3.4 Linear Least-Squares Filter

- Two characteristics of *linear least-squares filter:*

  1. The filter is built around a single linear neuron.
  2. The cost function is the sum of error squares, as defined in the Gauss-Newton method.

- The error of the input vector $\mathbf{x}(i)$ at time n is (section 3.2)

$$e(i) = d(i) - \mathbf{x}^T(i)\mathbf{w}(n)$$

- Collecting these equations into matrix-vector form yields

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)$$

  where $\mathbf{d}(n)$ is the $n \times 1$ *desired response vector:*

$$\mathbf{d}(n) = [d(1), d(2), \ldots, d(n)]^T$$

  and $\mathbf{X}(n)$ is the $n \times m$ *data matrix:*

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

- The Jacobian of the error vector $\mathbf{e}(n)$ with respect to the weight vector $\mathbf{w}(n)$ is simply

$$\mathbf{J}(n) = -\mathbf{X}(n)$$

- Substituting this into the Gauss-Newton iteration formula yields the linear least-squares filter

$$\mathbf{w}(n+1) = [\mathbf{X}^T(n)\mathbf{X}(n)]^{-1}\mathbf{X}^T(n)\mathbf{d}(n) = \mathbf{X}^+(n)\mathbf{d}(n)$$

  where

$$\mathbf{X}^+(n) = [\mathbf{X}^T(n)\mathbf{X}(n)]^{-1}\mathbf{X}^T(n)$$

  is the *pseudoinverse* of $\mathbf{X}(n)$.

- The $n \times m$ matrix $\mathbf{X}(n)$ has usually not a standard inverse.

Wiener Filter

- Assume now that the input vectors $\mathbf{x}(i)$ and the desired responses $d(i)$ are *stationary*.
- This means that their statistical properties do not depend on the time instant i.
- Then expectations (ensemble averages) can be computed from long-term sample or time averages.
- In particular, the following second-order statistics are used:
- The correlation matrix $\mathbf{R}_x = \mathsf{E}[\mathbf{x}(i)\mathbf{x}^T(i)]$ of the inputs $\mathbf{x}(i)$.
- The cross-correlation vector $\mathbf{r}_{xd} = \mathsf{E}[\mathbf{x}(i)d(i)]$ between the input vector $\mathbf{x}(i)$ and the desired response $d(i)$.
- Using the stationarity assumption, the correlation matrix can be com-

puted from the formula

$$\mathbf{R}_x = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}(i)\mathbf{x}^T(i) = \lim_{n\to\infty} \frac{1}{n} \mathbf{X}^T(n)\mathbf{X}(n)$$

- Similarly, the cross-correlation vector becomes

$$\mathbf{r}_{xd} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}(i)d(i) = \lim_{n\to\infty} \frac{1}{n} \mathbf{X}^T(n)\mathbf{d}(n)$$

- Using these formulas, the linear least-squares (pseudoinverse) solution to the filtering problem can be expressed in the form

$$\mathbf{w}_o = \lim_{n\to\infty} \mathbf{w}(n+1) = \mathbf{R}_x^{-1}\mathbf{r}_{xd}$$

- This is called the *Wiener filter*.

3.5 Least-Mean-Square Algorithm

- Wiener filter requires the knowledge or estimation of the correlation matrix $\mathbf{R}_x$ and the cross-correlation vector $\mathbf{r}_{xd}$.
- The least-mean-square (LMS) algorithm is a simple adaptive way to estimate the Wiener filter.
- It uses the *instantaneous values* of the cost function:

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}e^2(n)$$

  where $e(n)$ is the error signal at time n.

- One can easily derive the result

$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)e(n) = \hat{\mathbf{g}}(n)$$

  where $\hat{\mathbf{g}}(n)$ is the instantaneous estimate for the gradient vector at time n.

- Using this estimate for the gradient vector in the method of steepest descent yields the LMS algorithm

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta \mathbf{x}(n) e(n).$$

- LMS algorithm is a standard tool in adaptive signal processing.
- LMS algorithm is a *stochastic gradient algorithm*.
- After convergence, $\hat{\mathbf{w}}(n)$ fluctuates randomly around the correct Wiener solution.

Summary of LMS algorithm

- Training sample: Input signal vector $= \mathbf{x}(n)$
  Desired response $= d(n)$
- User selected parameter: η
- Initialization. Set $\hat{\mathbf{w}}(0) = 0$
- Computation. For $n = 1, 2, \ldots$ compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$
$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$$

- The LMS algorithm converges near the correct Wiener solution if the learning parameter satisfies the condition

$$0 < \eta < \frac{2}{\text{tr}[\mathbf{R}_x]}$$

- Here tr$[\mathbf{R}_x]$ denotes the trace of the correlation matrix $\mathbf{R}_x$.
- Generally, the trace of a square matrix is the sum of its diagonal elements.
- Therefore, tr$[\mathbf{R}_x]$ can be estimated easily by computing the average of $\mathbf{x}^T(i)\mathbf{x}(i)$, or squares of the inputs.

Properties of the LMS Algorithm

- Computationally simple.
- Model independent, leading to *robustness*.
- Robustness: small model uncertainties result in small estimation errors.
- It has recently been shown that the LMS algorithm minimizes the maximum possible estimation error.
- In practice, LMS algorithm is often used for tracking nonstationary variations in the input data.
- On the other hand, the LMS algorithm converges slowly.
- It is also sensitive to the condition number (eigenvalue spread) of the data correlation matrix $\mathbf{R}_x$.
- Typically the learning parameter of the LMS algorithm is chosen to be a suitable constant $\eta(n) = \eta_0$ for all n.

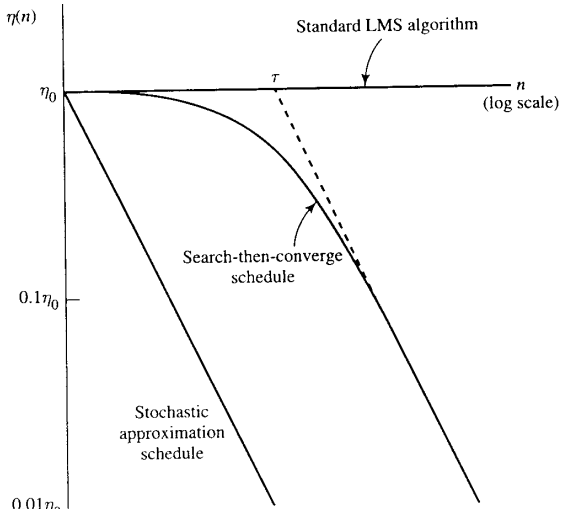
- In a stationary environment, a better strategy is to use a variable learning rate

$$\eta(n) = \frac{\eta_0}{1 + n/\tau}$$

- Using the *search time constant* τ allows the algorithm first get fairly close to the correct solution.

- After this, the decreasing $\eta(n)$ makes the algorithm converge closer to the optimal solution.

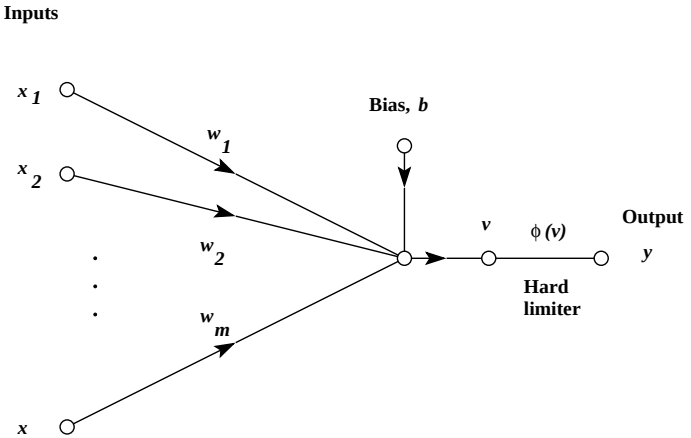
- A similar time-varying learning parameter $\eta(n)$ is useful also in other stochastic gradient algorithms.


$\eta(n)$
Standard LMS algorithm
η_0
τ
n
(log scale)
Search-then-converge
schedule
$0.1\eta_0$
Stochastic
approximation
schedule
$0.01\eta_0$

3.8 Perceptron

- Whereas the LMS algorithm uses a linear neuron, Rosenblatt's *perceptron* is built around a nonlinear neuron.
- It uses the classic *McCulloch-Pitts neuron model* discussed briefly in Chapter 1.
- This consists of a linear combiner followed by a hard limiter (signum function).
- The output of the neuron is thus either $+1$ or -1.
- The hard limiter input or induced local field of the neuron is

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



Signal-flow graph of the perceptron

- The goal of the perceptron is to correctly classify the set of inputs $x_1, x_2, \ldots, x_m$ into one of two classes, $\mathcal{C}_1$ or $\mathcal{C}_2$.

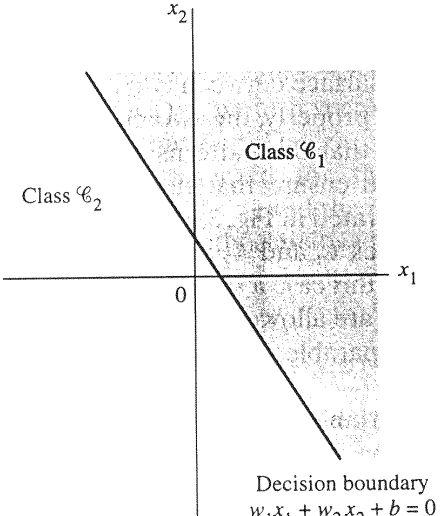
- The decision rule is: assign the point represented by the inputs $x_1, x_2, \ldots, x_m$ to:
  - class $\mathcal{C}_1$ if the perceptron output y is $+1$;
  - class $\mathcal{C}_2$ if the output y is -1.

- In the simplest form of the perceptron, there are two decision regions corresponding to classes $\mathcal{C}_1$ and $\mathcal{C}_2$.

- They are separated by a *hyperplane* defined by the equation

$$\sum_{i=1}^{m} w_i x_i + b = 0.$$

- An example for the case of two input variables x_1 and x_2.



- In this case, the hyperplane becomes a straight line.
- A nonzero bias merely shifts the decision boundary (hyperplane) away from the origin.

3.9 Perceptron Convergence Theorem

- In the following, an error-correction type learning algorithm is derived for the perceptron.
- It is more convenient to work with $(m+1)$-dimensional augmented input column vector

$$\mathbf{x}(n) = \ [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$$

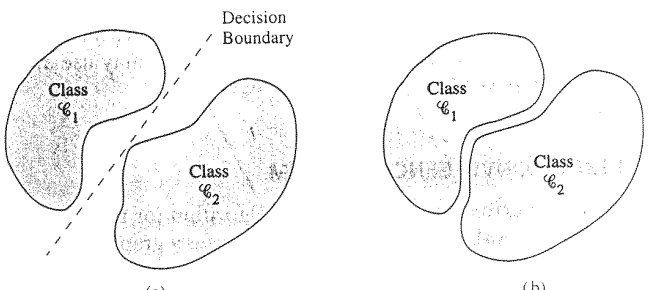
and the respective augmented weight vector

$$\mathbf{w}(n) = \ [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T.$$

- Here the bias $b(n)$ is treated as a synaptic weight $w_0(n)$ driven by a fixed input $x_0(n) = 1$.
- Then the linear combiner output (local field) is simply

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

- For fixed n, the equation $\mathbf{w}^T\mathbf{x} = 0$ defines a hyperplane.
- This is the decision surface between the two input classes.
- The perceptron functions properly if the two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ are *linearly separable* by some hyperplane.



- See figure; in case (b), there does not exist any separating hyperplane (a line in the 2-dimensional case shown here).

- Assume now that the input variables of the perceptron originate from two linearly separable classes.
- Denote by $\mathbf{x}_1(1), \mathbf{x}_1(2), \ldots$ the subset of training vectors belonging to the class $\mathcal{C}_1$.
- Respectively, the training vectors $\mathbf{x}_2(1), \mathbf{x}_2(2), \ldots$ come from the class $\mathcal{C}_2$.
- The total training set is the union of these sets.
- The perceptron learning algorithm adjusts the weight vector $\mathbf{w}$ until a separating hyperplane is found.
- After convergence, the weight vector $\mathbf{w}$ satisfies the conditions

$$\mathbf{w}^T\mathbf{x} > 0 \text{ for every input vector } \mathbf{x} \in \mathcal{C}_1$$

$$\mathbf{w}^T\mathbf{x} \leq 0 \text{ for every input vector } \mathbf{x} \in \mathcal{C}_2$$

- The algorithm for adapting the weight vector of the elementary perceptron may now be formulated as follows:

1. If the n-th training vector $\mathbf{x}(n)$ is correctly classified by the weight vector $\mathbf{w}(n)$ computed at the n-th iteration, no correction is made:

$$\mathbf{w}(n+1) = \mathbf{w}(n), \text{ if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \in \mathcal{C}_1$$

$$\mathbf{w}(n+1) = \mathbf{w}(n), \text{ if } \mathbf{w}^T\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \in \mathcal{C}_2$$

2. Otherwise, the weight vector $\mathbf{w}(n)$ is updated using the rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \in \mathcal{C}_2$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \in \mathcal{C}_1$$

Here again $\eta(n)$ is a positive learning-rate parameter.

- If $\eta(n) = \eta > 0$ is a constant, this learning algorithm is called a *fixed increment adaptation rule* for the perceptron.
- One can prove that the fixed increment rule *always converges* to a separating weight vector for linearly separable classes.
- The proof can be found in Haykin's book, pp. 139-141.

- The convergence takes place in a *finite* number of iterations.
- The separating weight vector is not unique.
- Another variant of perceptron learning algorithm: *absolute error correction procedure.*
- In this algorithm, the learning parameter $\eta(n)$ is variable.
- It is defined as the smallest integer for which

$$\eta(n) \parallel \mathbf{x}(n) \parallel^2 > \mid \mathbf{w}^T(n)\mathbf{x}(n) \mid$$

- Assume that the inner product $\mathbf{w}^T(n)\mathbf{x}(n)$ at iteration n has an incorrect sign.
- Using this procedure $\mathbf{w}^T(n+1)\mathbf{x}(n)$ at iteration $n+1$ would have a correct sign.
- The absolute error procedure can be realized using standard fixed increment adaptation.

- This is done by showing each pattern $\mathbf{x}(i)$ repeatedly to the perceptron until $\mathbf{x}(i)$ is classified correctly.

- The quantized response of the perceptron

$$y(n) = \text{ sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

  is written compactly using the signum function sgn(.)

- Recall that sgn$(v) = +1$, if $v > 0$; sgn$(v) = -1$, if $v < 0$.

- The *quantized desired response* is used:
  $d(n) = +1$ if $\mathbf{x}(n)$ belongs to the class $\mathcal{C}_1$;
  $d(n) = -1$ if $\mathbf{x}(n) \in \mathcal{C}_2$.

- This allows to write the adaptation algorithm compactly in the form of the error-correction learning rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)[d(n) - y(n)]\mathbf{x}(n)$$

- Here the learning parameter is a positive constant limited to the range $0 < \eta \leq 1$.

- A small learning parameter η provides stable weight vector estimates by averaging the input data.
- However, then the algorithm responds slowly to statistical variations in the input data.

Summary of the Perceptron Convergence Algorithm

Variables and parameters:

$$
\begin{aligned}
\mathbf{x}(n) &= (m+1)\text{-by-}1 \text{ input vector} \\
&= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T \\
\mathbf{w}(n) &= (m+1)\text{-by-}1 \text{ weight vector} \\
&= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T \\
b(n) &= \text{bias} \\
y(n) &= \text{actual response (quantized)} \\
d(n) &= \text{desired response} \\
\eta &= \text{learning-rate parameter, a positive constant less than unity}
\end{aligned}
$$

1. Initialization. Set $\mathbf{w}(0) = 0$. Then perform the following computations for time step $n = 1, 2, \ldots$

2. Activation. At time step n activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$

3. Computation of actual reponse.

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

   sgn$(\cdot)$ is the signum function.

4. Adaptation of weight vector.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

   where

$$d(n) = \begin{cases} +1 \text{ if } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_1 \\ -1 \text{ if } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_2 \end{cases}$$

5. Continuation. Increment time step n by one and go back to step 2.

3.10 Bayes Classifier and Perceptron

Bayes Classifier

- Bayes classifier is the classical *optimal* statistical classifier.
- It is discussed more thoroughly in our course "Principles of Pattern Recognition".
- Bayes classifier is needed also in neural computing, so we shall review it briefly in the following.
- It minimizes *the average risk* $\mathcal{R}$.
- Drawback: requires knowledge of the class distributions.
- This information is often not available in practice.

- For two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, the average risk is defined by

$$\begin{aligned}\mathcal{R} &= c_{11}p_1 \int_{\mathcal{H}_1} f(\mathbf{x} \mid \mathcal{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathcal{H}_2} f(\mathbf{x} \mid \mathcal{C}_2)d\mathbf{x} \\ &+ c_{21}p_1 \int_{\mathcal{H}_2} f(\mathbf{x} \mid \mathcal{C}_1)d\mathbf{x} + c_{12}p_2 \int_{\mathcal{H}_1} f(\mathbf{x} \mid \mathcal{C}_2)d\mathbf{x}\end{aligned}$$

- Here the various terms are defined as follows:
  - p_i is the a priori probability that the observation (pattern) vector $\mathbf{x}$ belongs to the class $\mathcal{C}_i$. The sum of class probabilities $p_1 + p_2 = 1$.
  - c_{ij} is the cost of the decision that $\mathbf{x}$ belongs to the class $\mathcal{C}_i$ when it actually comes from the class $\mathcal{C}_j$.
  - $f(\mathbf{x} \mid \mathcal{C}_i)$ is the conditional probability density of the vectors $\mathbf{x}$ belonging to the class $\mathcal{C}_i$.
  - $\mathcal{H}_i$ is a tentative decision region of the class $\mathcal{C}_i$. It is called also the *hypothesis* that the input vector $\mathbf{x}$ comes from the class $\mathcal{C}_i$.

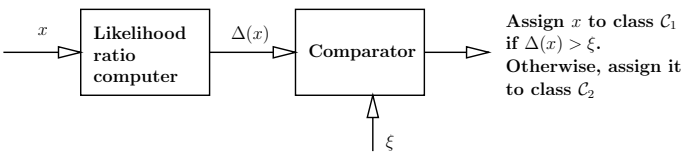
- The first two term of the risk $\mathcal{R}$ represent correct decisions (classifications).
- Note: they are often omitted by setting the costs of correct classifications to zero: $c_{11} = c_{22} = 0$.
- The last two terms of the risk $\mathcal{R}$ represent incorrect decisions (misclassifications).
- The task is now to minimize the average risk $\mathcal{R}$.
- This leads to the following **Bayes classification rule**:
- Define the *likelihood ratio* of the class densities

$$\Lambda(\mathbf{x}) = \frac{f(\mathbf{x} \mid \mathcal{C}_1)}{f(\mathbf{x} \mid \mathcal{C}_2)}$$

and the *threshold* of the Bayes classifier

$$\xi = \frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})}.$$

- *If $\Lambda(\mathbf{x}) > \xi$, assign the observation vector $\mathbf{x}$ to class $\mathcal{C}_1$. Otherwise, assign it to class $\mathcal{C}_2$.*

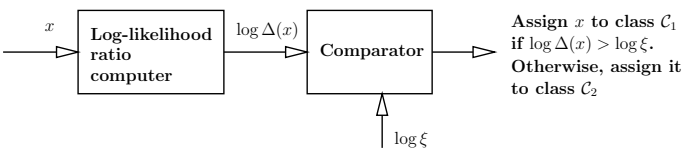


- Note that all the data processing is involved in the computation of the likelihood ratio $\Lambda(\mathbf{x})$.

- The prior probabilities and costs affect only the value of the threshold ξ.

- Note that both $\Lambda(\mathbf{x})$ and ξ are positive.

- Therefore, Bayes classifier may as well be implemented by first taking logarithms of both these quantities.
- This log-likelihood ratio is often more convenient in practice.



Bayes Classifier for a Gaussian Distribution

- Consider now the special case of a two-class problem where both classes have a *multivariate Gaussian distribution*:

$$f(\mathbf{x} \mid \mathcal{C}_i) = \kappa \exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T \mathbf{C}_i^{-1}(\mathbf{x}-\mathbf{m}_i)\right]$$

  The terms here have the following general meaning:

  - $\mathbf{m}_i = \mathsf{E}[\mathbf{x} \mid \mathcal{C}_i]$ is the mean vector of the class $\mathcal{C}_i$.
  - $\mathbf{C}_i = \mathsf{E}[(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^T \mid \mathcal{C}_i]$ is the covariance matrix of the class $\mathcal{C}_i$.
  - κ is a scaling constant which normalises the multivariate Gaussian distribution to have a unit probability mass.
    It is defined by

$$\kappa = \frac{1}{(2\pi)^{m/2}[\mathsf{det}\mathbf{C}_i]^{1/2}}$$

  - In κ, m is the dimensionality of the observation vector $\mathbf{x}$.

- In this specific classification problem, we make the following further assumption:
  - The classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the same covariance matrix $\mathbf{C}$.
  - $\mathbf{C}$ is nonsingular (its inverse exists) and nondiagonal (variables are correlated).
  - The classes $\mathcal{C}_1$ and $\mathcal{C}_2$ are equiprobable: $p_1 = p_2 = 1/2$.
  - Misclassifications have the same cost $c_{21} = c_{12}$, and correct classifications have zero costs $c_{11} = c_{22} = 0$.
- It is now fairly easy to derive the Bayes classifier by inserting these values into its formula.

- After some manipulations, one gets (Haykin pp. 146-147)

$$y = \mathbf{w}^T\mathbf{x} + b$$

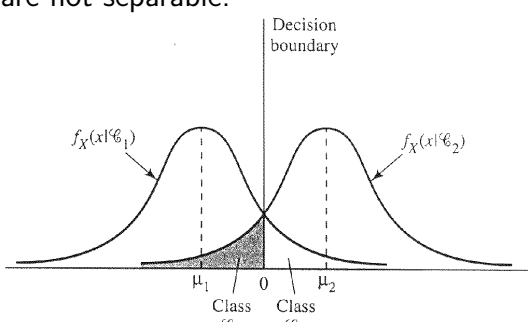
where:
- $y = \log\Lambda(\mathbf{x})$
- $\mathbf{w} = \mathbf{C}_i^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$
- b a constant given in Haykin's formula (3.87).

- The classification rule is simply: *if* $y > 0$, $\mathbf{x}$ *belongs to the class* $\mathcal{C}_1$, *otherwise to the class* $\mathcal{C}_2$.

- Thus the Bayes classifier reduces to a linear classifier in this Gaussian special case.

A Comparison with the Perceptron

- Perceptron is also a linear classifier.
- However, there are some important differences between the perceptron and the Bayes classifiers in Gaussian case.
- The perceptron assumes that the classes are linearly separable, while the Gaussian distributions of the two classes overlap each other and are not separable.



- The Bayes classifier minimizes the probability of the classification error.

- The shaded areas in the figure correspond to misclassifications.
- The perceptron convergence algorithm is *nonparametric*: it makes no assumptions on the underlying distributions.
- It may work well for non-Gaussian, heavily skewed input distributions.
- The Bayes classifier assumes Gaussian distributions (in the special case leading to the linear classifier), and is therefore *parametric.*
- The perceptron convergence algorithm is adaptive and simple to implement.

3.11 Summary and Discussion

- Both the perceptron and the LMS algorithms are based on error-correction learning.
- They can be used for a single neuron.
- The LMS algorithm uses a linear neuron.
- The perceptron uses a hard-limiting (signum) nonlinearity.
- The LMS algorithm is still used widely in adaptive signal processing.
- The perceptron algorithm is not used in practice currently.
- Reason: perceptron is not capable of making some global generalizations based on locally learned examples.
- Minsky and Papert (1969) showed this in their book.
- They conjectured that the same would hold also for multilayer perceptrons.

- This guess led to a decrease in interest in neural networks.
- Neural networks did not become important until in the mid-1980s.
- Both multilayer perceptrons (Chapter 4) and radial basis function networks (Chapter 5) overcome the limitations of the single-layer perceptron.