# Chapter 7

# Neural Networks: A Statistical Pattern Recognition Perspective

Neural Networks: A Classroom Approach

Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

# Statistical Framework

- The natural framework for studying the design and capabilities of pattern classification machines is statistical
- Nature of information available for decision making is probabilistic

# Feedforward Neural Networks

- Have a natural propensity for performing classification tasks
- Solve the problem of *recognition of patterns* in the input space or *pattern space*
- Pattern recognition:
  - Concerned with the problem of decision making based on complex patterns of information that are probabilistic in nature.
- Network outputs can be shown to find proper interpretation of conventional statistical pattern recognition concepts.

# Pattern Classification

- Linearly separable pattern sets:
  - only the simplest ones
- Iris data: classes overlap
- Important issue:
  - Find an optimal placement of the discriminant function so as to minimize the number of misclassifications on the given data set, and simultaneously minimize the probability of misclassification on unseen patterns.

# Notion of Prior

- The prior probability $P(C_k)$ of a pattern belonging to class $C_k$ is measured by the fraction of patterns in that class assuming an infinite number of patterns in the training set.

- Priors influence our decision to assign an unseen pattern to a class.

# Assignment Without Information

- In the absence of all other information:
- Experiment:
  - In a large sample of outcomes of a coin toss experiment the ratio of Heads to Tails is 60:40
  - Is the coin biased?
  - Classify the next (unseen) outcome and minimize the probability of mis-classification
  - (Natural and safe) Answer: Choose Heads!

# Introduce Observations

- Can do much better with an observation…
- Suppose we are allowed to make a single measurement of a feature x of each pattern of the data set.
- x is assigned a set of discrete values $\{x^1, x^2, \ldots, x^d\}$

# Joint and Conditional Probability

- Joint probability $P(C_k, x^l)$ that $x^l$ belongs to $C_k$ is the fraction of total patterns that have value $x^l$ while belonging to class $C_k$

- Conditional probability $P(x^l | C_k)$ is the fraction of patterns that have value $x^l$ given only patterns from class $C_k$

# Joint Probability = Conditional Probability × Class Prior

Number of patterns with value $x^l$ in class $C_k$

Total number of patterns

$$\frac{n_{x^l|C_k}}{n} = \frac{n_{x^l|C_k}}{n_{C_k}} \times \frac{n_{C_k}}{n}$$

Number of patterns in class $C_k$

$$\underbrace{P(C_k, x^l)}_{\text{joint}} = \underbrace{P(x^l|C_k)}_{\text{class conditional}} \times \underbrace{P(C_k)}_{\text{prior}}$$

# Posterior Probability: Bayes' Theorem

- Note: $P(C_k, x^l) = P(x^l, C_k)$
- $P(C_k, x^l)$ is the posterior probability: probability that feature value $x^l$ belongs to class $C_k$

$$P(x^l, \mathcal{C}_k) = P(\mathcal{C}_k|x^l)P(x^l)$$
$$= P(x^l|\mathcal{C}_k)P(\mathcal{C}_k) = P(\mathcal{C}_k, x^l)$$

- **Bayes' Theorem**

$$P(\mathcal{C}_k|x^l) = \frac{P(x^l|\mathcal{C}_k)P(\mathcal{C}_k)}{P(x^l)}$$

# Bayes' Theorem and Classification

- Bayes' Theorem provides the key to classifier design:
  - Assign pattern $x^l$ to class $C_K$ for which the posterior is the highest!
- Note therefore that all posteriors must sum to one $\sum_{k=1}^{C} P(\mathcal{C}_k|x^l) = 1$
- And $P(x^l) = \sum_{k=1}^{C} P(x^l|\mathcal{C}_k)P(\mathcal{C}_k)$

# Bayes' Theorem for Continuous Variables

- Probabilities for discrete intervals of a feature measurement are then replaced by probability density functions p(x)

$$P(x_l \le x \le x_u) = \int_{x_l}^{x_u} p(x)dx$$

$$p(x) = \sum_{k=1}^{C} p(x|\mathcal{C}_k)P(\mathcal{C}_k)$$

$$P(\mathcal{C}_k|x) = \frac{p(x|\mathcal{C}_k)P(\mathcal{C}_k)}{p(x)}$$

# Gaussian Distributions

- Two-class one dimensional Gaussian probability density function

Distribution Mean and Variance

$$\bar{x}_i = E[x_i] = \int_{-\infty}^{\infty} x_i \, p(x_i) dx_i$$

$$\sigma_i^2 = E[(x_i - \bar{x}_i)^2] = \int_{-\infty}^{\infty} (x_i - \bar{x}_i)^2 \, p(x_i) dx_i$$

$$p(x_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left( -\frac{(x_i - \bar{x}_i)^2}{2\sigma_i^2} \right)$$

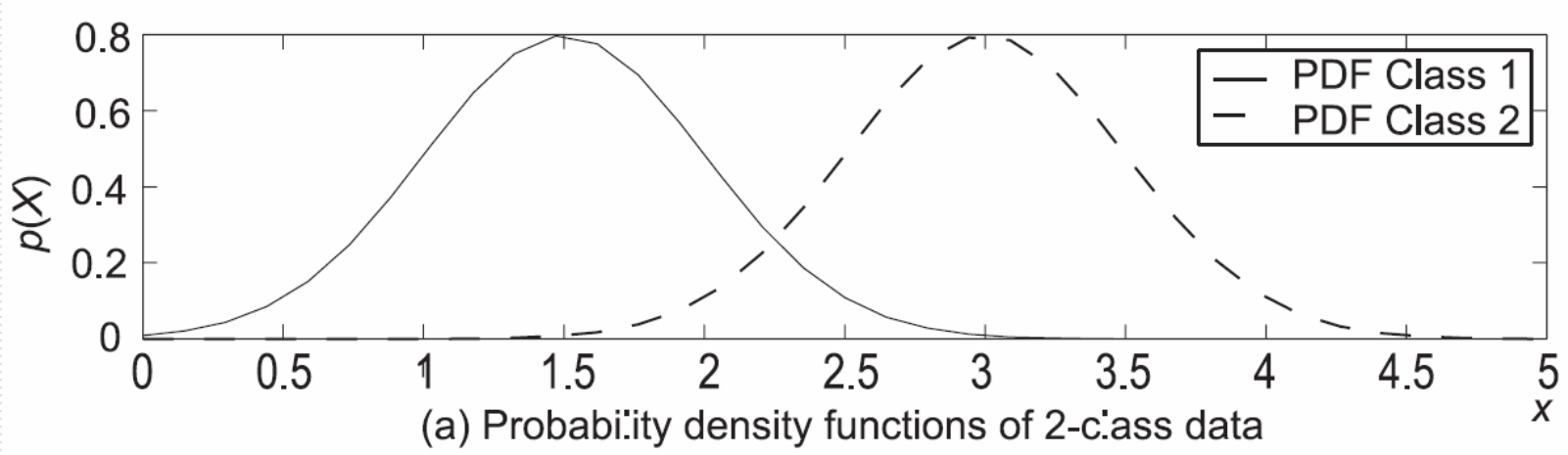
normalizing factor

variance

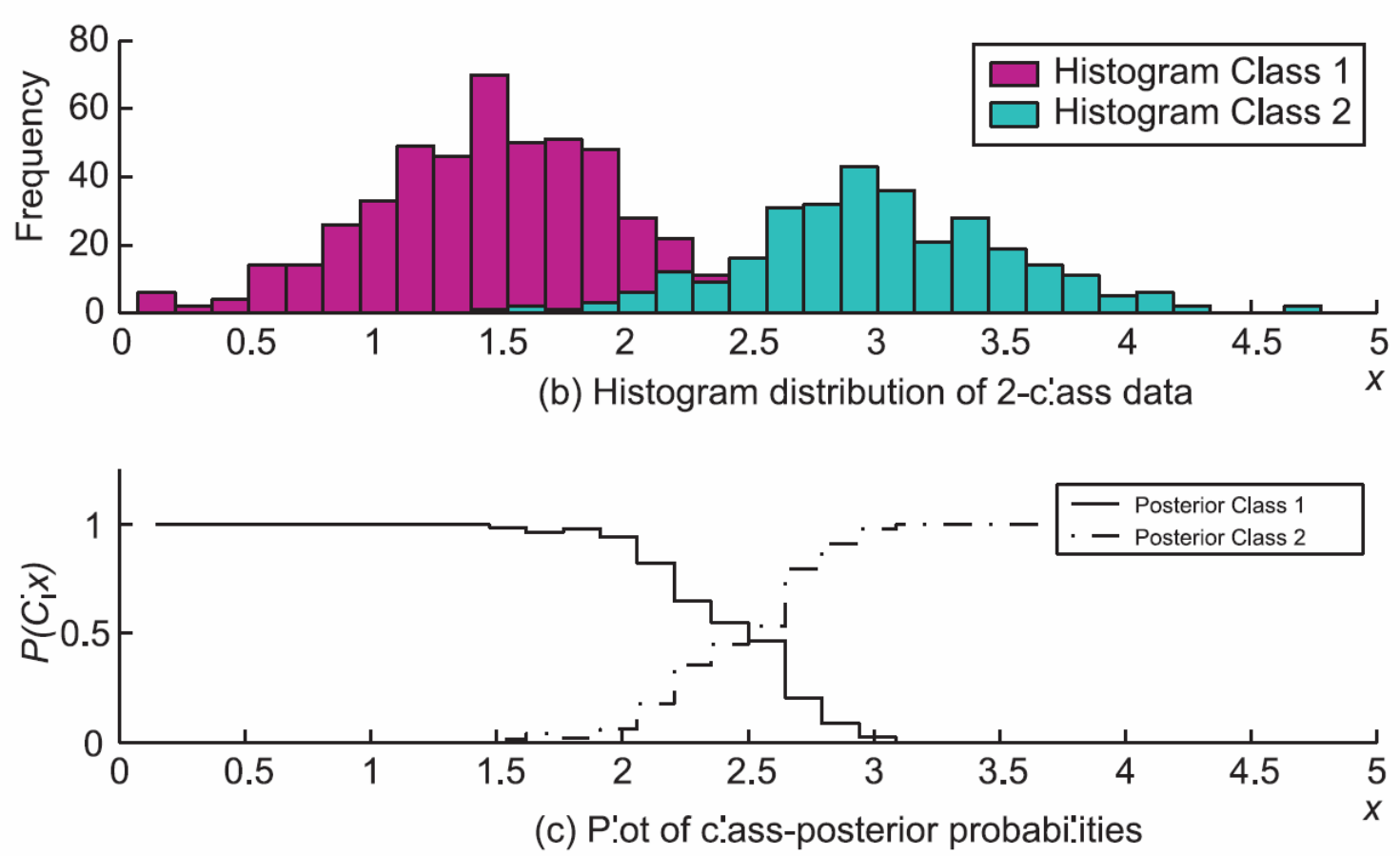
mean

# Example of Gaussian Distribution

- Two classes are assumed to be distributed about means 1.5 and 3 respectively, with equal variances 0.25.



# Example of Gaussian Distribution



# Extension to n-dimensions

- The probability density function expression extends to the following

$$p(X) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{K}|}} \exp\left(-\frac{1}{2}(X - \bar{X})^T \mathbf{K}^{-1} (X - \bar{X})\right)$$

- Mean $\bar{X} = E[X]$
- Covariance matrix

$$\mathbf{K} = E[(X - \bar{X})(X - \bar{X})^T]$$

# Covariance Matrix and Mean

- Covariance matrix
  - describes the shape and orientation of the distribution in space
- Mean
  - describes the translation of the scatter from the origin

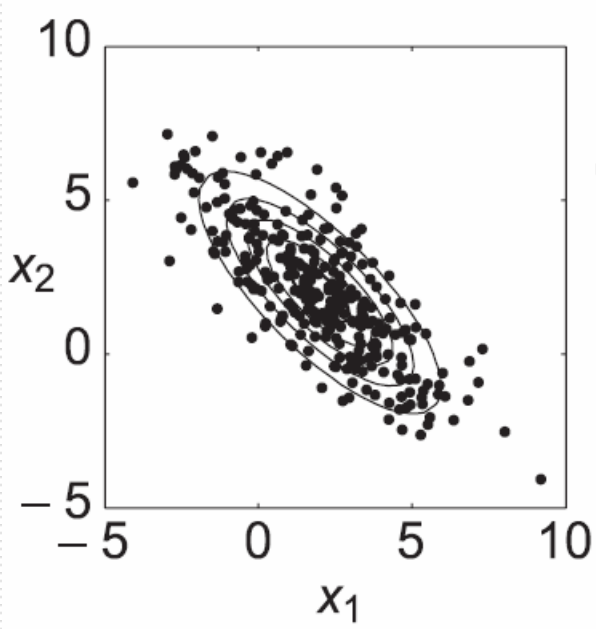
# Covariance Matrix and Data Scatters

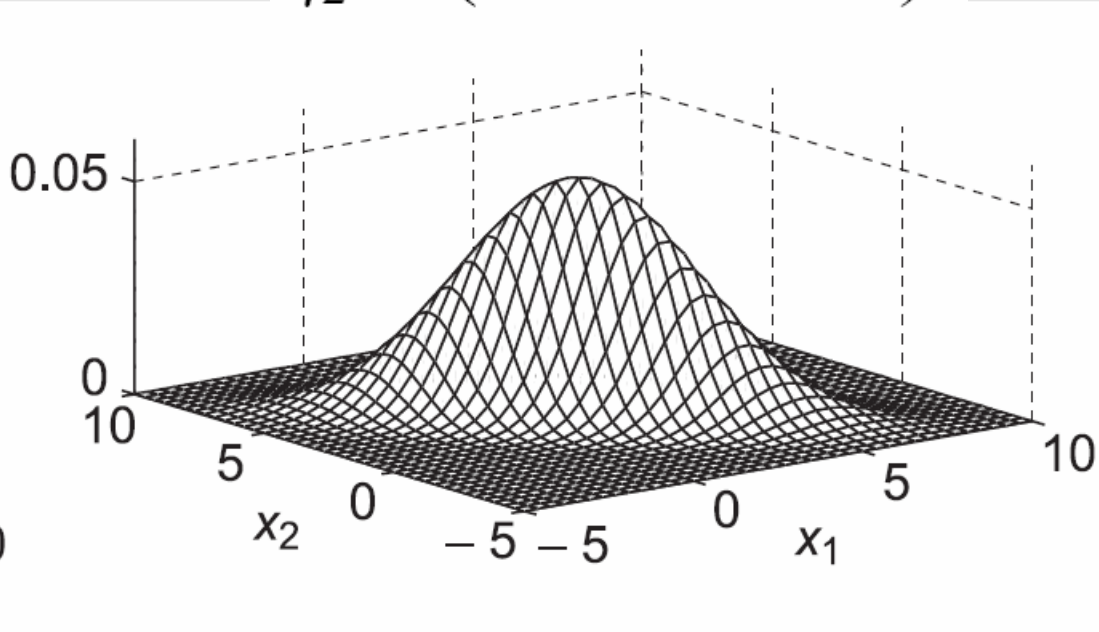
$$\mathbf{K}_1 = \begin{pmatrix} 0.9 & -0.7 \\ -0.7 & 0.9 \end{pmatrix}$$

$$\lambda_1 = 0.2, \lambda_2 = 1.6$$

$$\eta_1 = (-0.707 \ -0.707)^T$$

$$\eta_2 = (-0.707 \ 0.707)^T$$



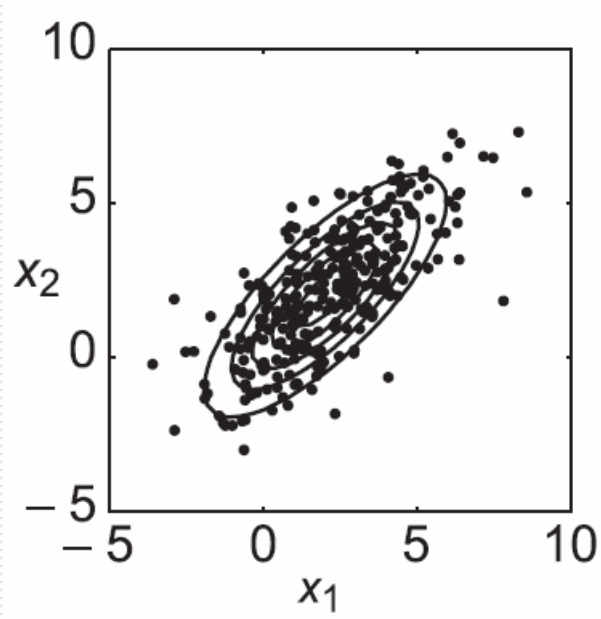
# Covariance Matrix and Data Scatters

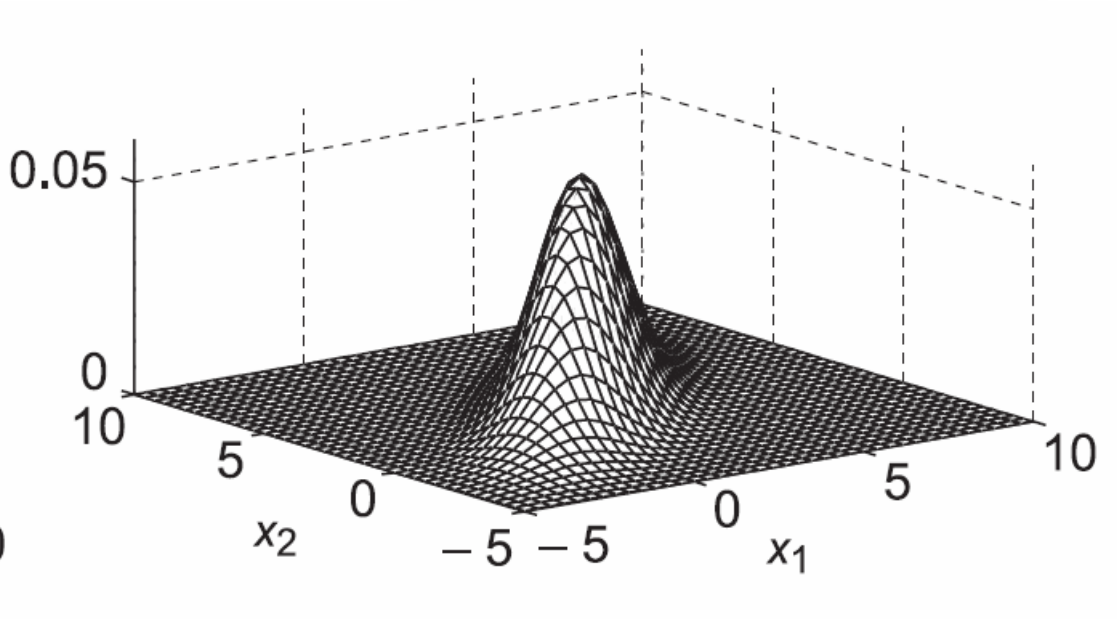
$$\mathbf{K}_2 = \begin{pmatrix} 0.9 & 0.7 \\ 0.7 & 0.9 \end{pmatrix}$$

$$\lambda_1 = 0.2, \lambda_2 = 1.6$$

$$\eta_1 = (-0.707\ 0.707)^T$$

$$\eta_2 = (0.707\ 0.707)^T$$



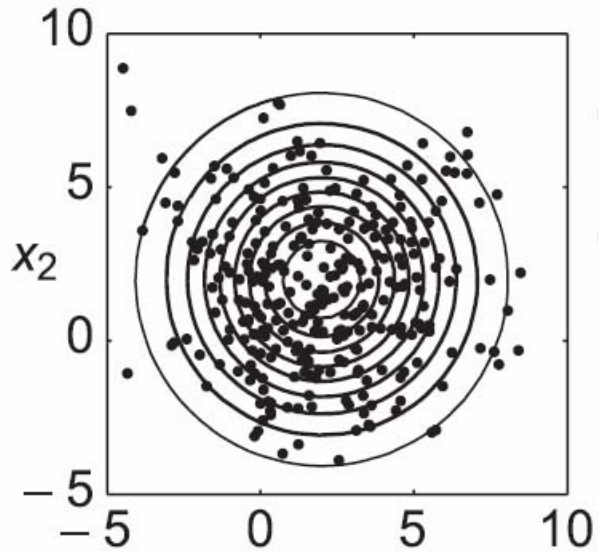



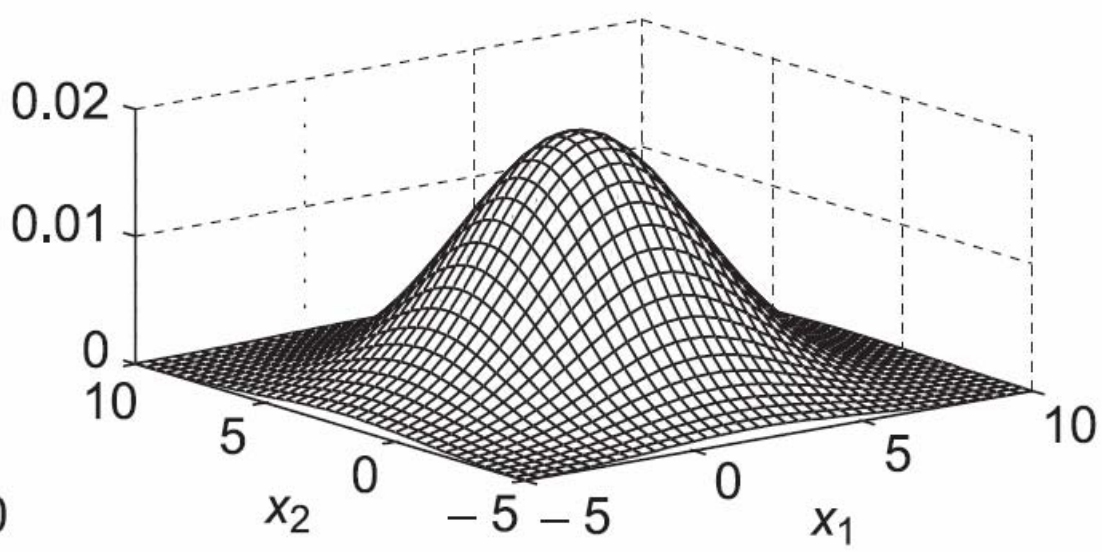
# Covariance Matrix and Data Scatters

$$\mathbf{K}_3 = \begin{pmatrix} 1.6 & 0 \\ 0 & 1.6 \end{pmatrix}$$

$$\lambda_1 = 1.6, \lambda_2 = 1.6$$

$$\eta_1 = (1\ 0)^T, \eta_2 = (0\ 1)^T$$



# Probability Contours

- Contours of the probability density function are loci of equal *Mahalanobis distance*

$$\Delta^2 = (X - \bar{X})^T \mathbf{K}^{-1} (X - \bar{X})$$

# MATLAB Code to Generate Probability Contours

```
mu = [2 2];
alpha1= 0.1;
alpha2= 0.8;
covar = alpha1*[1,1]'*[1,1] + alpha2*[1,-
      1]'*[1,-1];
invcov = inv(covar);
gridres = 30;
min = 0;
max = 5;
data = linspace(min, max, gridres);
[X1, X2] = meshgrid(data, data);
X = [X1(:), X2(:)];
[n, d] = size(X); data

X = X - ones(n, 1)*mu;
mahadist = sum(((X*invcov).*X), 2);
p = exp(-0.5*mahadist)./sqrt((2*pi)^d
      *det(covar));
P = reshape(p, gridres, gridres);
figure;
contour(X1, X2, P, 'b');
hold on

numsamp = 300;
[eta, lambda] = eig(covar);
coeffs = randn(numsamp,
      d)*sqrt(lambda);

samples = coeffs*eta' +  ones(numsamp,
      1)*mu ;
%Plot the samples
plot(samples(:,1), samples(:,2), 'k.');
xlabel('X1')
ylabel('X2')
```

# Classification Decisions with Bayes' Theorem

- Key: Assign X to Class $C_k$ such that

or,

$$P(\mathcal{C}_k|X) = \max_j \{P(\mathcal{C}_j|X)\}$$

$$p(X|\mathcal{C}_k)P(\mathcal{C}_k) > p(X|\mathcal{C}_j)P(\mathcal{C}_j) \quad \forall j \neq k$$

# Placement of a Decision Boundary

- Decision boundary separates the classes in question
- Where do we place decision region boundaries such that the probability of misclassification is minimized?

# Quantifying the Classification Error

- Example: 1-dimension, 2 classes identified by regions $R_1$, $R_2$
- $P_{error} = P(x \in R_1, C_2) + P(x \in R_2, C_1)$

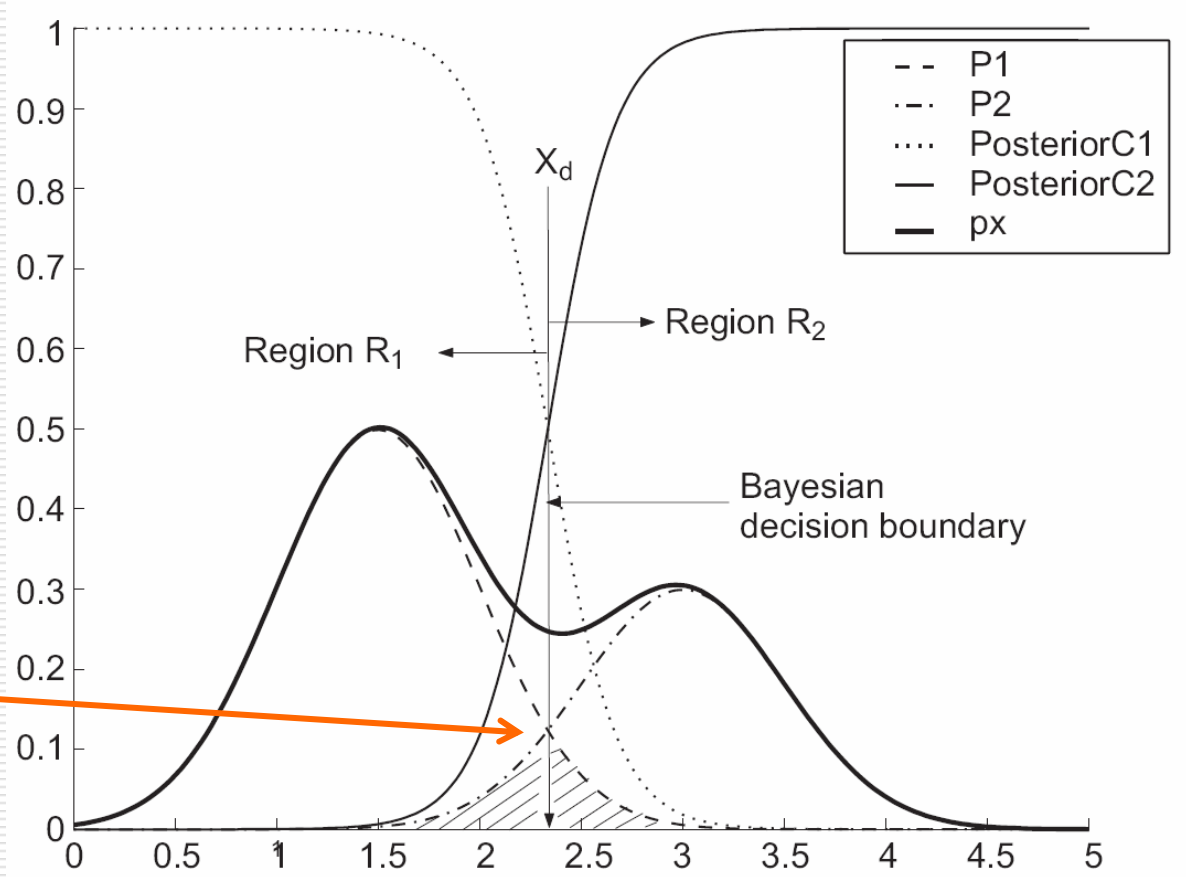
$$P_{\text{error}} = \int_{-\infty}^{x_d} p(x|\mathcal{C}_2)P(\mathcal{C}_2)dx + \int_{x_d}^{\infty} p(x|\mathcal{C}_1)P(\mathcal{C}_1)dx$$

- Place decision boundary such that
  - point x lies in $R_1$ (decide $C_1$) if $p(x|C_1)P(C_1) > p(x|C_2)P(C_2)$
  - point x lies in $R_2$ (decide $C_2$) if $p(x|C_2)P(C_2) > p(x|C_1)P(C_1)$

# Optimal Placement of A Decision Boundary

Bayesian Decision Boundary:

The point where the unnormalized probability density functions crossover



# Probabilistic Interpretation of a Neuron Discriminant Function

- An artificial neuron implements the discriminant function:

$$y_j(X) = \sum_{i=1}^{n} w_{ij} x_i + w_{0j}$$
$$= W_j^T X + w_{0j}$$

- Each of $C$ neurons implements its own discriminant function for a $C$-class problem

- An arbitrary input vector $X$ is assigned to class $C_k$ if neuron $k$ has the largest activation

$$y_k(X) = \max_j \{y_j(X)\}$$

# Probabilistic Interpretation of a Neuron Discriminant Function

- An optimal Bayes' classification chooses the class with maximum posterior probability $P(C_j|X)$
- Discriminant function $y_j = P(X|C_j)\,P(C_j)$
  - $y_j$ notation re-used for emphasis
- Relative magnitudes are important: use any monotonic function of the probabilities to generate a new discriminant function

$$y_j(X) = \ln p(X|\mathcal{C}_j) + \ln P(\mathcal{C}_j)$$

# Probabilistic Interpretation of a Neuron Discriminant Function

- Assume an n-dimensional density function
- This yields,

$$y_j(X) = -\frac{1}{2}(X - \bar{X}_j)^T \mathbf{K}_j^{-1}(X - \bar{X}_j) - \frac{1}{2}\ln|\mathbf{K}_j| - \frac{n}{2}\ln 2\pi + \ln P(\mathcal{C}_j)$$

- Ignore the constant term, assume that all covariance matrices are the same:

$$\begin{aligned} y_j(X) &= -\frac{1}{2}(X - \bar{X}_j)^T \mathbf{K}^{-1}(X - \bar{X}_j) + \ln P(\mathcal{C}_j) \\ &= \bar{X}_j^T \mathbf{K}^{-1} X - \frac{1}{2}\bar{X}_j^T \mathbf{K}^{-1}\bar{X}_j + \ln P(\mathcal{C}_j) \\ &= W_j^T X + w_{0j} \end{aligned}$$

# Plotting a Bayesian Decision Boundary: 2-Class Example

- Assume classes $C_1$, $C_2$, and discriminant functions of the form,

$$y_1(X) = -\frac{1}{2}(X - \bar{X}_1)^T \mathbf{K}_1^{-1}(X - \bar{X}_1) - \frac{1}{2}\ln|\mathbf{K}_1| - \frac{n}{2}\ln 2\pi + \ln P(\mathcal{C}_1)$$

$$y_2(X) = -\frac{1}{2}(X - \bar{X}_2)^T \mathbf{K}_2^{-1}(X - \bar{X}_2) - \frac{1}{2}\ln|\mathbf{K}_2| - \frac{n}{2}\ln 2\pi + \ln P(\mathcal{C}_2)$$

- Combine the discriminants $y(X) = y_2(X) - y_1(X)$
- New rule:
  - Assign X to $C_2$ if $y(X) > 0$; $C_1$ otherwise

# Plotting a Bayesian Decision Boundary: 2-Class Example

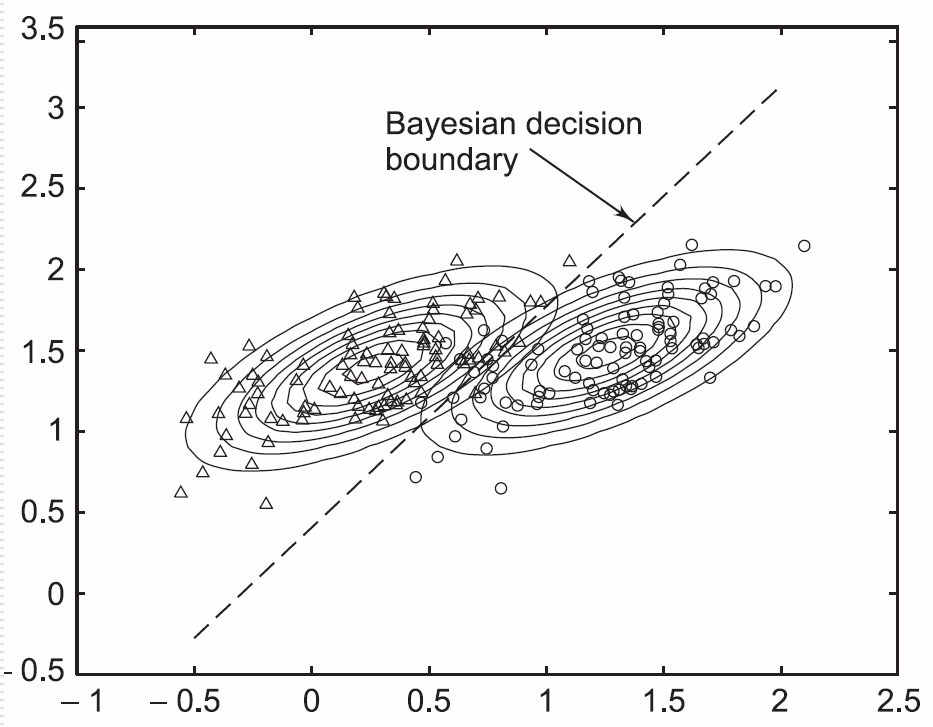
- This boundary is elliptic

$$y(X) = \frac{1}{2}(X - \bar{X}_1)^T \mathbf{K}_1^{-1}(X - \bar{X}_1) - \frac{1}{2}(X - \bar{X}_2)^T \mathbf{K}_2^{-1}(X - \bar{X}_2)$$

$$+ \frac{1}{2}\ln\frac{|\mathbf{K}_1|}{|\mathbf{K}_2|} - \ln\frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)}$$

- If $K_1 = K_2 = K$ then the boundary becomes linear…

# Bayesian Decision Boundary




$$\bar{X}_1 = (0.1597 \quad 1.3541)$$

$$\bar{X}_2 = (1.1597 \quad 1.4541)$$

$$\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{K} = \begin{pmatrix} 0.1726 & 0.0912 \\ 0.0912 & 0.1020 \end{pmatrix}$$

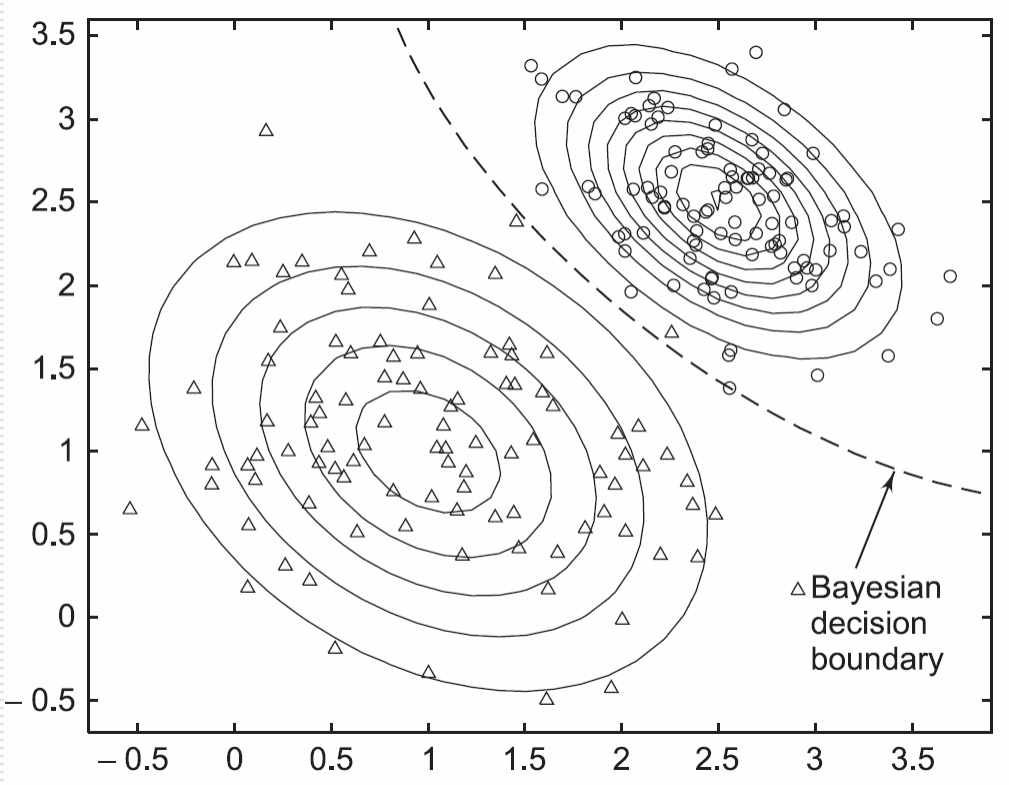
# Bayesian Decision Boundary

$$\bar{X}_1 = (1 \quad 1)$$

$$\bar{X}_2 = (2.5 \quad 2.5)$$

$$\mathbf{K}_1 = \begin{pmatrix} 0.6 & -0.2 \\ -0.2 & 0.6 \end{pmatrix}$$

$$\mathbf{K}_2 = \begin{pmatrix} 0.2 & -0.1 \\ -0.1 & 0.2 \end{pmatrix}$$



# Cholesky Decomposition of Covariance Matrix K

- Returns a matrix Q such that $Q^TQ = K$ and Q is upper triangular

$$
\begin{aligned}
(X-\bar{X})^T\mathbf{K}^{-1}(X-\bar{X}) &= (X-\bar{X})^T(\mathbf{Q}^T\mathbf{Q})^{-1}(X-\bar{X}) \\
&= (X-\bar{X})^T\mathbf{Q}^{-1}(\mathbf{Q}^T)^{-1}(X-\bar{X}) \\
&= (X-\bar{X})^T\mathbf{Q}^{-1}\left((X-\bar{X})^T\mathbf{Q}^{-1}\right)^T
\end{aligned}
$$

# MATLAB Code to Plot Bayesian Decision Boundaries

```
[X Y]=meshgrid(xspace, yspace);
gridpoints=[X(:) Y(:)];
npoints = size(gridpoints, 1);

% Class conditionals are to be
      stored columnwise
px_C = zeros(npoints, 2);

% Assuming a 2-dimensional input
normalfact = (2*pi);
for i = 1:2
dist = gridpoints-
      (ones(npoints,1)*centres(i,:));
Q = chol(covars(:,:,i));
term = dist*inv(Q);
px_C(:,i) = exp(-0.5*sum(term.*term,
      2))./(normalfact*prod(diag(Q)));
end
postnum =
      (ones(nsamples,1)*priors).*px_C;
px = sum(postnum, 2);
posteriors = postnum./(px*ones(1, 2));
p1_x = reshape(posteriors(:, 1),
      size(X));
p2_x = reshape(posteriors(:, 2),
      size(X));
px_1 = reshape(px_C(:,1), size(X));
px_2 = reshape(px_C(:,2), size(X));
contour(xrange,yrange, p1_x,[0.5
      0.5],'k-.');
contour(xrange, yrange, px_1);
contour(xrange, yrange, px_2);
```

# Interpreting Neuron Signals as Probabilities: Gaussian Data

- Gaussian Distributed Data
- 2-Class data, $K_2 = K_1 = K$

$$p(X|\mathcal{C}_k) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{K}|}} \exp\left(-\frac{1}{2}(X - \bar{X})^T \mathbf{K}^{-1}(X - \bar{X})\right)$$

- From Bayes' Theorem, we have the posterior probability

$$P(\mathcal{C}_j|X) = \frac{p(X|\mathcal{C}_j)P(\mathcal{C}_j)}{p(X|\mathcal{C}_1)P(\mathcal{C}_1) + p(X|\mathcal{C}_2)P(\mathcal{C}_2)} \qquad j = 1, 2$$

# Interpreting Neuron Signals as Probabilities: Gaussian Data

- Consider Class 1

$$P(\mathcal{C}_1|X) = \frac{p(X|\mathcal{C}_1)P(\mathcal{C}_1)}{p(X|\mathcal{C}_1)P(\mathcal{C}_1) + p(X|\mathcal{C}_2)P(\mathcal{C}_2)}$$

$$= \frac{1}{1 + \frac{p(X|\mathcal{C}_2)P(\mathcal{C}_2)}{p(X|\mathcal{C}_1)P(\mathcal{C}_1)}}$$

$$= \frac{1}{1 + e^{-z_1}}$$

Sigmoidal neuron ?

# Interpreting Neuron Signals as Probabilities: Gaussian Data

- We substituted

$$e^{-z_1} = \frac{p(X|\mathcal{C}_2)P(\mathcal{C}_2)}{p(X|\mathcal{C}_1)P(\mathcal{C}_1)}$$

- or,

$$z_1 = \ln\left[\frac{p(X|\mathcal{C}_1)P(\mathcal{C}_1)}{p(X|\mathcal{C}_2)P(\mathcal{C}_2)}\right]$$

$$= \ln p(X|\mathcal{C}_1) - \ln p(X|\mathcal{C}_2) + \ln \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)}$$

$$= (\bar{X}_1^T - \bar{X}_2^T)\mathbf{K}^{-1}X - \frac{1}{2}\bar{X}_1^T\mathbf{K}^{-1}\bar{X}_1 + \frac{1}{2}\bar{X}_2^T\mathbf{K}^{-1}\bar{X}_2 + \ln \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)}$$

$$= W_1^T X + w_{01}$$

Neuron activation !

# Interpreting Neuron Signals as Probabilities

- **Bernoulli Distributed Data**
- Random variable $x_i$ takes values 0,1
- Bernoulli distribution

$$p(x_i|\mathcal{C}_k) = (P_{ik})^{x_i}(1 - P_{ik})^{1-x_i}$$

- Extending this result to an n-dimensional vector of *independent* input variables

$$p(X|\mathcal{C}_k) = \prod_{i=1}^{n}(P_{ik})^{x_i}(1 - P_{ik})^{1-x_i}$$

# Interpreting Neuron Signals as Probabilities: Bernoulli Data

- Bayesian discriminant $y_k(X) = \ln p(X|\mathcal{C}_k) + \ln P(\mathcal{C}_k)$

$$y_k(X) = \ln\left[\prod_{i=1}^{n}(P_{ik})^{x_i}(1-P_{ik})^{1-x_i}\right] + \ln P(\mathcal{C}_k)$$

$$= \sum_{i=1}^{n}\left[x_i \ln P_{ik} + (1-x_i)\ln(1-P_{ik})\right] + \ln P(\mathcal{C}_k)$$

$$= \sum_{i=1}^{n}\left(\ln P_{ik} - \ln(1-P_{ik})\right)x_i + \sum_{i=1}^{n}\ln(1-P_{ik}) + \ln P(\mathcal{C}_k)$$

$$= \sum_{i=1}^{n} w_{ik}x_i + w_{0k}$$

Neuron activation

# Interpreting Neuron Signals as Probabilities: Bernoulli Data

- Consider the posterior probability for class $C_1$

$$P(\mathcal{C}_1|X) = \frac{p(X|\mathcal{C}_1)P(\mathcal{C}_1)}{p(X|\mathcal{C}_1)P(\mathcal{C}_1) + p(X|\mathcal{C}_2)P(\mathcal{C}_2)} = \frac{1}{1+e^{-z_1}}$$

where

$$z_1 = \ln \frac{p(X|\mathcal{C}_1)\ P(\mathcal{C}_1)}{p(X|\mathcal{C}_2)\ P(\mathcal{C}_2)}$$

# Interpreting Neuron Signals as Probabilities: Bernoulli Data

$$z_1 = \ln\left[\frac{\prod_{i=1}^{n} P_{i1}{}^{x_i}(1 - P_{i1})^{1-x_i} P(\mathcal{C}_1)}{\prod_{i=1}^{n} P_{i2}{}^{x_i}(1 - P_{i2})^{1-x_i} P(\mathcal{C}_2)}\right]$$

$$= \sum_{i=1}^{n}\left(\ln\frac{P_{i1}}{P_{i2}} - \ln\frac{1 - P_{i1}}{1 - P_{i2}}\right)x_i + \sum_{i=1}^{n}\ln\frac{(1 - P_{i1})}{(1 - P_{i2})} + \ln\frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)}$$

$$= \sum_{i=1}^{n} w_{i1}x_i + w_{01}$$

# Multilayered Networks

- The computational power of neural networks stems from their multilayered architecture
  - What kind of interpretation can the outputs of such networks be given?
  - Can we use some other (more appropriate) error function to train such networks?
  - If so, then with what consequences in network behaviour?

# Likelihood

- Assume a training data set $T=\{X_k,D_k\}$ drawn from a joint p.d.f. $p(X,D)$ defined on $\Re^{n\times p}$
- Joint probability or likelihood of T

$$\mathcal{L} = \prod_{k=1}^{Q} p(X_k, D_k)$$

$$= \prod_{k=1}^{Q} p(D_k|X_k)p(X_k)$$

# Sum of Squares Error Function

- Motivated by the concept of maximum likelihood
- Context: neural network solving a classification or regression problem
- Objective: maximize the likelihood function
- Alternatively: minimize negative likelihood:

$$\mathcal{E} = -\ln \mathcal{L} = -\sum_{k=1}^{Q} \ln\ p(D_k|X_k) - \sum_{k=1}^{Q} \ln\ p(X_k)$$

Drop this constant

# Sum of Squares Error Function

- Error function is the negative sum of the log-probabilities of desired outputs conditioned on inputs

$$\mathcal{E} = -\sum_{k=1}^{Q} \ln \; p(D_k|X_k)$$

- A feedforward neural network provides a framework for modelling p(D|X)

# Normally Distributed Data

- Decompose the p.d.f. into a product of individual density functions

$$p(D|X) = \prod_{j=1}^{p} p(d_j|X)$$

- Assume target data is Gaussian distributed

$$d_j = g_j(X) + \epsilon_j$$

- $\epsilon_j$ is a Gaussian distributed noise term
- $g_j(X)$ is an underlying deterministic function

$$g_j(X) = \int d_j \, p(d_j|X) \, dd_j$$
$$= E[d_j|X]$$

# From Likelihood to Sum Square Errors

- Noise term has zero mean and s.d. $\sigma$

$$p(\epsilon_j) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\epsilon_j^2}{2\sigma^2}\right)$$

- Neural network expected to provide a model of $g(X)$

$$d_j \approx f_j(X, W) + \epsilon_j$$

- Since $f(X,W)$ is deterministic $p(d_j|X) = p(\epsilon_j)$

$$p(d_j|X) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(d_j - f_j(X, W))^2}{2\sigma^2}\right)$$

# From Likelihood to Sum Square Errors

$$\mathcal{E} = -\sum_{k=1}^{Q} \ln \prod_{j=1}^{p} \left[ \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( -\frac{\left(d_j^k - f_j(X_k, W)\right)^2}{2\sigma^2} \right) \right]$$

$$= \frac{1}{2\sigma^2} \sum_{k=1}^{Q} \sum_{j=1}^{p} \left( f_j(X_k, W) - d_j^k \right)^2 + pQ \ln\sigma + \frac{pQ}{2} \ln(2\pi)$$

- Neglecting the constant terms yields

$$\mathcal{E} = \frac{1}{2} \sum_{k=1}^{Q} \sum_{j=1}^{p} \left(d_j^k - f_j(X_k, W)\right)^2$$

# Interpreting Network Signal Vectors

- Re-write the sum of squares error function

$$\mathcal{E} = \lim_{Q\to\infty}\left(\frac{1}{2Q}\sum_{k=1}^{Q}\sum_{j=1}^{p}\left(d_j^k - f_j(X_k, W)\right)^2\right)$$

- 1/Q provides averaging, permits replacement of the summations by integrals

$$\mathcal{E} = \frac{1}{2}\sum_{j=1}^{p}\left[\lim_{Q\to\infty}\left(\frac{1}{Q}\sum_{k=1}^{Q}\left(d_j^k - f_j(X_k, W)\right)^2\right)\right]$$

$$= \frac{1}{2}\sum_{j=1}^{p}\iint\left(d_j - f_j(X, W)\right)^2 p(X, d_j)\, dX\, dd_j$$

# Interpreting Network Signal Vectors

- Algebra yields

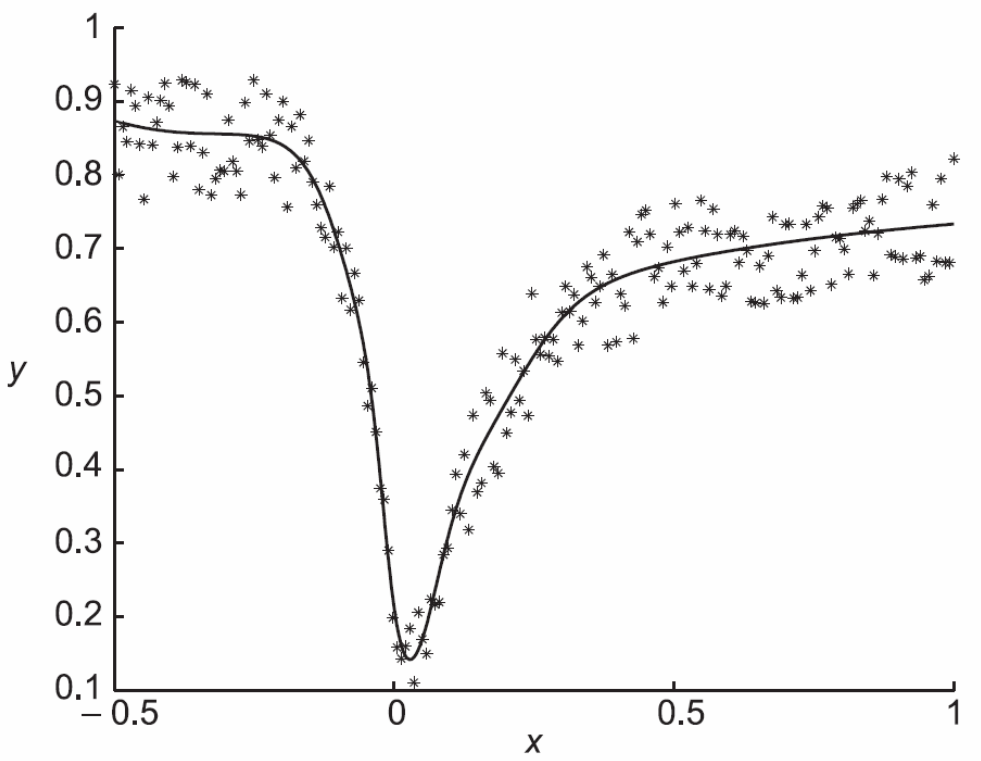
$$\mathcal{E} = \frac{1}{2}\sum_{j=1}^{p}\int \big(g_j(X) - f_j(X, W)\big)^2 p(X)\, dX$$

$$= \frac{1}{2}\sum_{j=1}^{p}\int \big(E[d_j|X] - f_j(X, W)\big)^2 p(X)\, dX$$

- Error is minimized when $f_j(X,W) = E[d_j|X]$ for each $j$.
- The error minimization procedure tends to drive the network map $f_j(X,W)$ towards the conditional average $E[d_j,X]$ of the desired outputs
- At the error minimum, network map approximates the regression of $d$ conditioned on $X$!

# Numerical Example

- Noisy distribution of 200 points distributed about the function
- Used to train a neural network with 7 hidden nodes
- Response of the network is plotted with a continuous line



# Residual Error

- The error expression just presented neglected an integral term shown below

$$\mathcal{E} = \frac{1}{2}\sum_{j=1}^{p}\left[\int \left(g_j(X) - f_j(X, W)\right)^2 \left(\int p(d_j|X)\,dd_j\right) p(X)\,dX + \int \sigma^2(X)p(X)\,dX\right]$$

- If the training environment does manage to reduce the error on the first integral term in to zero, a residual error still manifests due to the second integral term

# Notes…

- The network cannot reduce the error below the average variance of the target data!
- The results discussed rest on the three assumptions:
  - The data set is sufficiently large
  - The network architecture is sufficiently general to drive the error to zero.
  - The error minimization procedure selected does find the appropriate error minimum.

# An Important Point

- Sum of squares error function was derived from maximum likelihood and Gaussian distributed target data
- Using a sum of squares error function for training a neural network does not require target data be Gaussian distributed.
- A neural network trained with a sum of squares error function generates outputs that provide estimates of the average of the target data and the average variance of target data
- Therefore, the specific selection of a sum of squares error function does not allow us to distinguish between Gaussian and non-Gaussian distributed target data which share the same average desired outputs and average desired output variances…

# Classification Problems

- For a $C$-class classification problem, there will be $C$-outputs
- Only 1-of-$C$ outputs will be one
- Input pattern $X_k$ is classified into class $J$ if

$$J = \arg \max_{1 \le j \le C} \mathcal{S}(y_j^k)$$

- A more sophisticated approach seeks to represent the outputs of the network as posterior probabilities of class memberships.

# Advantages of a Probabilistic Interpretation

- We make classification decisions that lead to the smallest error rates.
- By actually computing a prior from the network pattern average, and comparing that value with the knowledge of a prior calculated from class frequency fractions on the training set, one can measure how closely the network is able to model the posterior probabilities.
- The network outputs estimate posterior probabilities from training data in which class priors are naturally estimated from the training set. Sometimes class priors will actually differ from those computed from the training set. A compensation for this difference can be made easily.

# NN Classifers and Square Error Functions

- Recall: feedforward neural network trained on a squared error function generates signals that approximate the conditional average of the desired target vectors
- If the error approaches zero,

$$\mathcal{S}(y_j^k) = E[d_j|X] = \int d_j \; p(d_j|X)\, dd_j$$

- The probability that desired values take on 0 or 1 is the probability of the pattern belonging to that class

$$p(d_j|X) = \sum_{i=1}^{C} \delta(d_j - \delta_{ji}) \; p(\mathcal{C}_i|X)$$

# Network Output = Class Posterior

- The $j^{th}$ output $s_j$ is

$$s_j = E[d_j|X] = \int d_j\, p(d_j|X)\, dd_j$$

$$= \int d_j \left( \sum_i \delta(d_j - \delta_{ji}) p(\mathcal{C}_i|X) \right) dd_j$$

$$= p(\mathcal{C}_j|X)$$

Class posterior

# Relaxing the Gaussian Constraint

- Design a new error function
  - Without the Gaussian noise assumption on the desired outputs
  - Retain the ability to interpret the network outputs as posterior probabilities
  - Subject to constraints:
    - signal confinement to (0,1) and
    - sum of outputs to 1

# Neural Network With A Single Output

- Output $s$ represents Class 1 posterior
- Then $1-s$ represents Class 2 posterior
- The probability that we observe a target value $d_k$ on pattern $X_k$

$$p(d_k|X_k) = s_k^{d_k}(1-s_k)^{1-d_k}$$

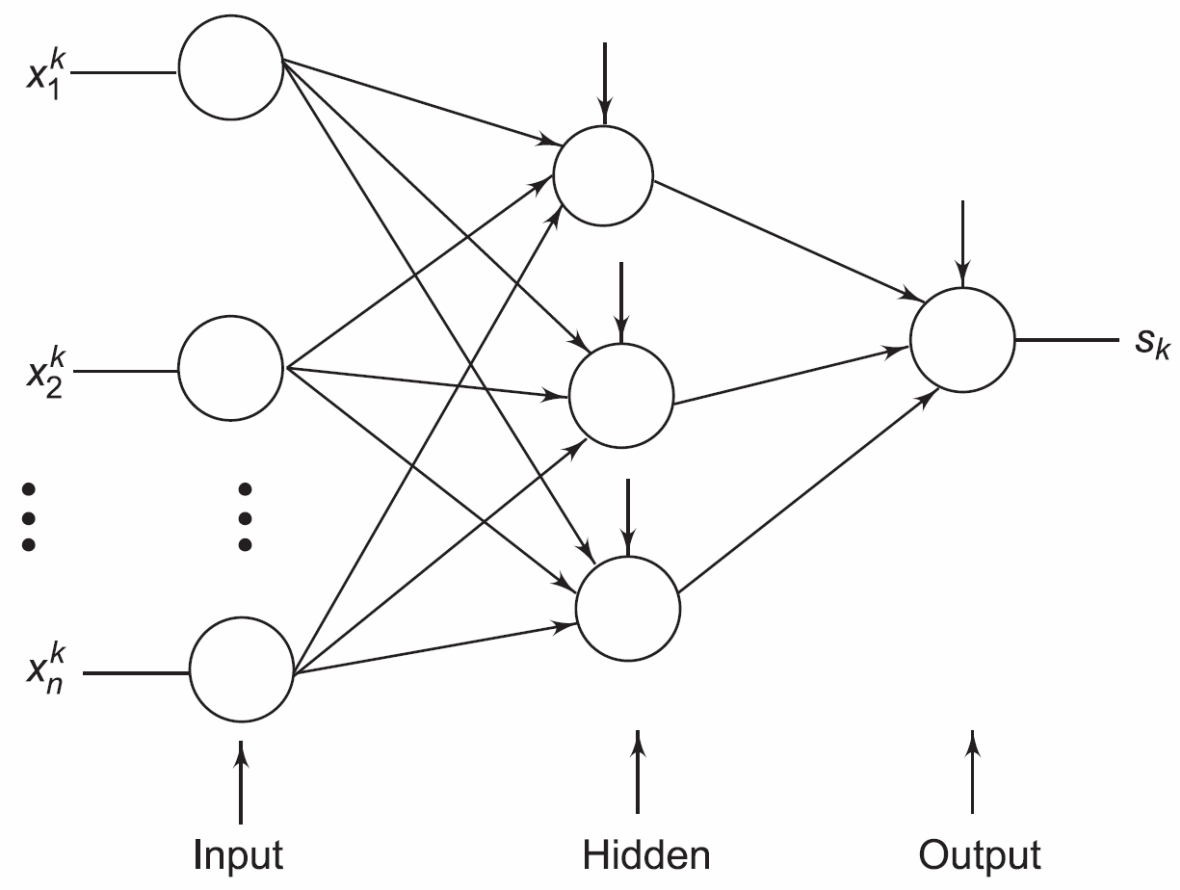
- Problem: Maximize the likelihood of observing the training data set

# Cross Entropy Error Function

- Maximizing the probability of observing desired value $d_k$ for input $X_k$ on each pattern in T
- Likelihood $$\mathcal{L} = \prod_k p(d_k|X_k) = \prod_k (s_k)^{d_k}(1-s_k)^{1-d_k}$$
- Convenient to minimize the negative log-likelihood, which we denote as the error:

$$\begin{aligned}\mathcal{E} &= -\ln\ \mathcal{L} \\ &= -\ln\left(\prod_k (s_k)^{d_k}(1-s_k)^{1-d_k}\right) \\ &= -\sum_k \left(d_k \ln\ s_k + (1-d_k)\ \ln(1-s_k)\right)\end{aligned}$$

# Architecture of Feedforward Network Classifier



# Network Training

- Using the chain rule (Chapter 6) with the cross entropy error function

$$\frac{\partial \mathcal{E}}{\partial w_h^k} = -\delta_k s_h^k$$

- Input – hidden weight derivatives can be found similarly

# C-Class Problem

- Assume a 1 of $C$ encoding scheme
- Network has $C$ outputs

$$p(\mathcal{C}_j|X_k) = s_j^k$$

and

$$p(D_k|X_k) = \prod_{j=1}^{C} (s_j^k)^{d_j^k}$$

- Likelihood function

$$\mathcal{L} = \prod_k \prod_j (s_j^k)^{d_j^k}$$

# Modified Error Function

- Cross entropy error function for the $C$-class case

$$\mathcal{E} = -\sum_k \sum_j d_j^k \ln s_j^k$$

$$= \sum_k \mathcal{E}_k$$

- Minimum value

$$\mathcal{E}_{\min} = -\sum_k \sum_j d_j^k \ln d_j^k$$

- Subtracting the minimum value ensures that the minimum is always zero

$$\tilde{\mathcal{E}} = \mathcal{E} - \mathcal{E}_{\min}$$

$$= -\sum_k \sum_j d_j^k \ln \frac{s_j^k}{d_j^k}$$

# Softmax Signal Function

- Ensures that
  - the outputs of the network are confined to the interval (0,1) and
  - simultaneously all outputs add to 1

$$s_j^k = \frac{e^{y_j^k}}{\sum_{i=1}^{C} e^{y_i^k}}$$

- Is a close relative of the sigmoid

# Error Derivatives

- For hidden-output weights

$$\frac{\partial \mathcal{E}_k}{\partial w_{hj}^k} = -(d_j^k - s_j^k)s_h^k$$

- The remaining part of the error backpropagation algorithm remains intact