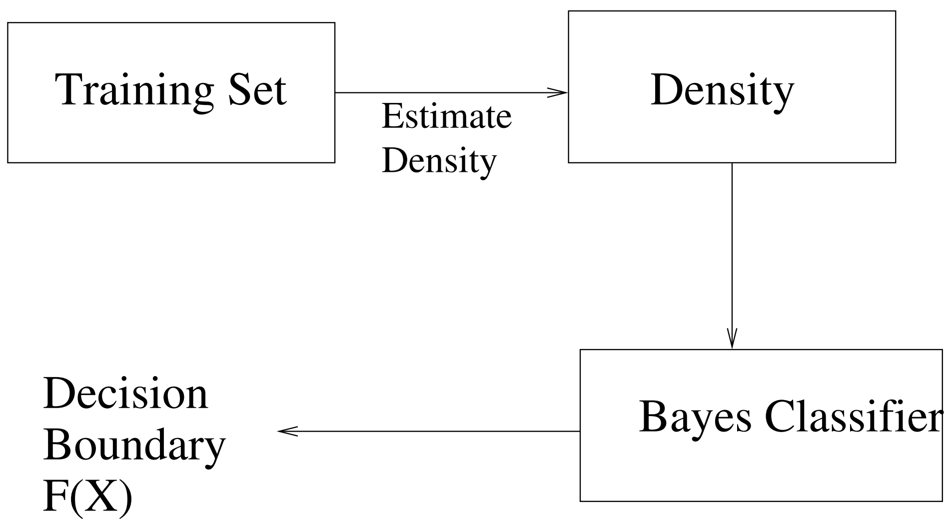
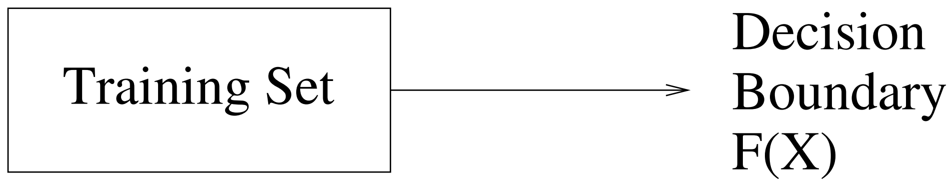
# Linear Discriminant Functions

# Discriminant functions




- This seems an indirect way!
- Density estimation is a much more general problem than finding a classifier.
- Vapnik says, “find the classifier directly, instead of solving a big intermediate problem”.

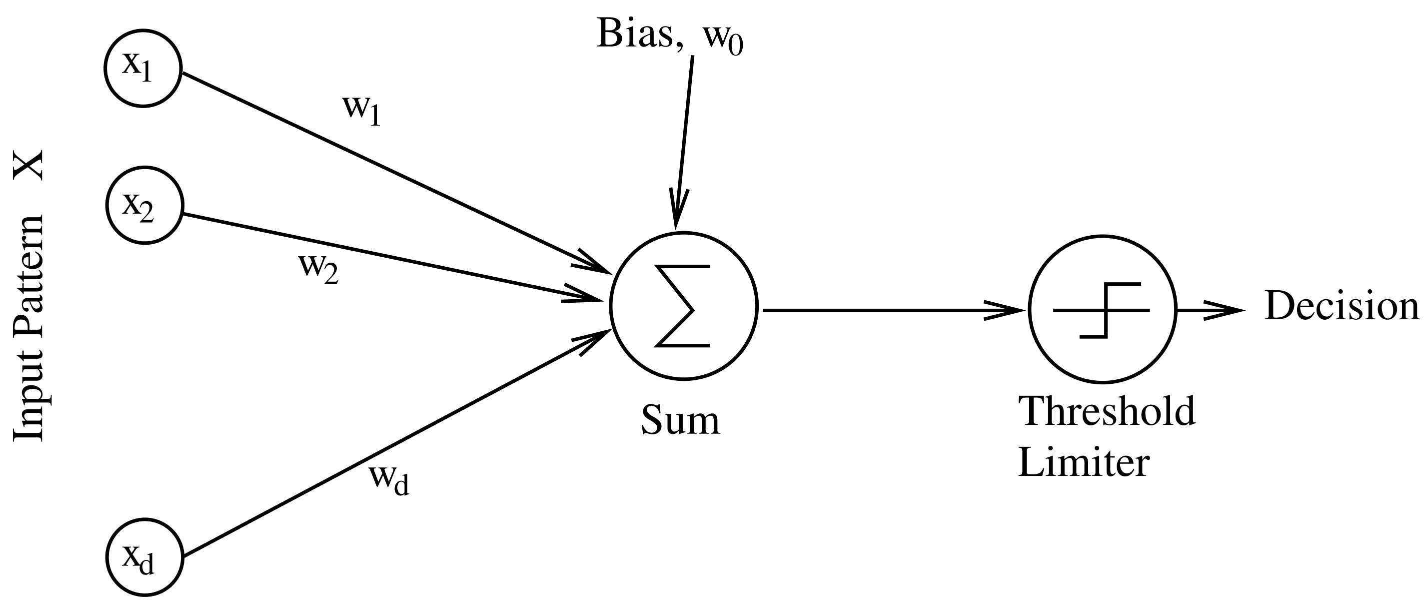
# Discriminant Functions




- Form of the discriminant function is assumed and its parameters are found using the training set.
- If the discriminant function is a linear combination of the feature values, then it is called a *linear discriminant function*.
- Linear discriminant functions are also called *Perceptrons* or *Single layer Perceptrons*.

# Perceptron: some historical remarks ...

- Its inventor is Frank Rosenblatt (1958), a psychologist.
- It is one of the early artificial neural network models.



# Perceptron: some historical remarks ...

- Rosenblatt gave an algorithm which can be implemented with a machine.
- He gave a convergence theorem to establish certain important properties of his method.
- It raised lot of interest among many researchers.

# Perceptron: some historical remarks ...

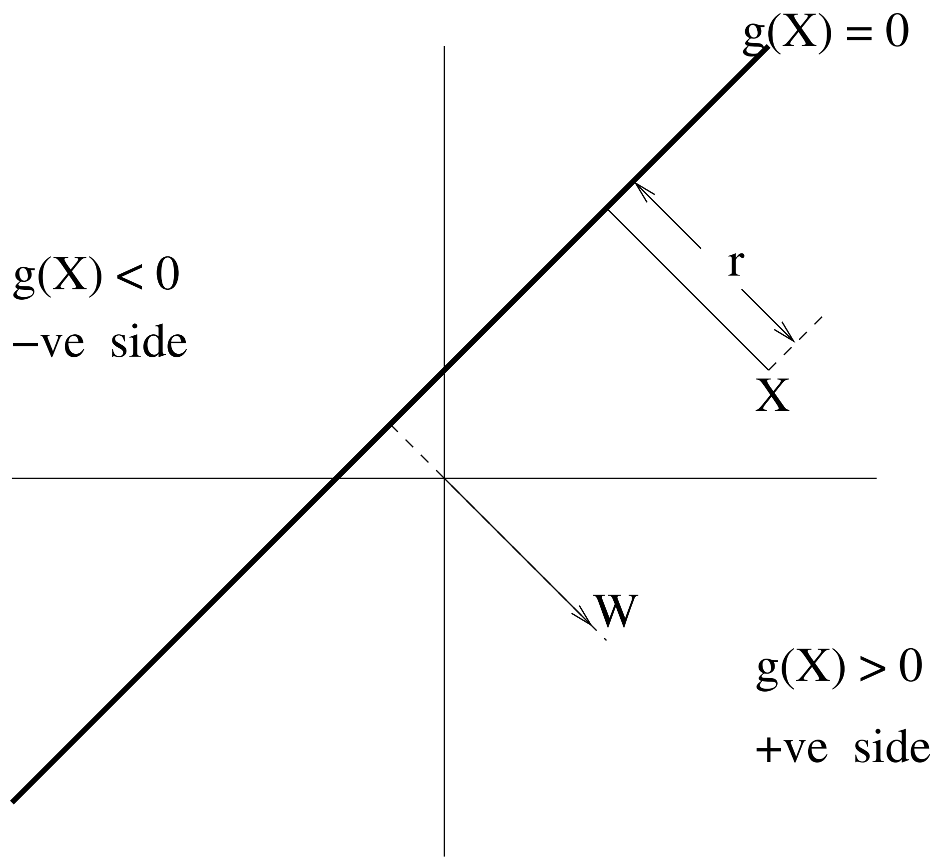
- Minsky and Papert (1969) proved various theorems about single layer perceptrons, some of which indicated their limited pattern-classification and function approximation capabilities. For example, they proved that it can not implement Exclusive OR logical function.
- Many people blamed Misky and Papert that, their work actually dampened research in neural networks.
- Multi-layer perceptron can overcome many shortcomings of the perceptron.
- An algorithm called *error backpropagation* is found to train the multi-layer perceptron around 1986 which can be seen as a second birth of artificial neural networks.

# Perceptron: a linear discriminant function

- Consider a two class problem, $\Omega = \{\omega_1, \omega_2\}$
  A pattern $X = (x_1, \ldots, x_d)^t$
- The discriminant function
  $g(X) = w_1 x_1 + \cdots + w_d x_d + w_0 \;=\; W^t X + w_0$
- $g(X) = 0$ defines a hyperplane in the feature space.
- Classification rule:

$$
\begin{aligned}
g(X) > 0 \quad &\Rightarrow \quad \text{class is } \omega_1 \\
g(X) < 0 \quad &\Rightarrow \quad \text{class is } \omega_2 \\
g(X) = 0 \quad &\Rightarrow \quad \text{class is decided arbitrarily}
\end{aligned}
$$

# Linear Discriminant Function




- $r = g(X)/||W||$ is the perpendicular distance of a point X with the hyperplane.

# General Discriminant Functions

- Linear : $g(X) = w_0 + \sum w_i x_i$
- Quadratic: $g(X) = w_0 + \sum w_i x_i + \sum \sum w_{ij} x_i x_j$
- Generalized: $g(X) = \sum_{i=1}^{\hat{d}} a_i y_i(X)$ where $y_i(X)$ is an arbitrary function of $X$.
  - If we create $\hat{d}$ features in such a way that the $i^{th}$ feature is $y_i(X)$ then in the new $\hat{d}$ dimensional space, $g(X)$ is indeed linear.
  - We call the new space as the $Y$-Space.
  - Multi-layer Perceptrons and Support Vector Machines (SVMs) tries to find a linear discriminat in an appropriate $Y$-Space which is a non-linear function in the $X$-Space.

# Augmented Feature Space

- $X$-Space: $X = (x_1, \ldots, x_d)^t$, $W = (w_1, \ldots, w_d)^t$ and $g(X) = w_0 + W \cdot X$
  - $g(X) = 0$ may not pass through the origin.
- $Y$-Space: $Y = (1, x_1, \ldots, x_d)^t$, $a = (w_0, w_1, \ldots, w_d)^t$ and $\hat{g}(Y) = a \cdot Y$
  - $\hat{g}(Y) = 0$ passes through the origin.
  - That is, $\hat{g}(Y) = 0$ is in homogeneous form.
  - This $Y$-Space is called as *augmented space* and $Y$ is called as *augmented vector* of $X$.
- It is easy to work with the augmented space.
- We want to find the vector $a$ which is called *separating vector* or more generally *solution vector*. It need not be unique.

# Two category case

- Augmented feature space is used.

$$g(Y) = a^t Y = \begin{cases} > 0 & \text{for } Y \in \omega_1 \\ < 0 & \text{for } Y \in \omega_2 \\ = 0 & \text{we do not consider this.} \end{cases}$$

- Normalization
  - For all $Y_i \in \omega_2$ replace $Y_i$ by $-Y_i$
  - Then $a^t Y > 0$ for all patterns irrespective of their class.
- If a linear discriminant functin can correctly classify the given dataset, then the dataset is called *linearly separable*.

# How to find $a$ ?

- We should find a solution to the set of linear inequalities $a^t Y_i > 0$ for all $i$.
- A easy way is to define a criterion function $J(a)$ that is minimized if $a$ is a solution vector.
- Directly solving $\nabla J(a) = 0$ may not be always possible.
- An iterative method for finding a solution is to apply *gradient descent (Hill climbing)* methods.
- Gradient descent procedures are popular in many engineering applications.

# Gradient Descent Procedures

- Basic gradient descent is very simple.
- We call the $a$ in $i$ th iteration as $a(i)$.
- We start with some arbitrarily chosen weight vector $a(1)$ and compute the gradient vector $\nabla J(a(1))$.
- The next value $a(2)$ is obtained by moving some distance from $a(1)$ in the direction of steepest descent, i.e., along the negative of the gradient.

# Gradient Descent Procedures

- In general $a(k+1)$ is obtained from $a(k)$ by the equation

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

  where $\eta(k)$ is the learning rate that sets the step size. $\eta(k)$ depends on $k$.
- The procedure may be terminated if $|\eta(k)\nabla J(a(k))| < \theta$ for some small $\theta$.
- Step size $\eta(k)$ should be carefully chosen.
  - If $\eta(k)$ is too small then the convengence process will be needlessly slow. That is, number of iterations will be large.
  - On the otherhand, if $\eta(k)$ is too large, the correction process will overshoot and can even diverge.

# How to find the step size?

- Let us write $a(k)$ as $a_k$.
- Then, $a_{k+1} = a_k - \eta_k \nabla J(a_k)$.
- We like to get a principled method for finding the learning rate.

# Taylor series

- $f(x + \delta x) = f(x) + f'(x)\delta x + (1/2!)f''(x)\delta x^2 + \cdots$
- A good approximation (in most cases) of $f(x + \delta x)$ is $f(x) + f'(x)\delta x + (1/2!)f''(x)\delta x^2$.
  This is called as second order approximation.
- Similarly for multidimensional case, approximately $f(X + \delta X)$ is

$$f(X) + (\nabla f(X))^t \delta X + \frac{1}{2!}\delta X^t \mathbf{H} \delta X$$

where $\mathbf{H}$ is *Hessian Matrix*, $d \times d$ matrix for which the $i, j$ th entry is $\partial^2 f / \partial x_i \partial x_j$.

# Taylor series for the criterion

- $$J(a_{k+1}) = J(a_k) + (\nabla J(a_k))^t(a_{k+1} - a_k) \quad + \frac{1}{2!}(a_{k+1} - a_k)^t \mathbf{H}(a_{k+1} - a_k)$$

- $$\begin{aligned} a_{k+1} &= a_k - \eta_k \nabla J(a_k) \\ a_{k+1} - a_k &= -\eta_k \nabla J(a_k) \end{aligned}$$

- So,

$$J(a_{k+1}) = J(a_k) + \underbrace{(-\eta_k ||\nabla J(a_k)||^2) + \frac{1}{2!}\eta_k^2\, \nabla J(a_k)^t \mathbf{H} \nabla J(a_k)}_{P}$$

- P should be minimized to get $J(a_{k+1})$ as low as possible.

# Optimal learning rate

- $$P = (-\eta_k ||\nabla J(a_k)||^2) + \frac{1}{2!}\eta_k^2 \, \nabla J(a_k)^t \mathbf{H} \nabla J(a_k)$$

  By doing

  $$\frac{dP}{d\eta_k} = -||\nabla J(a_k)||^2 + \eta_k \nabla J(a_k)^t \mathbf{H} \nabla J(a_k) = 0$$

  We get

  $$\eta_k = \frac{||\nabla J(a_k)||^2}{\nabla J(a_k)^t \mathbf{H} \nabla J(a_k)}$$

# Newton's Descent

- This is yet another unconstrained optimization method
- $a_{k+1}$ is so chosen to minimize the second order expansion of $J(a)$.
  - *This is in contrast with the gradient descent method where only upto the first order is considered.*
- Let $a_s$ denotes the solution vector, i.e., $J(a_s)$ is minimum.

$$J(a) = J(a_s) + (\nabla J(a_s))^t(a - a_s) + \frac{1}{2}(a - a_s)^t\mathbf{H}(a - a_s)$$

But $\nabla J(a_s) = 0$
So, $J(a) = J(a_s) + \frac{1}{2}(a - a_s)^t\mathbf{H}(a - a_s)$

# Newton's Descent

- $J(a) = J(a_s) + \frac{1}{2}(a - a_s)^t \mathbf{H}(a - a_s)$
  So, $\nabla J(a) = 0 + \mathbf{H}(a - a_s)$
  So we get, $a_s = a - \mathbf{H}^{-1} \nabla J(a)$
  That is, $a_{k+1} = a_k - \mathbf{H}^{-1} \nabla J(a)$
- The corrections in the *gradient descent* are in the negative direction of the gradient.
- But in Newton's method the direction of correction can be in any direction, depending upon the Hessian Matrix.

# Newton's Descent

- Newton's method can give greater improvement per step than simple gradient descent method.
- If the objective is quadratic, then Newton's method can find the solution in a single step.
- Biggest drawback of this method is that it requires finding inverse of the Hessian, which is computationally a demanding step.

# Perceptron: Gradient descent

- Perceptron criterion:

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

where $\mathcal{Y}$ is the set of misclassified patterns by the discriminant defined by $a$.

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

- Hence the update rule is, $a_{k+1} = a_k + \eta_k \sum_{Y \in \mathcal{Y}_k} Y$ where $\mathcal{Y}_k$ is the set of misclassified patterns by $a_k$.

# Batch Perceptron

- $$a_{k+1} = a_k + \eta_k \sum_{Y \in \mathcal{Y}_k} Y$$

  where $\mathcal{Y}_k$ is the set of misclassified patterns by $a_k$.
- Normally, $\eta_k$ is taken as 1. This is called *fixed increment* rule.
- For linearly separable datasets, it is proved that the learning converges to a solution.

# Perceptron: Single Sample Correction

- This is a variant of the Batch Perceptron.
- Start with a arbitrary $a_0$.
- Whenever a pattern $Y$ is misclassified, i.e., $a_k^t Y < 0$ then $a_{k+1} = a_k + Y$.
- The above step needs to be done repeatedly (training set needs to be scanned again and again) until all the training patterns are correctly classified.
- These kind of learning procedures are called *error correcting procedures* because $a$ is updated only when error occurs.

# Perceptron: Single Sample Correction

- It is easy to see geometrically what is happening.
- If $Y$ is misclassified by $a_k$ then $a_k^t Y < 0$.
- $a_{k+1} = a_k + Y \;\Rightarrow\; a_{k+1}^t Y = a_k^t Y + ||Y||^2$.
- Hence, the correction is to move the weight vector in a good direction.

# Perceptron Convergence

- **Theorem 5.1 (Page 230):** If the training samples are linearly separable, then the single sample correction method finds a solution in a finite number of steps.
- Let $\hat{a}$ be a solution, then $\hat{a}^t Y_i > 0$ for all $i$.
- Let

$$\beta^2 = \text{max}_i ||Y_i||^2,$$

$$\gamma = \text{min}_i [\hat{a}^t Y_i],$$

$$\alpha = \frac{\beta^2}{\gamma},$$

# Perceptron Convergence

- Then the proof is to show that, after every correction,

$$||a_{k+1} - \alpha\hat{a}||^2 \leq ||a_k - \alpha\hat{a}||^2 - \beta^2$$

- An upperbound on the number of corrections is

$$\frac{||a_1 - \alpha\hat{a}||^2}{\beta^2}$$

- Read the text from page 230 to page 232.

# Improvements to Perceptron

- There are many improvements to the Perceptron.
- Some extensions are to find the best possible solution among various solutions. Instead of saying $a^t Y_i > 0$, to allow a margin $b$, i.e., to say that $a^t Y_i > b$ for all $Y_i$ in the training set.
- Some extensions are towards finding a good linear discriminant even when the dataset is non-separable. For example, The *Widrow-Hoff* or *Least Mean Square(LMS)* procedure is one of such kind.
- Linear Support Vector Machines can also be seen as an improvement over the Perceptron.