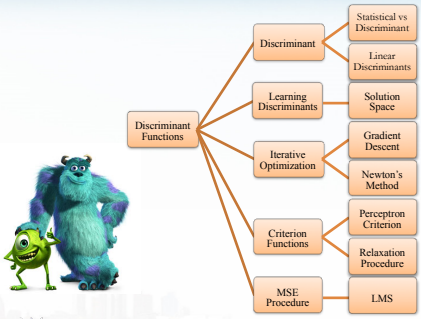
Lecture Notes on

Pattern Recognition

Session 06(A): Discriminant Functions (I)

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Road map



Discriminant Approach

- Discriminant Approach
- Discriminant Function
- Linear Discriminant Function

Statistical vs Discriminant Approach

- ***Statistical*** approaches find the decision boundary by first estimating the probability distribution of the patterns belonging to each class.
- In the ***discriminant*** approach, the decision boundary is constructed explicitly without assuming a probability distribution.

Discriminant Approach

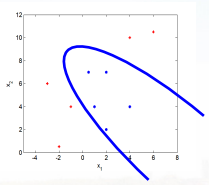
- Specify ***parametric form*** of the decision boundary (e.g., linear or quadratic) .
- Find the ***best*** decision boundary of the specified form using a set of training examples x_k.
  - This is performed by ***minimizing*** a criterion function (e.g., "training error" or "sample risk"):

$$J(w) = \frac{1}{n}\sum_{k=1}^{n}[z_k - g(x_k, w)]^2$$

correct class — predicted class

Discriminant Functions: two-categories case

- Decide w_1 if $g(\mathbf{x}) > 0$ and w_2 if $g(\mathbf{x}) < 0$
  - If $g(\mathbf{x})=0$, then $\mathbf{x}$ lies on the decision boundary and can be assigned to either class.
- Classification is viewed as finding a decision boundary that separates the data belonging to different classes.



Linear Discriminant Functions: two-categories case

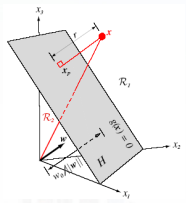
- A linear discriminant function has the following form:

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

- The decision boundary, given by $g(\mathbf{x})=0$, is a ***hyperplane*** where the orientation of the hyperplane is determined by $\mathbf{w}$ and its location by w_0.
  - $\mathbf{w}$ is the normal to the hyperplane
  - If $w_0=0$, the hyperplane passes through the origin

Geometric Interpretation of g(x)

- g(x) provides ***an algebraic measure of the distance*** of x from the hyperplane.



$\mathbf{x}$ can be expressed as follows:

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

- direction of r

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

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

Geometric Interpretation of g(x)

- Therefore, the distance of x from the hyperplane is given by:

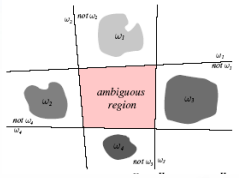
$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$

- w_0 determines the distance of the hyperplane from the origin:
  - setting x=0: $r = \frac{w_0}{\|\mathbf{w}\|}$

Linear Discriminant Functions: multi-category case

- There are several ways to devise multi-category classifiers using linear discriminant functions:
  1. One against the rest (i.e., c-1 two-class problems)

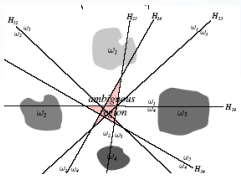
problem
- ambiguous regions



Linear Discriminant Functions: multi-category case

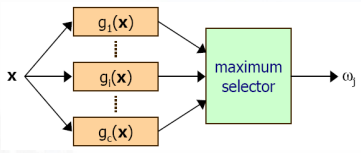
- There are several ways to devise multi-category classifiers using linear discriminant functions:
  2. One against another (i.e., c(c-1)/2 pairs of classes)

problem
- ambiguous regions



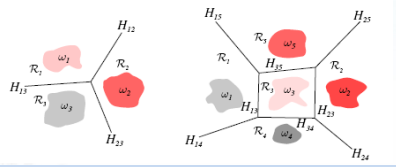
Linear Discriminant Functions: multi-category case

- To avoid the problem of ambiguous regions:
  - Define c linear discriminant functions
  - Assign $\mathbf{x}$ to w_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$.
- The resulting classifier is called a ***linear machine***



Linear Discriminant Functions: multi-category case

- A linear machine divides the feature space in c ***convex*** decisions regions.
  - If x is in region R_i, the $g_i(x)$ is the largest.



- c(c-1)/2 pairs of regions but typically less decision boundaries

Linear Discriminant Functions: multi-category case

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

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

- $(w_i - w_j)$ is normal to H_{ij} and the signed distance from x to H_{ij} is

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

Higher Order Discriminant Functions

- Can produce more complicated decision boundaries than linear discriminant functions.

Linear discriminant: $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 = \sum_{i=1}^{d} w_i x_i + w_0$

Quadratic discriminant: obtained by adding terms corresponding to products of pairs of components of x

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

Polynomial discriminant: obtained by adding terms such as $x_i x_j x_k w_{ijk}$.

Generalized discriminants

- Generalized linear discriminant functions

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

  - a is a $\hat{d}$-dimensional weight vector
  - φ functions $y_i(\mathbf{x})$ map points from the d-dimensional $\mathbf{x}$-space to the $\hat{d}$-dimensional $\mathbf{y}$-space
    - usually $\hat{d} >> d$

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

Generalized discriminants

- Generalized linear discriminant functions

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

  - The resulting discriminant function is ***not*** linear in x but it ***is*** linear in y.
  - The generalized discriminant separates points in the transformed space by a ***hyperplane*** passing through the ***origin***.

Generalized discriminants

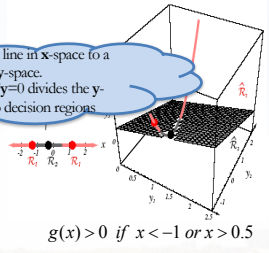
- Example

$$g(x) = -1 + x + 2x^2$$

$$\mathbf{a} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$$

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

- The corresponding decision regions R_1, R_2 in the **x**-space are **not** simply connected!
- $g(\mathbf{x})$ maps a line in **x**-space to a parabola in **y**-space.
- The plane $\mathbf{a}^t\mathbf{y}=0$ divides the **y**-space in two decision regions



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

Learning Discriminants

- Linear Separable
- Solution Space

Learning: two-category, linearly separable case

- Task: Given a linear discriminant function, **learn** the weights **w** using a set of *n* labeled samples $\mathbf{x_i}$
  - i.e., each $\mathbf{x_i}$ has a class label ω_1 or ω_2

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

Augmented feature/weight vectors

- Increasing dimensionality simplifies things:
  - Find a single weight vector **a** in **y**-space.
  - Decision hyperplane, given by g(**y**)=0, in **y**-space passes through the origin (but could be at any position in **x**-space).

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = \sum_{i=1}^{d} w_i x_i + x_0 w_0 = \sum_{i=0}^{d} w_i x_i = \mathbf{a}^t\mathbf{y}$$

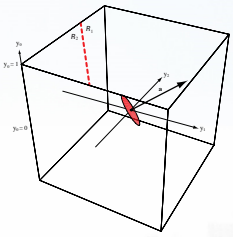
$$(i.e.,\ x_0 = 1)$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix} \Rightarrow \mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_d \end{bmatrix} \Rightarrow \mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \dots \\ x_d \end{bmatrix}$$

- dimensionality: d → (d+1)

Classification in augmented space

- Given a discriminant function g(**x**)=$\mathbf{a}^t\mathbf{y}$, the classification rule is:
  - If $\mathbf{a}^t\mathbf{y}_i$>0 assign $\mathbf{y}_i$ to ω_1
  - else if $\mathbf{a}^t\mathbf{y}_i$<0 assign $\mathbf{y}_i$ to ω

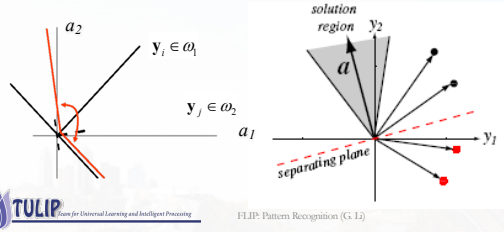


Learning: two-category, linearly separable case

- Given a discriminant function g(**x**)=$\mathbf{a}^t\mathbf{y}$, the goal is to **learn** the weights **a** using a set of *n* labeled samples $\mathbf{y_i}$
  - Every training sample $\mathbf{y}_i$ places a ***constraint*** on the weight vector **a**.
    - $\mathbf{a}^t\mathbf{y}$=0 defines a hyperplane in solution space having **y** as a normal vector.
  - Given *n* examples, the solution **a** must lie on the ***intersection*** of *n* half-spaces

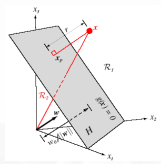
Effect of training examples on solution

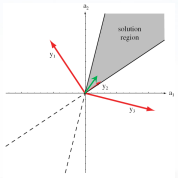
- Solution can be visualized either in the ***solution*** space or the ***feature*** space
  - solution space (a_1, a_2)
  - feature space (y_1, y_2)



Solution Uniqueness/Constraints

- Solution vector is usually not unique; we can impose certain constraints, e.g.:
  - **Method 1**: find unit-length weight vector that maximizes the minimum distance from the samples to the separating plane.



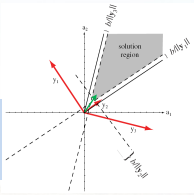


Solution Uniqueness/Constraints

- Solution vector is usually not unique; we can impose certain constraints, e.g.:
  - **Method 2**: find minimum-length weight vector satisfying the constraint below where b is a positive constant called margin.

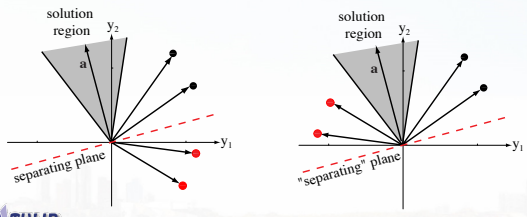
$$g(\mathbf{x}) = \mathbf{a}^t\mathbf{y}_i \geq b > 0$$

Main objective:
- move solution to the center of the feasible region as this solution is more likely to classify new test samples correctly.



Normalized Version

- If $\mathbf{y}_i$ in ω_2, replace $\mathbf{y}_i$ by $-\mathbf{y}_i$
- Find $\mathbf{a}$ such that: $\mathbf{a}^t\mathbf{y}_i > 0$



Iterative Optimization

- Gradient Descent
- Newton's Method

Iterative Optimization

- Define an error function *J(a)* (based on training samples) that is minimized if *a* is a solution vector.
- Minimize *J(a)* iteratively:

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{p}_k$$

$\mathbf{a}(k) \xrightarrow{\eta(k)\mathbf{p}_k} \mathbf{a}(k+1)$

$\mathbf{p}_k$ • search direction

$\eta(k)$ • learning rate

Gradient Descent

- Gradient descent rule:

$$\mathbf{p}_k = -\nabla J(\mathbf{a}(k))$$

Algorithm 1 (Basic gradient descent)

```
begin initialize a, criterion θ, η(·), k = 0
  do k ← k + 1
    a ← a − η(k)∇J(a)
  until η(k)∇J(a) < θ
return a
end
```

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{p}_k$$

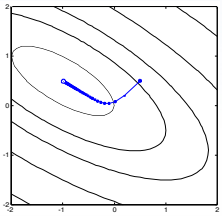
$\mathbf{a}(k) \xrightarrow{\eta(k)\mathbf{p}_k} \mathbf{a}(k+1)$

$\mathbf{p}_k$ • search direction

$\eta(k)$ • learning rate

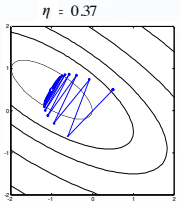
Gradient Descent

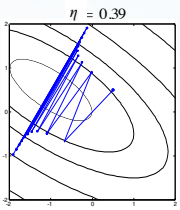
- solution space a



Gradient Descent

- What is the effect of the learning rate?





- slow but converges to solution
- fast by overshoots solution

Gradient Descent

- How to choose the learning rate $\eta(k)$?
  - Make sure $J(a(k+1))$ reaches the lowest point along the decreasing direction

Taylor series approximation

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

Hessian (2nd derivatives)

optimum learning rate

- If $J(\mathbf{a})$ is quadratic, then H is constant which implies that the learning rate is constant!

Newton's method

- Newton's descent rule:

$$\mathbf{p}_k = -H^{-1}\nabla J(\mathbf{a}(k))$$

requires inverting H

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{p}_k$$

$\mathbf{a}(k) \xrightarrow{\eta(k)\mathbf{p}_k} \mathbf{a}(k+1)$

Algorithm 2 (Newton descent)

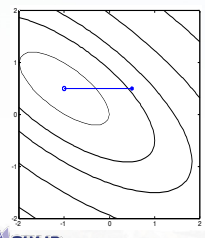
```
1 begin initialize a, criterion θ
2       do
3           a ← a − H^{-1}∇J(a)
4       until H^{-1}∇J(a) < θ
5       return a
6 end
```

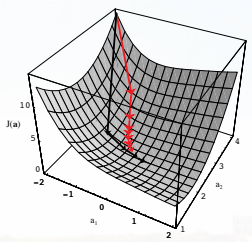
$\mathbf{p}_k$ • search direction

$\eta(k)$ • learning rate

Newton's method

- If $J(\mathbf{a})$ is quadratic, Newton's method converges in **one step**!



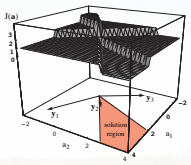


Criterion Function

- Perceptron Criterion Function
- Relaxation Procedure

Criterion Function

- Constructing a ***criterion function*** for solving the linear inequalities $\mathbf{a}^t\mathbf{y}_i>0$.
  - The simplest choice is let J(a) be the number of samples misclassified by a.
    - This leads to a piecewise constant, not suitable for gradient search

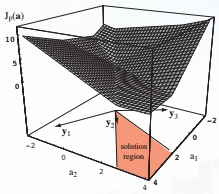


Perceptron Criterion Function

- Constructing a ***criterion function*** for solving the linear inequalities $\mathbf{a}^t\mathbf{y}_i>0$.
  - **Perceptron Criterion Function**:

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

- where $Y(\mathbf{a})$ is the set of samples misclassified by $\mathbf{a}$.
- If $Y(\mathbf{a})$ is empty, $J_p(\mathbf{a})=0$; otherwise, $J_p(\mathbf{a})>0$ (normalized version)



Perceptron rule (Batch Correction)

- The gradient of $J_p(\mathbf{a})$ is:

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

- The perceptron update rule is obtained using gradient descent:

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

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

Algorithm 3 (Batch Perceptron)

```
1 begin initialize a, η(·), criterion θ, k = 0
2     do k ← k + 1
3         a ← a + η(k) Σ_{y∈Yk} y
4     until η(k) Σ_{y∈Yk} y < θ
5     return a
6 end
```

- consider all examples misclassified

Perceptron rule (Single-Sample Correction)

- Fix the learning rate $\eta(k) = 1$
- Modify the weight vector upon each misclassified single sample
- The misclassified sample will be input infinitely often
- The perceptron update rule is obtained using gradient descent:

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)y$$

Perceptron rule (Single-Sample Correction)

- **Perceptron Convergence Theorem**:
  - If training samples are linearly separable, then the sequence of weight vectors will terminate at a solution vector in a finite number of steps.

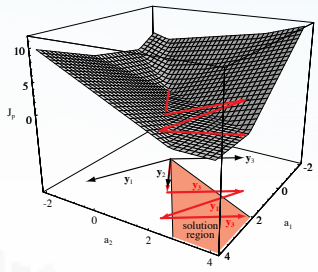
Algorithm 4 (Fixed-increment single-sample Perceptron)

```
1 begin initialize a, k = 0
2     do k ← (k + 1) mod n
3         if y_k is misclassified by a then a ← a + y_k
4     until all patterns properly classified
5     return a
6 end
```

η(k)=1

- consider one example at a time

Perceptron rule



Perceptron rule (Margins)

- Generalizations:
  - Variable learning rate and a margin

Algorithm 5 (Variable increment Perceptron with margin)

```
1 begin initialize a, criterion θ, margin b, η(·), k = 0
2       do k ← k + 1
3          if a^t y_k + b < 0 then a ← a + η(k) y_k
4       until a^t y_k + b ≤ 0 for all k
5    return a
6 end
```


Relaxation Procedures

- Note that different criterion functions exist
- One possible choice is:

$$J_q(a) = \sum_{y \in Y} (a^t y)^2$$

  - Where Y is again the set of the training samples that are misclassified by **a**
  - However, there are two problems with this criterion
    - The function is too smooth and can converge to a=0
    - J_q is dominated by training samples with large magnitude

Relaxation Procedures

- A modified version that avoids two problems is

$$J_r(a) = \frac{1}{2} \sum_{y \in Y} \frac{(a^t y - b)^2}{\|y\|^2}$$

  - Here Y is the set of samples for which its gradient is given by:

$$\nabla J_r = \sum_{y \in Y} \frac{(a^t y - b)}{\|y\|^2} y$$

Relaxation Procedures

Algorithm 8 (Batch relaxation with margin)

```
1  begin initialize a, η(·), k = 0
2        do k ← k + 1
3           Y_k = {}
4           j = 0
5           do j ← j + 1
6              if y_j is misclassified then Append y_j to Y_k
7           until j = n
8           a ← a + η(k) Σ_{y∈Y_k} (b − a^t y)/‖y‖² y
9        until Y_k = {}
10 return a
11 end
```

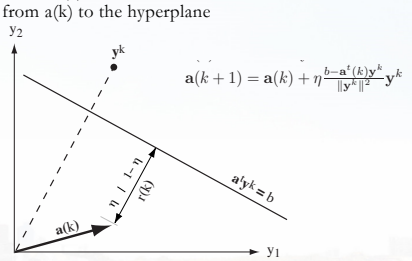

Relaxation Procedures

Algorithm 9 (Single-sample relaxation with margin)

```
1 begin initialize a, η(·), k = 0
2       do k ← k + 1
3          if y_k is misclassified then a ← a + η(k) (b − a^t y_k)/‖y_k‖² y_k
4          until all patterns properly classified
5    return a
6 end
```


Relaxation Procedures

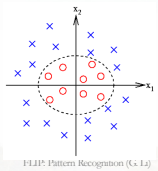
- Since $y/|y|$ is the unit normal vector for the hyperplane. The correction calls for a(k) to be moved a certain fraction of η of the distance from a(k) to the hyperplane



$$a(k+1) = a(k) + \eta \frac{b - a^t(k) y^k}{\|y^k\|^2} y^k$$

Nonseparable Behavior

- Perceptron and relaxation procedures are all error-correction procedure.
  - Their success on separable problems is largely due to its relentless search for an error-free solution.
  - For nonseparable problems, correction may never converge.



MSE Procedure

- MSE and Discrimants
- Pseudoinverse

Minimum Squared Error Procedures

- Define a criterion function for all samples, rather than the misclassified samples
- Try to find the vector such that $\mathbf{a}^t\mathbf{y}_i > \mathbf{b}_i$
  - Where $\mathbf{b}_i$ is any positive value

$$\begin{pmatrix} Y_{10} & Y_{11} & \cdots & Y_{1d} \\ Y_{20} & Y_{21} & \cdots & Y_{2d} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ Y_{n0} & Y_{n1} & \cdots & Y_{nd} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{pmatrix}$$

Minimum Squared Error Procedures

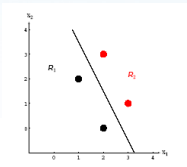
- Minimum squared error and pseudoinverse
  - The problem is to find a weight vector $\mathbf{a}$ satisfying $\mathbf{Ya}=\mathbf{b}$
  - If we have more equations than unknowns, $\mathbf{a}$ is over-determined.
  - We want to choose the one that minimizes the sum-of-squared-error criterion function

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t\mathbf{y}_i - b_i)^2.$$

Minimum Squared Error Procedures

- Pseudoinverse

$$\begin{aligned} \mathbf{a} &= (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{b} \\ &= \mathbf{Y}^\dagger\mathbf{b}, \end{aligned}$$

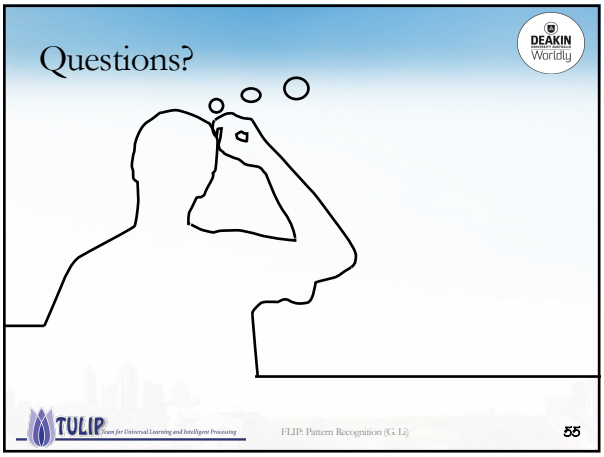


Algorithm 10 (LMS)

```
begin initialize a, b, criterion θ, η(·), k = 0
    do k ← k + 1
        a ← a + η(k)(b_k − a^t y^k) y^k
    until η(k)(b_k − a^t y^k) y^k < θ
    return a
end
```


Seminar S09

- **Topics**
  - Feature Selection and Feature Extraction
  - From Perceptron to Neural Network
  - Widrow-Hoff Algorithm and LMS
- **Requirements**
  - Prepare a **15 minutes** talk on your chosen topic
  - Make **ppt** to assist your talk
  - Prepare **at least 3 questions** to ask the audience after your talk
  - Get ready to **take questions** from the audience
- **Hints**
  - You can search for research articles from Google Scholar


Questions?
DEAKIN
Worldly
TULIP
Team for Universal Learning and Intelligent Processing
FLIP: Pattern Recognition (G. Li)
55