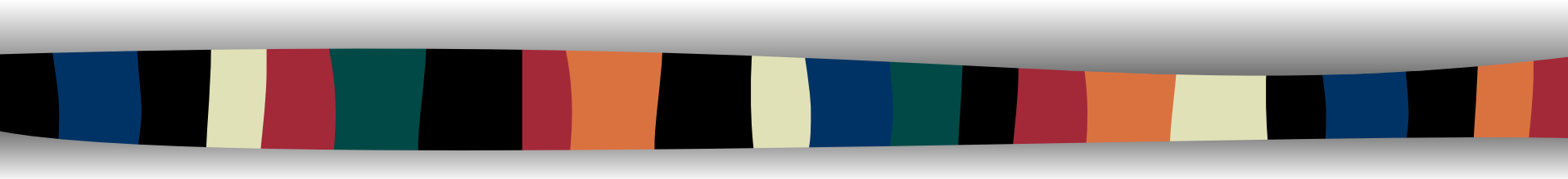
CS407 Neural Computation



Lecture 4:

Single Layer Perceptron (SLP) Classifiers

Lecturer: A/Prof. M. Bennamoun

Outline

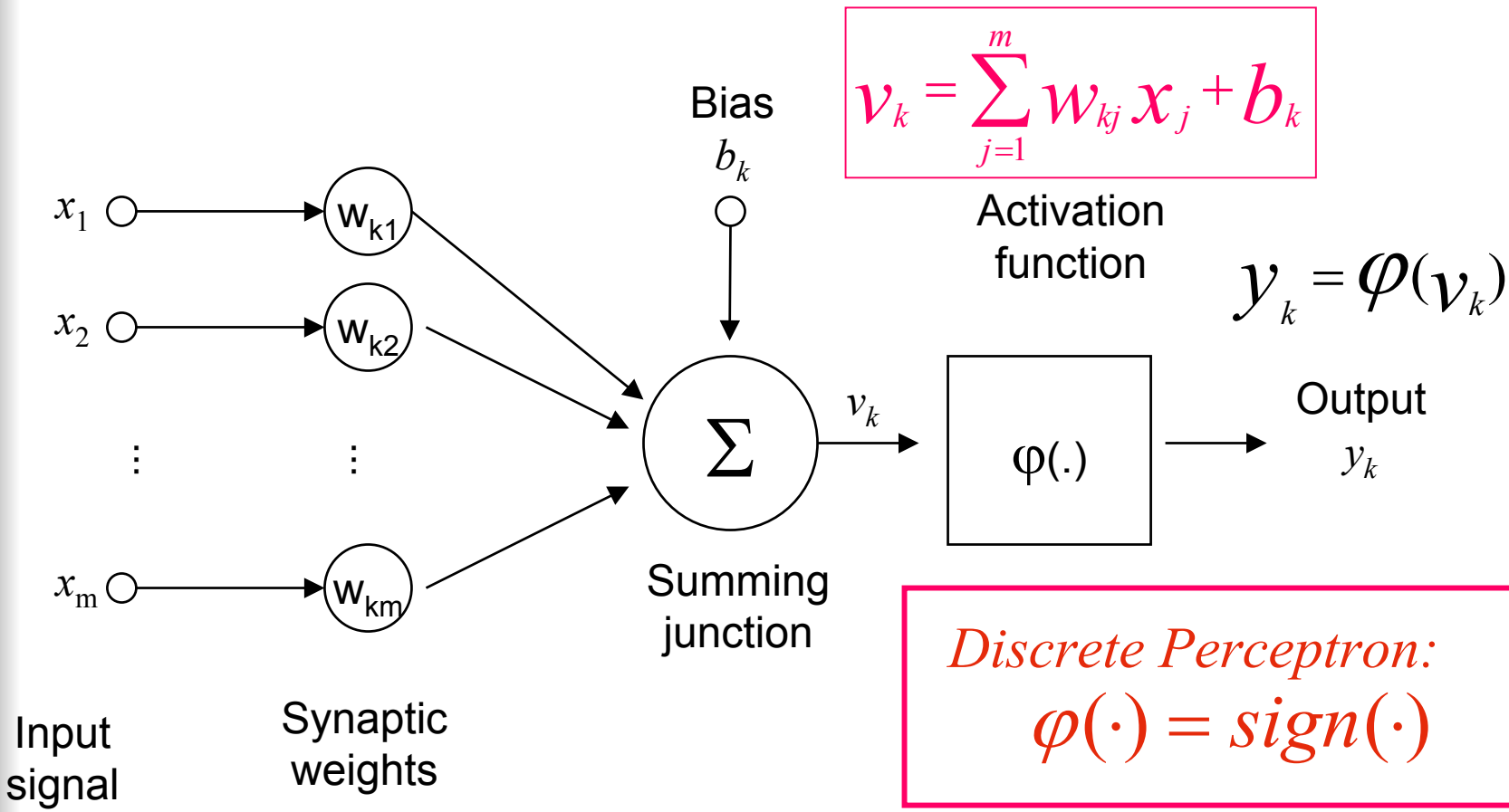
- What's a SLP and what's classification?
- Limitation of a single perceptron.
- Foundations of classification and Bayes Decision making theory
- Discriminant functions, linear machine and minimum distance classification
- Training and classification using the Discrete perceptron
- Single-Layer Continuous perceptron Networks for linearly separable classifications
- **Appendix A**: Unconstrained optimization techniques
- **Appendix B**: Perceptron Convergence proof
- Suggested reading and references

What is a perceptron and what is a Single Layer Perceptron (SLP)?

Perceptron

- The simplest form of a neural network
- consists of a single neuron with adjustable synaptic weights and bias
- performs pattern classification with only two classes
- perceptron convergence theorem :
  - Patterns (vectors) are drawn from two linearly separable classes
  - During training, the perceptron algorithm converges and positions the decision surface in the form of hyperplane between two classes by adjusting synaptic weights

What is a perceptron?



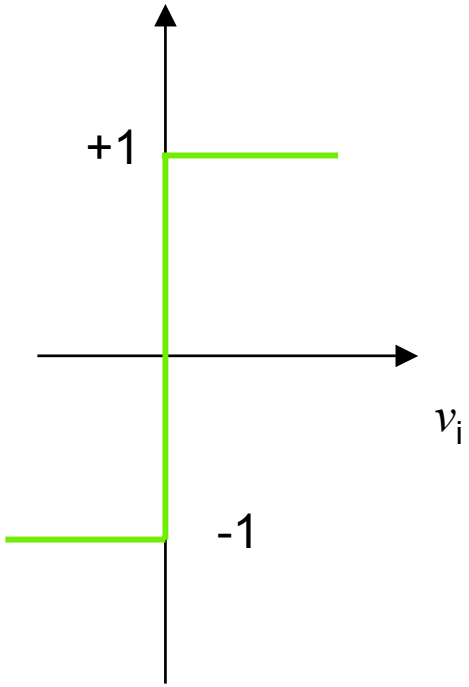
Discrete Perceptron:

$$\varphi(\cdot) = sign(\cdot)$$

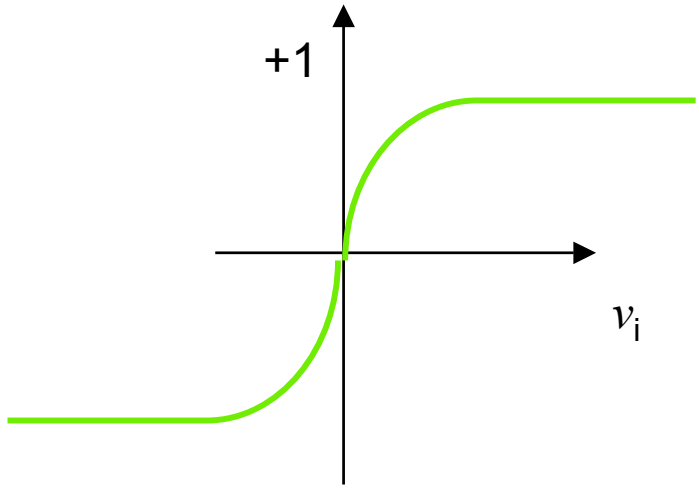
Continous Perceptron:

$$\varphi(\cdot) = S - shape$$

Activation Function of a perceptron



Signum Function
(sign)



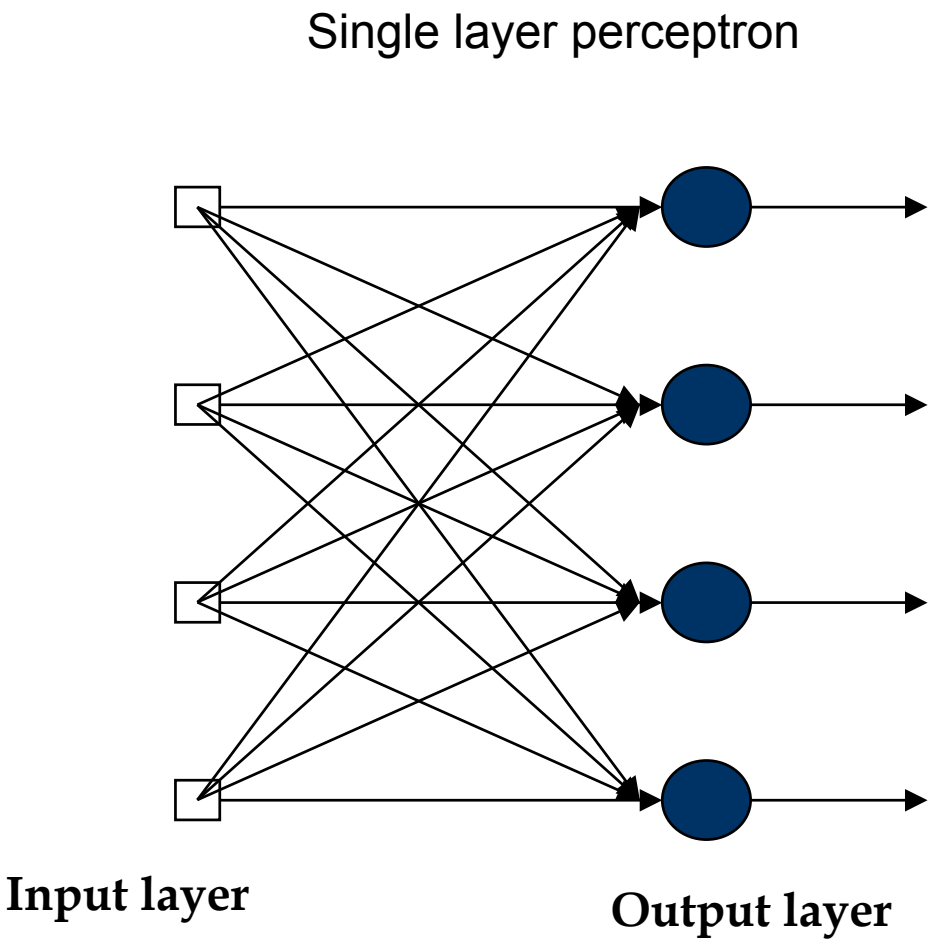
Discrete Perceptron:

$$\varphi(\cdot) = sign(\cdot)$$

Continous Perceptron:

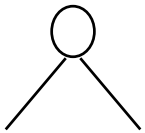
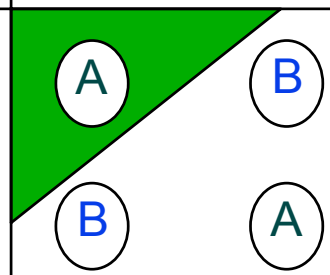
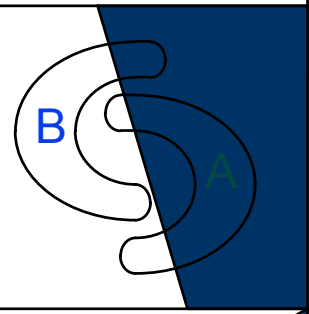
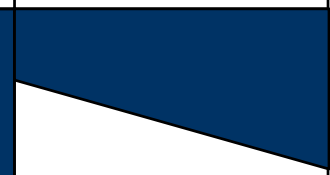
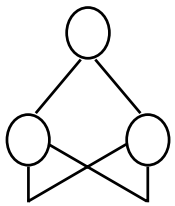
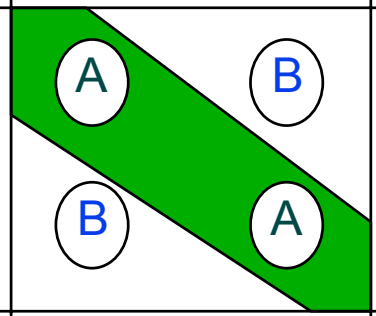
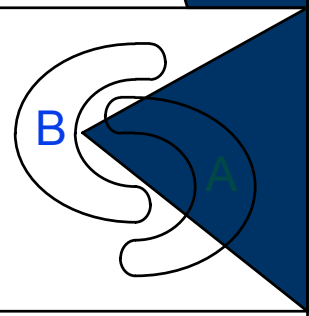
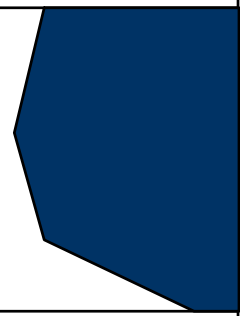
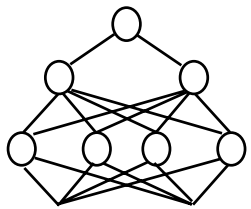
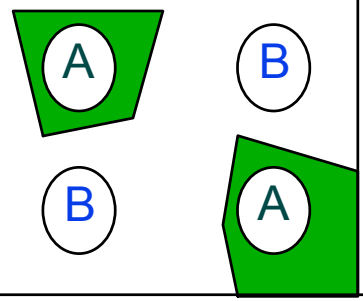
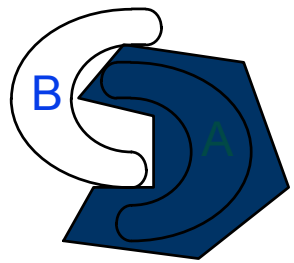
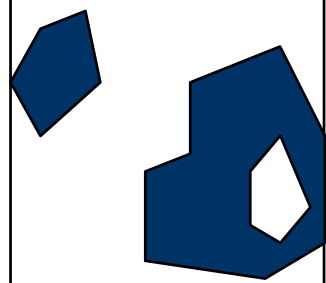
$$\varphi(v) = s - shape$$

SLP Architecture



Where are we heading? Different Non-Linearly Separable Problems

http://www.zsolutions.com/light.htm

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Review from last lectures:

Implementing Logic Gates with Perceptrons

http://www.cs.bham.ac.uk/~jxb/NN/l3.pdf

- We can use the perceptron to implement the basic logic gates (AND, OR and NOT).
- All we need to do is find the appropriate connection weights and neuron thresholds to produce the right outputs for each set of inputs.
- We saw how we can construct simple networks that perform NOT, AND, and OR.
- It is then a well known result from logic that we can construct **any** logical function from these three operations.
- The resulting networks, however, will usually have a much more complex architecture than a simple Perceptron.
- We generally want to avoid decomposing complex problems into simple logic gates, by finding the weights and thresholds that work directly in a Perceptron architecture.

Implementation of Logical NOT, AND, and OR

- In each case we have inputs in_i and outputs *out*, and need to determine the weights and thresholds. It is easy to find solutions by inspection:

NOT

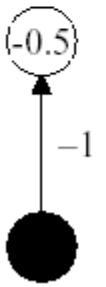
in	*out*
0	1
1	0

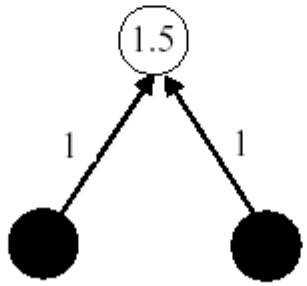
AND

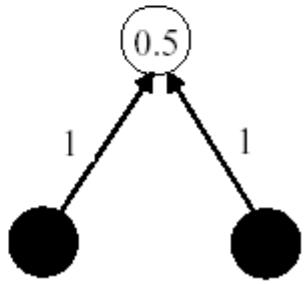
in_1	in_2	*out*
0	0	0
0	1	0
1	0	0
1	1	1

OR

in_1	in_2	*out*
0	0	0
0	1	1
1	0	1
1	1	1





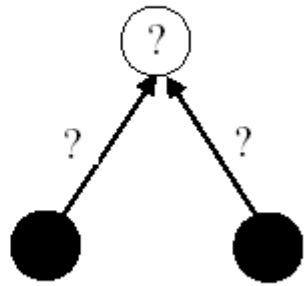


The Need to Find Weights Analytically

- Constructing simple networks by hand is one thing. But what about harder problems? For example, what about:

XOR

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0



- How long do we keep looking for a solution? We need to be able to calculate appropriate parameters rather than looking for solutions by trial and error.
- Each training pattern produces a linear inequality for the output in terms of the inputs and the network parameters. These can be used to compute the weights and thresholds.

Finding Weights Analytically for the AND Network

- We have two weights w_1 and w_2 and the threshold θ, and for each training pattern we need to satisfy

$$out = \mathrm{sgn}(w_1 in_1 + w_2 in_2 - \theta)$$

- So the training data lead to four inequalities:

in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$\Rightarrow$

$$\begin{aligned} w_1 0 + w_2 0 - \theta &< 0 \\ w_1 0 + w_2 1 - \theta &< 0 \\ w_1 1 + w_2 0 - \theta &< 0 \\ w_1 1 + w_2 1 - \theta &\geq 0 \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} \theta &> 0 \\ w_2 &< \theta \\ w_1 &< \theta \\ w_1 + w_2 &\geq \theta \end{aligned}$$

- It is easy to see that there are an infinite number of solutions. Similarly, there are an infinite number of solutions for the NOT and OR networks.

Limitations of Simple Perceptrons

- We can follow the same procedure for the XOR network:

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

$\Rightarrow$

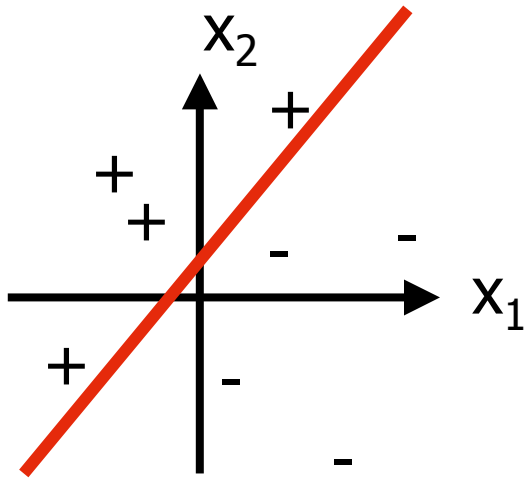
$$
\begin{aligned}
w_1 0 + w_2 0 - \theta &< 0 \\
w_1 0 + w_2 1 - \theta &\geq 0 \\
w_1 1 + w_2 0 - \theta &\geq 0 \\
w_1 1 + w_2 1 - \theta &< 0
\end{aligned}
$$

$\Rightarrow$

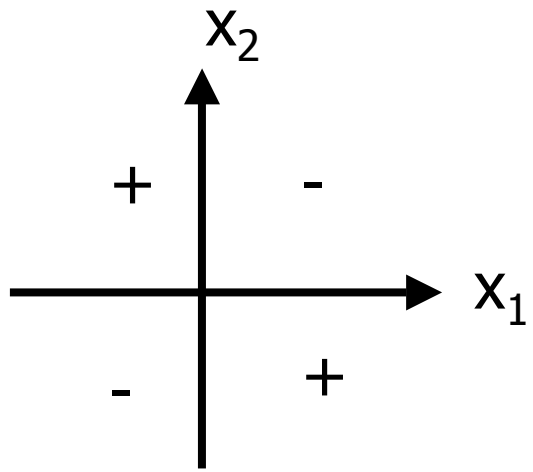
$$
\begin{aligned}
\theta &> 0 \\
w_2 &\geq \theta \\
w_1 &\geq \theta \\
w_1 + w_2 &< \theta
\end{aligned}
$$

- Clearly the second and third inequalities are incompatible with the fourth, so there is in fact no solution. We need more complex networks, e.g. that combine together many simple networks, or use different activation/thresholding/transfer functions.
- It then becomes much more difficult to determine all the weights and thresholds by hand.
- These weights instead are adapted using learning rules. Hence, need to consider learning rules (see previous lecture), and more complex architectures.

E.g. Decision Surface of a Perceptron



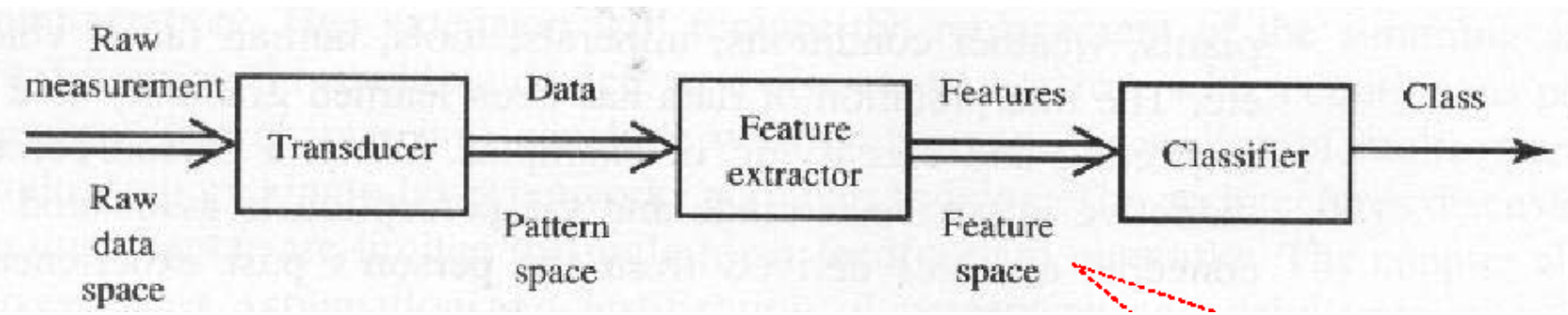
Linearly separable



Non-Linearly separable

- Perceptron is able to represent some useful functions
- But functions that are not linearly separable (e.g. XOR) are not representable

What is classification?



Classification ?

http://140.122.185.120

- Pattern classification/recognition

  - Assign the input data (a physical object, event, or phenomenon) to one of the pre-specified classes (categories)

- The block diagram of the recognition and classification system

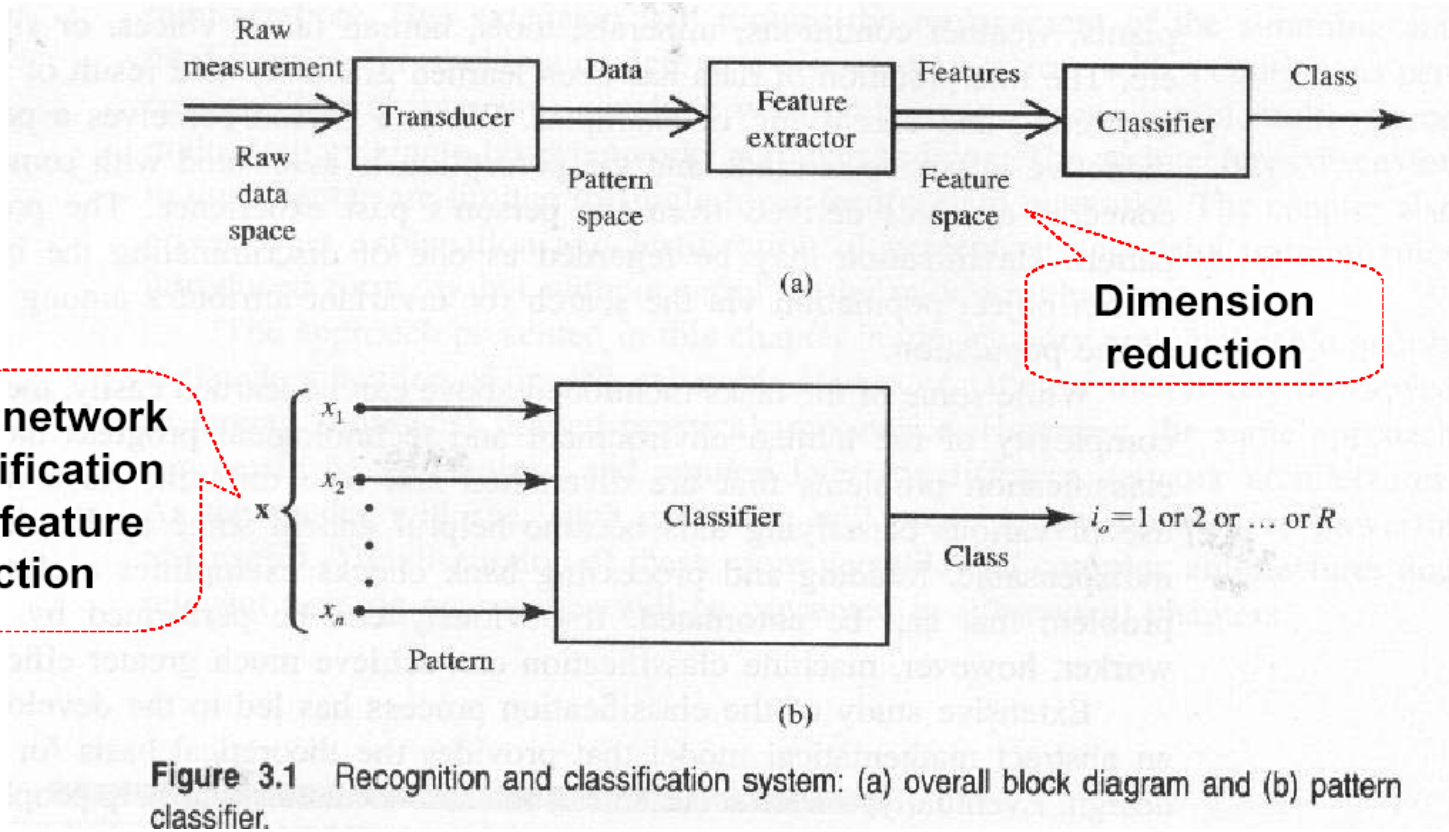


Figure 3.1 Recognition and classification system: (a) overall block diagram and (b) pattern classifier.

Classification: an example

http://webcourse.technion.ac.il/236607/Winter2002-2003/en/ho.htm
Duda & Hart, Chapter 1

- **Automate the process of sorting incoming fish on a conveyor belt according to species (Salmon or Sea bass).**
  - Set up a camera
  - Take some sample images
  - Note the physical differences between the two types of fish
    - Length
    - Lightness
    - Width
    - No. & shape of fins (“sanfirim”)
    - Position of the mouth

Classification an example…

Classification: an example…

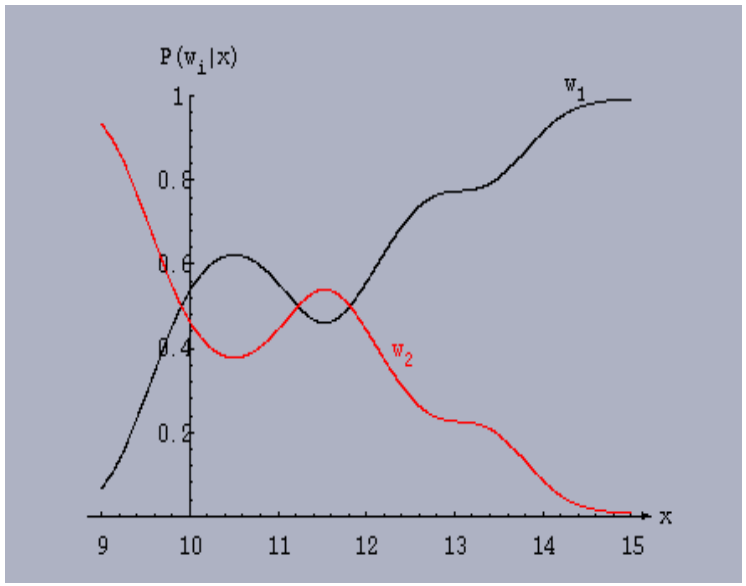
- Cost of misclassification: depends on application

  Is it better to misclassify salmon as bass or vice versa?

  - Put salmon in a can of bass $\Rightarrow$ loose profit
  - Put bass in a can of salmon $\Rightarrow$ loose customer

- There is a cost associated with our decision.
- Make a decision to minimize a given cost.
- **Feature Extraction:**
  - *Problem & Domain dependent*
  - *Requires knowledge of the domain*
  - *A good feature extractor would make the job of the classifier trivial.*

Bayesian decision theory



Bayesian Decision Theory

http://webcourse.technion.ac.il/236607/Winter2002-2003/en/ho.html
Duda & Hart, Chapter 2

- Bayesian decision theory is a fundamental statistical approach to the problem of pattern classification.
  - Decision making when all the probabilistic information is known.
  - For given probabilities the decision is optimal.
  - When new information is added, it is assimilated in optimal fashion for improvement of decisions.

Bayesian Decision Theory …

- Fish Example:
- Each fish is in one of 2 states: sea bass or salmon
- Let ω denote the ***state of nature***
  - $\omega = \omega_1$ for sea bass
  - $\omega = \omega_2$ for salmon

Bayesian Decision Theory …

- The State of nature is unpredictable $\Rightarrow$ ω is a variable that must be described probabilistically.
- If the catch produced as much salmon as sea bass the next fish is equally likely to be sea bass or salmon.
- Define
  - $P(\omega_1)$: a priori probability that the next fish is sea bass
  - $P(\omega_2)$: a priori probability that the next fish is salmon.

Bayesian Decision Theory …

- If other types of fish are irrelevant:

$$P(\omega_1) + P(\omega_2) = 1.$$

- *Prior probabilities reflect our prior knowledge (e.g. time of year, fishing area, …)*
- *Simple decision Rule:*
  - *Make a decision without seeing the fish.*
  - *Decide w_1 if $P(\omega_1) > P(\omega_2)$; ω_2 otherwise.*
  - *OK if deciding for one fish*
  - *If several fish, all assigned to same class.*

Bayesian Decision Theory ...

- In general, we will have some features and more information.
- Feature: lightness measurement = x
  - *Different fish yield different lightness readings (x is a random variable)*

Bayesian Decision Theory ….

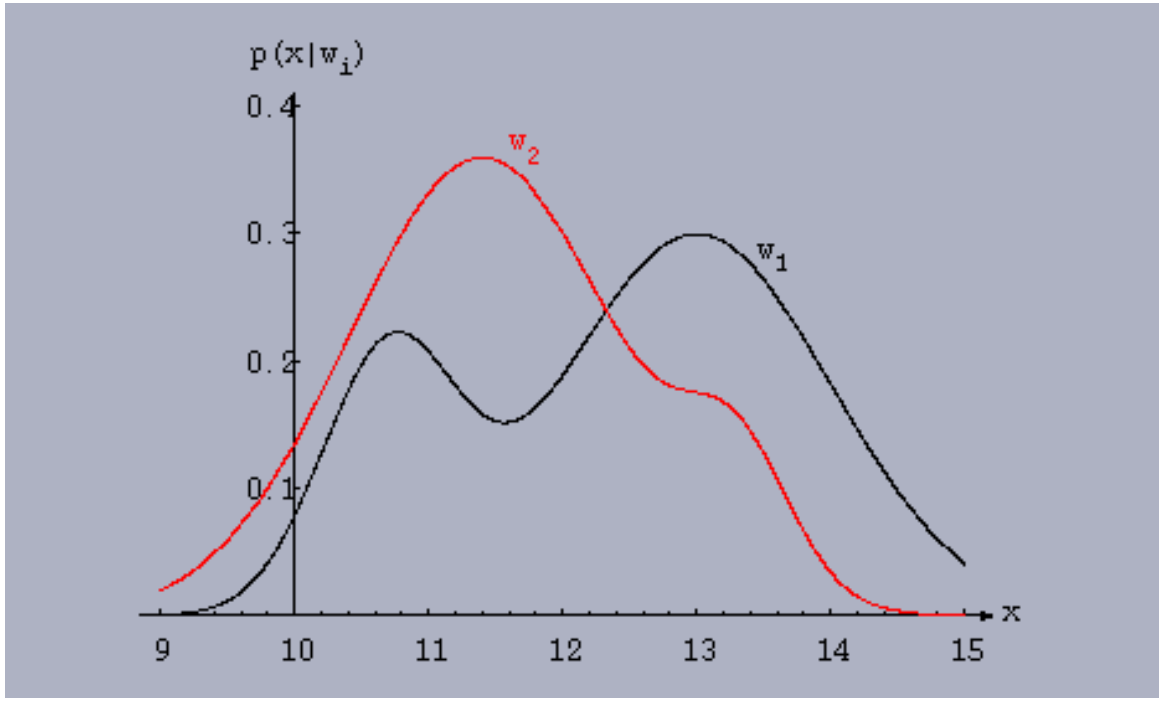
- Define

  $p(x|\omega_1)$ = Class Conditional Probability Density

  Probability density function for x given that the state of nature is ω_1

- The difference between $p(x|\omega_1)$ and $p(x|\omega_2)$ describes the difference in lightness between sea bass and salmon.

Class conditioned probability density: p(x|ω)



Hypothetical class-conditional probability
Density functions are normalized (area under each curve is 1.0)

Bayesian Decision Theory ...

- Suppose that we know

  The prior probabilities $P(\omega_1)$ and $P(\omega_2)$,

  The conditional densities $p(x \mid \omega_1)$ and $p(x \mid \omega_2)$

  Measure lightness of a fish = x.

- What is the category of the fish $p(\omega_j \mid x)$?

Bayes Formula

- Given
  - Prior probabilities $P(\omega_j)$
  - Conditional probabilities $p(x|\ \omega_j)$
- Measurement of particular item
  - Feature value x
- Bayes formula:

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)}$$

$$\text{Posterior} = \frac{\text{Likelihood} * \text{Prior}}{\text{Evidence}}$$

(from $p(\omega_j, x) = p(x|\omega_j)P(\omega_j) = P(\omega_j|x)p(x)$)

where $\sum_i P(\omega_i|x) = 1$

so $p(x) = \sum_i p(x|\omega_i)P(\omega_i)$

Bayes' formula ...

- $p(x|\omega_j)$ is called the ***likelihood*** of ω_j with respect to ***x.***

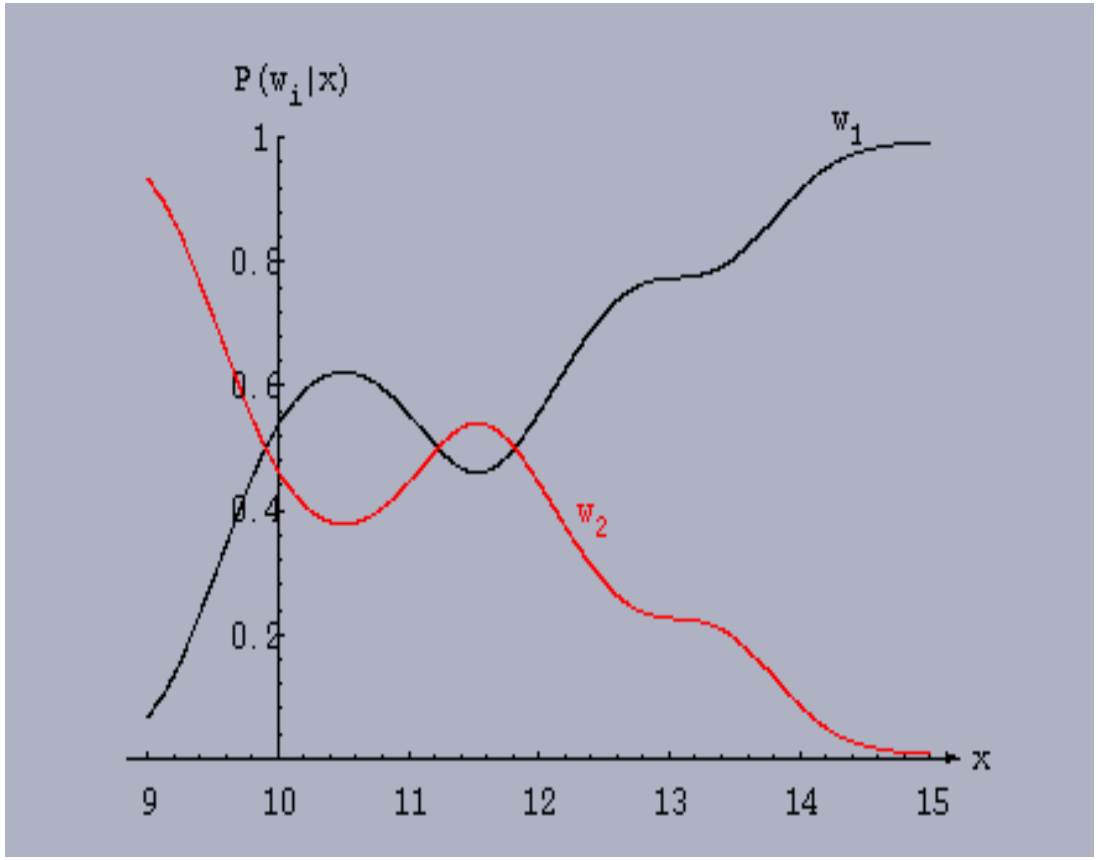
  (the ω_j category for which $p(x|\omega_j)$ is large is more "likely" to be the true category)

- p(x) is the ***evidence***

  how frequently we will measure a pattern with feature value x.

  Scale factor that guarantees that the posterior probabilities sum to 1.

Posterior Probability



Posterior probabilities for the particular priors P(ω_1)=2/3 and P(ω_2)=1/3. At every x the posteriors sum to 1.

Error

$$P(error \mid x) = \begin{cases} \text{If we decide } \omega_2 \Rightarrow P(\omega_1 \mid x) \\ \text{If we decide } \omega_1 \Rightarrow P(\omega_2 \mid x) \end{cases}$$

For a given *x*, we can minimize the probability of error by deciding ω_1 *if* $P(\omega_1|x) > P(\omega_2|x)$ *and* ω_2 *otherwise.*

Bayes' Decision Rule
(Minimizes the probability of error)

ω_1 : *if* $P(\omega_1|x) > P(\omega_2|x)$ *i.e.* $P(\omega_1|\underline{x}) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} P(\omega_2|\underline{x})$

ω_2 : *otherwise*

or

ω_1 : *if* $P(x|\omega_1)\,P(\omega_1) > P(x|\omega_2)\,P(\omega_2)$

ω_2 : *otherwise*

$$p(\underline{x}\mid\omega_1)P(\omega_1) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} p(\underline{x}\mid\omega_2)P(\omega_2) \Leftrightarrow \underbrace{\frac{p(\underline{x}\mid\omega_1)}{p(\underline{x}\mid\omega_2)}}_{\text{Likelihood ratio}} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \underbrace{\frac{P(\omega_1)}{P(\omega_2)}}_{\text{Threshold}}$$

and

$P(Error|x) = \min [P(\omega_1|x)\,,\, P(\omega_2|x)]$

Decision Boundaries

- Classification as division of feature space into non-overlapping regions

$$X_1, \ldots, X_R \ such \ that$$

$$\underline{x} \in X_k \leftrightarrow \underline{x} \ assigned \ to \ \omega_k$$

- Boundaries between these regions are known as **decision surfaces** or **decision boundaries**

Optimum decision boundaries

- Criterion:
  - minimize miss-classification
  - Maximize correct-classification

$$P(correct) = \sum_{k=1}^{R} P(\underline{x} \in X_k, \omega_k)$$

$$= \sum_{k=1}^{R} P(\underline{x} \in X_k | \omega_k) P(\omega_k)$$

$$Here \quad R = 2$$

$$Classify \quad \underline{x} \in X_k \quad if \quad \forall j \neq k$$

$$p(\underline{x}|\omega_k) P(\omega_k) > p(\underline{x}|\omega_j) P(\omega_j)$$

$$i.e.$$

maximum posterior probabilit y

$$\forall j \neq k \quad P(\omega_k|\underline{x}) > P(\omega_j|\underline{x})$$

Discriminant functions

- Discriminant functions determine classification by comparison of their values:

$$Classify \quad \underline{x} \in X_k \quad if$$

$$\forall j \neq k \quad g_k(\underline{x}) > g_j(\underline{x})$$

- Optimum classification: based on posterior probability $P(\omega_k|\underline{x})$
- Any monotone function g may be applied without changing the decision boundaries

$$g_k(\underline{x}) = g(P(\omega_k|\underline{x}))$$

$$e.g. \qquad g_k(\underline{x}) = \ln(P(\omega_k|\underline{x}))$$

The Two-Category Case

- Use 2 discriminant functions g_1 and g_2, and assigning $\mathbf{x}$ to ω_1 if $g_1 > g_2$.
- Alternative: define a single discriminant function $g(x) = g_1(x) - g_2(x)$, decide ω_1 if $g(x) > 0$, otherwise decide ω_2.
- Two category case

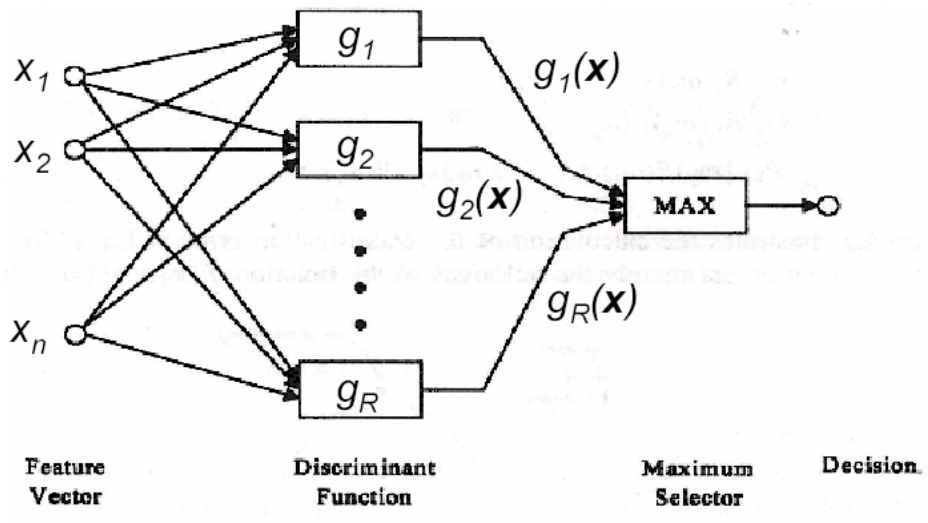
$$g(\mathbf{x}) = P(\omega_1 \mid \mathbf{x}) - P(\omega_2 \mid \mathbf{x})$$

$$g(\mathbf{x}) = \ln \frac{p(\mathbf{x} \mid \omega_1)}{p(\mathbf{x} \mid \omega_2)} + \ln \frac{P(\omega_1)}{P(\omega_2)}$$

Summary

- Bayes approach:
  - Estimate class-conditioned probability density
  - Combine with prior class probability
  - Determine posterior class probability
  - Derive decision boundaries
- Alternate approach implemented by NN
  - Estimate posterior probability directly
  - i.e. determine decision boundaries directly

DISCRIMINANT FUNCTIONS



Discriminant Functions

http://140.122.185.120

- Determine the membership in a category by the classifier based on the comparison of R **discriminant functions** $g^1(\boldsymbol{x}), g^2(\boldsymbol{x}),\ldots, g^R(\boldsymbol{x})$

– When $\boldsymbol{x}$ is within the region $\boldsymbol{X}_k$ if $g_k(\boldsymbol{x})$ has the largest value $i_0(\boldsymbol{x}) = k \text{ if } g_k(\boldsymbol{x}) > g_j(\boldsymbol{x}) \text{ for } k,\ j = 1,\ 2,\ldots,\ R,\ k \neq j$

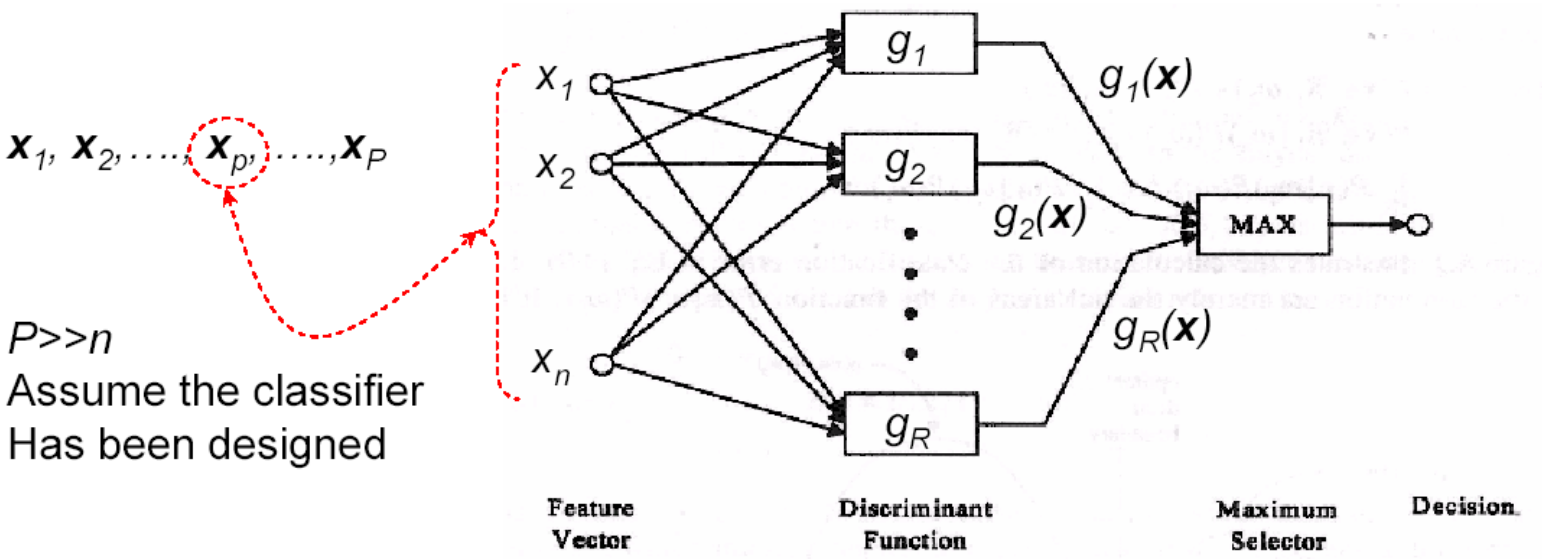


Figure 4.2 Block diagram of a classifier based on discriminant functions [22].

Do not mix between n = dim of each I/P vector (dim of feature space); P= # of I/P vectors; and R= # of classes.

Discriminant Functions…

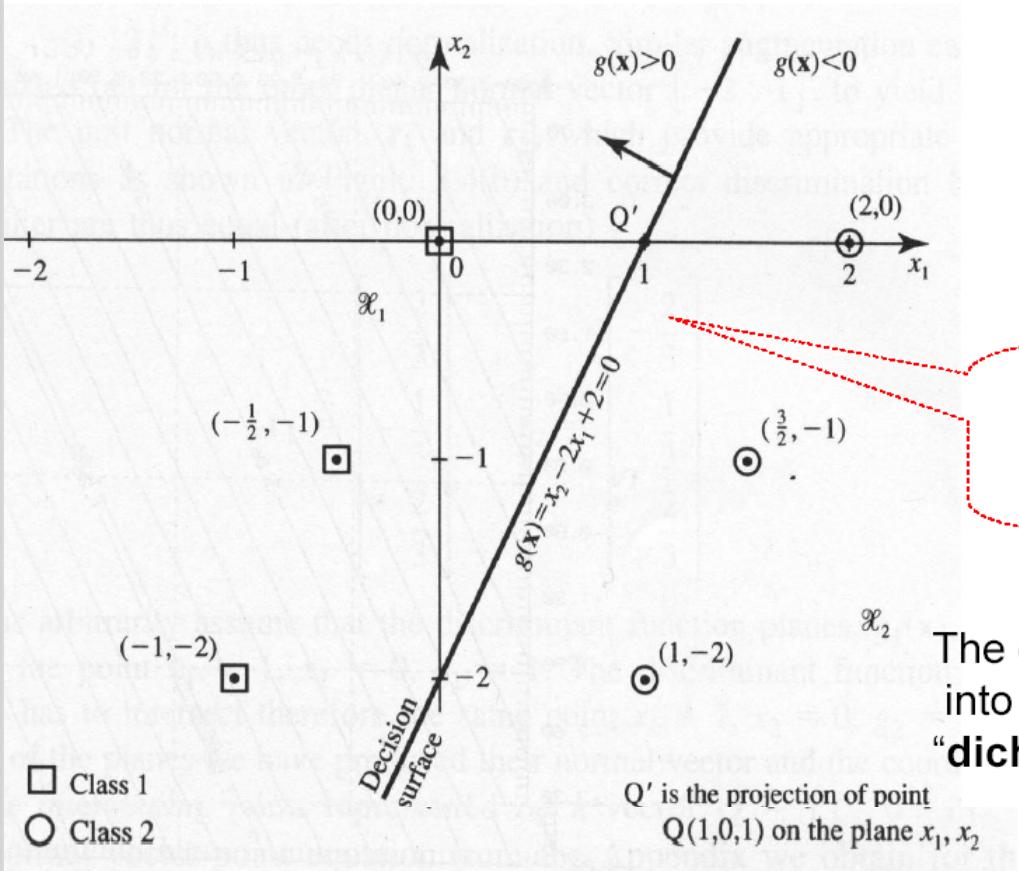
- Example 3.1

Decision surface Equation: $g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$

$$= -2x_1 + x_2 + 2$$

$$g(\boldsymbol{x}) > 0 : \text{class1}$$

$$g(\boldsymbol{x}) < 0 : \text{class2}$$



The **decision surface** does not uniquely specify the discriminant functions

The classifier that classifies patterns into two classes or categories is called "**dichotomizer**"

"two" "cut"

Discriminant Functions…

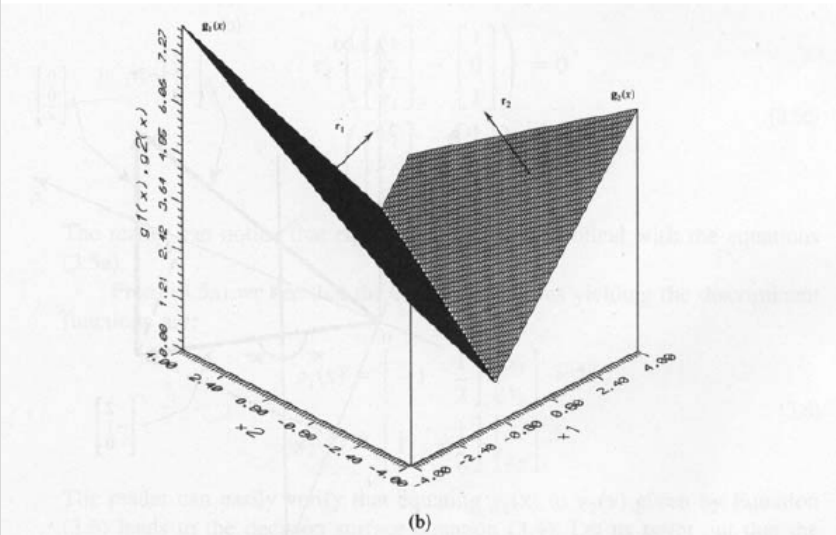


Figure 3.4a,b Illustration for Example 3.1: (a) pattern display and decision surface, (b) discriminant functions.

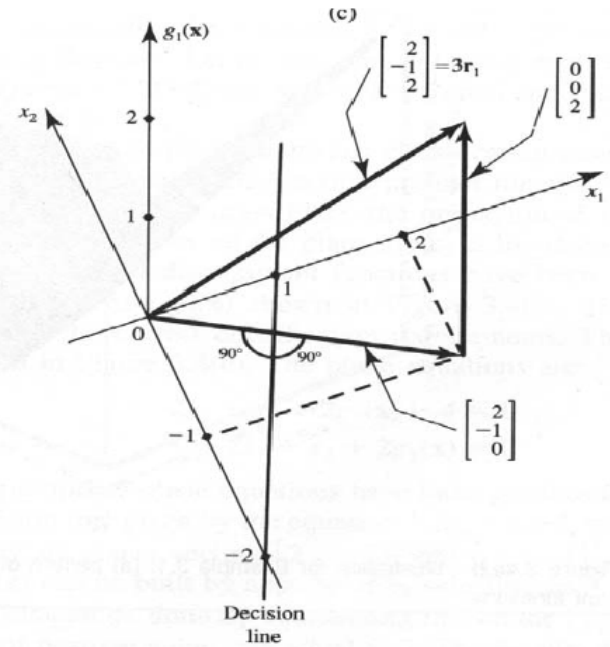


Figure 3.4c,d Illustration for Example 3.1 *(continued):* (c) contour map of discriminant functions, and (d) construction of the normal vector for $g_1(\mathbf{x})$.

Discriminant Functions…

$(x\text{-}0, y+2, g_1 - 1)(2,-1,1)=0$
$2x-y-2+g_1-1=0$
$g_1=-2x+y+3$

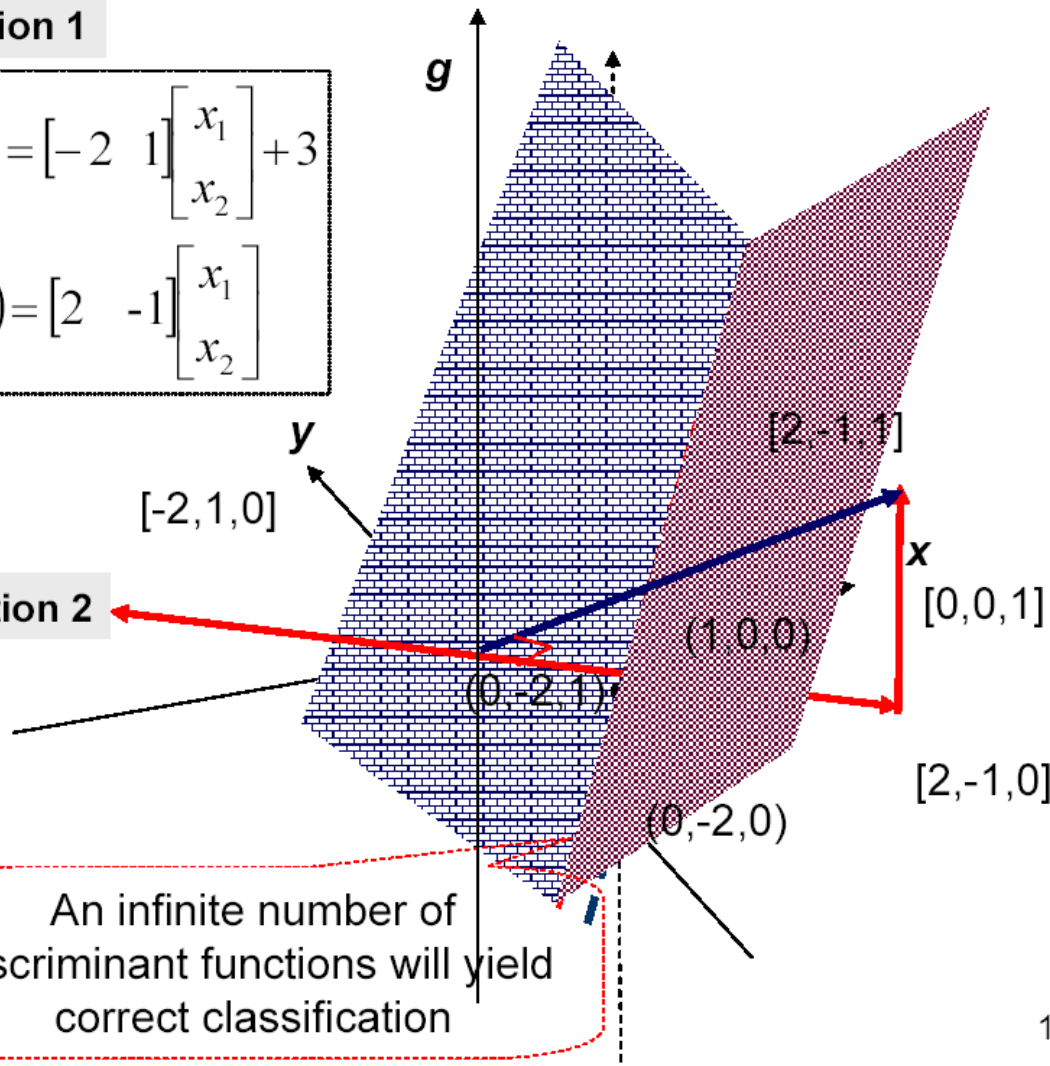
$(x\text{-}0, y+2, g_2 - 1)(-2,1,1)=0$
$-2x+y+2+g_2-1=0$
$g_2=2x-y-1$

$g=g_1-g_2=0$
$-4x+2y+4=0$
$-2x+y+2=0$

Solution 1

$$g_1(\boldsymbol{x}) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 3$$

$$g_2(\boldsymbol{x}) = \begin{bmatrix} 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



$(x\text{-}0, y+2, g_1 - 1)(2,-1,2)=0$
$2x-y-2+2g_1-2=0$
$g_1=-x+1/2y+2$

$(x\text{-}0, y+2, g_2 - 1)(-2,1,2)=0$
$-2x+y+2+2g_2-2=0$
$g_2=x-1/2y$

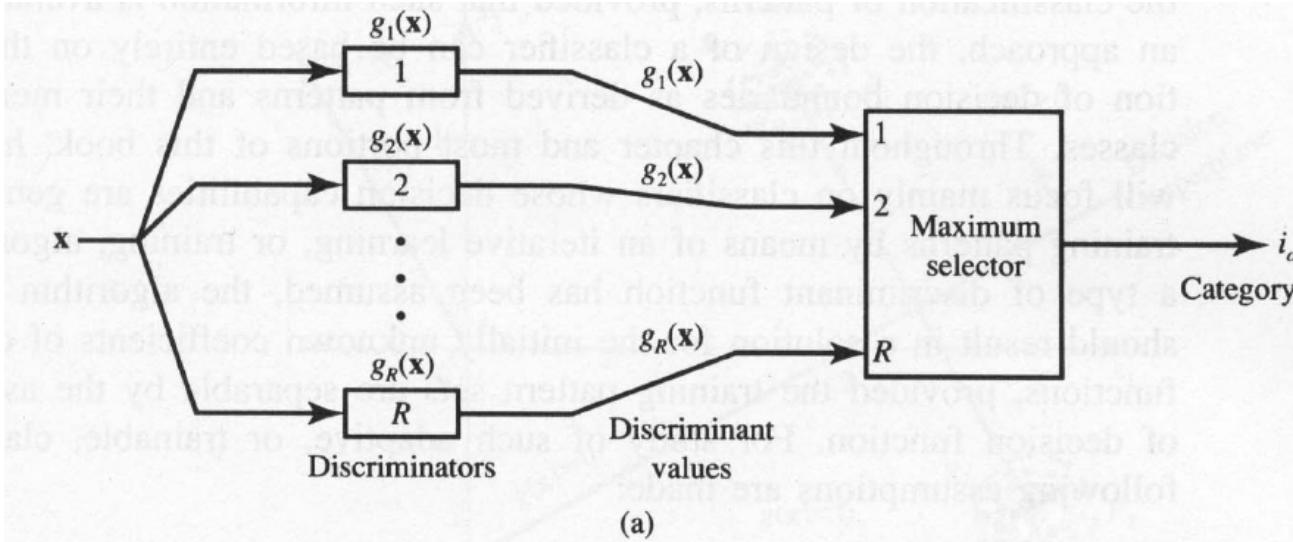
$g=g_1-g_2=0$
$-2x+y+2=0$

Solution 2

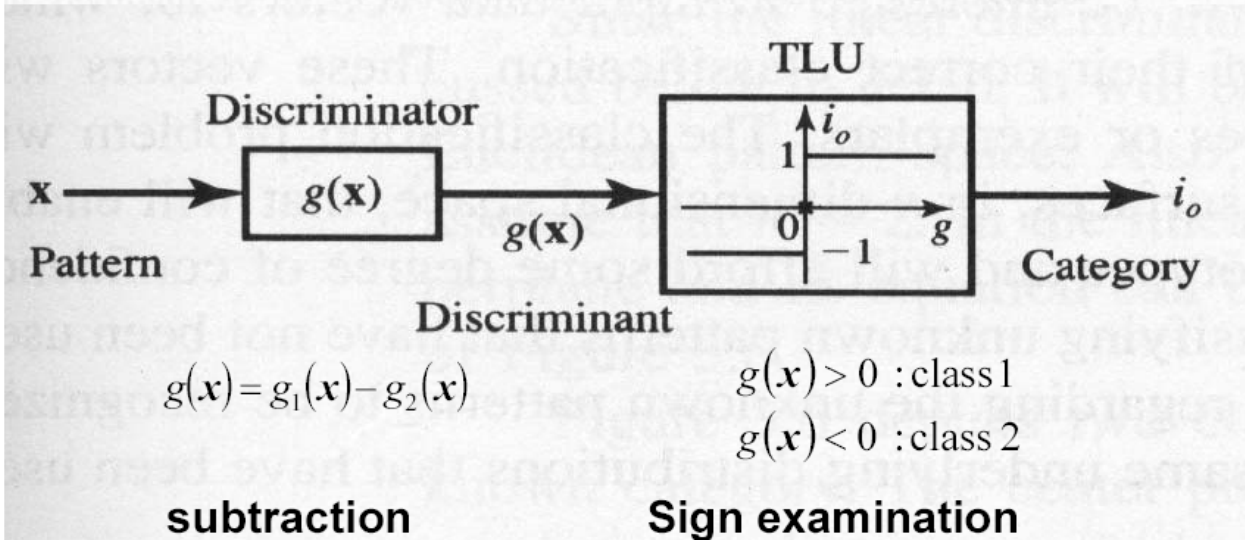
An infinite number of discriminant functions will yield correct classification

Discriminant Functions…

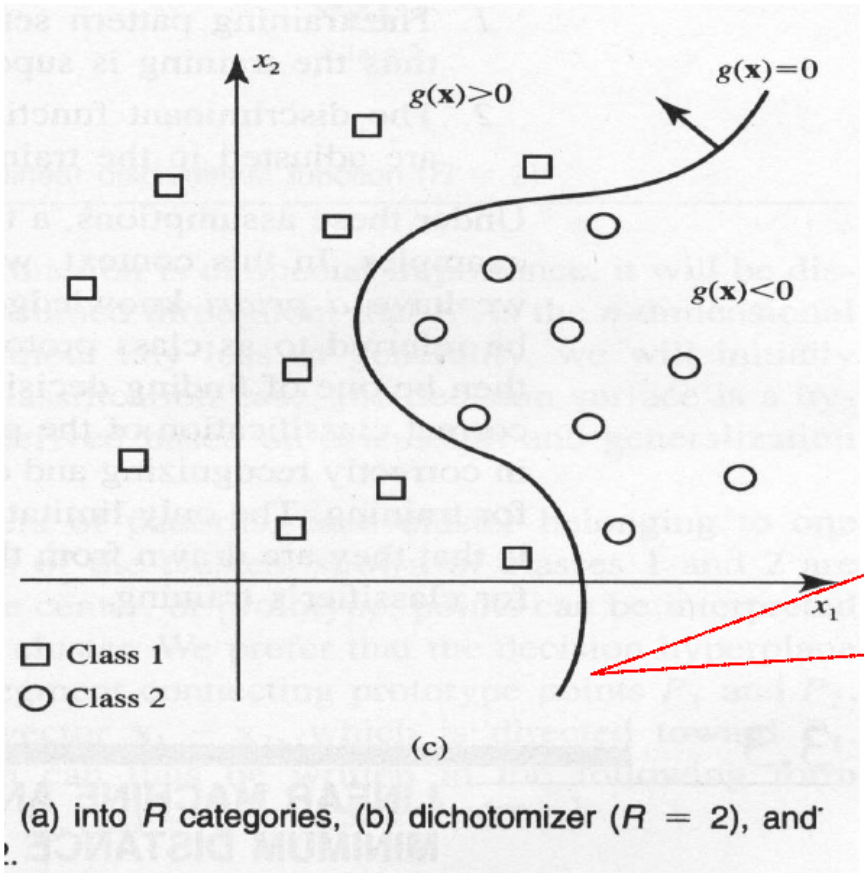
Multi-class



Two-class



Discriminant Functions…



(a) into R categories, (b) dichotomizer ($R = 2$), and

The design of discriminator for this case is not straightforward.
The discriminant functions may result as nonlinear functions of x_1 and x_2

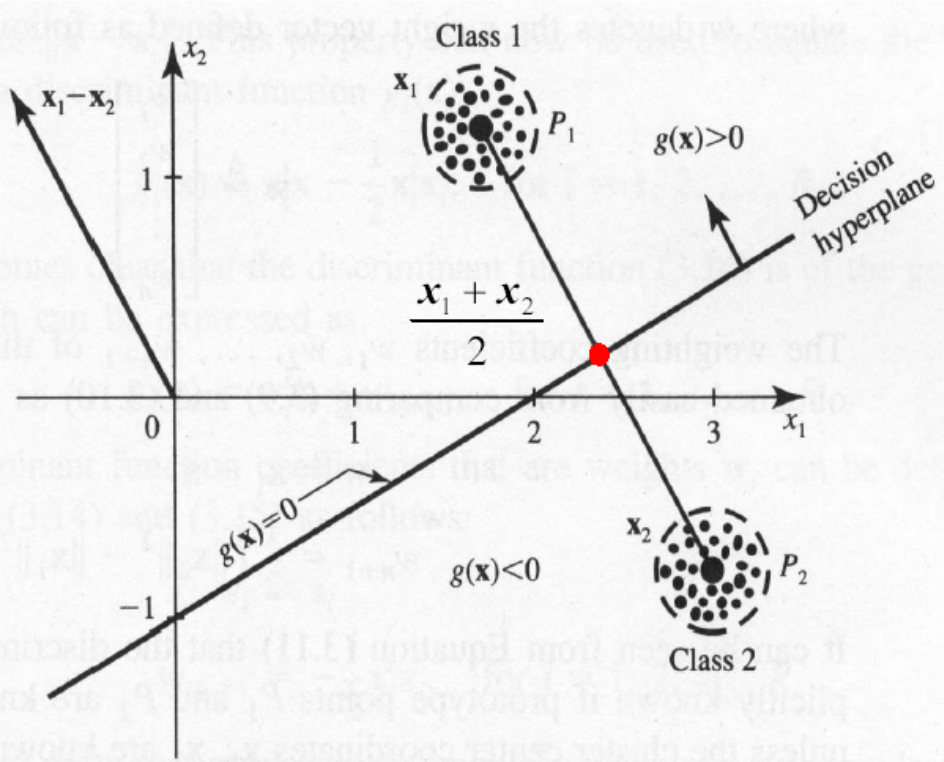
Linear Machine and Minimum Distance Classification

- Find the linear-form discriminant function for two class classification when the class prototypes are known

- **Example 3.1**: Select the decision hyperplane that contains the midpoint of the line segment connecting center point of two classes

Linear Machine and Minimum Distance Classification… (dichotomizer)

- The dichotomizer's discriminant function $g(\boldsymbol{x})$:



$$(\boldsymbol{x}_1 - \boldsymbol{x}_2)^t \left(\boldsymbol{x} - \frac{\boldsymbol{x}_1 + \boldsymbol{x}_2}{2}\right) = 0$$

$$(\boldsymbol{x}_1 - \boldsymbol{x}_2)^t \boldsymbol{x} + \frac{1}{2}\left(\|\boldsymbol{x}_2\|^2 - \|\boldsymbol{x}_1\|^2\right) = 0$$

Taken as $\begin{bmatrix} \boldsymbol{w} \\ w_{n+1} \end{bmatrix}^t \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = 0$, where

$$\boldsymbol{w} = \boldsymbol{x}_1 - \boldsymbol{x}_2$$

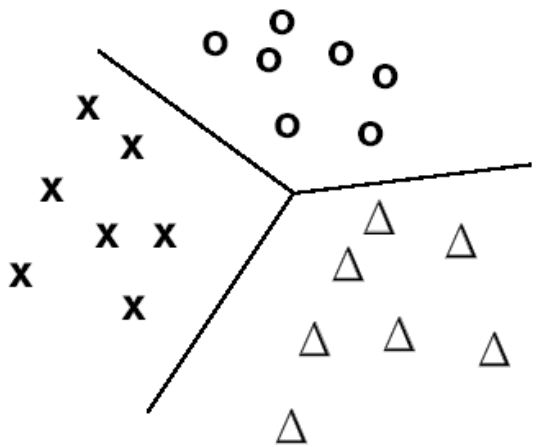
$$w_{n+1} = \frac{1}{2}\left(\|\boldsymbol{x}_2\|^2 - \|\boldsymbol{x}_1\|^2\right)$$

Augmented input pattern

It is a simple minimum-distance classifier.

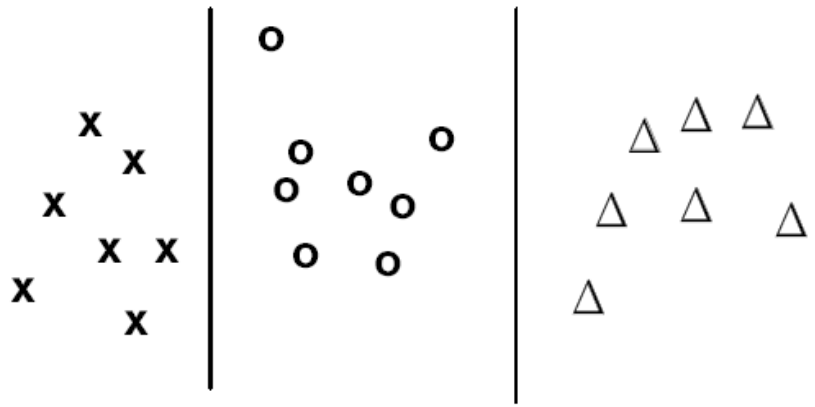
Linear Machine and Minimum Distance Classification…(multiclass classification)

- The linear-form discriminant functions for multiclass classification
  - There are up to R(R-1)/2 decision hyperplanes for R pairwise separable classes



Some classes may not be contiguous

(i.e. next to or touching another)



Linear Machine and Minimum Distance Classification… (multiclass classification)

- Linear machine or minimum-distance classifier
  - Assume the class prototypes are known for all classes
  - Euclidean distance between input pattern $\boldsymbol{x}$ and the center of class i, $\mathbf{X}_i$: $\|x - x_i\| = \sqrt{(x - x_i)^t (x - x_i)}$
    - Minimizing $\|x - x_i\|^2 = x^t x - 2x_i^t x + x_i^t x_i$ is equal to maximizing $x_i^t x - \frac{1}{2} x_i^t x_i$

      The same for all classes
  - Set the discriminant function for each class i to be:

$$g_i(x) = x_i^t x - \frac{1}{2} x_i^t x_i \Longrightarrow g_i(x) = w_i^t y$$

$$g_i(x) = \begin{bmatrix} w_i \\ w_{i,n+1} \end{bmatrix}^t \begin{bmatrix} x \\ 1 \end{bmatrix}, \text{ where } \quad w_i = x_i \quad w_{i,n+1} = -\frac{1}{2}(x_i^t x_i)$$

Linear Machine and Minimum Distance Classification… (multiclass classification)

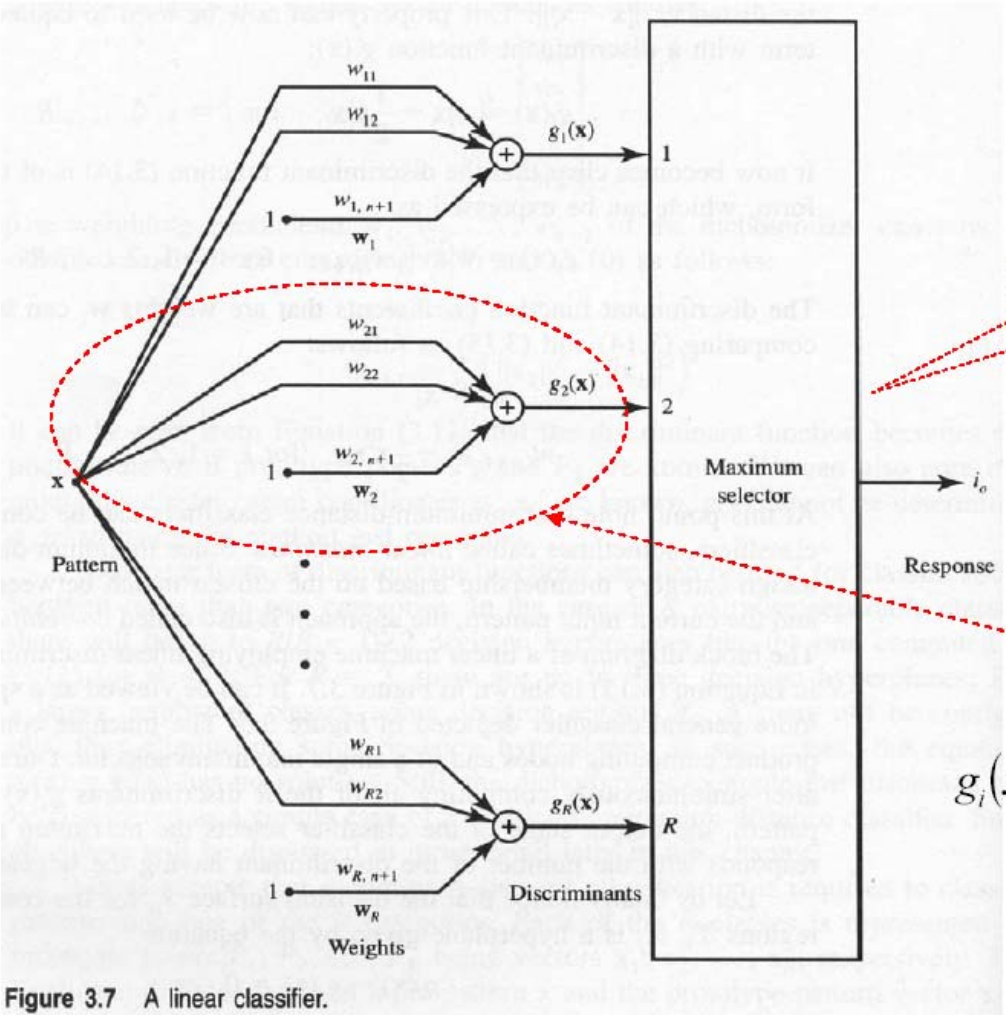


Figure 3.7 A linear classifier.

This approach is also called **correlation** classification

An 1 as the *n+1*'th component of the input pattern

$$g_i(\boldsymbol{x}) = \boldsymbol{x}_i^t \boldsymbol{x} - \frac{1}{2}\boldsymbol{x}_i^t \boldsymbol{x}_i \Rightarrow g_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{y}$$

Linear Machine and Minimum Distance Classification…

P_1, P_2, P_3 are the centres of gravity of the prototype points, we need to design a minimum distance classifier. Using the formulas from the previous slide, we get w_i

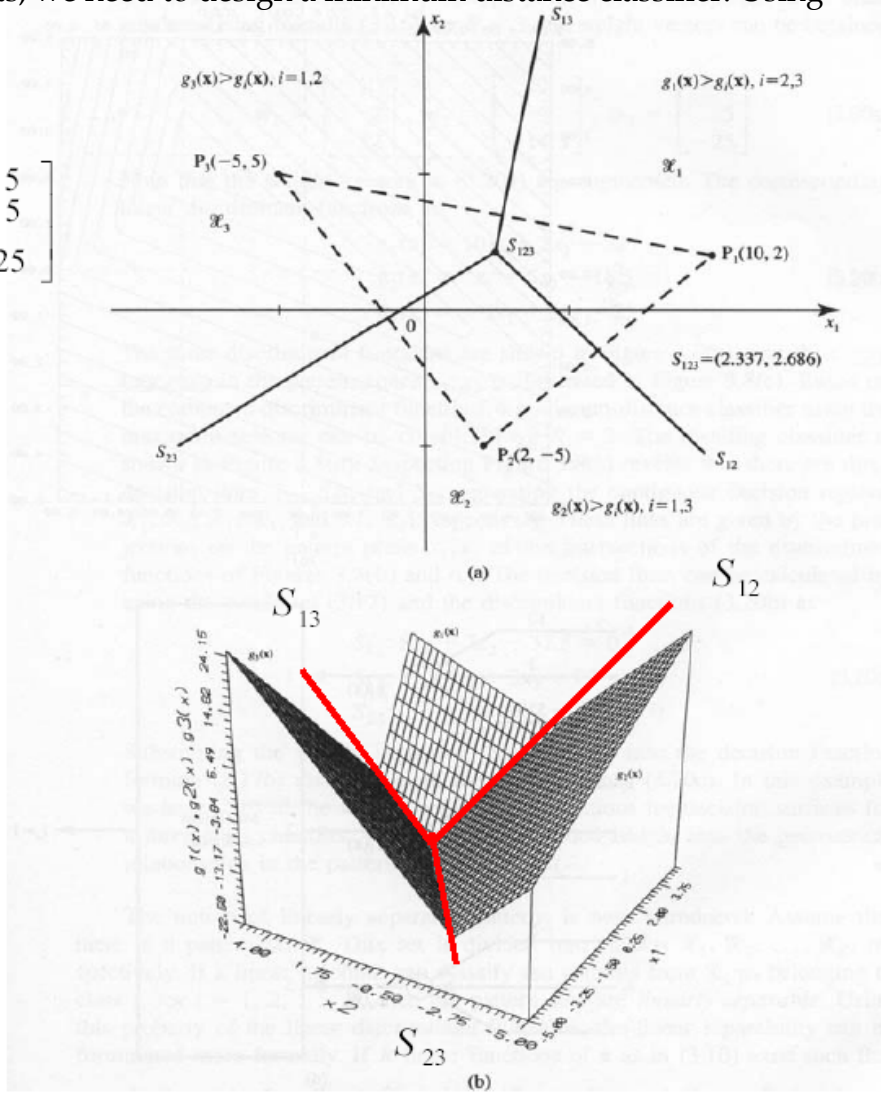
- Example 3.2

$$w_1 = \begin{bmatrix} 10 \\ 2 \\ -52 \end{bmatrix}, \quad w_2 = \begin{bmatrix} 2 \\ -5 \\ -14.5 \end{bmatrix}, \quad w_3 = \begin{bmatrix} -5 \\ 5 \\ -25 \end{bmatrix}$$

$$g_1(x) = 10x_1 + 2x_2 - 52$$
$$g_2(x) = 2x_1 - 5x_2 - 14.5$$
$$g_3(x) = -5x_1 + 5x_2 - 25$$

$$S_{12}: 8x_1 + 7x_2 - 37.5 = 0$$
$$S_{13}: -15x_1 + 3x_2 + 27 = 0$$
$$S_{23}: -7x_1 + 10x_2 - 10.5 = 0$$

$$g_i(x) = x_i^t x - \frac{1}{2} x_i^t x_i$$



Note: to find S_{12} we need to compute $(g_1\text{-}g_2)$

Linear Machine and Minimum Distance Classification…

- If R linear discriminant functions exist for a set of patterns such that

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \text{ for } \mathbf{x} \in \text{Class } i,$$

$$i = 1, 2, \ldots, R, \qquad j = 1, 2, \ldots, R, \qquad i \neq j$$

- The classes are linearly separable.

Linear Machine and Minimum Distance Classification… Example:

For the minimum-distance (linear) dichotomizer, the weight and augmented pattern vectors are

$$\mathbf{w} = \begin{bmatrix} 2 \\ -1 \\ 2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

(a) Find the equation of the decision surface in the pattern space.

(b) Find the equation of the decision surface in the augmented pattern space.

(c) Compute the new solution weight vector if the two class prototype points are

$$\mathbf{x}_1 = [2 \quad 5]^t \text{ and } \mathbf{x}_2 = [-1 \quad -3]^t.$$

(d) Sketch the decision surfaces for each case in parts (a), (b), and (c).

Linear Machine and Minimum Distance Classification… Example…

(a) $2x_1-x_2+2=0$, decision surface is a line

(b) $2x_1-x_2+2=0$, decision surface is a plane

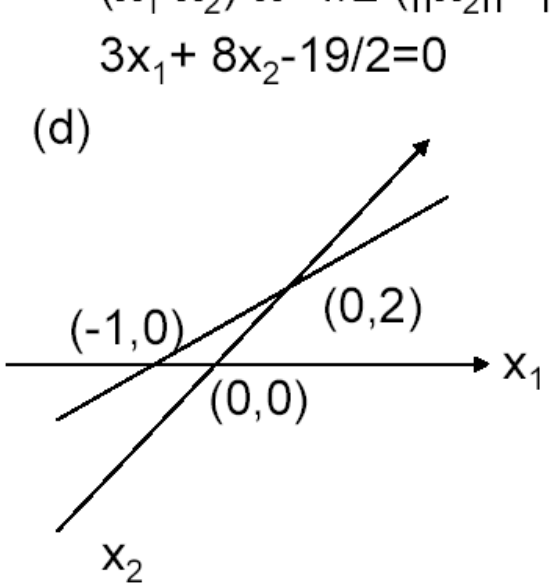
(c) $\mathbf{x}_1=[2,5]$, $\mathbf{x}_2=[-1,-3]$

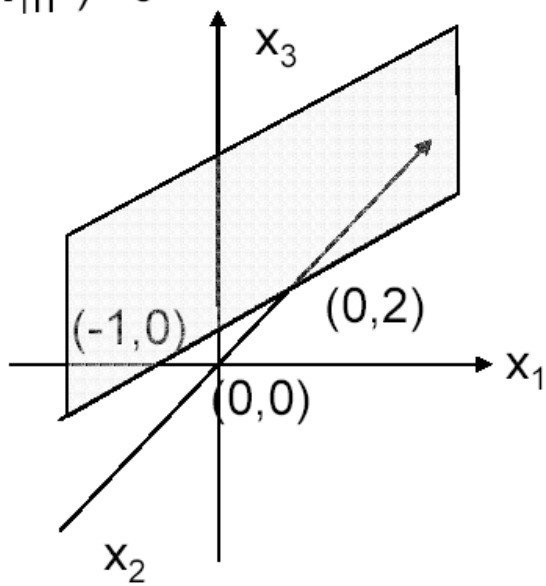
=>The decision surface for minimum distance classifier

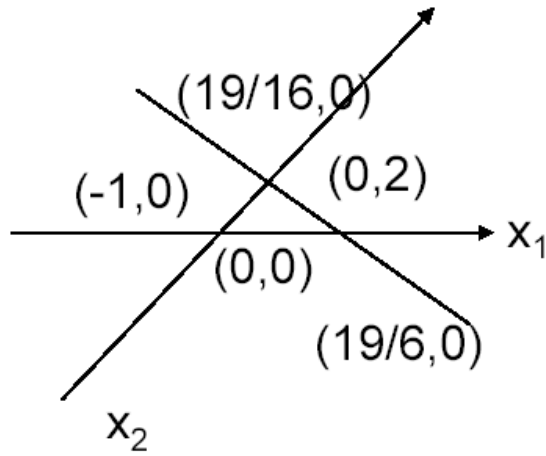
$(\mathbf{x}_1-\mathbf{x}_2)^t\,\mathbf{x}+1/2\,(||\mathbf{x}_2||^2-||\mathbf{x}_1||^2)^t=0$

$3x_1+8x_2-19/2=0$

(d)







Linear Machine and Minimum Distance Classification…

- Examples 3.1 and 3.2 have shown that the coefficients (weights) of the linear discriminant functions can be determined if the a priori information about the sets of patterns and their class membership is known
- In the next section (Discrete perceptron) we will examine neural networks that derive their weights during the learning cycle.

Linear Machine and Minimum Distance Classification…

- The example of linearly non-separable patterns

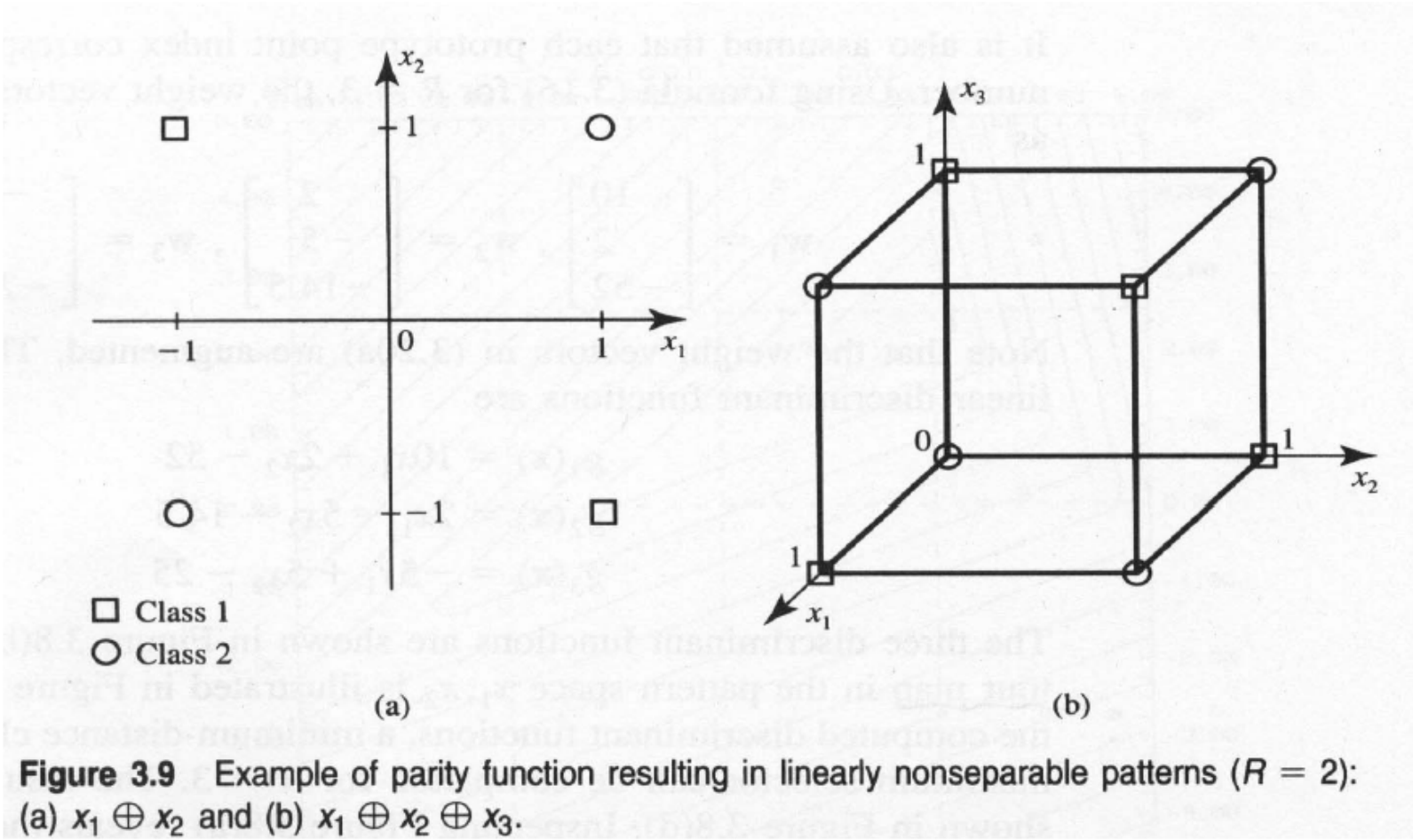
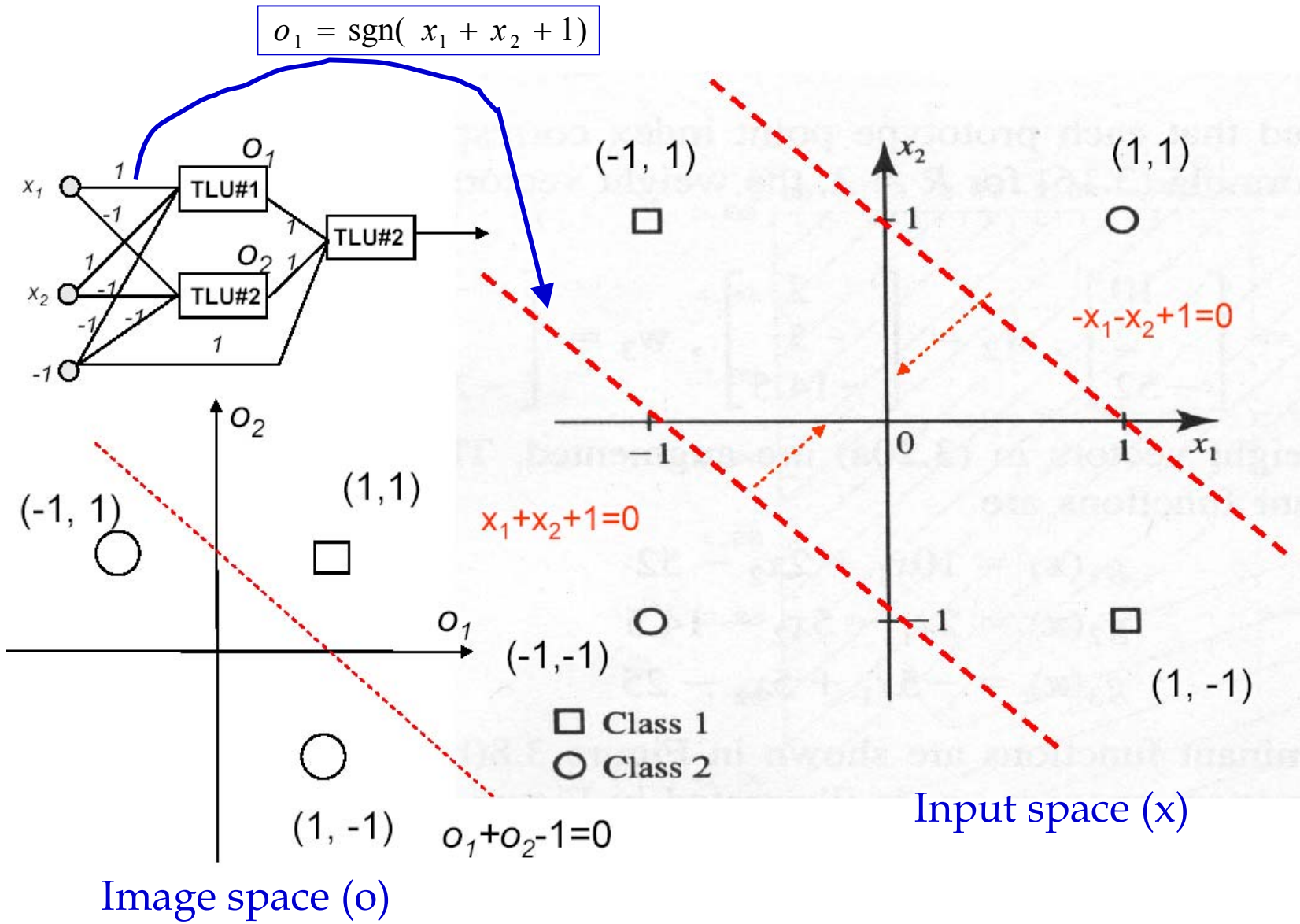


Figure 3.9 Example of parity function resulting in linearly nonseparable patterns ($R = 2$): (a) $x_1 \oplus x_2$ and (b) $x_1 \oplus x_2 \oplus x_3$.

Linear Machine and Minimum Distance Classification…

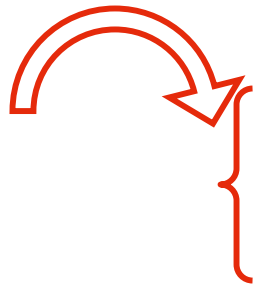


Linear Machine and Minimum Distance Classification…

$$o_1 = \text{sgn}(x_1 + x_2 + 1)$$

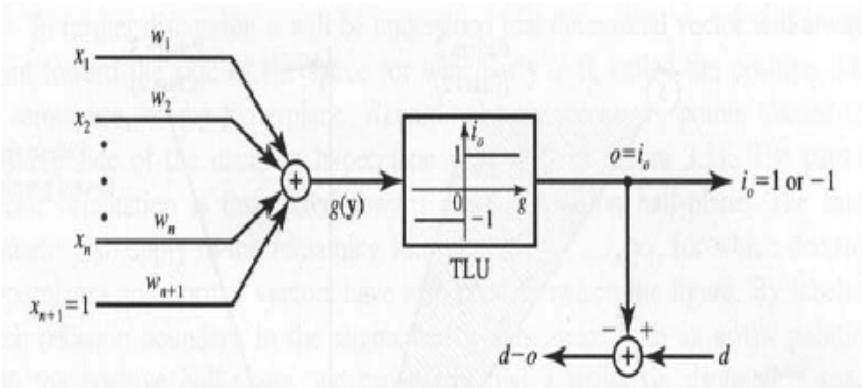
$$o_2 = \text{sgn}(-x_1 - x_2 + 1)$$

These 2 inputs map to the same point (1,1) in the image space



x_1	x_2	o_1	o_2
-1	-1	-1	1
-1	1	1	1
1	-1	1	1
1	1	1	-1

The Discrete Perceptron



Discrete Perceptron Training Algorithm

- So far, we have shown that **coefficients** of linear discriminant functions called **weights** can be determined based on *a priori* information about sets of patterns and their class membership.
- In what follows, we will begin to examine neural network classifiers that derive their weights during the learning cycle.
- The sample pattern vectors $x_1, x_2, \ldots, x_p$, called the training sequence, are presented to the machine along with the correct response.

Discrete Perceptron Training Algorithm - Geometrical Representations

http://140.122.185.120
Zurada, Chapter 3

- Examine the neural network classifiers that derive/training their weights based on the **error-correction scheme**

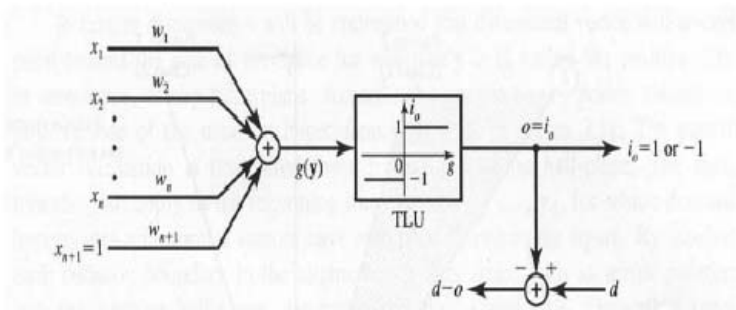


Figure 3.10 Linear dichotomizer using hard-limiting threshold element, or the TLU-based perceptron.

$$g(y) = w^t y$$

Augmented input pattern

(Intersects the origin point **w=0**)

Class 1: $w^t y > 0$

Class 2: $w^t y < 0$

Vector Representations in the **Weight Space**

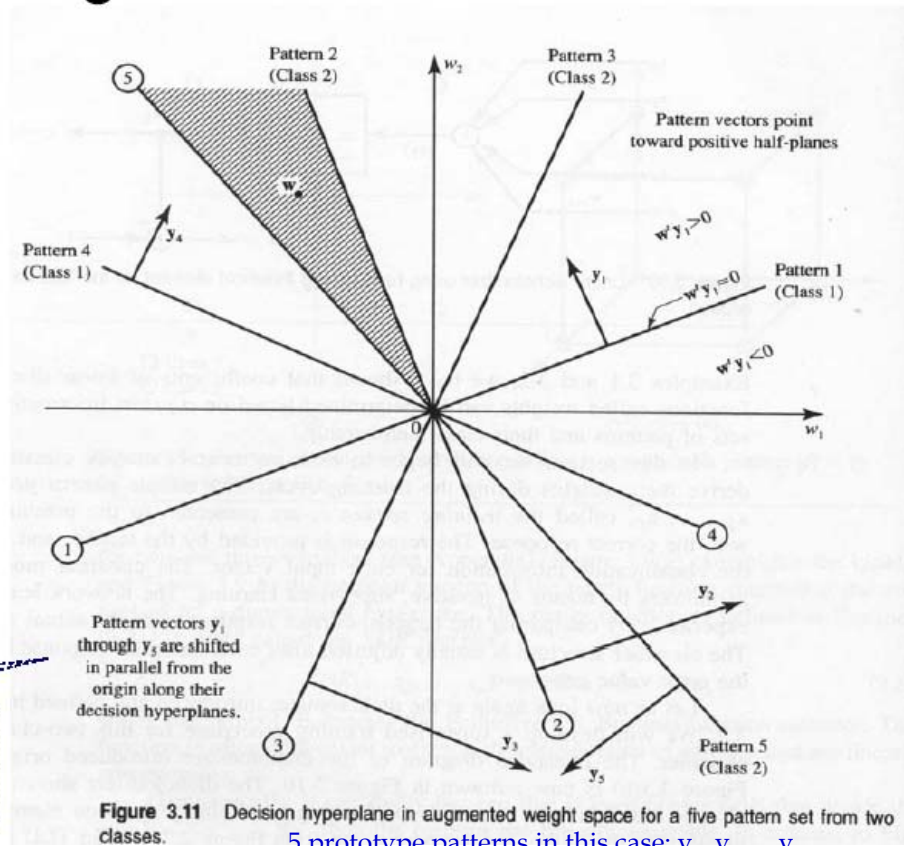


Figure 3.11 Decision hyperplane in augmented weight space for a five pattern set from two classes.

5 prototype patterns in this case: y_1, y_2, …y_5

If dim of augmented pattern vector is > 3, our power of visualization are no longer of assistance. In this case, the only recourse is to use the analytical approach.

Discrete Perceptron Training Algorithm - Geometrical Representations…

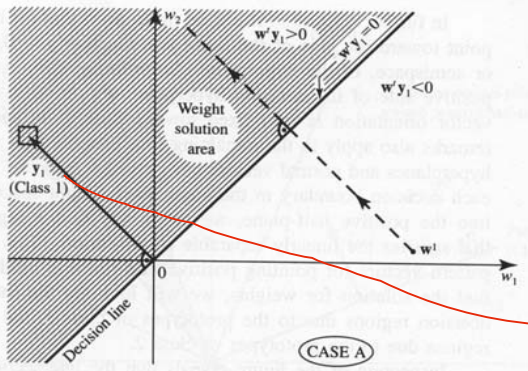
•Devise an analytic approach based on the geometrical representations

– E.g. the decision surface for the training pattern y_1

y_1 in Class 1

(see previous slide)

Weight Space



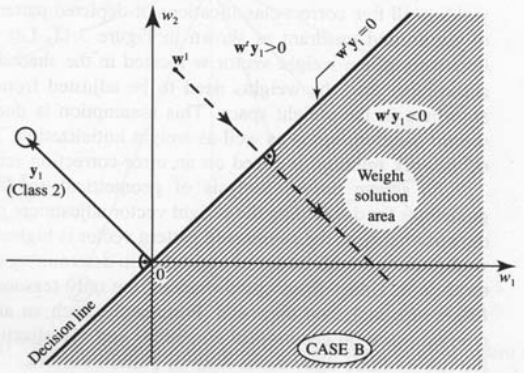


Figure 3.12a Weight adjustments of learning dichotomizer: (a) steepest descent.

y_1 in Class 2

Weight Space

$$\nabla_w \left(w^t y_1 \right) = y_1$$

Gradient (the direction of steepest increase)

If y_1 in Class 1:

$$w' = w^1 + c y_1$$

If y_1 in Class 2:

$$w' = w^1 - c y_1$$

c controls the size of adjustment

c (>0) is the correction increment (*is two times the learning constant ρ introduced before*)

(correction in negative gradient direction)

Discrete Perceptron Training Algorithm - Geometrical Representations…

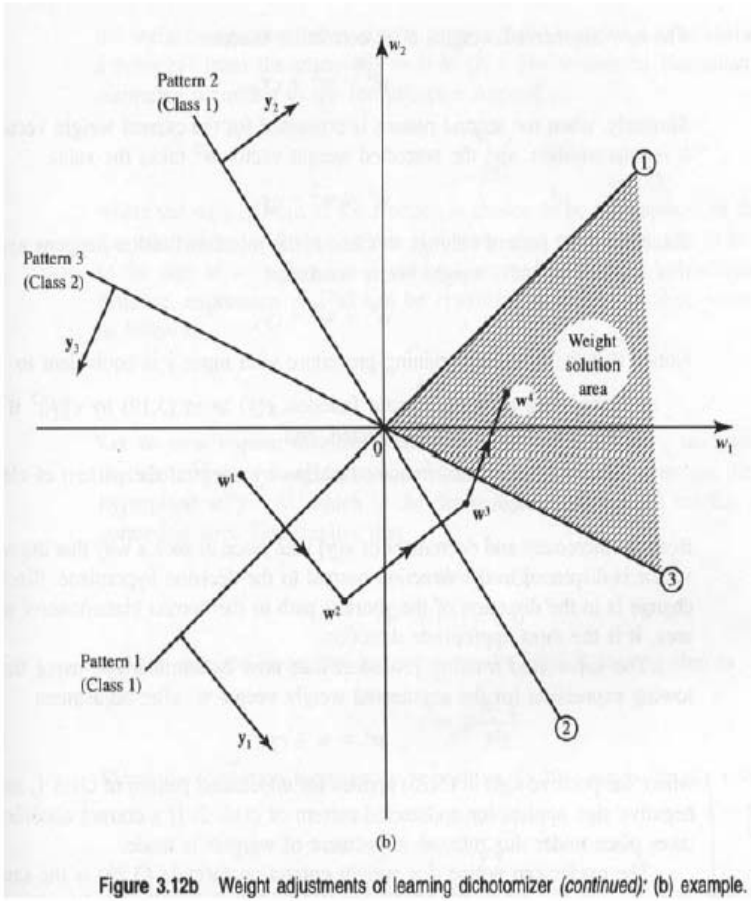


Figure 3.12b Weight adjustments of learning dichotomizer *(continued)*: (b) example.

Weight adjustments of three augmented training pattern $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, $\boldsymbol{y}_3$, shown in the weight space

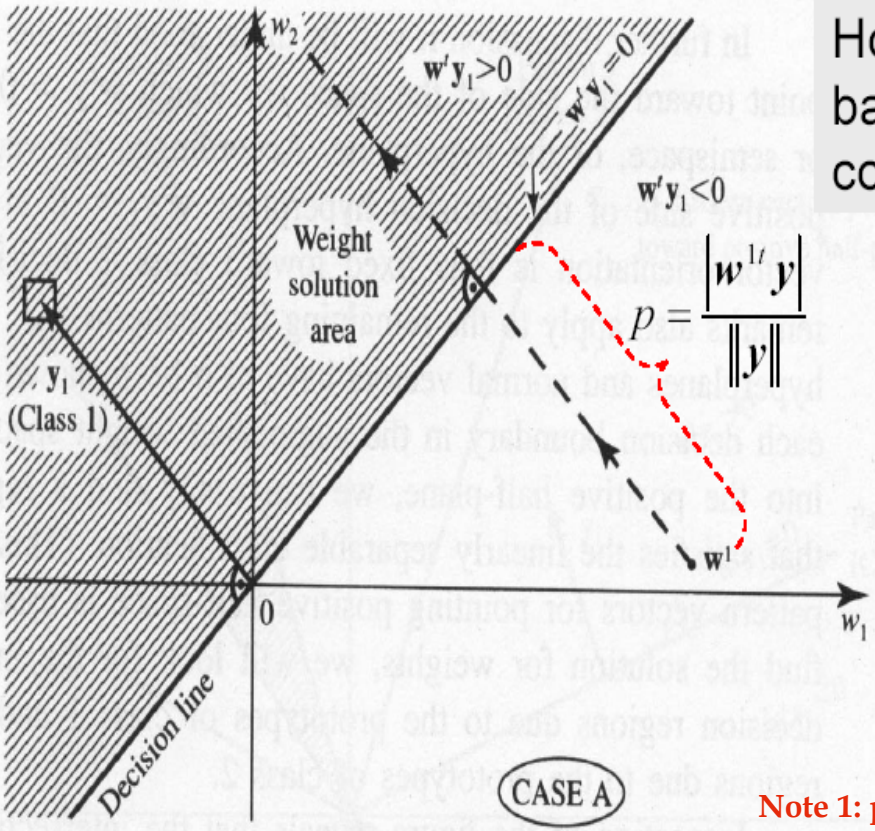
$$\boldsymbol{y}_1 \in C_1$$
$$\boldsymbol{y}_2 \in C_1$$
$$\boldsymbol{y}_3 \in C_2$$

- Weights in the shaded region are the solutions
- The three lines labeled are fixed during training

Weight Space

Discrete Perceptron Training Algorithm - Geometrical Representations…

- More about the correction increment *c*
  - If it is not merely a constant, but related to the current training pattern



How to select the correction increment based on the dislocates of $\boldsymbol{w}^1$ and the corrected weight vector $\boldsymbol{w}$

$$\left(w^1 \pm cy\right)^t y = 0$$

$$c = \mp \frac{w^{1t} y}{y^t y} = \frac{|w^{1t} y|}{\|y\|^2}, \text{ because } \quad c > 0$$

$$\Rightarrow cy = \frac{|w^{1t} y|}{\|y\|^2} y \Rightarrow \|cy\| = \frac{|w^{1t} y|}{y^t y}\|y\| = p$$

Note 1: p=distance so >0

Note 2: c is not constant and depends on the current training pattern as expressed by eq. Above.

Discrete Perceptron Training Algorithm - Geometrical Representations…

- For **fixed correction rule:** c=constant, the correction of weights is always the same fixed portion of the current training vector
  - The weight can be initialised at any value

$$\boldsymbol{w}' = \boldsymbol{w} \pm c\boldsymbol{y} \quad \text{or} \quad \boldsymbol{w}' = \boldsymbol{w} + \Delta\boldsymbol{w}, \quad \Delta\boldsymbol{w} = c\left[d - \operatorname{sgn}\left(\boldsymbol{w}^t \boldsymbol{y}\right)\right]\boldsymbol{y}$$

- For **dynamic correction rule:** c depends on the distance from the weight (i.e. the weight vector) to the decision surface in the weight space. Hence

Current weight ← $c\boldsymbol{y} = \dfrac{\left|\boldsymbol{w}^{1t}\boldsymbol{y}\right|}{\|\boldsymbol{y}\|^2}\boldsymbol{y}$ → Current input pattern

  - The initial weight should be different from 0.

(if $\mathbf{w}^1$=0, then c$\mathbf{y}$ =0 and $\mathbf{w}'=\mathbf{w}^1+c\mathbf{y}=\mathbf{0}$, therefore no possible adjustments).

Discrete Perceptron Training Algorithm - Geometrical Representations…

•Dynamic correction rule: Using the value of c from previous slide as a reference, we devise an adjustment technique which depends on the length $\mathbf{w}^2$-$\mathbf{w}^1$

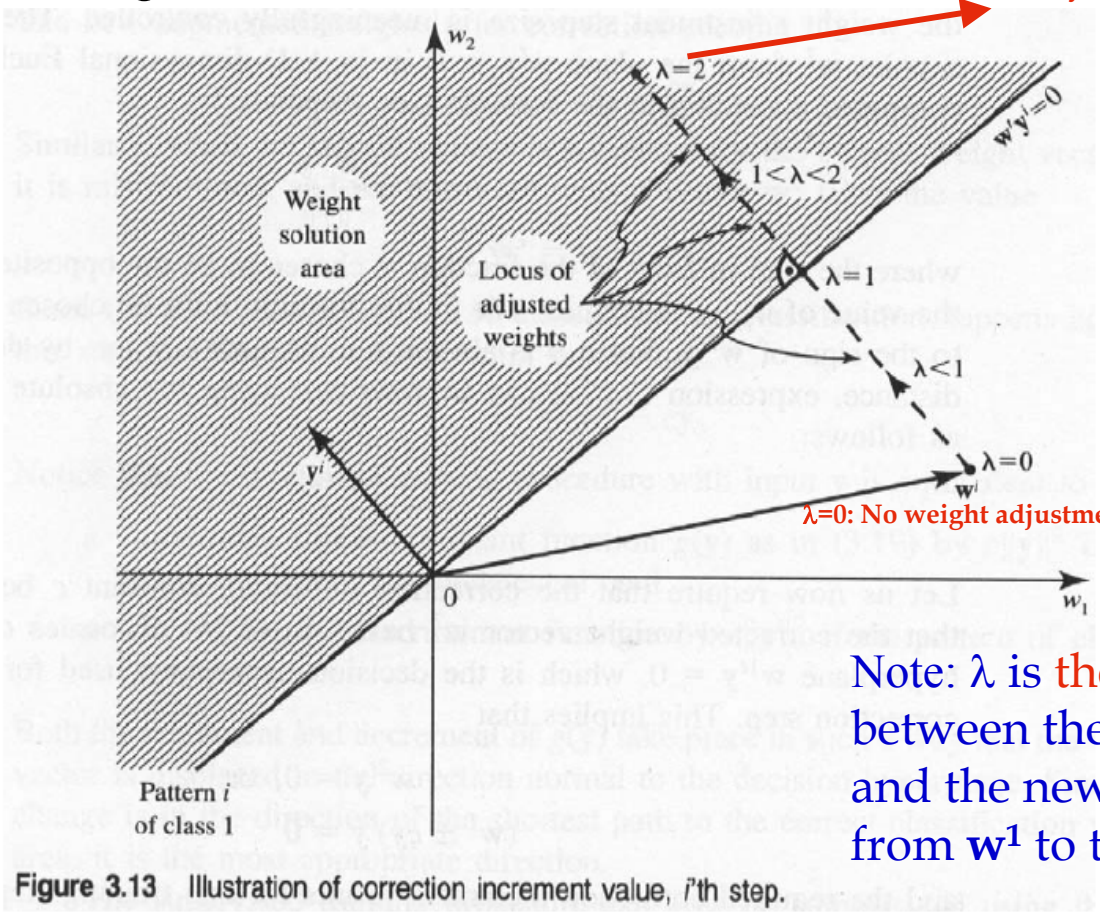


Figure 3.13 Illustration of correction increment value, i'th step.

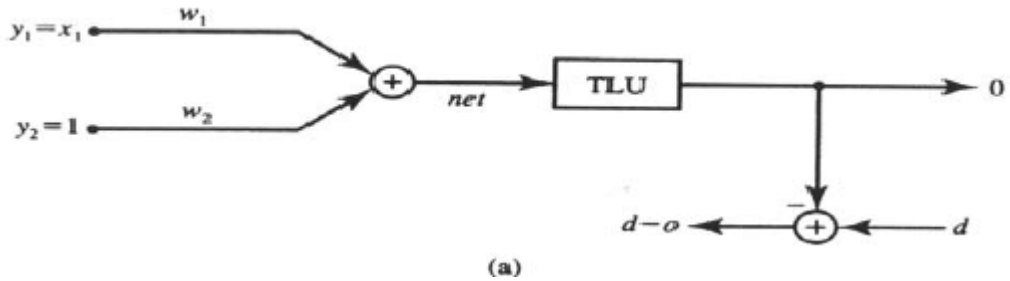
$$c = \lambda \frac{\left| \mathbf{w}^{1t} \mathbf{y} \right|}{\|\mathbf{y}\|^2}$$

$$c\mathbf{y} = \lambda \frac{\left| \mathbf{w}^{1t} \mathbf{y} \right|}{\|\mathbf{y}\|} \frac{\mathbf{y}}{\|\mathbf{y}\|}$$

Note: λ is the ratio of the distance between the old weight vector $\mathbf{w}^1$ and the new $\mathbf{w}^2$, to the distance from $\mathbf{w}^1$ to the pattern hyperplane

Discrete Perceptron Training Algorithm - Geometrical Representations…

•Example:



$$x_1 = 1, \quad x_3 = 3, \quad d_1 = d_3 = 1 : \text{class } 1$$
$$x_2 = -0.5, \quad x_4 = -2, \quad d_2 = d_4 = -1 : \text{class } 2$$

•The augmented input vectors are:

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{y}_2 = \begin{bmatrix} -0.5 \\ 1 \end{bmatrix}, \quad \mathbf{y}_3 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

•The decision lines $\mathbf{w}^t\mathbf{y}_i=0$, for i=1, 2, 3, 4 are sketched on the augmented weight space as follows:

Discrete Perceptron Training Algorithm - Geometrical Representations…

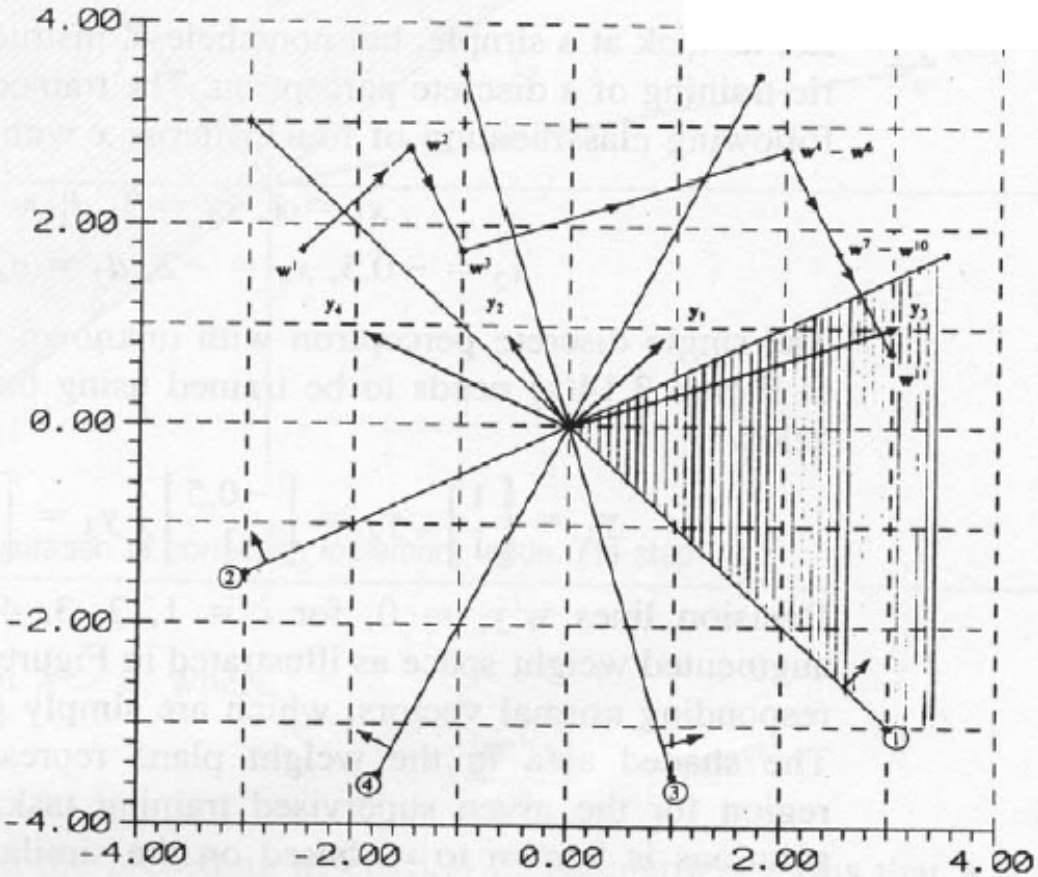


Figure 3.14a,b Discrete perceptron classifier training in Example 3.3: (a) network diagram, (b) fixed correction rule training.

Discrete Perceptron Training Algorithm - Geometrical Representations…

$$\text{For } c = 1 \text{ and } \mathbf{w}^1 = \begin{bmatrix} -2.5 & 1.75 \end{bmatrix}^t$$

- Using $\mathbf{w}' = \mathbf{w} \pm c\mathbf{y}$ the weight training with each step can be summarized as follows:

$$\Delta \mathbf{w}^k = \frac{c}{2}[d_k - \text{sgn}(\mathbf{w}^{kt}\mathbf{y}_k)]\mathbf{y}_k$$

- We obtain the following outputs and weight updates:
- **Step 1**: Pattern $\mathbf{y}_1$ is input

$$o_1 = \text{sgn}\left(\begin{bmatrix} -2.5 & 1.75 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = -1$$

$$d_1 - o_1 = 2$$

$$\mathbf{w}^2 = \mathbf{w}^1 + \mathbf{y}^1 = \begin{bmatrix} -1.5 \\ 2.75 \end{bmatrix}$$

Discrete Perceptron Training Algorithm - Geometrical Representations…

- **Step 2**: Pattern $\mathbf{y}_2$ is input

$$o_2 = \text{sgn}\left(\begin{bmatrix} -1.5 & 2.75 \end{bmatrix} \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} \right) = 1$$

$$d_2 - o_2 = -2$$

$$\mathbf{w}^3 = \mathbf{w}^2 - \mathbf{y}^2 = \begin{bmatrix} -1 \\ 1.75 \end{bmatrix}$$

- **Step 3**: Pattern $\mathbf{y}_3$ is input

$$o_3 = \text{sgn}\left(\begin{bmatrix} -1 & 1.75 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right) = -1$$

$$d_3 - o_3 = 2$$

$$\mathbf{w}^4 = \mathbf{w}^3 + \mathbf{y}^3 = \begin{bmatrix} 2 \\ 2.75 \end{bmatrix}$$

Discrete Perceptron Training Algorithm - Geometrical Representations…

• Since we have no evidence of correct classification of weight $\mathbf{w}^4$ the training set consisting of an ordered sequence of patterns $\mathbf{y}_1$,$\mathbf{y}_2$ and $\mathbf{y}_3$ needs to be recycled. We thus have $\mathbf{y}^4 = \mathbf{y}_1$, $\mathbf{y}^5 = \mathbf{y}_2$, etc (the superscript is used to denote the following training step number).

•**Step 4, 5**: $\mathbf{w}^6 = \mathbf{w}^5 = \mathbf{w}^4$ (no misclassification, thus no weight adjustments).

•You can check that the adjustment following in steps 6 through 10 are as follows:

$$\mathbf{w}^7 = \begin{bmatrix} 2.5 & 1.75 \end{bmatrix}^t$$

$$\mathbf{w}^{10} = \mathbf{w}^9 = \mathbf{w}^8 = \mathbf{w}^7$$

$$\mathbf{w}^{11} = \begin{bmatrix} 3 & 0.75 \end{bmatrix}^t$$

$\mathbf{w}^{11}$ is in solution area.

The Continuous Perceptron

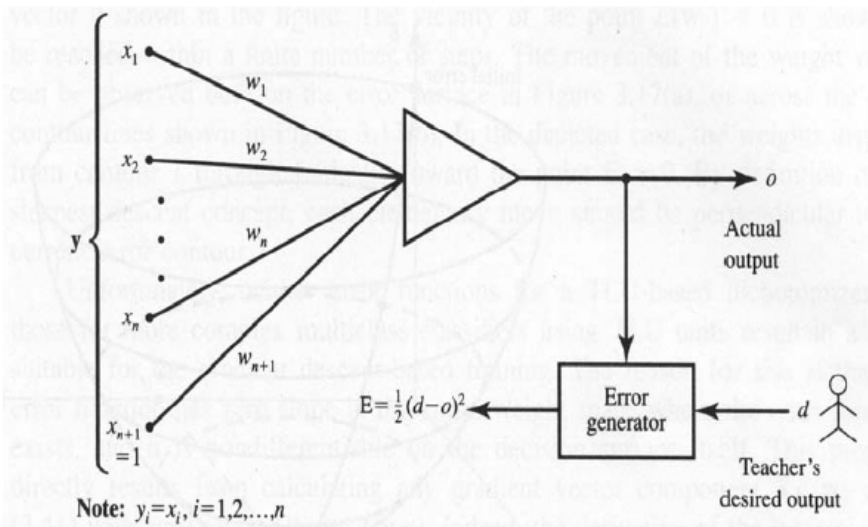


Figure 3.16 Continuous perceptron training using square error minimization.

Continuous Perceptron Training Algorithm

http://140.122.185.120
Zurada, Chapter 3

- Replace the TLU (Threshold Logic Unit) with the sigmoid activation function for two reasons:
  - Gain finer control over the training procedure
  - Facilitate the differential characteristics to enable computation of the error gradient

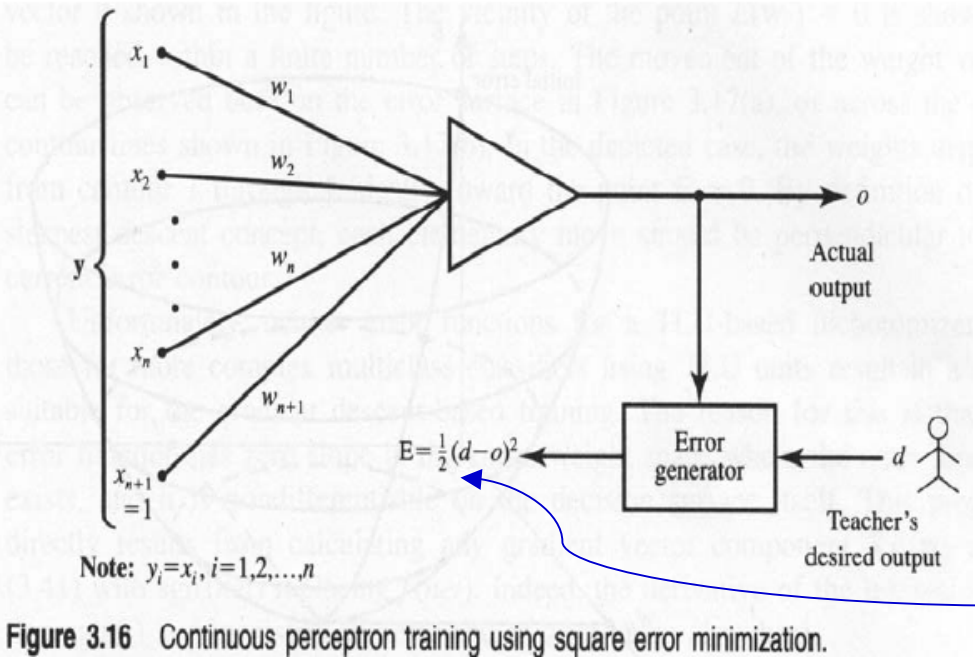


Figure 3.16 Continuous perceptron training using square error minimization.

$$\hat{w} = w - \eta \nabla E(w)$$

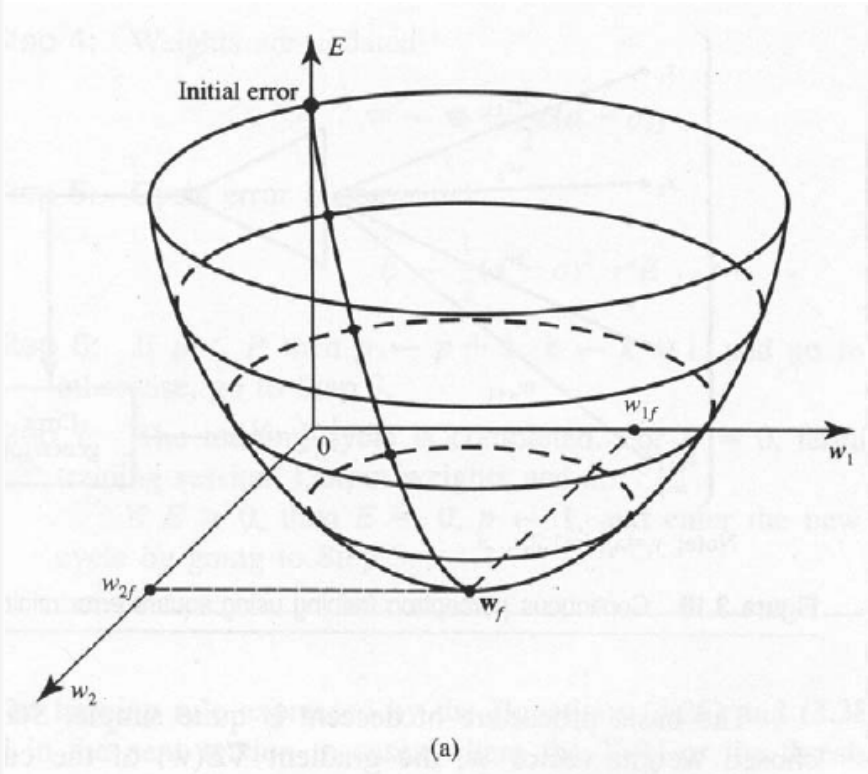
learning constant

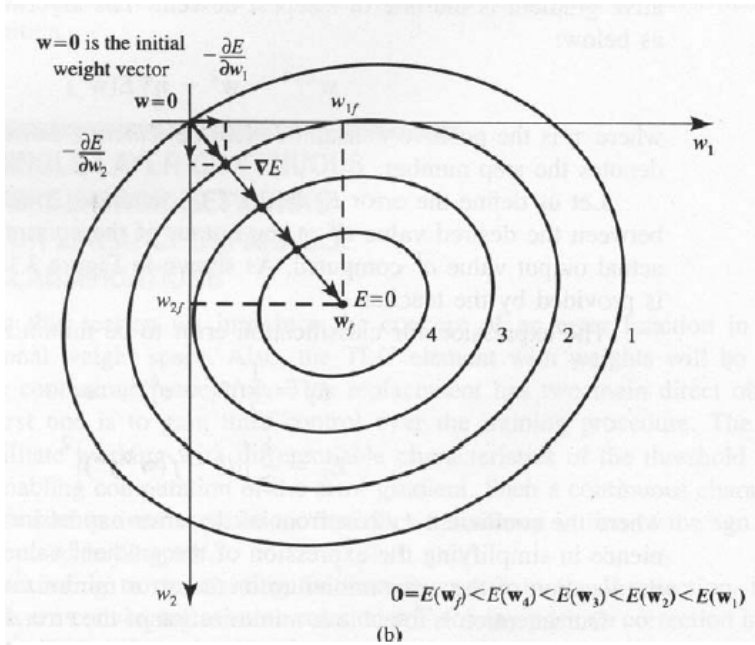
error gradient (of current error function)

The factor ½ does not affect the location of the error minimum

Continuous Perceptron Training Algorithm…

- The new weights is obtained by moving in the direction of the negative gradient along the multidimensional error surface





By definition of the steepest descent concept, each elementary move should be perpendicular to the current error contour.

Continuous Perceptron Training Algorithm…

- Define the error as the squared difference between the desired output and the actual output

$$E = \frac{1}{2}(d-o)^2$$

$$\text{or} \quad E = \frac{1}{2}\left[d - f\left(\mathbf{w}^t \mathbf{y}\right)\right]^2 = \frac{1}{2}\left[d - f(net)\right]^2$$

$$\nabla E(\mathbf{w}) = \frac{1}{2}\nabla\left(\left[d - f(net)\right]^2\right)$$

$$\nabla E(\mathbf{w}) \stackrel{\Delta}{=} \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_{n+1}} \end{bmatrix} = -(d-o) f'(net) \begin{bmatrix} \frac{\partial (net)}{\partial w_1} \\ \frac{\partial (net)}{\partial w_2} \\ \vdots \\ \frac{\partial (net)}{\partial w_{n+1}} \end{bmatrix} = -(d-o) f'(net)\, y$$

Since $net = \mathbf{w}^t \mathbf{y}$, we have $\frac{\partial (net)}{\partial w_i} = y_i \quad i = 1,2,\ldots,n+1$

Training rule of continous perceptron (equivalent to delta training rule)

Continuous Perceptron Training Algorithm…

- Bipolar Continuous Activation Function

$$f(net)=\frac{2}{1+\exp(-\lambda\cdot net)}-1 \qquad f'(net)=\lambda\cdot\frac{2\exp(-\lambda\cdot net)}{[1+\exp(-\lambda\cdot net)]^2}=\lambda\cdot\{1-[f(net)]^2\}=\lambda(1-o^2)$$

$$\hat{\boldsymbol{w}} = \boldsymbol{w} + \frac{1}{2}\eta\cdot\lambda\,(d-o)(1-o^2)\boldsymbol{y}$$

- Unipolar Continuous Activation Function

$$f(net)=\frac{1}{1+\exp(-\lambda\cdot net)} \qquad f'(net)=\frac{\lambda\cdot\exp(-\lambda\cdot net)}{[1+\exp(-\lambda\cdot net)]^2}=\lambda\cdot f(net)[1-f(net)]=\lambda\cdot o(1-o)$$

$$\hat{\boldsymbol{w}} = \boldsymbol{w} + \eta\cdot\lambda\cdot(d-o)o(1-o)\boldsymbol{y}$$

Continuous Perceptron Training Algorithm…

Same as previous example (of discrete perceptron) but with a continuous activation function and using the delta rule.

- Example 3.3

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

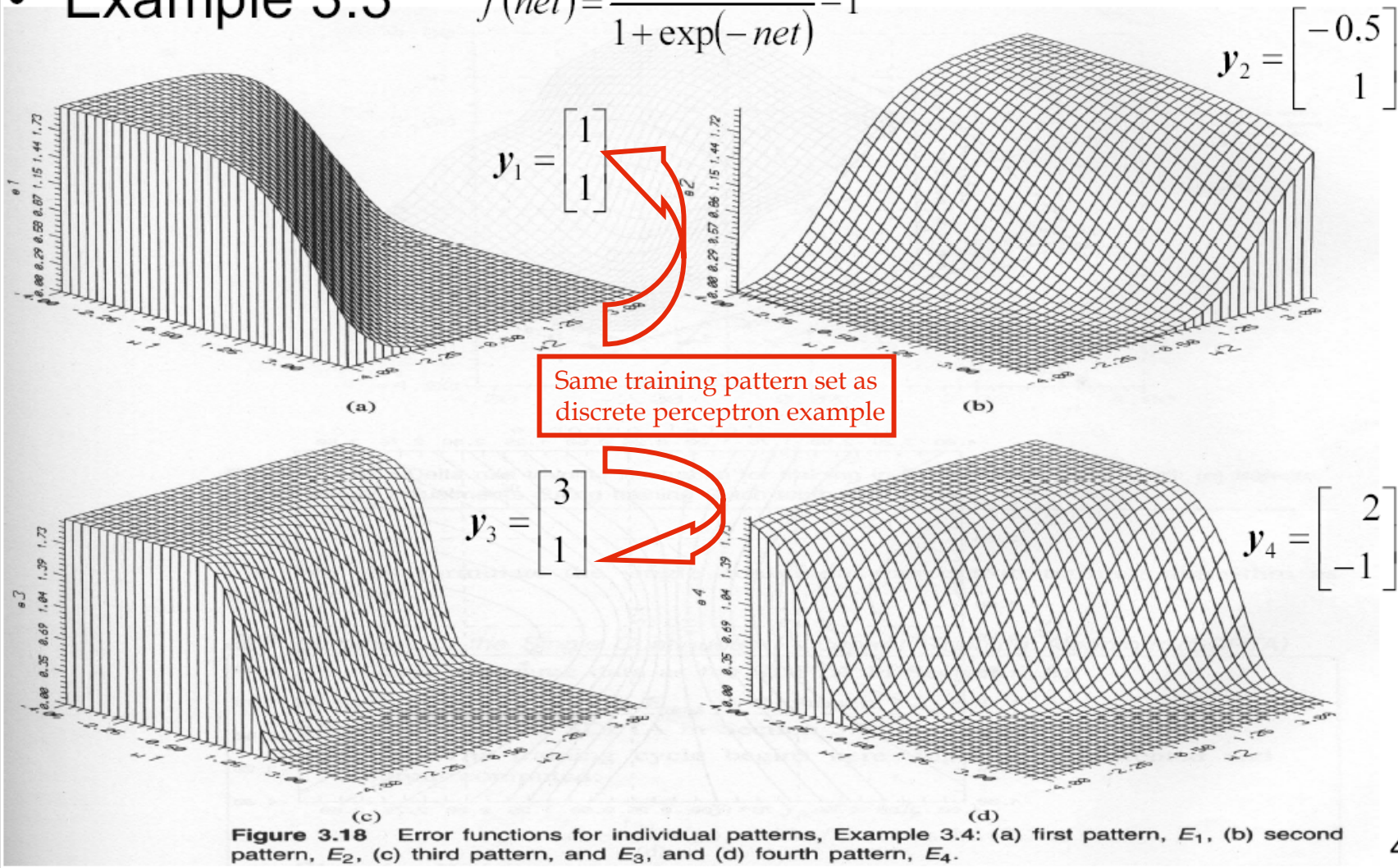


Figure 3.18 Error functions for individual patterns, Example 3.4: (a) first pattern, E_1, (b) second pattern, E_2, (c) third pattern, and E_3, and (d) fourth pattern, E_4.

Continuous Perceptron Training Algorithm…

$$E_k = \frac{1}{2}\left\{ d^k - \left[\frac{2}{1+\exp(-\lambda\, net^k)} - 1 \right] \right\}^2$$

$$E_1(\mathbf{w}) = \frac{1}{2}\left\{ 1 - \left[\frac{2}{1+\exp\left[-\lambda\,(w_1 + w_2)\right]} - 1 \right] \right\}^2$$

$\lambda = 1$ and reducing the terms simplifies this expression to the following form

$$E_1(\mathbf{w}) = \frac{2}{\left[1+\exp(w_1 + w_2)\right]^2}$$

similarly

$$E_2(\mathbf{w}) = \frac{2}{\left[1+\exp(0.5w_1 - w_2)\right]^2}$$

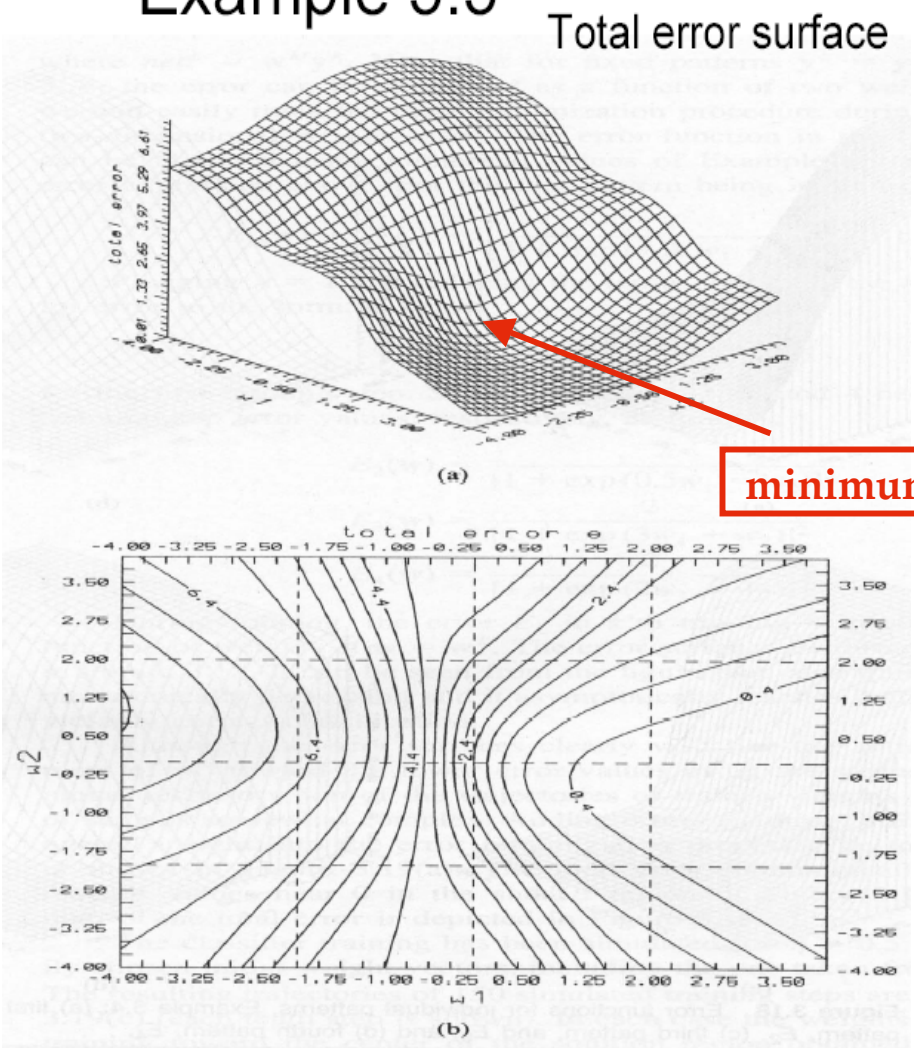
$$E_3(\mathbf{w}) = \frac{2}{\left[1+\exp(3w_1 + w_2)\right]^2}$$

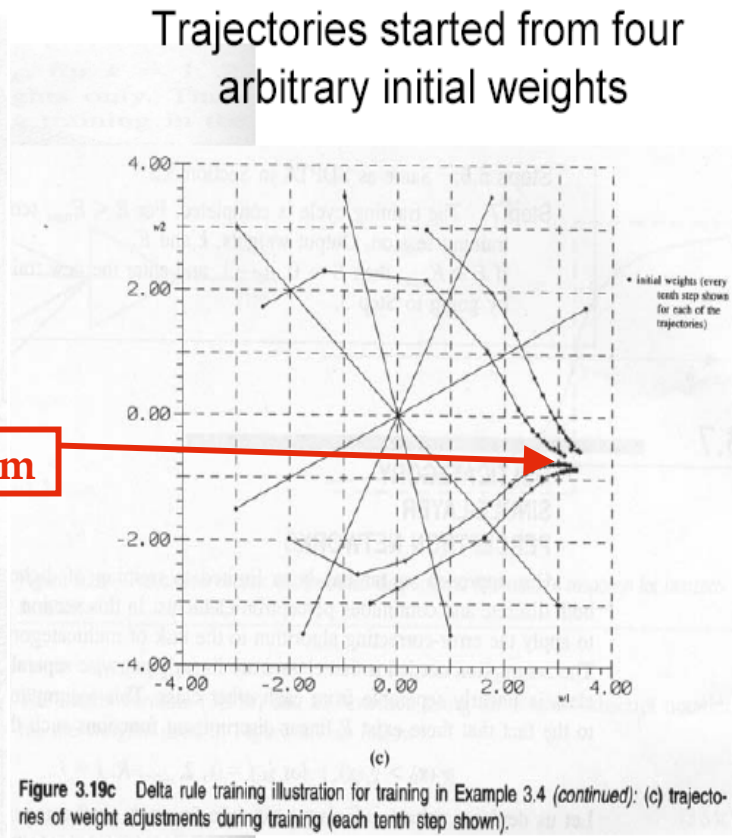
$$E_4(\mathbf{w}) = \frac{2}{\left[1+\exp(2w_1 - w_2)\right]^2}$$

These error surfaces are as shown on the previous slide.

Continuous Perceptron Training Algorithm…

- Example 3.3





minimum

Figure 3.19c Delta rule training illustration for training in Example 3.4 *(continued)*: (c) trajectories of weight adjustments during training (each tenth step shown).

Figure 3.19a,b Delta rule training illustration for training in Example 3.4: (a) total error surface, (b) total error contour map.

Mutlicategory SLP

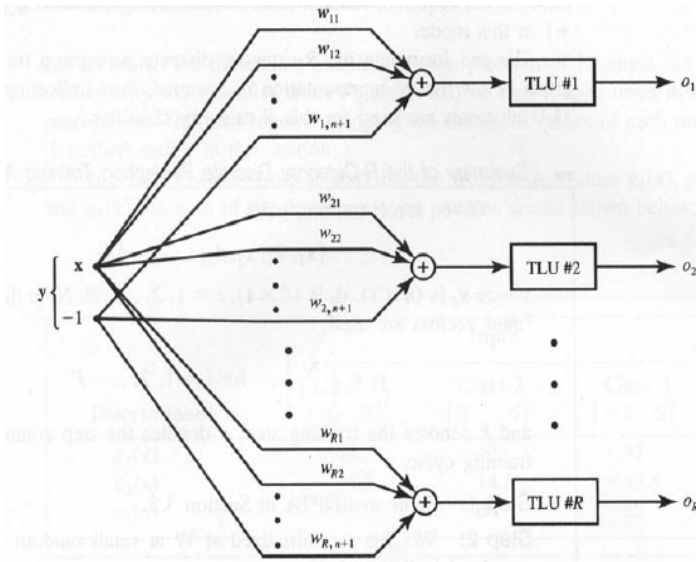


Figure 3.21 *R*-category linear classifier using *R* discrete perceptrons.

Multi-category Single layer Perceptron nets

- Treat the last fixed component of input pattern vector as the neuron activation threshold…. $T=w_{n+1}$

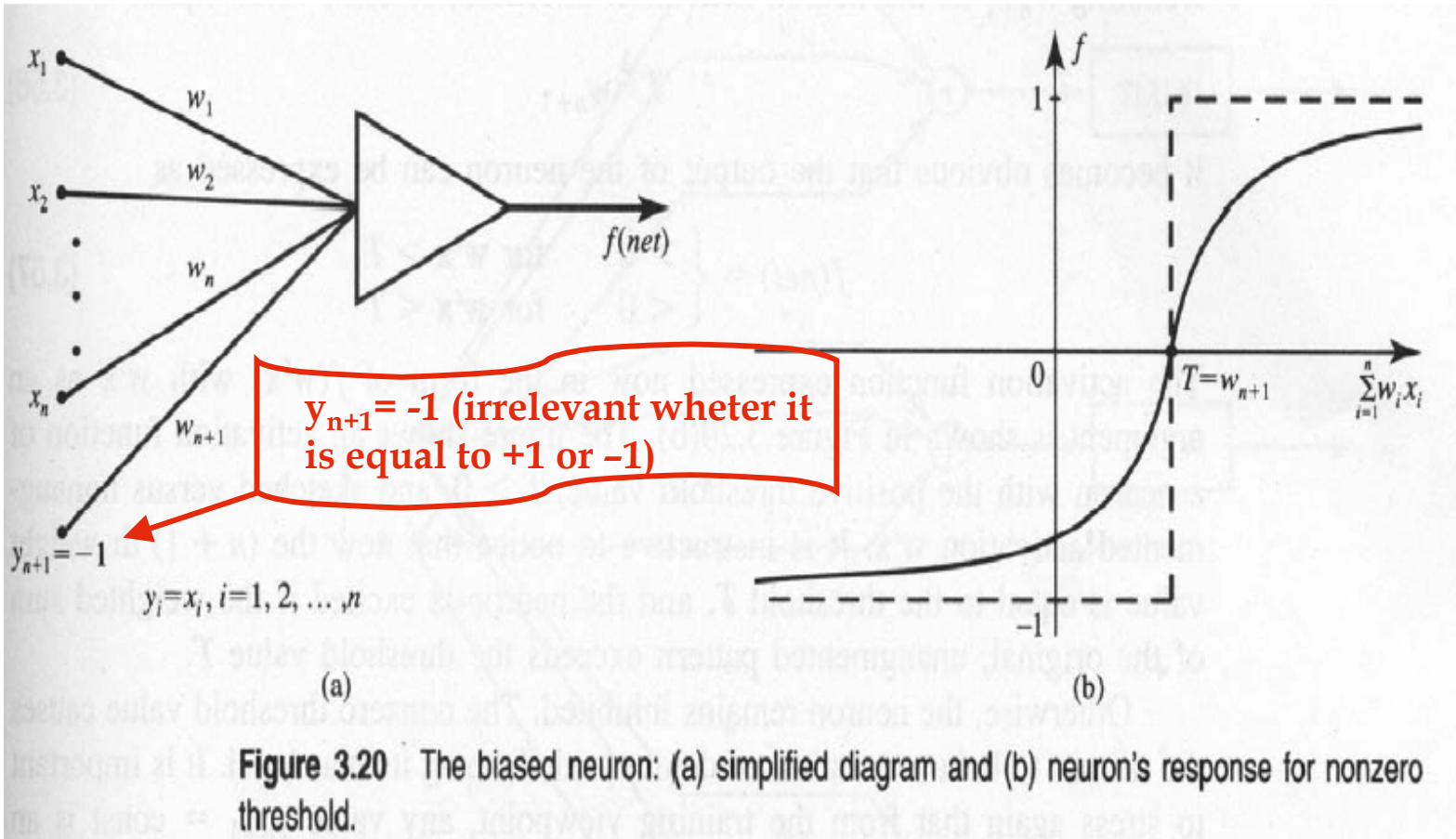


Figure 3.20 The biased neuron: (a) simplified diagram and (b) neuron's response for nonzero threshold.

Multi-category Single layer Perceptron nets…

- R-category linear classifier using R discrete **bipolar perceptrons**
  - Goal: The i-th TLU response of +1 is indicative of class i and all other TLU respond with -1

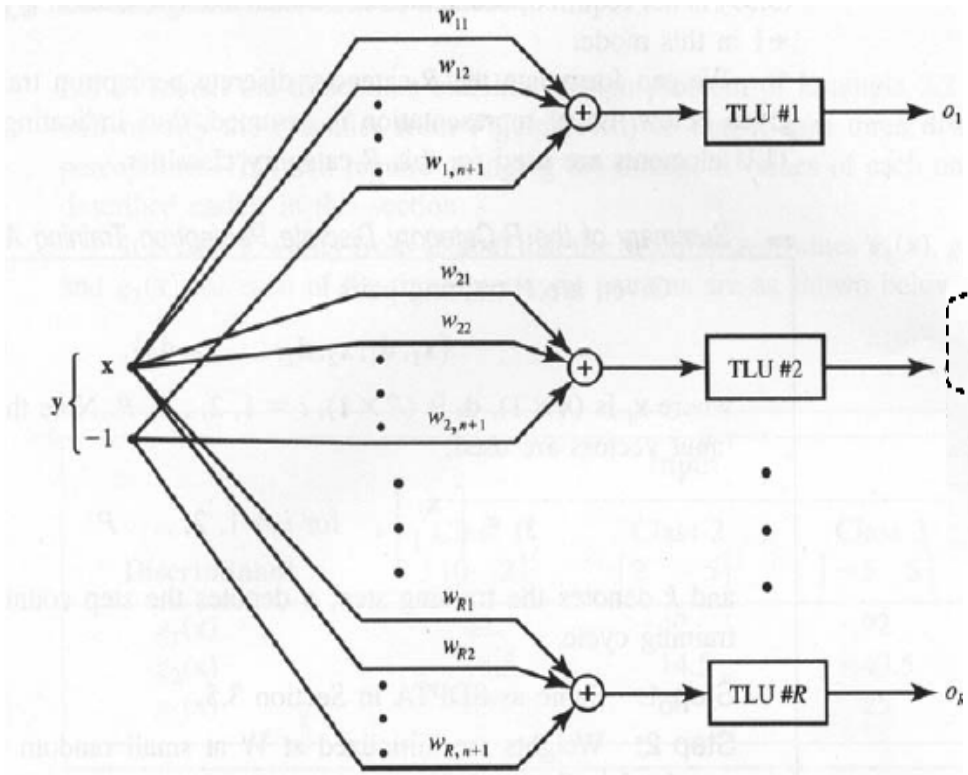


Figure 3.21 R-category linear classifier using R discrete perceptrons.

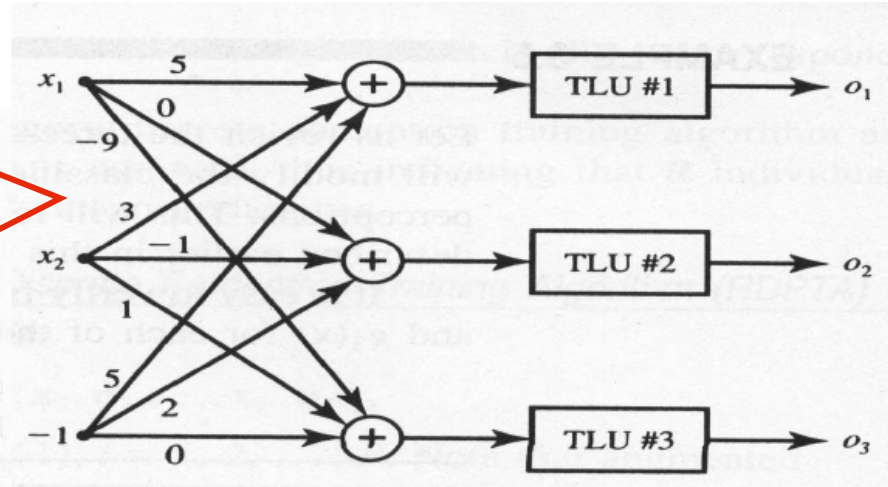
$$\hat{\boldsymbol{w}}_i = \boldsymbol{w}_i + \frac{1}{2} c \cdot (d_i - o_i)\boldsymbol{y}$$

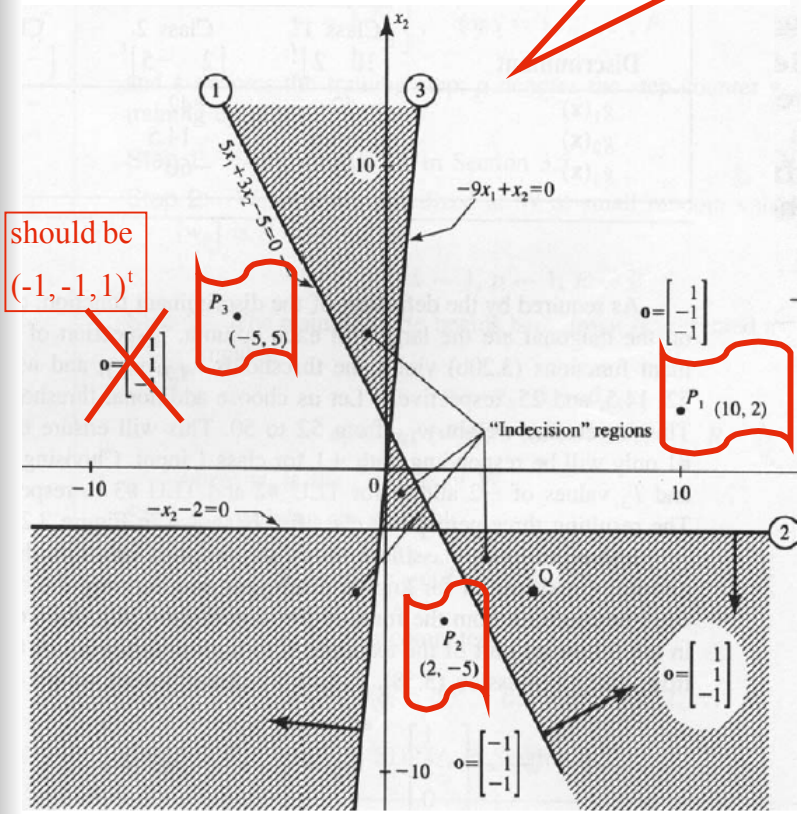
$$d_i = 1, d_j = -1, \text{for } j = 1,2,..,R,\ j \neq i$$

For "local representation"

Multi-category Single layer Perceptron nets…

- Example 3.5





Indecision regions = regions where no class membership of an input pattern can be uniquely determined based on the response of the classifier (patterns in shaded areas are not assigned any reasonable classification. E.g. point Q for which $o=[1\ 1\ -1]^t$ => indecisive response). However no patterns such as Q have been used for training in the example.

Multi-category Single layer Perceptron nets…

For $c = 1$ and $\mathbf{w}_1^1 = [1 \quad -2 \quad 0]^t$ $\mathbf{w}_2^1 = [0 \quad -1 \quad 2]^t$ and $\mathbf{w}_3^1 = [1 \quad 3 \quad -1]^t$

- **Step 1**: Pattern $\mathbf{y}_1$ is input

$$\text{sgn}\left(\begin{bmatrix} 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} \right) = 1$$

$$\text{sgn}\left(\begin{bmatrix} 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} \right) = -1$$

$$\text{sgn}\left(\begin{bmatrix} 1 & 3 & -1 \end{bmatrix} \begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} \right) = 1^*$$

Since the only incorrect response is provided by TLU3, we have

$$\mathbf{w}_1^2 = \mathbf{w}_1^1$$

$$\mathbf{w}_2^2 = \mathbf{w}_2^1$$

$$\mathbf{w}_3^2 = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix} - \begin{bmatrix} 10 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} -9 \\ 1 \\ 0 \end{bmatrix}$$

Multi-category Single layer Perceptron nets…

- **Step 2**: Pattern $\mathbf{y}_2$ is input

$$\operatorname{sgn}\left(\begin{bmatrix}1 & -2 & 0\end{bmatrix}\begin{bmatrix}2\\-5\\-1\end{bmatrix}\right)=1^{*}$$

$$\operatorname{sgn}\left(\begin{bmatrix}0 & -1 & 2\end{bmatrix}\begin{bmatrix}2\\-5\\-1\end{bmatrix}\right)=1$$

$$\operatorname{sgn}\left(\begin{bmatrix}-9 & 1 & 0\end{bmatrix}\begin{bmatrix}2\\-5\\-1\end{bmatrix}\right)=-1$$

$$\mathbf{w}_1^3=\begin{bmatrix}1\\2\\0\end{bmatrix}-\begin{bmatrix}2\\-5\\-1\end{bmatrix}=\begin{bmatrix}-1\\3\\1\end{bmatrix}$$

$$\mathbf{w}_2^3=\mathbf{w}_2^2$$

$$\mathbf{w}_3^3=\mathbf{w}_3^2$$

Multi-category Single layer Perceptron nets…

- **Step 3**: Pattern $\mathbf{y}_3$ is input

$$\text{sgn}\left(\mathbf{w}_1^{3t}\mathbf{y}_3\right) = 1 *$$
$$\text{sgn}\left(\mathbf{w}_2^{3t}\mathbf{y}_3\right) = -1$$
$$\text{sgn}\left(\mathbf{w}_3^{3t}\mathbf{y}_3\right) = 1$$

$$\mathbf{w}_1^4 = \begin{bmatrix} 4 \\ -2 \\ 2 \end{bmatrix}$$
$$\mathbf{w}_2^4 = \mathbf{w}_2^3$$
$$\mathbf{w}_3^4 = \mathbf{w}_3^3$$

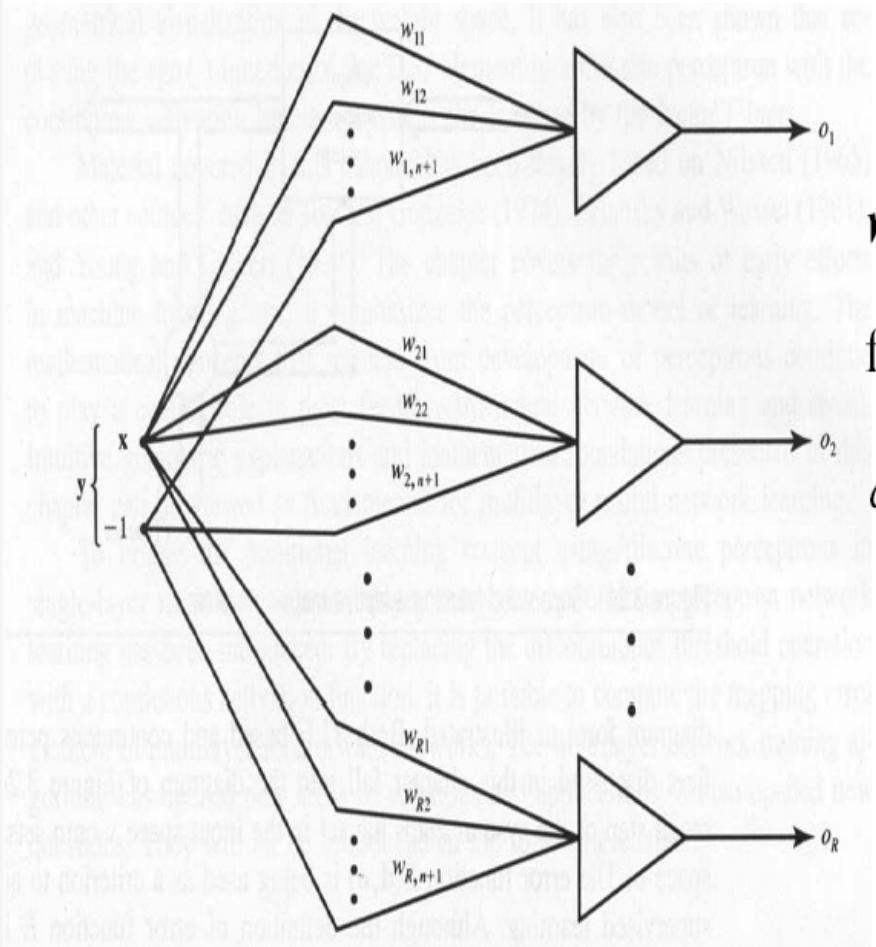
One can verify that the only adjusted weights from now on are those of TLU1

- During the second cycle:

$$\mathbf{w}_1^5 = \mathbf{w}_1^4$$
$$\mathbf{w}_1^6 = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}$$
$$\mathbf{w}_1^7 = \begin{bmatrix} 7 \\ -2 \\ 4 \end{bmatrix}$$
$$\mathbf{w}_1^8 = \mathbf{w}_1^7$$
$$\mathbf{w}_1^9 = \begin{bmatrix} 5 \\ 3 \\ 5 \end{bmatrix}$$

Multi-category Single layer Perceptron nets…

- R-category linear classifier using R **continuous bipolar perceptrons**



$$\hat{\boldsymbol{w}}_i = \boldsymbol{w}_i + \frac{1}{2}\eta \cdot \lambda \left(d_i - o_i\right)\left(1 - o_i^2\right)\boldsymbol{y}$$

$$\text{for } i = 1,2,...,R$$

$$d_i = 1,\ d_j = -1, \text{ for } j = 1,2,..,R,\ j \neq i$$

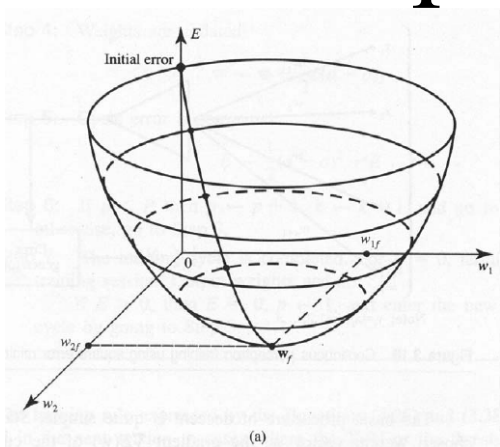
Figure 3.23 *R*-category linear classifier using continuous perceptrons.

Comparison between Perceptron and Bayes' Classifier

- Perceptron operates on the promise that the patterns to be classified are *linear separable* (otherwise the training algorithm will oscillate), while Bayes classifier can work on nonseparable patterns
- Bayes classifier minimizes the probability of misclassification which is independent of the underlying distribution
- Bayes classifier is a linear classifier on the assumption of Gaussianity
- The perceptron is non-parametric, while Bayes classifier is parametric (its derivation is contingent on the assumption of the underlying distributions)
- The perceptron is adaptive and simple to implement
- the Bayes' classifier could be made adaptive but at the expense of increased storage and more complex computations

APPENDIX A

Unconstrained Optimization Techniques



Unconstrained Optimization Techniques

http://ai.kaist.ac.kr/~jkim/cs679/
Haykin, Chapter 3

- Cost function $E(\mathbf{w})$
  - continuously differentiable
  - a measure of how to choose **w** of an adaptive filtering algorithm so that it behaves in an optimum manner
- we want to find an optimal solution $\mathbf{w}^*$ that minimize $E(\mathbf{w})$ $\quad \nabla E(\vec{w}^*) = 0$
  - local iterative descent :

    starting with an initial guess denoted by **w**(0), generate a sequence of weight vectors **w**(1), **w**(2), …, such that the cost function $E(\mathbf{w})$ is reduced at each iteration of the algorithm, as shown by

    $$E(\mathbf{w}(n+1)) < E(\mathbf{w}(n))$$
  - Steepest Descent, Newton's, Gauss-Newton's methods

Method of Steepest Descent

- Here the successive adjustments applied to **w** are in the direction of steepest descent, that is, in a direction opposite to the grad(E(**w**))

$$\mathbf{w}(n+1) = \mathbf{w}(n) - a\,\mathbf{g}(n)$$

  a : small positive constant called *step size* or *learning-rate* parameter.

  g(n) : grad(E(**w**))

- The method of steepest descent converges to the optimal solution **w*** slowly
- The learning rate parameter a has a profound influence on its convergence behavior
  - overdamped, underdamped, or even unstable(diverges)

Newton's Method

- Using a second-order Taylor series expansion of the cost function around the point $\mathbf{w}(n)$

$$\Delta E(\mathbf{w}(n)) = E(\mathbf{w}(n+1)) - E(\mathbf{w}(n))$$

$$\sim \mathbf{g}^T(n)\,\Delta\mathbf{w}(n) + \frac{1}{2}\,\Delta\mathbf{w}^T(n)\,\mathbf{H}(n)\,\Delta\mathbf{w}(n)$$

where $\Delta\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n)$,

$\mathbf{H}(n)$: Hessian matrix of $E(n)$

We want $\Delta\mathbf{w}^*(n)$ that minimize $\Delta E(\mathbf{w}(n))$ so differentiate $\Delta E(\mathbf{w}(n))$ with respect to $\Delta\mathbf{w}(n)$:

$$\mathbf{g}(n) + \mathbf{H}(n)\,\Delta\mathbf{w}^*(n) = 0$$

so,

$$\Delta\mathbf{w}^*(n) = -\mathbf{H}^{-1}(n)\,\mathbf{g}(n)$$

Newton's Method…

Finally,

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

$$= \mathbf{w}(n) - \mathbf{H}^{-1}(n)\,\mathbf{g}(n)$$

- Newton's method converges quickly asymptotically and does not exhibit the zigzagging behavior
- the Hessian $\mathbf{H}(n)$ has to be a positive definite matrix for all n

Gauss-Newton Method

- The Gauss-Newton method is applicable to a cost function

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

- Because the error signal e(i) is a function of **w**, we linearize the dependence of e(i) on **w** by writing

$$e'(i,\vec{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \vec{w}}\right]^T_{\vec{w}=\vec{w}(n)} (\vec{w} - \vec{w}(n))$$

- Equivalently, by using matrix notation we may write

$$\vec{e}'(n,\vec{w}) = \vec{e}(n) + J(n)(\vec{w} - \vec{w}(n))$$

Gauss-Newton Method…

where **J**(n) is the n-by-m Jacobian matrix of **e**(n) (see bottom of this slide)

- We want updated weight vector **w**(n+1) defined by

$$\vec{w}(n+1) = \arg\min_{\vec{w}} \left\{ \frac{1}{2} \left\| \vec{e}'(n, \vec{w}) \right\|^2 \right\}$$

- simple algebraic calculation tells…

$$\frac{1}{2}\left\|\vec{e}'(n,\vec{w})\right\|^2 = \frac{1}{2}\left\|\vec{e}(n)\right\|^2 + \vec{e}^T(n)J(n)(\vec{w}-\vec{w}(n)) + \frac{1}{2}(\vec{w}-\vec{w}(n))^T J^T(n)J(n)(\vec{w}-\vec{w}(n))$$

Now differentiate this expression with respect to **w** and set the result to 0, we obtain

$$J = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \cdots & \frac{\partial e(1)}{\partial w_\alpha} & \cdots & \frac{\partial e(1)}{\partial w_M} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ \frac{\partial e(k)}{\partial w_1} & \cdots & \frac{\partial e(k)}{\partial w_\alpha} & \cdots & \frac{\partial e(k)}{\partial w_M} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \cdots & \frac{\partial e(n)}{\partial w_\alpha} & \cdots & \frac{\partial e(n)}{\partial w_M} \end{bmatrix}$$

Gauss-Newton Method…

$$J^T(n)\vec{e}(n) + J^T(n)J(n)(\vec{w} - \vec{w}(n)) = 0$$

Thus we get

$$\vec{w}(n+1) = \vec{w}(n) - (J^T(n)J(n))^{-1} J^T(n)\vec{e}(n)$$

- To guard against the possibility that the matrix product $\mathbf{J}^T(n)\mathbf{J}(n)$ is singular, the customary practice is

$$\vec{w}(n+1) = \vec{w}(n) - (J^T(n)J(n) + \delta I)^{-1} J^T(n)\vec{e}(n)$$

where δ is a small positive constant.

This modification effect is progressively reduced as the number of iterations, n, is increased.

Linear Least-Squares Filter

- The single neuron around which it is built is linear
- The cost function consists of the sum of error squares
- Using $y(i) = \mathbf{x}^T(i)\mathbf{w}(i)$ and $e(i) = d(i) - y(i)$ the error vector is

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

- Differentiating with respect to $\mathbf{w}(n)$

$$\nabla \mathbf{e}(n) = -\mathbf{X}^T(n)$$

correspondingly,

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

- From Gauss-Newton method, (eq. 3.22)

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

LMS Algorithm

- Based on the use of instantaneous values for cost function :

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

- Differentiating with respect to $\mathbf{w}$,

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = e(n) \frac{\partial e(n)}{\partial \mathbf{w}}$$

- The error signal in LMS algorithm :

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

hence,

$$\frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\mathbf{x}(n) \quad \text{so,} \quad \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)e(n)$$

LMS Algorithm …

- Using $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}(n)}$ as an *estimate* for the gradient vector,

$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n)$$

- Using this for the gradient vector of steepest descent method, LMS algorithm as follows :

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

  - η : learning-rate parameter

- The inverse of η is a *measure* of the memory of the LMS algorithm
  - When η is small, the adaptive process progress slowly, more of the past data are remembered and a more accurate filtering action'

LMS Characteristics

- LMS algorithm produces an *estimate* of the weight vector
  - Sacrifice a distinctive feature
    - Steepest descent algorithm : $\mathbf{w}(n)$ follows a well-defined trajectory
    - LMS algorithm : $\hat{\mathbf{w}}(n)$ follows a random trajectory
  - Number of iterations goes infinity, $\hat{\mathbf{w}}(n)$ performs a random walk
- But importantly, LMS algorithm does *not* require knowledge of the statistics of the environment

Convergence Consideration

- Two distinct quantities, η and $\mathbf{x}(n)$ determine the convergence
  - the user supplies η , and the selection of $\mathbf{x}(n)$ is important for the LMS algorithm to converge
- *Convergence of the mean*

$$E[\hat{\mathbf{w}}(n)] \to \mathbf{w}_0 \quad \text{as} \quad n \to \infty$$

  - This is not a practical value
- *Convergence in the mean square*

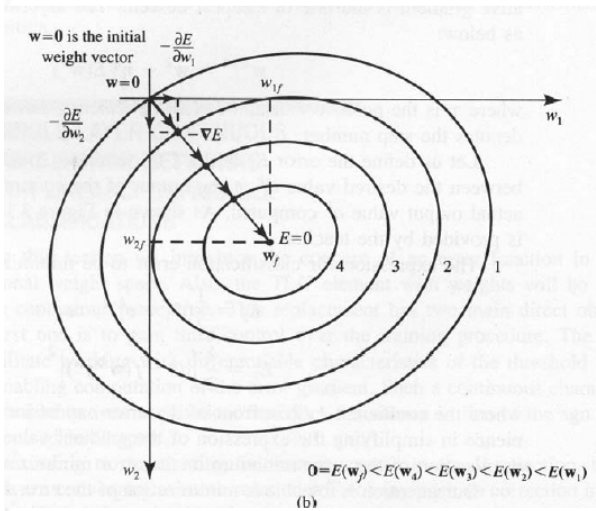
$$E[e^2(n)] \to \text{constant} \quad \text{as} \quad n \to \infty$$

- Convergence condition for LMS algorithm in the mean square

$$0 < \eta < \frac{2}{\text{sum of mean-square values of the sensor inputs}}$$

APPENDIX B

Perceptron Convergence Proof

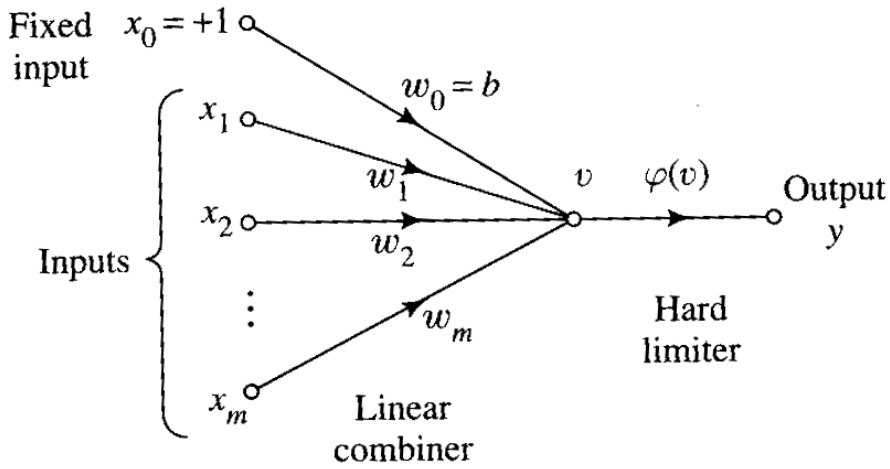


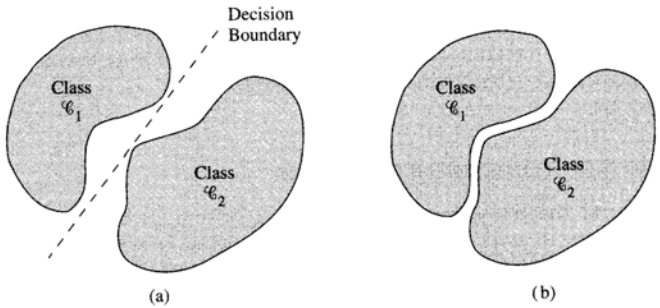
Perceptron Convergence Proof

Consider the following perceptron:

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

$$= \mathbf{w}^T(n)\mathbf{x}(n)$$





$\mathbf{w}^T\mathbf{x} > 0$ for every input vector $\mathbf{x}$ belonging to class C_1

$\mathbf{w}^T\mathbf{x} \leq 0$ for every input vector $\mathbf{x}$ belonging to class C_2

Perceptron Convergence Proof…

- The algorithm for the weight adjustment for the perceptron
  - if x(n) is correctly classified no adjustments to **w**

$$\mathbf{w}(n+1) = \mathbf{w}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) \le 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_2$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_1$$

  - otherwise

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_2$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T\mathbf{x}(n) \le 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } C_1$$

  - learning rate parameter $\eta(n)$ controls adjustment applied to weight vector

Perceptron Convergence Proof

For $\eta(n) = 1$ and $\mathbf{w}(0) = \mathbf{0}$

Suppose the perceptron **incorrectly** classifies the vectors $\mathbf{x}(1), \mathbf{x}(2), \ldots$ such that

$$\mathbf{w}^T \mathbf{x}(n) \le 0 \text{ so that : } \mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n)$$

But since $\eta = 1 \Rightarrow$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad \text{for } \mathbf{x}(\text{n}) \text{ belonging to } \text{C}_1$$

Since $\mathbf{w}(0) = \mathbf{0}$, iteratively we find $\mathbf{w}(n+1)$

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \ldots + \mathbf{x}(n) \qquad \text{(Bl)}$$

Since the classes C1 and C2 are assumed to be linearly separable, there exists a solution $\mathbf{w}_0$ for which $\mathbf{w}^T\mathbf{x}(n)>0$ for the vectors $\mathbf{x}(1), \ldots \mathbf{x}(n)$ belonging to the subset H_1 (subset of training vectors that belong to class C1).

Perceptron Convergence Proof

For a fixed solution $\mathbf{w}_0$, we may then define a positive number α as

$$\alpha = \min_{\mathbf{x}(n) \in H_1} \mathbf{w}_0^T \mathbf{x}(n) \qquad (B2)$$

Hence equation (B1) above implies

$$\mathbf{w}_0^{\mathrm{T}} \mathbf{w}(n+1) = \mathbf{w}_0^{\mathrm{T}} \mathbf{x}(1) + \mathbf{w}_0^{\mathrm{T}} \mathbf{x}(2) + \ldots + \mathbf{w}_0^{\mathrm{T}} \mathbf{x}(n)$$

Using equation B2 above, (since each term is greater or equal than α), we have

$$\mathbf{w}_0^{\mathrm{T}} \mathbf{w}(n+1) \geq n\alpha$$

Now we use the Cauchy-Schwartz inequality:

$$(\mathbf{a}.\mathbf{b})^2 \leq \|\mathbf{a}\|^2 \|\mathbf{b}\|^2 \qquad \text{or}$$

$$\|\mathbf{a}\|^2 \geq \frac{(\mathbf{a}.\mathbf{b})^2}{\|\mathbf{b}\|^2} \qquad \text{for } \|\mathbf{b}\|^2 \neq 0$$

Perceptron Convergence Proof

This implies that:

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|\mathbf{w}_0\|^2} \qquad (B3)$$

Now let's follow another development route (notice index k)

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \quad \text{for } k = 1, \ldots, n \quad \text{and } \mathbf{x}(k) \in H_1$$

By taking the squared Euclidean norm of both sides, we get:

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k)$$

But under the assumption the the perceptron **incorrectly** classifies an input vector $\mathbf{x}(k)$ belonging to the subset H_1, we have $\mathbf{w}^T(k)\mathbf{x}(k) < 0$ and hence:

$$\|\mathbf{w}(k+1)\|^2 \leq \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2$$

Perceptron Convergence Proof

Or equivalently,

$$\|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \le \|\mathbf{x}(k)\|^2; \quad k = 1,\ldots n$$

Adding these inequalities for $k=1,\ldots n$, and invoking the initial condition $\mathbf{w}(0)=\mathbf{0}$, we get the following inequality:

$$\|\mathbf{w}(n+1)\|^2 \le \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2 \le n\beta \qquad (B4)$$

Where β is a positive number defined by;

$$\beta = \max_{\mathbf{x}(k)\in H_1} \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2$$

Eq. B4 states that the squared Euclidean norm of $\mathbf{w}(n+1)$ grows at most linearly with the number of iterations n.

Perceptron Convergence Proof

The second result of B4 is clearly in conflict with Eq. B3.

- Indeed, we can state that n cannot be larger than some value $n_{\max}$ for which Eq. B3 and B4 are both satisfied with the equality sign. That is $n_{\max}$ is the solution of the eq.

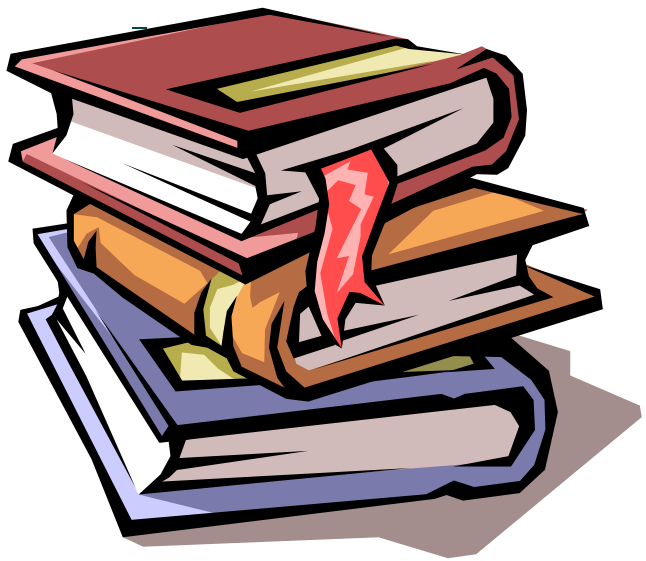
$$\frac{n_{\max}^2 \alpha^2}{\|\mathbf{w}_0\|^2} = n_{\max}\beta$$

- Solving for $n_{\max}$ given a solution $\mathbf{w}_0$, we find that

$$n_{\max} = \frac{\beta \|\mathbf{w}_0\|^2}{\alpha^2}$$

We have thus proved that for $\eta(n)=1$ for all n, and for $\mathbf{w}(0)=\mathbf{0}$, given that a sol' vector $\mathbf{w}_0$ exists, the rule for adapting the synaptic weights of the perceptron must terminate after at most $n_{\max}$ iterations.

MORE READING



Suggested Reading.

- S. Haykin, “Neural Networks”, Prentice-Hall, 1999, chapter 3.
- L. Fausett, “Fundamentals of Neural Networks”, Prentice-Hall, 1994, Chapter 2.
- R. O. Duda, P.E. Hart, and D.G. Stork, “Pattern Classification”, 2nd edition, Wiley 2001. Appendix A4, chapter 2, and chapter 5.
- J.M. Zurada, “Introduction to Artificial Neural Systems”, West Publishing Company, 1992, chapter 3.

References:

These lecture notes were based on the references of the previous slide, and the following references

1. Berlin Chen Lecture notes: Normal University, Taipei, Taiwan, ROC. http://140.122.185.120
2. Ehud Rivlin, IIT: http://webcourse.technion.ac.il/236607/Winter2002-2003/en/ho.html
3. *Jin Hyung Kim, KAIST Computer Science Dept., CS679 Neural Network lecture notes* http://ai.kaist.ac.kr/~jkim/cs679/detail.htm
4. Dr John A. Bullinaria, Course Material, Introduction to Neural Networks, http://www.cs.bham.ac.uk/~jxb/inn.html