PART II:
COMPUTER EXPERIMENTS

Contents

List of Figures

NOTES FOR THE TEACHER

The solutions to the end-of-chapter computer experiments are all included in Part II of the Solutions Manual.

The majority of MATLAB codes used to perform the computer experiments are listed separately in the same Prentice Hall website where the Solutions Manual is posted.

Chapter 1: Rosenblatt's Perceptron

Problem 1.6

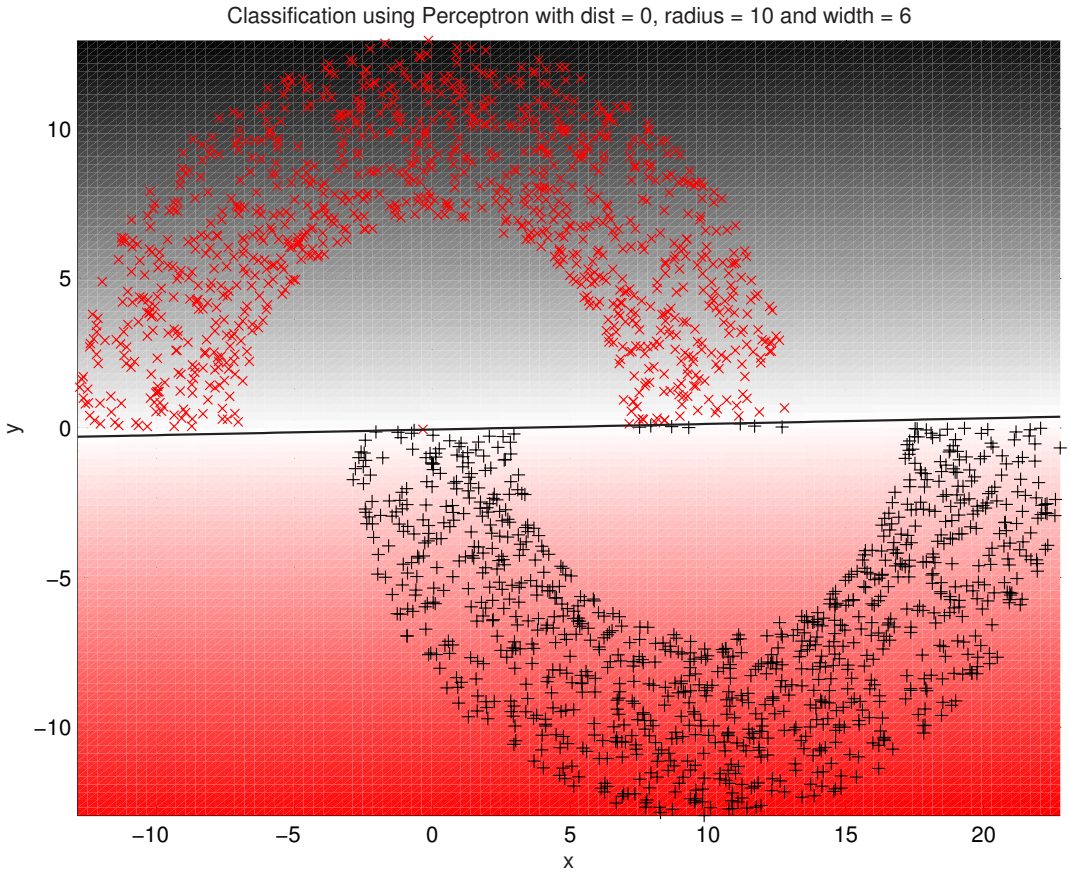


Figure 1 (Problem 1.6): Perceptron with distance $d = 0$, Error points : $8(0.40\%)$

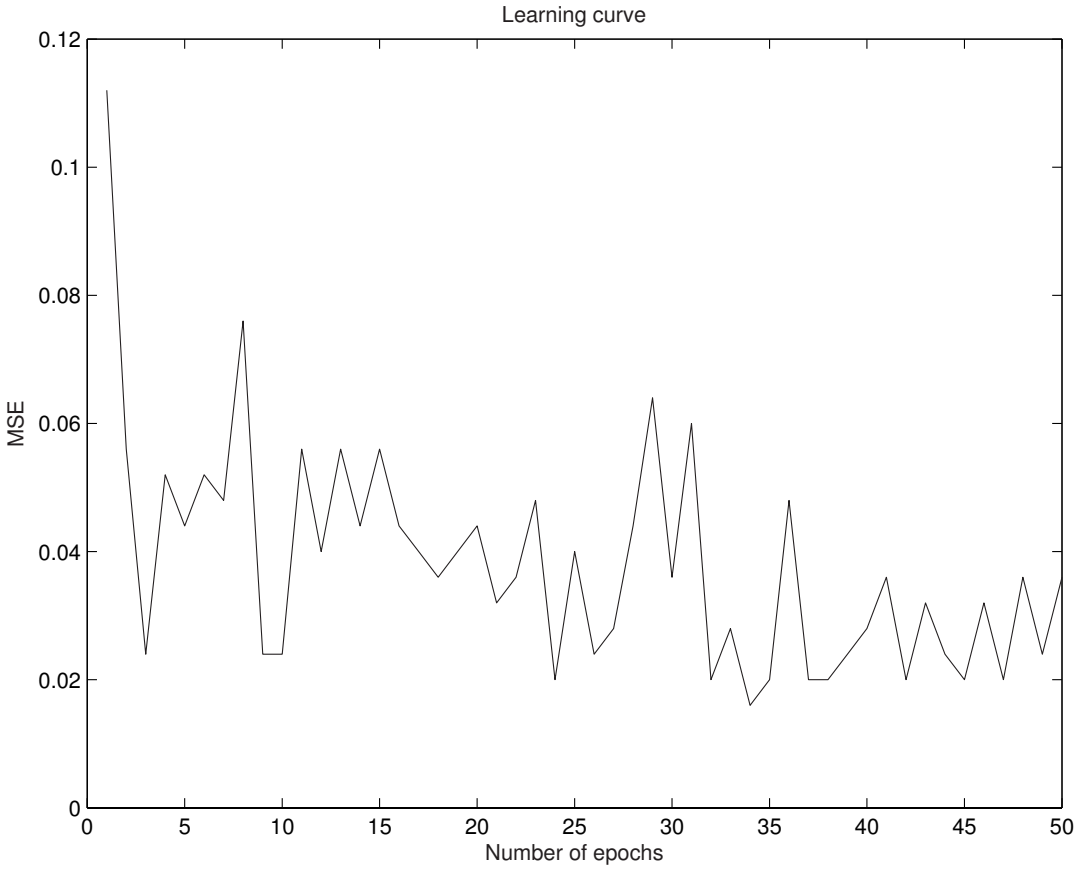


Figure 2 (Problem 1.6): Perceptron with distance $d = 0$, Learning curve

Chapter 2: Model Building through Regression

Problem 2.8

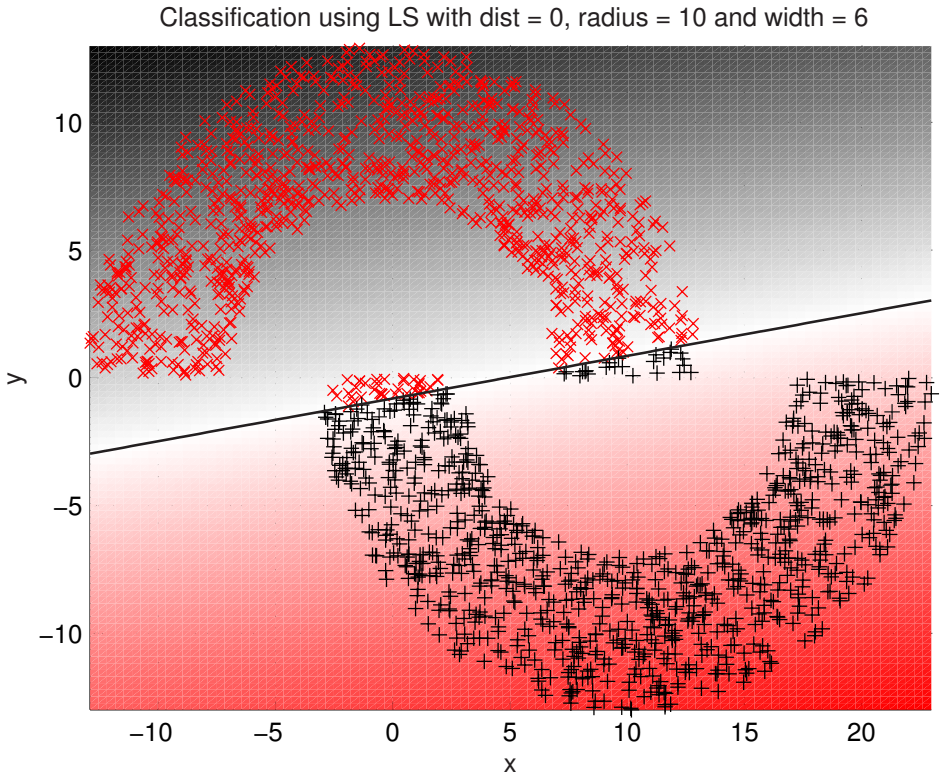


Figure 1 (Problem 2.8): Least-squares classification with distance $d = 0$. Error points : 62 (3.10%)

Summarizing remarks:

- On the verge of separability, the perceptron outperforms the least-squares classifier.

Problem 2.9

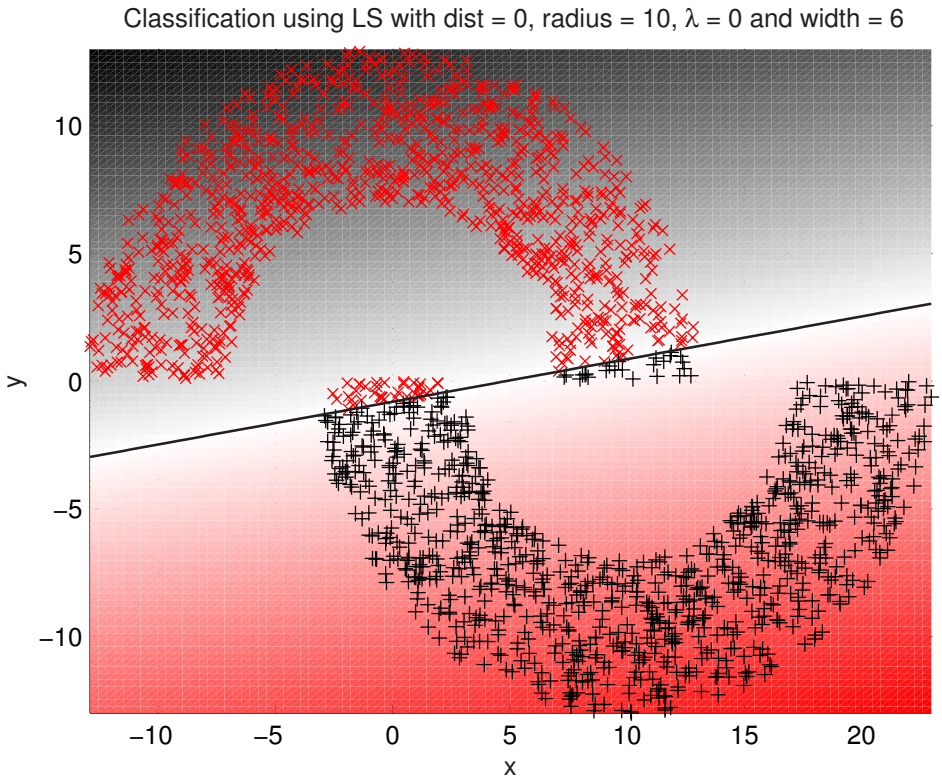


Figure 1 (Problem 2.9): Least-squares classification with distance $d = 0$ and $\lambda = 0$. Error points : 62 (3.10%)

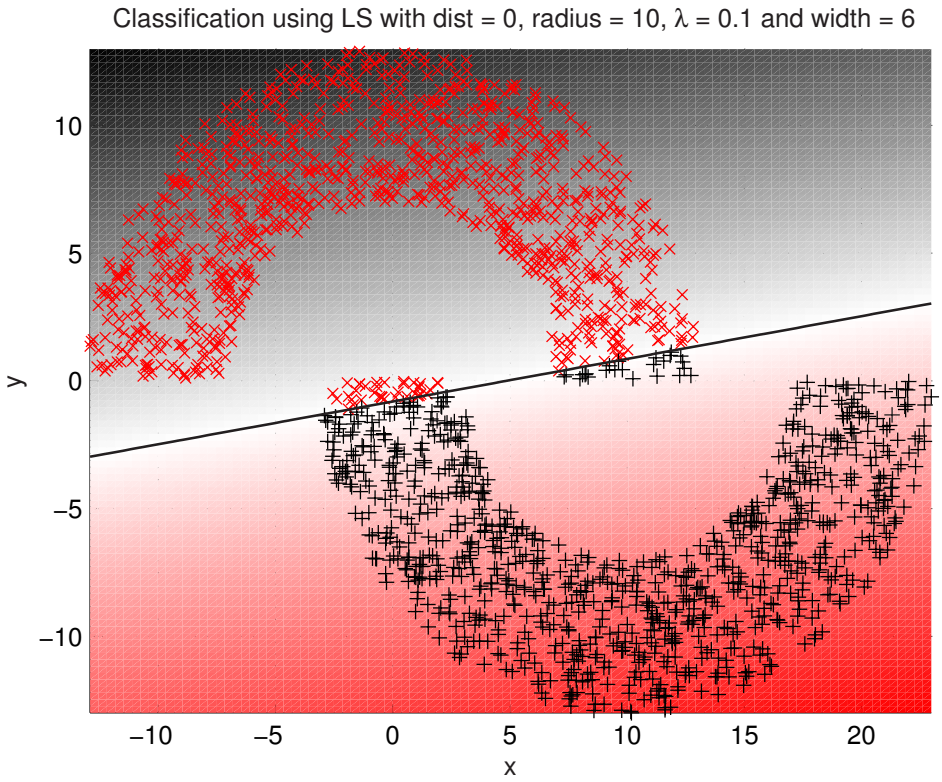


Figure 2 (Problem 2.9): Least-squares classification with distance $d = 0$ and $\lambda = 0.1$. Error points : 62 (3.10%)

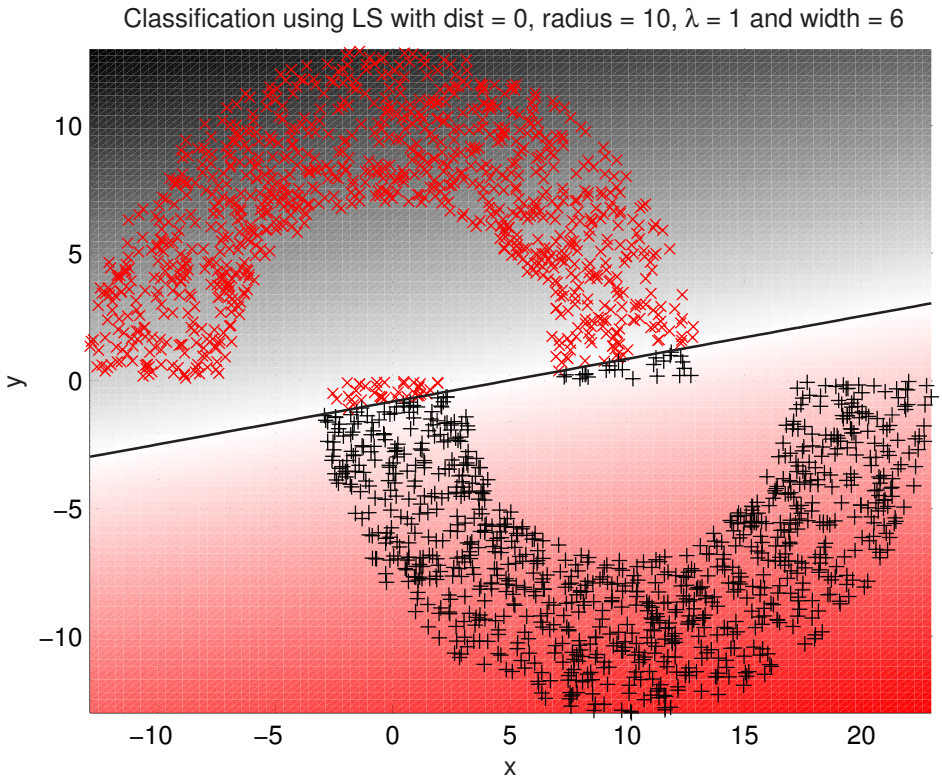


Figure 3 (Problem 2.9): Least-squares classification with distance $d = 0$ and $\lambda = 1$. Error points : 62 (3.10%)

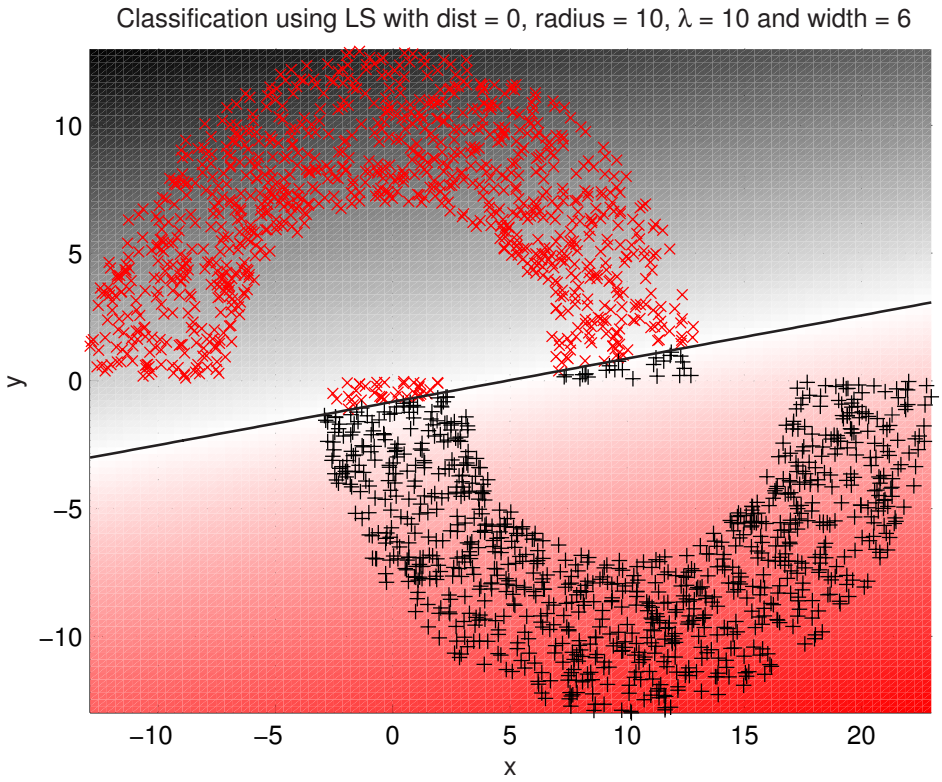


Figure 4 (Problem 2.9): Least-squares classification with distance $d = 0$ and $\lambda = 10$. Error points : 62 (3.10%)

Summarizing results:

- Regularization makes no practical difference in the performance of the method of least squares.

Chapter 3: The Least-Mean-Square Algorithm

Problem 3.10

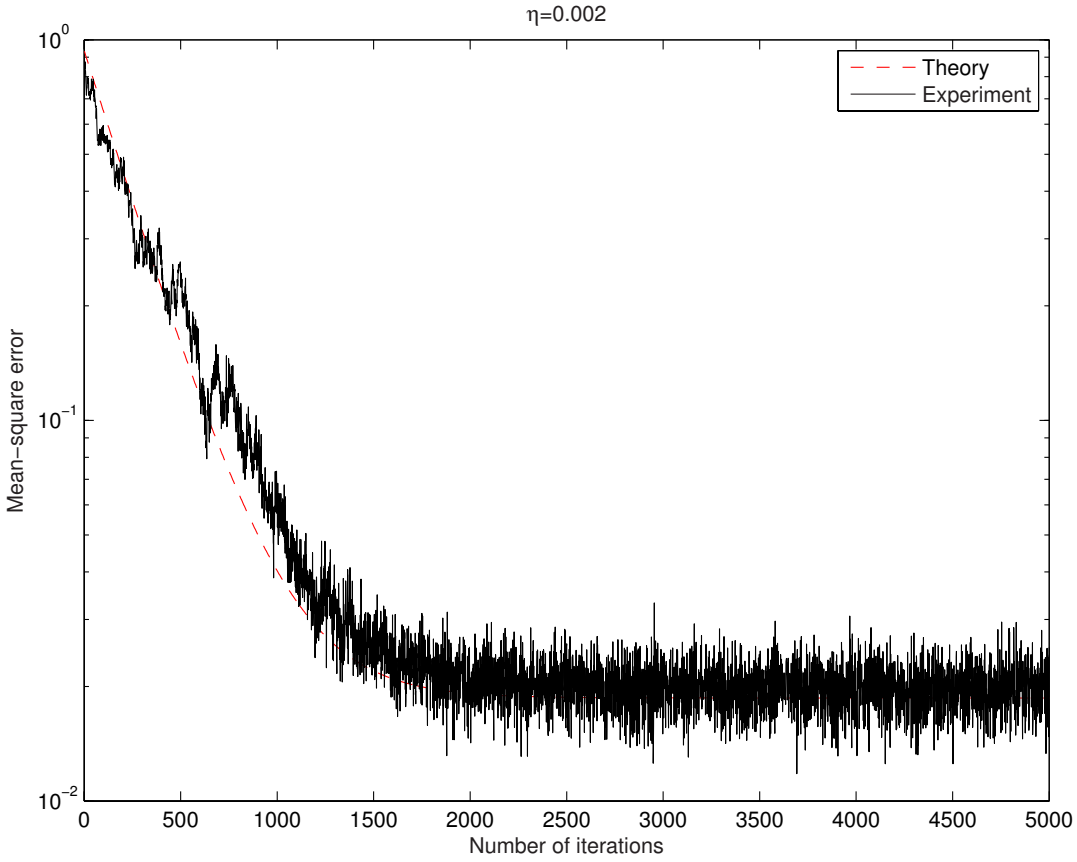


Figure 1 (Problem 3.10): LMS for AR process $\eta = 0.002$

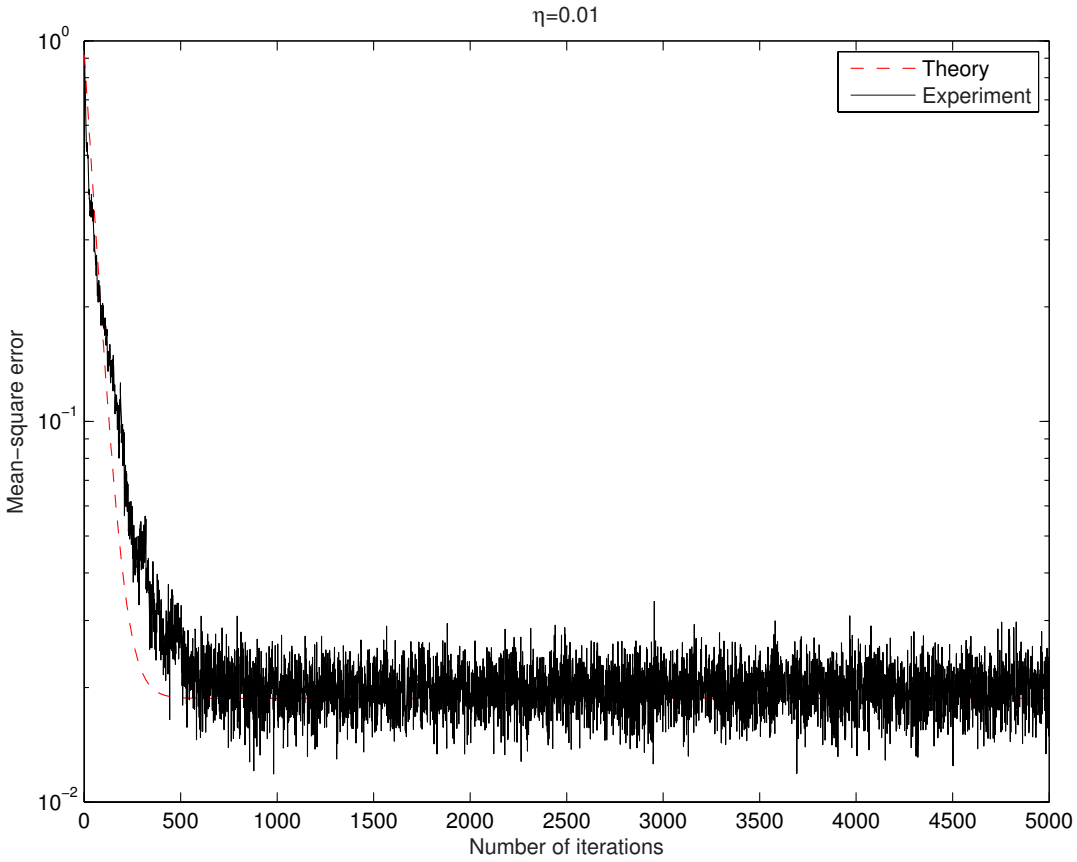


Figure 2 (Problem 3.10): LMS for AR process $\eta = 0.01$

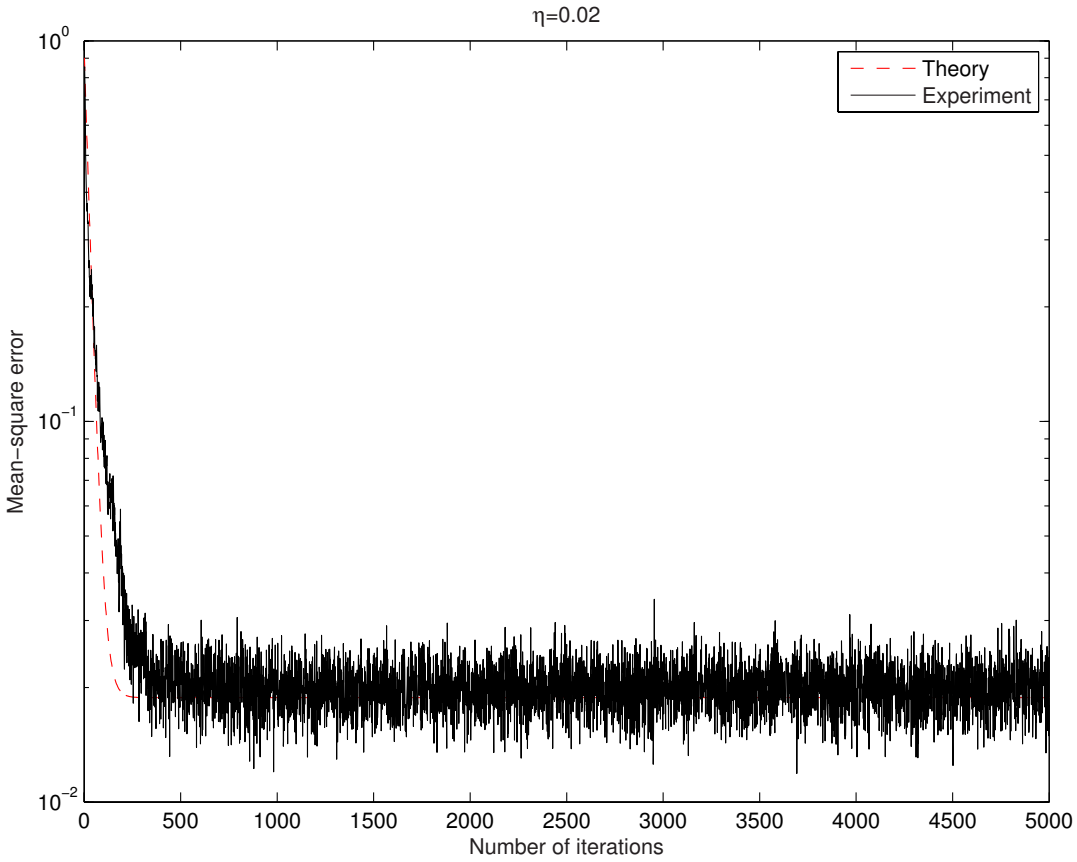


Figure 3 (Problem 3.10): LMS for AR process $\eta = 0.02$

Table 1: Summary of Results

Learning rate η #	Number of time-steps for convergence (approx.)
0.002	1200
0.01	450
0.02	300

Problem 3.11

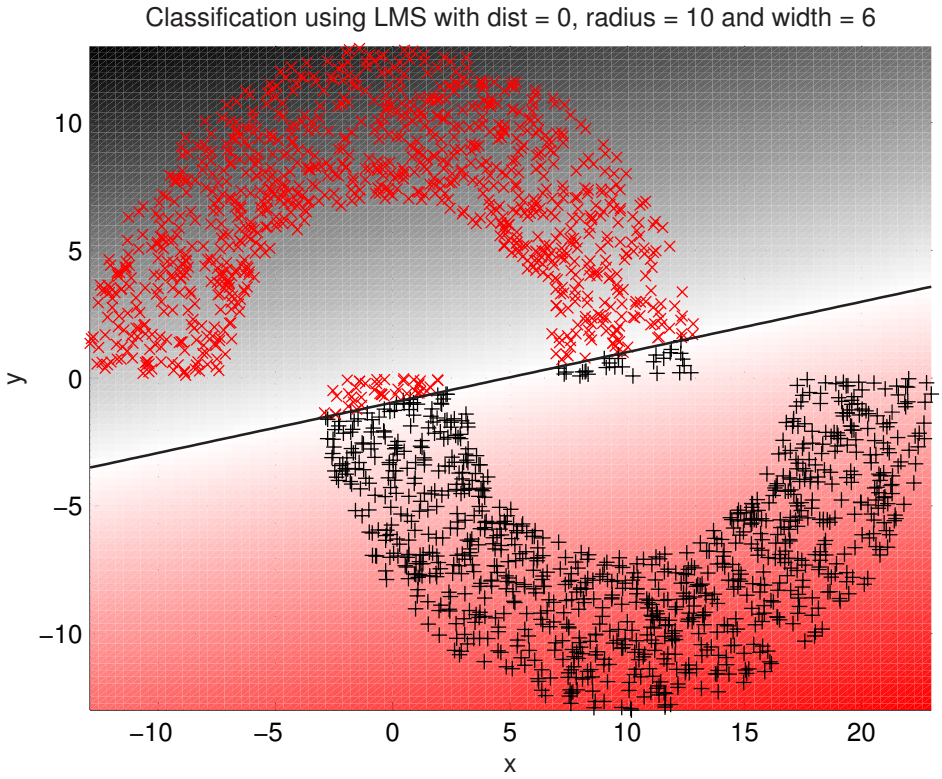


Figure 1 (Problem 3.11): Classification using LMS for $d = 0$, Error points : 71 (3.55%)

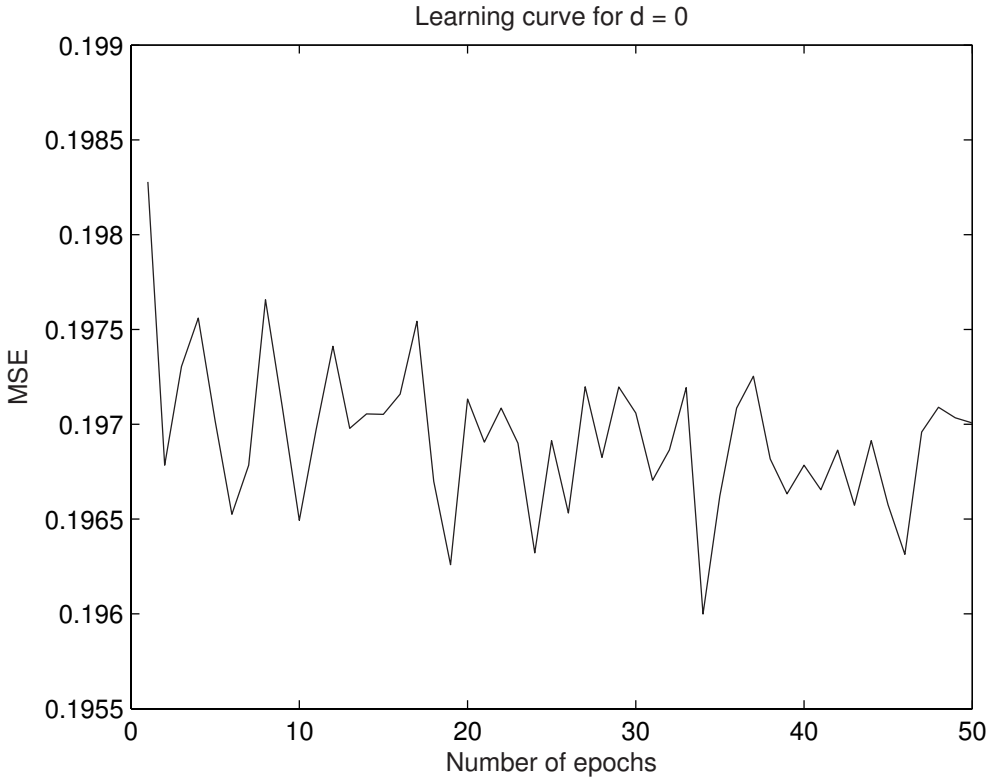


Figure 2 (Problem 3.11): Classification using LMS for $d = 0$ learning curve

Summary of results:

- Classification error for perceptron: 0.4%
- Classification error for LMS: 3.55%

Problem 3.12

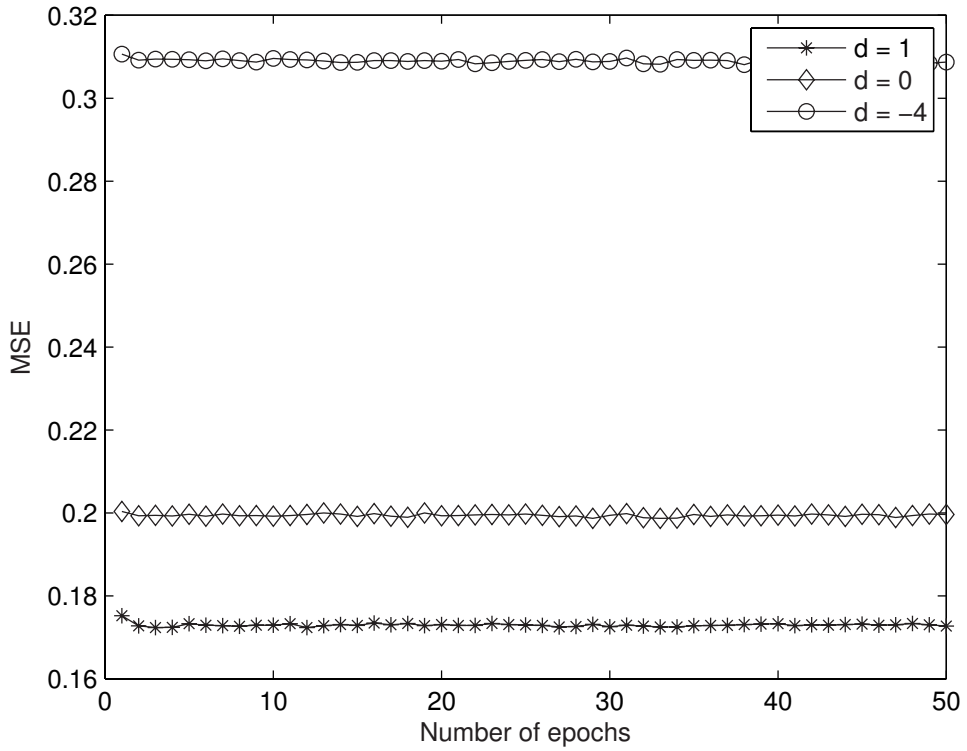


Figure 1 (Problem 3.12): MSE for Classification using LMS for $d = 1, 0, -4$

Summary of results:

- On the verge of linear separability ($d = 0$), the mean-square error (approx.):
  - LMS: 0.2
  - Perceptron: 0.02

Chapter 4: Multilayer Perceptrons

Problem 4.15

In this problem, we explore the operation of a fully connected multilayer perceptron trained with the back-propagation algorithm for the approximation of different algebraic functions.

(a) $f(x) = 1/x$ for $1 \leq x \leq 100$

The network is trained with:

learning-rate parameter $\eta = 0.3$, and

momentum constant $\alpha = 0.7$.

Ten different network configurations were trained to learn this mapping. Each network was trained identically, that is, with the same η and α, with bias terms, and with 10,000 passes of the training vectors (with one exception noted below). Once each network was trained, the test dataset was applied to compare the performance and accuracy of each configuration. Table 1 summarizes the results obtained:

Table 1

Number of hidden neurons	Average percentage error at the network output
3	4.73%
4	4.43
5	3.59
7	1.49
10	1.12
15	0.93
20	0.85
30	0.94
100	0.9
30 (trained with 100,000 passes)	0.19

The results of Table 1 indicate that even with a small number of hidden neurons, and with a relatively small number of training passes, the network is able to learn the mapping described in (a) quite well.

(b) $f(x) = \log_{10} x$ for $1 \leq x \leq 10$

The results of this second experiment are presented in Table 2:

Table 2

Number of hidden neurons	Average percentage error at the network output
2	2.55%
3	2.09
4	0.46
5	0.48
7	0.85
10	0.42
15	0.85
20	0.96
30	1.26
100	1.18
30 (trained with 100,000 passes)	0.41

Here again, we see that the network performs well even with a small number of hidden neurons. Interestingly, in this second experiment the network peaked in accuracy with 10 hidden neurons, after which the accuracy of the network to produce the correct output started to decrease.

(c) $f(x) = e^{-x}$ for $1 \leq x \leq 10$

The results of this third experiment (using the logistic function as with experiments (a) and (b)), are summarized in Table 3:

Table 3

Number of hidden neurons	Average percentage error at the network output
2	244.0‘%
3	185.17
4	134.85
5	133.67
7	141.65
10	158.77
15	151.91
20	144.79
30	137.35
100	98.09
30 (trained with 100,000 passes)	103.99

These results are unacceptable since the network is unable to generalize when each neuron is driven to its limits.

The experiment with 30 hidden neurons and 100,000 training passes was repeated, but this time the hyperbolic tangent function was used as the nonlinearity. The result obtained this time was an average percentage error of 3.87% at the network output. This last result shows that the hyperbolic tangent function is a better choice than the logistic function as the sigmoid function for realizing the mapping $f(x) = e^{-x}$.

(d) $f(x) = \sin x$ for $0 \leq x \leq \pi/2$

Finally, the following results were obtained using the logistic function with 10,000 training passes, except for the last configuration:

Table 4

Number of hidden neurons	Average percentage error at the network output
2	1.63‘%
3	1.25
4	1.18
5	1.11
7	1.07
10	1.01
15	1.01
20	0.72
30	1.21
100	3.19
30 (trained with 100,000 passes)	0.4

The results of Table 4 show that the accuracy of the network peaks around 20 neurons, where after the accuracy deteriorates.

Problem 4.16

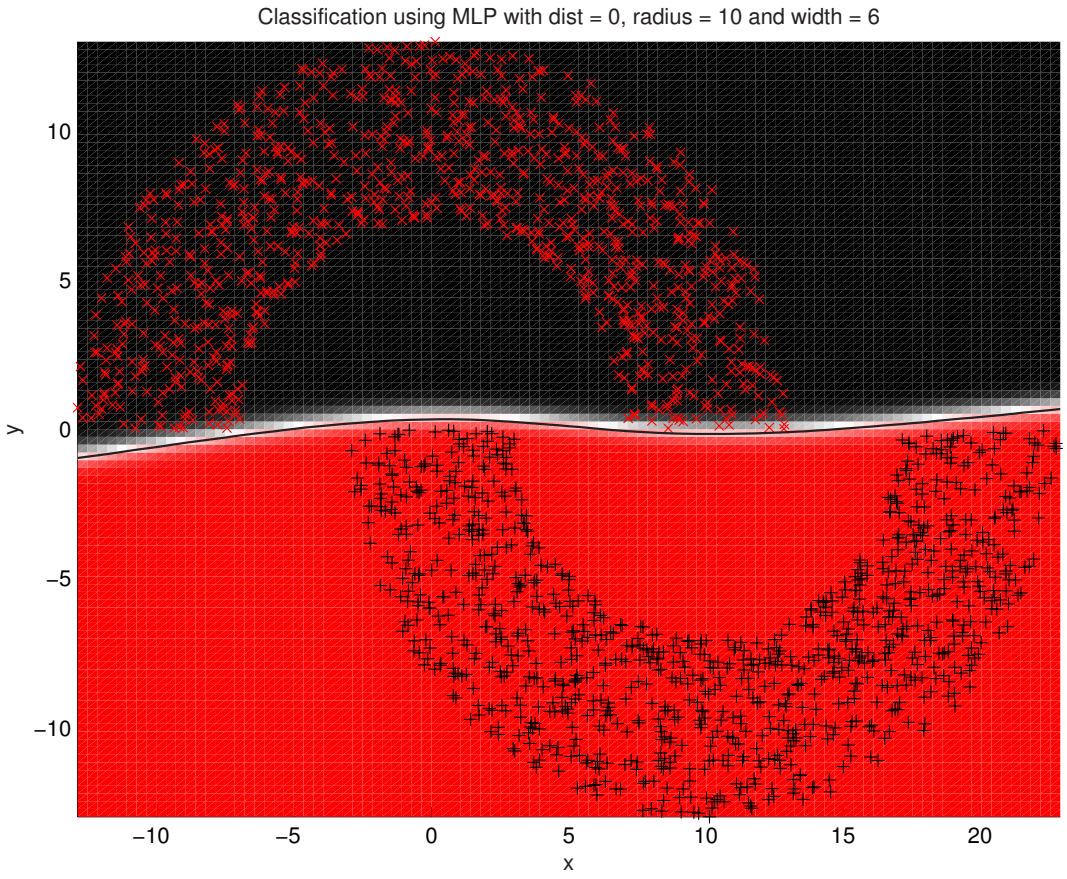


Figure 1 (Problem 4.16): Classification using MLP for $d = 0$, Error points : 0 (0.00%)

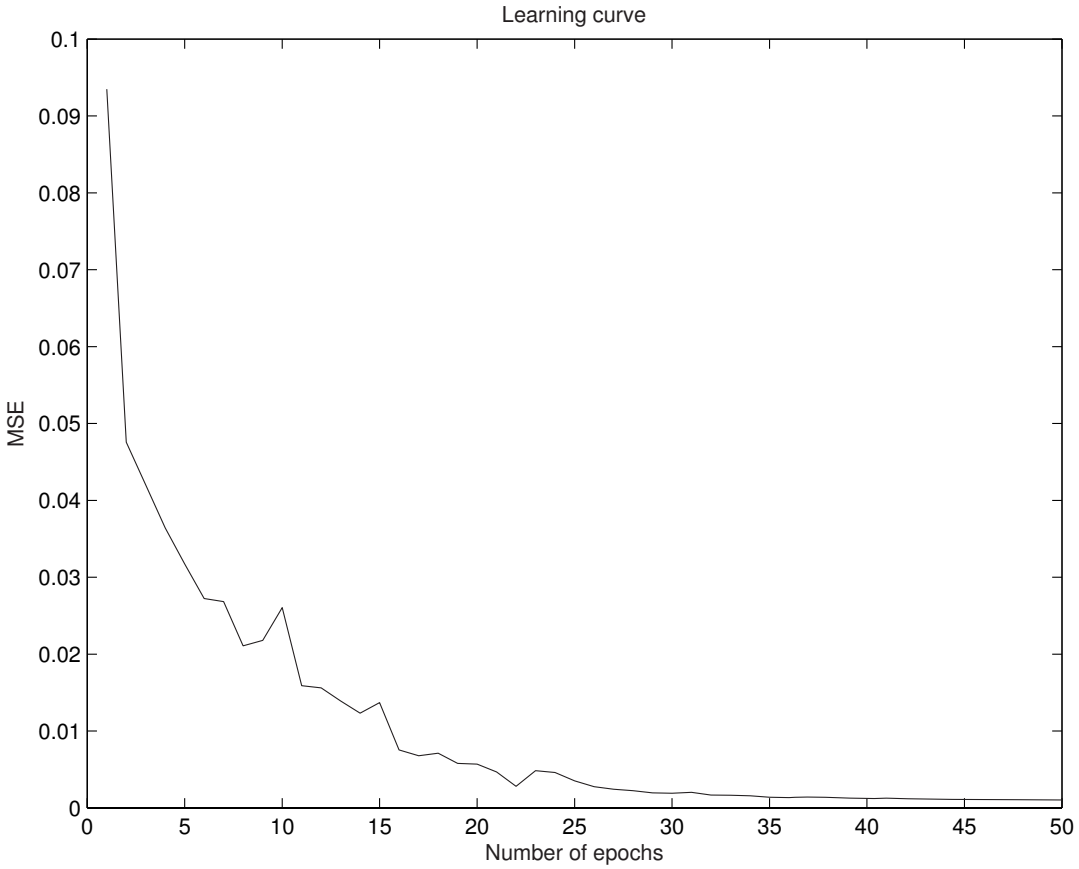


Figure 2 (Problem 4.16): Classification using MLP for $d = 0$: learning curve. Error points : 0 (0.00%)

Problem 4.17

The objective of this experiment is to distinguish between two classes of "overlapping", two-dimensional, Gaussian-distributed patterns labeled 1 and 2. Let C_1 and C_2 denote the set of events for which a random vector $\mathbf{x}$ belongs to patterns 1 and 2, respectively. We may then express the conditional probability density functions for the two classes as follows:

$$\text{Class } C_1: \quad f_{\mathbf{X}}(\mathbf{x}|C_1) = \frac{1}{2\pi\sigma_1^2}\exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\mu_1\|^2\right) \tag{1}$$

where

μ_1 = mean vector = $[0,0]^T$

σ_1^2 = variance = 1

$$\text{Class } C_2: \quad f_{\mathbf{X}}(\mathbf{x}|C_2) = \frac{1}{2\pi\sigma_2^2}\exp\left(-\frac{1}{2\sigma_2^2}\|\mathbf{x}-\mu_2\|^2\right) \tag{2}$$

where

μ_2 = $[2,0]^T$

σ_2^2 = 4

The two classes are assumed to be equiprobable; that is

$$p_1 = p_2 = \frac{1}{2}$$

Figure 1a shows three-dimensional plots of the two Gaussian distributions defined by (1) and (2). The input vector is $\mathbf{x} = [x_1, x_2]^T$, and the dimensionality of the input space is $m_0 = 2$. Figure 2 shows individual scatter diagrams for classes C_1 and C_2 and the joint scatter diagram representing the superposition of scatter plots of 500 points taken from each of the two processes. This latter diagram clearly shows that the two distributions overlap each other significantly, indicating that there is inevitably a significant probability of misclassification.

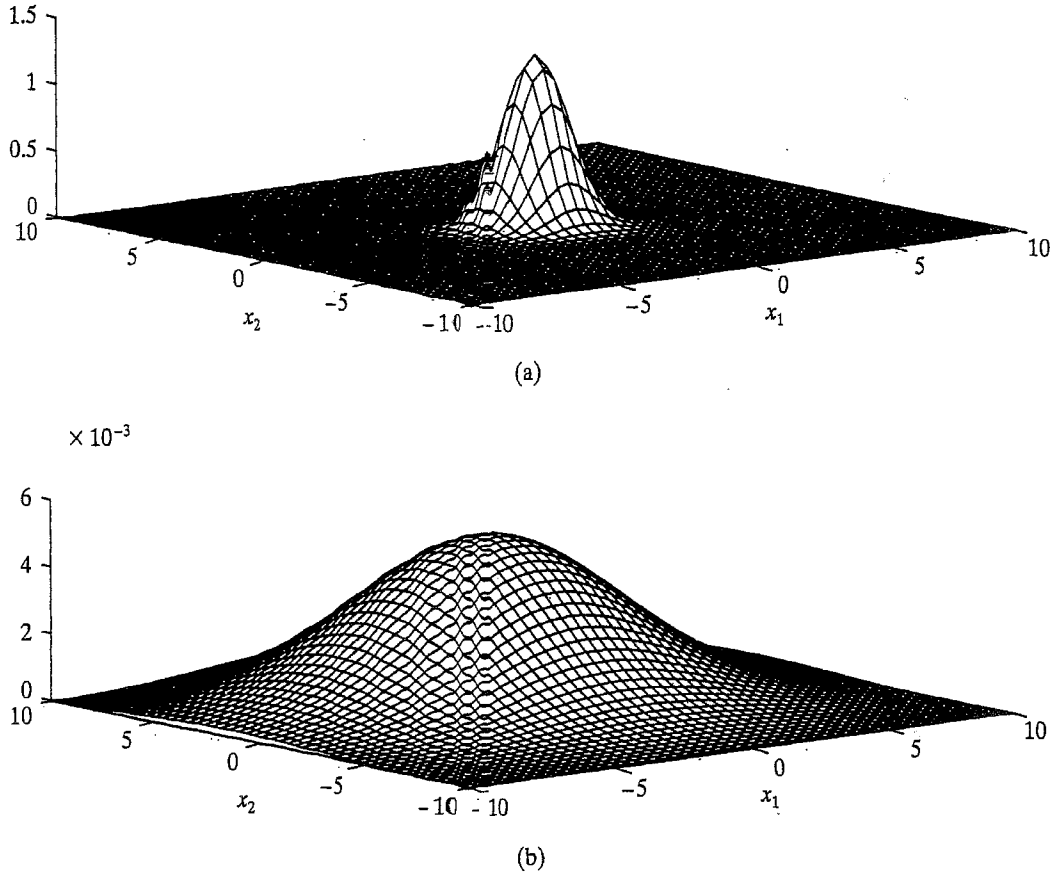


Figure 1: Problem 4.17
(a) Probability density function $p_{\mathbf{x}}(\mathbf{x}|C_1)$
(b) Probability density function $p_{\mathbf{x}}(\mathbf{x}|C_2)$

Bayesian Decision Boundary

The Bayes criterion for optimum classification is discussed in Chapter 2 of the text. Assuming that for a two-class problem, (1) classes C_1 and C_2 are equiprobable, (2) the costs for correct classifications are zero, and (3) the costs for misclassifications are equal, we find that the optimum decision boundary is found by applying the likelihood ratio test:

$$\Lambda(\mathbf{x}) \underset{C_1}{\overset{C_2}{\lessgtr}} \xi \tag{3}$$

where $\Lambda(\mathbf{x})$ is the *likelihood ratio*, defined by

$$\Lambda(\mathbf{x}) = \frac{p_{\mathbf{X}}(\mathbf{x}|C_1)}{p_{\mathbf{X}}(\mathbf{x}|C_2)} \tag{4}$$

and ξ is the *threshold of the test*, defined by

$$\xi = \frac{p_2}{p_1} = 1 \tag{5}$$

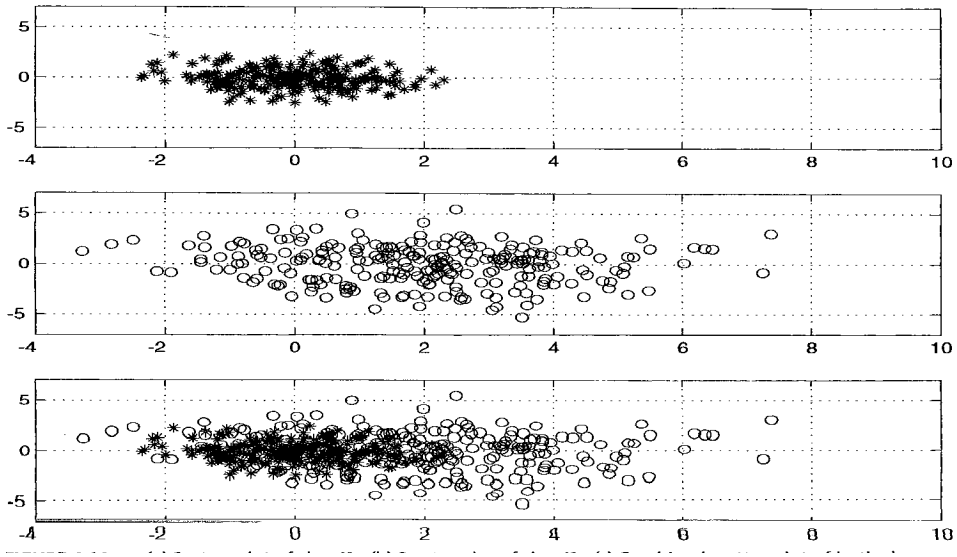


Figure 2: Problem 4.17
(a) Scatter plot of class C_1.
(b) Scatter plot of class C_2.
(c) Combined scatter plot of both classes.

For the example being considered, we have

$$\Lambda(\mathbf{x}) = \frac{\sigma_2^2}{\sigma_1^2}\exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\mu_1\|^2 + \frac{1}{2\sigma_2^2}\|\mathbf{x}-\mu_2\|^2\right)$$

The optimum (Bayesian) decision boundary is therefore defined by

$$\frac{\sigma_2^2}{\sigma_1^2}\exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\mu_1\|^2 + \frac{1}{2\sigma_2^2}\|\mathbf{x}-\mu_2\|^2\right) = 1$$

or equivalently,

$$\frac{1}{\sigma_2^2}\|\mathbf{x}-\mu_2\|^2-\frac{1}{\sigma_1^2}\|\mathbf{x}-\mu_2\|^2 \;=\; 4\log\left(\frac{\sigma_1}{\sigma_2}\right) \tag{6}$$

Using straightforward manipulations, we may redefine the optimum decision boundary of (6) simply as

$$\|\mathbf{x}-\mathbf{x}_c\|^2 \;=\; r^2 \tag{7}$$

where

$$\mathbf{x}_c \;=\; \frac{\sigma_2^2\mu_1-\sigma_1^2\mu_2}{\sigma_2^2-\sigma_1^2} \tag{8}$$

and

$$r^2 \;=\; \frac{\sigma_1^2\sigma_2^2}{\sigma_2^2-\sigma_1^2}\left[\frac{\mu_1-\mu_2}{\sigma_2^2-\sigma_1^2}+4\log\left(\frac{\sigma_2}{\sigma_1}\right)\right] \tag{9}$$

Equation (7) represents a circle with center $\mathbf{x}_c$ and radius r. Let Ω_1 define the region lying inside this circle. The Bayesian classification rule for the problem at hand may now be stated as follows:

> *Classify the observation vector* $\mathbf{x}$ *as belonging to class* C_1 *if the likelihood ratio* $\Lambda(\mathbf{x})$ *is greater than the threshold* ξ *and to class* C_2 *otherwise.*

For the particular parameters of this experiment, we have a circular decision boundary whose center is located at

$$\mathbf{x}_c \;=\; \begin{bmatrix} -2/3 \\ 0 \end{bmatrix}$$

and whose radius is

$$r \approx 2.34$$

Let c denote the set of classification outcomes, and e the set of erroneous classification outcomes. The *probability of error* (misclassification), P_e, of a classifier operating according to the Bayesian decision rule is

$$P_e \;=\; p_1P(e|C_1)+p_2P(e|C_2) \tag{10}$$

where $P(e|C_1)$ is the conditional probability of error given that the classifier input vector was drawn from the distribution of class C_1, and similarly for $P(e|C_2)$; p_1 and p_2 are the prior probabilities of classes C_1 and C_2, respectively. For our problem we may numerically evaluate the probability integrals to obtain

$$P(e|C_1) \approx 0.1056$$

and

$$P(e|C_2) \approx 0.2642$$

With $p_1 = p_2 = 1/2$, the probability of misclassification is therefore

$$P_e \approx 0.1849$$

Equivalently, the *probability of correct classification* is

$$\begin{aligned} P_c &= 1 - P_e \\ &\approx 0.8151 \end{aligned}$$

Note that P_e and P_c add up to unity, as they should.

Experimental Determination of Optimal Multilayer Perceptron

Table 1 lists the variable parameters of a multilayer perceptron (MLP) that involves a single layer of hidden neurons, and that is trained with the back-propagation algorithm operating in the sequential (on-line) mode. Since the ultimate objective of a pattern classifier is to achieve an acceptable rate of correct classification, this criterion is used to judge when the variable parameters of the MLP (used as a pattern classifier) are optimal.

Table 1: Variable Parameters of Multilayer Perceptron

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

Optimal Number of Hidden Neurons. Reflecting on practical approaches to the problem of determining the optimal number of hidden neurons, m_1, the criterion used is the smallest number of hidden neurons that yields a performance "close" to the Bayesian classifier--usually within 1 percent. Thus, the experimental study begins with two hidden neurons as the starting point for the simulation results summarized in Table 2. Since the purpose of the first set of simulations is

merely to ascertain the sufficiency of two hidden neurons or otherwise, the learning-rate parameter η and momentum constant α are arbitrarily set to some nominal values. For each simulation run, a training set of examples, randomly generated from the Gaussian distributions for classes C_1 and C_2 with equal probability, is repeatedly cycled through the network, with each training cycle representing an *epoch*. The number of epochs is chosen so that the total number of training examples used for each run is constant. By so doing, any potential effects arising from variations of the training set sizes are averaged out.

Table 2: Simulation Results for Two hidden Neurons[a]

Run Number	Training Set Size	Number of Epochs	Mean-Square Error	Probability of Correct Classification, P_e
1	500	320	0.2375	80.36%
2	2000	80	0.2341	80.33%
3	8000	20	0.2244	80.47%

[a]Learning rate parameter $\eta = 0.1$ and momentum $\alpha = 0$

In Table 2 and subsequent tables, the mean-square error is included in these tables only as a matter of record, since a *small mean-square error does not necessarily imply good generalization* (i.e., good performance with data not seen before).

After convergence of a network trained with a total number of N patterns, the probability of correct classification can in theory be calculated as follows:

$$P(c, N) = p_1 P(c, N|C_1) + p_2 P(c, N|C_2) \tag{11}$$

where $p_1 = p_2 = 1/2$, and

$$P(c, N|C_1) = \int_{\Omega_1(N)} p_{\mathbf{X}}(\mathbf{x}|C_1) d\mathbf{x} \tag{12}$$

$$P(c, N|C_2) = - \int_{\Omega_1(N)} p_{\mathbf{X}}(\mathbf{x}|C_2) d\mathbf{x} \tag{13}$$

and $\Omega_1(N)$ is the region in the decision space over which the multilayer perceptron (trained with N patterns) classifies the vector $\mathbf{x}$ (representing a realization of the random vector $\mathbf{X}$) as belonging to class C_1. This region is usually found experimentally by evaluating the mapping function learned by the network. Unfortunately, the numerical evaluation of $P(c,N|C_1)$ and $P(c,N|C_2)$ is problematic because closed-form expressions describing the decision boundary $\Omega_1(N)$ cannot easily be found.

Accordingly, we resort to the use of an experimental approach that involves testing the trained multilayer perceptron against another independent set of examples that are again drawn randomly from the distributions for classes C_1 and C_2 with equal probability. Let A be a random

variable that counts the number of patterns out of the N test patterns that are classified correctly. Then the ratio

$$p_N = \frac{A}{N}$$

is a random variable that provides the maximum-likelihood unbiased estimate of the actual classification performance p of the network. Assuming that p is constant over the N input-output pairs, we may apply the Chernoff bound (Duda, Hart, and Stork, 2001, p.47) to the estimator p_N of p, obtaining

$$P(p_N = p|\varepsilon > \exp(-2\varepsilon^2 N) = \delta$$

Application of the Chernoff bound yields $N \approx 26,500$ for $\varepsilon = 0.01$, and $\delta = 0.01$ (i.e., 99 percent certainty that the estimate p has the given tolerance). We thus picked a test set of size N = 32,000. The last column of Table 2 presents the probability of correct classification estimated for this test set size, with each result being the average of 10 independent trials of the experiment.

The classification performance presented in Table 2 for a multilayer perceptron using two hidden neurons is already reasonably close to the Bayesian performance P_c = 81.51 percent. On this basis we may conclude that for the pattern-classification problem described here the use of two hidden neurons is adequate. To emphasize this conclusion, in Table 3 we present the results of simulations repeated for the case of four hidden neurons, with all other parameters held constant. Although the mean-square error in Table 3 for four hidden neurons is slightly lower than that in Table 2 for two hidden neurons, the average rate of correct classification does not show improvement; in fact, it is slightly worse. For the remainder of the computer experiment described here, the number of hidden neurons is held at two.

Optimal Learning and Momentum Constants. For the "optimal" values of the learning-rate parameter η and momentum constant α, we may use any one of three definitions:

1. The η and α that, on average, yield convergence to a local minimum in the error surface of the network with the least number of epochs.
2. The η and α that, for either the worst-case or on average, yield convergence to the global minimum in the error surface with the least number of epochs.

Table 3: Simulation Results for Multilayer Perceptron Using Four Hidden Neurons[a]

Run Number	Training Set Size	Number of Epochs	Mean-Square Error	Probability of Correct Classification
1	500	320	0.2199	80.80%
2	2000	80	0.2108	80.81%
3	8000	20	0.2142	80.19%

[a]Learning-rate parameter η = 0.1 and momentum constant α = 0.

3. The η and α that, on average, yield convergence to the network configuration that has the best generalization over the entire input space, with the least number of epochs.

The terms "average" and "worst-case" used here refer to the distribution of the training input-output pairs. Definition 3 is the ideal in practice; however it is difficult to apply since minimizing the mean-square error is usually the mathematical criterion for optimality during network training and, as stated previously, a lower mean-square error over a training set does not necessarily imply good generalization. From a research point of view, definition 2 is more interesting than definition 1. In general, however, heuristic and experimental procedures dominate the optimal selection of η and α when using definition 1. For the experiment described here, we therefore consider optimality in the sense of definition 1.

Using a multilayer perceptron with two hidden neurons, combinations of learning-rate parameter $\eta \in \{0.01, 0.1, 0.5, 0.9\}$ and momentum constant $\alpha \in \{0.0, 0.1, 0.5, 0.9\}$ are simulated to observe their effect on network convergence. Each combination is trained with the same set of initial random weights and the same set of 500 examples, so that the results of the experiment may be compared directly. The learning process was continued for 700 epochs, after which it was terminated; this length of training was considered to be adequate for the back-propagation algorithm to reach a local minimum on the error surface. The ensemble-averaged learning curves so computed are plotted in Figs. 3a - 3d, which are individually grouped by η.

The experimental learning curves shown in Fig. 3 suggest the following trends:

- While, in general, a small learning-rate parameter η results in slower convergence, it can locate "deeper" local minima in the error surface than a larger η. This finding is intuitively satisfying, since a smaller η implies that the search for a minimum should cover more of the error surface than would be the case for a larger η.
- For $\eta \to 0$, the use of $\alpha \to 1$ produces increasing speed of convergence. On the other hand, for $\eta \to 1$, the use of $\alpha \to 0$ is required to ensure learning stability.
- The use of the constants $\eta = \{0.5, 0.9\}$ and $\alpha = 0.9$ causes oscillations in the mean-square error during learning and a higher value for the final mean-square error at convergence, both of which are undesirable effects.

In Fig. 4, we show plots of the "best" learning curves selected from each group of the learning curves plotted therein, so as to determine an "overall" best learning curve, "best" being defined in the sense of point 1 described previously. From Fig. 4, it appears that the optimal learning-rate parameter η_{opt} is about 0.1 and the optimal momentum constant α_{opt} is about 0.5. Thus, Table 4 summarizes the "optimal" values of network parameters used in the remainder of the experiment. The fact that the final mean-square error of each curve in Fig. 4 does not vary significantly over

the range of η and α suggests a "well-behaved" (i.e., relatively smooth) error surface for the problem.

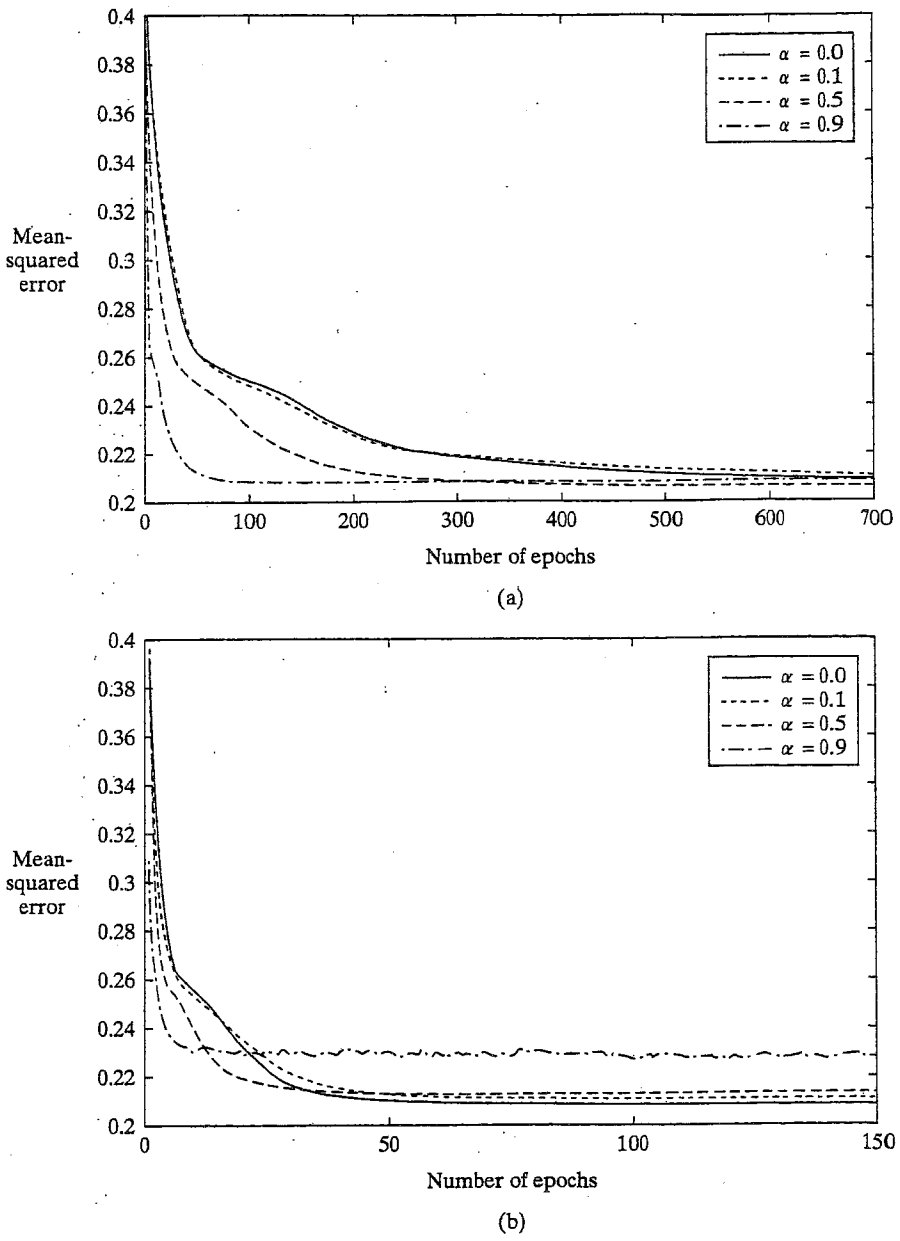
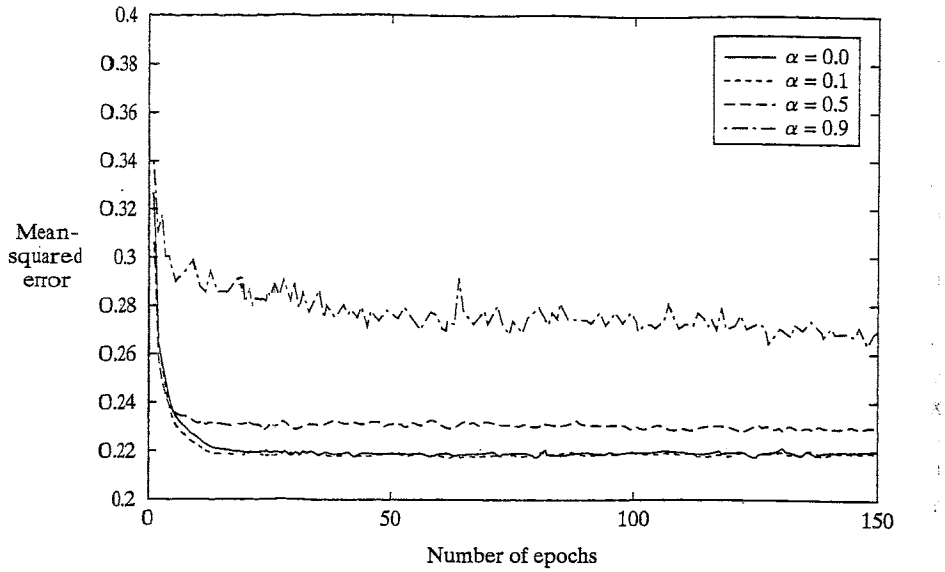
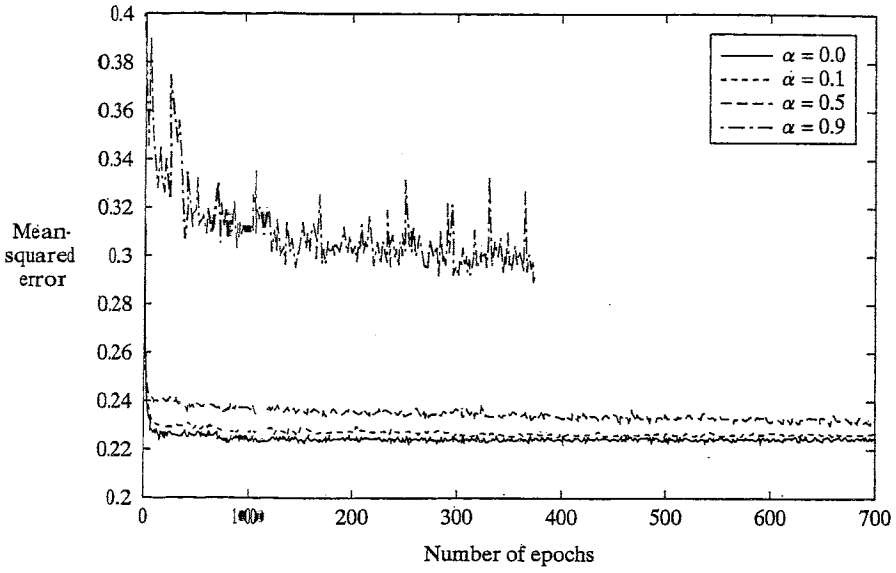


Figure 3: Problem 4.17
Ensemble-averaged learning curves for varying momentum α, and the following values of learning-rate parameters: (a) η = 0.01, (b) η = 0.1, (c) η = 0.5, and (d) η = 0.9



(c)



(d)

Figure 3: continued

Evaluation of Optimal Network Design. Given the "optimized" multilayer perceptron having the parameters summarized in Table 4, the final network is evaluated to determine its decision boundary, ensemble-averaged learning curve, and probability of correct classification. With finite-size training sets, the network function learned with the optimal parameters is "stochastic" in nature. Accordingly, these performance measures are ensemble-averaged over 20 independently trained networks. Each training set consists of 1000 examples, drawn from the distributions for classes C_1 and C_2 with equal probability, and which are presented to the network in random order. As before, the training was continued for 700 epochs. For the experimental determination of the

probabilities of correct classification, the same test set of 32,000 examples used previously is used again.

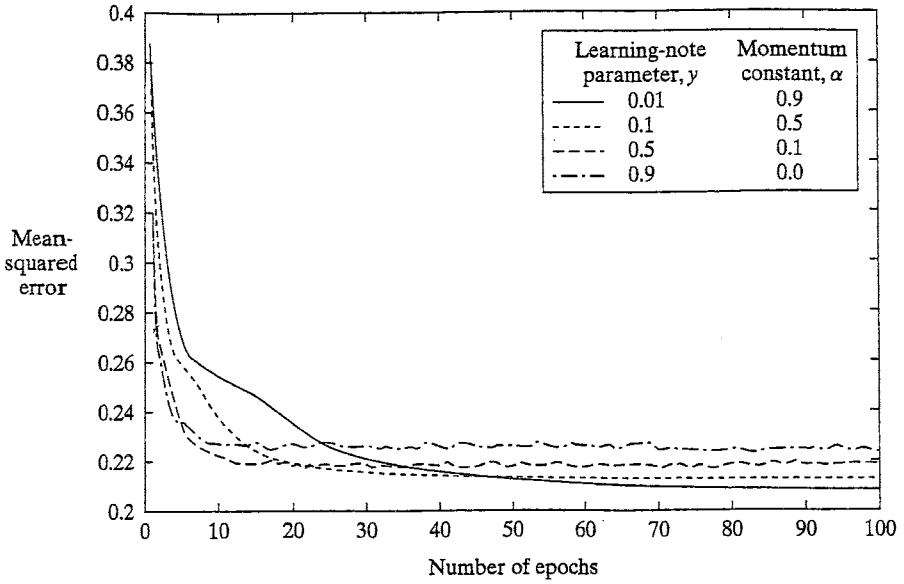


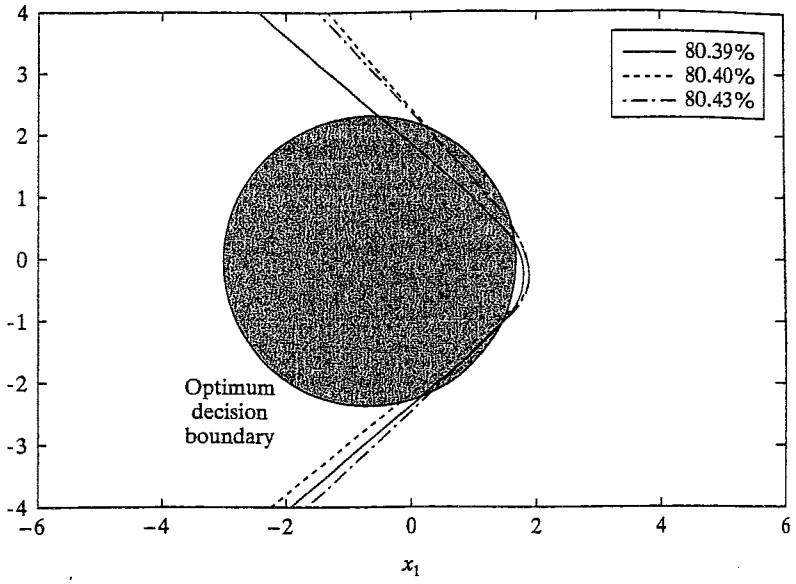
Figure 4: Problem 4.17
Best learning curves selected from the four parts of Fig. 3

Table 4: Configuration of Optimized Multilayer Perceptron

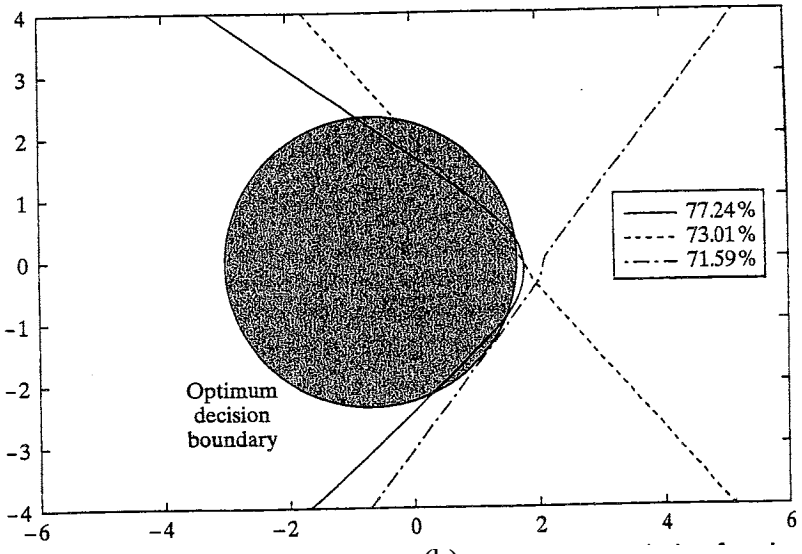
Parameter	Symbol	Value
Optimum number of hidden neurons	m_{opt}	2
Optimum learning-rate parameter	η_{opt}	0.1
Optimum momentum constant	α_{opt}	0.5

Figure 5a shows three of the "best" decision boundaries for three networks in the ensemble of 20. Figure 5b shows three of the "worst" decision boundaries for three other networks in the same ensemble. The shaded (circular) Bayesian decision boundary is included in both figures for reference. From these figures we observe that the decision boundaries constructed by the back-propagation algorithm are convex with respect to the region where they classify the observation vector $\mathbf{x}$ as belonging to class C_1 or class C_2.

The ensemble statistics of the performance measures, probability of correct classification and final mean-square error, computed over the training sample are listed in Table 5. The probability of correct classification for the optimum Bayes classifier is 81.51%.



(a)



(b)

Figure 5: Problem 4.17

(a) Plot of three "best" decision boundaries for the classification accuracies: 80.39, 80.40, and 80.43%.

(b) Plot of three "poorest" decision boundaries for the following classification accuracies: 77.14, 73.01 and 71.59%.

Table 5: Ensemble Statistics of Performance Measures (Sample Size = 20)

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

The conclusion to be drawn from this computer experiment is that a multilayer perceptron, with two hidden neurons and trained with the back-propagation algorithm having the learning-rate parameter $\gamma = 0.1$ and momentum $\alpha = 0.5$, provides a reasonably good approximation to the optimum Bayesian classifier.

Problem 4.18

Analysis of the time series

$$x(n) = \nu(n) + \beta\nu(n-1)\nu(n-2)$$

reveals that $x(n)$ has zero mean and variance

$$\sigma_x^2 = \sigma_\nu^2 + \beta^2\sigma_\nu^4$$

The autocorrelation function of $x(n)$ for lag k is

$$\begin{aligned}\mathbf{E}[x(n)x(n+k)] &= \mathbf{E}[\nu(n)\nu(n+k)] \\ &\quad + \beta\mathbf{E}[\nu(n)\nu(n+k-1)\nu(n+k-2)] \\ &\quad + \beta\mathbf{E}[\nu(n+k)\nu(n-1)\nu(n-2)] \\ &\quad + \beta^2\mathbf{E}[\nu(n+k-1)\nu(n+k-2)\nu(n-1)\nu(n-2)] \\ &= 0 \quad \text{for} \quad k \neq 0\end{aligned}$$

The time series $\{x(n)\}$ is therefore uncorrelated, and has a white spectrum. If we were to base our analysis on the first two moments, we would conclude that the optimum predictor is given by the mean of $x(n)$, which is zero. However, since the time series samples are not independent of each other, a higher order predictor can be constructed. Solving for this predictor we find that

$$\begin{aligned}\mathbf{E}[\hat{x}(n)] &= \mathbf{E}[x(n)|x(n-1)\cdot x(n-1), \ldots] \\ &= \mathbf{E}[\nu(n)] + \beta\mathbf{E}[\nu(n-1)\nu(n-2)] \\ &= \beta[\mathbf{E}\nu(n-1)\nu(n-2)]\end{aligned}$$

The above formula for the prediction $\hat{x}(n)$ is given in terms of $\nu(n-1)$ and $\nu(n-2)$. To express the prediction $\hat{x}(n)$ in terms of past samples of $x(n)$, the model would have to be inverted, which is a difficult task.

Proceeding then with a neural-network solution to the problem, Fig. 1 shows the learning curve for a multilayer perceptron trained with the back-propagation algorithm. For comparison, we have also included the learning curve for a linear predictor using the LMS algorithm in the figure.

Examining the results of Fig. 1, we may make the following observations:

- The network appears first to converge to a local minimum for which the theoretical variance of the prediction is zero. During this initial phase, the multilayer perceptron closely follows the linear predictor.

- After some time, the higher-order structure of the input time series is captured by the multilayer perceptron and the output variance increases, converging to the theoretical limit. By comparison, the linear predictor's output variance never rises significantly above the local minimum.

In other words, the error surface exhibits a local minimum for which the prediction error variance is zero, and a global minimum for which the prediction error variance is 0.25. The linear predictor can never go past the local minimum, whereas the multilayer perceptron (nonlinear predictor) is able to rise to the global minimum. The table below summarizes the final results of the experiment:

	Prediction-error variance
Multilayer perceptron	0.173
Linear predictor with momentum $\alpha = 0.9$	0.027
Linear predictor with momentum $\alpha = 0$	0.0053

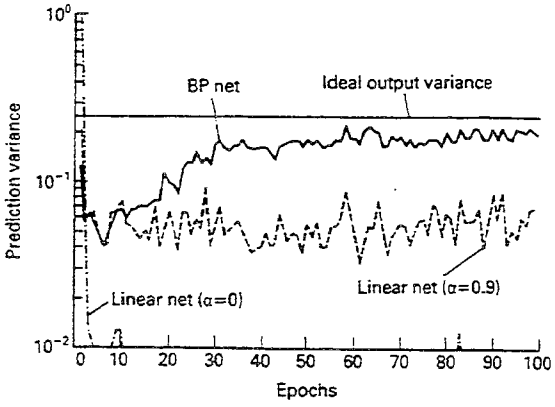


Figure 1: Problem 4.18

Problem 4.19

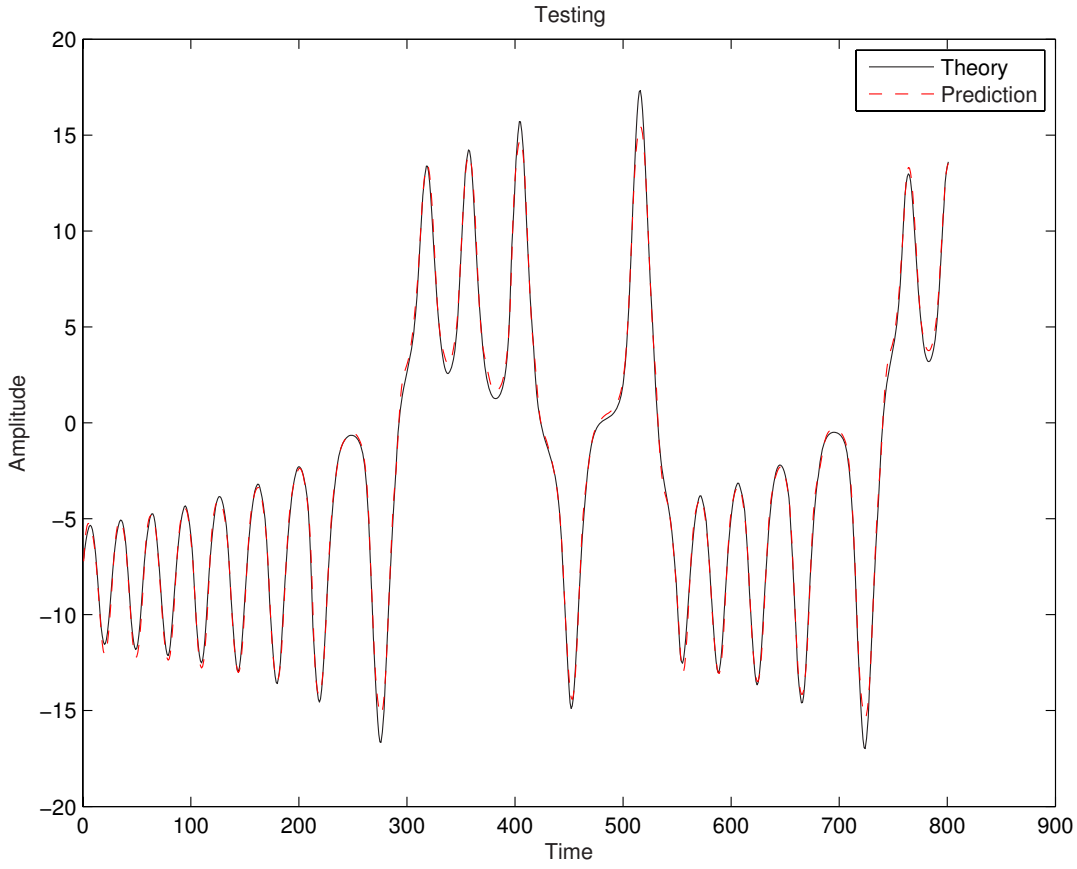


Figure 1 (Problem 4.19): Lorenz attractor

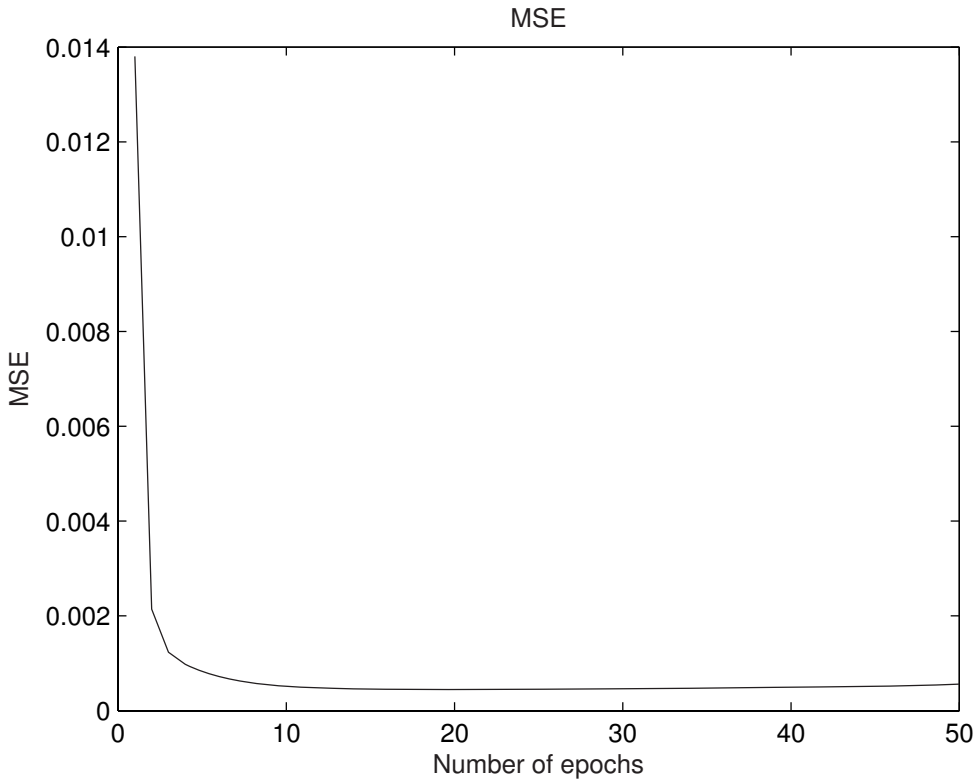


Figure 2 (Problem 4.19): MSE curve for Lorenz attractor

Summarizing Remarks:

- The computed one-step prediction of the Lorenz attractor follows the actual evolution of the attractor almost exactly.

Chapter 5: Kernel Methods and Radial-Basis Function Networks

Problem 5.10(a)

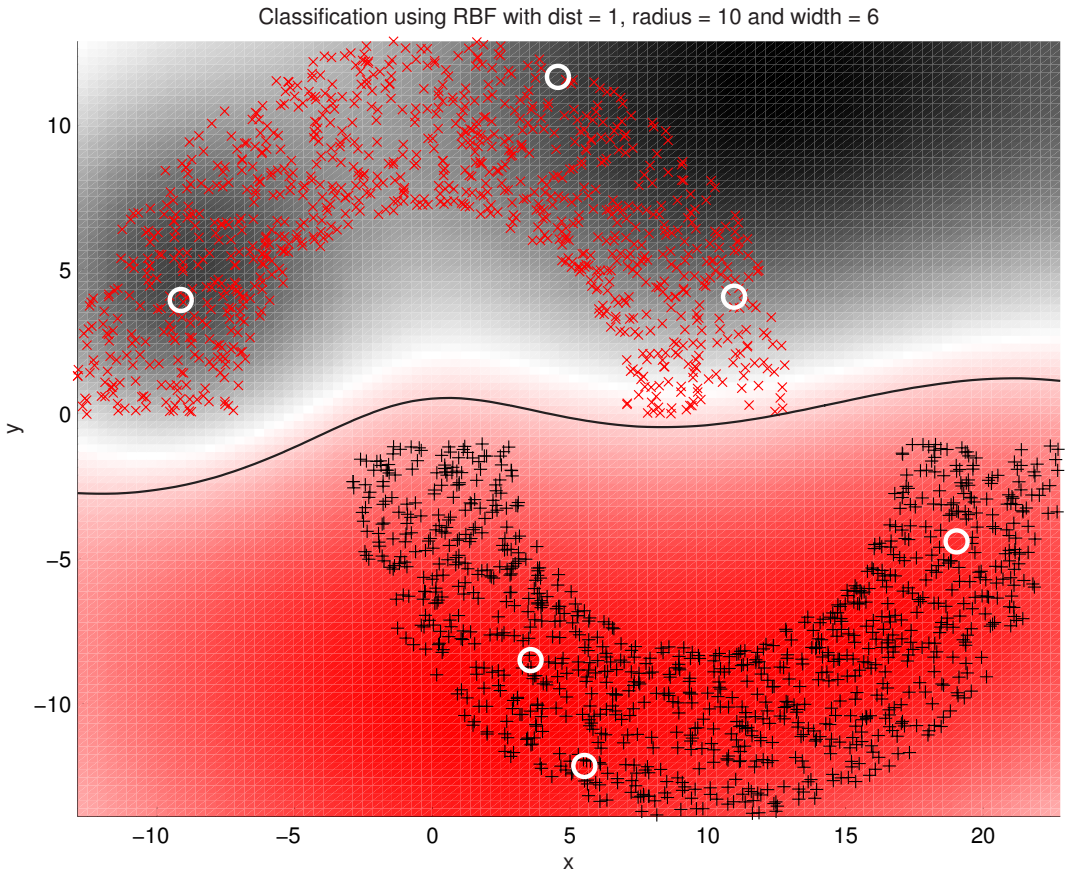


Figure 1 (Problem 5.10): Classification using RBF-RLS with $d = 1$, Error points : 0 (0.00%)

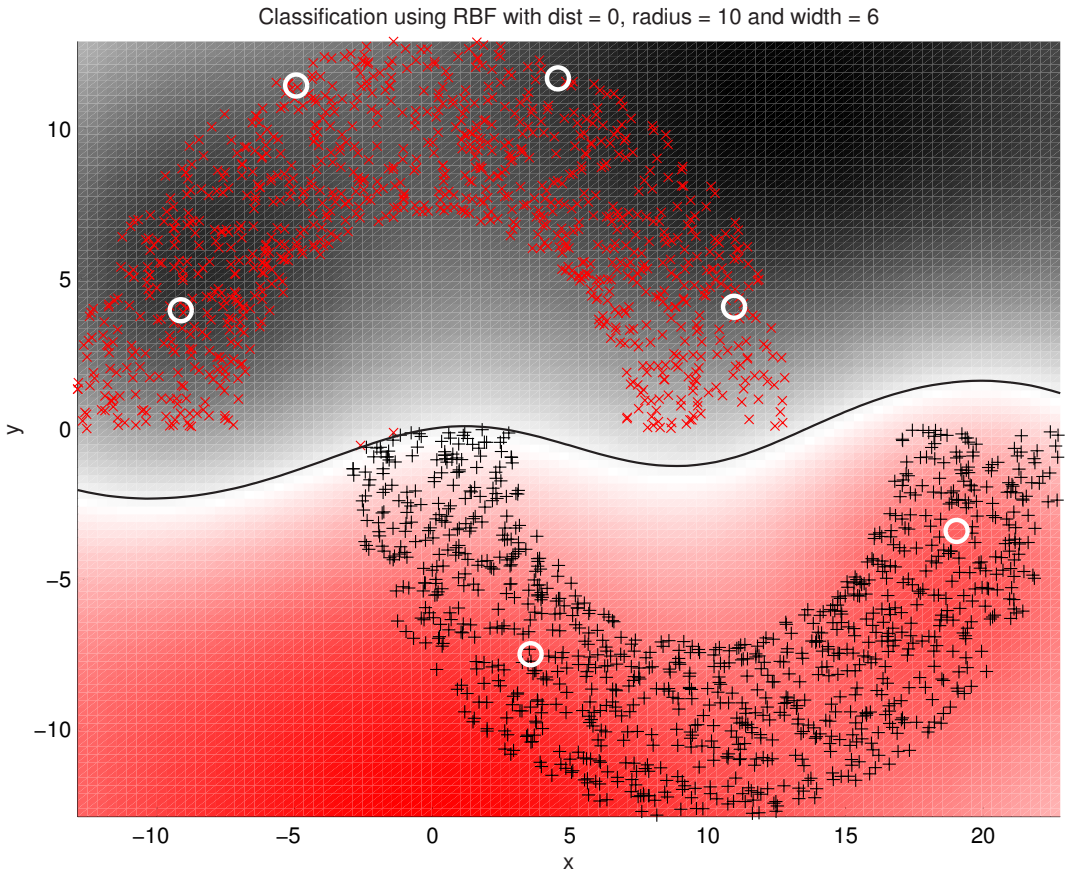


Figure 2 (Problem 5.10): Classification using RBF-RLS with $d = 0$, Error points : 2 (0.10%)

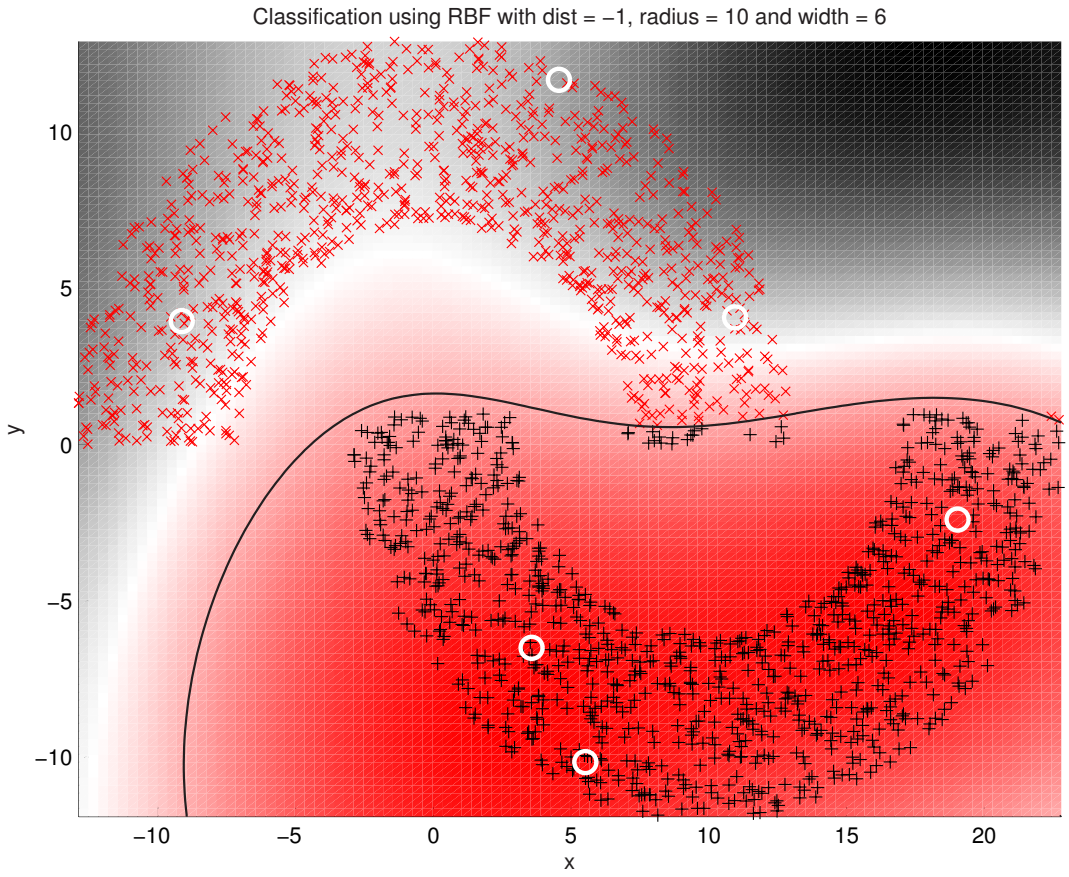


Figure 3 (Problem 5.10): Classification using RBF with $d = -1$, Error points : 20 (1.00%)

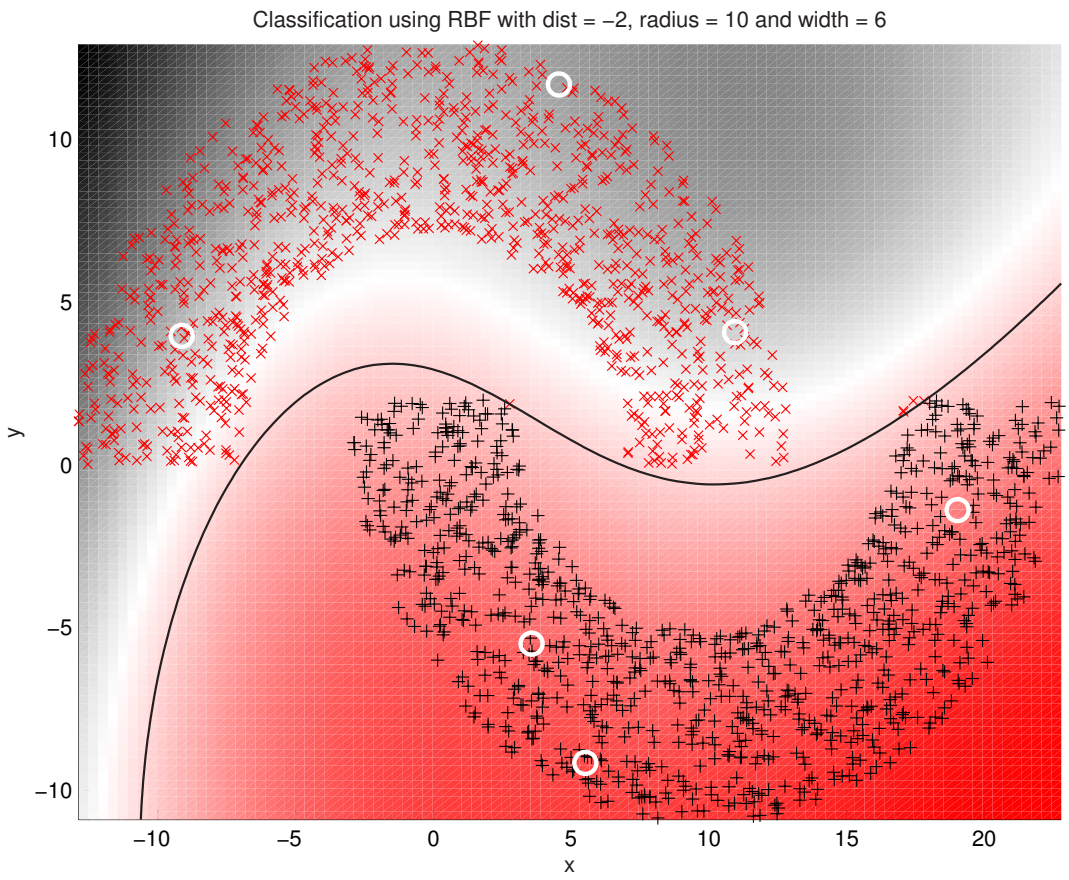


Figure 4 (Problem 5.10): Classification using RBF with $d = -2$, Error points : 5 (0.25%)

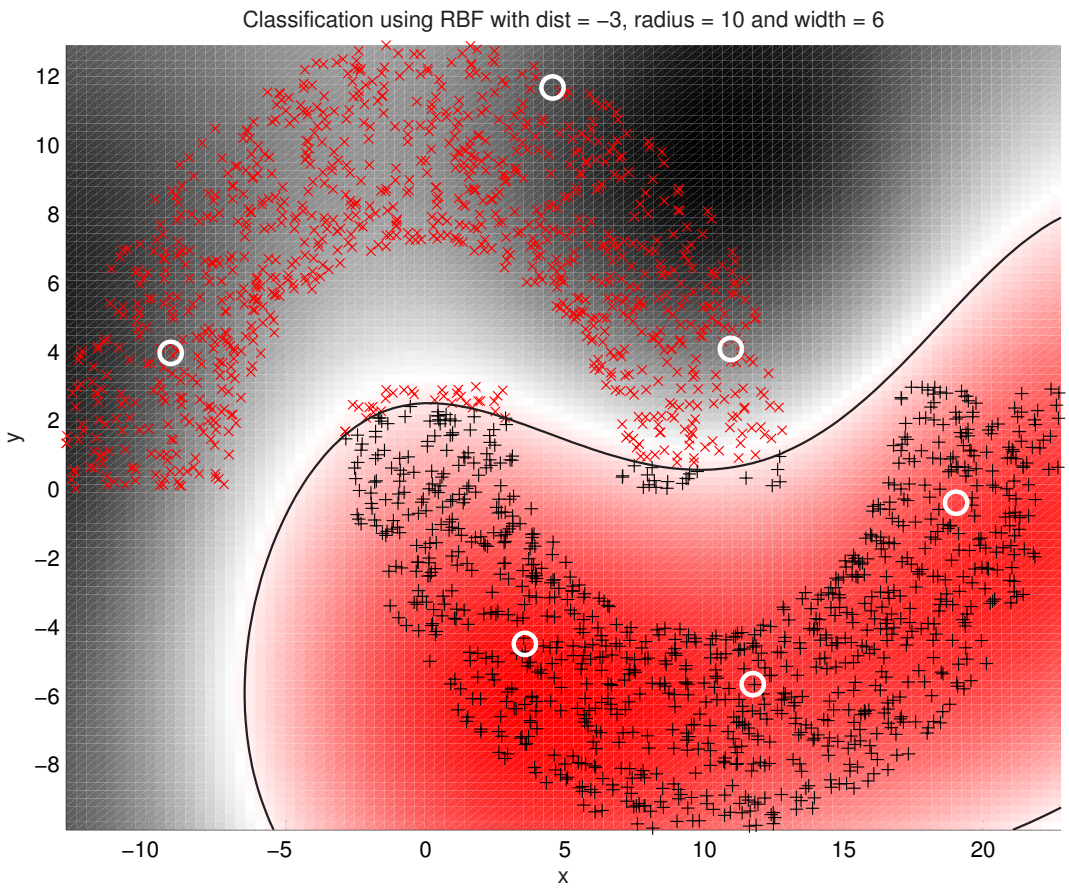


Figure 4 (Problem 5.10): Classification using RBF with $d = -3$, Error points : 53 (2.65%)

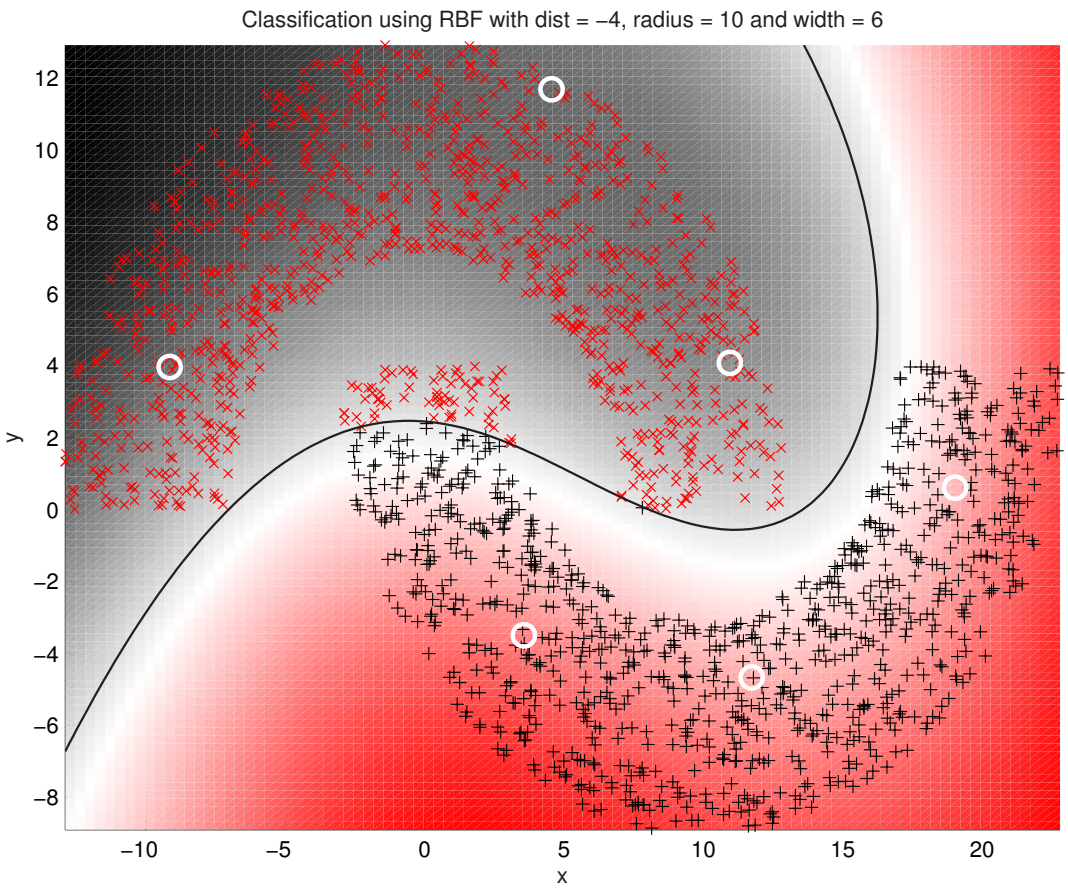


Figure 5 (Problem 5.10): Classification using RBF with $d = -4$, Error points : 69 (3.45%)

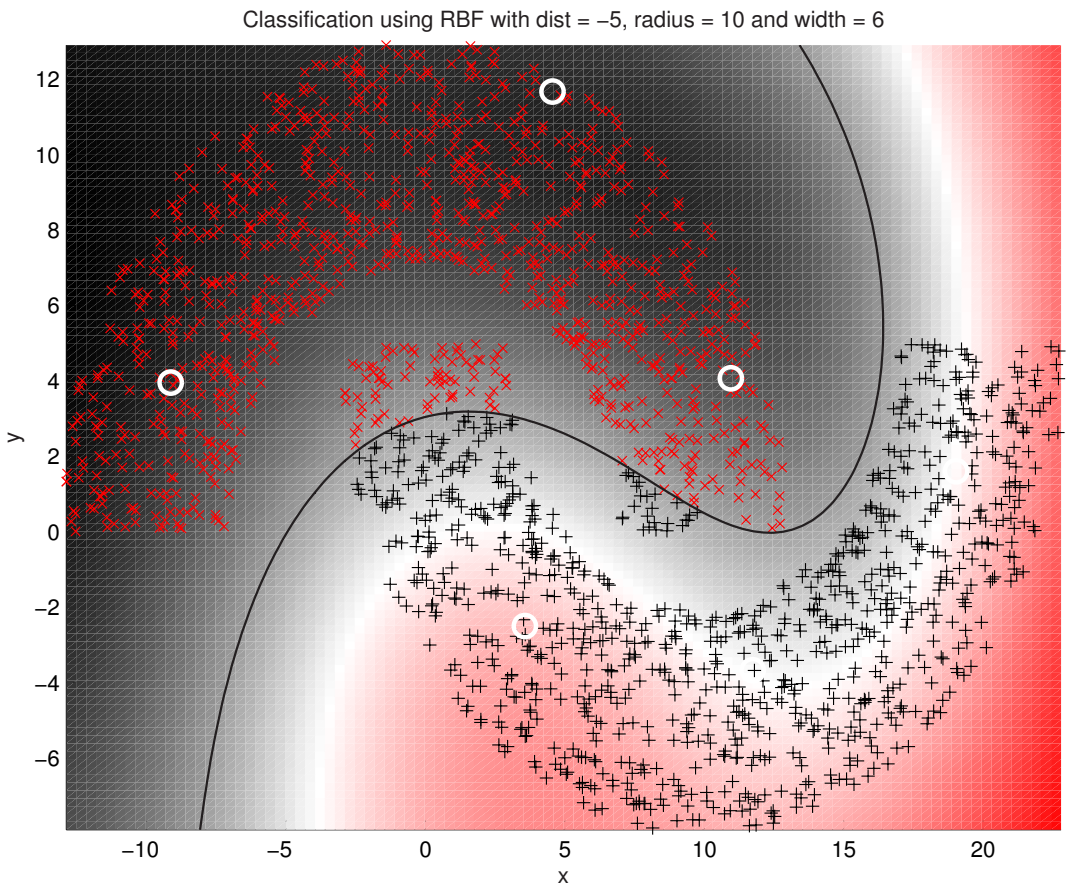


Figure 6 (Problem 5.10): Classification using RBF with $d = -5$, Error points : 101 (5.05%)

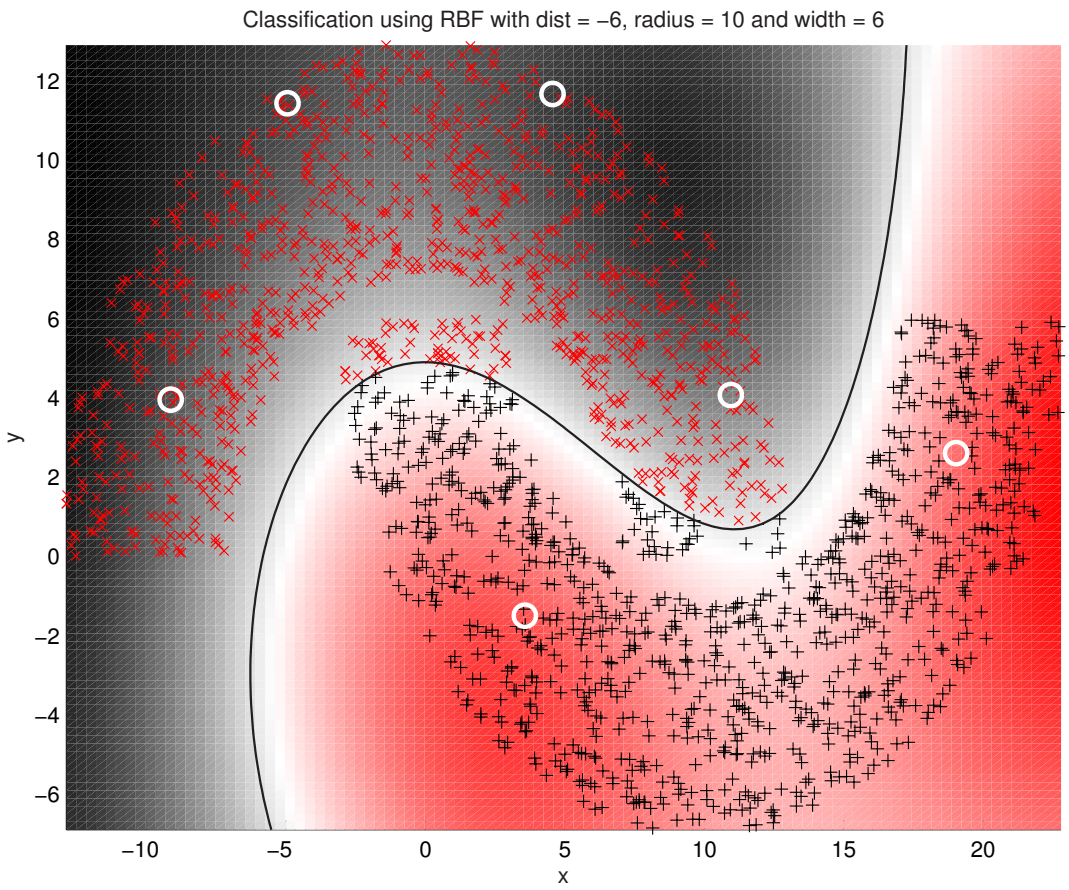


Figure 7 (Problem 5.10): Classification using RBF with $d = -6$, Error points : 92 (4.60%)

Problem 5.10(b)

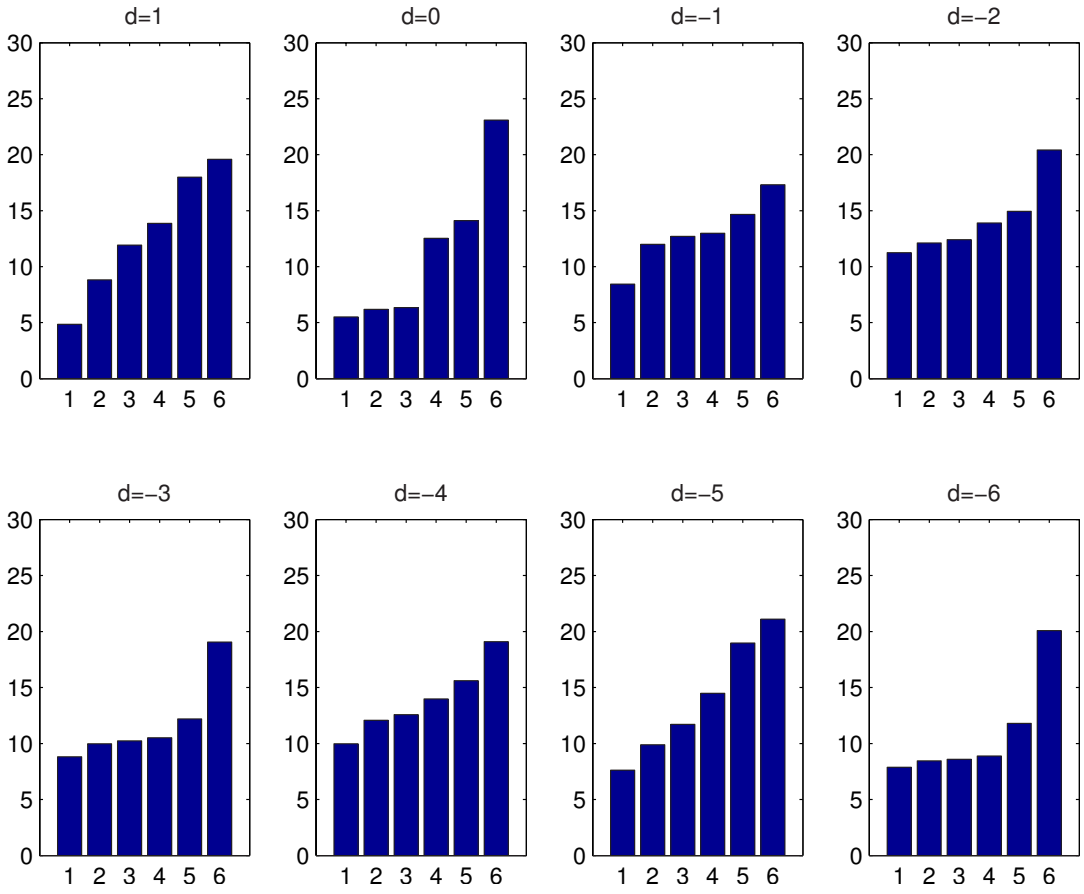


Figure 8 (Problem 5.10): Plot of width for 6 centers

Problem 5.11

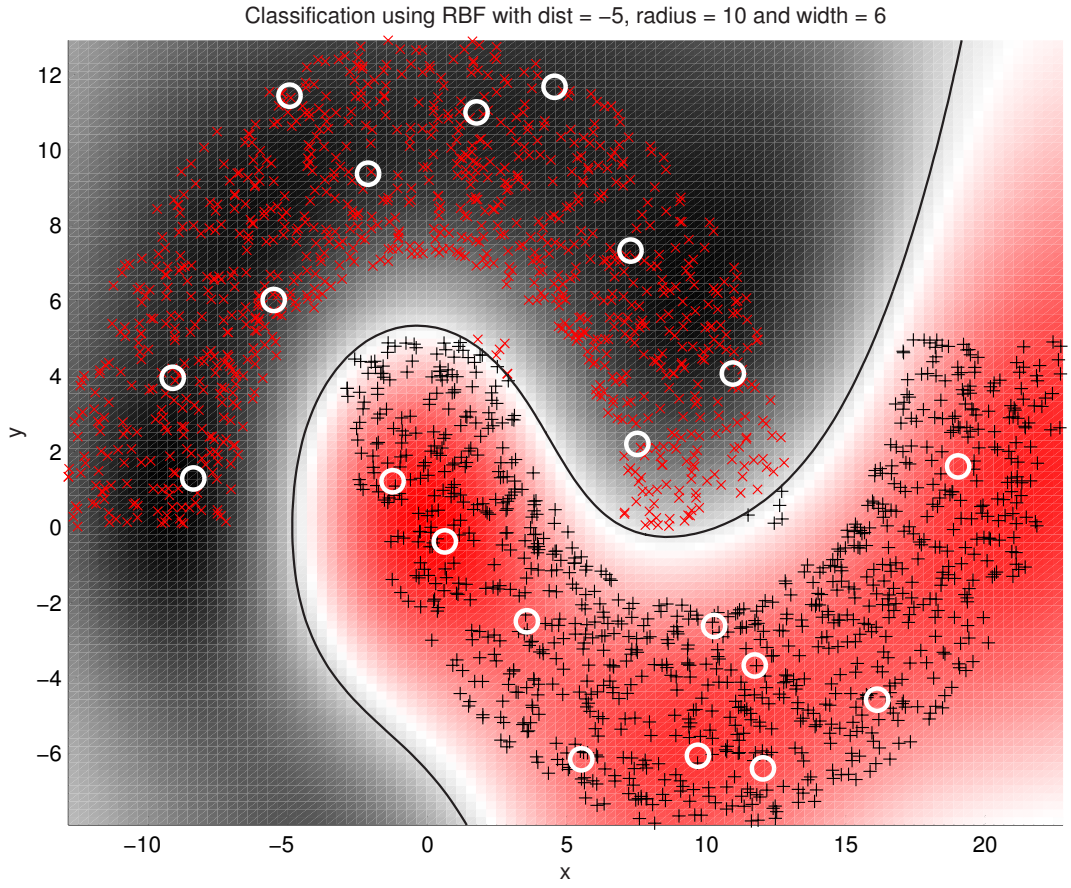


Figure 1 (Problem 5.11(a)): Classification using RBF with $d = -5$, Error points : 11 (0.55%)

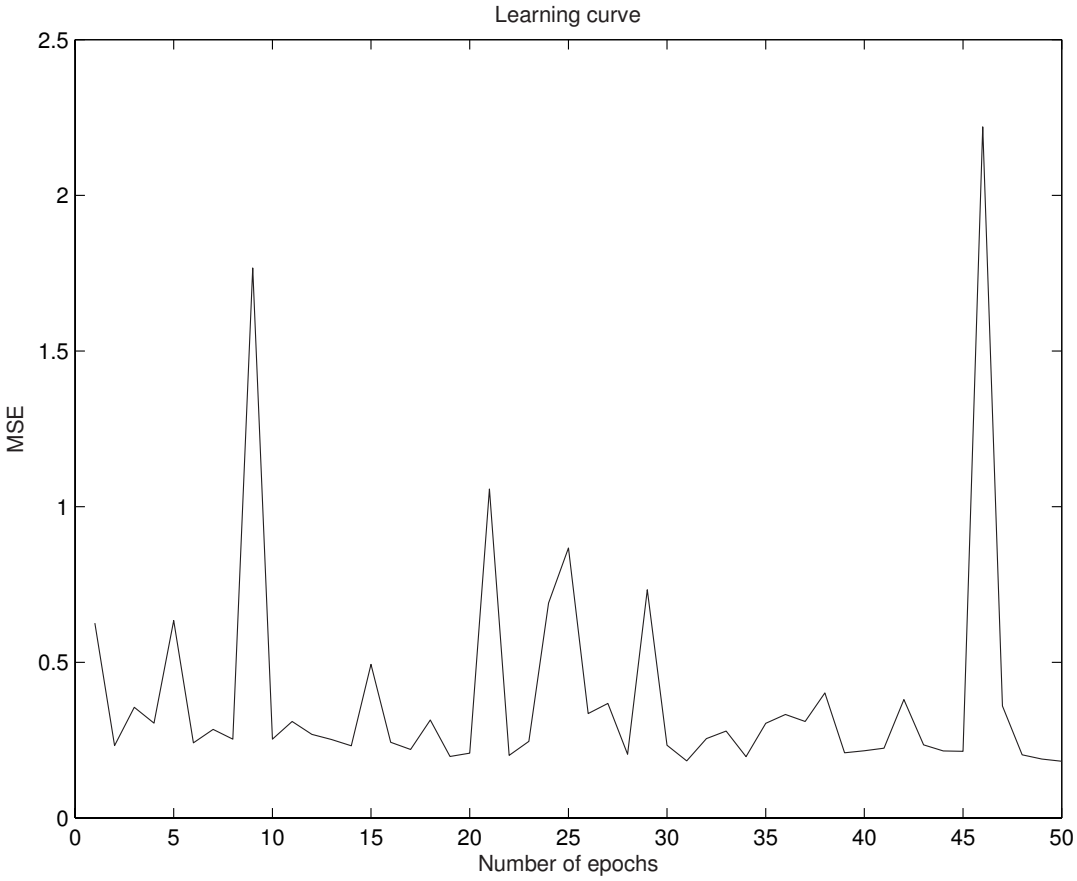


Figure 2 (Problem 5.11(a)): Classification using RBF with $d = -5$: learning curve, Error points : 11 (0.55%)

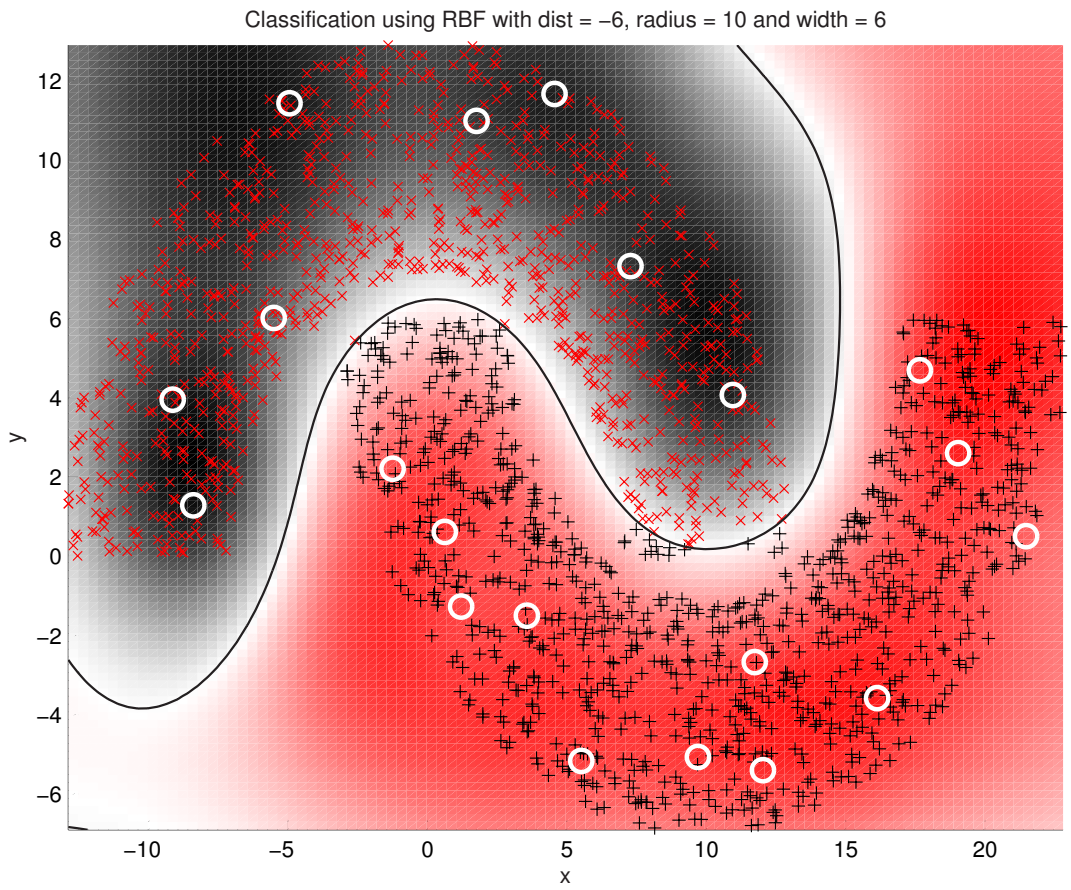


Figure 3 (Problem 5.11(b)): Classification using RBF with $d = -6$, Error points : 17 (0.85%)

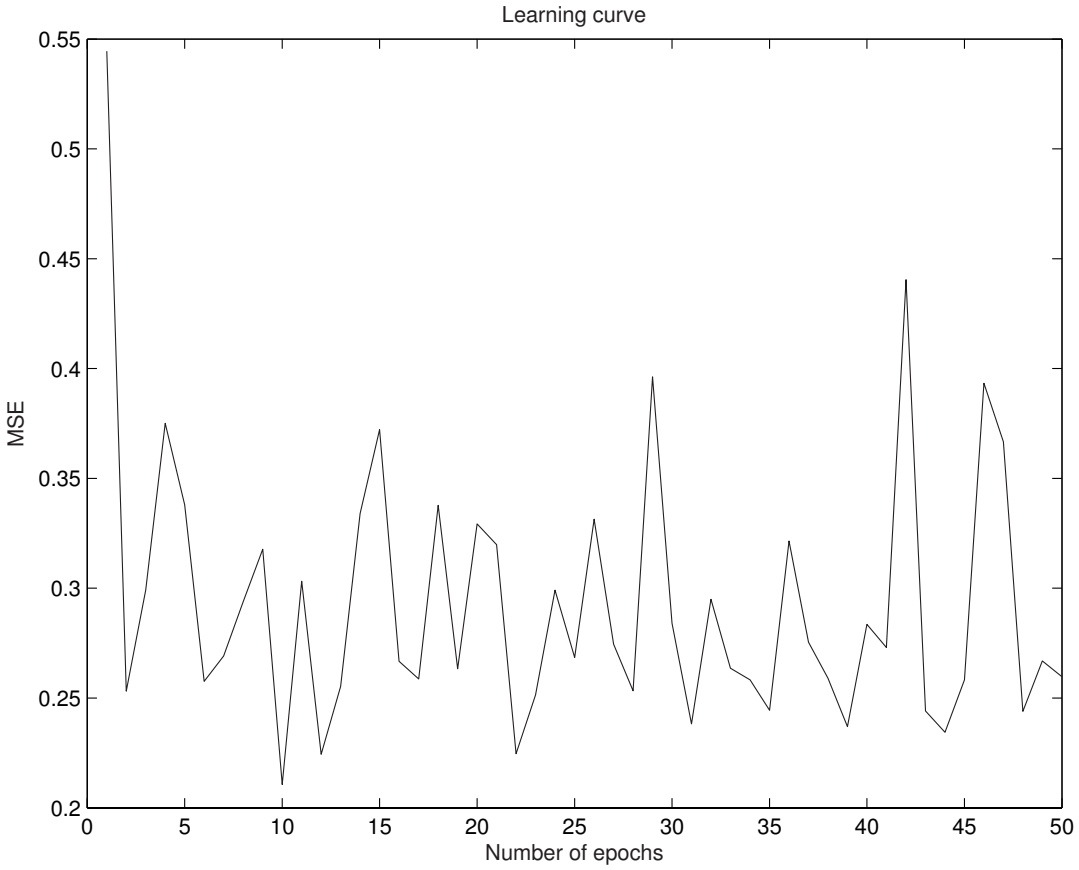


Figure 4 (Problem 5.11(b)): Classification using RBF with $d = -6$: learning curve, Error points : 17 (0.85%)

Summary of results:

- The computational complexity of the K-means LMS is linear withe respect to the size of the output layer, and in the case of the K-means RLS, the computational complexity follows a square law.

Chapter 6: Support Vector Machines

Problem 6.24

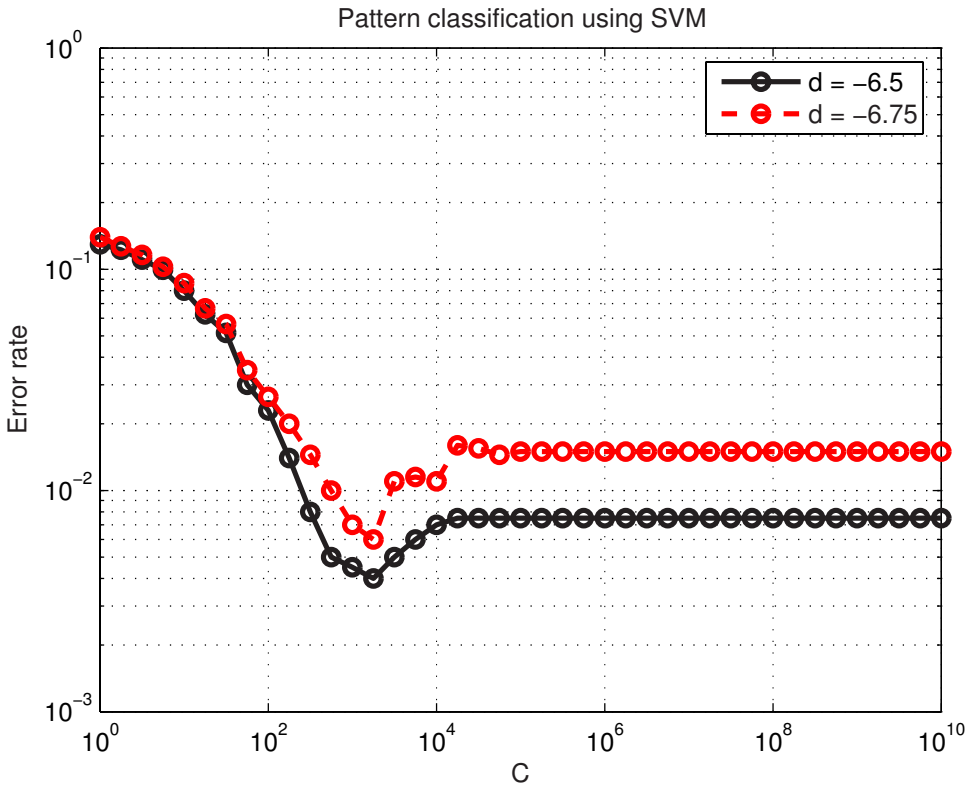


Figure 1 (Problem 6.24): Classification using SVM with $d = -6.5, -6.75$: C vs Error rate

Summarizing remarks:

- (a) The approximate value is $C = 1780$ when the classification error is reduced to a minimum.
- (b) As the distance of separation between the two half-moons is reduced from $d = -6.5$ to $d = -6.75$, the mean-square error is correspondingly reduced, going from 0.18 to 0.0075.

Problem 6.25

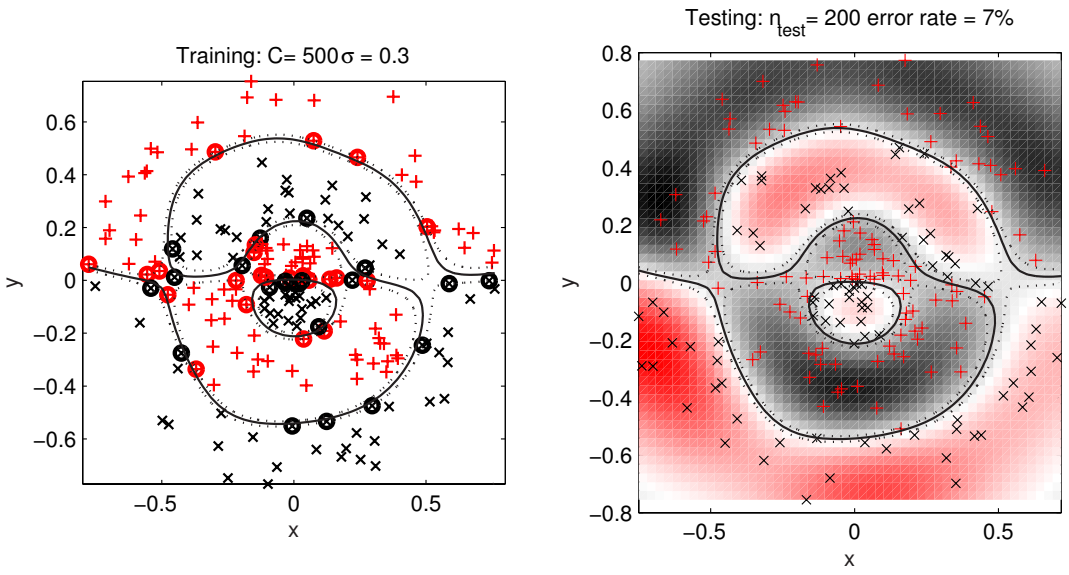


Figure 1 (Problem 6.25): Classification using SVM for tight-fisted problem with $d_1 = 0.2, d_2 = 0.5, d_3 = 0.8$ **and** $C = 500$**. Error points : 14 (7.00%)**

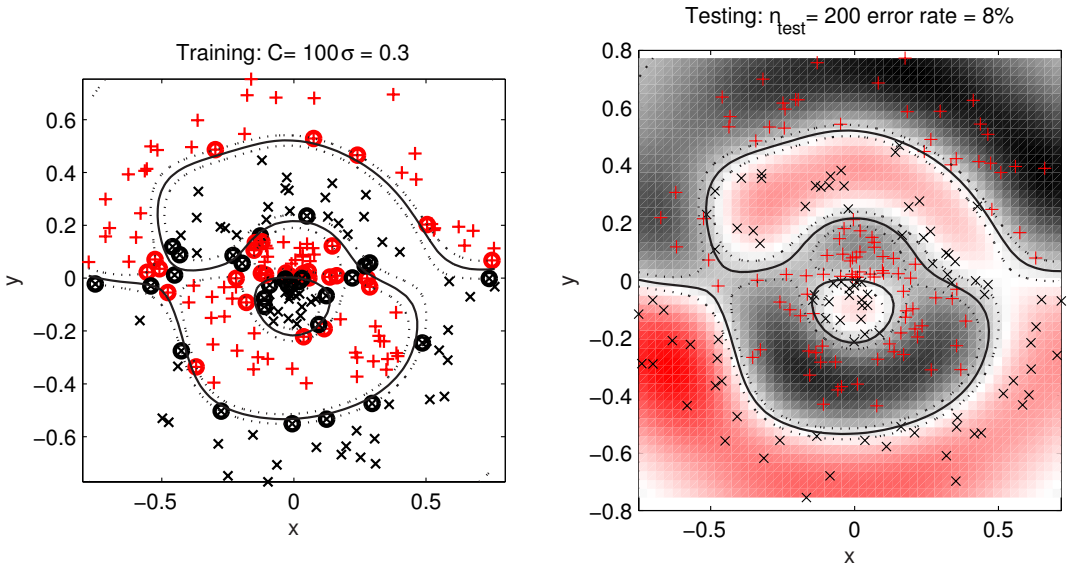


Figure 2 (Problem 6.25): Classification using SVM for tight-fisted problem with $d_1 = 0.2, d_2 = 0.5, d_3 = 0.8$ **and** $C = 100$**. Error points : 16 (8.00%)**

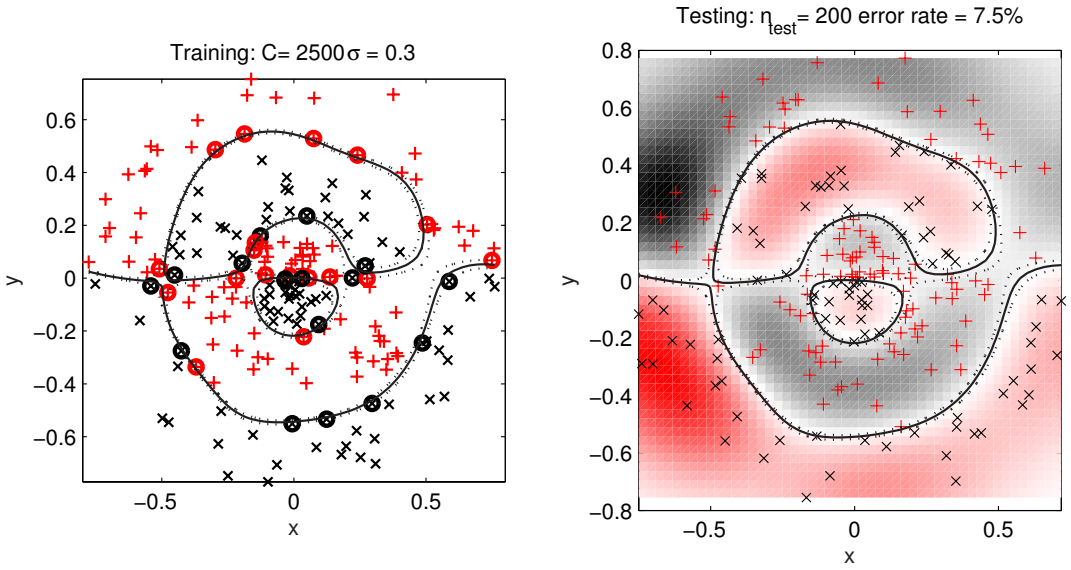


Figure 3 (Problem 6.25): Classification using SVM for tight-fisted problem with $d_1 = 0.2, d_2 = 0.5, d_3 = 0.8$ **and** $C = 2500$**. Error points : 15 (7.50%)**

Summarizing remarks:

- The classification error rate is 7.00% for $C = 500$.
- The classification error rate is 8.00% for $C = 100$.
- The classification error rate is 7.50% for $C = 2500$.
- It therefore appears that $C = 500$ is near-optimal.

Chapter 7: Regularization Theory

Problem 7.20

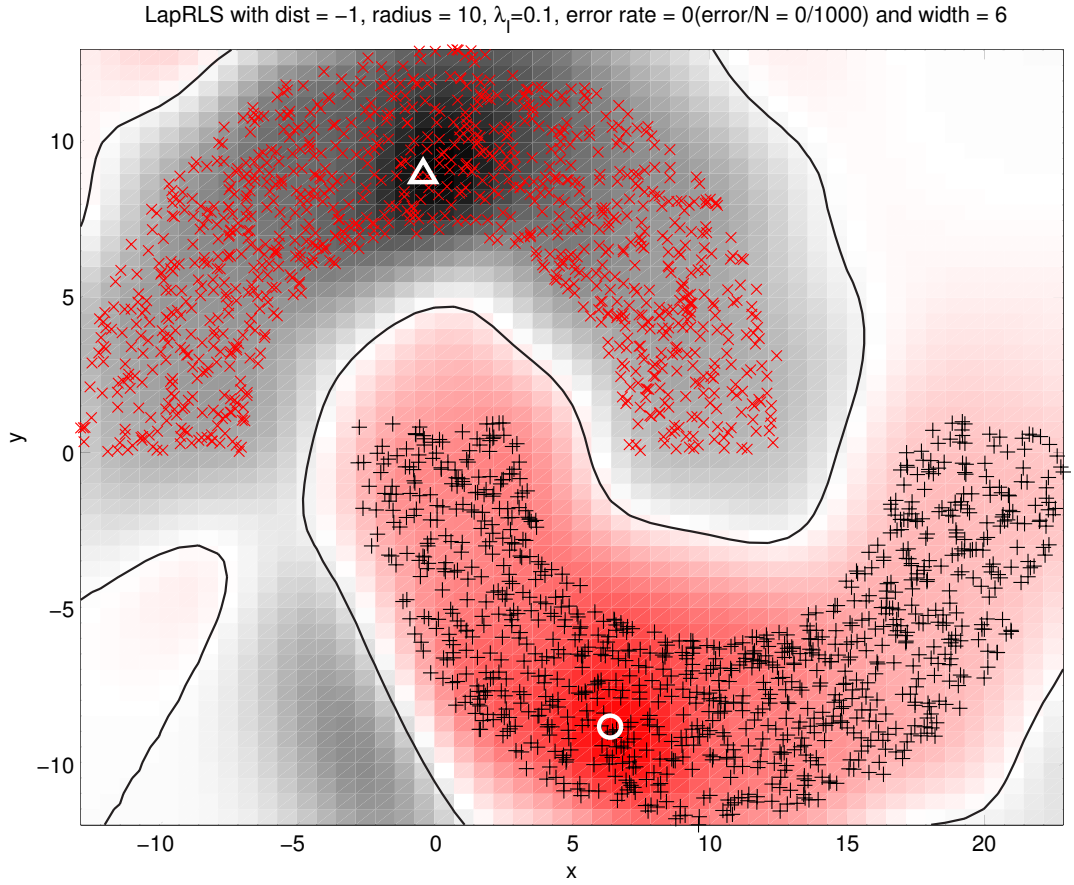


Figure 1 (Problem 7.20(a)): Classification using LapRLS for 1-labeled data with $\lambda = 0.1$: Case A

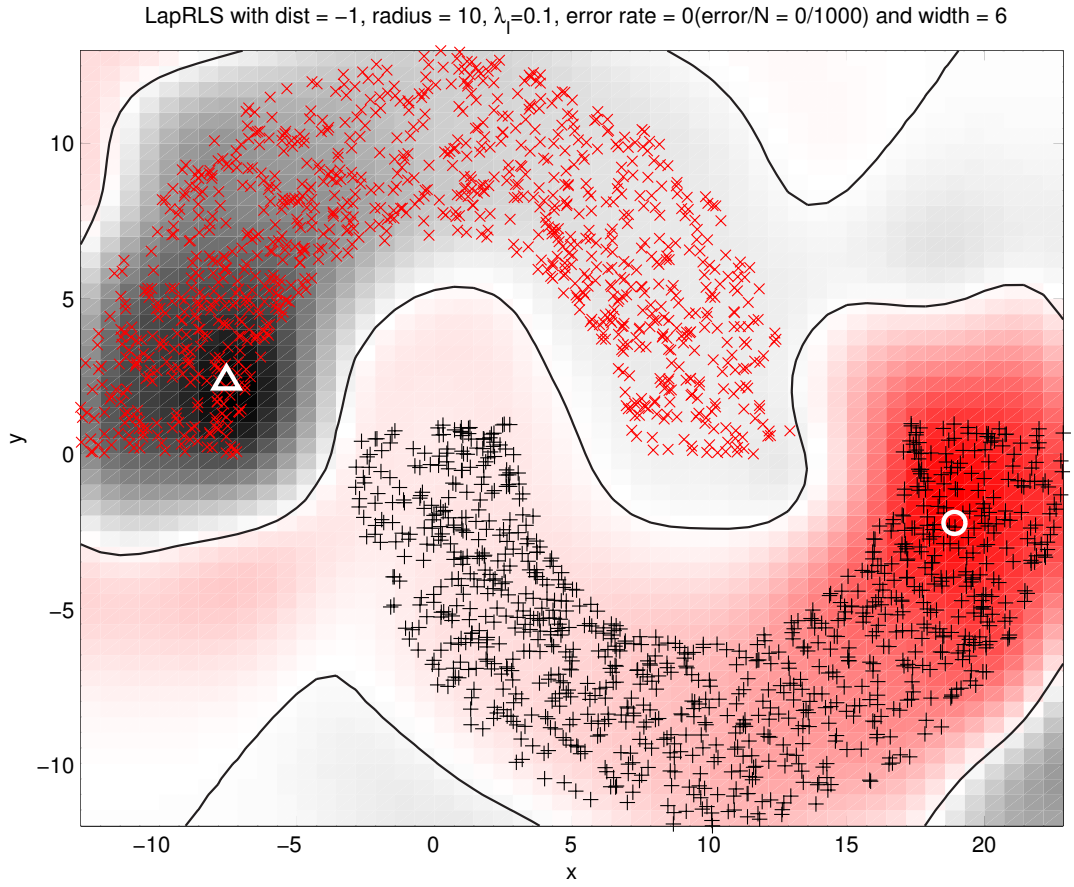


Figure 2 (Problem 7.20(a)): Classification using LapRLS for 1-labeled data with $\lambda = 0.1$: Case B

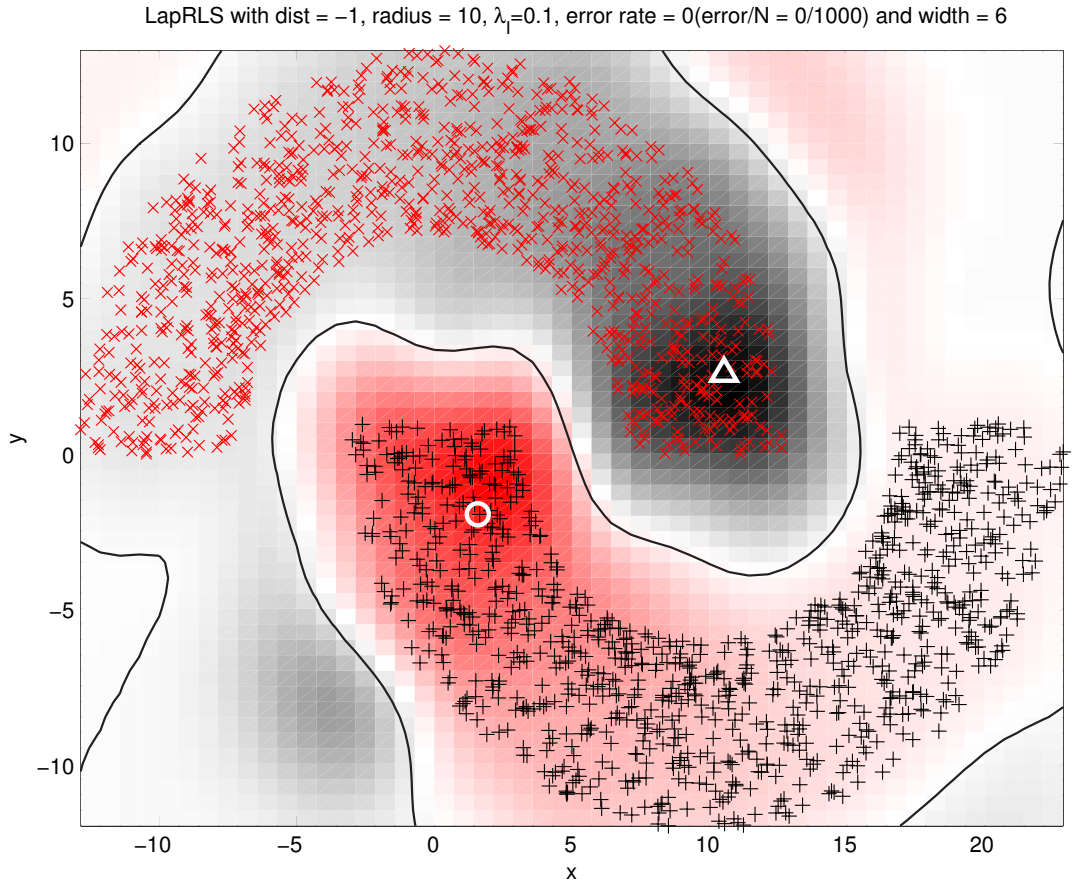


Figure 3 (Problem 7.20(a)): Classification using LapRLS for 1-labeled data with $\lambda = 0.1$: Case C

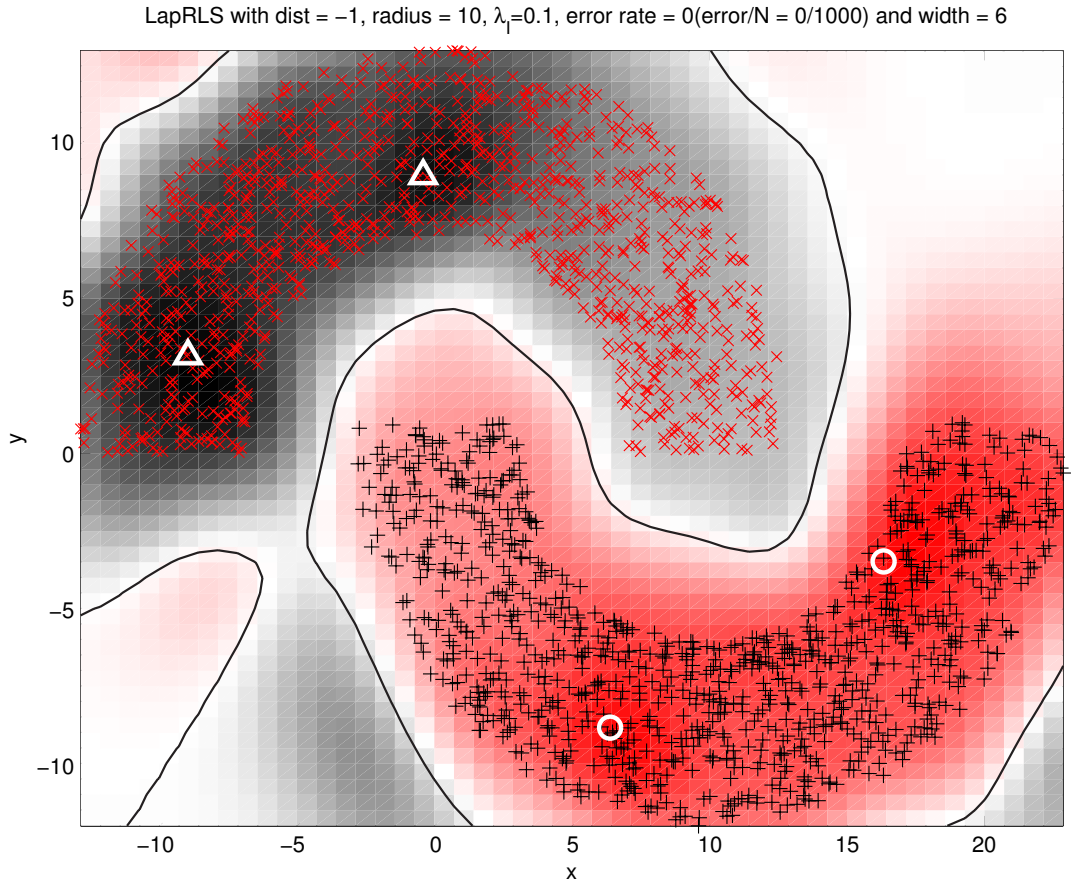


Figure 4 (Problem 7.20(b)): Classification using LapRLS for 2-labeled data with $\lambda = 0.1$: Case A

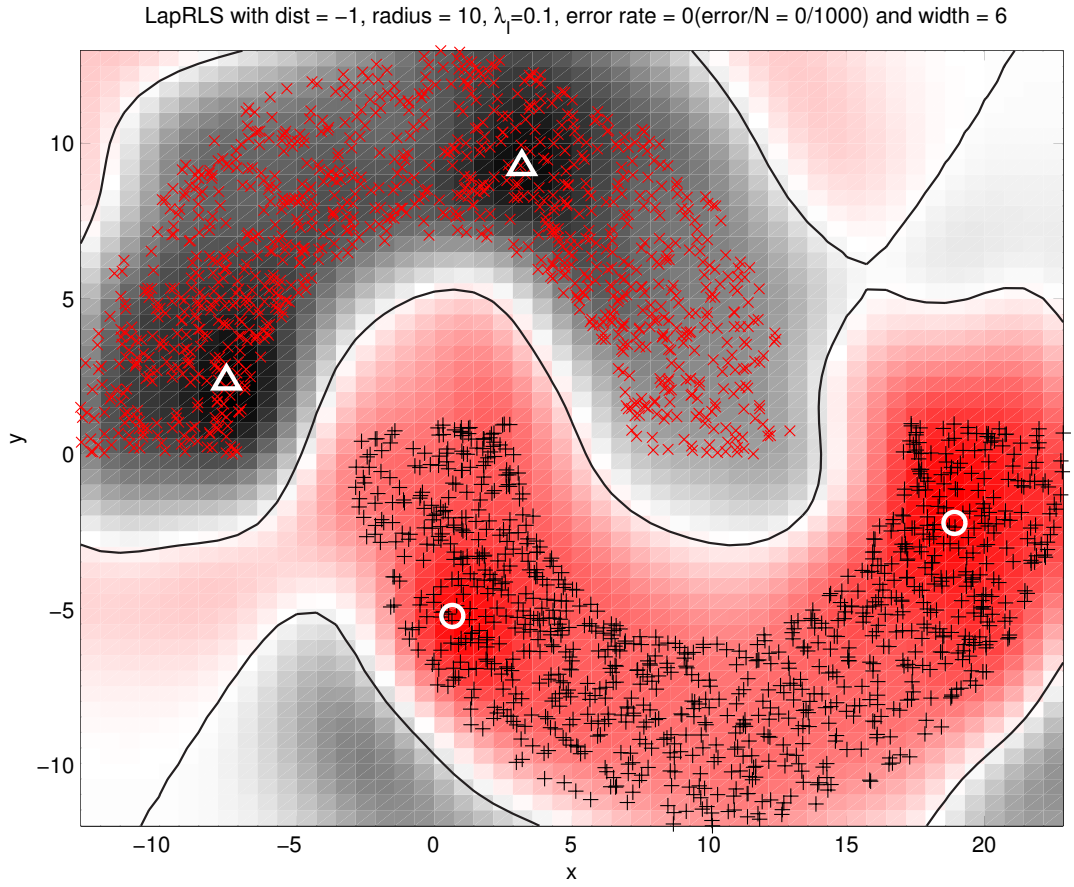


Figure 5 (Problem 7.20(b)): Classification using LapRLS for 2-labeled data with $\lambda = 0.1$: Case B

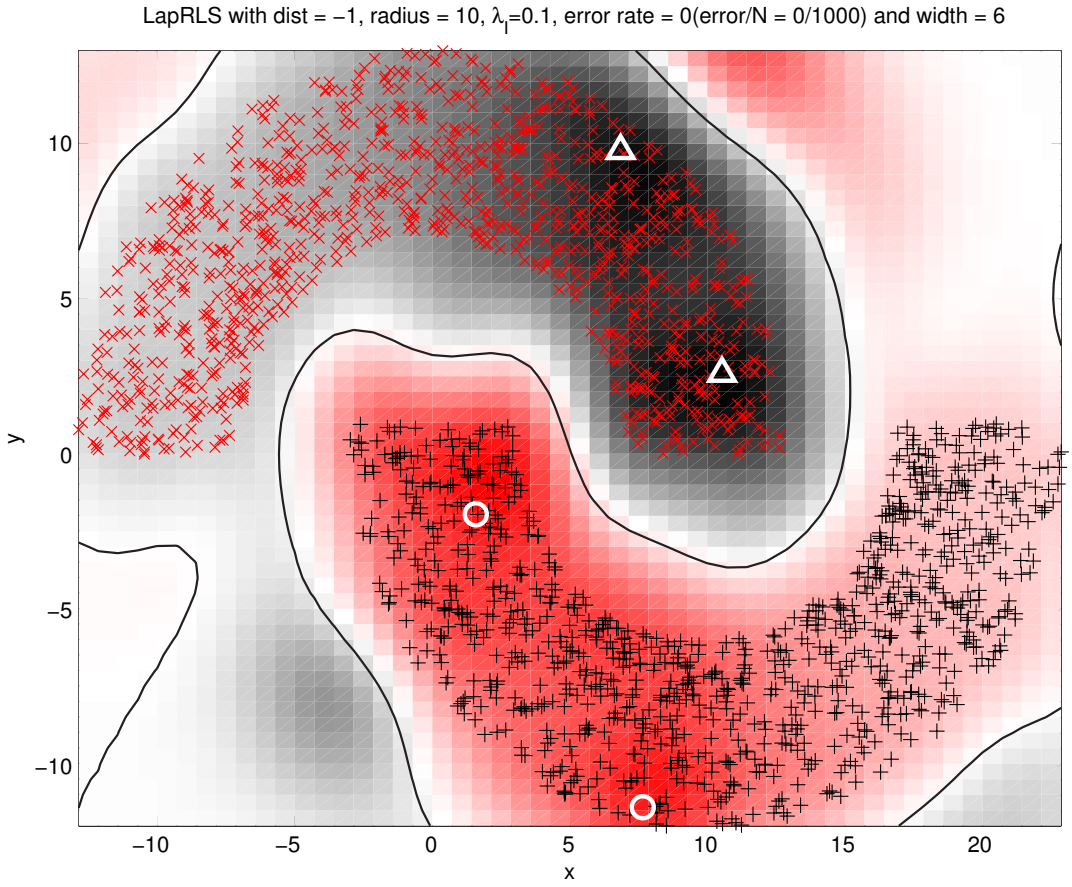


Figure 6 (Problem 7.20(b)): Classification using LapRLS for 2-labeled data with $\lambda = 0.1$: Case C

Summarizing remarks:

- For both cases of combined one-labeled data point and two-labeled data points, the decision boundary depends on the position of the labeled data points. Moreover, we could see clustering of the data around the labeled data points.

Chapter 8: Principal-Components Analysis

Problem 8.17

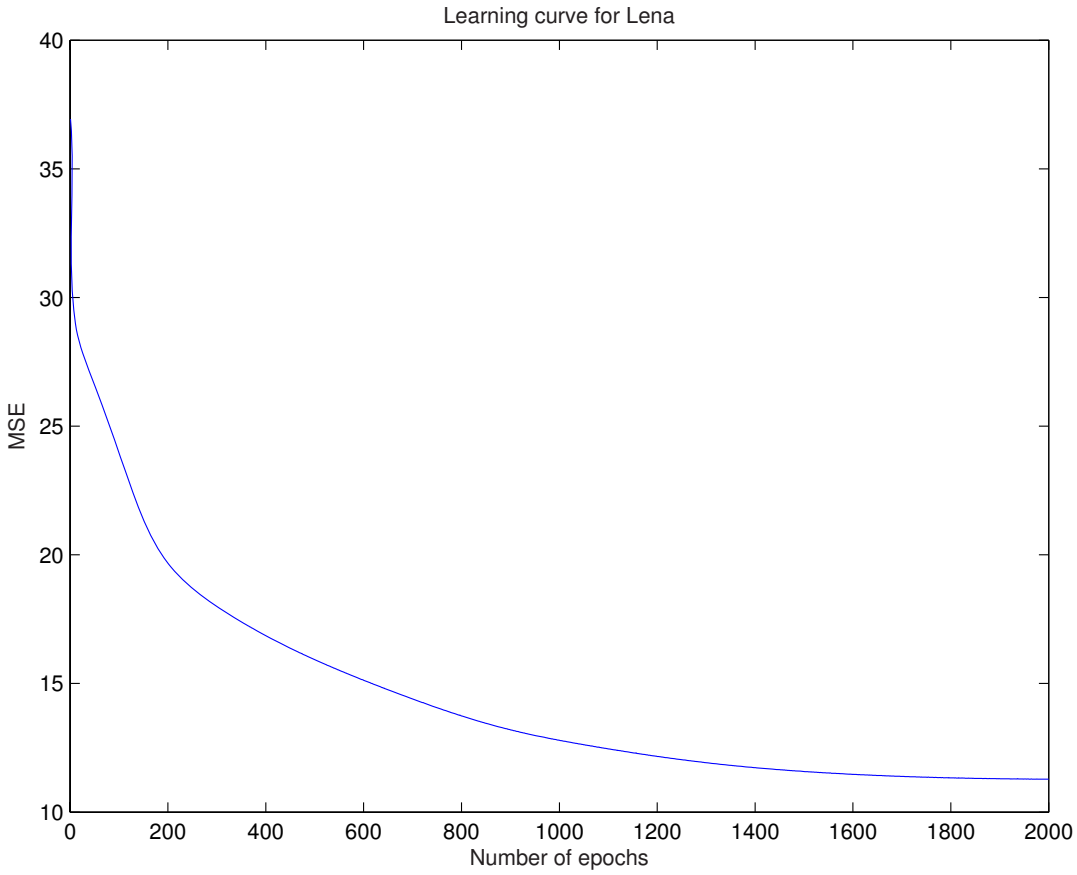


Figure 1 (Problem 8.17(a)): Learning curve of GHA for Lena image

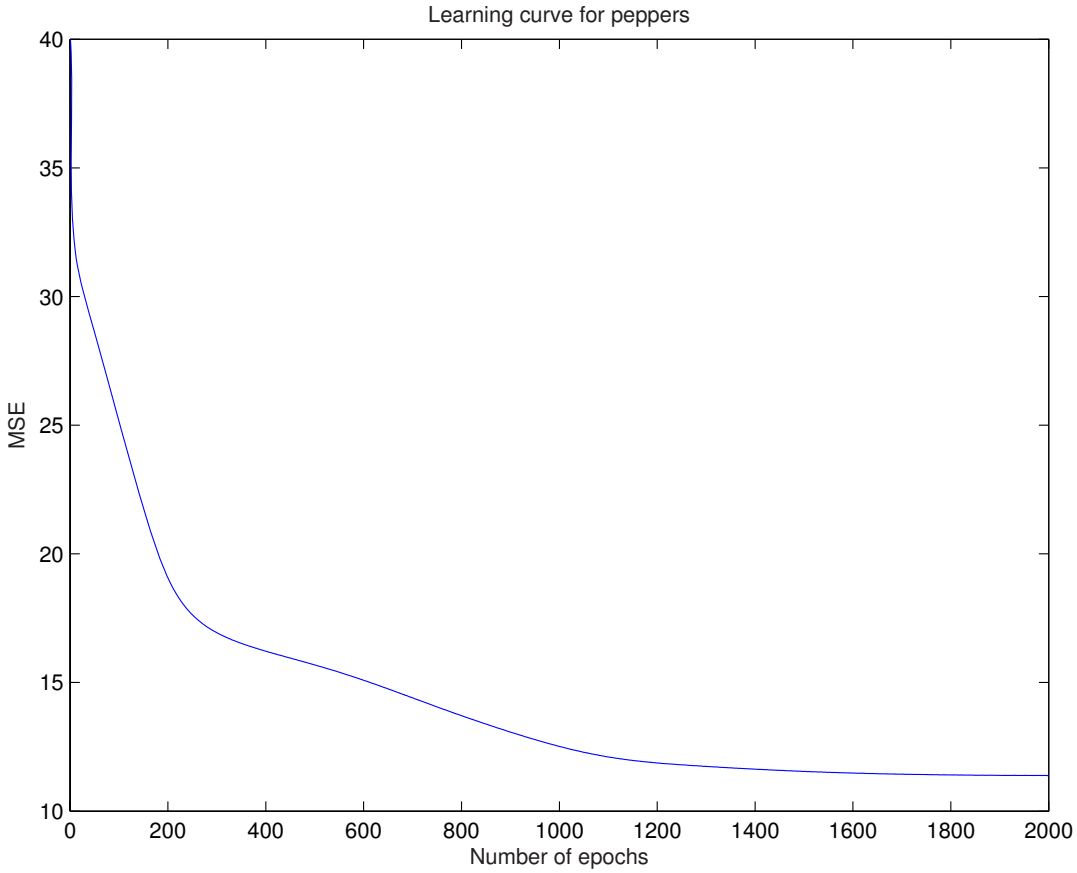


Figure 2 (Problem 8.17(b)): Learning curve of GHA for peppers image

Problem 8.18

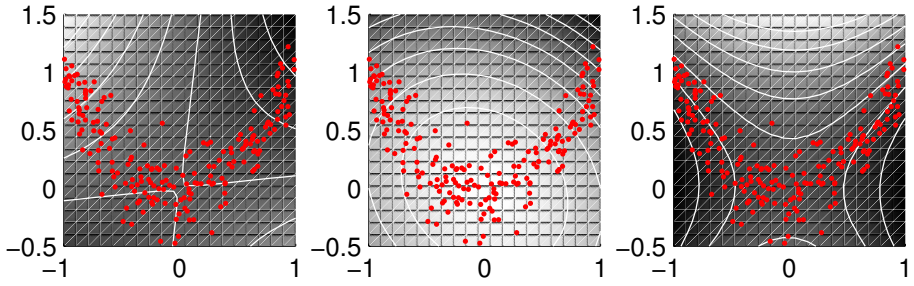


Figure 1 (Problem 8.18): 2D example illustrating the Kernel Hebbian algorithm

Summarizing remarks:

- (a) In Figure 1, we have used the KHA to plot the results for the assigned two-dimensional distribution:

  - Uniform distribution, over the interval $[-1, +1]$ along the x-axis,

  - Distribution along the y-axis in accordance with the formula:

    $$y = -x^2 + v$$

    where v denotes Gaussian noise of zero mean and standard deviation 0.2 (i.e., variance 0.04).

  Figure 1 plots the contour lines produced by the KHA, using the prescribed set of 150 data points. These results were obtained under the following conditions:

  - number of weights updates: 1.5×10^4

  - learning-rate parameter: 0.05

- (b) Comparing the results plotted in Figure 1 with the corresponding kernel PCA results displayed along the second column of Fig. 8.13 in the text for $d = 2$, we make two observations:

– Visual similarity of the two sets of principal components, except for a sign difference,

– Capability of the KHA to produce a solution close enough to the kernel PCA for all practical purposes.

Chapter 9: Self-Organizing Maps

Problem 9.11

The results of computer simulation for a one-dimensional lattice with a two-dimensional (triangular) input are shown in Fig. 1 on the next page for an increasing number of iterations. The experiment begins with random weights at zero time, and then the neurons start spreading out.

Two distinct phases in the learning process can be recognized from this figure:

- The neurons become ordered (i.e., the one-dimensional lattice becomes untangled), which happens at about 20 iterations.
- The neurons spread out to match the density of the input distribution, culminating in the steady-state condition attained after 25,000 iterations.

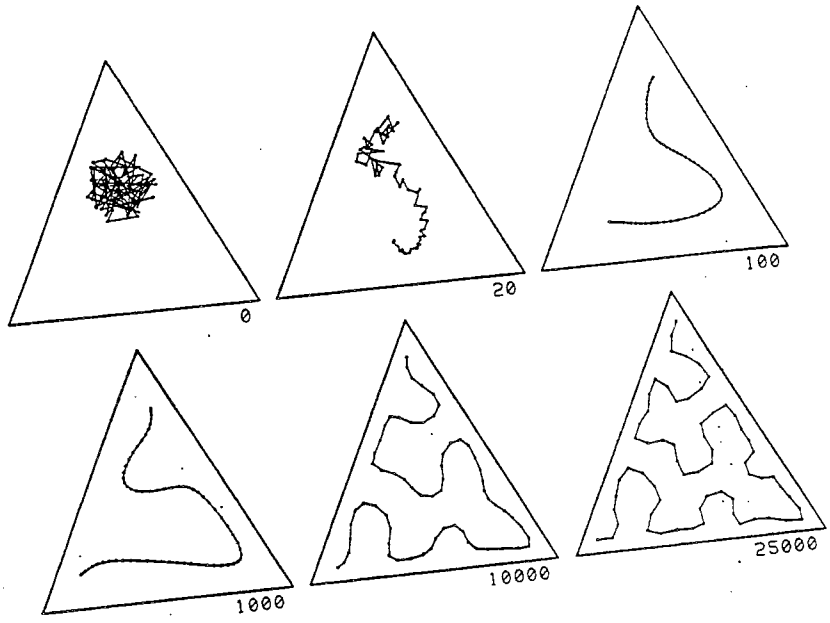


Figure 1: Problem 9.11

Problem 9.12

Parts (a), (b), and (c) of Fig. 1 display the two-dimensional feature map trained on a three-dimensional input distribution defined in the thin sheet $\{(0 < x_1 < 1); (0 < x_2 < 1); (0 < x_3 < 0.2)\}$. The input distribution is indicated by the dotted lines. The SOM algorithm generates a two-dimensional projection of the input (data) space onto the output (neuron) space. Specifically, parts (a), (b), and (c) of the figure show the status of the map after 50, 1000, and 10,000 iterations, respectively.

Figures 2 and 3 display the solutions computed by the SOM algorithm for the following respective input sheets:

Figure 2: $\{(0 < x_1 < 1); (0 < x_2 < 1); (0 < x_3 < 0.4)\}$
Figure 3: $\{(0 < x_1 < 1); (0 < x_2 < 1); (0 < x_3 < 1)\}$

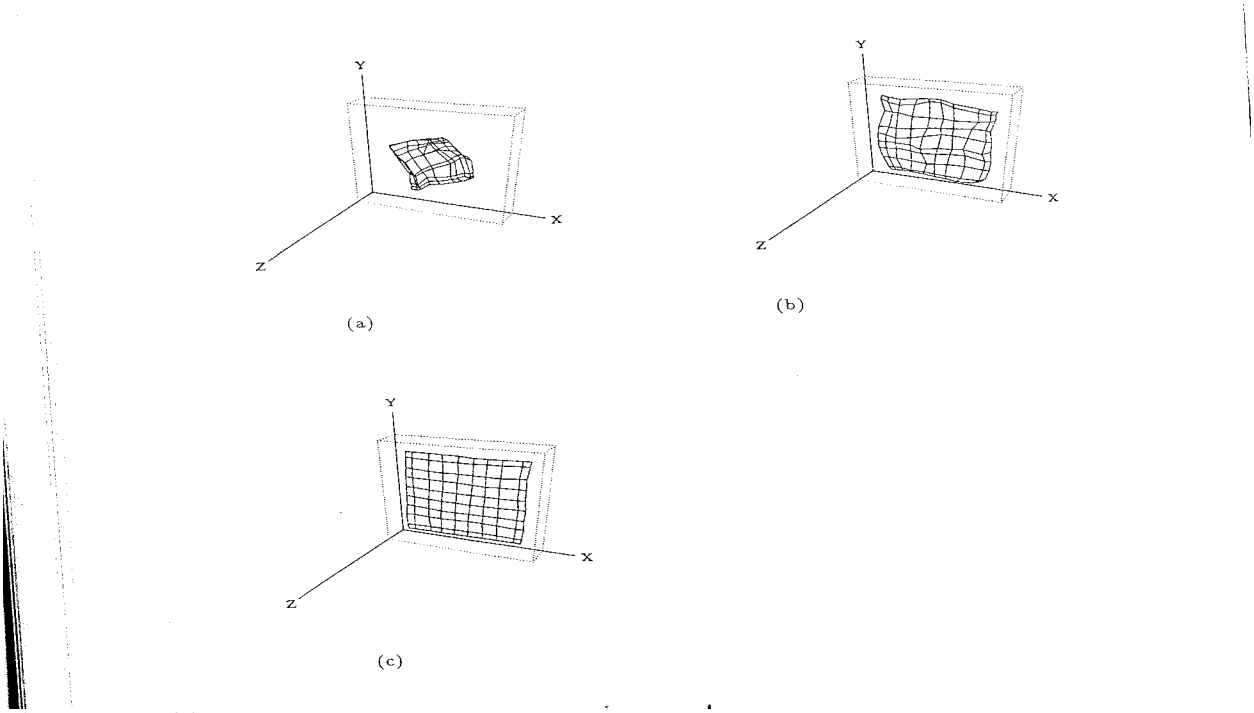


Figure 1: Problem 9.12

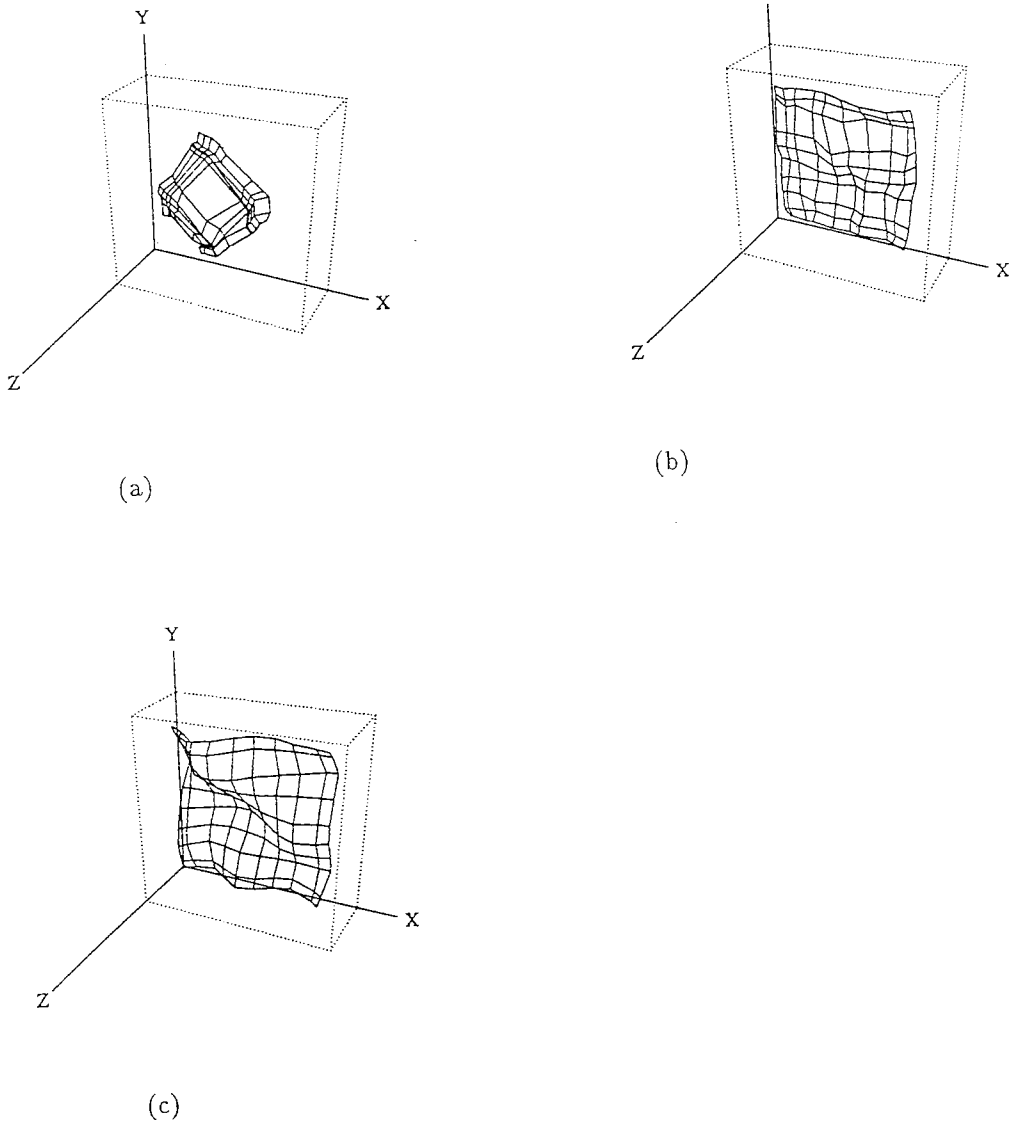


Figure 2: Problem 9.12

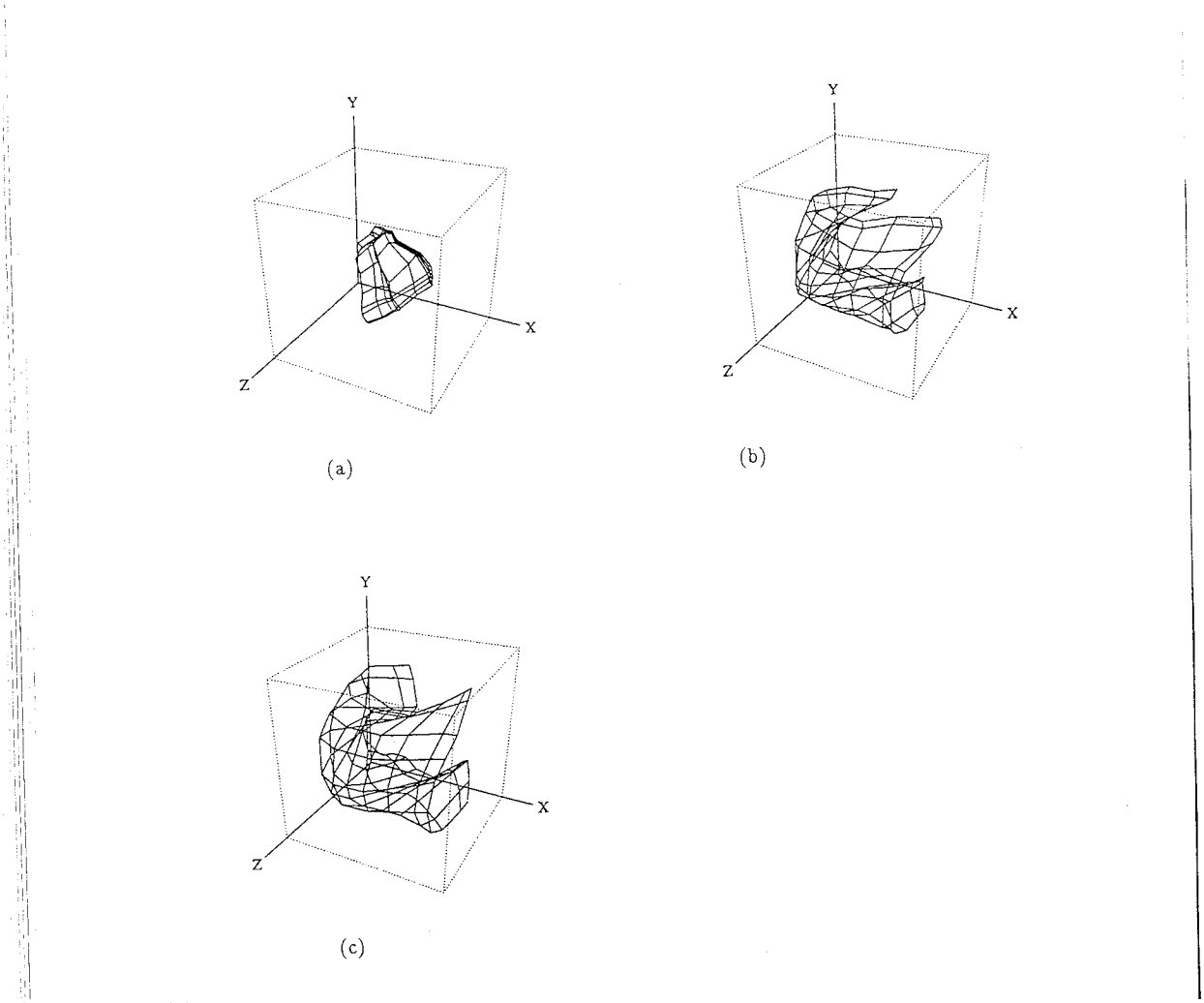


Figure 3: Problem 9.12

Problem 9.13

Typically, if the size of the neighborhood function around a winning neuron is shrunk very slowly, the map will become topologically ordered. On the other hand, if the neighborhood function is shrunk too rapidly, the map may not have time to become topologically ordered. In particular, shrinking the neighborhood function linearly in finite time will usually result in topological ordering. But, occasionally, we may end up with a "folded" map.

This phenomenon is illustrated in the map displayed in Fig. 1. In a rather loose sense, the formation of a folded map may be viewed as some form of local minimum of the ordering process experienced by the SOM algorithm.

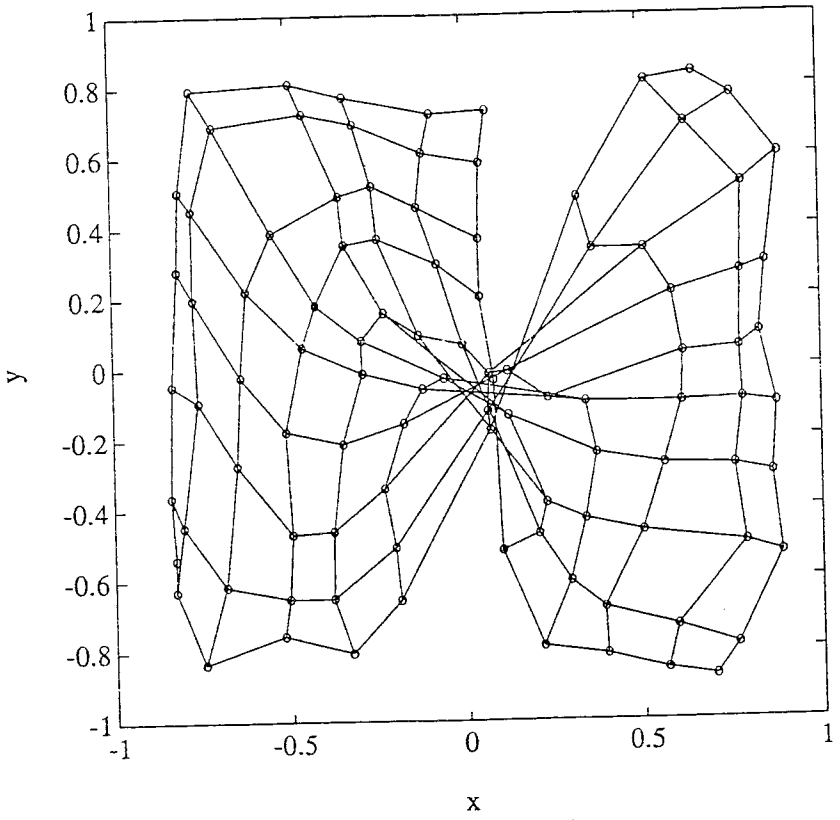


Figure 1: Problem 9.13

Problem 9.14

In this problem we explore a two-dimensional feature map trained with an input consisting of 4 Gaussian clouds in an eight-dimensional input space. The clouds have unit variance, and centered at (0, 0, ..., 0), (4, 0, 0, ..., 0), (4, 4, 0, ..., 0), and (0, 4, 0, ..., 0). They are assigned class labels 1, 2, 3, and 4, respectively, as shown in Fig. 1(a).

The SOM algorithm is trained with the input data, and each neuron is labelled with the class most represented by the input points around it. Figure 1(b) shows the labelled feature map formed by printing each neuron label at its position (row and column) in the map. The resulting labelled map captures the underlying planar topology of the input classes.

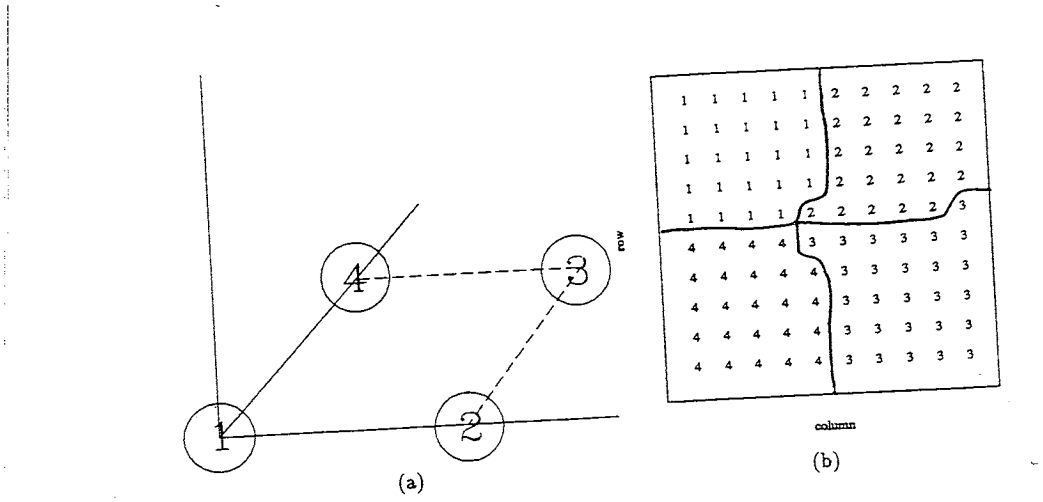


Figure 1: Problem 9.14

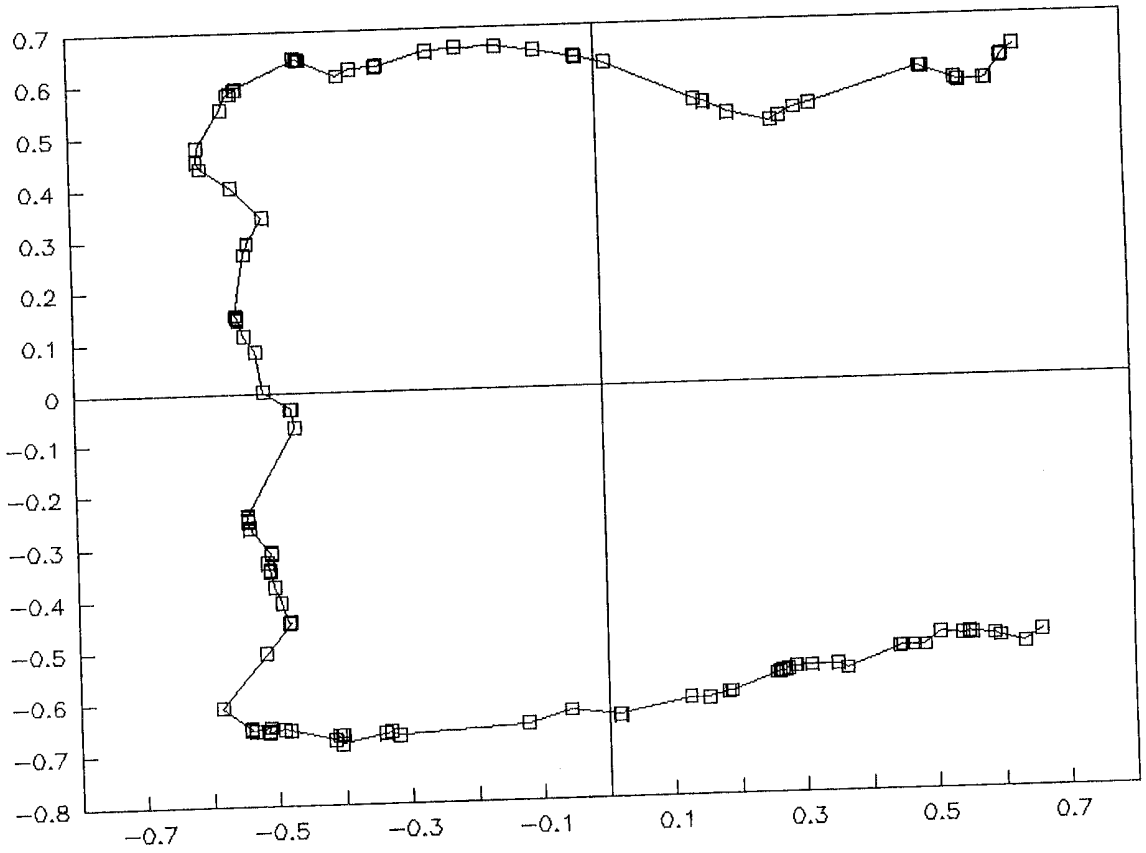
Problem 9.15

Parts (a) and (b) of Fig. 1 display the evolution of the standard SOM algorithm for a one-dimensional lattice of 257 neurons. The network was trained on a two-dimensional (square) distribution. The two parts of Fig. 1 are for 1,000 and 60,000 iterations of the SOM algorithm.

Figure 2 present the corresponding results obtained using the renormalized SOM algorithm. A total of 10 updates per codebook were used for this computation. Figure 2a shows the initial state of the algorithm. Figures 2b and 2c show the results obtained after 20 iterations of the algorithm, followed by a split. Figures 2d and 2e show the results after 50 iterations, followed by a split. Figures 2f and 2g show the results after 100 iterations, followed by a split. Figures 2h and 2i show the results after 190 iterations, followed by a split. Figures 2j and 2k show the results after 360 iterations, followed by a split. Figures 2l and 2m show the results after 690 iterations, followed by a split. Figures 2n and 2o show the results after 2630 iterations, followed by a split. Finally, Fig. 2p shows the feature map computed after 5000 iterations.

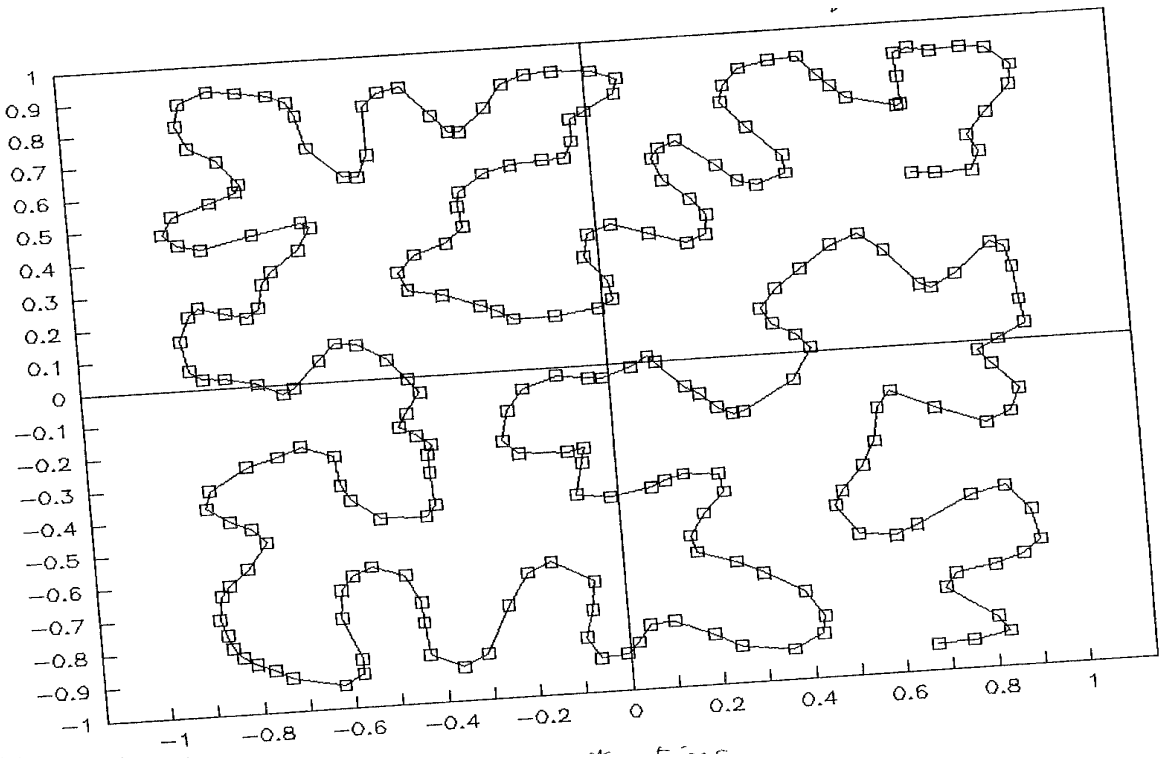
Comparing Figures 1b and 2p, we see that the renormalized SOM algorithm converges faster than the standard SOM algorithm.

The computational advantage of the renormalized SOM algorithm over the standard SOM algorithm increases with the size of the lattice. The reader is invited to repeat the experiment of Problem 9.15 for a larger lattice of 2049 neurons.



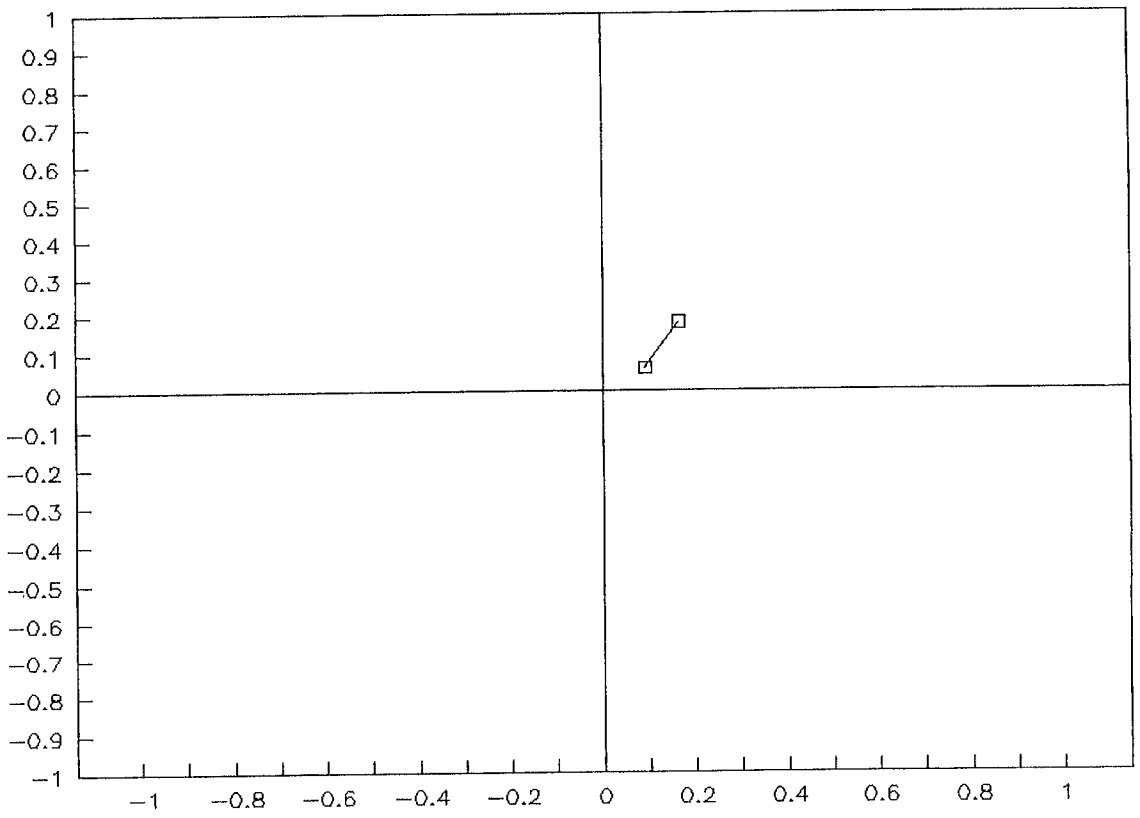
(a) 1,000 iterations

Figure 1a: Standard SOM Algorithm - Problem 9.15



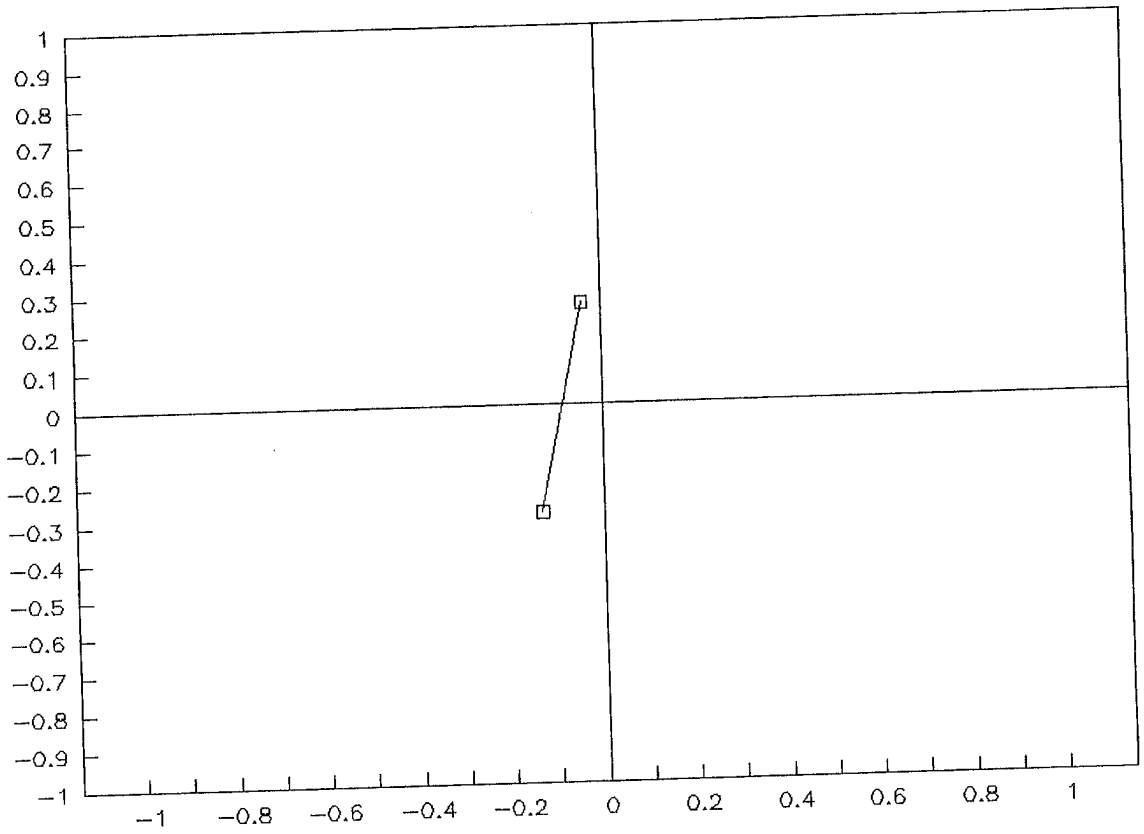
(6) 6,000 iterations

Figure 1b: Standard SOM Algorithm - Problem 9.15



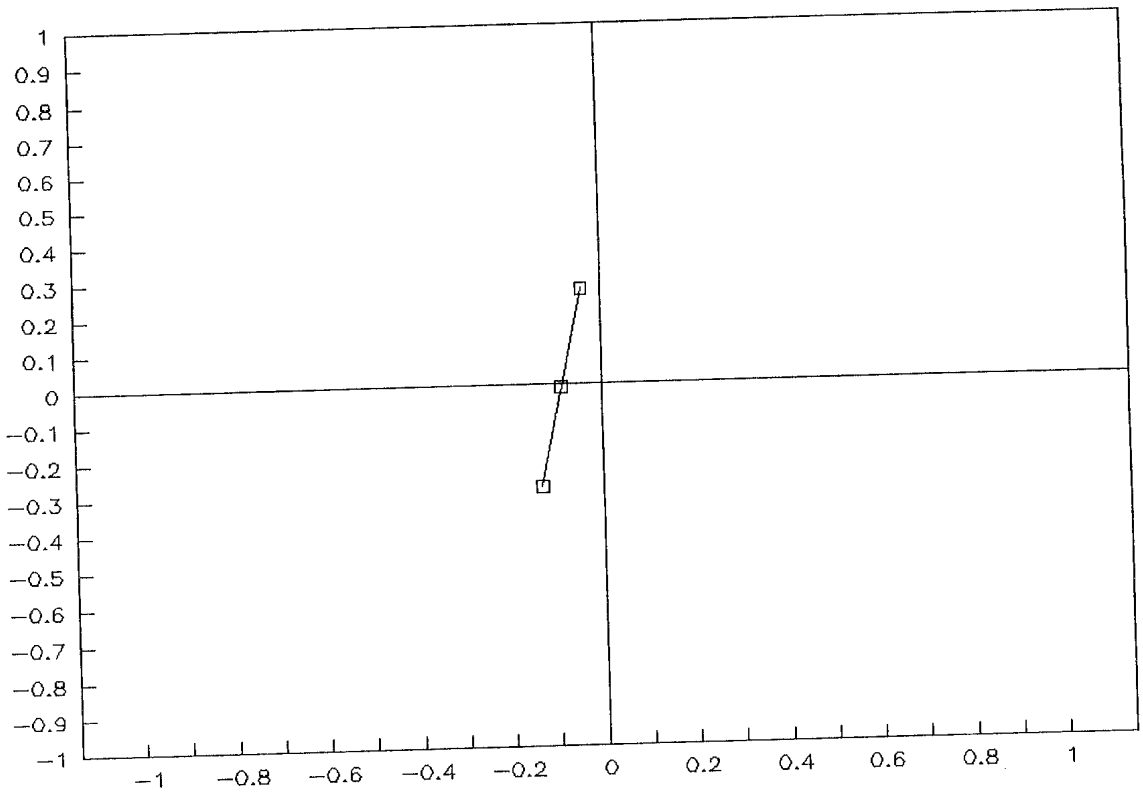
(a) Initial state

Figure 2a: Renormalized SOM algorithm - Problem 9.15



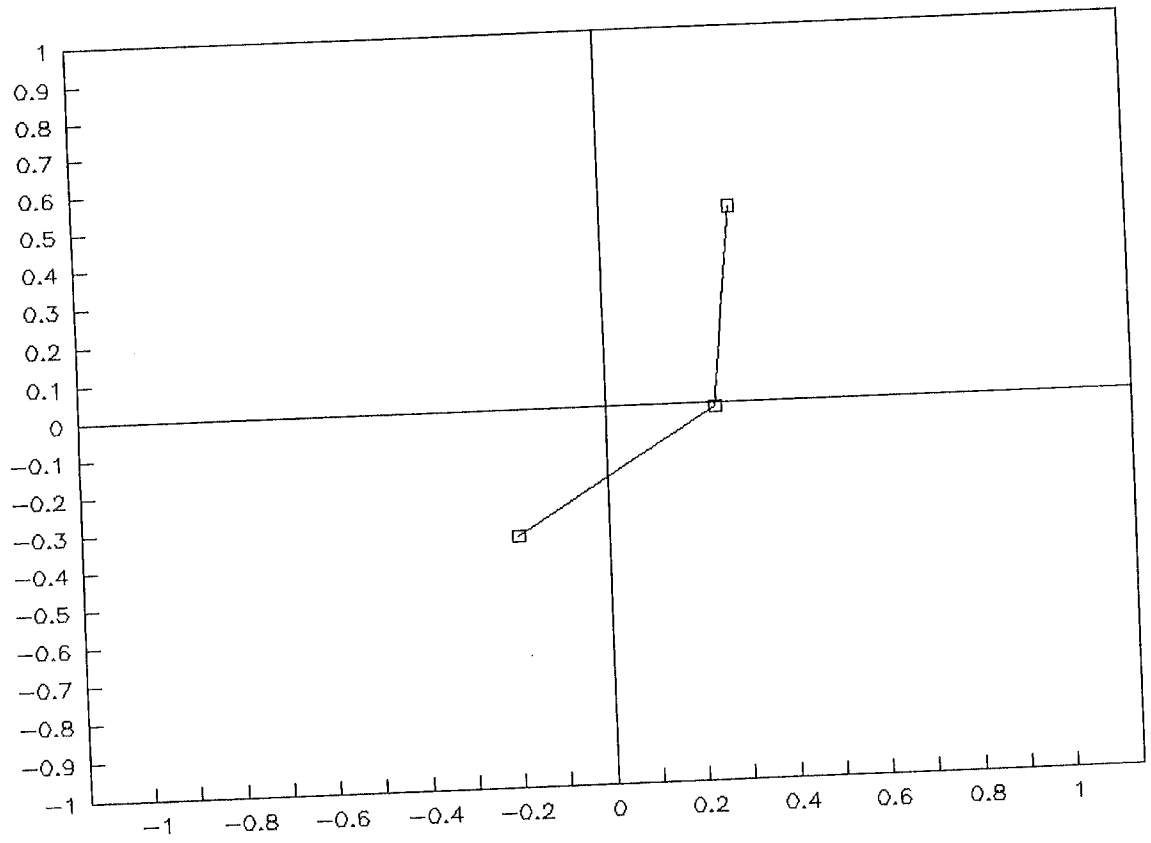
(b) 20 iterations

Figure 2b: Renormalized SOM algorithm - Problem 9.15



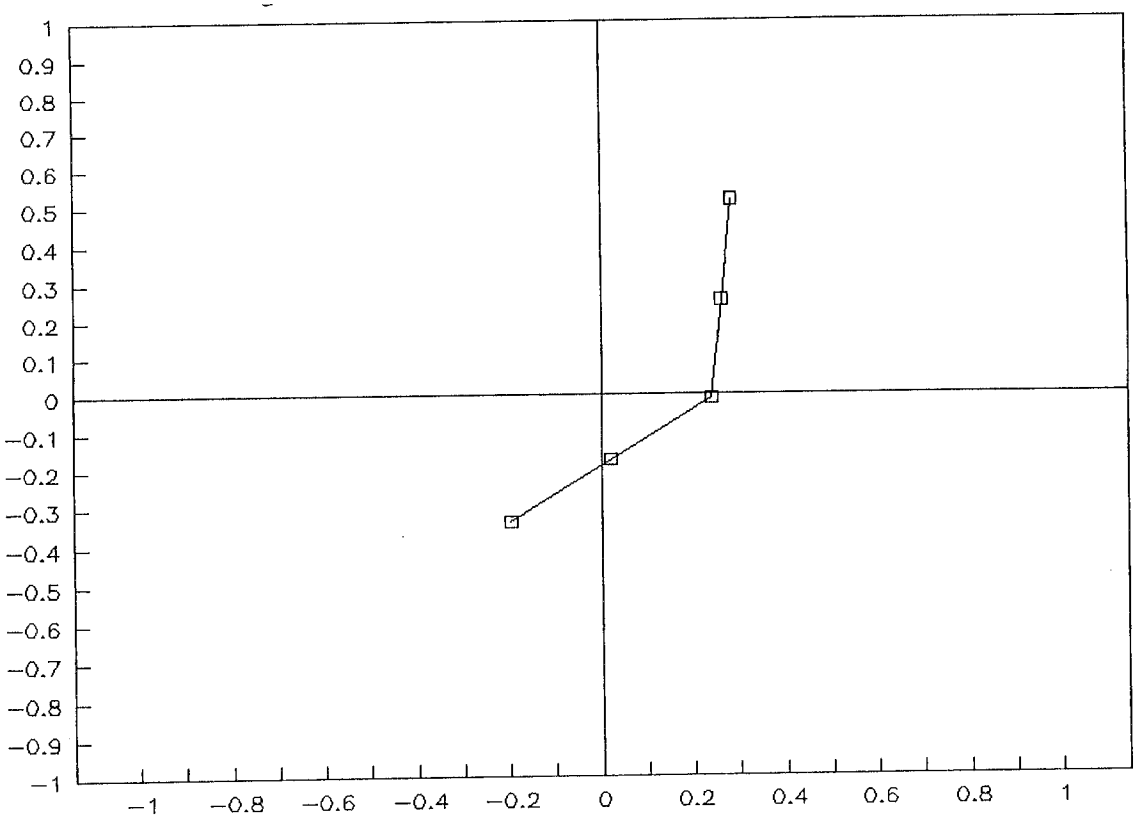
(c) 20 iterations - split

Figure 2c: Renormalized SOM algorithm - Problem 9.15



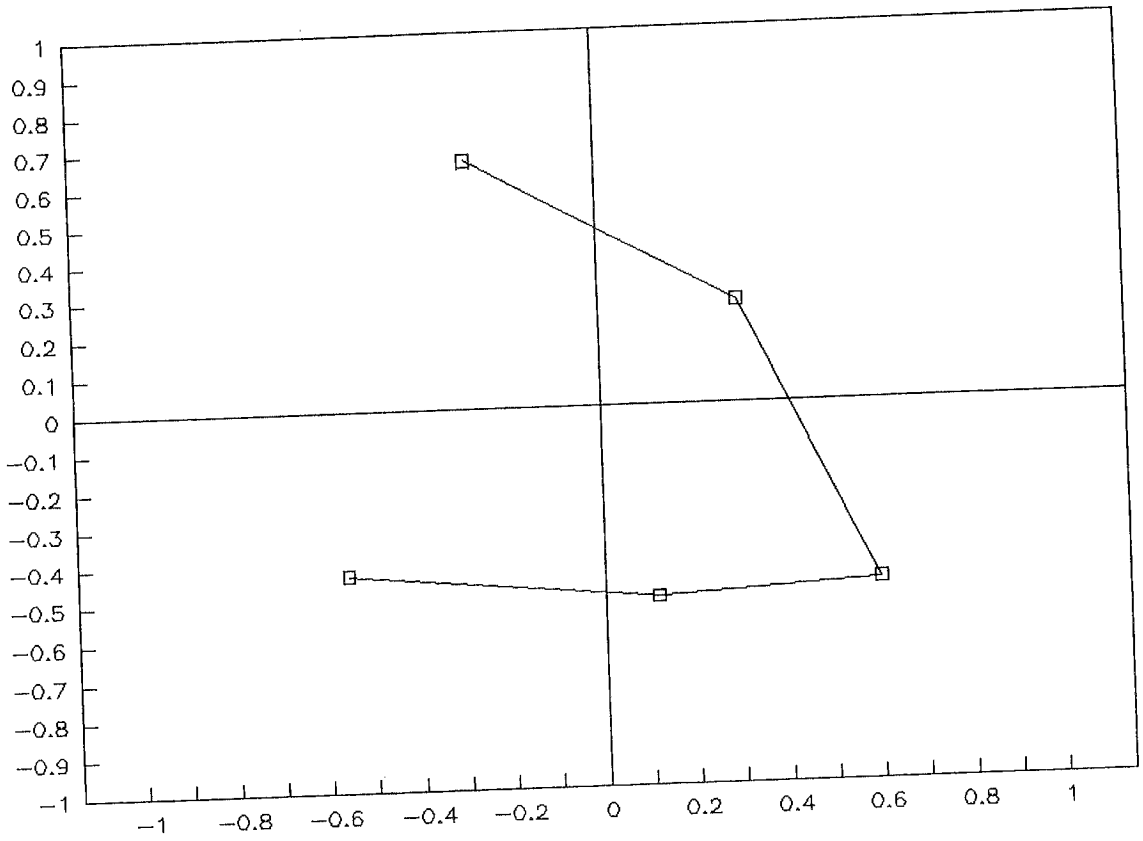
(d) 50 iterations

Figure 2d: Renormalized SOM algorithm - Problem 9.15



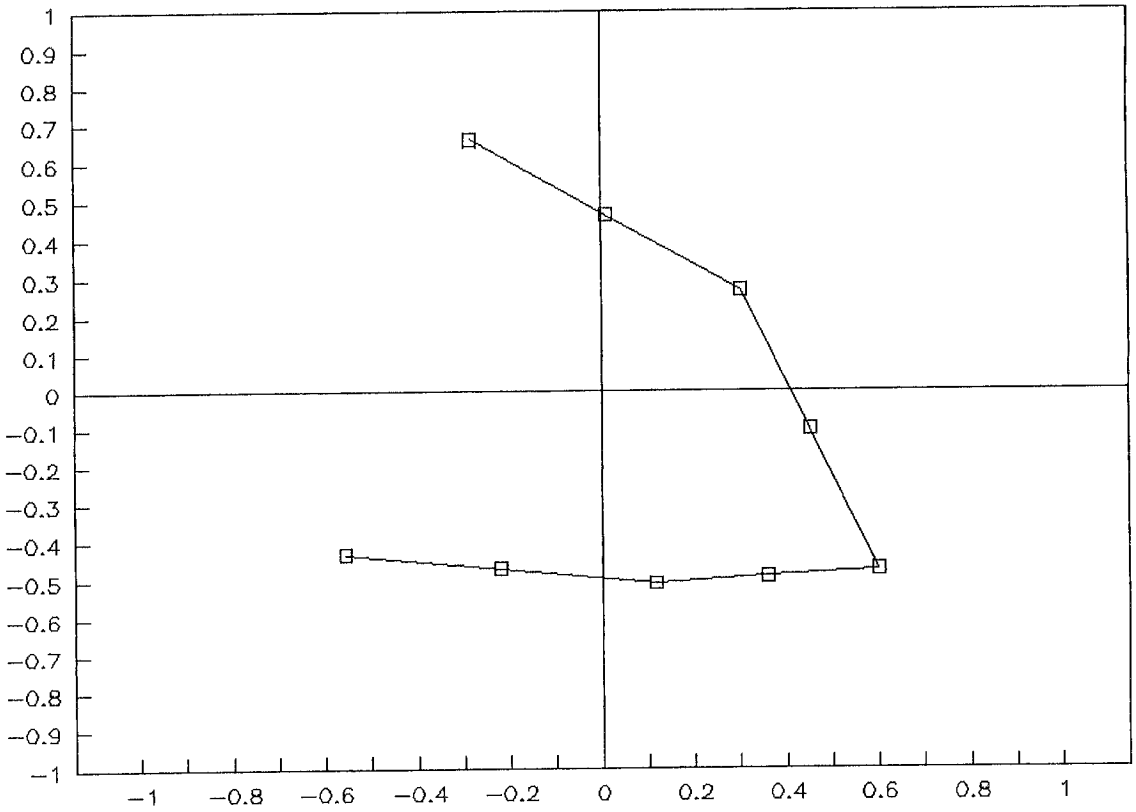
(e) 50 iterations - split

Figure 2e: Renormalized SOM algorithm - Problem 9.15



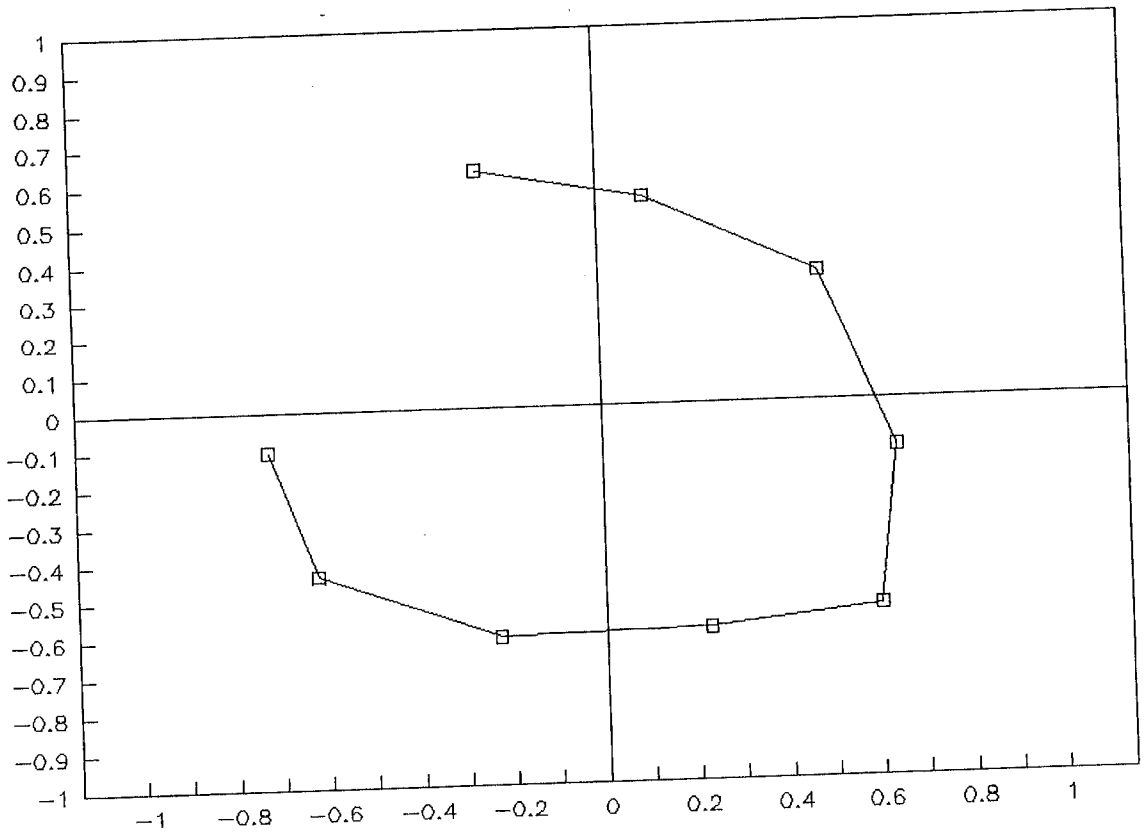
(f) 100 iterations

Figure 2f: Renormalized SOM algorithm - Problem 9.15



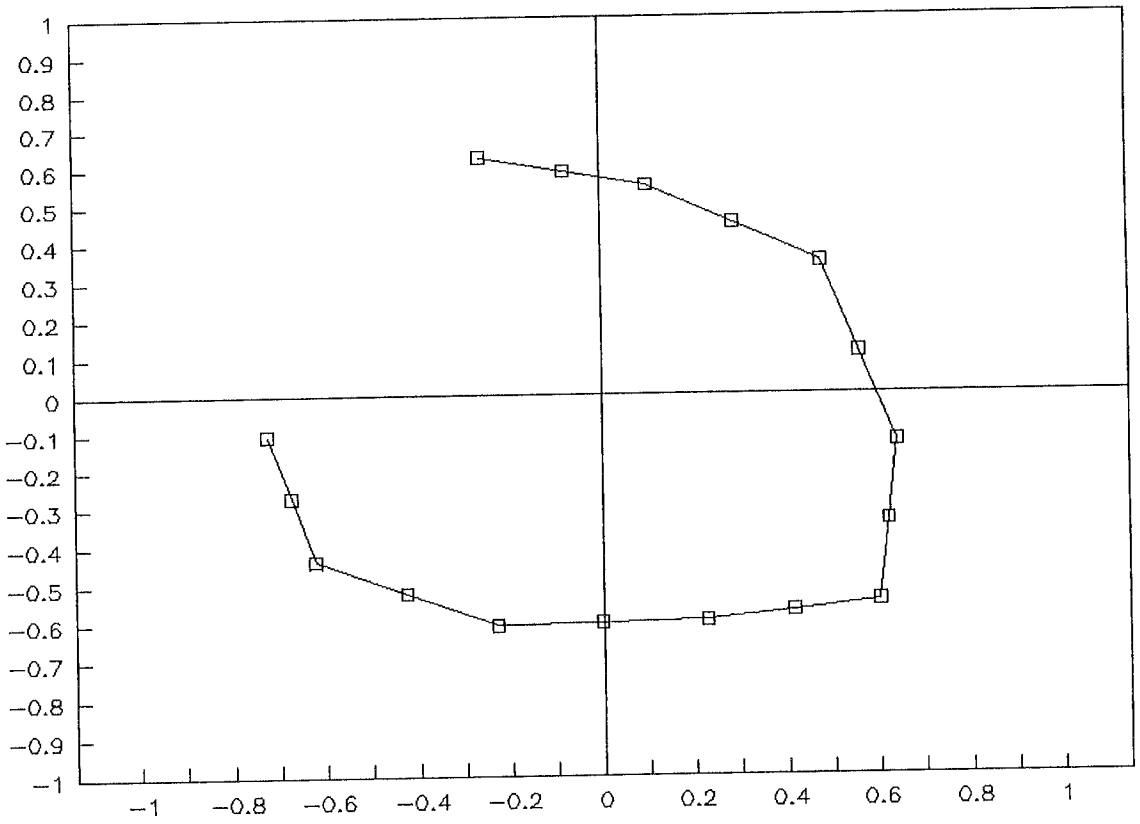
(g) 100 iterations - split

Figure 2g: Renormalized SOM algorithm - Problem 9.15

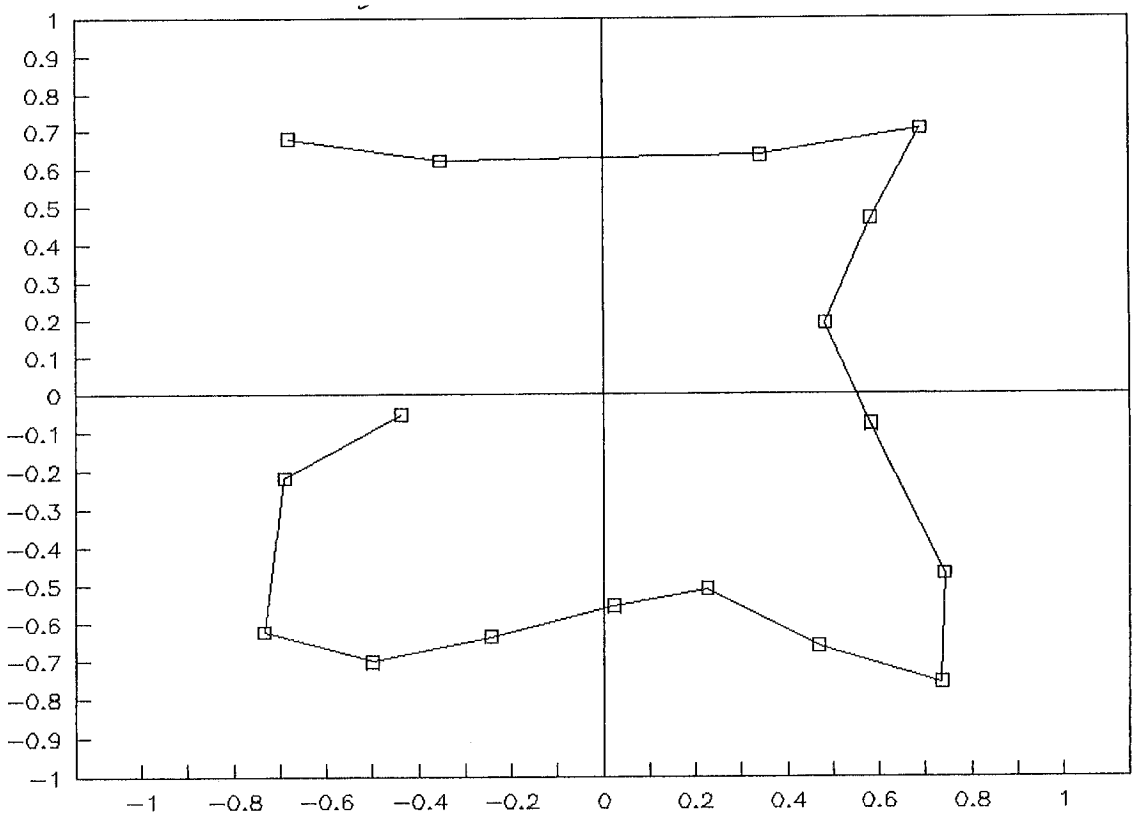


(h) 190 iterations

Figure 2h: Renormalized SOM algorithm - Problem 9.15

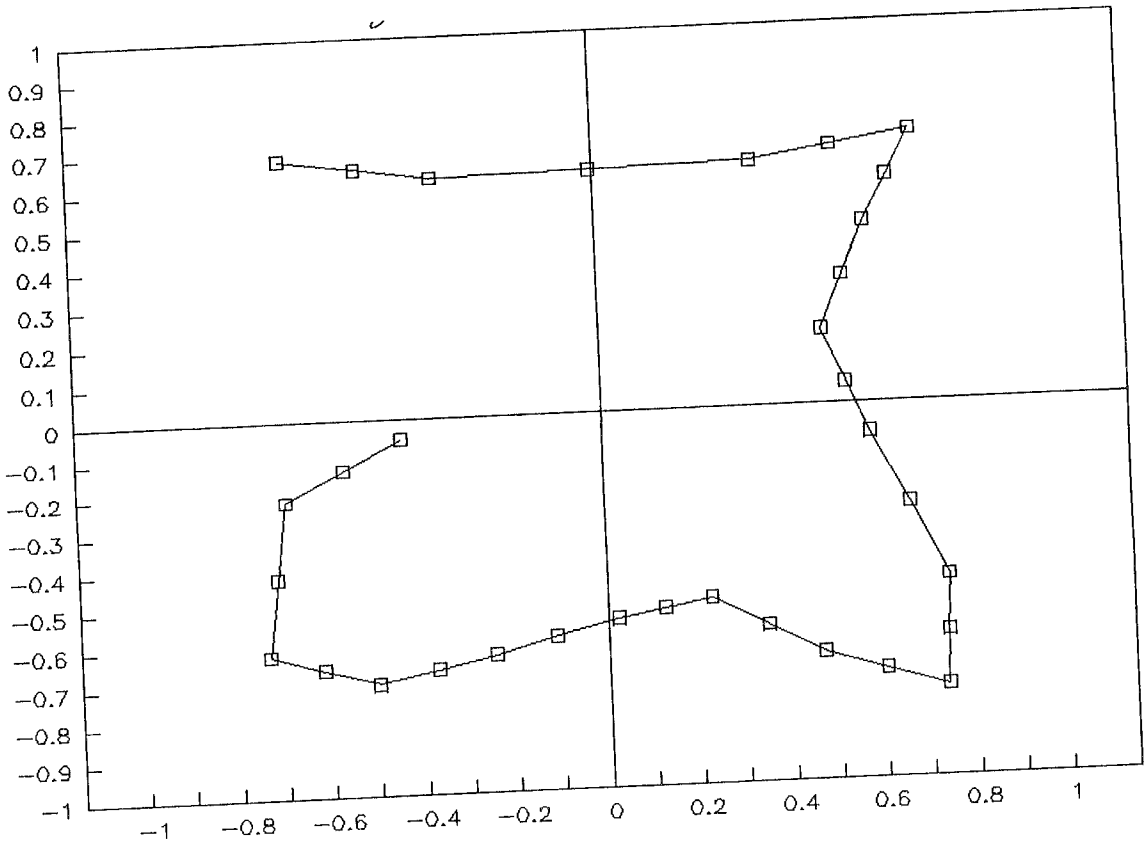


(i) 190 iterations - split
Figure 2(i) - Renormalized SOM algorithm - Problem 9.15



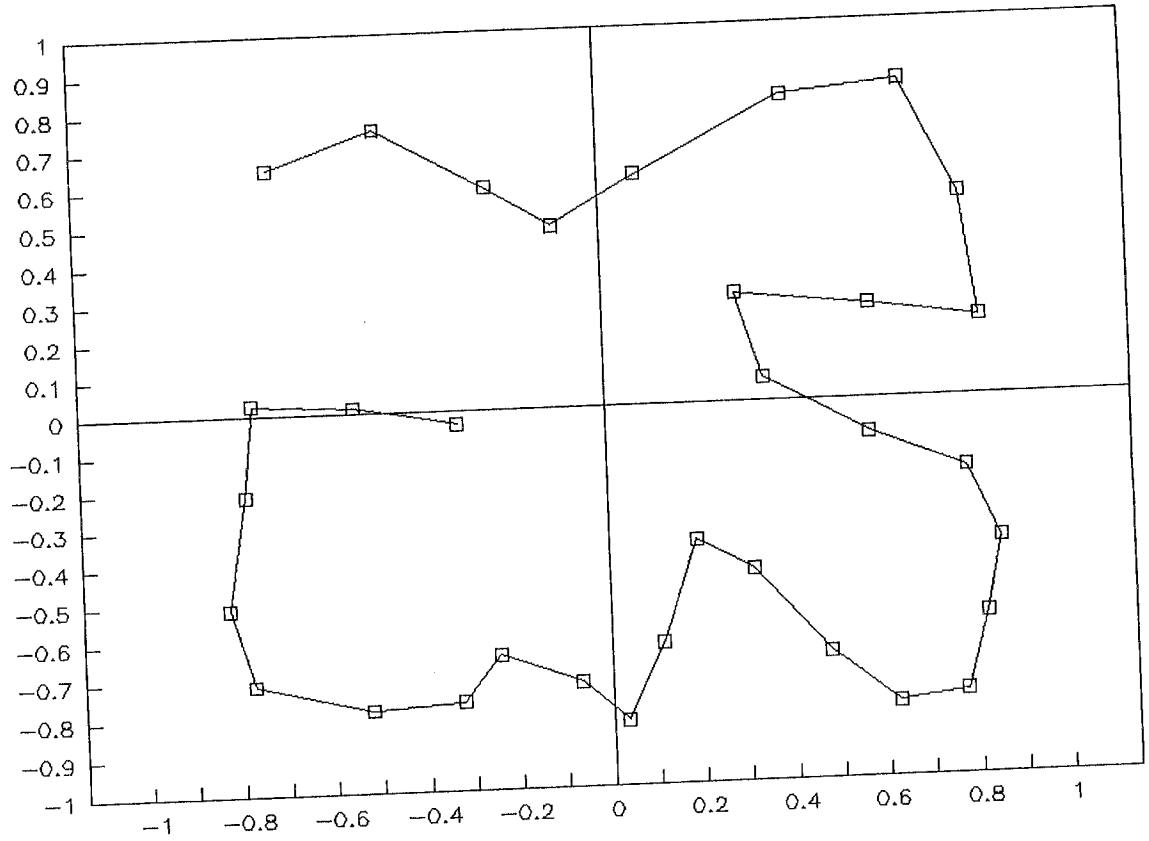
(j) 360 iterations

Figure 2j: Renormalized SOM algorithm - Problem 9.15

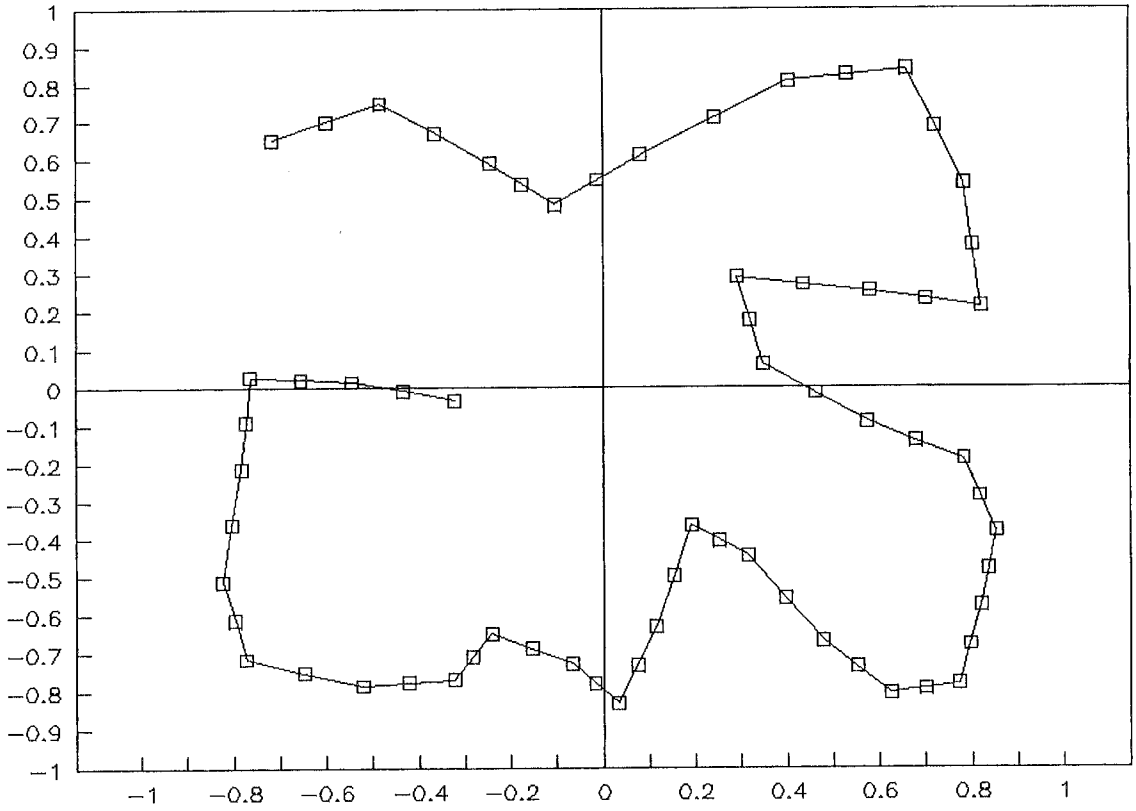


(k) 360 iterations

Figure 2k: Renormalized SOM algorithm - Problem 9.15

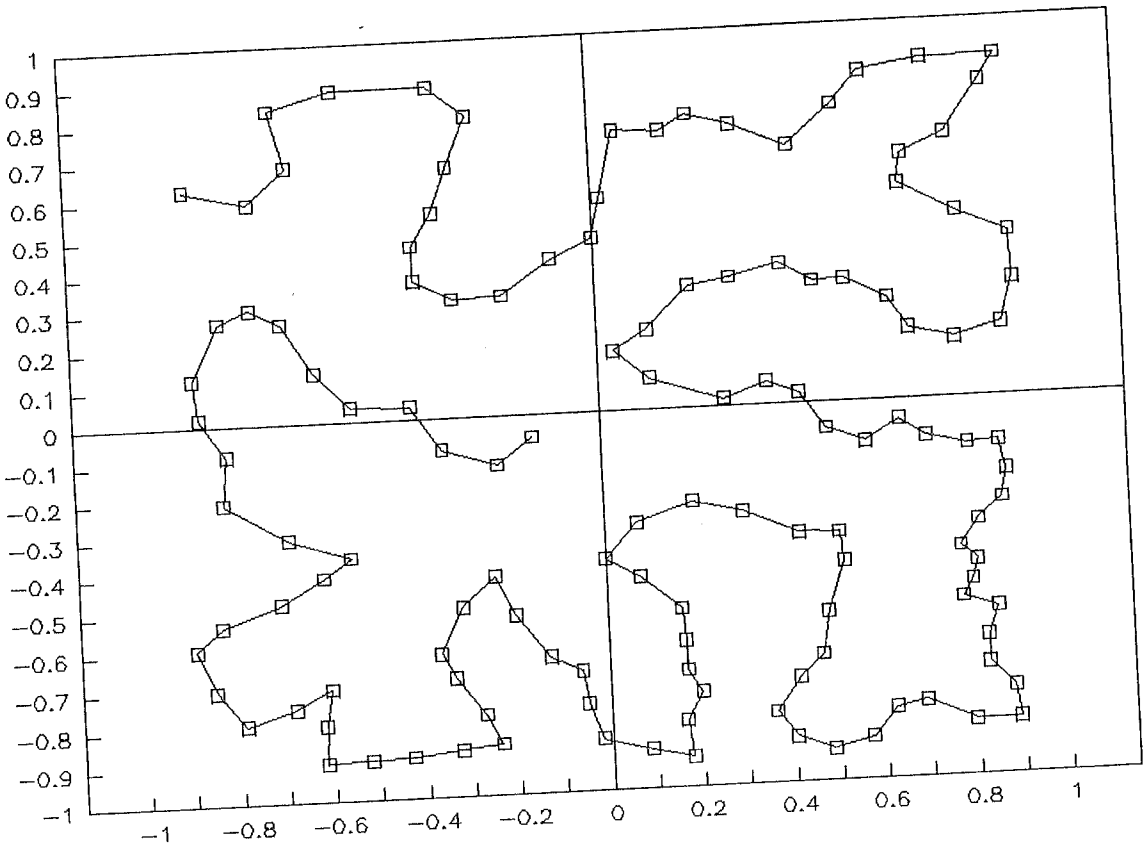


(l) 690 iterations
Figure 2l: Renormalized SOM algorithm - Problem 9.15



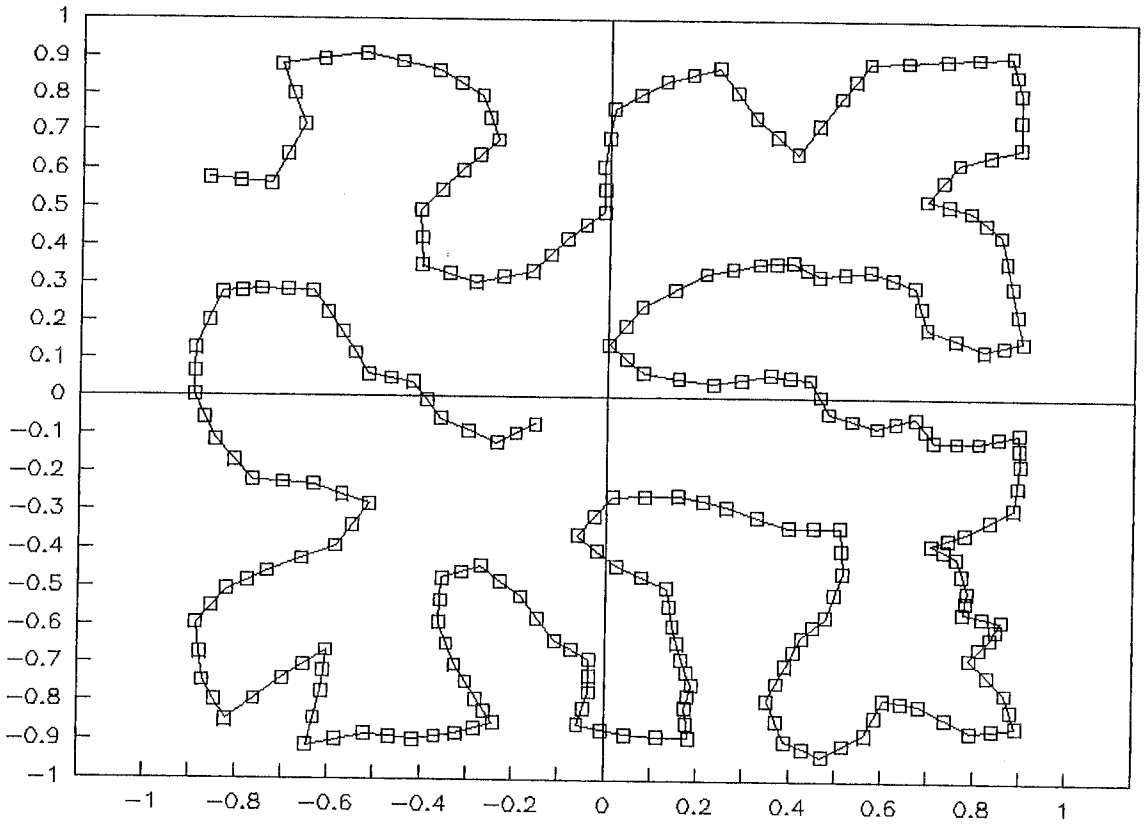
(m) 690 iterations - split

Figure 2m: Renormalized SOM algorithm - Problem 9.15



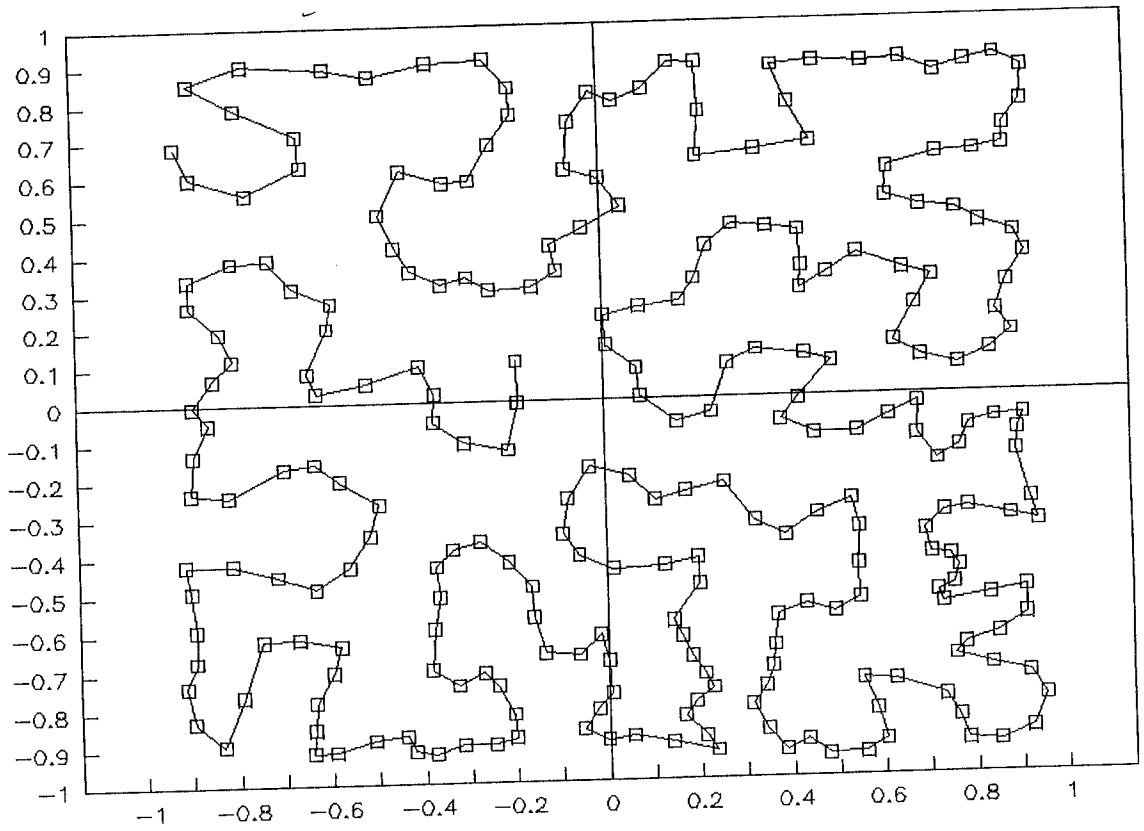
(n) 2,630 iterations

Figure 2n: Renormalized SOM algorithm - Problem 9.15



(o) 2,630 iterations - split

Figure 2o: Renormalized SOM algorithm - Problem 9.15



(p) 5,000 iterations

Figure 2p: Renormalized SOM algorithm - Problem 9.15

Problem 9.16

For the evaluation of the SOM algorithm, a square neighborhood function is used, which is relatively simple to implement. The map is configured with 8 source nodes and a one-dimensional lattice of 64 neurons. The algorithm is presented with data points selected from the pulse-amplitude modulated (PAM) waveform embedded in white Gaussian noise for 3 different signal-to-noise ratio.

Each of the 3 maps (for SNR = 10, 20, 30 dB) were trained identically. The initial learning-rate parameter was set equal to 0.8 and then decreased linearly by 0.015 after every 1000 input patterns. The rate was limited at the lower level by a value of 0.1. When the number of input patterns presented to the algorithm reached 70,000, the rate was fixed at 0.01, and the algorithm was presented with 30,000 new input patterns. Thus the total number of iterations of the algorithm was 10^5, which means that each of the 8 possible signal levels was presented approximately 12,5000 times; this yields approximately 2000 input patterns per neuron.

The initial value defining the radius of the neighborhood function was set at 64; it was also decreased linearly (by 1) after every 1000 input patterns until it eventually reached a radius of zero, at which point it was fixed for the duration of the training sequence. The large initial value of the radius of the neighborhood function permits for all the neurons to participate initially in the learning process regardless of the physical location of the winning neuron.

During the learning process, the weight vectors were saved at various stages of the map formation. Snapshots of the weight vectors were recorded for 5,000, 10,000, 25,000, 50,000, 75,000 and 100,000 input patterns. After each map had organized itself, 8 further input patterns were presented to the maps. These patterns represented the 8 possible signal levels of the noisy PAM signal. The Euclidean distance for each neuron was calculated and plotted for each of the 8 patterns, thus illustrating the clustering of output neurons and topological ordering property of the self-organizing feature map.

Figures 1a through 1g illustrate the formation of the weight vectors for the SOM algorithm for an SNR = 30 dB. In each figure the horizontal axis represents the 64 neurons in the output layer of the network. The vertical axis represents the values of the weight vectors associated with each neuron. Figure 1a displays the initial randomly distributed weight vectors. Figures 1b through 1g demonstrate the formation of the weight vectors at various stages of the learning process. Figure 1g illustrates the final weight vector values associated with each neuron.

As the learning process progresses, the weight vector plots begin to mimic the topology of the input data. After the learning was completed, the map is clearly formed into 8 distinct regions corresponding to the 8 possible signal levels of the PAM signal.

Figures 2a through 2g present the corresponding result for SNR = 20 dB. Finally, Figs. 3a through 3g present the results for SNR = 10 dB.

The results presented in Figs. 1 through 3 show that each of the feature maps is capable of learning the underlying patterns of the input data embedded in noise. As the SNR of the input patterns is reduced, the organization of the map into distinct clusters is less precise, but nevertheless still intelligible. The experiments presented here demonstrate the following:

- The SOM algorithm is able to learn the underlying distribution of the input data in an unsupervised manner.
- The algorithm preserves topological neighborhood of the input.

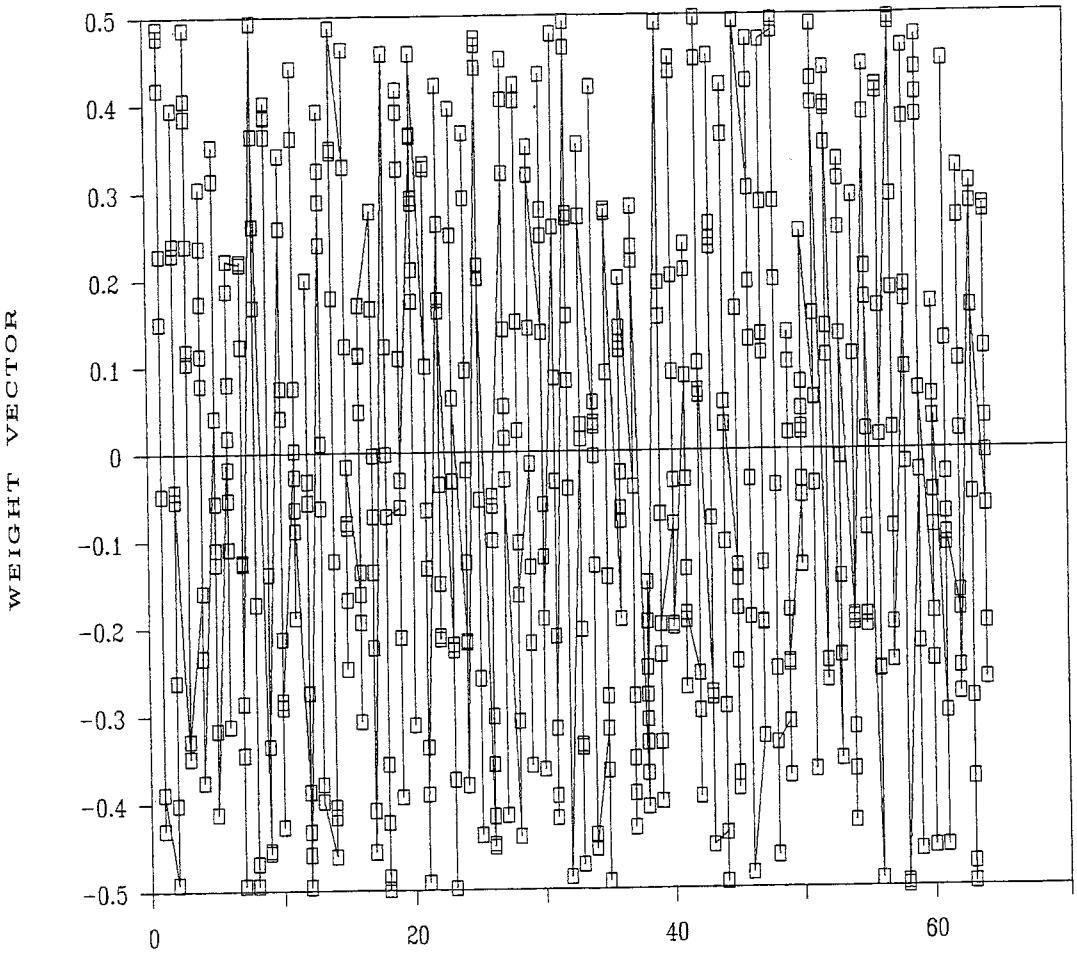


Figure 1a: Problem 9.16
Initial map for SNR = 30 dB

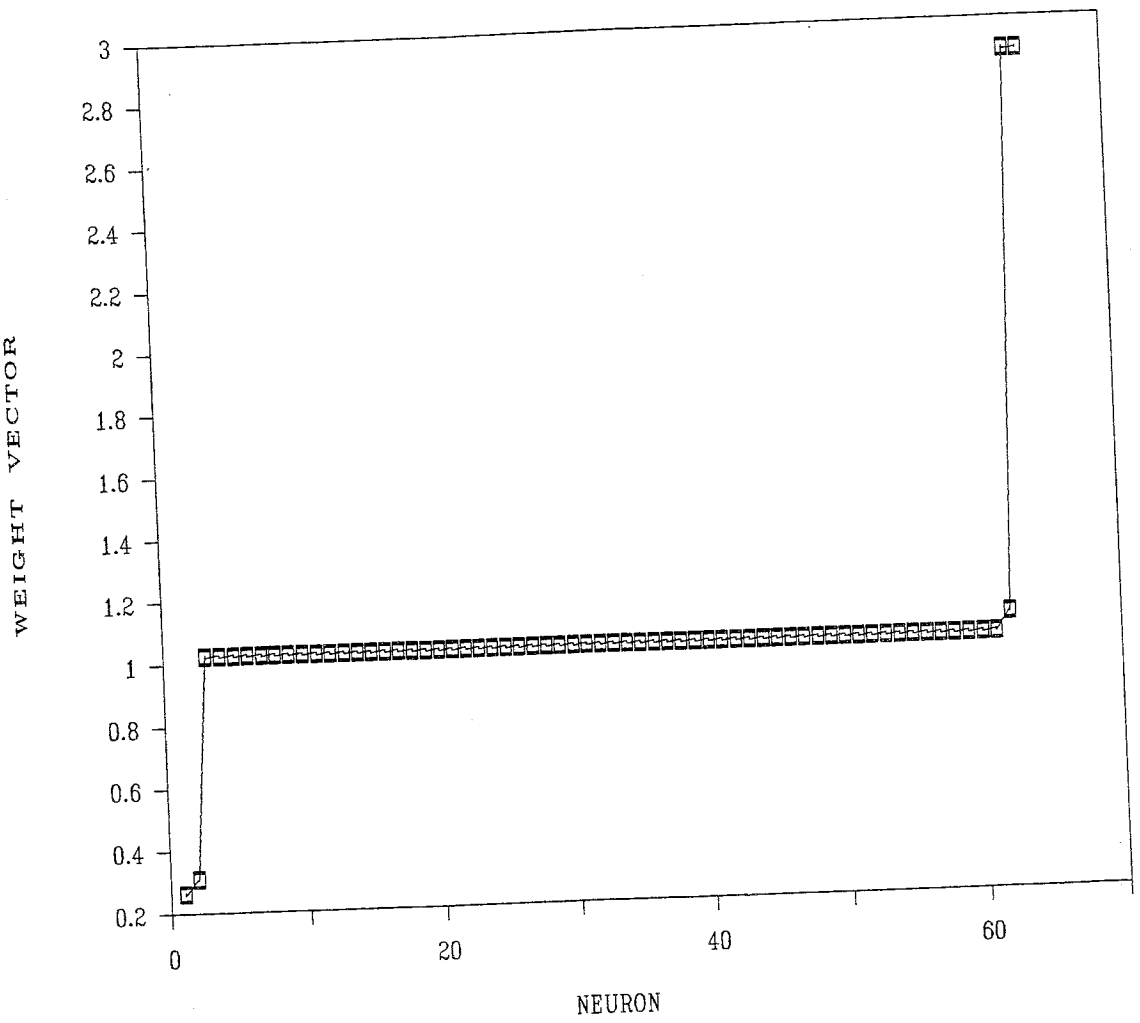


Figure 1b: Problem 9.16
SNR = 30 dB, map after 5,000 patterns

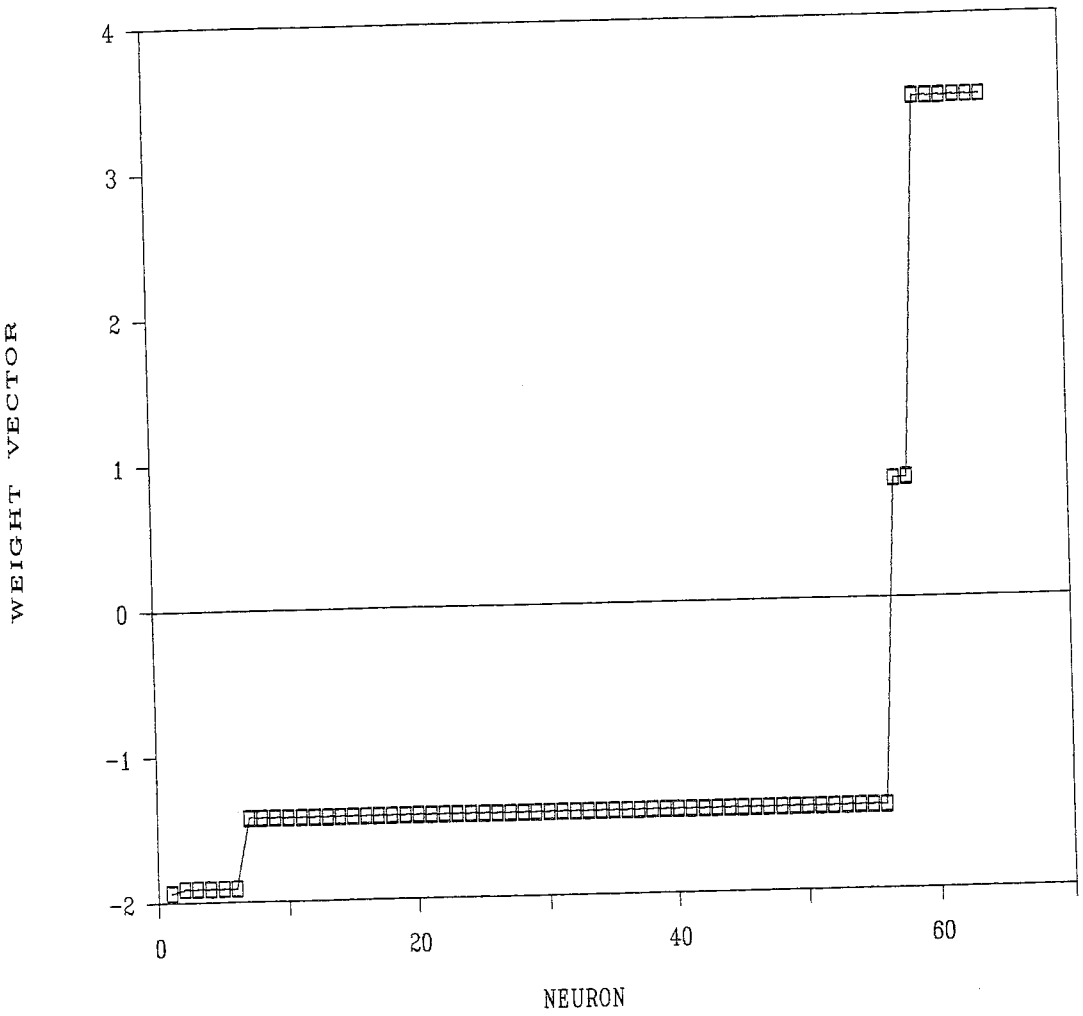


Figure 1c: Problem 9.16
SNR = 30 dB; map after 10,000 patterns

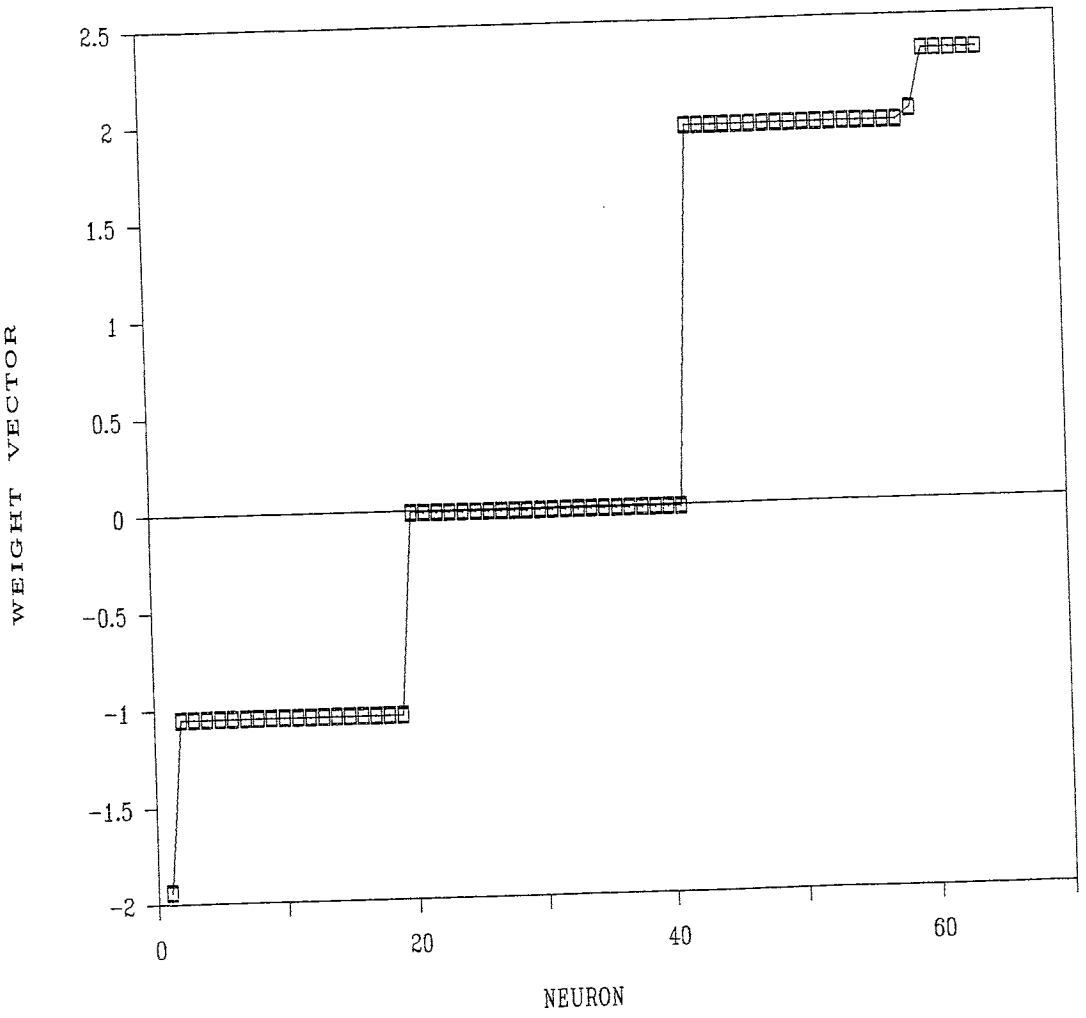


Figure 1d: Problem 9.16
SNR = 30 dB; map after 25,000 patterns

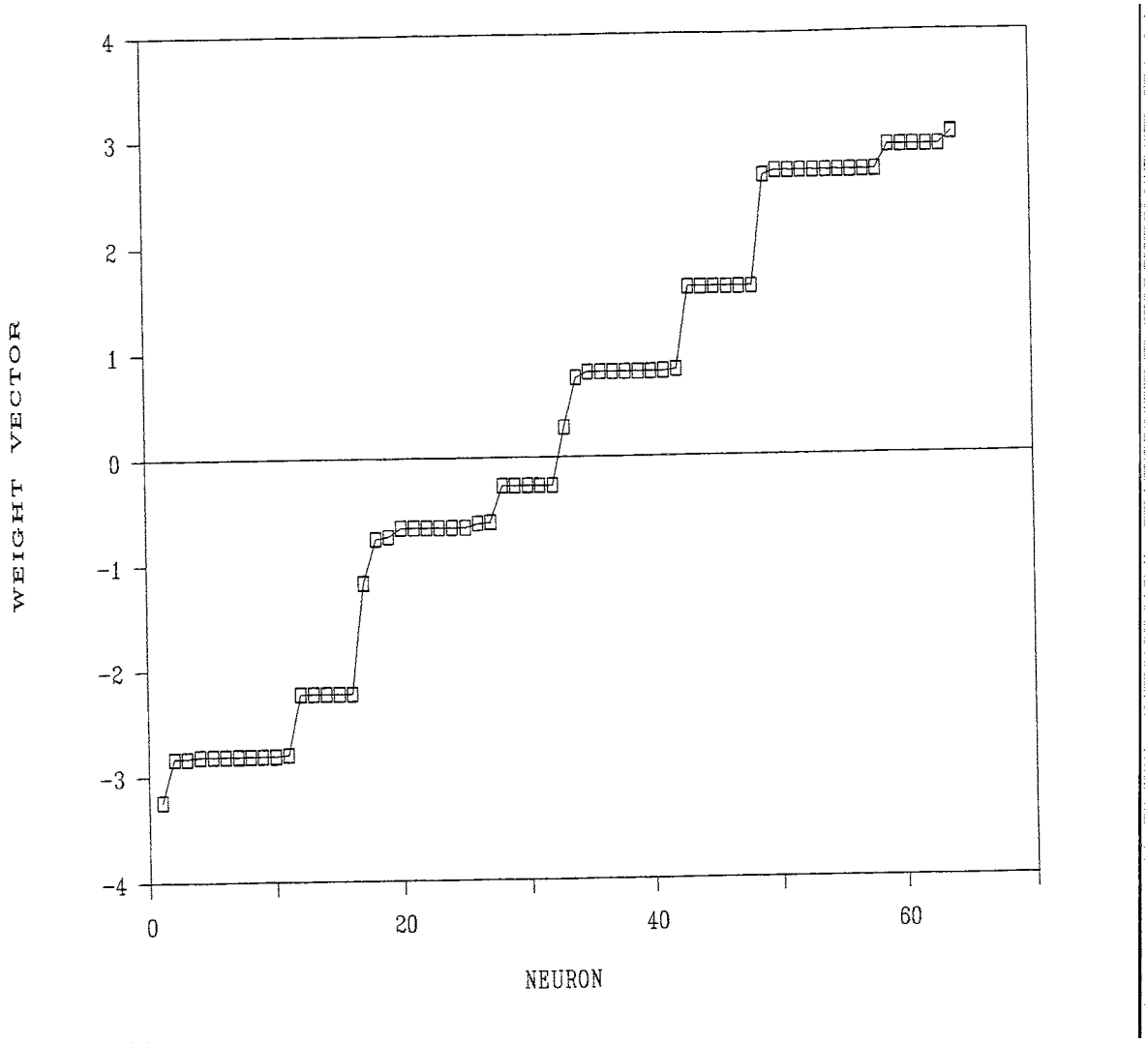


Figure 1e: Problem 9.16
SNR = 30 dB; map after 50,000 patterns

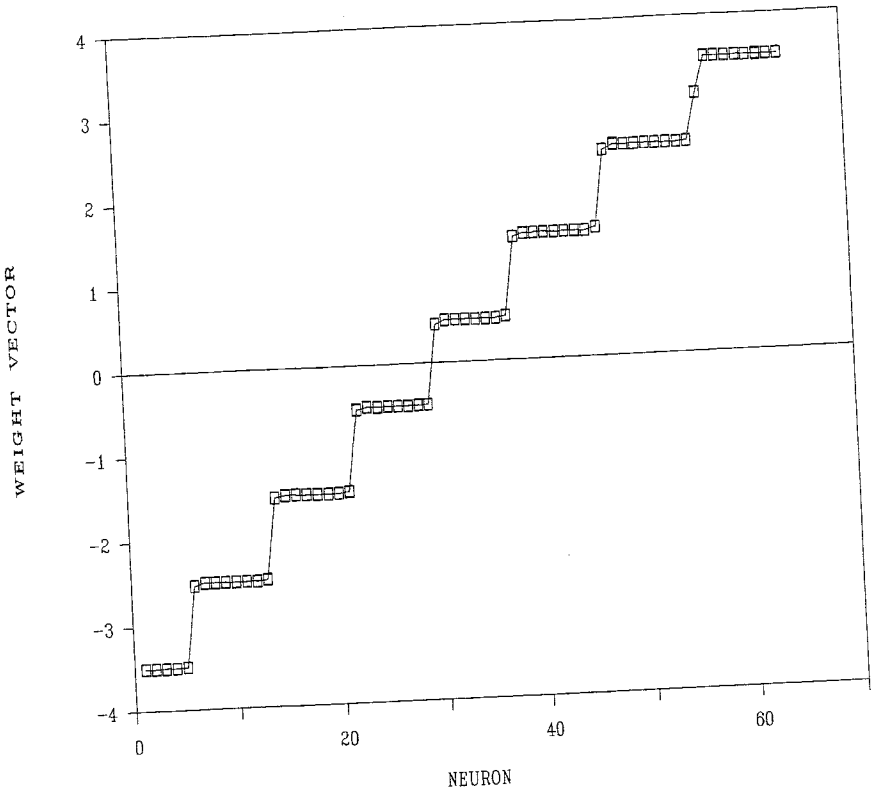


Figure 1f: Problem 9.16
SNR = 30 dB; map after 75,000 patterns

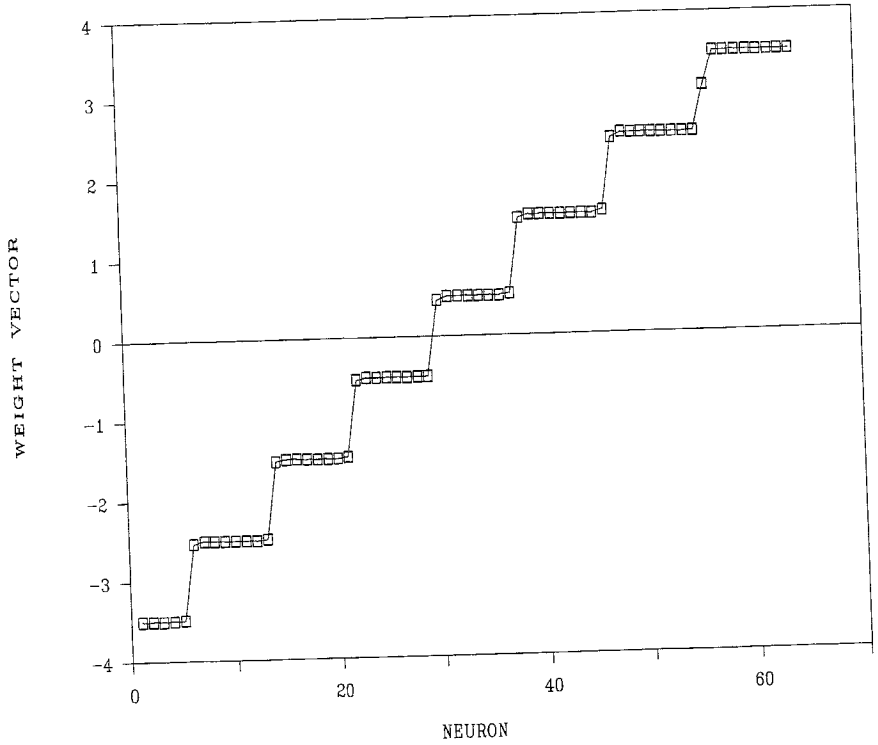


Figure 1g: Problem 9.16
SNR = 30 dB; map after 100,000 patterns

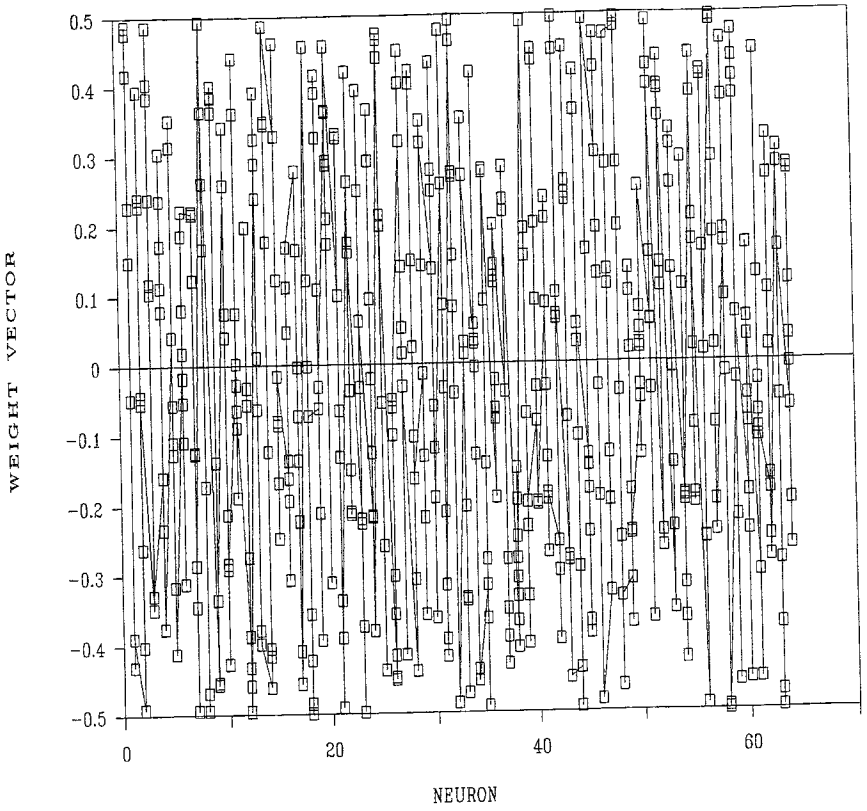


Figure 2a: Problem 9.16
Initial map for SNR = 20 dB

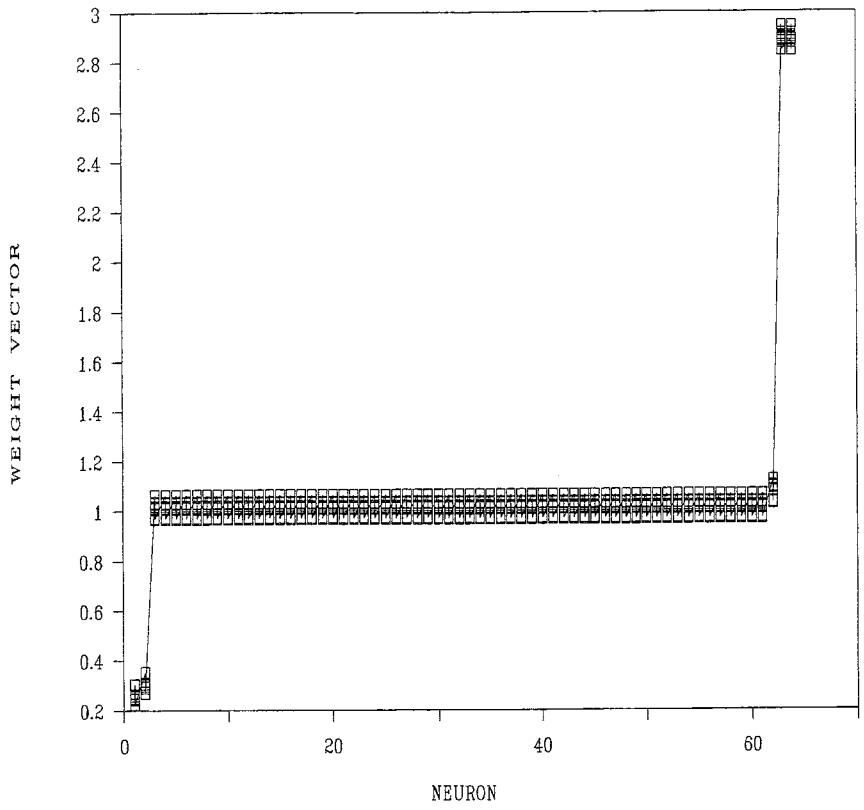


Figure 2b: Problem 9.16
SNR = 20 dB; map after 5,000 patterns

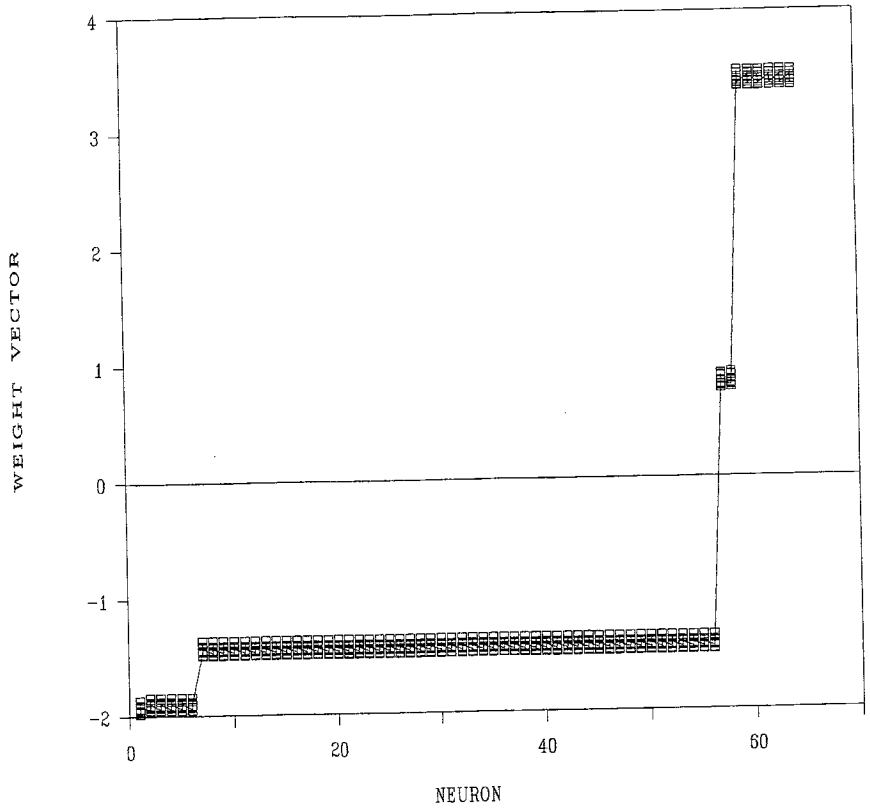


Figure 2c: Problem 9.16
SNR = 20 dB; map after 10,000 patterns

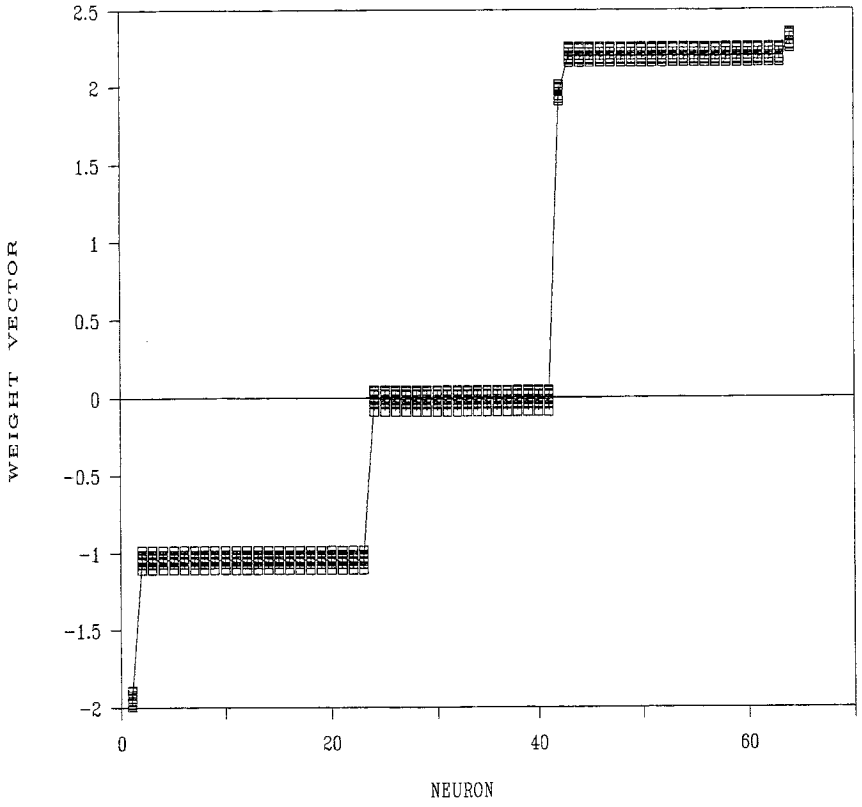


Figure 2d: Problem 9.16
SNR = 20 dB; map after 25,000 patterns

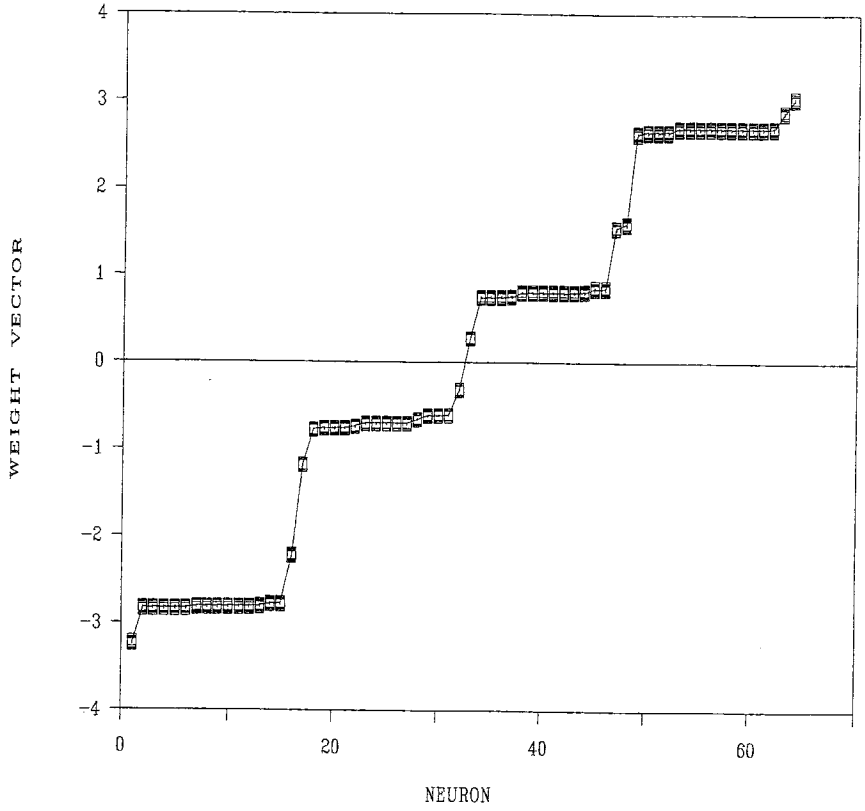


Figure 2e: Problem 9.16
SNR = 20 dB; map after 50,000 patterns

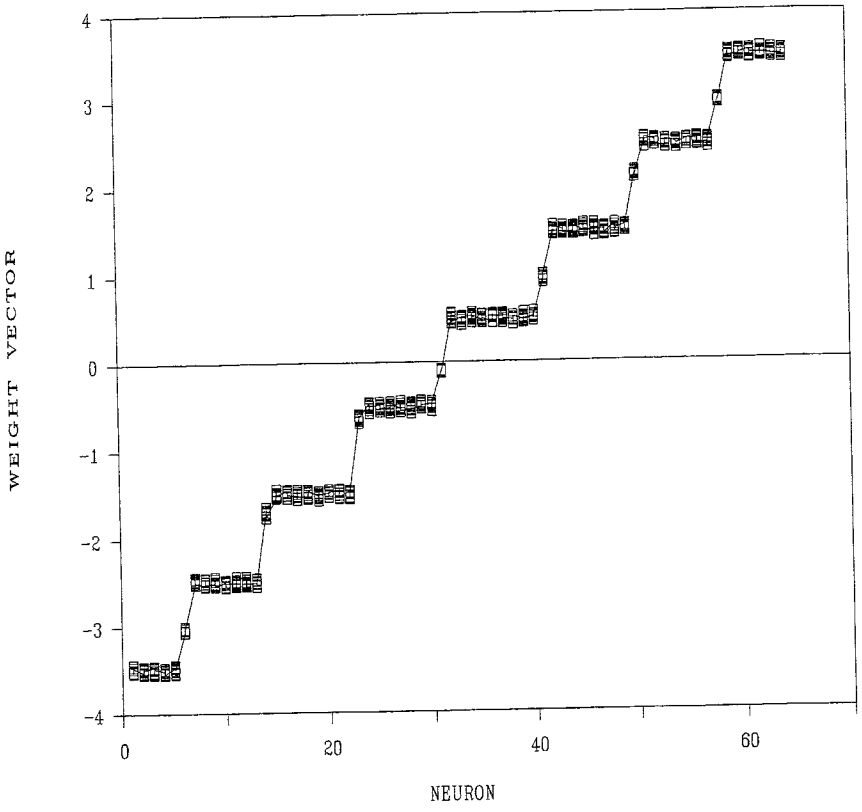


Figure 2f: Problem 9.16
SNR = 20 dB; map after 75,000 patterns

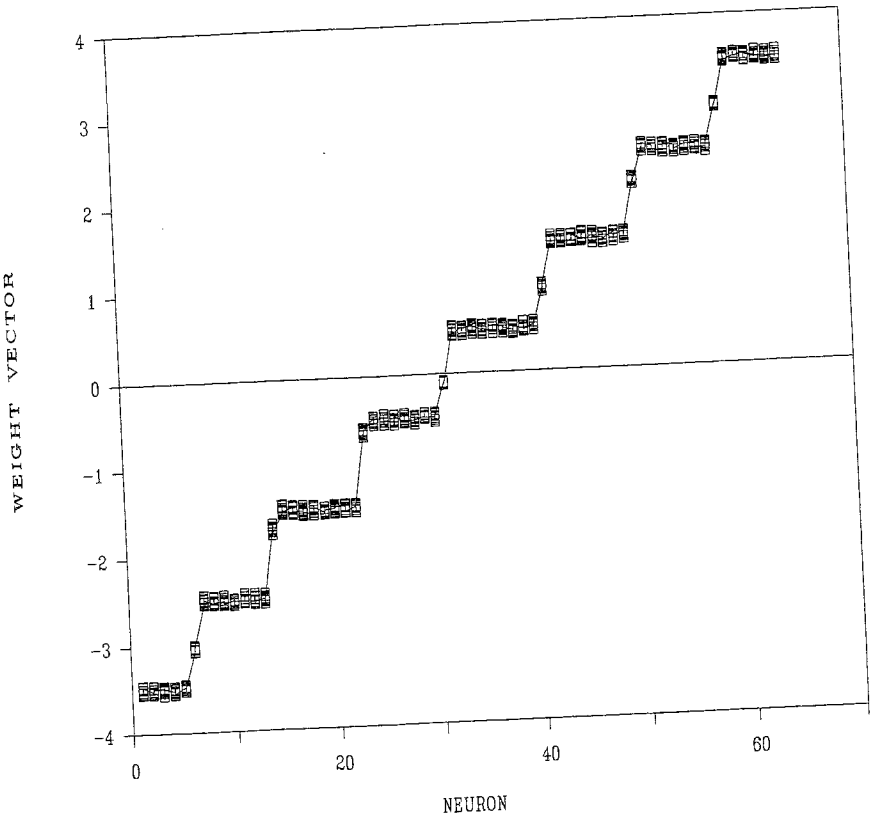


Figure 2g: Problem 9.16
SNR = 20 dB; map after 100,000 patterns

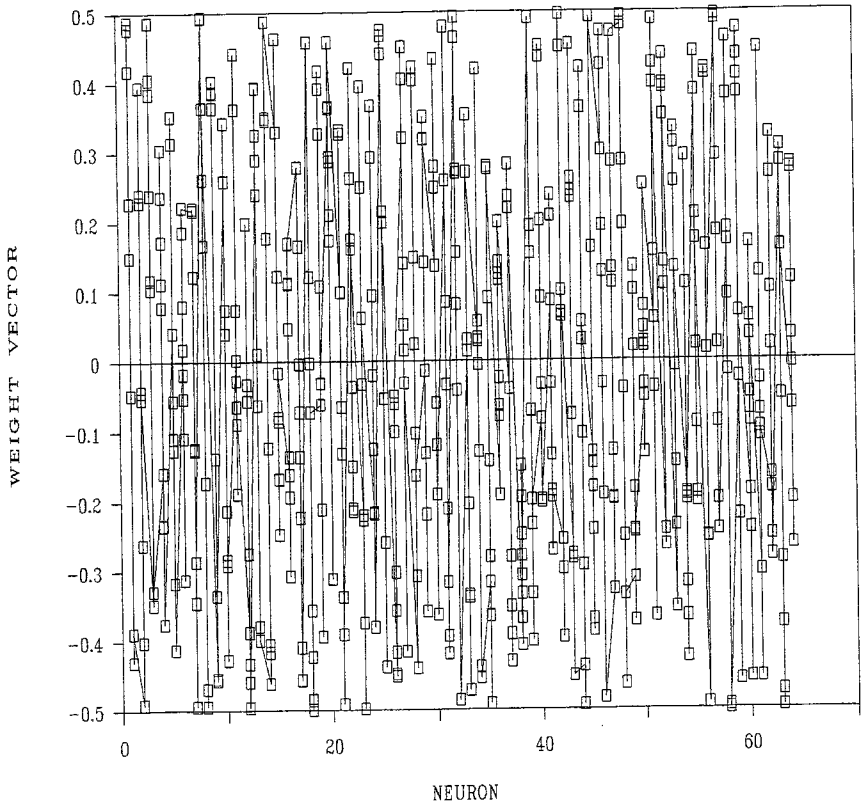


Figure 3a: Problem 9.16
Initial map for SNR = 10 dB

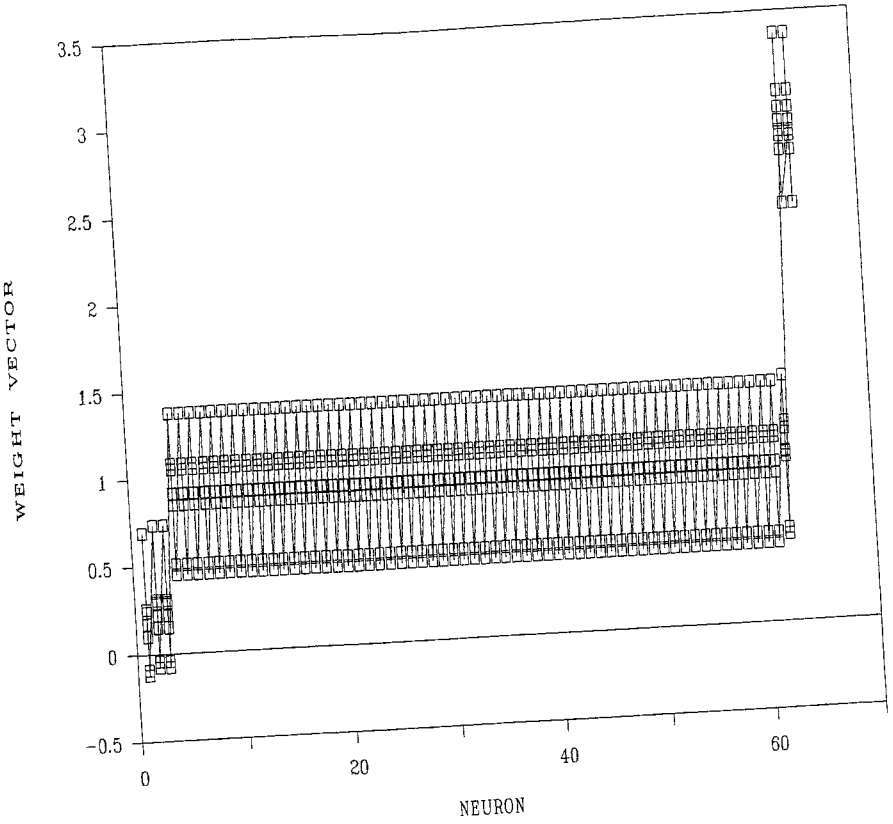


Figure 3b: Problem 9.16
SNR = 10 dB; map after 5,000 patterns

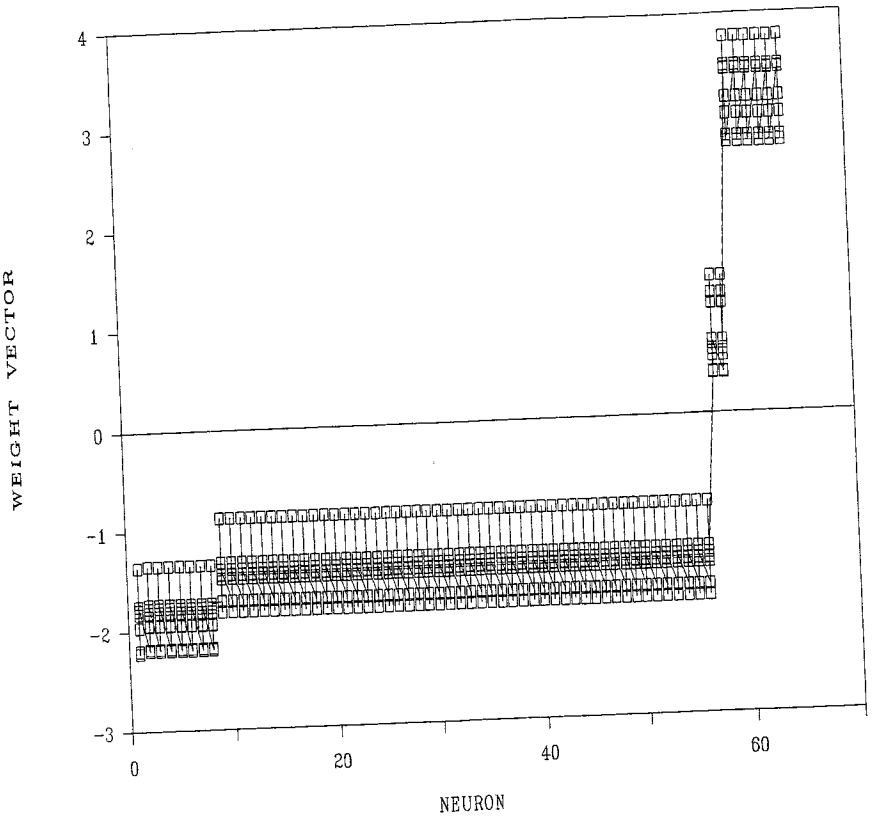


Figure 3c: Problem 9.16
SNR = 10 dB; map after 10,000 patterns

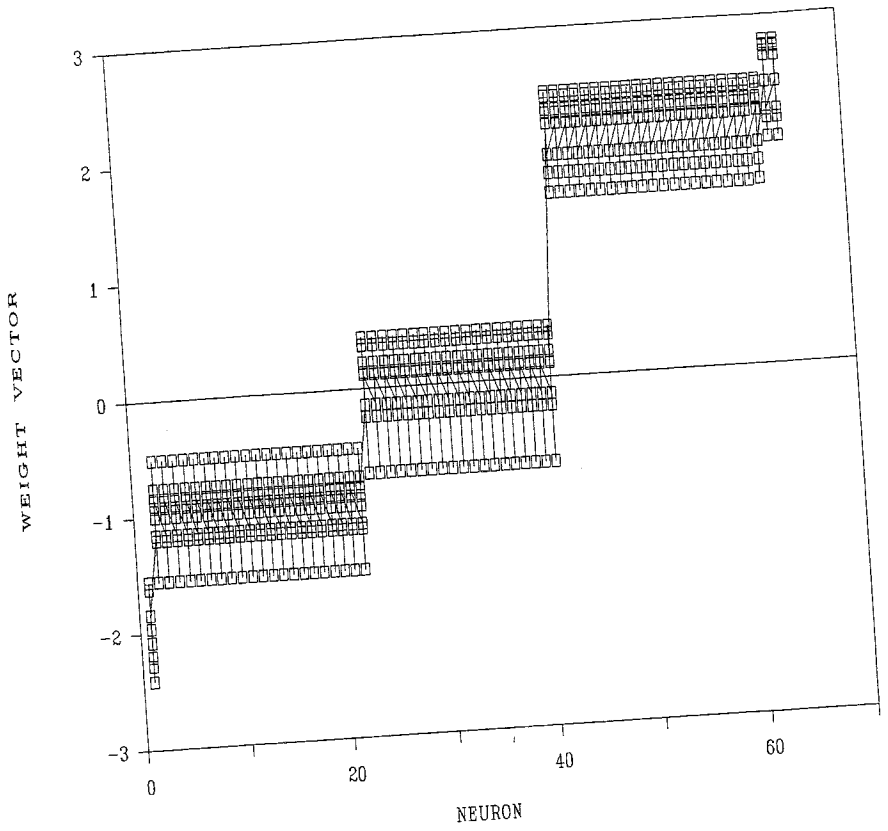


Figure 3d: Problem 9.16
SNR = 10 dB; map after 25,000 patterns

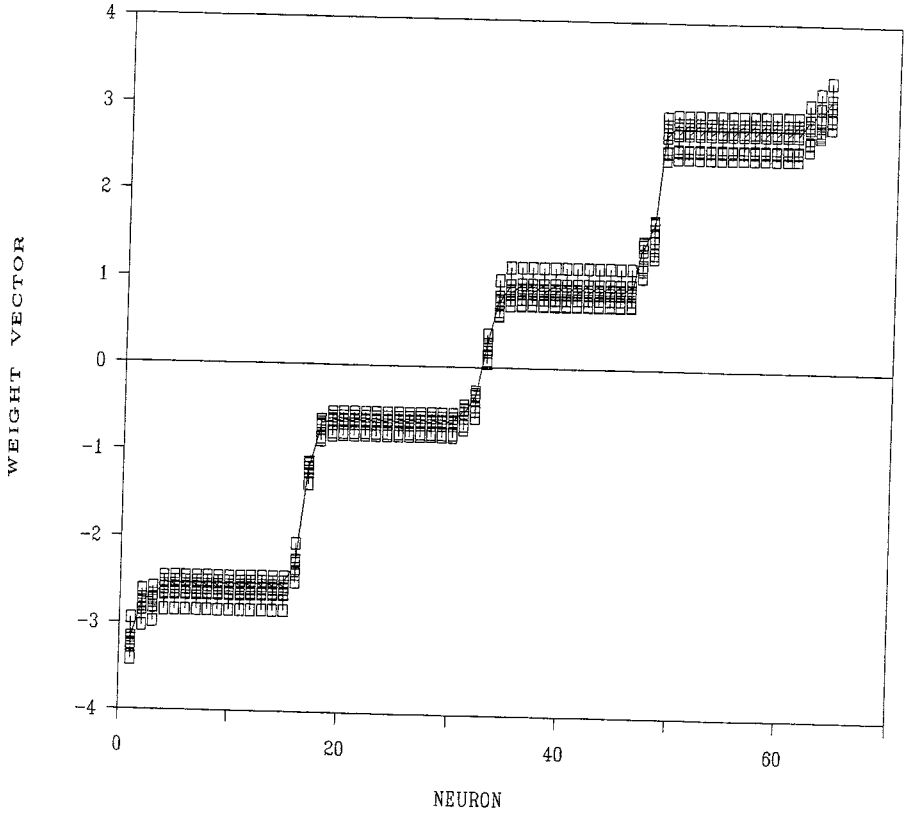


Figure 3e: Problem 9.16
SNR = 10 dB; map after 50,000 patterns

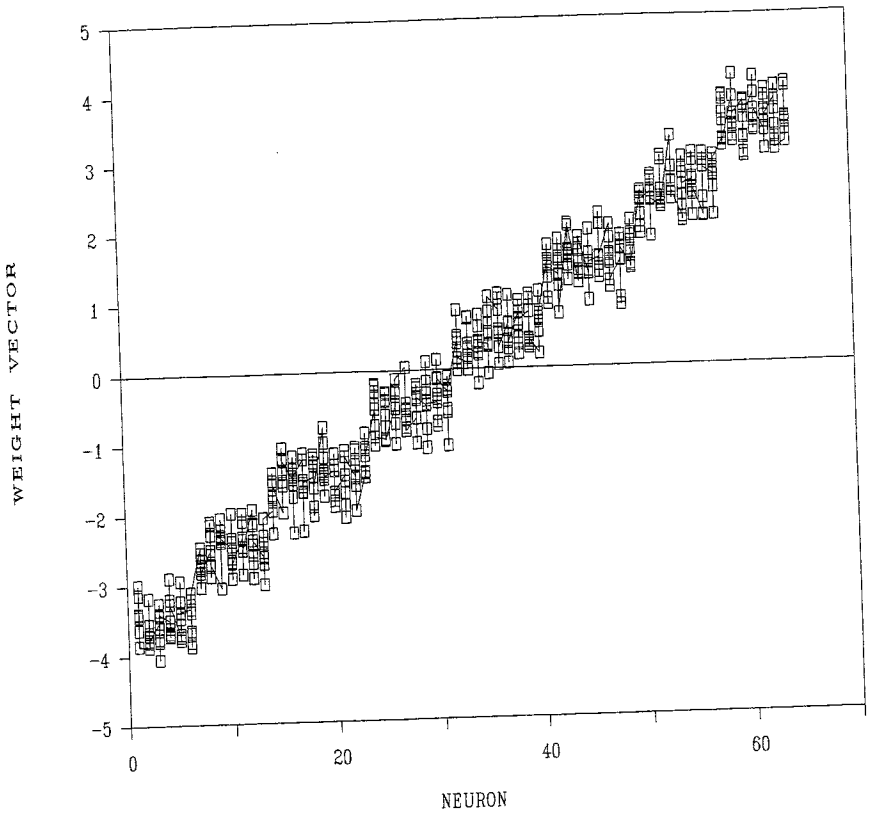


Figure 3f: Problem 9.16
SNR = 10 dB; map after 75,000 patterns

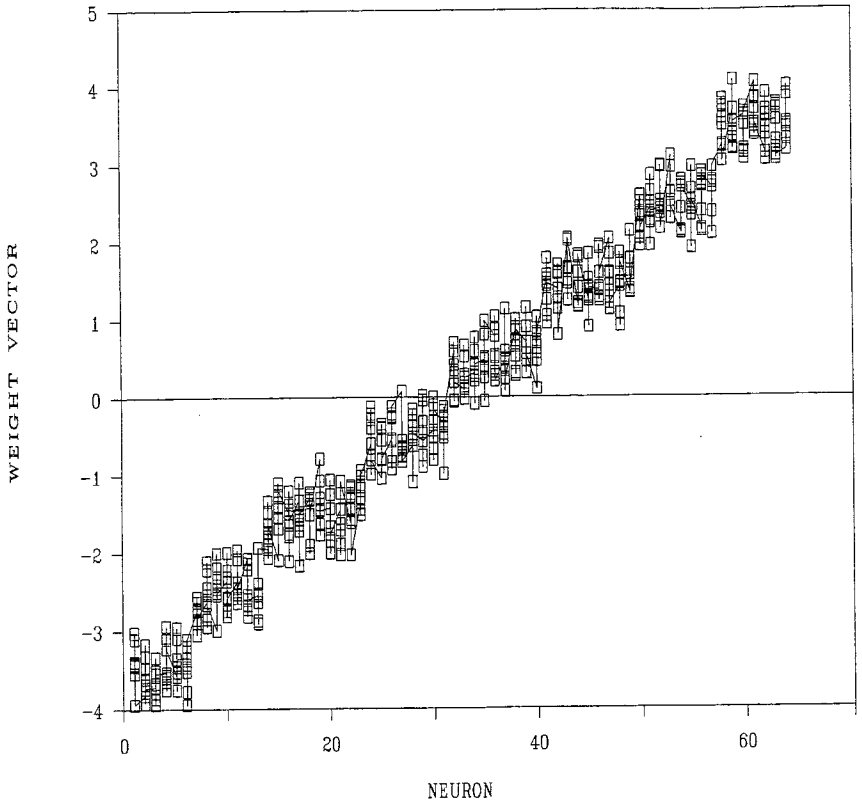


Figure 3g: Problem 9.16
SNR = 10 dB; map after 100,000 patterns

Chapter 10: Information-Theoretic Learning Models

Problem 10.30

Consider the linear system described in Figure 10.9 in the text involving the following three independent sources:

$s_1(n) = 0.1 \sin(400n)\cos(30n)$
$s_2(n) = 0.01 \operatorname{sgn}(\sin(500n + 9\cos(40n)))$
$s_3(n) =$ noise uniformly distributed in the range [-1, 1]

The mixing matrix $\mathbf{A}$ is

$$\mathbf{A} = \begin{bmatrix} 0.56 & 0.79 & -0.37 \\ -0.75 & 0.65 & 0.86 \\ 0.17 & 0.32 & -0.48 \end{bmatrix}$$

The waveforms of the source signals are displayed in the left-hand side of Figure 1.

For the demixing, we used the batch version of the update rule described in Eq. (10.100); see Problem 10.18. Batch processing was chosen for the primary reason of improved convergence. The algorithm was implemented using the following conditions:

- *Initialization*. To initialize the algorithm, the weights in the demixing matrix $\mathbf{W}$ were picked from a random number generator with a uniform distribution inside the range [0.0, 0.05].
- *Learning rate*. The learning-rate parameter was fixed at $\eta = 0.1$.
- *Signal duration*. The time series produced at the mixer output had a sampling period of 10^{-4}s and contained $N = 65{,}000$ samples.

The right-hand side of Figure 1 displays the waveforms of the signals produced at the demixer's output after 200 iterations. Except for rescaling and permutation of the unknown source outputs, there are no discernible differences between the two sets of waveforms shown in the left-hand and right-hand sides of Figure 1. For the results presented here, the actual weight matrix used for initialization of the algorithm was

$$\mathbf{W}(0) = \begin{bmatrix} 0.0109 & 0.0340 & 0.0260 \\ 0.0024 & 0.0467 & 0.0415 \\ 0.0339 & 0.0192 & 0.0017 \end{bmatrix}$$

The algorithm converged to the final weight matrix

$$\mathbf{W} = \begin{bmatrix} 0.2222 & 0.0294 & -0.6213 \\ -10.1932 & -9.8141 & -9.7259 \\ 4.1191 & -1.7879 & -6.3765 \end{bmatrix}$$

The corresponding value of the matrix product **WA** is

$$\mathbf{WA} = \begin{bmatrix} -0.0032 & -0.0041 & 0.2413 \\ -0.0010 & -17.5441 & -0.0002 \\ 2.5636 & 0.0515 & -0.0009 \end{bmatrix}$$

Rearranging terms in this matrix product so that the output signals appear in the same order as the original source signals, we may write

$$\mathbf{WA} = \begin{bmatrix} 2.5636 & 0.0515 & -0.0009 \\ -0.0010 & -17.5441 & -0.0002 \\ -0.0032 & -0.0041 & 0.2413 \end{bmatrix}$$

The first, second, and third rows of the matrix product correspond to the amplitude-modulated signal, clipped frequency-modulated signal, and noise, respectively. The diagonal elements of **WA** define the factors by which the output waveforms on the right-hand side of Figure 1 have been scaled with respect to the original source waveforms on the left-hand side of the figure.

For a quantitative evaluation of the demixer's performance, we may use a global rejection index defined by (Amari et al., 1996)

$$I = \sum_{i=1}^{m}\left(\sum_{j=1}^{m}\frac{|p_{ij}|}{\max\limits_{k}|p_{ik}|} - 1\right) + \sum_{j=1}^{m}\left(\sum_{i=1}^{m}\frac{|p_{ij}|}{\max\limits_{k}|p_{ki}|} - 1\right)$$

where $\mathbf{P} = \{p_{ij}\} = \mathbf{WA}$. The performance index I is a measure of the *diagonality* of matrix **P**. If the matrix **P** is perfectly diagonal, $I = 0$. For a matrix **P** whose elements are not concentrated on the principal diagonal, the performance index I will be high.

For the waveforms displayed in Figure 1, $I = 0.0606$.

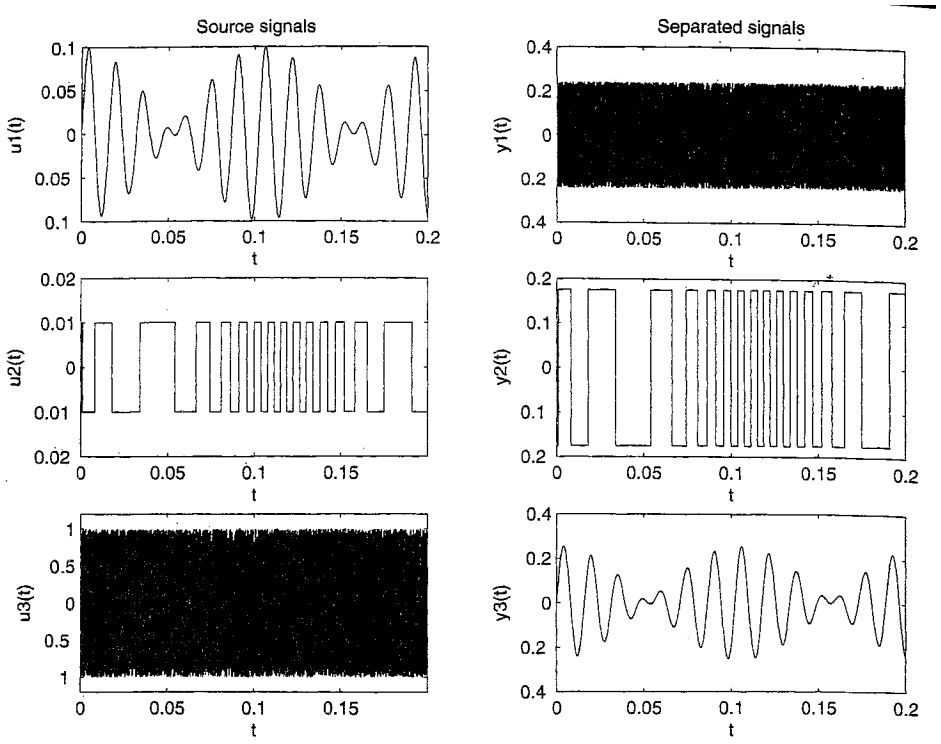


Figure 1: Problem 10.30

Problem 10.31

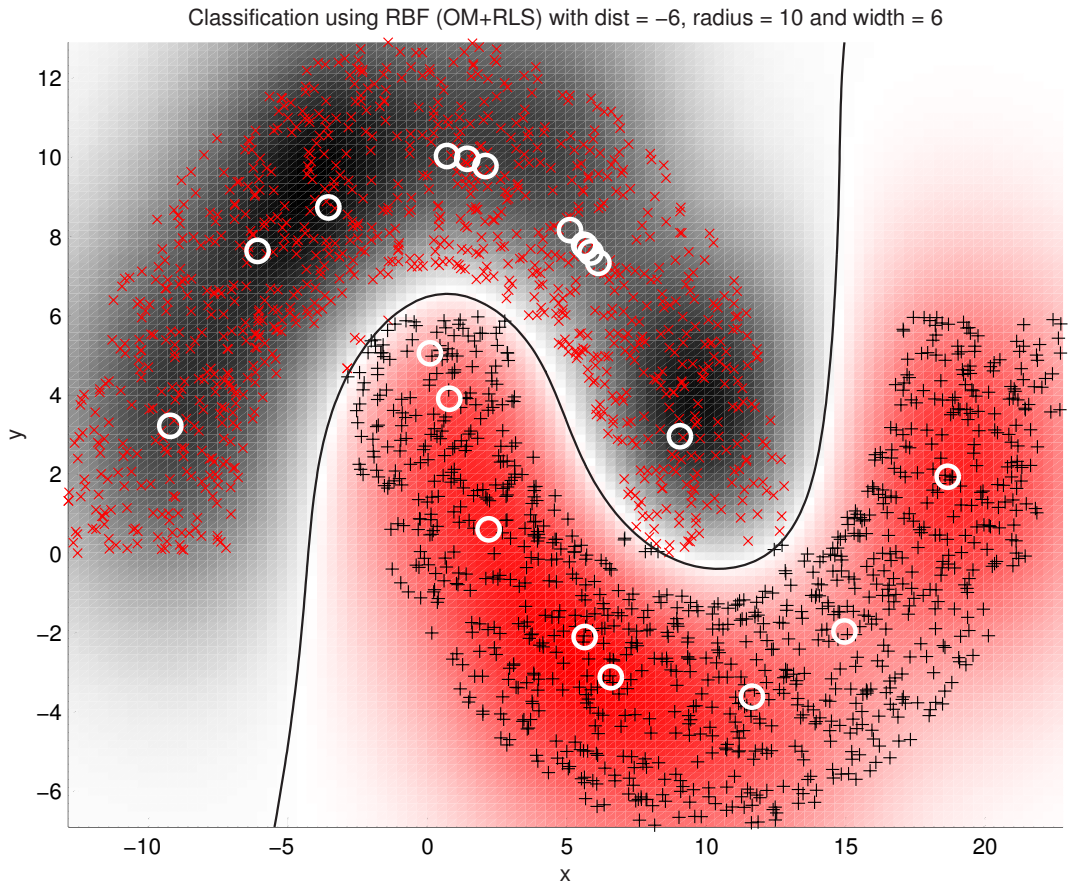


Figure 1 (Problem 10.31): Classification using optimal manifold + RLS algorithm with $d = -6$. Error points : 12 (0.60%)

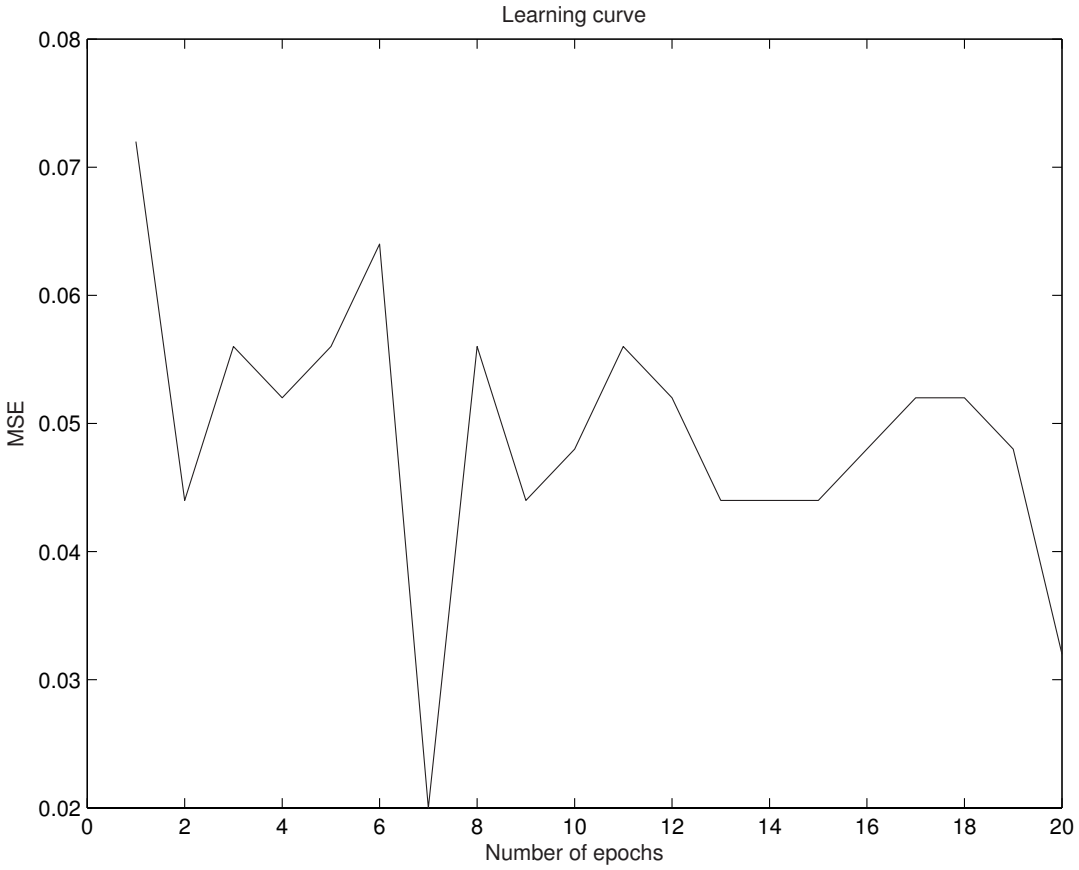


Figure 2 (Problem 10.31): Learning curve for the optimal manifold + RLS with $d = -6$. Error points : 12 (0.60%)

Summarizing remarks:

- (a) Comparing the classification results of Figure P10.31, based on the RLS algorithm for the output layer, with that of Section 10.21 in the text, based on the LMS algorithm, we see that the percentage error rate has doubled.

- (b) Moreover, computational complexity of the RLS algorithm is more demanding than that of the LMS algorithm.

In light of these two points, it appears that the LMS algorithm is the preferred choice over the RLS algorithm for the output layer of the classifier.

Chapter 14: Bayesian Filtering for State Estimation of Dynamic Systems

Problem 14.21

The simulation results, presented herein, will hardly ever match each other unless a common seed value for the random number generator is used.

Setting up the simulation environment is not a hard but necessary task. It should also show that it is often easier to develop new methods using simulated data, where we can change parameters and do not have to cope with all difficulties from the beginning.

For using the particle filter, we need to formulate the prior distribution as well as the likelihood. We define the state space as $\mathrm{x}_t = [x, y, v_x, v_y]^T$, where (x, y) are the center coordinates of the tracked object and (v_x, v_y) are the velocities in the x and y direction, respectively. The process model defines the velocities to be constant, adding the noise term $\upsilon_t = N(0|\Sigma_\upsilon)$ to cope with deviations from this assumption. This leads to an autoregressive model for the prior density, represented by

$$x_{t+1} = A \cdot x_t + \upsilon_t$$

where our criteria are fulfilled if

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The observation takes place by simply finding a point of the tracked object, which is colored in gray. This is obviously simplified compared to real–world computer vision tasks, but it is an assumption that keeps the experiment manageable. The observation still includes some noise, because the object is of finite size and we do not always detect the centroid. Thus, the likelihood function and therefore the weights are defined as

$$\omega_{\mathrm{t}}^{\mathrm{i}} = e^{-\frac{1}{2}(y_t^{obs} - y_t^i)^T R^{-1} (y_t^{obs} - y_t^i)} + \varepsilon$$

with y_t^{obs} being the observed state. The predicted observation of the $i-th$ particle y_t^i equals the state x_t^i, so the measurement model comes down to an equality in our case. R is the measurement variance, and ε is a very small value to avoid ill–conditioning of the weights.

If no observation is possible, this likelihood definition fails as y_t^{obs} is not available. In our case, this means that no gray point indicating the object position can be found in the current frame. It is

quite easy to handle this case by setting

$$\omega_t^i = \frac{1}{N_P}$$

where N_P is the number of particles.

The last design decision is the resampling method. For this experiment a systematic resampling approach has been used, as it is fast and shows good results.

Figure 1 shows average tracking results for height $h = 10$ and $h = 40$ pixels. The crosses indicate the observed position, and the line shows the trajectory of the mean of all particles.

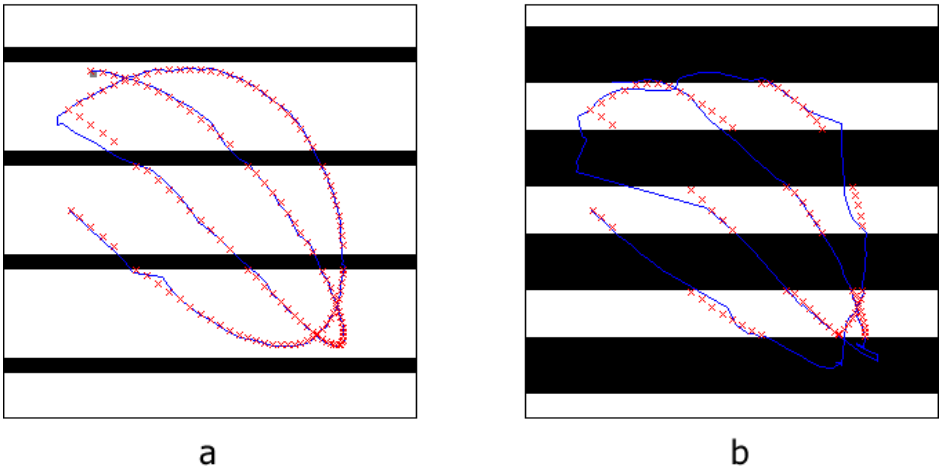


Figure 1 (Problem 14.21): Average tracking results (N_P=100): (a) bar height h=10, (b) h=40.

It can be quite hard to find good values for the process variance Σ_v. High values increase the observed area around the predicted position of the object, so the particle filter can cope with objects which change their position and direction of movement very fast. However, high variance also decreases the precision of tracking as the particles spread out quickly and so the object might get lost for a sequence of bad measurements or missing observations. A simple solution is to increase the number of particles, which, in turn, increases the complexity of computation and slows down the simulation.

The frame rate influences the number of observations and therefore the necessary velocity of the particles. For high frame rates it is possible to decrease the process variance Σ_v and the number of particles, as new observations are close to previous ones. However, there are also more frames

where the object is not observable. Low frame rates show the opposite behavior, so the frame rate should be chosen well.

Figure 2 (Problem 14.21): Average depth estimation results (N_P=100): (a) bar height h=10, (b) h=40.

If the tracked object is not visible, we can tell that some other object to occlude it. We also know the estimated position of the object because of our tracker. Therefore, it is possible to estimate the relative depth of stationary objects in relation to the tracked object, as long as the objects are well segmented and interact with each other.

For assigning the depth to the image elements, some preparation steps have to be executed. We use an edge detector to get the object boundaries from the segmented image (which, in this simplified case, is the input image). Furthermore, we use a morphological closing operation to get closed boundaries around each relevant region, called the *bins* of the image. For depth evaluation, a bin always has the same depth. This means, that a tracked point inside a bin always changes the depth for the entire bin.

If the tracker was able to observe the object in a frame, then the area is obviously background and the relative depth is added. If it cannot, we observe the object by subtracting the relative depth is subtracted from the depth image. This procedure turns areas with high probability of being in foreground dark and areas in background light. Figure 2 shows average-depth estimation results for height $h = 10$ and $h = 40$ pixels. As we can see from this figure, the method works quite well for both scenarios, considering that the foreground bars occlude more than 50% of the second image.

Chapter 15: Dynamically Driven Recurrent Networks

Problem 15.17

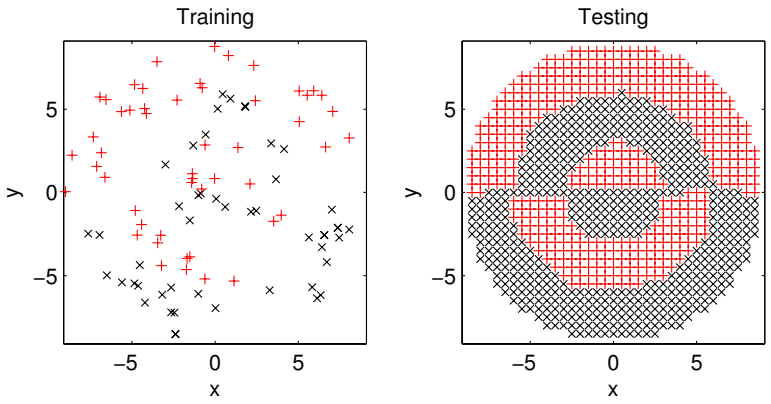


Figure 1 (Problem 15.17): Classification using EKF algorithm for tight-fisted problem with $d_1 = 3, d_2 = 6$ and $d_3 = 9$. Error rate (approx.) 9%.

Summarizing remarks:

- The classification error rate obtained using the EKF algorithm, is about 9%.
- The corresponding result reported in the solution to Problem 6.25, obtained using the support vector machine (SVM) with $C = 500$ is 7%.

It therefore appears that training the multilayer perceptron using the EKF is slightly worse than when SVM is used. However, use of the EKF outperforms the SMV when it comes to the time needed for training.