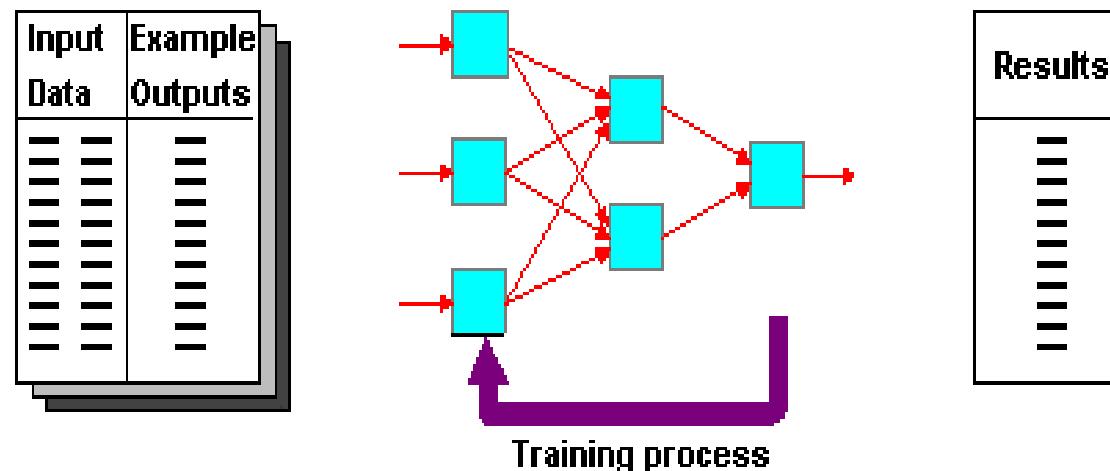


Linear Models

- Training and test data sets
- Training set: input & target
- Test set: only input



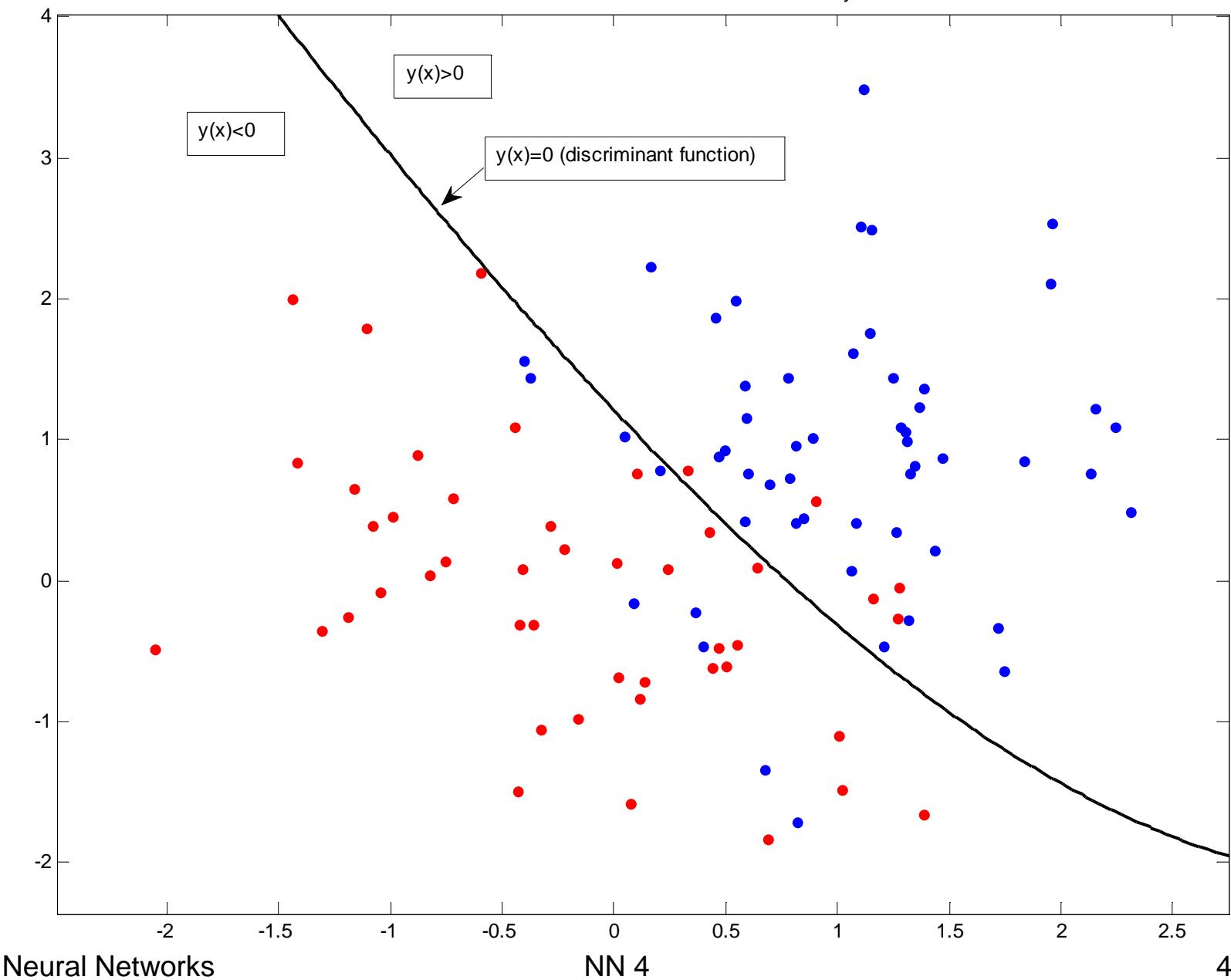
Single Layer Networks (Linear Models)

- Linear Discriminants
- Single Layer Perceptron
- Linear Separability and Cover's Theorem
- Learning Algorithms:
 - Perceptron Rule and Convergence Theorem
 - Pocket Algorithm
 - Least-Squares Method (Adaline)
 - Gradient Descent and Logistic Regression
 - Support Vector Machines
- Generalized Linear Discriminants
- Multi-class perceptron learning algorithm

Motivation

- Sometimes modeling $P(C_i|X)$ is impossible (very few data points, very high dimensionality of data, lack of background knowledge, etc.)
- Then one can try to find class boundaries directly, assuming certain form of discriminant functions and optimizing a suitable error measure
- *The simplest case of a boundary: a line (a hyperplane) single layer networks (Perceptron; Adaline; GLM; SVM)*
均为单层网络
- General case: multi-layer networks
(Multi-layer Perceptron; RBF-newtorks)

A Classification Problem and a Class Boundary



Discriminant Functions

- A *discriminant function for classes C_1 and C_2* is any function $y(x)$ such that an input vector x is assigned to class C_1 if $y(x) > 0$ (and to C_2 if $y(x) < 0$) [what to do with ties?]

- In case of N classes, we need N discriminant functions $y_1(x), y_2(x), \dots, y_N(x)$, such the k -th function $y_k(x)$ discriminates class C_k from all other classes, for $k=1, \dots, N$, i.e.,

x is assigned to class C_k iff $y_k(x) > y_n(x)$ for all $k \neq n$

N类需要N个判别函数，
函数值最大者对应类

- Examples:

- $y_k(x)=P(C_k|x)$ (posterior probability)
- $y_k(x)=P(x|C_k)*P(C_k)$ (normalization not needed)
- $y_k(x)=\ln(P(x|C_k))+\ln P(C_k)$ (monotonic transformation doesn't affect final decisions!!!)

判别函数
可能形式

- In case of two classes (C_1 and C_2) we often use $y(x)=y_1(x)-y_2(x)$ as a discriminant; then the sign of $y(x)$ decides on the class:
if $y(x)>0$ then x in C_1 else x in C_2

Linear Discriminant Functions

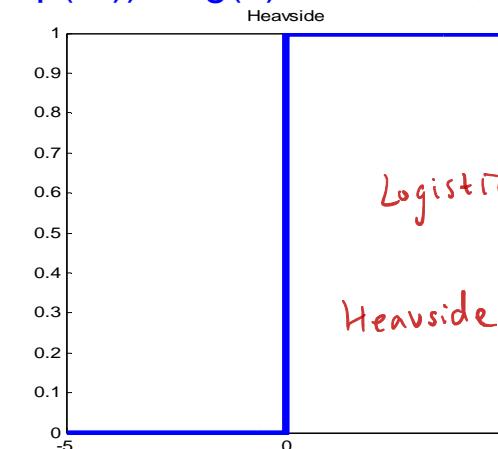
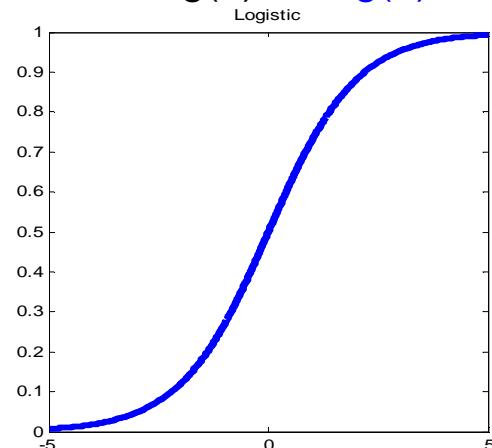
A *linear discriminant function in n-dimensional space* is a function $y(\mathbf{x})$ in the form:

$$y(\mathbf{x}) = w_0 + w_1 x_1 + \dots + w_n x_n = w_0 + \mathbf{w}^T \mathbf{x}$$

It defines a point (in \mathbb{R}^1), a line (in \mathbb{R}^2), a plane (in \mathbb{R}^3), a hyperplane (in \mathbb{R}^n), that splits the space into “positive” and “negative” half-spaces.

A composition of a linear discriminant $y(\mathbf{x})$ with a monotonic function $g(a)$ also defines a linear boundary: $g(y(\mathbf{x})) > 0$ iff $y(\mathbf{x}) > g^{-1}(0)$. 线性判别函数的单调变化仍为线性边界

Common choices of $g(a)$ are: $g(a) = 1/(1+\exp(-a))$ or $g(a) = 0$ for $a < 0$; 1 otherwise



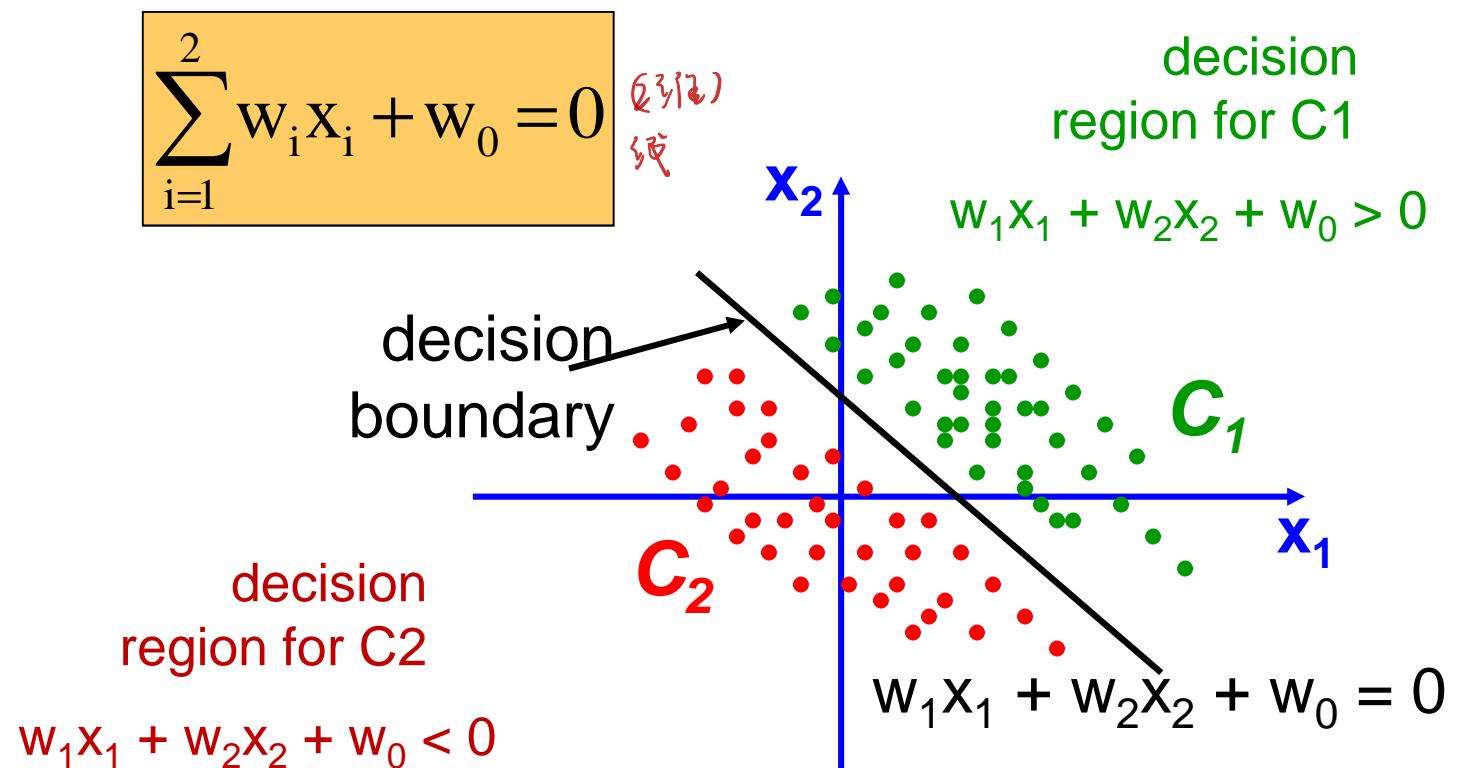
一般用的单调变换

$$\text{Logistic} \leftarrow g(a) = \frac{1}{1 + \exp(-a)}$$

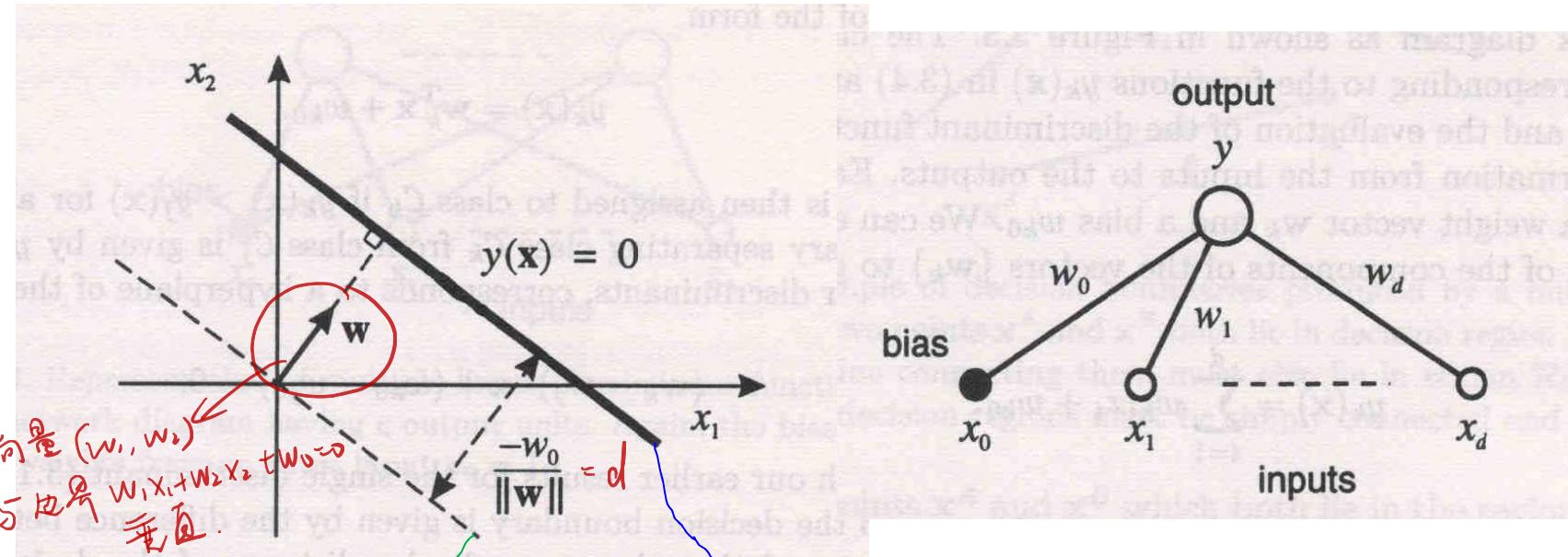
$$\text{Heaviside} \leftarrow g(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

Geometric View in 2D

The equation below describes a (hyper-)plane in the input space consisting of real valued 2D vectors. The plane splits the input space into two regions, each of them describing one class.



Linear Discriminant and Perceptron



$w_1x_1 + w_2x_2 = 0$ 垂直
 $\langle w_1, w_2 \rangle$ is perpendicular to the boundary!

Neural Networks

$$d = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}}$$

NN 4

8

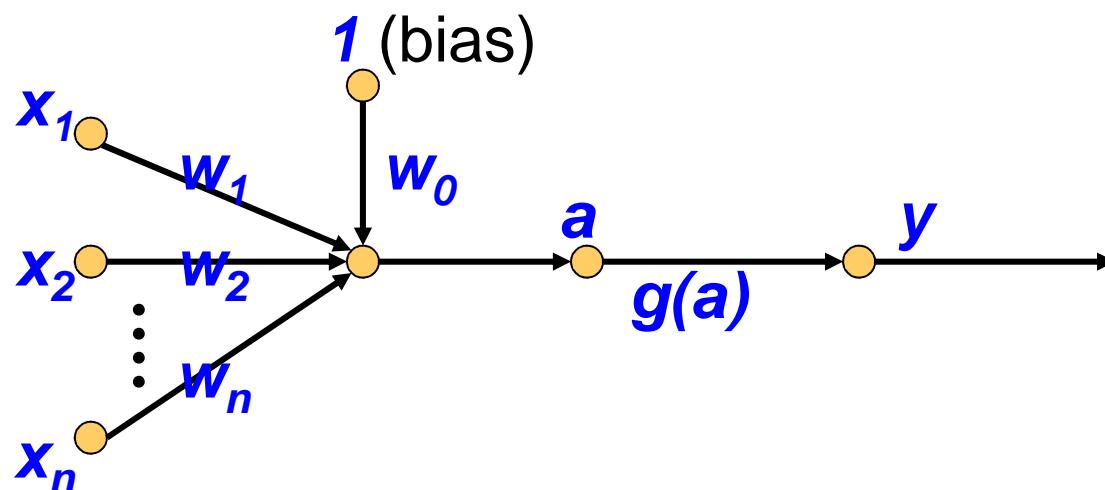
Perceptron: McCulloch-Pitts Model

The (McCulloch-Pitts) **perceptron** is a single node NN (or a single layer NN) with a non-linear function $g(a)$:

$$a = w_0 + \sum w_i x_i; \quad g(a) = \begin{cases} +1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}$$

a = 活性函数

非线性变换



Perceptron Training

- How can we train a perceptron for a classification task?
- We try to find suitable values for the **weights** in such a way that the training examples are correctly classified.
- Geometrically, we try to find a **hyper-plane** that separates the examples of the two classes.
线性可分(存在一个超平面可以划分两类)
- Two classes C_1 and C_2 are *linearly separable* if there exists a hyperplane that separates them.

Perceptron learning algorithm

initialize w randomly; 随机初始化 w .

while (there are misclassified training examples) 进入条件：存在错判.

Select a misclassified example (x, d) 选出错判样本和方向(相当于 y).

$$w_{\text{new}} = w_{\text{old}} + \eta d x;$$

感知器学习算法

end-while;

$\eta > 0$ is a learning rate parameter (step size); 步长率.

Motivation:

If x missclassified and $d=1 \Rightarrow wx$ should be bigger,

若 x 错判，且原 $d=1$ 时令 $y=-1$
则需加大 w

If x missclassified and $d=-1 \Rightarrow wx$ should be smaller

否则减小 w

$$(w + \eta d x)x = wx + \eta d x^2 \Rightarrow \text{迭代后的 } w,$$

Conventions:
• w is a row vector;
• x is a column vector
 $w = ()$
 $x = ()$
 x^2 is $x^T * x$

This rule does exactly what we want (x^2 is > 0) !!.

Main properties

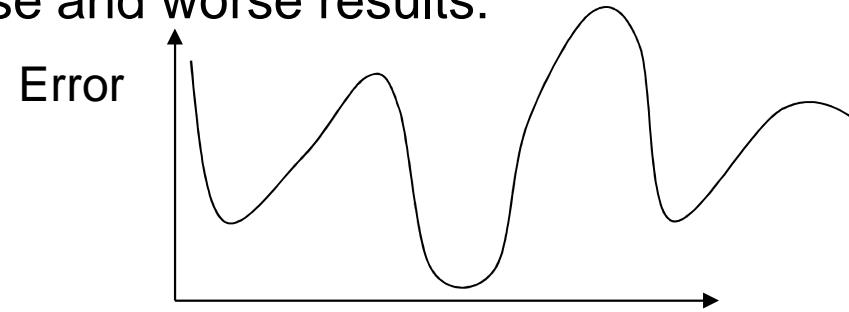
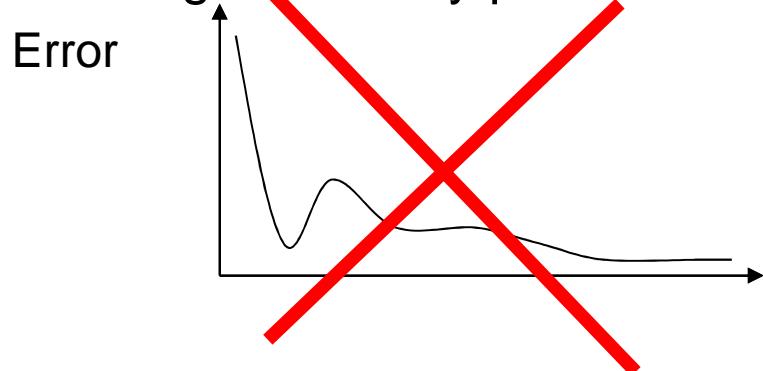
感知器收敛定理：若 C_1, C_2 线性可分，则感知算法在有限次迭代后可成功收敛

1) Perceptron Convergence Theorem:

If the classes C_1, C_2 are linearly separable (that is, there exists a hyper-plane that separates them) then the perceptron algorithm applied to $C_1 \cup C_2$ terminates successfully after a finite number of iterations. [*the value of the learning rate η is not essential*]

2) Bad Behavior:

If the classes C_1, C_2 are **not** linearly separable then the perceptron algorithm may produce worse and worse results:



Improvement: (Naive) Pocket Algorithm

$(w_0, w_1, \dots, w_2, \dots, w_N)$

0. Start with a random set of weights; 随机生成一组 w : 放入一个 pocket 中
put them in a 'pocket'
1. Select at random a pattern 随机选一个样本.
2. If it is misclassified then 若错判
 - 2A. apply the perceptron update rule 使用感知器算法
得到新权重
 - 2B. check whether new weights are better than those kept in the pocket; if so put new weights to the pocket (and remove the old ones) 检查新权重比 pocket 中的所有权重都更好?
是 → 替换
否 → 保留
3. goto 1.

Testing if new weights are better (less errors) is very expensive !!!

Gallant's Pocket Algorithm

内部的 pocket .
main idea: 仅计算正确
分类次数来判断权重好坏.

main idea: measure the quality of weights by counting the number of consecutive correct classifications ('current_run')

0. Start with a random set of weights; put them in the 'pocket'; 随机生成一袋权重 weights
 $best_run := 0$; $current_run := 0$; 初始化 .
1. Select a pattern at random 选最近一个样本
2. If it is misclassified then 其错判
apply the update rule; $current_run := 0$; 更新 w, 重置 mn = 0
else
 $current_run++$; +1
 if $current_run > best_run$ then 若 current-run > best-run .
 put new weights to the pocket; $best_run := current_run$ 挑新权重加入 .
3. Goto 1

Gallant's Pocket Algorithm with ratchet

0. Start with a random setting of weights; put it in the 'pocket'

1. $\text{best_run}:=0; \text{current_run}:=0;$

2. Select at random a pattern 

3. IF it is correctly classified THEN 

$\text{current_run}:=\text{current_run}+1$

IF $\text{best_run} < \text{current_run}$ AND [current weights
are better than those kept in the pocket] THEN

$\text{pocket}:=\text{current weights}; \text{best_run}=\text{current_run};$

ELSE

$\text{current_run}=0;$

update weights



口袋收敛定理：即使分类间隔不可分
也可以收敛，收敛到最优权重

Pocket convergence theorem

Pocket Convergence Theorem:

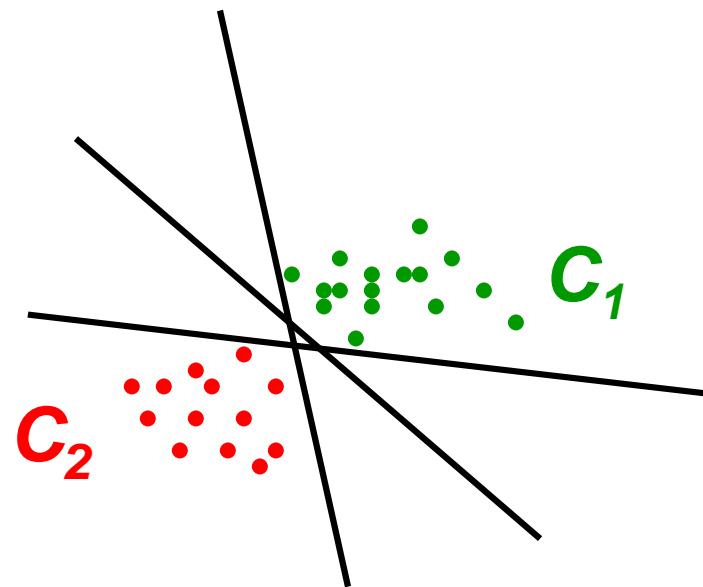
The pocket algorithm converges with probability 1 to optimal weights (even if sets are not separable!)

Practice:

- 1) The convergence rate is quite high
- 2) Pocket algorithm is better than the Perceptron algorithm
- 3) Both algorithms have very limited use
- 4) There may be many separating hyperplanes ...

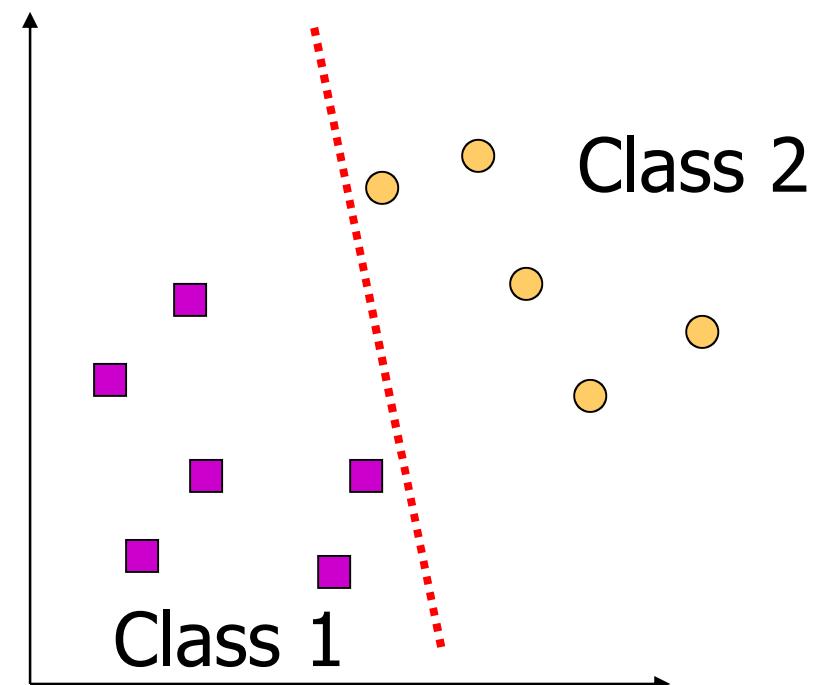
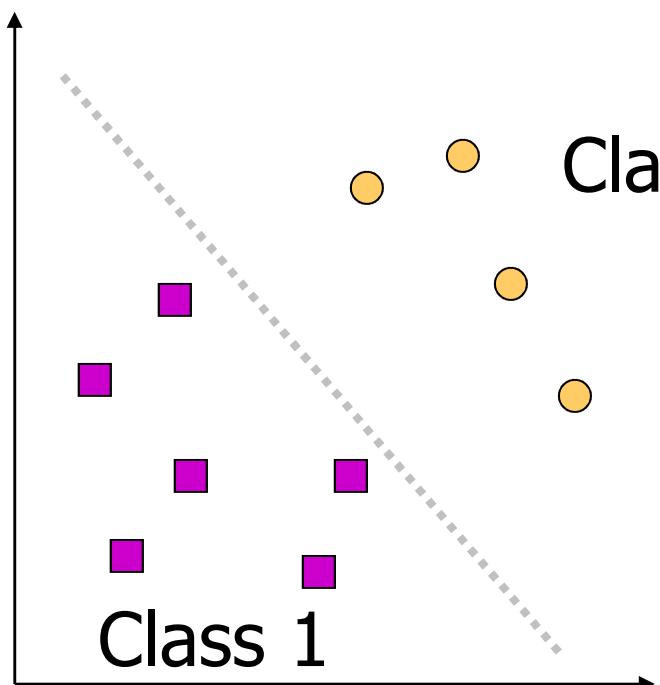
可能有多个超平面

Which separating hyperplane is best?



We are interested in accuracy on the test set!

Examples of Bad Decision Boundaries



Idea:

define a “continuous error measure” and try to minimize it !

Cover's Theorem (1965):

What is the chance that a **randomly labeled set of N points** (in general position) **in d -dimensional space, is linearly separable?**

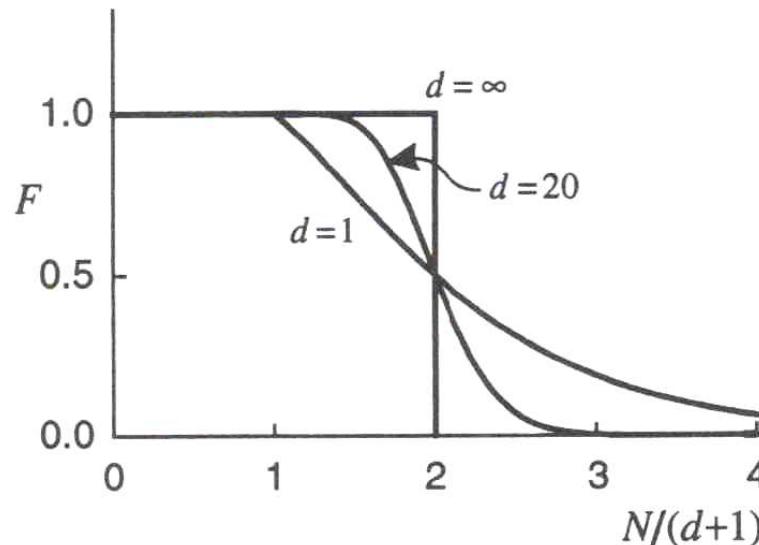


Figure 3.7. Plot of the fraction $F(N, d)$ of the dichotomies of N data points in d dimensions which are linearly separable, as a function of $N/(d + 1)$, for various values of d .

$$F(N, d) = \begin{cases} 1 & \text{when } N \leq d + 1 \\ \frac{1}{2^{N-1}} \sum_{i=0}^d \binom{N-1}{i} & \text{when } N \geq d + 1 \end{cases} \quad (3.30)$$

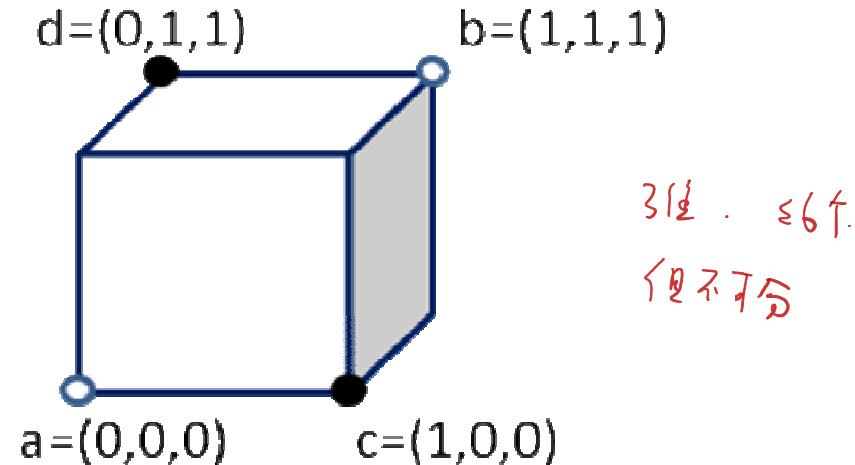
Cover's Theorem in highly dimensional spaces:

- 1) if the number of points in d-dimensional space is smaller than 2^d then they are almost always linearly separable 若 d 维空间中的点个数 $\leq 2^d$ 则基本上线性可分。
- 2) if the number of points in d-dimensional space is bigger than 2^d then they are almost always linearly non-separable 否则。非线性不可分。

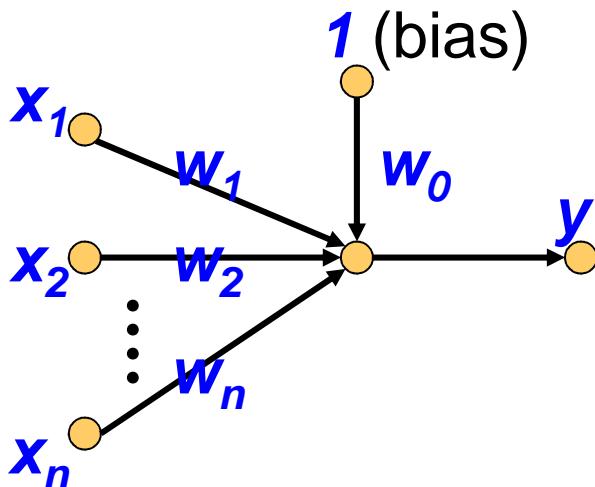
A quick check: 反例

Are points $\{a, b\}$ linearly separable from $\{c, d\}$?

How does it relate to the Cover's theorem?



Adaline: Adaptive Linear Element



不使用 0-1 转换函数.
不使用错判率为优化指标
使用误差平方和为优化指标.

$$y = w_0 + \sum w_i x_i$$

最后再
加入转换
函数

- Activation function $g(x)=x$ (identity)
- The desired outputs are -1's or 1's, the actual outputs (y 's) are "real numbers"
- Main idea: minimize the squared error:

$$Error = \sum_{Examples} (y_i - d_i)^2$$

样本的方向(1, -1)

NN 4

线性函数的值

Error function

on pattern i :

$$E(i) = (d_i - y_i)^2 \quad \text{误差. (有 n 个).}$$

on all patterns:

$$E = \sum (d_i - y_i)^2$$

But

$$y_i = w_0 + w_1 x_1 + \dots + w_k x_k,$$

so

$$E = \sum (d_i - (w_0 + w_1 x_1 + \dots + w_k x_k))^2$$

thus

E is a function of w_0, w_1, \dots, w_k .

How can we find the minimum of $E(w_0, w_1, \dots, w_k)$?

By the gradient descent algorithm ...

Gradient Descent Algorithm

How to find a minimum of a function $f(x,y)$?

1. Start with an arbitrary point (x_0, y_0)
2. Find a direction in which f is decreasing most rapidly

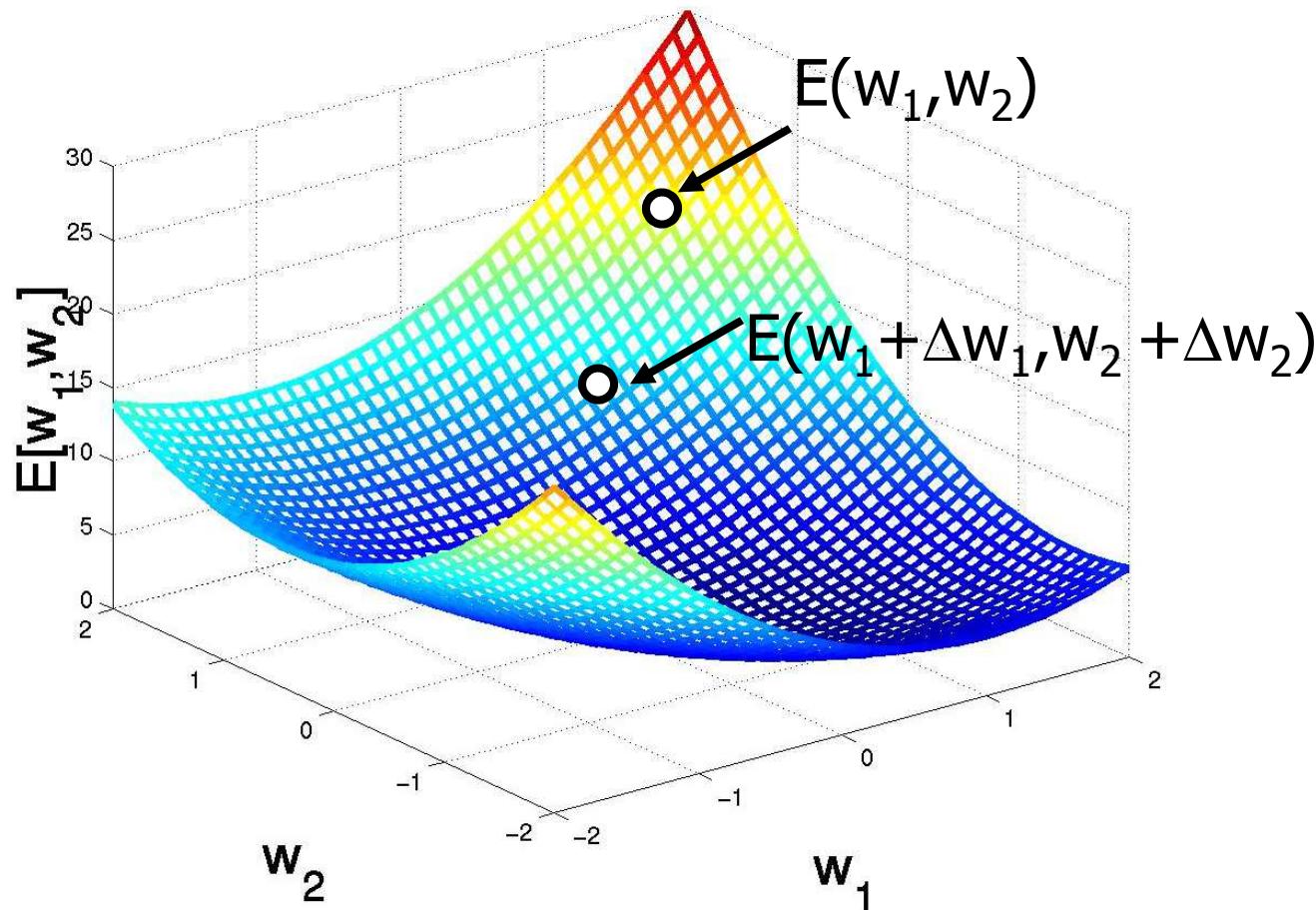
$$-\left[\frac{\partial f(x_0, y_0)}{\partial x}; \frac{\partial f(x_0, y_0)}{\partial y} \right]$$

3. Make a small step in this direction

$$(x_0, y_0) = (x_0, y_0) - \eta \left[\frac{\partial f(x_0, y_0)}{\partial x}; \frac{\partial f(x_0, y_0)}{\partial y} \right]$$

4. Repeat the whole process

Gradient Descent



Adaline: Gradient Descent

- Find w_i 's that minimize the squared error

$$E(w_0, \dots, w_m) = \sum (d - (w_0 + w_1 x_1 + w_2 x_2 + \dots + w_m x_m))^2$$

- Gradient:

$$\nabla E[w] = [\partial E / \partial w_0, \dots, \partial E / \partial w_m]; \Delta w = -\eta \nabla E[w]$$

$$\boxed{\partial E / \partial w_i} = \partial / \partial w_i \sum (d - y)^2 = \partial / \partial w_i \sum (d - \sum_i w_i x_i)^2 = \boxed{2 \sum (d - y)(-x_i)}$$

(summation over all examples)

- The weights should be updated by: $w_i = w_i + \eta \sum (d - y)x_i$
- Summation over all cases? Split the summation by examples!
I.e., for each training example, $w_i = w_i + \eta(d - y)x_i$

增量

批量

Incremental versus Batch Learning

- Batch mode : gradient descent

$$w = w - \eta \nabla E_D[w] \text{ over the entire data } D$$

$$E_D[w] = \sum_d (t_d - o_d)^2$$

y 値

增量更新

逐个增量更新

- Incremental mode: gradient descent

$$w = w - \eta \nabla E_d[w] \text{ over individual training}$$

$$\text{examples } d: E_d[w] = (t_d - o_d)^2$$

- Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if η is small enough

Adaline Training Algorithm

1. start with a random set of weights w 隨机选一组 w .
2. select a pattern x (e.g., at random) 随机选一个样本 x .
3. update weights:

$$w := w + \eta x(d - y), \text{ (Adaline rule)} \quad \underline{w + \eta \cdot x \cdot (d - y)}$$

where d - desired output on input x and $y = wx$

4. goto 2 直到收敛.

感知器：
 $w + \eta \cdot d \cdot x$
 $y = 0$

The step size η should be relatively small

Adaline and Linear Regression

感知器仅用于分类

- Perceptron can be used for classification problems only
Adaline 不要求输出 d' 必须为 1 或 -1
- Adaline doesn't require that outputs are -1's or 1's - arbitrary values are allowed
- Therefore Adaline can be used for solving Linear Regression Problems
Adaline 可用于求解线性回归
- There is a direct (fast) algorithm for Linear Regression! $\Rightarrow LS$.
- **But:** Adaline requires no memory: it "learns on-the-fly";
优势:
it's biologically justified (?)

Linear Regression

- Assume

$$y = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

- Find values for w_0, \dots, w_n for which the squared error:

$$\text{error} = (d_1 - y_1)^2 + (d_2 - y_2)^2 + \dots + (d_k - y_k)^2$$

is minimized

d_1, \dots, d_k are desired values,

y_1, \dots, y_k are predictions

Finding Linear Regression Model: Least Squares Method

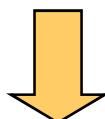
- Error function is a quadratic function of $n+1$ variables:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

$$\text{error} = (d_1 - y_1)^2 + (d_2 - y_2)^2 + \dots + (d_k - y_k)^2$$

$$\text{error}(\mathbf{w}) = (d_1 - y_1(\mathbf{w}))^2 + (d_2 - y_2(\mathbf{w}))^2 + \dots + (d_k - y_k(\mathbf{w}))^2$$

- all partial derivatives=0 ==> error is minimal
- partial derivatives are linear functions of \mathbf{W}



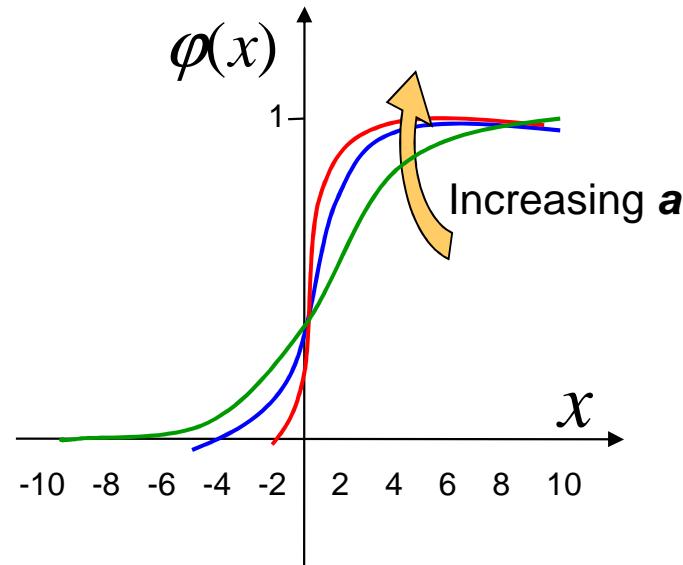
**finding optimal \mathbf{w} reduces to the problem of solving
a system of $(n+1)$ linear equations with $(n+1)$ variables
(no matter how big the training set)**

$$\text{将 } \begin{cases} 1 \\ -1 \end{cases} \text{ 替换为 } \frac{1}{1+e^{-ax}}$$

“Smooth Perceptron” => Logistic Regression

- Main Idea: $\begin{cases} 1 & y \geq 0 \\ 0 & y < 0 \end{cases}$
Replace the sign function by its “smooth approximation” and use the steepest descent algorithm to find weights that minimize the error (as with ADALINE)

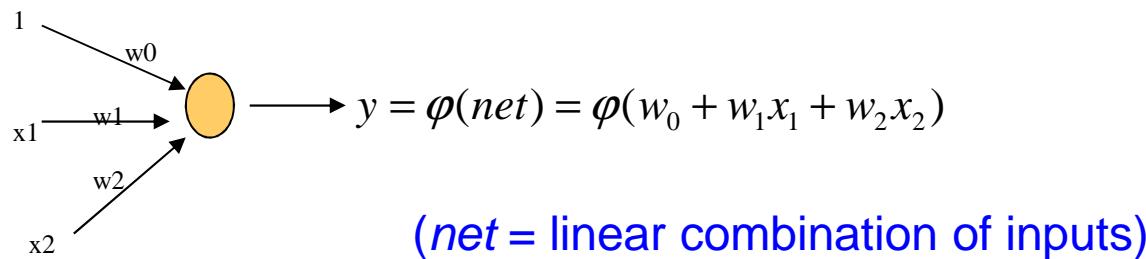
$$\varphi(x) = \frac{1}{1+e^{-ax}} \text{ with } a > 0$$



- The function to be optimized is a bit more complicated than in ADALINE case
- FORTUNATELY: the update rules are simple !

Derivation of update rules for simple net

- Derive the Delta rule for the following network



$$E(w_0, w_1, w_2) = \frac{1}{2} (y - d)^2 = \frac{1}{2} (\varphi(w_0 + w_1 x_1 + w_2 x_2) - d)^2$$

↙ Adaline

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ for } i \text{ in } \{0,1,2\}$$

- We need to find $\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}$

Derivation of Delta Rule

$$\begin{aligned}\frac{\partial E(w_0, w_1, w_2)}{\partial w_1} &= \frac{1}{2} \frac{\partial(\varphi(w_0 + w_1 x_1 + w_2 x_2) - d)^2}{\partial w_1} \\&= \frac{1}{2} 2(\varphi(w_0 + w_1 x_1 + w_2 x_2) - d) \frac{\partial(\varphi(w_0 + w_1 x_1 + w_2 x_2) - d)}{\partial w_1} \\&= (\varphi(\text{net}) - d) \varphi'(\text{net}) \frac{\partial(w_0 + w_1 x_1 + w_2 x_2)}{\partial w_1} \\&= \boxed{(\varphi(\text{net}) - d) \varphi'(\text{net}) x_1}\end{aligned}$$

- From similar calculations we get:

$$\frac{\partial E(w_0, w_1, w_2)}{\partial w_2} = (\varphi(\text{net}) - d) \varphi'(\text{net}) x_2$$

- and

Neural Networks

$$\frac{\partial E(w_0, w_1, w_2)}{\partial w_0} = (\varphi(\text{net}) - d) \varphi'(\text{net}) \cdot 1$$

NN 4

Concluding:

$$\Delta w_0 = \eta(d - \varphi(\text{net}))\varphi'(\text{net})$$

$$\Delta w_1 = \eta(d - \varphi(\text{net}))\varphi'(\text{net})x_1$$

$$\Delta w_2 = \eta(d - \varphi(\text{net}))\varphi'(\text{net})x_2$$

$$\Delta \mathbf{w} = \eta(d - \varphi(\text{net}))\varphi'(\text{net})\mathbf{x}$$

- It's good to know that for the logistic sigmoid function:
 $\varphi(x) = 1/(1+\exp(-x))$ we have: $\varphi'(x) = \varphi(x)(1-\varphi(x)) = \text{output}(1-\text{output})$
- Adaline: Linear Regression
- “A Neuron”: “Logistic Regression” (what is the error function?)

Logistic Regression

- A single neuron is equivalent to logistic regression:
 $y = g(\mathbf{w}\mathbf{x} + w_0)$, where $g(a) = 1/(1 + \exp(-a))$
- In case $P(x|C_1)$ and $P(x|C_2)$ can be modeled by Gaussians with the same matrix Σ , the logistic regression models the posterior probability:
当 $P(x|C_i)$ 为 高斯 分布 时
Logistic 回归 相当于 对后验概率进行 模型

$$P(C_1 | x) = \frac{p(x | C_1)P(C_1)}{p(x | C_1)P(C_1) + p(x | C_2)P(C_2)} = \\ = \frac{1}{1 + \exp(-a)} = g(a), \text{ where } a = \ln \frac{p(x | C_1)P(C_1)}{p(x | C_2)P(C_2)}$$

Summary of learning rules:

Perceptron learning rule:

$$\Delta w = \eta * x * (d - out) \text{ (for misclassified)}$$

Adaline learning rule:

$$\Delta w = \eta * x * (d - out)$$

“A Neuron” learning rule:

$$\Delta w = \eta * x * (d - out) * out'(x)$$

x = input vector
out = output of the network;
out = $f(\text{net})$, where:
net = linear comb. of inputs

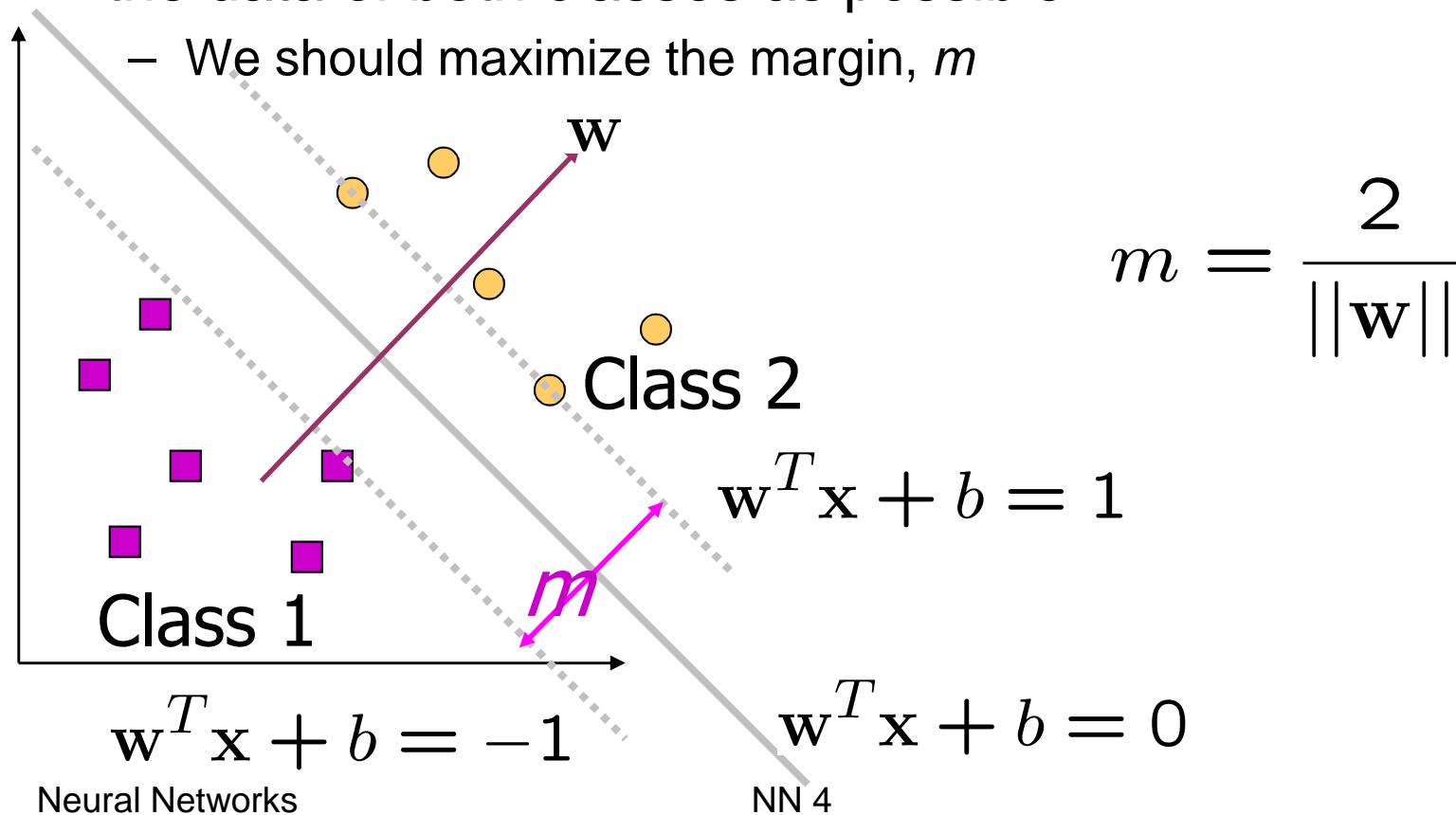
Perceptron: $f(\text{net}) = \text{sign}(\text{net})$

Adaline : $f(\text{net}) = \text{net}$

“Neuron”: $f(\text{net}) = 1/(1+\exp(-\text{net}))$

A yet another approach: Support Vector Machines (SVM)

- The decision boundary should be as far away from the data of both classes as possible



Finding the Decision Boundary

- Let $\{x_1, \dots, x_n\}$ be our data set and let $y_i \in \{1, -1\}$ be the class label of x_i
- The decision boundary should classify all points correctly $\Rightarrow y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall i$
- The decision boundary can be found by solving the following **constrained optimization problem**:

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad \forall i \end{aligned}$$

- A **Quadratic Programming Optimization Problem** (not so easy, but doable...) *doable*
- **Support Vectors**: the training points that are nearest to the separating hyperplane.

SVM for non-separable sets

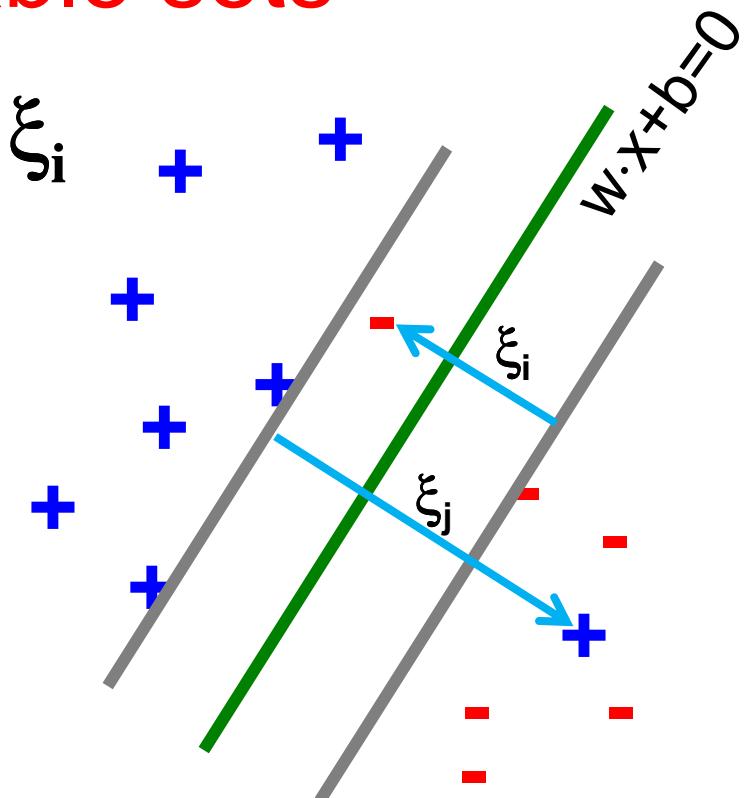
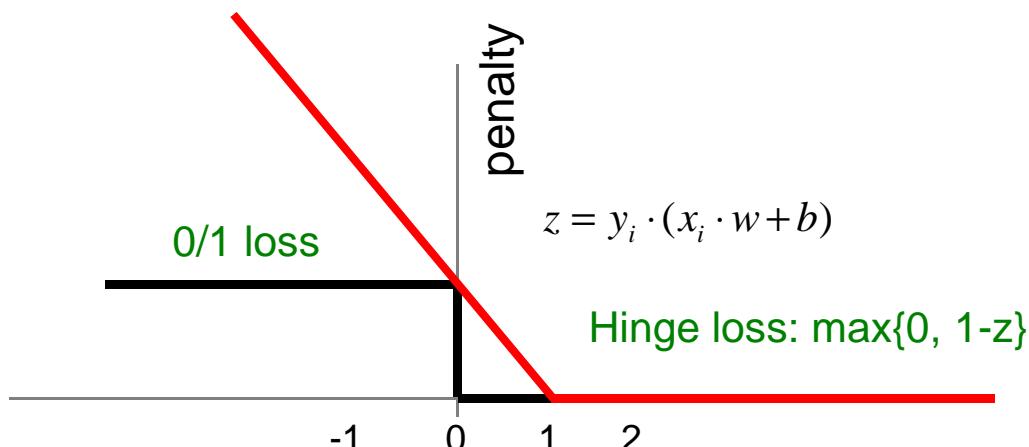
松弛变量

- Introduce **slack variables** ξ_i

$$\min_{w,b,\xi_i \geq 0} \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^n \xi_i$$

$$s.t. \forall i, y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

- If point x_i is on the wrong side of the margin then it gets penalty ξ_i



For each data point:
 If margin ≥ 1 , don't care
 If margin < 1 , pay linear penalty
 (can be expressed by linear cons.)

Support Vector Machines

优势

- State-of-the-art classification procedure
- Performs very well on highly dimensional data (character *高维下分类很好* recognition, text mining, bioinformatics)
- Computationally very expensive (quadratic programming) 计算复杂
an alternative: *Stochastic Gradient Descend (approximated)* 用随机梯度下降.
- *Generalization to non-linear boundaries*
(with help of kernel functions), and more than 2 classes
- Not mentioned in Bishop's book (too recent!)
- *Covered in the MMDS textbook, Ch. 12 (<http://www.mmds.org/>)*

Learning non-linear boundaries with linear models

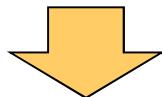
- How could we train a single perceptron to separate the interior of a circle from its exterior?
- It's easy: use as inputs x^2 and y^2 : the discriminant function: $y(x^2, y^2) = x^2 + y^2 - r^2$ linear in its arguments! 将输入的 $x, y \rightarrow x^2, y^2$
即可得出非线性边界
- When using as input x, y, x^2 and x^2 we could even "learn" an ellipse!
- What about using x, y, x^2, x^2 and xy ? What could we model then?
- Generalized Linear Discriminants:

$$y_k(\mathbf{x}) = \sum_{j=1}^M w_{kj} \Phi_j(\mathbf{x}) + w_{k0}$$

- The basis functions $\Phi_j(\mathbf{x})$ are fixed (e.g., polynomials, radial functions)

Generalized Linear Discriminants

- Allowing non-linear basis functions as inputs for linear models dramatically increases the power of these models: in theory such networks can model any boundary (or function), provided we use an appropriate set of basis functions



Linear Models (Single Layer Networks) are very important!

- An important class of basis functions are “radial functions” that measure the distance of x to specific “reference points” x_k . This leads to the concept of **Radial Basis Function Networks (RBF-networks)**
- RBF networks will be discussed later ...

Perceptron for multi-class problems

- So far, we were considering binary classification problems: how to separate two sets of points with a (linear) model? So we were looking for a single (linear) discriminant function...
- Multi-class classification problems: we want to separate $c > 2$ sets of points

Linear Separability for multi-class problems:

There exist **c linear discriminant functions** $y_1(x), \dots, y_c(x)$ such that each x is assigned to class C_k if and only if $y_k(x) > y_j(x)$ for all $j \neq k$

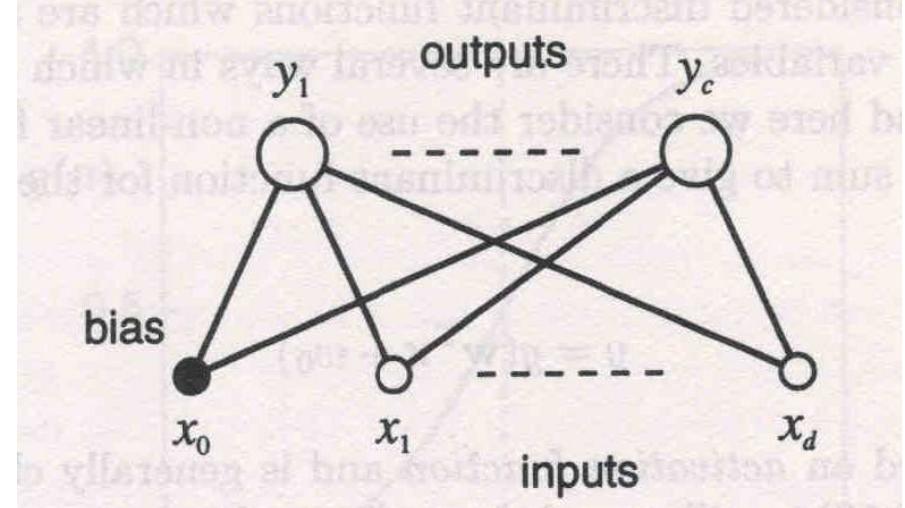
All algorithms discussed so far can be generalized to handle multi-class classification problems. In some cases the generalization is easy, in others not.

Generalized Perceptron convergence theorem:

If the c sets of points are linearly separable then the generalized perceptron algorithm terminates after a finite number of iterations, separating all classes.

线性可分才收敛

Generalized Perceptron Algorithm (Duda et al.)



initialize weights w at random

while (there are misclassified training examples)

Select a misclassified example (x, c_i) 错判样本

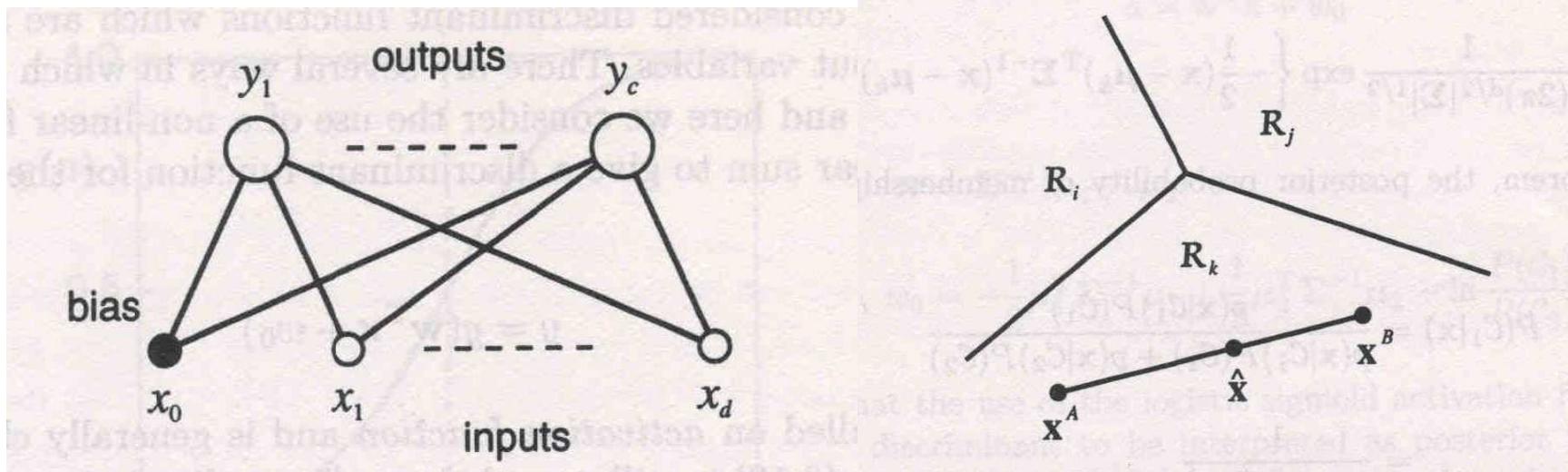
Then *some nodes* are activated more than *the node* c_i 有许 (; 输出节点
输出) (错判)

- 1) update weights of *these nodes* by $-x$: $w = w - x$; 错判之用 $-x$.
- 2) update weights of the node c_i by x : $w = w + x$; c_i 用 $+x$
- 3) leave weights of all other nodes unchanged

end-while:

Perceptron for multi-class problems

A network of c perceptrons that share the same input vector represent c linear discriminant functions and can be used for solving multi-class classification problems.



Note that $y_i - y_j$ is a linear function which separates class i from j , for all i, j .
So decision regions are intersections of half-spaces!

Decision regions are always **convex**: for any two points x^A and x^B from the same region, the whole line interval between x^A and x^B is also in this region.

To Remember

- Discriminant functions, linear discriminants, linear separability, Cover's theorem (the plot and the interpretation!)
- The perceptron learning algorithm and key properties: convergence and bad behaviour on non-separable data sets
- The pocket algorithm (also with ratchet) of Gallant
- Adaline and (logistic) Perceptron; Incremental vs Batch Learning
- Derivation of learning rules for Adaline and Perceptron
- The concept of SVM; quadratic optimization criterion
- Generalized Linear Discriminant
- Multi-class linear separability
- The generalized perceptron algorithm