Linear discriminant function based classifier

-Slides compiled by Sanghamitra De

Linear discriminant functions & decision surfaces

- Linear discriminant functions **have a variety of analytical properties**.
- They **can be optimal if the underlying distributions are cooperative**, e.g. Gaussians having equal covariance, as might be obtained through an intelligent choice of feature detectors.
- Linear discriminant functions are relatively **easy to compute** and in the absence of information suggesting otherwise, linear classifiers are attractive candidates for initial, trial classifiers.
- They also illustrate a number of very important principles which will be used more fully in neural networks

Linear discriminant functions & decision surfaces

- The problem of finding a linear discriminant function will be formulated as a problem of minimizing a criterion function.
- The obvious criterion function for classification purposes is the *sample risk*, or *training error* — the average loss incurred in classifying the set of training samples.

The Two-Category Case

- A discriminant function that is a linear combination of the components of x can be written as

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0, \qquad (1)$$

where threshold w is the weight vector and ω_0 the bias or threshold weight.

- A two-category weight linear classifier implements the following decision rule: Decide ω_1 if $g(\mathbf{x}) > 0$ and ω_2 if $g(\mathbf{x}) < 0$.
- Thus, x is assigned to ω_1 if the inner product $\mathbf{w}^t\mathbf{x}$ exceeds the threshold $-\omega_0$ and ω_2 otherwise.

The Two-Category Case

- Figure next shows a typical implementation, a clear example of the general structure of a pattern recognition system

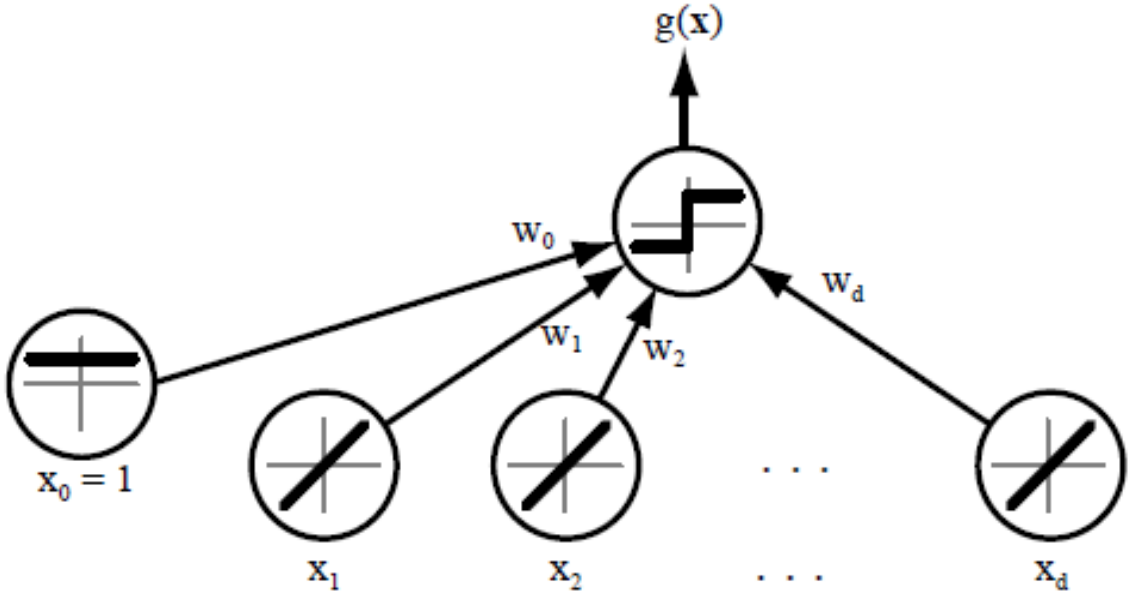


Figure 5.1: A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the output unit sums all these products and emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a -1 otherwise.

The Two-Category Case

- The equation g(x) = 0 defines the decision surface that separates points assigned to ω_1 from points assigned to ω_2.
- When g(x) is linear, this decision surface is a *hyperplane*. If x_1 and x_2 are both on the decision surface, then

$$\mathbf{w}^t\mathbf{x}_1 + w_0 = \mathbf{w}^t\mathbf{x}_2 + w_0$$

or

$$\mathbf{w}^t(\mathbf{x}_1 - \mathbf{x}_2) = 0,$$

The Two-Category Case

- This shows that w is normal to any vector lying in the hyperplane.
- In general, the hyperplane H divides the feature space into two half spaces, decision region R_1 for ω_1 and region R_2 for ω_2.
- Since g(x) > 0 if x is in R_1, it follows that the normal vector w points into R_1.
- It is sometimes said that any x in R_1 is on the positive side of H, and any x in R_2 is on the *negative* side.

The Two-Category Case

- In particular, the distance from the origin to H is given by $w_0/\|w\|$
- If $w_0 > 0$ the origin is on the positive side of H, and if $w_0 < 0$ it is on the negative side.
- If $w_0 = 0$, then g(x) has the homogeneous form $\mathbf{w}^t\mathbf{x}$, and the hyperplane passes through the origin.
- A geometric illustration of these algebraic results is given in the figure next.

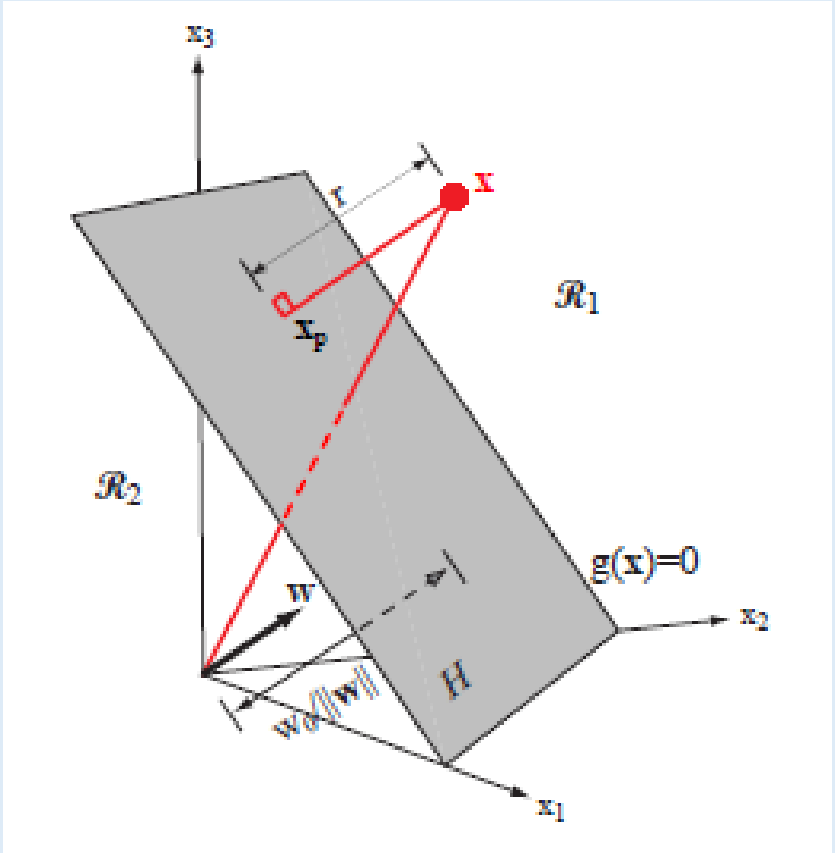


Figure 5.2: The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$).

The Two-Category Case

- So, **a linear discriminant function divides the feature space by a hyperplane decision surface.**
- The **orientation of the surface is determined by the normal vector w, and the location of the surface is determined by the bias ω_0.**
- The discriminant function g(x) is proportional to the signed distance from x to the hyperplane, with g(x) > 0 when x is on the positive side, and g(x) < 0 when x is on the negative side.

The Multicategory Case

- There is more than one way to devise multicategory classifiers employing linear discriminant functions.
- **Method 1:** Reduce the problem to $c - 1$ two-class problems, where the i^{th} problem is solved by a linear discriminant function that separates points assigned to ω_i from those not assigned to ω_i.
- **Method 2:** Use $c(c-1)/2$ linear discriminants, one for every pair of classes.
- As illustrated in next figure, both of these approaches can lead to regions in which the classification is undefined.

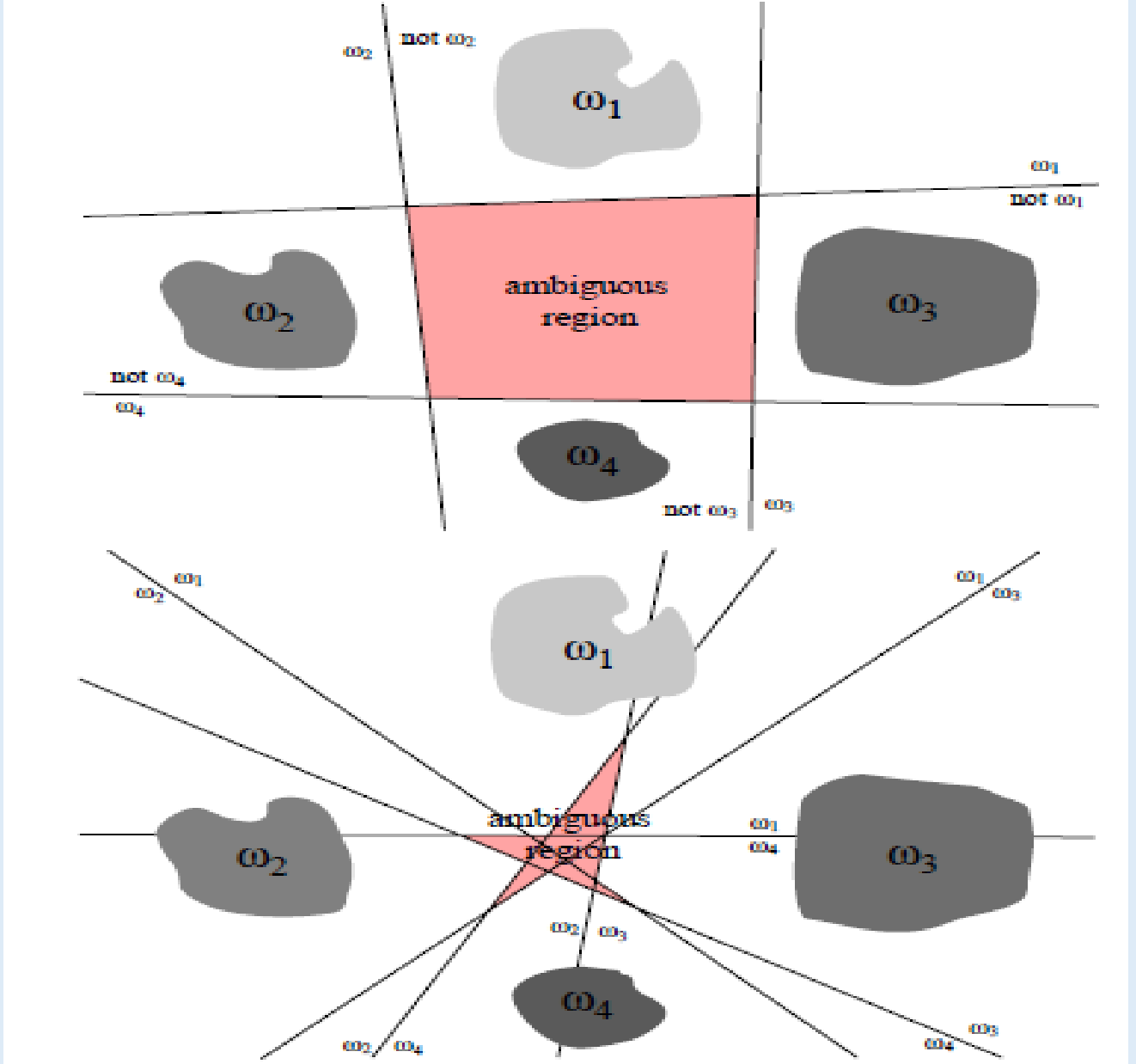


Figure 5.3: Linear decision boundaries for a four-class problem. The top figure shows ω_i/not ω_i dichotomies while the bottom figure shows ω_i/ω_j dichotomies. The pink regions have ambiguous category assigments.

The Multicategory Case

- This problem can be avoided by defining c linear discriminant functions and assigning **x** to ω_i if $g_i(x) > g_j(x)$ for all $j \neq i$; in case of ties, the classification is left undefined.

$$g_i(\mathbf{x}) = \mathbf{w}^t \mathbf{x}_i + w_{i0} \qquad i = 1, \ldots, c,$$

- The resulting classifier is called a ***linear machine***.
- A linear machine divides the feature space into c decision regions, with $g_i(\mathbf{x})$ being the largest discriminant if **x** is in region R_i.
- If R_i and R_j are contiguous, the boundary between them is a portion of the hyperplane H_{ij} defined by:

$$g_i(\mathbf{x}) = g_j(\mathbf{x})$$

or,

$$(\mathbf{w}_i - \mathbf{w}_j)^t \mathbf{x} + (w_{i0} - w_{j0}) = 0.$$

The Multicategory Case

- It follows at once that $w_i - w_j$ is normal to H_{ij} , and the signed distance from $\mathbf{x}$ to H_{ij} is given by $(g_i(\mathbf{x}) - g_j(\mathbf{x}))/\|w_i - w_j\|$.
- Thus, with the linear machine it is not the weight vectors themselves but their *differences* that are important.
- While there are $c(c - 1)/2$ pairs of regions, they need not all be contiguous, and the total number of hyperplane segments appearing in the decision surfaces is often fewer than $c(c-1)/2$, as shown in next figure.

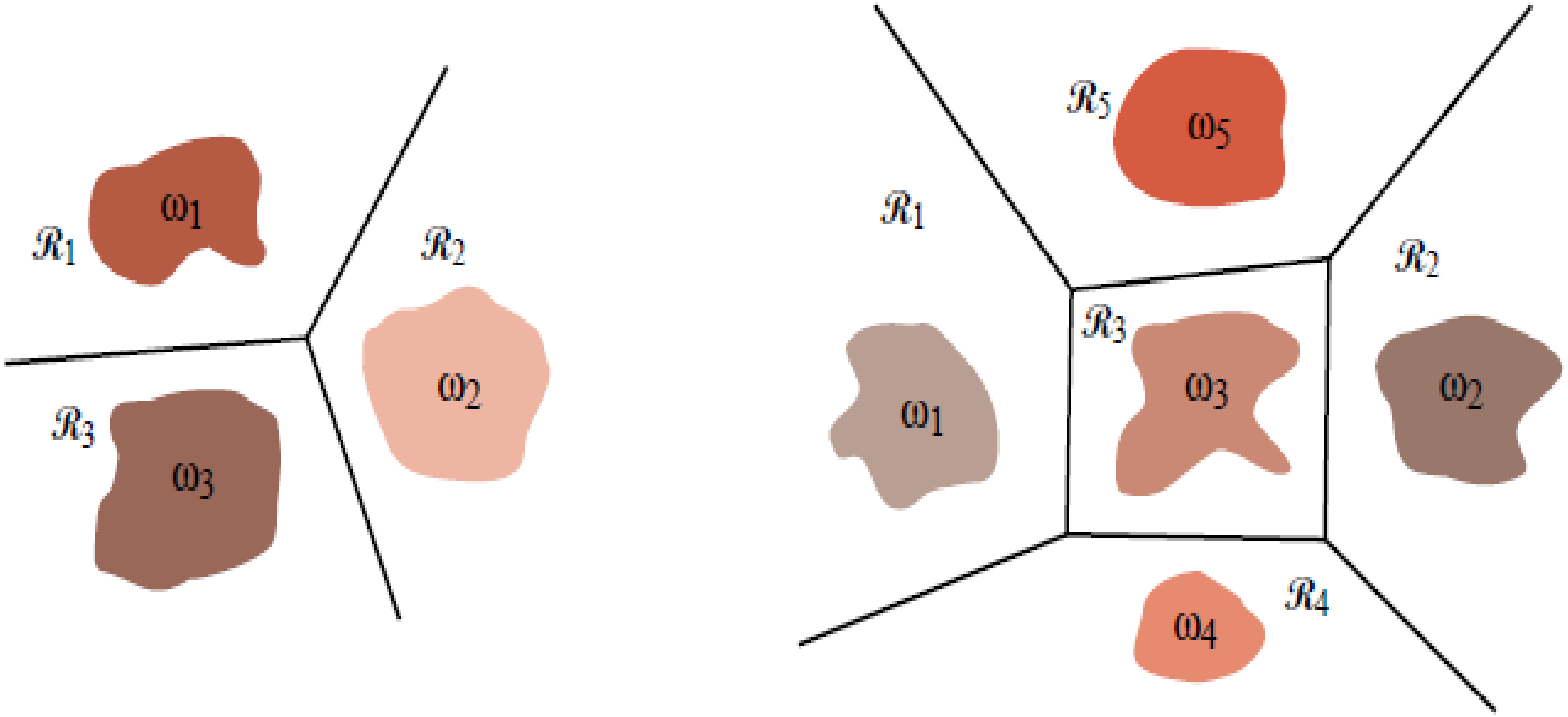
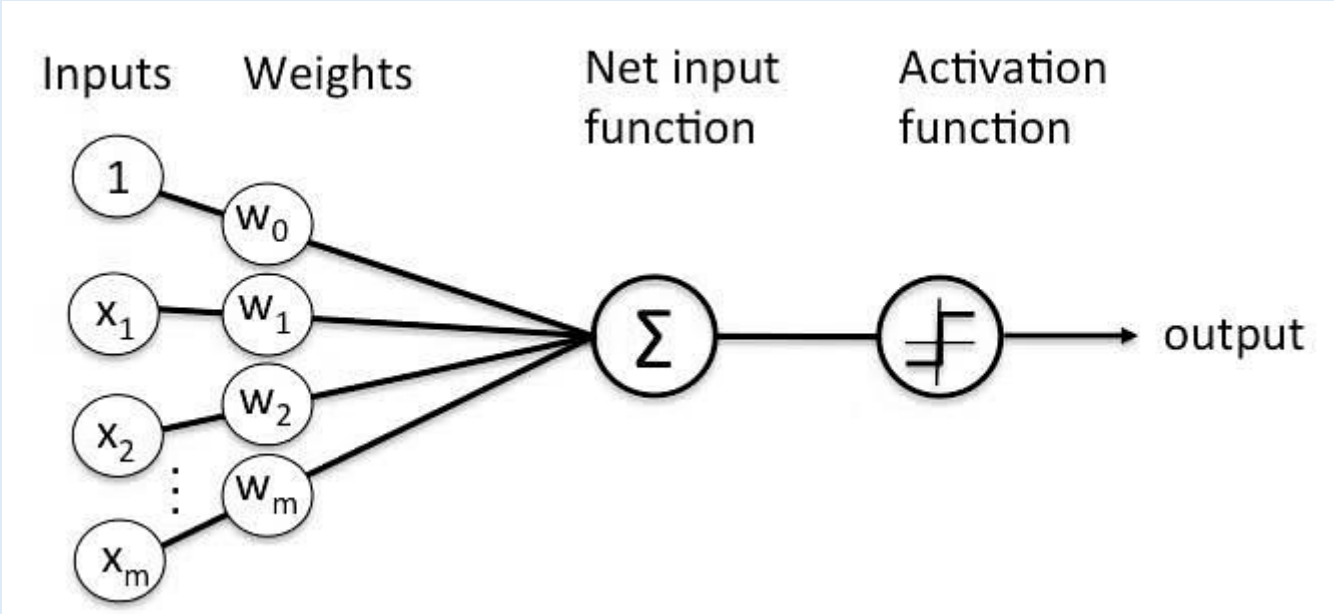


Figure 5.4: Decision boundaries produced by a linear machine for a three-class problem and a five-class problem.

Perceptron

Perceptron

- Perceptron was introduced by Frank Rosenblatt in 1957. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.

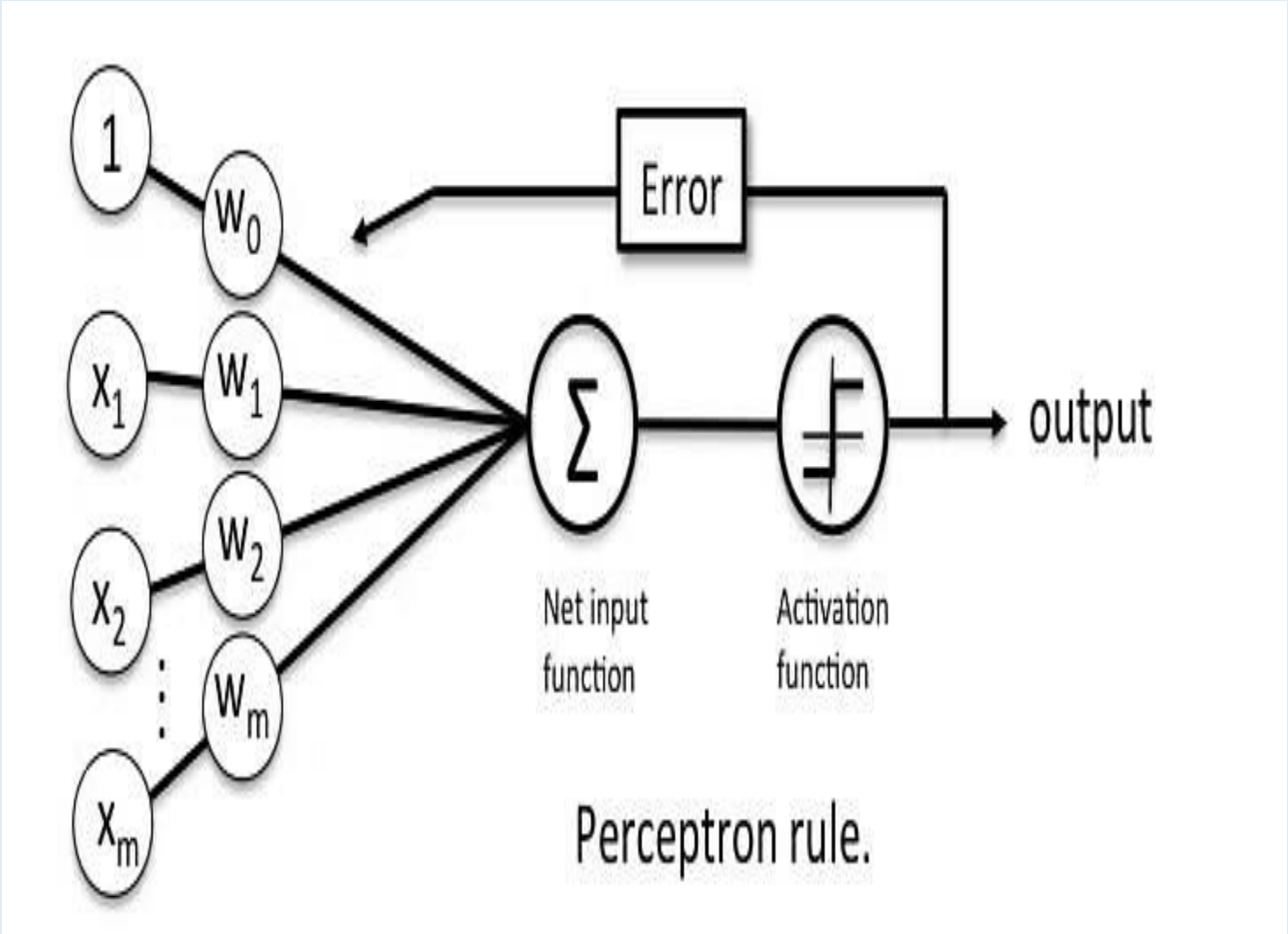


Perceptron

- There are two types of Perceptrons: Single layer and Multilayer.
  - ✓Single layer - Single layer perceptrons can learn only linearly separable patterns
  - ✓Multilayer - Multilayer perceptrons or feedforward neural networks with two or more layers have the greater processing power
- The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.
- This enables you to distinguish between the two linearly separable classes +1 and -1.

Perceptron Learning Rule

- Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.
- The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output. In the context of supervised learning and classification, this can then be used to predict the class of a sample.



Perceptron rule.

Perceptron Function

- Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value ”f(x)”is generated.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{i=1}^{m} w_i x_i$$

- In the equation given above:

“w” = vector of real-valued weights

“b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)

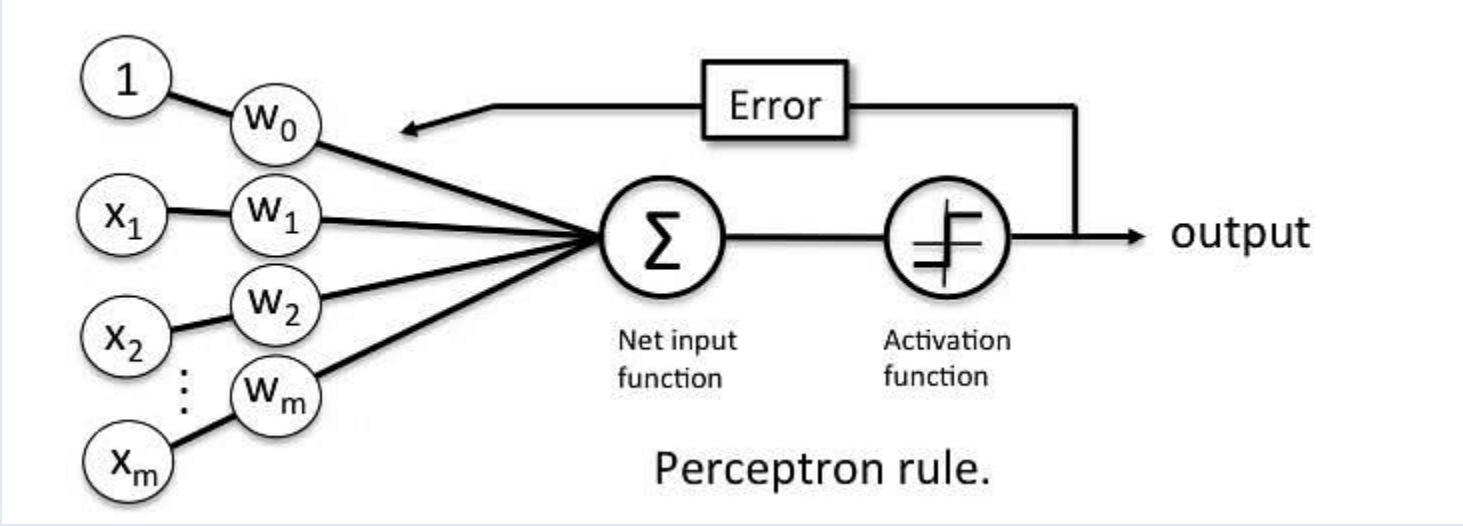
“x” = vector of input x values

“m” = number of inputs to the Perceptron

The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

Inputs of a Perceptron

- A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The image below shows a Perceptron with a Boolean output.



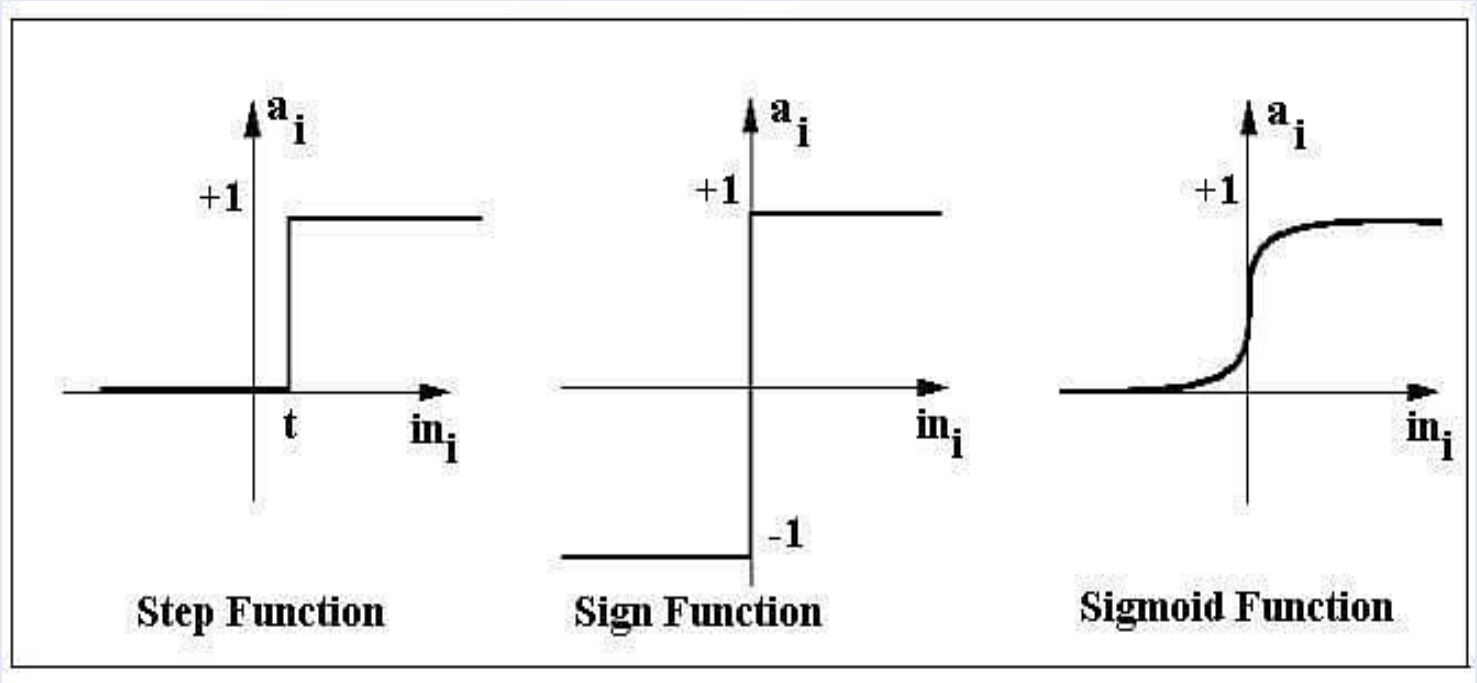
Perceptron Function

- A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function "∑" multiplies all inputs of "x" by weights "w" and then adds them up as follows:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Activation Functions of Perceptron

- The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.



Activation Functions of Perceptron

- For example:

  If $\sum w_i x_i > 0$ => then final output “o” = 1 (issue bank loan)

  Else, final output “o” = -1 (deny bank loan)

- Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

Output of Perceptron

- Perceptron with a Boolean output:
- Inputs: x1…xn
- Output: o(x1….xn)

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

- Weights: wi=> contribution of input xi to the Perceptron output;
- w0=> bias or threshold
- If $\sum w.x > 0$, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.

Output of Perceptron

- An output of +1 specifies that the neuron is triggered. An output of -1 specifies that the neuron did not get triggered.

- “sgn” stands for sign function with output +1 or -1.

Error in Perceptron

- In the Perceptron Learning Rule, the predicted output is compared with the known output. If it does not match, the error is propagated backward to allow weight adjustment to happen.

Perceptron: Decision Function

- A decision function φ(z) of Perceptron is defined to take a linear combination of x and w vectors.

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Perceptron: Decision Function

- The value z in the decision function is given by:

$$z = w_1 x_1 + \ldots + w_m x_m$$

- The decision function is +1 if z is greater than a threshold θ, and it is -1 otherwise.

$$\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

- This is the Perceptron algorithm.

Bias Unit

- For simplicity, the threshold θ can be brought to the left and represented as w0x0, where w0= -θ and x0= 1.

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

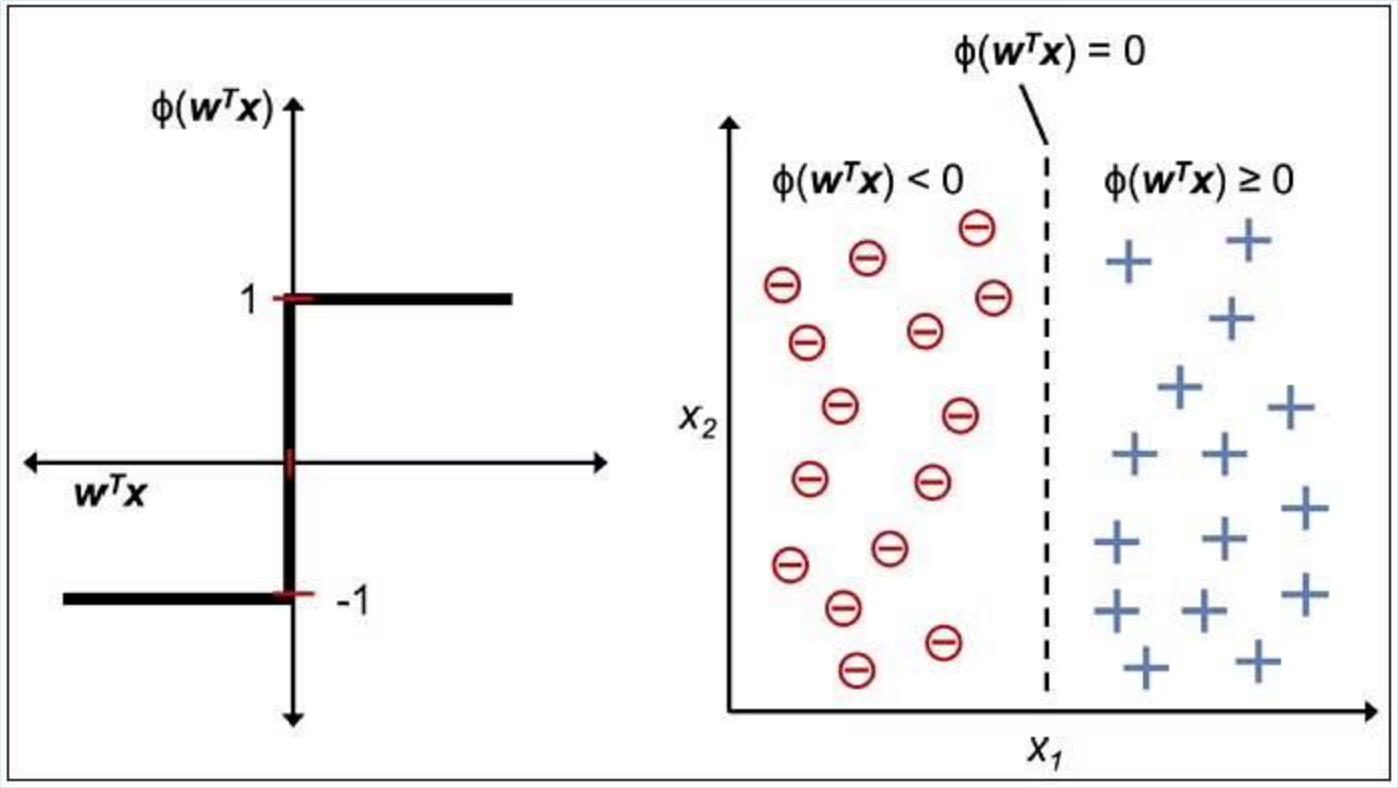
The value w0 is called the bias unit.

The decision function then becomes:

$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

Output:

- The figure shows how the decision function squashes wTx to either +1 or -1 and how it can be used to discriminate between two linearly separable classes.



Perceptron Criterion Function

- Consider the problem of constructing a criterion function for solving the linear inequalities $\mathbf{a}^t\mathbf{y}_i > 0$.
- The most obvious choice is to let $J(\mathbf{a}; \mathbf{y}_1, ..., \mathbf{y}_n)$ be the number of samples misclassified by **a**.
- However, because this function is piecewise constant, it is obviously a poor candidate for a gradient search.
- A better choice is the *Perceptron criterion function*

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t\mathbf{y}),$$

where $Y(\mathbf{a})$ is the set of samples *misclassified* by **a**. (If no samples are misclassified, Y is empty and we define J_p to be zero.)

Perceptron Criterion Function

- Since $\mathbf{a}^t\mathbf{y} \leq 0$ if $\mathbf{y}$ is misclassified, $J_p(\mathbf{a})$ is never negative, being zero only if $\mathbf{a}$ is a solution vector, or if $\mathbf{a}$ is on the decision boundary.
- Geometrically, $J_p(\mathbf{a})$ is proportional to the sum of the distances from the misclassified samples to the decision boundary.
- Figure next illustrates J_p for a simple two-dimensional example.

Y: upsilon

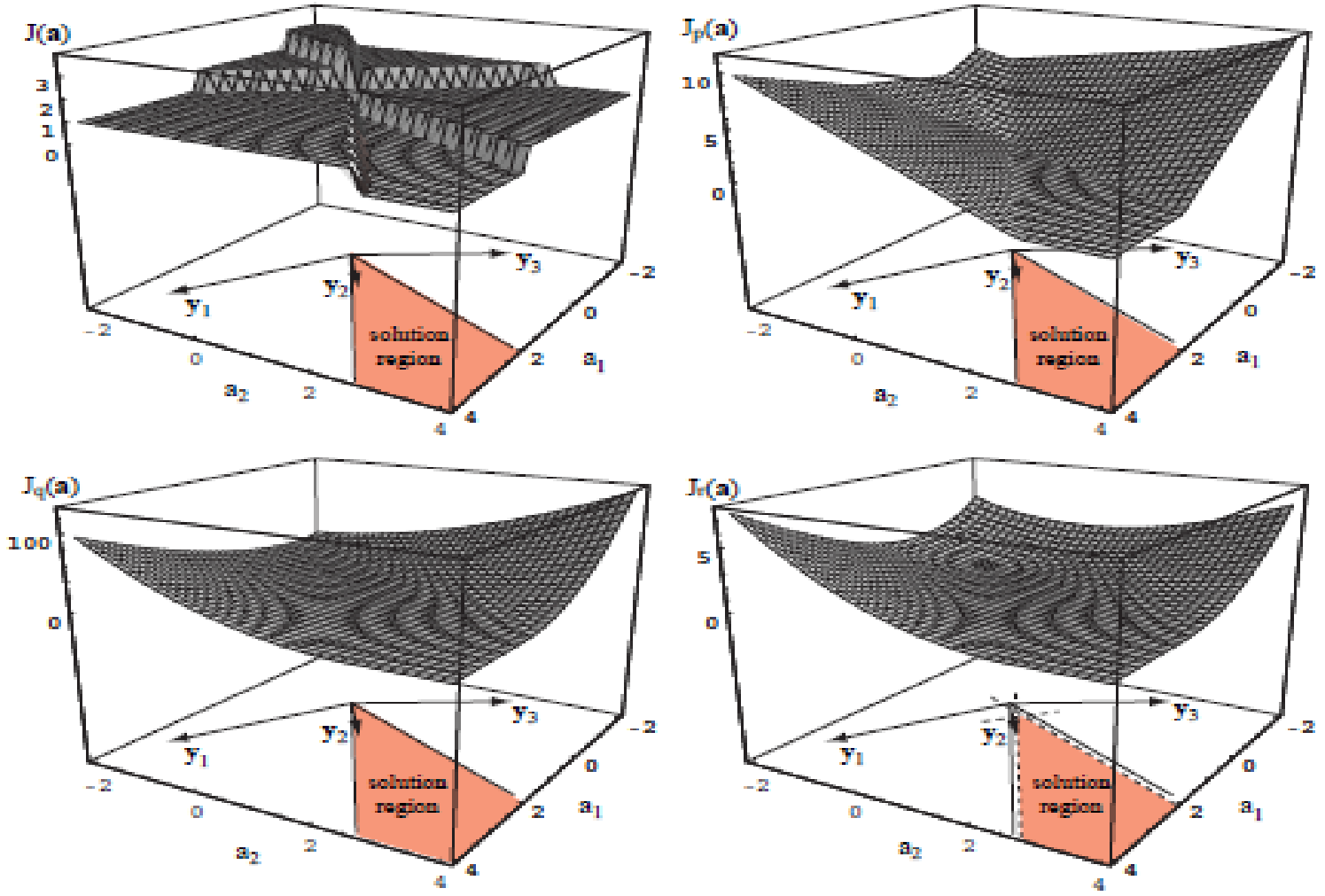


Figure 5.11: Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the Perceptron criterion (Eq. 16), which is piecewise linear and acceptable for gradient descent. The lower left is squared error (Eq. 32), which has nice analytic properties and is useful even when the patterns are not linearly separable. The lower right is the square error with margin (Eq. 33). A designer may adjust the margin b in order to force the solution vector to lie toward the middle of the $b = 0$ solution region in hopes of improving generalization of the resulting classifier.

Perceptron Criterion Function

- Since the j^{th} component of the gradient of J_p is $\partial J_p/\partial a_j$, we see from the earlier equation that

$$\nabla J_p = \sum_{\mathbf{y}\in\mathcal{Y}} (-\mathbf{y}),$$

and hence the update rule becomes

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum_{\mathbf{y}\in\mathcal{Y}_k} \mathbf{y},$$

where Y_k is the set of samples misclassified by **a**(k). Thus the Perceptron algorithm is:

nabla or ∇

Perceptron Criterion Function

Algorithm 3 (Batch Perceptron)

1 **begin initialize** $\mathbf{a}, \eta(\cdot)$, criterion $\theta, k = 0$
2 **do** $k \leftarrow k + 1$
3 $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$
4 **until** $\eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y} < \theta$
5 **return** a
6 **end**

Perceptron Criterion Function

- Thus, the batch Perceptron algorithm for finding a solution vector can be stated very simply : the next weight vector is obtained by adding some multiple of the sum batch of the misclassified samples to the present weight vector.
- We use the term “batch” training to refer to the fact that (in general) a large group of samples is used when computing each weight update.
- Figure next shows how this algorithm yields a solution vector for a simple two-dimensional example with $a(1) = 0$, and $\eta(k) = 1$.

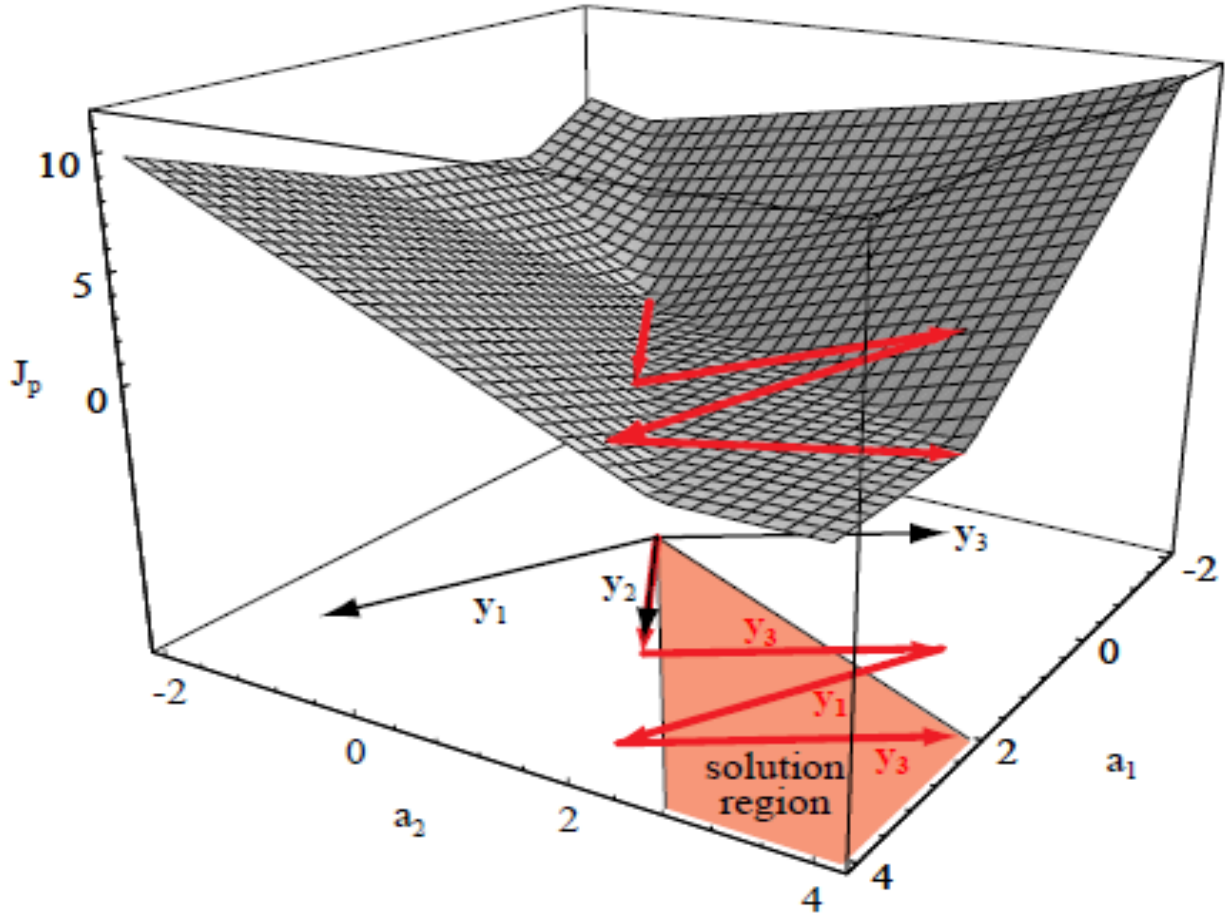


Figure 5.12: The Perceptron criterion, J_p is plotted as a function of the weights a_1 and a_2 for a three-pattern problem. The weight vector begins at $\mathbf{0}$, and the algorithm sequentially adds to it vectors equal to the "normalized" misclassified patterns themselves. In the example shown, this sequence is $\mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_1, \mathbf{y}_3$, at which time the vector lies in the solution region and iteration terminates. Note that the second update (by $\mathbf{y}_3$) takes the candidate vector *farther* from the solution region than after the first update (cf. Theorem 5.1. (In an alternate, batch method, *all* the misclassified points are added at each iteration step leading to a smoother trajectory in weight space.)

Support Vector Machine (SVM)

SVM

- *Support Vector Machines* rely on preprocessing the data to represent patterns in a high dimension — typically much higher than the original feature space.
- With an appropriate nonlinear mapping $\phi(.)$ to a sufficiently high dimension, data from two categories can always be separated by a hyperplane.
- Here we assume each pattern $\mathbf{x}_k$ has been transformed to $\mathbf{y}_k = \phi(\mathbf{x}_k)$; we return to the choice of $\phi(.)$ below.
- For each of the n patterns, k = 1, 2, ..., n, we let $z_k = \pm 1$, according to whether pattern k is in $\omega 1$ or $\omega 2$.
- A linear discriminant in an augmented **y** space is

$$g(\mathbf{y}) = \mathbf{a}^t\mathbf{y}, \qquad (104)$$

SVM

where both the weight vector and the transformed pattern vector are augmented (by $a_0 = w_0$ and $y_0 = 1$, respectively).

$$z_k g(\mathbf{y}_k) \geq 1 \quad k = 1, \ldots, n, \tag{105}$$

- The goal in training a Support Vector Machine is to find the separating hyperplane with the largest margin; we expect that the larger the margin, the better generalization of the classifier.
- As the distance from any hyperplane to a (transformed) pattern y is |g(y)|/||a||, and assuming that a positive margin b exists, Eq. 105 implies the goal is to find the weight vector **a** that maximizes *b*.

$$\frac{z_k g(\mathbf{y}_k)}{\|\mathbf{a}\|} \geq b \quad k = 1, \ldots, n; \tag{106}$$

SVM

- The solution vector can be scaled arbitrarily and still preserve the hyperplane, and thus to insure uniqueness we impose the constraint $b\,||\mathbf{a}|| = 1$; that is, we demand the solution to Eqs. 104 & 105 also minimize $||\mathbf{a}||^2$.
- The support vectors are the (transformed) training patterns for which Eq. 105 represents an equality — that is, the support vectors are (equally) close to the hyperplane (Fig. 5.19).
- The support vectors are the training samples that define the optimal separating hyperplane and are the most difficult patterns to classify.
- Informally speaking, they are the patterns most informative for the classification task.

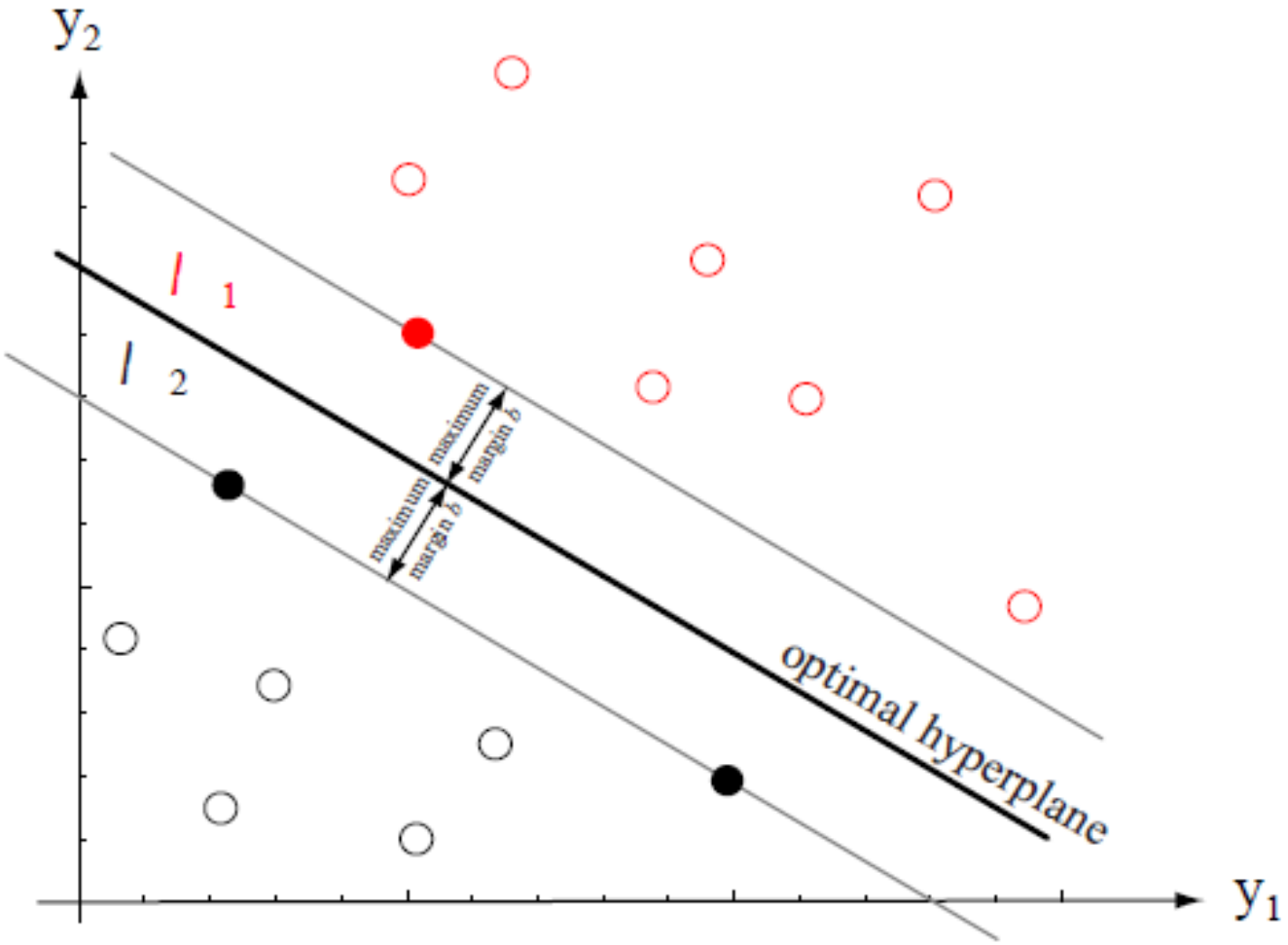


Figure 5.19: Training a Support Vector Machine consists of finding the optimal hyperplane, i.e., the one with the maximum distance from the nearest training patterns. The support vectors are those (nearest) patterns, a distance b from the hyperplane. The three support vectors are shown in solid dots.

SVM training

- The first step is, of course, to choose the nonlinear ϕ-functions that map the input to a higher dimensional space.
- Often this choice will be informed by the designer's knowledge of the problem domain.
- In the absence of such information, one might choose to use polynomials, Gaussians or yet other basis functions.
- The dimensionality of the mapped space can be arbitrarily high (though in practice it may be limited by computational resources).
- We begin by recasting the problem of minimizing the magnitude of the weight vector constrained by the separation into an unconstrained problem by the method of Lagrange undetermined multipliers.
- Thus from Eq. 106 and our goal of minimizing ||a||, we construct the functional

$$L(\mathbf{a}, \alpha) = \frac{1}{2}\|\mathbf{a}\|^2 - \sum_{k=1}^{n} \alpha_k [z_k \mathbf{a}^t \mathbf{y}_k - 1]. \qquad (108)$$

SVM training

and seek to minimize L() with respect to the weight vector **a**, and maximize it with respect to the undetermined multipliers $\alpha_k \geq 0$.

- The last term in Eq. 108 expresses the goal of classifying the points correctly.
- An important benefit of the Support Vector Machine approach is that the complexity of the resulting classifier is characterized by the number of support vectors — independent of the dimensionality of the transformed space.
- As a result, SVMs tend to be less prone to problems of overfitting than some other methods.

References

- Pattern Classification- Duda, Stork, Hart, Wiley Student Edition

End