Machine learning

Linear Discriminant Functions

Mohammad-Reza A. Dehaqani

dehaqani@ut.ac.ir

Generative vs Discriminant Approach

- A classifier that uses discriminant functions **assigns** a feature vector **x** to class ω_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall j \neq i$$

- where $g_i(x)$, $i = 1, \ldots, c$, are the discriminant functions for c classes
- **Dichotomizer (case of two classes):** More common to use a single discriminant function

$$g(\mathbf{x}) \equiv g_1(\mathbf{x}) - g_2(\mathbf{x}),$$

Decide ω_1 if $g(\mathbf{x}) > 0$; otherwise decide ω_2

Example:

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

- **Generative** approaches estimate the **discriminant function** by first estimating the **probability distribution** of the patterns belonging to each class.
- **Discriminant** approaches estimate the **discriminant function** **explicitly**, without assuming a probability distribution.

Linear Discriminants

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

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

- where **w** is the weight vector and w_0 is the **bias** (or **threshold** weight).
- For the two-category case, the **decision rule** can be written as

$$\text{Decide} \quad \begin{cases} w_1 & \text{if } g(\mathbf{x}) > 0 \\ w_2 & \text{otherwise} \end{cases}$$

- The equation $g(\mathbf{x}) = 0$ defines the **decision boundary** that separates points assigned to ω_1 from points assigned to ω_2.
- When $g(\mathbf{x})$ is **linear**, the **decision surface** is a **hyperplane** whose **orientation** is determined by the normal **vector w** (normal to the hyperplane) and **location** is determined by the **bias** w_0.

Minimizing an error function

- The solution can be found by **minimizing** an error function (e.g., "**training error**" or "empirical risk"):
  - The average loss incurred in classifying training the set of training samples
  - Use "**learning**" algorithms to find the solution

$$J(\mathbf{w}, w_0) = \frac{1}{n}\sum_{k=1}^{n}[z_k - \hat{z}_k]^2$$

true   predicted

true class label:

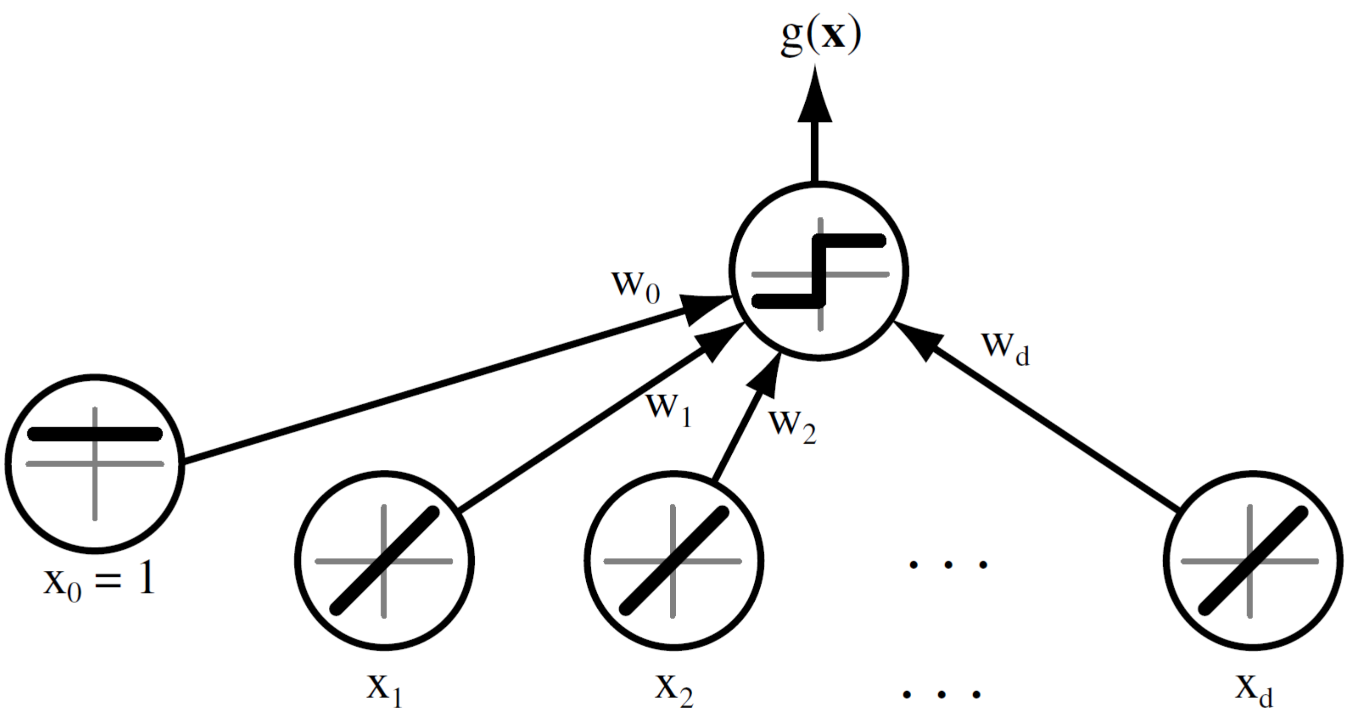
$$z_k = \begin{cases} +1 \; if \;\; \mathbf{x}_k \in \omega_1 \\ -1 \; if \;\; \mathbf{x}_k \in \omega_2 \end{cases}$$

predicted class label:

$$\hat{z}_k = \begin{cases} +1 \; if \;\; g(\mathbf{x}_k) > 0 \\ -1 \; if \;\; g(\mathbf{x}_k) < 0 \end{cases}$$

General structure

- Linear discriminant function as a **general structure** of a pattern recognition system



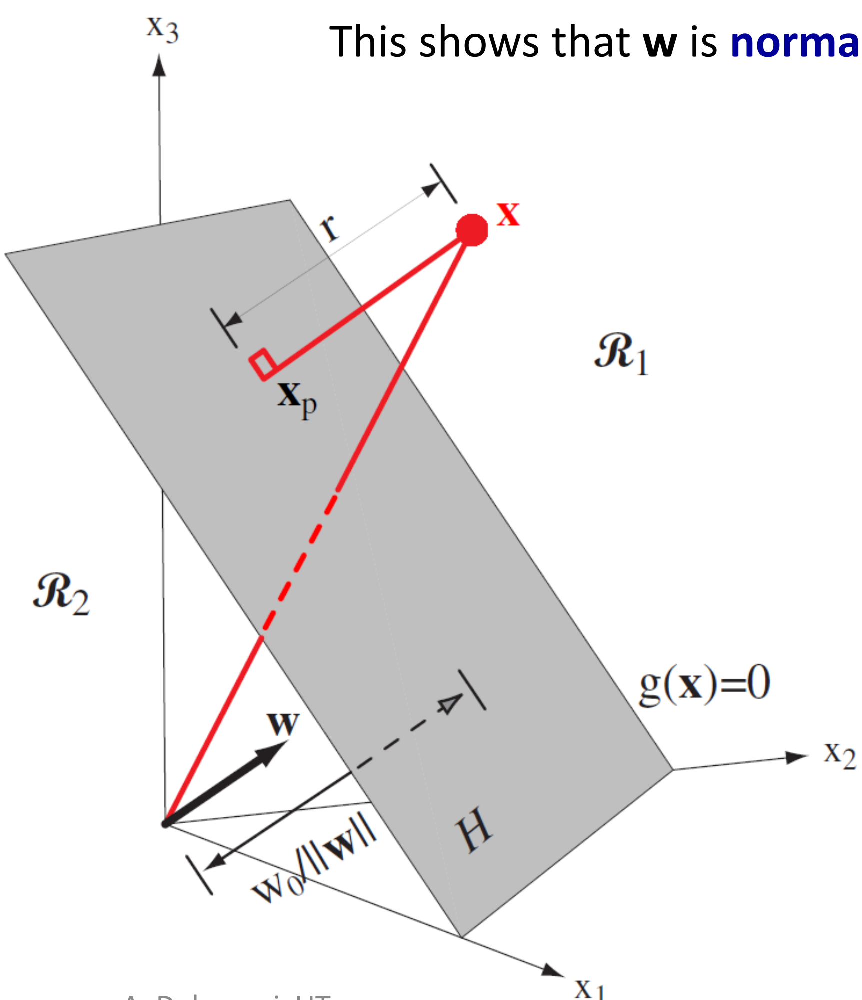
Each input **feature value** x_i is multiplied by its corresponding **weight** w_i; the output unit **sums all these products** and emits a +1 if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a −1 otherwise

Geometric Interpretation of g(**x**)

- If $\mathbf{x}_1$ and $\mathbf{x}_2$ are **both** on the decision surface, then

$$\mathbf{w}^t\mathbf{x}_1 + w_0 = \mathbf{w}^t\mathbf{x}_2 + w_0 \Longrightarrow \mathbf{w}^t(\mathbf{x}_1 - \mathbf{x}_2) = 0,$$

This shows that **w** is **normal** to **any vector** lying in the hyperplane



- g(**x**) provides an **algebraic** measure of the **distance** of **x** from the hyperplane.
- **x** can be expressed as follows:

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|},$$

- where $\mathbf{x}_p$ is the **normal projection** of **x** onto H *(*the hyperplane) , and *r* is the desired algebraic distance

Signed distance from **x** to the hyperplane

- Substitute **x** in g(**x**):

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = \mathbf{w}^t(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + w_0$$
$$= \mathbf{w}^t\mathbf{x}_p + r\frac{\mathbf{w}^t\mathbf{w}}{\|\mathbf{w}\|} + w_0$$
$$= r\|\mathbf{w}\|$$

since g($\mathbf{x}_p$) = 0 ($\mathbf{w}^t\mathbf{x}_p + w_0 = 0$) and $\mathbf{w}^t\mathbf{w} = \|\mathbf{w}\|^2$

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = r\|\mathbf{w}\|, \Rightarrow r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}.$$

- In particular, the **distance from the origin to H** is given by $w_0/||\mathbf{w}||$
- The discriminant function $g(\mathbf{x})$ is **proportional** to the **signed distance from x** to the hyperplane

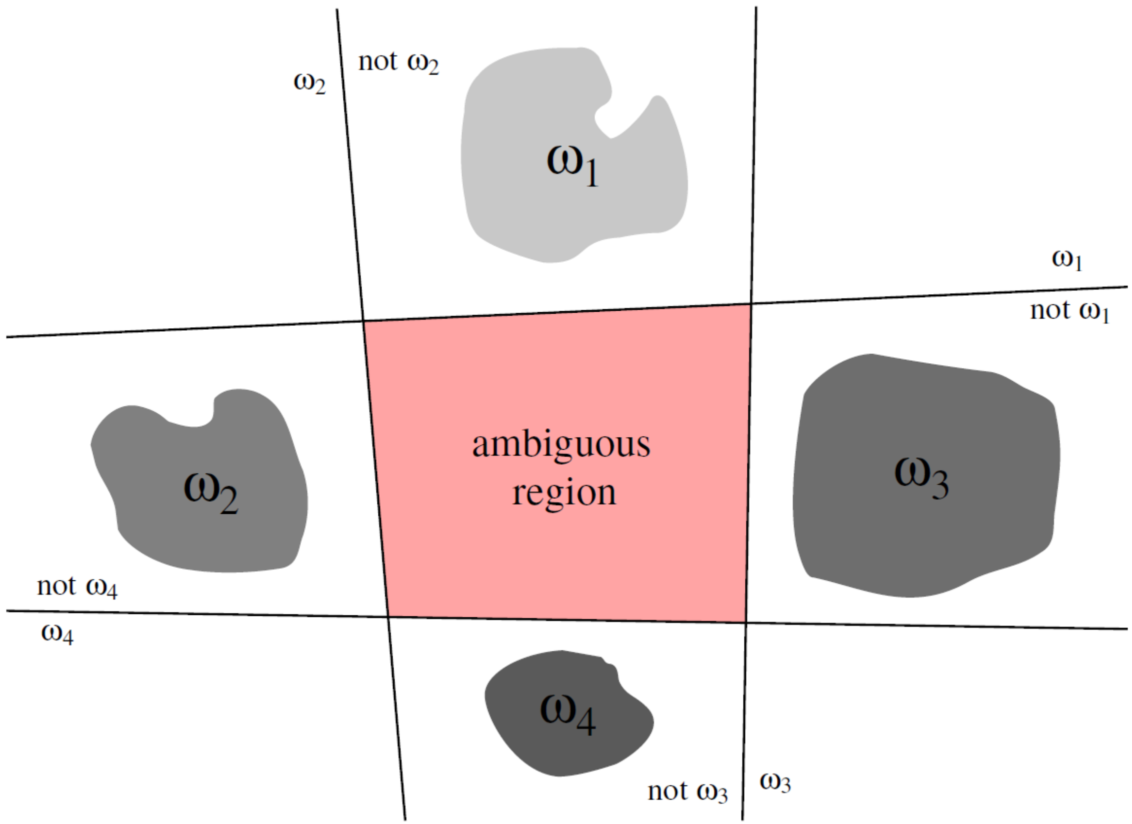
Linear Discriminant Functions: multi-category case

- There is more than one way to devise multicategory classifiers with linear discriminant functions.
- **(1)** One against the rest

We can pose the problem as **c two-class problems**, where the i'th problem is solved by a linear discriminant that separates points **assigned to** ω_i from those **not assigned to** ω_i.

problem:

ambiguous regions

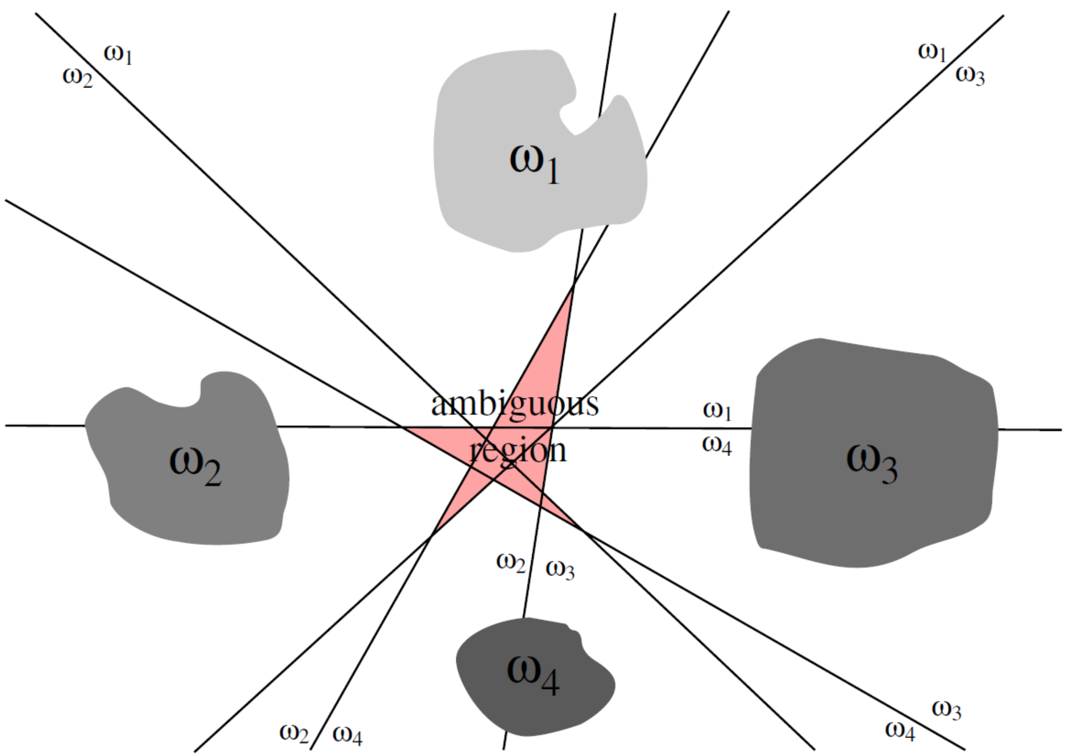


One against another

- We can use c(c−1)/2 linear discriminants, one for **every pair of classes**.

Problem:

Ambiguous regions

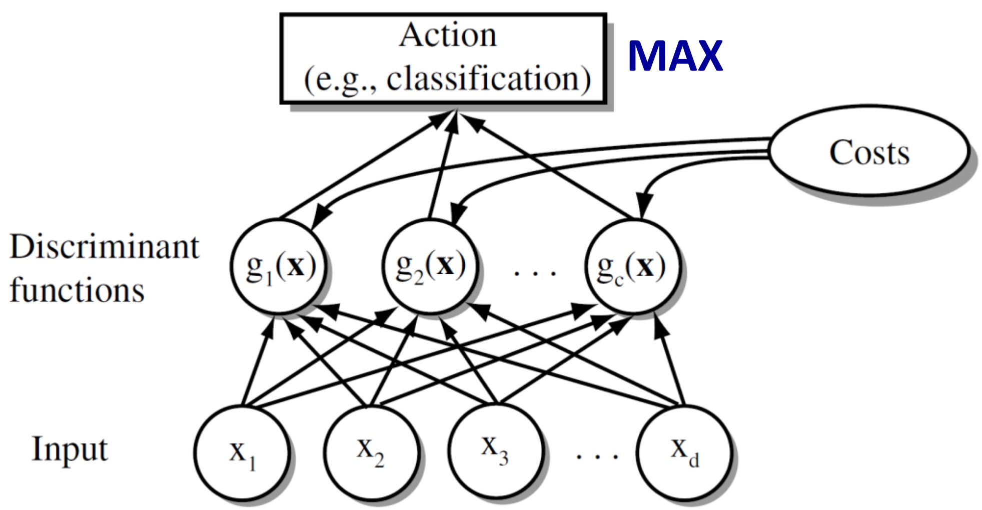


Solving problem of ambiguous regions

- Defining ***c* linear discriminant** functions

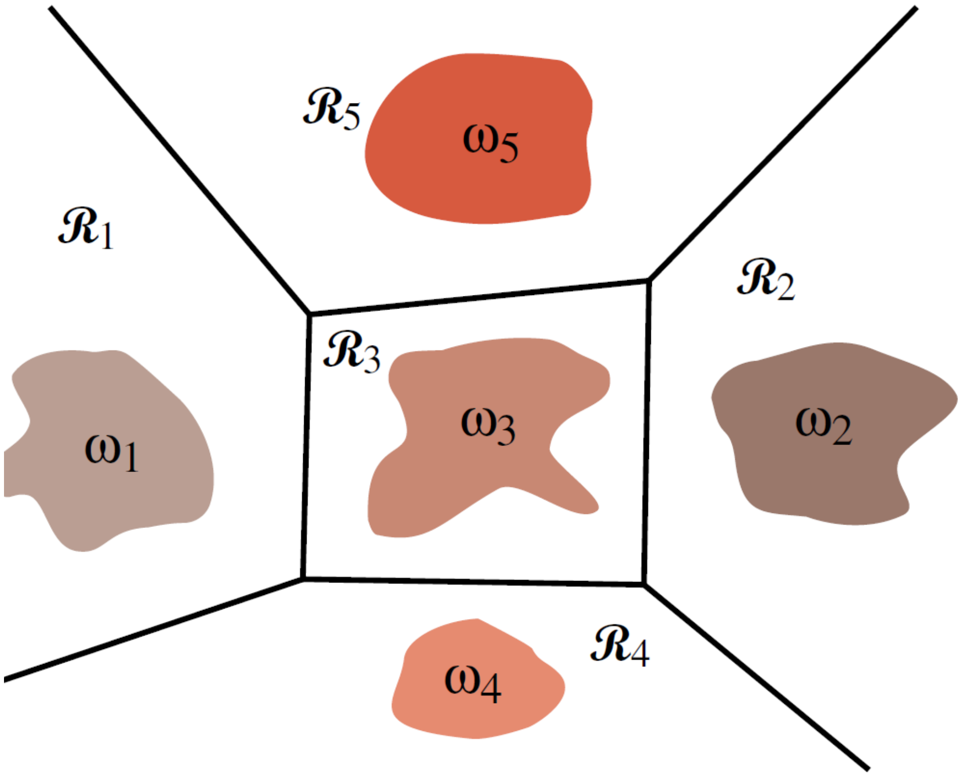
$$g_i(\mathbf{x}) = \mathbf{w}^t \mathbf{x}_i + w_{i0} \qquad i = 1, \ldots, c,$$

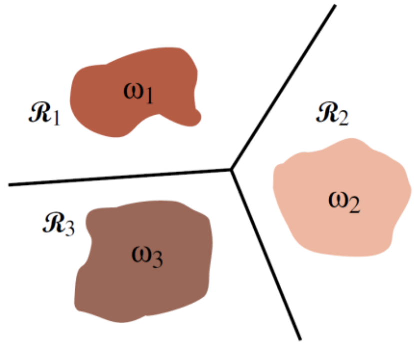
- Assigning **x** to ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$;
- The resulting classifier is called a **linear machine**



Decision boundaries produced bya linear machine

- A **linear machine** divides the feature space in **c convex** decisions regions.
  - If $\mathbf{x}$ is in region R_i, the $g_i(\mathbf{x})$ is the largest.
- Most **suitable** for problems for which the conditional densities $p(\mathbf{x}|\omega_i)$ are **unimodal**.
- Although there are c(c-1)/2 pairs of regions, there typically **less** decision boundaries





Geometric Interpretation

- The decision boundary between adjacent regions R_i and R_j is a **portion** of the hyperplane H_{ij} given by:

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

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

- It follows at once that $\mathbf{w}_i - \mathbf{w}_j$ is **normal** to H_{ij} , and the signed distance from $\mathbf{x}$ to H_{ij} is given by

$$r = \frac{g_i(\mathbf{x}) - g_j(\mathbf{x})}{\| \mathbf{w}_i - \mathbf{w}_j \|}$$

Higher Order Discriminant Functions

- Higher order discriminants yield more **complex decision boundaries** than linear discriminant functions
- By adding additional terms involving the **products of pairs** of components of **x**, we obtain the **quadratic discriminant function**

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j .$$

- The separating surface defined by $g(\mathbf{x}) = 0$ is a second-degree or ***hyperquadric*** surface (additional $d(d+1)/2$ **coefficients**)
- By continuing to add terms such as $w_{ijk}x_i x_j x_k$ we can obtain the class of **polynomial discriminant functions**

Generalized linear discriminant function

- **Mapping** the data to a space of **higher dimensionality**

$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\mathbf{x}) = \mathbf{a}^t \mathbf{y},$$

- This is done by **transforming** the data through **properly** chosen functions $y_i(\mathbf{x})$, i=1,2,…, (called φ functions):

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_d \end{bmatrix} \overset{\varphi}{\Rightarrow} \begin{bmatrix} y_1(\mathbf{x}) \\ y_2(\mathbf{x}) \\ \dots \\ y_{\hat{d}}(\mathbf{x}) \end{bmatrix}$$

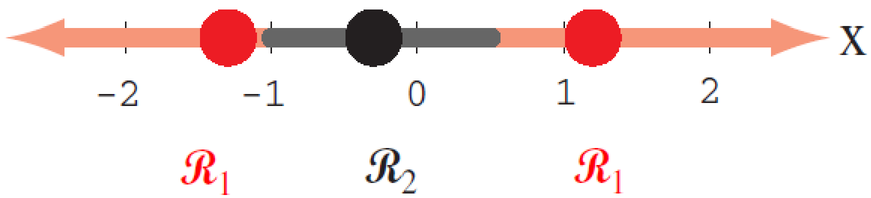
$d \rightarrow \hat{d}$ where $\hat{d} > d$

Linearly-separable by proper transformation

- By **properly** choosing the φ functions, a problem which is **not linearly-separable** in the d-dimensional space, might **become** linearly separable in the $\hat{d}$-dimensional space

- Example:

$$g(x) > 0 \ \ if \ \ x < -1 \ or \ x > 0.5$$



  - The corresponding decision regions R_1, R_2 in the 1D-space are **not** simply connected (not linearly separable).

- Consider the following **mapping** and parameters :

-

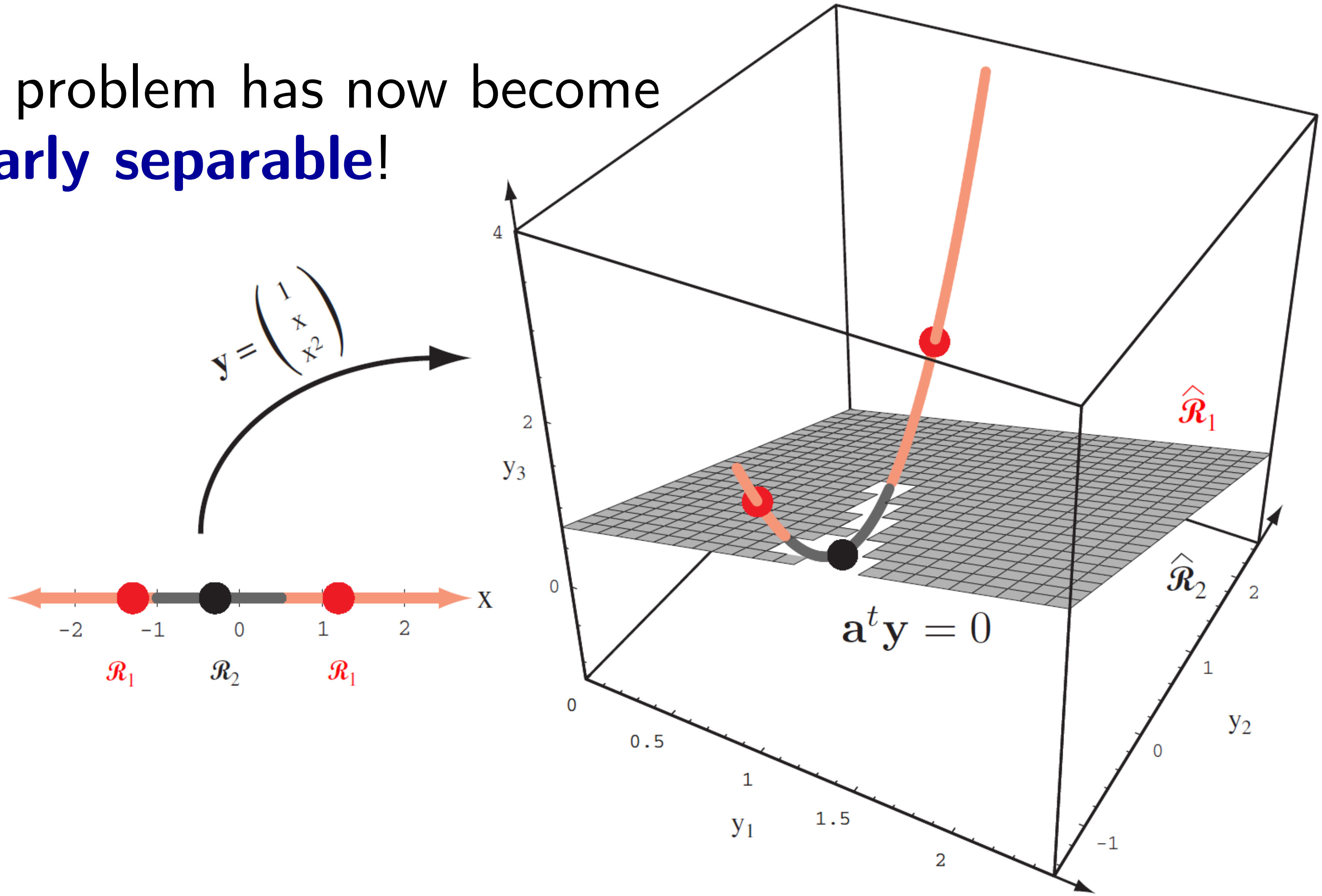
$$\mathbf{y} = \begin{bmatrix} y_1(x) \\ y_2(x) \\ y_3(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

$$g(x) = \overbrace{-1 + x + 2x^2}^{\mathbf{a}^t\mathbf{y}}$$

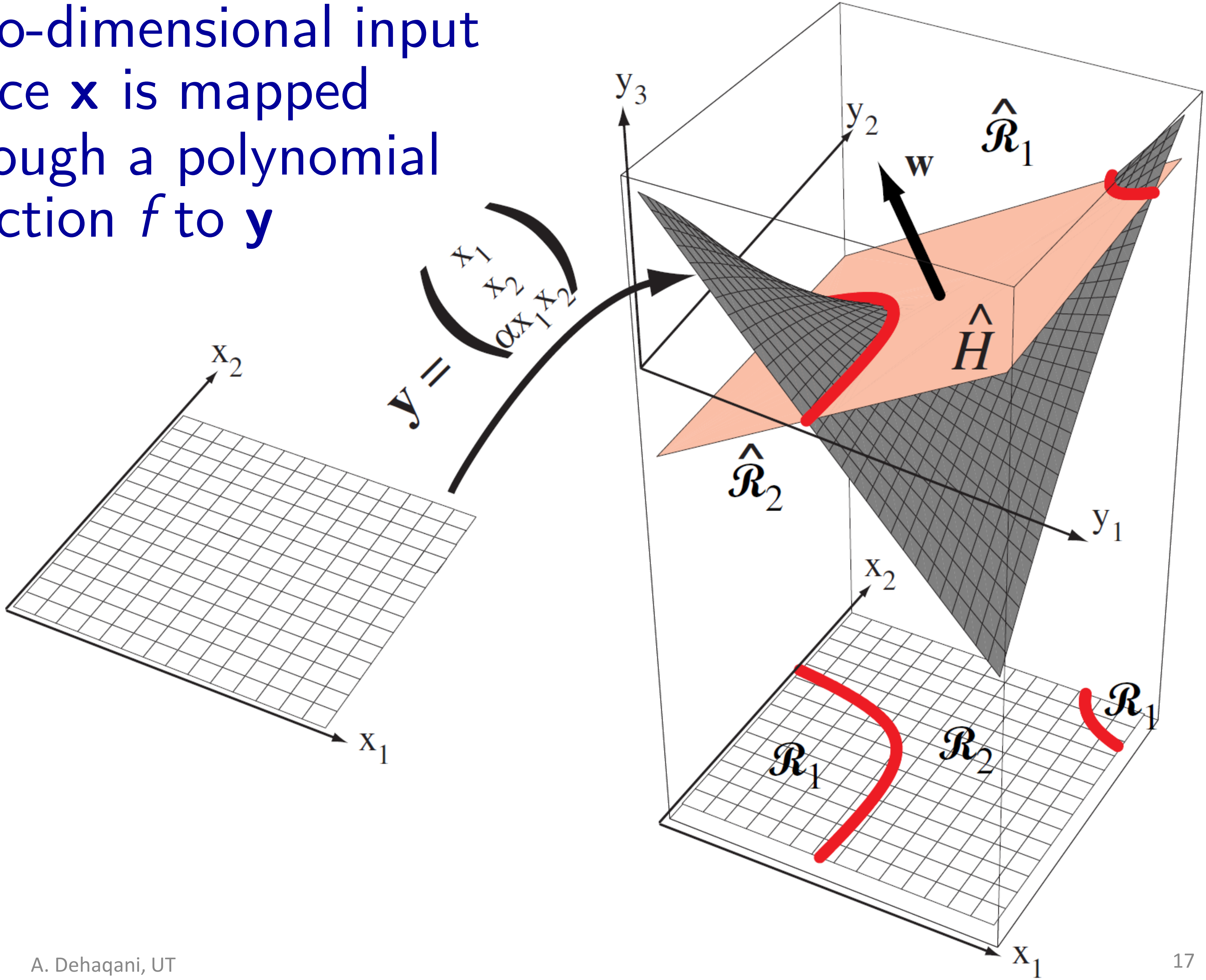
$$\mathbf{a} = (-1, 1, 2)^t$$

The mapping $\mathbf{y} = (1, x, x^2)^t$ takes a line and transforms it to a **parabola** in three dimensions.

The problem has now become **linearly separable**!



Two-dimensional input space **x** is mapped through a polynomial function *f* to **y**



Augmented feature/parameter space

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i$$

- where we set $x_0 = 1$. Thus we can write

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \\ \mathbf{w} \\ \\ \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \\ \mathbf{x} \\ \\ \end{bmatrix}$$

- This mapping from d-dimensional **x**-space to (d+1)-dimensional **y**-space is mathematically trivial but nonetheless quite **convenient**
- The hyperplane decision surface $\widehat{H}$ defined by $\mathbf{a}^t\mathbf{y} = 0$ passes through the **origin** in **y**-space
- The distance from **y** to $\widehat{H}$ is given by $|\mathbf{a}^t\mathbf{y}|/||\mathbf{a}||$, or $|g(\mathbf{x})|/||\mathbf{a}||$. Since $||\mathbf{a}|| > ||\mathbf{w}||$, this distance is **less** than, or at most **equal** to the distance from **x** to H.

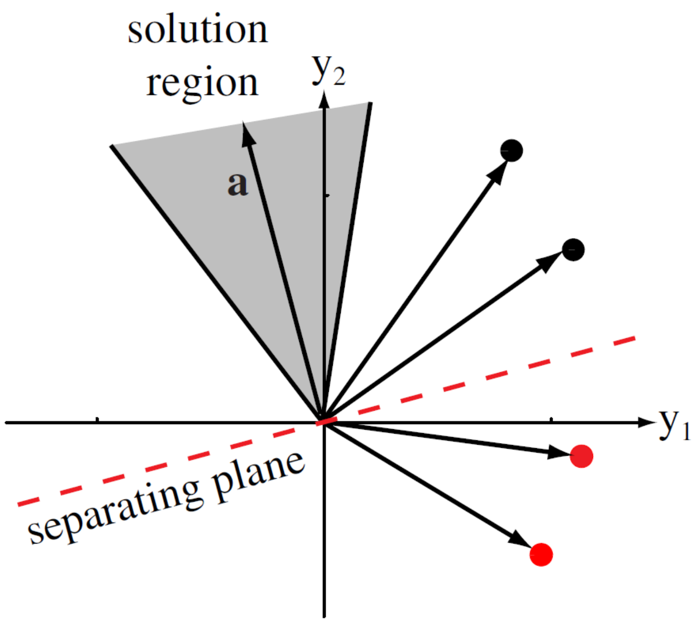
Learning: linearly separable case

- Given a linear discriminant function

$$g(\mathbf{x}) = \mathbf{a}^t\mathbf{y}.$$

the goal is to **"learn"** the parameters (weights) **a** from a set of ***n*** labeled samples $\mathbf{y_i}$, where each $\mathbf{y_i}$ has a class label ω_1 or ω_2.

- Every training sample $\mathbf{y}_i$ places a **constraint** on the weight vector **a**
- Visualize solution in "**feature space**":
  - $\mathbf{a}^t\mathbf{y}=0$ defines a hyperplane in the **feature space** with **a** being the normal vector.
  - Given *n* examples, the solution **a** must lie within a **certain region**.

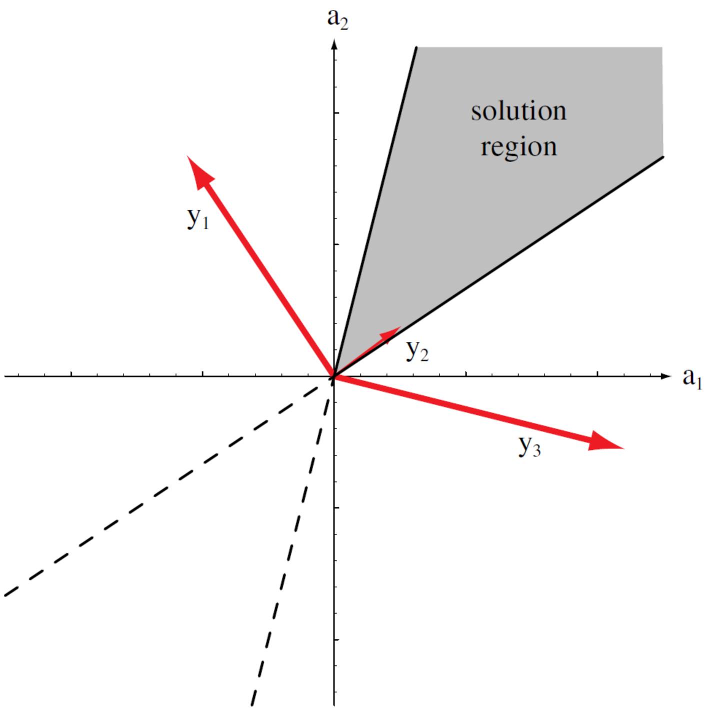


Visualize solution in “parameter space”:

- $\mathbf{a}^t\mathbf{y}=0$ defines a hyperplane in the **parameter space** with **y** being the normal vector.
- Given *n* examples, the solution **a** must lie on the **intersection** of *n* half-spaces

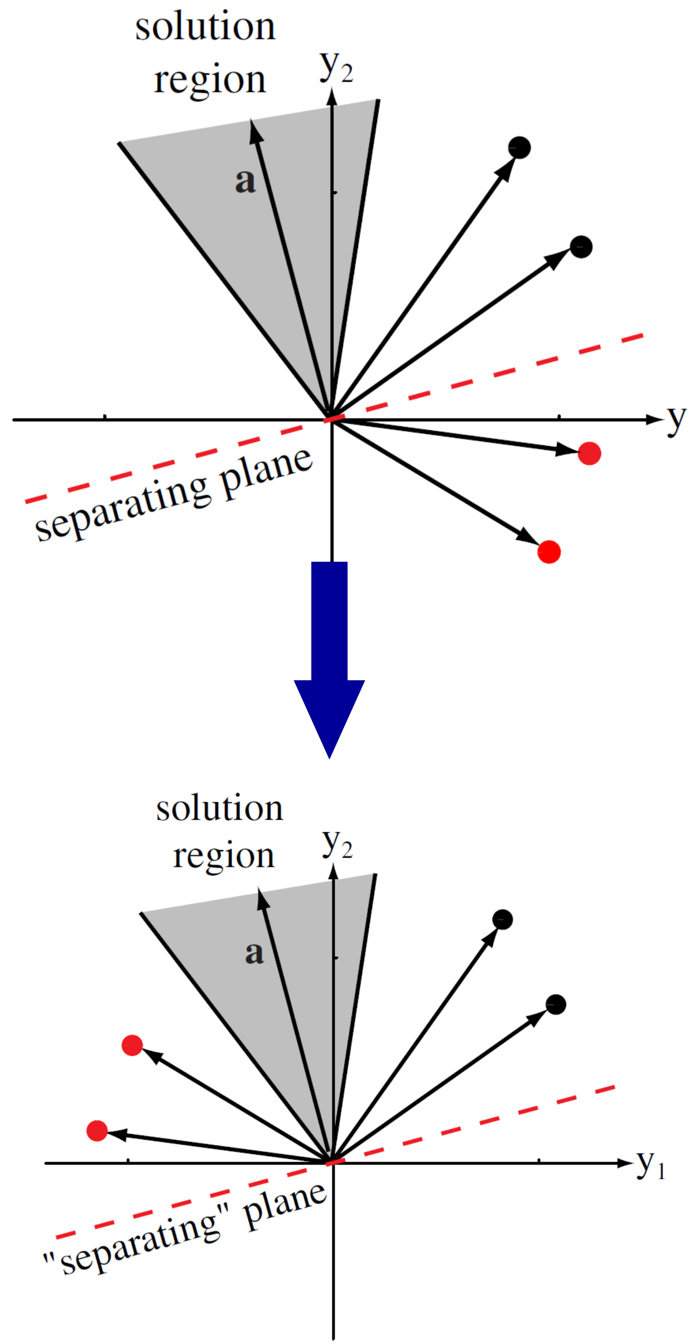
Solution vector **a** is usually **not unique**; we can impose certain constraints to enforce uniqueness, e.g.,:

“Find **unit-length** weight vector **a** that **maximizes** the **minimum distance** from the training examples to the separating plane”



Normalization

- This suggests a "**normalization**" that simplifies the treatment of the two-category case
- The replacement of all samples labelled ω_2 by their negatives. (replace $\mathbf{y}_i$ by -$\mathbf{y}_i$)
- With this "normalization" we can **forget** the **labels** and look for a weight vector **a** such that $\mathbf{a}^t\mathbf{y}_i > 0$ for *all* of the samples.
- Such a weight vector is called a *separating vector* or more generally a ***solution vector***

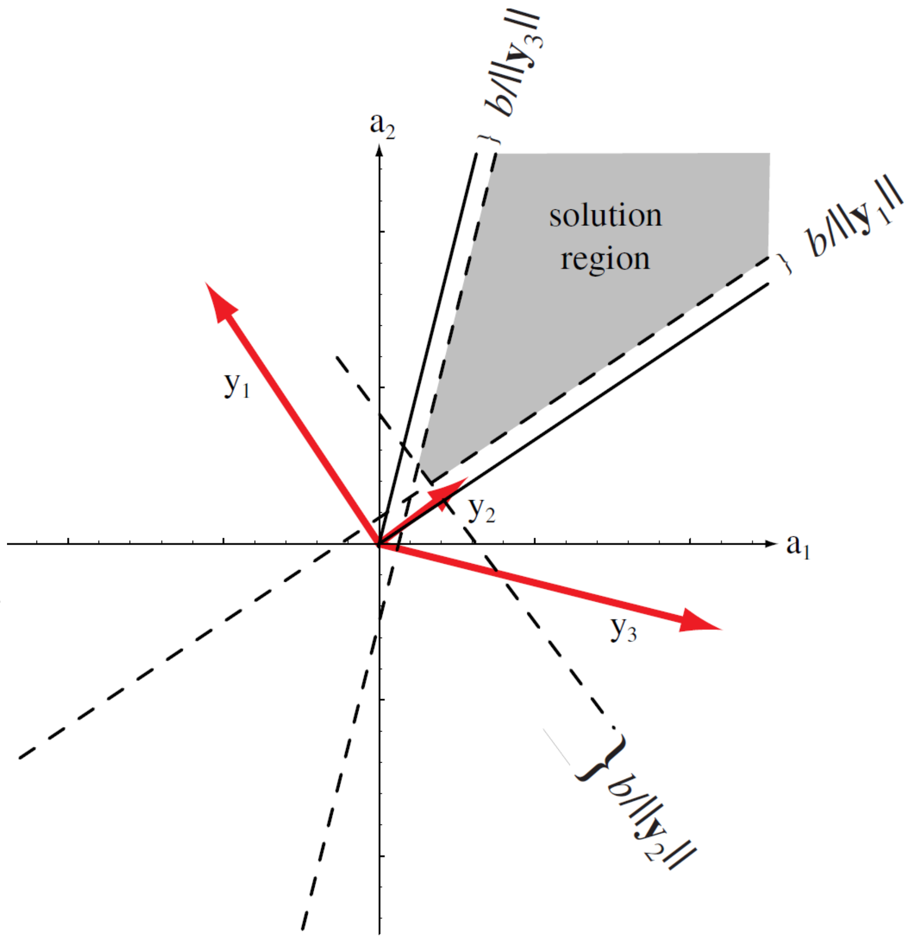


The effect of the **margin** on the solution region

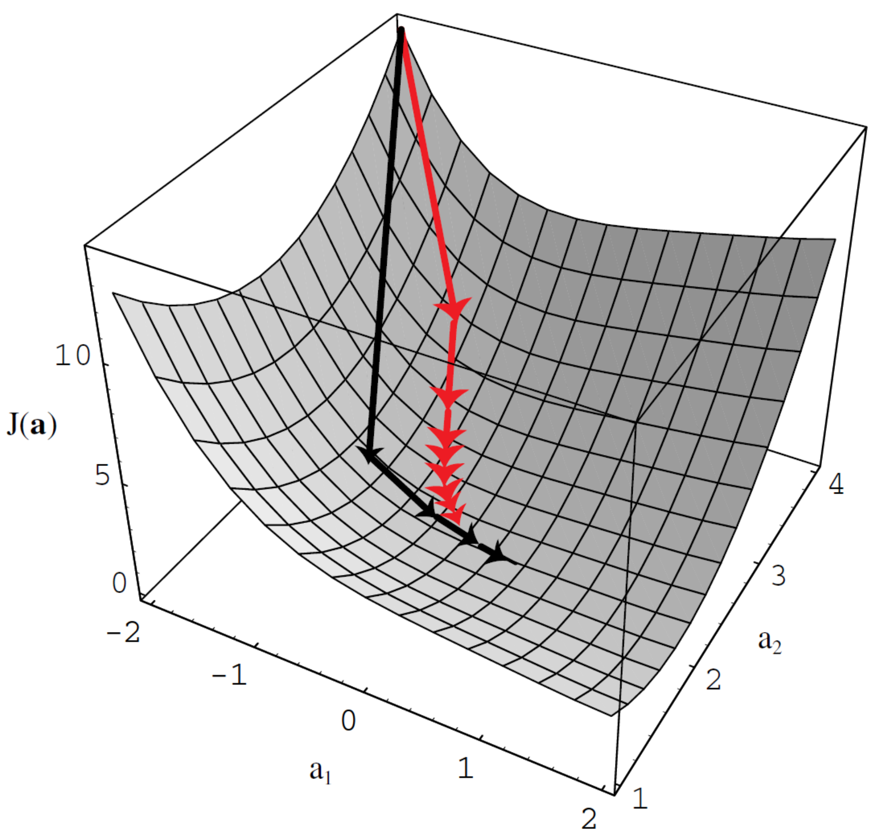
- **seek** the minimum-length weight vector satisfying $\mathbf{a}^t\mathbf{y}_i \geq b$ for all i,
- where b is a positive constant called the **margin**
- The solution region resulting form the intersections of the halfspaces for which

$$\mathbf{a}^t\mathbf{y}_i \geq b > 0$$

- lies within the previous solution region, being **insultated** from the old boundaries by the distance $b/||\mathbf{y}_i||$.
- We find a solution vector closer to the "**middle" of the solution region** to get more likely to classify **new test samples** correctly



"**Learning**" Using Iterative Optimization



$$J(\mathbf{a}) = \frac{1}{n}\sum_{k=1}^{n}[z_k - \hat{z}_k]^2$$

- **Gradient Descent Procedures:**
- finding a solution vector **a** for $\mathbf{a}^t\mathbf{y}_i > 0$
- define a **criterion function** $J(\mathbf{a})$ that is minimized if **a** is a **solution vector**

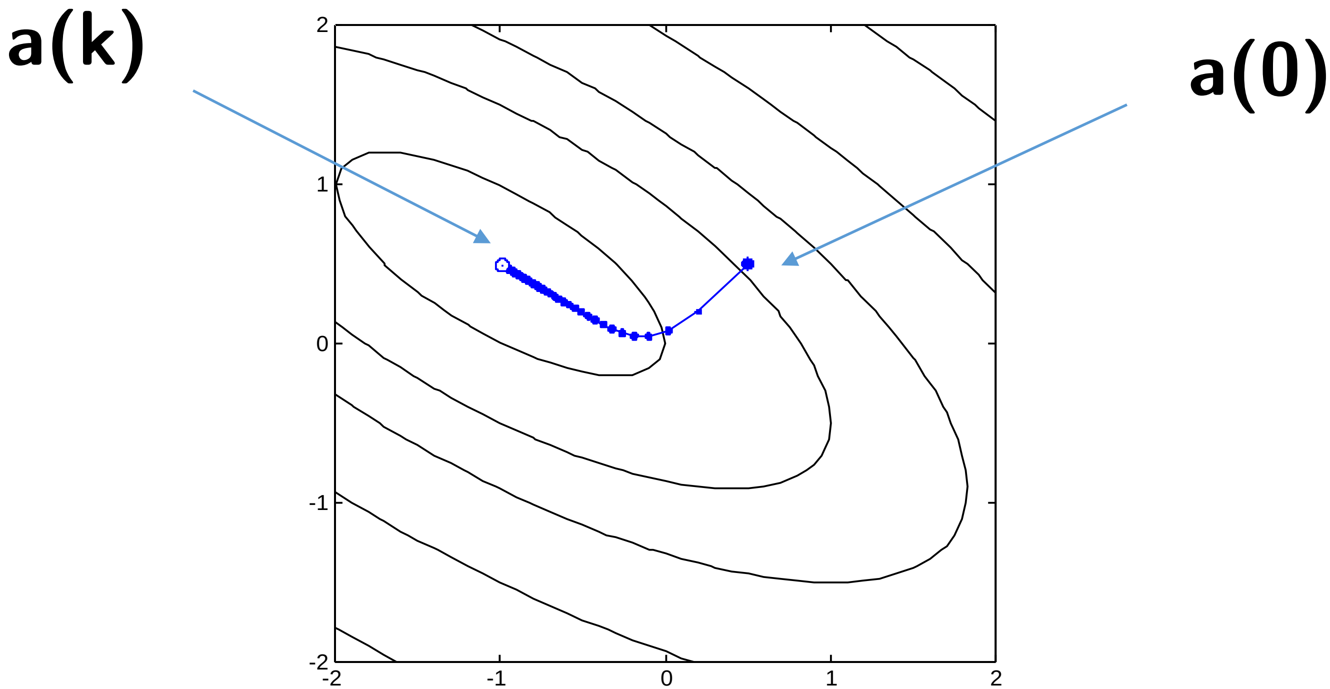
search direction

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k)),$$

- η is a positive scale factor or **learning rate**

Gradient Descent

search space



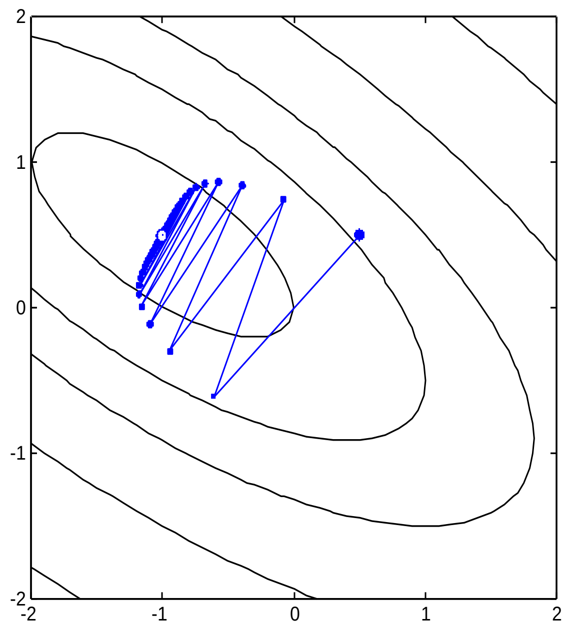
j(a)

begin **initialize** $\mathbf{a}$, criterion $\theta, \eta(\cdot), k = 0$
  do $k \leftarrow k + 1$
    $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\nabla J(\mathbf{a})$
  until $\eta(k)\nabla J(\mathbf{a}) < \theta$
return $\mathbf{a}$
end

Effect of the learning rate

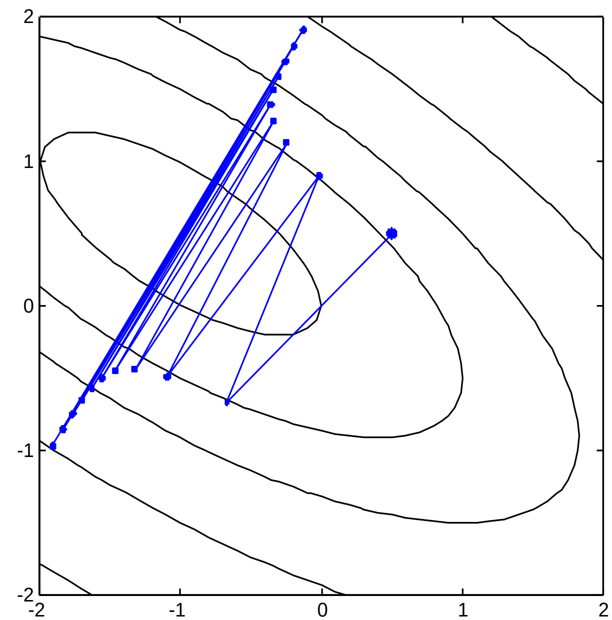
$J(\mathbf{a})$

$\eta = 0.37$



slow but converges to solution

$\eta = 0.39$



fast but overshoots solution

Optimum learning rate

- Suppose that the criterion function can be well **approximated by the second-order** expansion around a value **a**(k) as

Hessian (2nd derivatives)

$$J(\mathbf{a}) \simeq J(\mathbf{a}(k)) + \nabla J^t(\mathbf{a}-\mathbf{a}(k)) + \frac{1}{2}(\mathbf{a}-\mathbf{a}(k))^t\mathbf{H}\,(\mathbf{a}-\mathbf{a}(k)),$$

- Evaluating J(**a**) at a=a(k+1) and using $\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k)),$
- We find:

$$J(\mathbf{a}(k+1)) \simeq J(\mathbf{a}(k)) - \eta(k)\|\nabla J\|^2 + \frac{1}{2}\eta^2(k)\nabla J^t\mathbf{H}\nabla J.$$

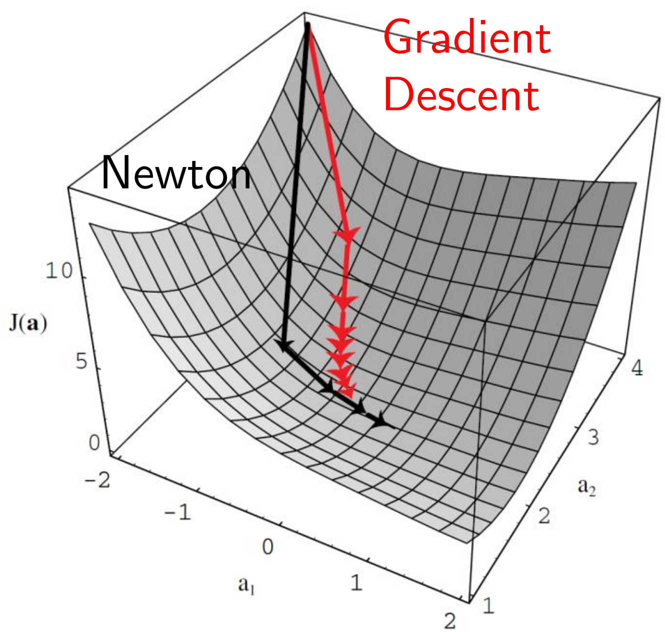
- It is **expensive** in practice

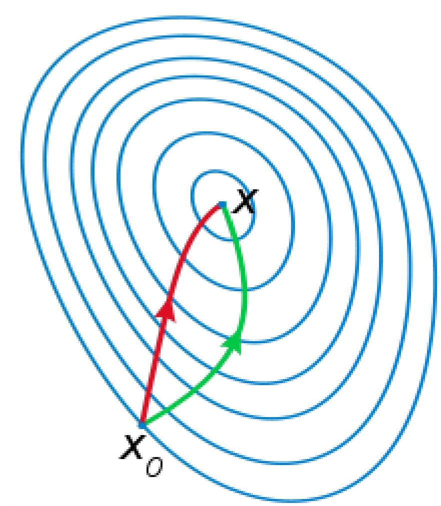
$$\boxed{\eta(k) = \frac{\|\nabla J\|^2}{\nabla J^t\mathbf{H}\nabla J},}$$

Newton's algorithm

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \mathbf{H}^{-1}\boldsymbol{\nabla} J,$$

- It is not applicable if the Hessian matrix **H** is **singular**
- It requires **inverting** H ($O(d^3)$; too expensive)
- If J(**a**) is **quadratic**, Newton's method converges in one ite





gradient descent (green) and Newton's method (red) for minimizing a function

Newton's method uses curvature **information** (i.e. the second derivative) to take a more direct ro

Perceptron rule

- The most obvious choice for **criterion function** is to let $J(\mathbf{a}; \mathbf{y}_1, \ldots, \mathbf{y}_n)$ be the **number of samples misclassified** by **a**.
- This function is **piecewise constant**, it is a poor candidate for a **gradient search**
- Better choice is the ***Perceptron criterion function***

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

- Since $\mathbf{a}^t\mathbf{y} \leq 0$ if $\mathbf{y}$ is misclassified, $J_p(\mathbf{a})$ is never negative.
- $J_p(\mathbf{a})$ is **proportional** to the **sum of the distances** from the misclassified samples to the decision boundary
- **Gradient**

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

- **Update** rule becomes

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

Batch Perceptron

- The batch Perceptron algorithm for finding a solution vector can be stated very simply :
  - The next weight vector is obtained by adding some multiple of the **sum of the misclassified samples** to the present weight vector.
- **Keep changing** the orientation of the hyperplane until all training samples are on its positive side.

Fixed-increment single-sample Perceptron

- $\mathbf{y}^k$ is misclassified.

$$\left.\begin{array}{ll} \mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \mathbf{y}^k & k \geq 1 \end{array}\right\}$$

- Since $\mathbf{a}(k)$ misclassifies $\mathbf{y}^k$, $\mathbf{a}(k)$ is not on the positive side of the $\mathbf{y}^k$ hyperplane $\mathbf{a}^t\mathbf{y}^k = 0$.
  - The addition of $\mathbf{y}^k$ to $\mathbf{a}(k)$ **moves the weight vector directly toward** and perhaps across this hyperplane