## 4.4 Summary of the Back-Propagation Algorithm

- The initial values of the weights and biases can be chosen from a uniform distribution with zero mean unless some prior information is available.

$$\begin{pmatrix} \textit{Weight} \\ \textit{correction} \\ \Delta w_{ij}(n) \end{pmatrix} = \begin{pmatrix} \textit{Learning} \\ \textit{parameter} \\ \eta \end{pmatrix} \begin{pmatrix} \textit{Local} \\ \textit{gradient} \\ \delta_j(n) \end{pmatrix} \begin{pmatrix} \textit{Input signal} \\ \textit{of neuron } j \\ y_i(n) \end{pmatrix}$$

- The local gradient is given by

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{4.14}$$

when the neuron $j$ is in the output layer.

- In the hidden layer, the local gradient is

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{4.24}$$

## The Two Passes of Computation

- In applying the back-propagation algorithm, two distinct passes of computation are distinguished.
- **Forward pass**
  - The weights are not changed in this phase.
  - The function signal appearing at the output of neuron $j$ is computed as

$$y_j(n) = \varphi(v_j(n)) \tag{1}$$

  - Here the local field $v_j(n)$ of neuron $j$ is

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{2}$$

  - In the first hidden layer, $m = m_0$ is the number of input signals $x_i(n)$, $i = 1, \ldots, m_0$, and in Eq. (2)

$$y_i(n) = x_i(n)$$

– In the output layer, $m = m_L$ is the number of outputs Eq. (1).
– The outputs (components of the output vector) are denoted by

$$y_j(n) = \; o_j(n)$$

– These outputs are then compared with the respective desired responses $d_j(n)$, yielding the error signals $e_j(n)$.
– In the forward pass, computation starts from the first hidden layer and terminates at the output layer.

- **Backward pass**
  - In the backward pass, computation starts at the output layer, and ends at the first hidden layer.
  - The local gradient $\delta$ is computed for each neuron by passing the error signal through the network layer by layer.
  - The delta rule of Eq. (4.25) is used for updating the synaptic weights.
  - The weight updates are computed recursively layer by layer.
- The input vector is fixed through each round-trip (forward pass followed by a backward pass).
- After this, the next training (input) vector is presented to the network.

## 4.5 XOR Problem

- The exclusive OR (XOR) problem has been discussed already in exercises.
- The patterns in the first class are (1,1) and (0,0).
- The patterns in the second class are (0,1) and (1,0).
- A single-layer perceptron is not sufficient for solving this problem.
- Reason: the classes are not linearly separable.
- However, the problem may be solved by adding a hidden layer.
- McCulloch-Pitts neuron model (a hard-limiting nonlinearity) is used here.
- In Haykin's book, the XOR problem and its solution are presented in detail.
- The weight vectors given in the book differ somewhat from those in our exercise solution.

- Both the solutions are correct; recall that the weight vectors found by perceptron are not unique.
- This is especially true in this kind of problem where there are only four widely separated training examples.
- You may read Section 4.5 in the book for understanding the solution thoroughly.

## 4.6 Heuristics for Making Back-Propagation Perform Better

- Design of a MLP network using back-propagation learning is partly art, not science.
- Numerous heuristic methods have been proposed for improving the learning speed and performance of back-propagation.
- Some good heuristic methods are discussed below.

## 1. Sequential versus batch update

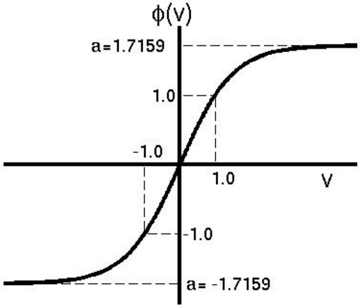
- Sequential learning mode is computationally faster than the batch mode.
- This is especially true when the training data set is large and highly redundant.

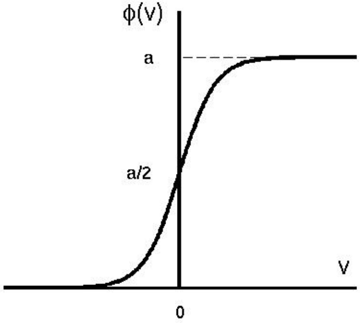
## 2. Maximizing information content

- Every training example should be chosen so that it contains as much as possible useful information for the learning task.
- Two ways of achieving this aim are:
  - Using an example that results in the largest training error.
  - Using an example that is radically different from the previously used ones.
- The training examples should be presented in randomized order in different epochs.
- A more refined technique is to emphasize difficult patterns in learning.
- However, this has problems also:
  - Distribution of the training data is distorted.
  - Outliers may have a catastrophic effect on performance.

## 3. Activation function

- An MLP network trained with backpropagation typically learns faster if an antisymmetric sigmoid function is used.
- An activation function $\varphi(v)$ is *antisymmetric (odd)* if $\varphi(-v) = -\varphi(v)$.
- The standard logistic function $a/[1 + \exp(-bv)]$ is nonsymmetric, but $\tanh(bv)$ is antisymmetric.





- A good choice for an activation function:

$$\varphi(v) = a\tanh(bv)$$

  where $a = 1.7159$ and $b = 2/3$.

- Then $\varphi(1) = 1$, $\varphi(-1) = -1$, and the first and second derivatives of $\varphi(v)$ have suitable values.

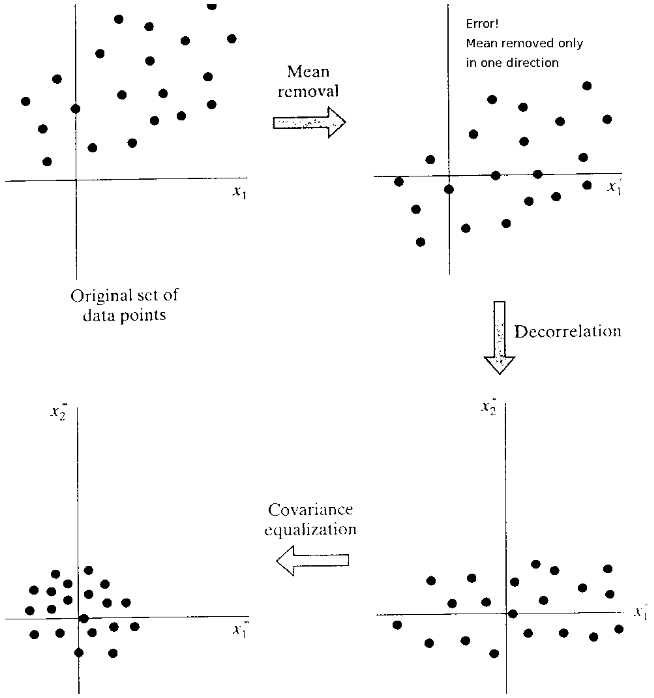
## 4. Target values

- The target values (desired responses) should be chosen within the range of the sigmoid activation function.
- The desired responses should be somewhat smaller than the extremal (saturation) values of the activation function.
- Otherwise, the back-propagation algorithm tends to drive the free parameters of the networks to infinity.
- This slows down the learning process by driving the hidden neurons into saturation.
- For example, for the activation function $\varphi(v) = 1.716 \tanh(0.667v)$ discussed before, convenient target values are $d_j = \pm 1$

## 5. Normalizing the inputs

- For speeding up back-propagation learning, the input vectors (variables) should be *preprocessed*.
- Recommended preprocessing steps for the training patterns:
  1. The mean value of the training vectors should be made zero (or small enough).
     - prevents slow, zigzagging type learning.
  2. The input variables (different components of training vectors) should be uncorrelated.
     - Can be realized using Principal Components Analysis (Chapter 8).
     - Removes second-order statistical redundancies.

3. The decorrelated input variables should be scaled to have approximately the same variances.
   - Ensures that different synaptic weights learn with roughly the same speed.


Original set of data points
$x_1$
Mean removal
Error!
Mean removed only
in one direction
$x_1'$
Decorrelation
$x_2''$
$x_1''$
Covariance equalization
$x_2'''$
$x_1'''$

## 6. Initialization

- Good initial values for the synaptic weights and thresholds (biases) of the network can help tremendously in designing a good network.
- Assume first that the synaptic weights have large initial values.
- Then it is likely that the neurons will be driven into saturation.
- Results in slow learning.
- Assume now that synaptic weights are assigned small initial values.
- Then the back-propagation algorithm may operate on a very flat area around the origin of the error surface.
- Unfortunately, this is a *saddle point*.
- There the gradient of the error surface is zero, but the saddle point is not a maximum nor minimum point.

- The proper choice of initialization lies somewhere between these two extreme cases.
- Assume now that:
  - The input variables have zero mean and unit variance.
  - They are mutually uncorrelated.
  - The tanh nonlinearity is used.
  - The thresholds (biases) are set to zero for all neurons.
  - The initial values of the synaptic weights are drawn from a uniform distribution with zero mean and the same variance $\sigma_w^2$.
- It is then fairly easy to show (see Haykin, pp. 183-184) that:
- For the activation function $\varphi(v) = 1.716 \tanh(0.667v)$ discussed earlier, we should choose $\sigma_w^2 = m^{-1}$.
- Here $m$ is the number of synaptic connections of a neuron.

## 7. Learning from hints

- The training examples are used for learning an approximation of an unknown input-output mapping $f(.)$.
- This may be generalized to include *learning from hints*.
- There possible prior information about the function $f(.)$ is utilized in learning.
- For example invariances, symmetries etc. may be used.
- Such prior information accelerates learning speed and improves the quality of the final estimate.

## 8. Learning rates

- Ideally, all the neurons in a MLP network should learn with the same rate.
- In practice, the last layers should typically use a smaller learning-rate parameter $\eta$.
- Reason: their local gradients tend to be larger.
- For a given neuron, the learning rate $\eta_j$ can be chosen inversely proportional to the square root of $m$.
- Again, $m$ is the number of synaptic connections of that neuron.

## 4.7 Output Representation and Decision Rule

- In theory, we need $M$ outputs for an $M$-class classification problem to represent all possible classification decisions.

- Let $\mathbf{x}_j$ denote the $j$th $m$-dimensional prototype to be classified by a multilayer perceptron (MLP) network.

- Let us denote by $\mathcal{C}_k$ the $k$th class.

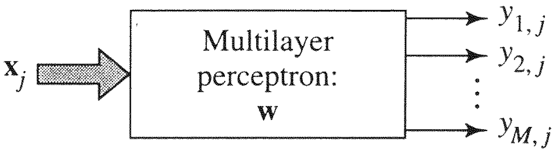
- Denote the $k$th output of the network by

$$y_{k,j} = F_k(\mathbf{x}_j), \;\; k = 1, \ldots, M$$

  corresponding to the prototype $\mathbf{x}_j$.

- The function $F_k(.)$ is the corresponding input-output mapping learned by the network.

$$y_{k,j} = F_k(\mathbf{x}_j), \qquad k = 1, 2, \ldots, M$$



Block diagram of a pattern classifier

- We can present these $M$ mappings conveniently in vector form

$$\mathbf{y}_j = \mathbf{F}(\mathbf{x}_j)$$

where

$$\mathbf{y}_j = [y_{1,j}, y_{2,j}, \ldots, y_{M,j}]^T,$$
$$\mathbf{F}(\mathbf{x}_j) = [F_1(\mathbf{x}_j), F_2(\mathbf{x}_j), \ldots, F_M(\mathbf{x}_j)]^T.$$

- Basic question: *what should be the optimum decision rule for classifying the $M$ outputs of a MLP network after training?*

- The continuous vector-valued function $\mathbf{y} = \mathbf{F}(\mathbf{x})$ minimizes the *empirical risk functional*

$$R = \frac{1}{2N} \sum_{J=1}^{N} \| \mathbf{d}_j - \mathbf{F}(\mathbf{x}_j) \|^2$$

- Here $\mathbf{d}_j$ is again the desired (target) output pattern for the prototype $\mathbf{x}_j$.

- $N$ is the total number of training vectors (prototypes).

- The risk $R$ is in essence similar to the average squared error $\mathcal{E}_{av}$.

- $\mathcal{E}_{av}$ was used as a cost function in deriving the back-propagation algorithm in Section 4.3.

- Typically, binary target values are used:

$$\begin{aligned} d_{kj} &= 1 \text{ when } \mathbf{x}_j \text{ belongs to class } \mathcal{C}_k, \\ d_{kj} &= 0 \text{ when } \mathbf{x}_j \text{ does not belong to class } \mathcal{C}_k. \end{aligned}$$

- Thus the class $\mathcal{C}_k$ is represented by the $M$-dimensional target vector

$$[0, \dots, 0, 1, 0, \dots, 0]^T$$

- This is the $k$th unit vector; only the $k$th element 1 is nonzero.

- In Haykin's book (pp. 185-186), justifications are given showing that a MLP classifier approximates the *a posteriori* class probabilities.

- A posteriori probability for the class $\mathcal{C}_j$ is the probability that a vector $\mathbf{x}$ with an unknown class actually belongs to the $j$th class.

- Prerequisites for this result:
  - The logistic nonlinearity is used.
  - The size of the training set is large enough.
  - Back-propagation learning does not get stuck at a local minimum.

- Hence an appropriate decision rule is the (approximate) Bayes rule generated by the a posteriori class probability estimates:

- *Classify* $\mathbf{x}$ *to the class* $\mathcal{C}_k$ *if*

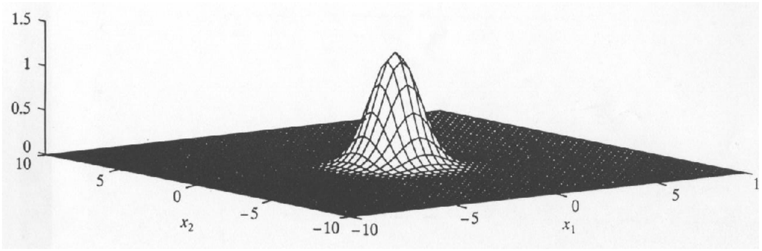
$$F_k(\mathbf{x}) > F_j(\mathbf{x}) \text{ for all } j \neq k$$

- If the underlying posterior class distributions are distinct, a unique largest output value exists with probability 1.

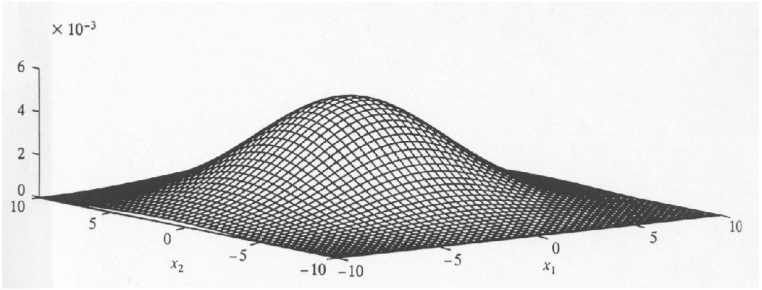
- Some less important comments on the derived approximate Bayes rule have been presented at the end of the section 4.7.

## 4.8 Computer Experiment

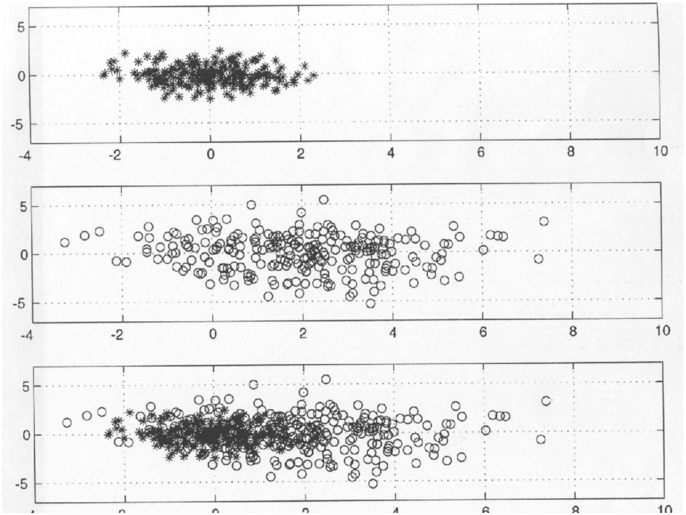
- A computer experiment is discussed thoroughly in this section.
- The example illustrates the learning behavior and performance of a multilayer perceptron in a simple pattern classification problem.
- Two overlapping, two-dimensional, Gaussian distributed pattern classes.
- The classes have different means and spherical covariance matrices.



Probability density function of class 1.



Probability density function of class 2.


Scatterplots of (a) class 1 and (b) class 2. (c) Combined scatterplot.

- Assume that:
  - the two classes are equiprobable.
  - the costs of correct classifications are zero.
  - the costs of misclassifications are equal.
- After straightforward calculations, it turns out that the optimal Bayes decision boundary is a circle.
- The centre of the circle is at $[-2/3, 0]^T$ and its radius is approximately $r = 2.34$.
- The vectors $\mathbf{x}$ falling inside the circle are classified to the first class $\mathcal{C}_1$, otherwise to the second class $\mathcal{C}_2$.
- See Haykin pp. 188-191 for a more detailed derivation.
- Furthermore, one can numerically evaluate for the probability of correct classification $P_c$ and misclassification $P_e$

$$P_c = 0.8151, \qquad P_e = 0.1849$$

## Experimental Determination of Optimal Multilayer Perceptron

- Parameters of multilayer perceptron

| Parameter | Symbol | Typical Range |
|---|---|---|
| Number of hidden neurons | $m_1$ | $(2, \infty)$ |
| Learning-rate parameter | $\eta$ | $(0, 1)$ |
| Momentum constant | $\alpha$ | $(0, 1)$ |

- First, the optimal number of hidden neurons is studied

- The smallest number of hidden neurons that yields a performance sufficiently close the Bayes classifier is chosen

- Simulation results for 2 hidden neurons ($\eta = 0.1, \alpha = 0$)

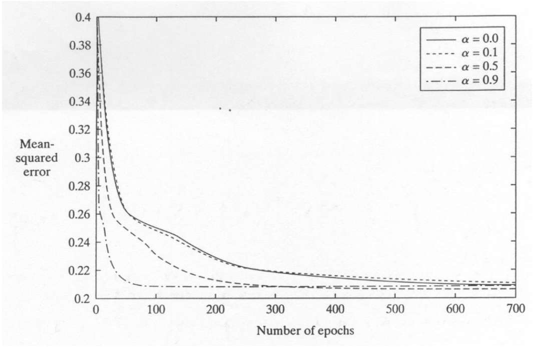
| Run No | Training Set Size | Number of Epochs | MSE | Probability of Correct Classification, $P_c$ |
|---|---|---|---|---|
| 1 | 500 | 320 | 0.2375 | 80.36% |
| 2 | 2000 | 80 | 0.2341 | 80.33% |
| 3 | 8000 | 20 | 0.2244 | 80.47% |

- Simulation results for 4 hidden neurons ($\eta = 0.1, \alpha = 0$)

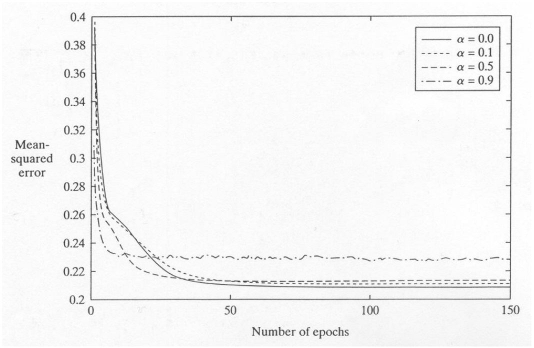
| Run No | Training Set Size | Number of Epochs | MSE | Probability of Correct Classification, $P_c$ |
|---|---|---|---|---|
| 1 | 500 | 320 | 0.2199 | 80.80% |
| 2 | 2000 | 80 | 0.2108 | 80.81% |
| 3 | 8000 | 20 | 0.2142 | 80.19% |

- It turns out that 2 hidden neurons perform equally well as 4
$\Rightarrow$ 2 hidden neurons are used

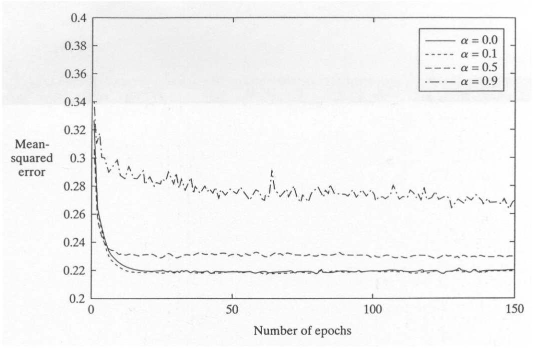
- Then the learning parameter $\eta$ and momentum parameter $\alpha$ are studied
  - $\eta = 0.01$, $0.1$, $0.5$ or $0.9$ and $\alpha = 0.0$, $0.1$, $0.5$ or $0.9$



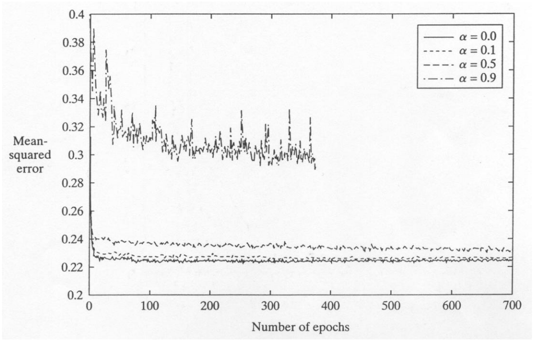
- $\eta = 0.01$



- $\eta = 0.1$

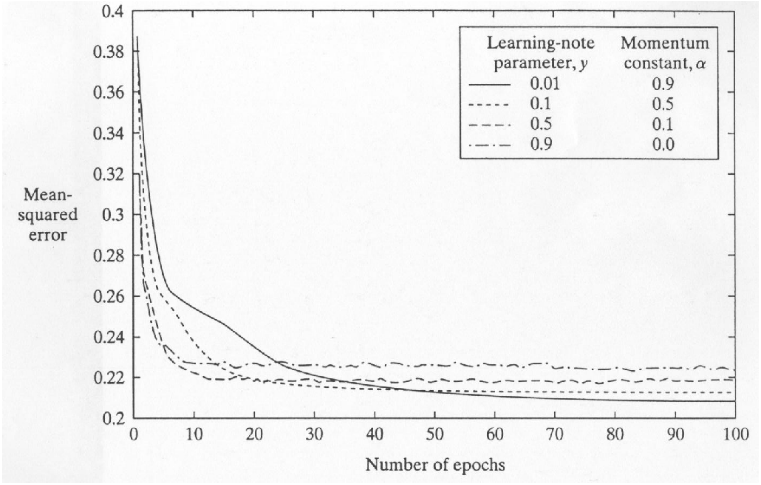
- $\eta = 0.5$ 

Mean-squared error
Number of epochs
$\alpha = 0.0$
$\alpha = 0.1$
$\alpha = 0.5$
$\alpha = 0.9$

- $\eta = 0.9$ 

Mean-squared error
Number of epochs
$\alpha = 0.0$
$\alpha = 0.1$
$\alpha = 0.5$
$\alpha = 0.9$

- From each subfigure, the best learning curve is selected



- the optimal values are chosen to be $\eta_{opt} = 0.1$, $\alpha_{opt} = 0.5$

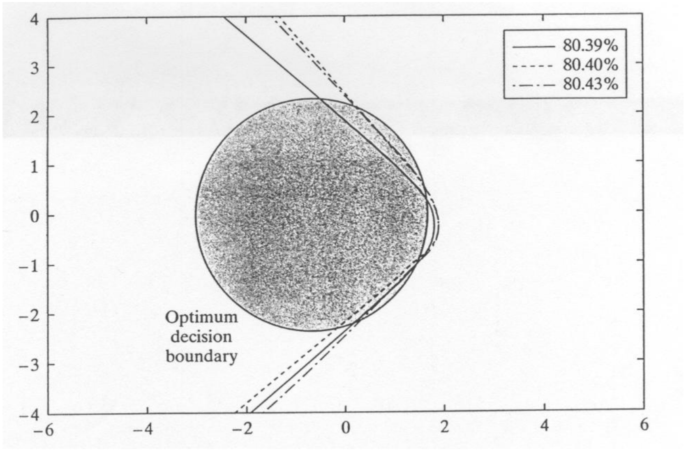
- Configuration of Optimized MLP

| Parameter | Symbol | Value |
|---|---|---|
| Optimum number of hidden neurons | $m_{opt}$ | 2 |
| Optimum learning-rate parameter | $\eta_{opt}$ | 0.1 |
| Optimum momentum constant | $\alpha_{opt}$ | 0.5 |

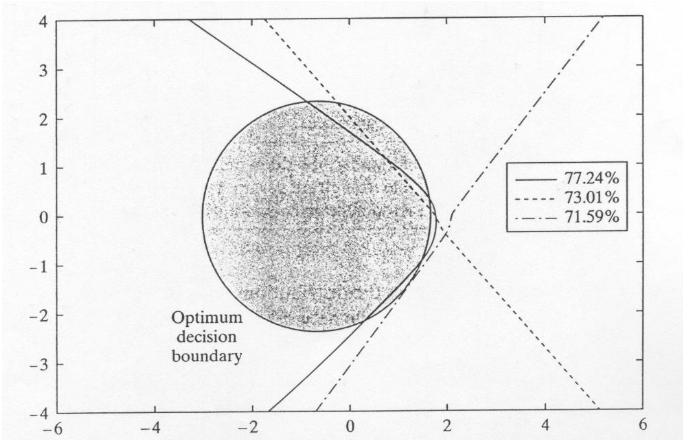
- For these values and $m = 2$ hidden neurons, 20 MLPs are trained independently to evaluate the performance

- In each of the 20 training sets, 1000 samples are chosen randomly for learning.

- The test set contains 32.000 samples



- The classification boundaries of the 3 best MLP networks



- Similarly, the decision boundaries of the 3 poorest MLP networks
- The average performance of the 20 learned MLP networks is

| Performance Measure | Mean | Standard Deviation |
|---|---|---|
| Probability of correct classification | 79.70% | 0.44% |
| Final mean-square error | 0.2277 | 0.0118 |