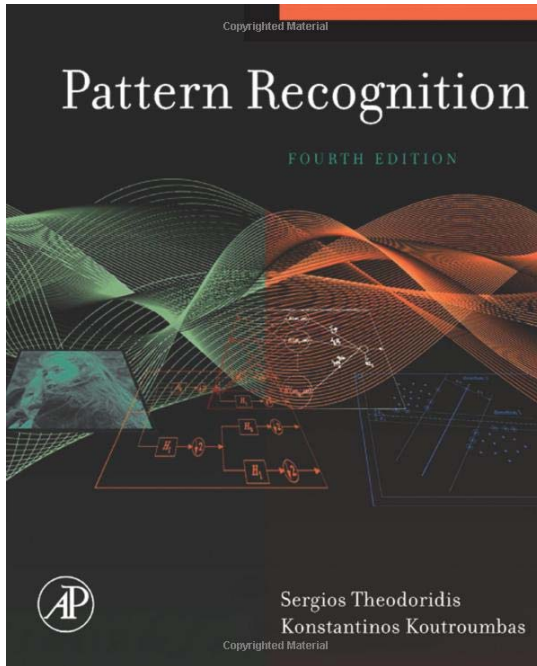
S. Theodoridis, K. Koutroumbas,
Pattern Recognition,
Fourth Edition, Academic Press, 2009.

Chapter 4 (4-1 .. 4-20)

CHAPTER 4

Nonlinear Classifiers

4.1 INTRODUCTION

In the previous chapter we dealt with the design of linear classifiers described by linear discriminant functions (hyperplanes) $g(x)$. In the simple two-class case, we saw that the perceptron algorithm computes the weights of the linear function $g(x)$, provided that the classes are linearly separable. For nonlinearly separable classes, linear classifiers were optimally designed, for example, by minimizing the squared error. In this chapter we will deal with problems that are not linearly separable and for which the design of a linear classifier, even in an optimal way, does not lead to satisfactory performance. The design of nonlinear classifiers emerges now as an inescapable necessity.

4.2 THE XOR PROBLEM

To seek nonlinearly separable problems one does not need to go into complicated situations. The well-known *Exclusive OR (XOR)* Boolean function is a typical example of such a problem. Boolean functions can be interpreted as classification tasks. Indeed, depending on the values of the input binary data $x = [x_1, x_2, \ldots, x_l]^T$, the output is either 0 or 1, and x is classified into one of the two classes $A(1)$ or $B(0)$. The corresponding truth table for the XOR operation is shown in Table 4.1.

Figure 4.1 shows the position of the classes in space. It is apparent from this figure that no single straight line exists that separates the two classes. In contrast, the other two Boolean functions, AND and OR, are linearly separable. The corresponding truth tables for the AND and OR operations are given in Table 4.2 and the respective class positions in the two-dimensional space are shown in Figure 4.2a and 4.2b. Figure 4.3 shows a perceptron, introduced in the previous chapter, with synaptic weights computed so as to realize an OR gate (verify).

Our major concern now is first to tackle the XOR problem and then to extend the procedure to more general cases of nonlinearly separable classes. Our kickoff point will be geometry.
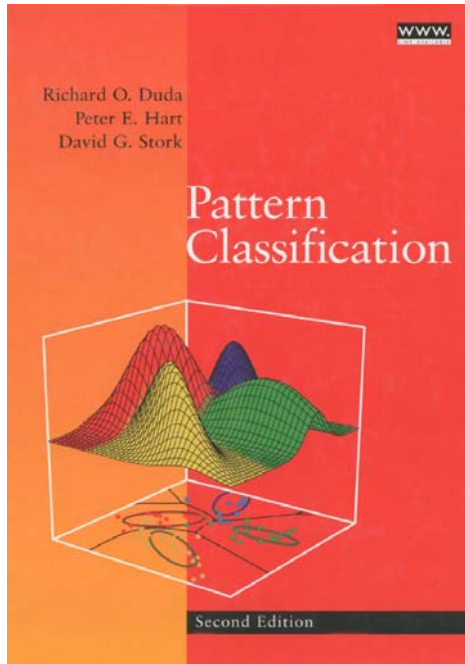
151

منبع کمکی

R.O. Duda, P.E. Hart, and D.G. Stork,
Pattern Classification,
Second Edition, John Wiley & Sons, Inc., 2001.

Chapter 6
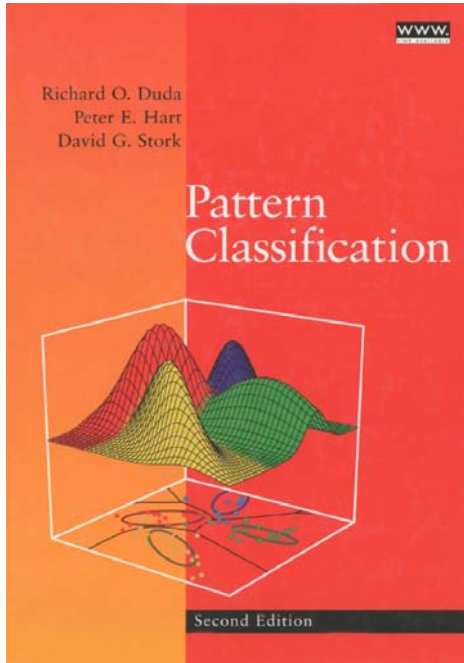
C H A P T E R

6

MULTILAYER NEURAL NETWORKS

6.1 INTRODUCTION

We saw in Chapter 5 a number of methods for training classifiers consisting of input units connected by modifiable weights to output units. The LMS algorithm, in particular, provided a powerful gradient descent method for reducing the error, even when the patterns are not linearly separable. Unfortunately, the class of solutions that can be obtained from such networks—comprising hyperplane decision boundaries—while surprisingly good on a range of real-world problems, is simply not general enough in demanding applications: there are many problems for which linear discriminants are insufficient for minimum error.

With a clever choice of nonlinear φ functions, however, we can obtain arbitrary decision regions, in particular those leading to minimum error. The central difficulty is, naturally, choosing the appropriate nonlinear functions. One brute force approach might be to choose a complete basis set such as all polynomials, but this will not work; such a classifier would have too many free parameters to be determined from a limited number of training patterns (Chapter 9). Alternatively, we may have prior knowledge relevant to the classification problem and this might guide our choice of nonlinearity. However, we have seen no principled or automatic method for finding the nonlinearities in the absence of such information. What we seek, then, is a way to *learn* the nonlinearity at the same time as the linear discriminant. This is the approach of multilayer neural networks or multilayer Perceptrons: The parameters governing the nonlinear mapping are learned at the same time as those governing the linear discriminant.

We shall revisit the limitations of the two-layer networks of the previous chapter,* and see how three- and four-layer nets overcome those drawbacks—indeed

___
*Some authors describe such networks as *single* layer networks because they have only one layer of modifiable weights, but we shall instead refer to them based on the number of layers of *units*.

282

R.O. Duda, P.E. Hart, and D.G. Stork,
Pattern Classification,
Second Edition, John Wiley & Sons, Inc., 2001.

Chapter 8 (8-1 .. 8-4)

C H A P T E R

8

NONMETRIC METHODS

8.1 INTRODUCTION

We have considered pattern recognition based on feature vectors of real-valued and discrete-valued numbers, and in all cases there has been a natural measure of distance between vectors. For instance in the nearest-neighbor classifier the notion figures conspicuously—indeed it is the core of the technique—while for neural networks the notion of similarity appears when two input vectors sufficiently "close" lead to similar outputs. Most practical pattern recognition methods address problems of this sort, where feature vectors are real-valued and there exists some notion of metric.

NOMINAL DATA

But suppose a classification problem involves *nominal* data—for instance descriptions that are discrete and without any natural notion of similarity or even ordering. Consider the use of information about teeth in the classification of fish and sea mammals. Some teeth are small and fine (as in baleen whales) for straining tiny prey from the sea. Others (as in sharks) come in multiple rows. Some sea creatures, such as walruses, have tusks. Yet others, such as squid, lack teeth altogether. There is no clear notion of similarity (or metric) for this information about teeth. For instance, the teeth of a baleen whale and the tusks of a walrus are no more "similar" than are the teeth of a shark to the teeth to a flounder.

PROPERTY
D-TUPLE

STRING

Thus in this chapter our attention turns away from describing patterns by vectors of real numbers and toward using *lists* of attributes. A common approach is to specify the values of a fixed number of properties by a *property d-tuple* For example, consider describing a piece of fruit by four properties: color, texture, taste and size. Then a particular piece of fruit might be described by the 4-tuple {*red, shiny, sweet, small*}, which is a shorthand for color = red, texture = shiny, taste = sweet and size = small. Another common approach is to describe the pattern by a variable length *string* of nominal attributes, such as a sequence of base pairs in a segment of DNA—for example, "AGCTTCAGATTCCA."* Such lists or strings might themselves be the output of other component classifiers of the type we have seen elsewhere. For instance, we might train a neural network to recognize different component brush strokes used in Chinese and Japanese characters (roughly a dozen basic forms); a classifier would then accept as inputs a list of these nominal attributes and make the final, full character classification.

*We often put strings between quotation marks, particularly if this will help to avoid ambiguities.

394