8-1-1997

POSTERIORI PROBABILITY ESTIMATION AND PATTERN CLASSIFICATION WITH HADAMARD TRANSFORMED NEURAL NETWORKS

Peter G. Gulden
Purdue University School of Electrical and Computer Engineering

Okan Ersoy
Purdue University School of Electrical and Computer Engineering

POSTERIORI PROBABILITY ESTIMATION AND PATTERN CLASSIFICATION WITH HADAMARD TRANSFORMED NEURAL NETWORKS

PETER G. GULDEN
OKAN ERSOY

TR-ECE 97-8
AUGUST 1997

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

POSTERIORI PROBABILITY ESTIMATION

AND PATTERN CLASSIFICATION

WITH HADAMARD TRANSFORMED

NEURAL NETWORKS

Peter G. Gulden
Okan Ersoy

School of Electrical and Computer Engineering
1285 Electrical Engineering Bldg.
Purdue University
West Lafayette, IN 47907-1285

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Neural networks, trained with the backpropagation algorithm have: been applied to various classification problems. For linearly separable and nonseparable problems, they have been shown to approximate the a posteriori probability of an input vector X belonging to a specific class C.

In order to achieve high accuracy, large training data sets have to be used. For a small number of input dimensions, the accuracy of estimation was inferior to estimates using the Parzen density estimation.

In this thesis, we propose two new techniques, lowering the mean square estimation error drastically and achieving better classification. In the past, the desired output patterns used for training have been of binary nature, using one for the class C the vector belongs to, and zero for the other classes. This work will show that by training against the columns of a Hadamard matrix, and then taking the inverse Hadamard transform of the network output, we can obtain more accurate estimates.

The second change proposed in comparison with standard backpropagation networks will be the use of redundant output nodes. In standard backpropagation the number of output nodes equals the number of different classes. In this thesis, it is shown that adding redundant output nodes enables us to decrease the mean square error at the output further, reaching better classification and lower mean square error rates than the Parzen density estimator.

Comparisons between the statistical methods, the Parzen density estimation and histogramming, the conventional neural network and the Hadamard transformed neural network with redundant output nodes are given.

Further, the effects of the proposed changes to the backpropagation algorithm on the convergence speed and the risk of getting stuck in a local minimum are: studied.

1. INTRODUCTION

It has been shown previously that neural networks whose learning is based on minimizing the mean square error function at the output approximate the a posteriori class probabilities $P(C_i|Xi)$ given the input vector Xi. [1], [2], [3], [4]. However, for these approximations to be correct, a very large set of training data is required [3], and the results are not significantly better than those of parametric estimation models. The correct approximation of the a posteriori probabilities is of great interest for classification problems. When the a posteriori probabilities are estimated correctly, we can give a classification confidence, being the difference between the 2 classes with the highest probability.

In order to distinguish between vectors and matrices on one side, and scalars on the other, we use bold letters for matrices and vectors. For example, $\mathbf{O_i}$ or $\mathbf{O(X_i)}$ being the vector containing the outputs for an input vector **Xi.** The output values of the output nodes are the components of $\mathbf{O_i}$. If we refer to probabilities, estimations etc. the bold notation stands for the probability, estimation, etc. of each component. For example, $\mathbf{P(C|Xi)}$ is a vector, where each component equals: $P(C_j|\mathbf{Xi})$.

We choose to show the model both in matrix and scalar notation. The underlying properties are more obvious in the scalar notation, while the matrix notation is more useful for cases with many classes. At one place, section 2.4, the matrix notation has to be dissolved into scalar notation. Otherwise it would have been impossible to resolve and simplify the problem.

1.1 Hadamard-Transformed Output Representations

We will refer to the 0-1 representation as the binary output representation in which the desired output value is 1 at the output node for the class the input vector belongs to,

and the desired output values are zero at the other output nodes. The Hadamard-transformed representation will be the product of the desired output vector D of length N with a Hadamard matrix of size N.

$$\mathbf{D_h = H\,D} \tag{1.1}$$

When the 0-1 representation is used, the outputs directly estimate the a posteriori probabilities. If a different binary output representation is used, i.e. 1 imd -1 instead of 0 and 1, the outputs have to be scaled and shifted to obtain the probability estimates. However, the meaning of the output values remains the same [2]. If we do not use such a binary representation (i.e. if we use other possible binary representations as in computers or digital communications), then the output values no longer show the a posteriori probabilities, and instead show the probability of the desired output of this node being one [2].

Chapter 2 will show that the Hadamard-transformed output representations can reduce the estimation error for the a posteriori probabilities significantly. First, a simple theoretical model will be set up, with the assumption of an unbiased independent output error. This will be explored experimentally, and a more detailed model, without underlying assumptions, will be given. Extensive treatment of Hadamard matrices, transformation etc. can be found in [5], [6].

1.3 Statistical Classifier versus Neural Networks

We will also compare these results with those of non parametric estimation models, in particular Parzen density estimation [7], [8], and histogramming [8]. Histogramming is probably the oldest known probability estimation technique. It is easy to apply, fast and well investigated. The Parzen density estimation was developed much later. It can be seen as a windowed average of all points within the kernel range of the estimator kernel at one specific point. So far, for the 1 and 2 dimensional case, the Parzen density estimation is the most accurate estimator. However, due to computing the kernel function

for all points, the computation can become excessive for high resolutions, and higher dimensions. Both methods have the disadvantage, that the distributions generated during training have to be stored in a lookup table.

Neural networks on the other hand, provide the desired probability values by forward propagation of the testing vectors, hence by simple matrix multiplication. On the other hand they require larger training times for 1 and 2 dimensional cases., and there results have been inferior to those of Parzen density estimators.

We will compare the results of a binary neural network, a Hadamard-transformed neural network, histogramming and the Parzen density estimation.

1.3 Redundant Nodes

Our previous findings, like in Chapter 2, suggest that the performance of the Hadarnard-transformed neural network improves when the size of the Hadamard matrix is increased. However, that would be equal to choosing a 16, or higher, class problem. We will show that we do not have to increase the number of classes, but can simply add zero-components to the binary vector of size F. We can then take the Hadamard-transform and obtain only the first F columns of the Hadamard matrix as desired outputs.

This chapter will show that the model set up in chapter 2 covers the case of redundant nodes as well. We will also investigate the limitations of zero - padding the binary output response. As in Chapter 3, a comparison is given between Parzen density estimation and the neural networks will be given, showing that with enough redundant output nodes, the neural networks can actually perform better. The problems accompanying the increase in network complexity are also investigated.

1.4 Convergence Issues

Neural networks learning can be seen as learning a function mapping **F(X,D)** between the input vector set **X** and the desired output vector set **D.** Clearly, this input - output

mapping is effected by the choice of the output representation. In Chapter 5 we will investigate the effects of both Hadamard-transforming and zero-padding the desired output vectors. We will also pay some attention to initialization. This issue seems to have lost some of its importance due to backpropagation algorithms with adaptive learning rate, like used throughout this thesis.

2. HADAMARD-TRANSFORMED OUTPUT REPRESENTATION

2.1 Introduction

This chapter provides a complete analysis of Hadamard-transformed output representations to neural networks. We define the Hadamard-transformed output representation to be given by

$$\mathbf{D_h = H\,D} \tag{2.1}$$

where D is the matrix of the different desired outputs. In our case, where the binary output representation is used, D is the identity matrix of size N.

The Hadamard-transform is used in statistical design of experiments and in systems such as optical spectrometers[5]. It reduces the variance of measurement errors by 1/N, where N is the size of the Hadamard matrix. In such applications, the size of the Hadamard matrix equals the number of measurements. Instead of measuring each variable separately, different combinations determined by the Hadamard matrix are measured. Then, the values of the variables are obtained using the inverse Hadamard-transform.

During testing, the outputs of the network are inverse Hadamard-transformed to obtain the results equivalent to the 0-1 representation [9], [10], [11]. In this work, we show that the Hadamard-transformed output representation in neural networks leads to the same advantages as in statistical design of experiments. The Hadamard-transformed output representation yields better classification results and a better approximation of the a posteriori probabilities.

In section 2.2, we set up a simple model for the expected results of the Hadamard-transformed neural network. In Section 2.3 we experimentally test the predictions of the model set up. Section 2.4 provides modified model, which is confirmed by a second set of experiments. In Section 2.6 we introduce a simple method to estimate the error

estimation results of a neural network without knowing the underlying distributions of the training and testing data.

2.2 Theoretical Model of Hadamard-Transformed Networks

Hadamard matrices are orthogonal and consist of elements h_{ij} which are either 1 or -1. The inverse of a Hadamard matrix can be obtained by transposing it and dividing it by its size N. For symmetrical (or Sylvester form) Hadamard matrices, this reduces to dividing the Hadamard matrix by its size N.

Let $P(C_j | Xi)$ be the a posteriori probability of occurrence of class Cj given that the input vector is Xi. Also let $\hat{P}(C_j | \mathbf{X_i})$ be the estimate of $P(C_j | \mathbf{X_i})$. We assume that we have trained the neural network with the Hadamard-transformed output $\mathbf{D_h}$ and then computed the inverse Hadamard-transform. We then compare the error e_{ji} between the estimated probability $\hat{P}(C_j | \mathbf{X_i})$ and the true probability $P(C_j | \mathbf{X_i})$:

$$e_{ji} = \hat{P}(C_j | \mathbf{X_i}) - P(C_j | \mathbf{X_i}) \qquad (2.2)$$

The error vector $\mathbf{e^o_i}$ at the output nodes is defined by

$$\mathbf{e^o_i} = \mathbf{O(X_i)} - H\ P(C | Xi) \qquad (2.3)$$

where $P(C | Xi)$ is the true probability vector of Xi.

We will assume that the error components e^o_{ji} are unbiased with different variances, and the dependencies between the errors at the different nodes are small enough to be neglected.

The square $\mathbf{S^2}$ of a matrix or vector $\mathbf{S}$ is defined as being obtained by squaring each component of the vector or matrix $\mathbf{S}$. Then, the following equations are obtained:

$$\mathbf{E\{e^o_i\} = 0} \tag{2.4}$$

$$\mathbf{S^o = E\{e^o_i\, e^o_i\, ^t\}} \tag{2.5}$$

$$\mathbf{E\{e^o_i\ e^o_k\} = 0 \quad i \neq k} \tag{2.6}$$

$$E\{e^o_{ji}\ e^o_{li}\} = 0 \quad j \neq l \tag{2.7}$$

where $\mathbf{S^o}$ is the covariance matrix at the output of the neural network. Its diagonals contain σ_{oj}^2 as components, while all other components are 0 due to independence. $\mathbf{0}$ is the null vector. Equation **(2.6)** is results from the independence of the different training vectors, while equation **(2.7)** results from the assumption of independence of the different probabilities for each class for the same vector.

The output vector $\mathbf{O(X_i)}$ can be written as

$$\mathbf{O(X_i) = H\, P(C \mid Xi) + e^o_i} \tag{2.8}$$

$\hat{\mathbf{P}}\ \mathbf{(C \mid Xi)}$ is obtained by inverting this equation:

$$\hat{\mathbf{P}}\ \mathbf{(C \mid Xi) = H^{-1}\ O(X_i) = P(C \mid Xi) + e_i} \tag{2.9}$$

where $\mathbf{e_i}$ is the estimation error. Its mean is given by

$$\mathbf{E\{\ e_i\} = E\{\ H^{-1}\ e^o_i\} = H^{-1}\ E\{e^o_i\} = 0} \tag{2.10}$$

The covariance matrix of the estimation error after the inverse Hadamard-transform is given by

$$\mathbf{S} = E\{ [\mathbf{H}^{-1} \mathbf{e}^{o}_{i}]^{2}\} \tag{2.12}$$

Using $\mathbf{H}^{-1} = \mathbf{H}^{t}/\mathbf{N}$ and the independence of the components of $\mathbf{e}^{o}_{i}$ we obtain a covariance matrix **S** with each diagonal element equal to:

$$\sigma^2 = \frac{1}{N^2} \sum_{i=1}^{n} E\{e_i^{o2}\} = \frac{1}{N^2} \sum_{i=1}^{n} \sigma_{oi}^2 \tag{2.13}$$

We can drop all the non-quadratic terms due to independence of e_{ji}. The Hadarnard matrix can be dropped as well, since its entries are 1 or -1, and all the remaining terms are quadratic.

If we assume $\sigma_{oij}^2 = \sigma_o^2$, then equation (2.12) would simplifies to

$$\mathbf{S} = \sigma^2 \mathbf{I} = (\sigma_o^2 / N) \mathbf{I} \tag{2.14}$$

with **I** as identity matrix of size N.

This result shows that the variance of the estimation error with the Hadamard representation is N times smaller than for the 0-1 representation.

For example, with N = 4, this is the same as

$$O_1(X_i) = P_{1i} + P_{2i} + P_{3i} + P_{4i} + e^{o}_{1i} \tag{2.15}$$

$$O_2(X_i) = P_{1i} - P_{2i} + P_{3i} - P_{4i} + e^{o}_{2i}$$

$$O_3(X_i) = P_{1i} + P_{2i} - P_{3i} - P_{4i} + e^{o}_{3i}$$

$$O_4(X_i) = P_{1i} - P_{2i} - P_{3i} + P_{4i} + e^{o}_{4i}$$

where P_{ji} equals $P(C_j \mid Xi)$. P_{ji} 's are estimated by inverting Eq. (2.15):

(2.16)

$$\hat{P}_{1i} = (O_{1i} + O_{2i} + O_{3i} + O_{4i})/4 = P_{1i} + (e^o_{1i} + e^o_{2i} + e^o_{3i} + e^o_{4i})/4 = P_{1i} + e_{1i}$$

$$\hat{P}_{2i} = (O_{1i} - O_{2i} + O_{3i} - O_{4i})/4 = P_{1i} + (e^o_{1i} - e^o_{2i} + e^o_{3i} - e^o_{4i})/4 = P_{2i} + e_{2i}$$

$$\hat{P}_{3i} = (O_{1i} + O_{2i} - O_{3i} - O_{4i})/4 = P_{3i} + (e^o_{1i} + e^o_{2i} - e^o_{3i} - e^o_{4i})/4 = P_{3i} + e_{3i}$$

$$\hat{P}_{4i} = (O_{1i} - O_{2i} - O_{3i} + O_{4i})/4 = P_{4i} + (e^o_{1i} - e^o_{2i} - e^o_{3i} + e^o_{4i})/4 = P_{4i} + e_{4i}$$

Since $E\{e^o_{ji}\} = 0$, we have

$$E\{e_{1i}\} = E\{(e^o_{1i} + e^o_{2i} + e^o_{3i} + e^o_{4i})/4 = 0 \qquad (2.17)$$

$$E\{e_{2i}\} = E\{(e^o_{1i} - e^o_{2i} + e^o_{3I} - e^o_{4i})/4\} = 0$$

$$E\{e_{3i}\} = E\{(e^o_{1i} + e^o_{2I} - e^o_{3i} - e^o_{4i})/4\} = 0$$

$$E\{e_{4i}\} = E\{(e^o_{1i} - e^o_{2I} - e^o_{3i} + e^o_{4i})/4\} = 0$$

Using the independence of e_{1i}, its variance is given by

$$\sigma^2 = E\{e_{1i}^2\} = E\{[(e^o_{1i} + e^o_{2i} + e^o_{3i} + e^o_{4i})/4]^2\} \qquad (2.18)$$

$$= E\{e^o_{1i}{}^2 + e^o_{2i}{}^2 + e^o_{3i}{}^2 + e^o_{4i}{}^2\} / 16$$

$$= (\sigma_{1o}{}^2 + \sigma_{2o}{}^2 + \sigma_{3o}{}^2 + \sigma_{4o}{}^2) / 16$$

where $\sigma_{1o}{}^2$ is the error variance before the inverse Hadamard-transform. Now, if the variances are the same for all nodes, σ^2 is given by

$$\sigma^2 = \sigma_o{}^2/4 \qquad (2.19)$$

The other $E\{e_{ji}^2\}$ are the same for all nodes. This follows from the independence of the errors at the different nodes, so only the quadratic terms remain.

2.3 Experiments

We trained a two stage backpropagation network, using the mean square error as the cost function. The tangent hyperbolic and linear activation function. were used at the hidden layer and at the output layer, respectively. **A** linear activation function at the output can produce slightly negative values if the class probability is very small, say smaller than the error variance. However, using a logsig function here would produce a biased estimation, especially for small probabilities close to 0 and events with large probabilities close to 1. We trained 2 different networks, one with binary output representation, the other one with Hadamard-transformed output representation.

2.3.1. The random variable generator

The problem the network was trained with was an 8 classes separation problem. The classes were linearly nonseparable. Figure 1 shows the X-Y scatter of the data, and Figure 2 shows the probability distribution $P(X)$ in the 2 dimensional space. Each class of training data was synthetically generated with the same Gaussian distribution, with the covariance matrix S equal to the identity matrix. Each class has a different mean, as shown in Figure 2. The data is then divided into parts. The two parts are then transformed onto opposite sides of the circle center, in order to obtain the 2 opposing clusters of data. The distribution function for one class C_i is given by

(2.20)

$$P(\mathbf{X} \mid C_i) = \frac{1}{4\pi\sqrt{|\mathbf{S}_{cluster}|}} \left(e^{-0.5(\mathbf{X}-\eta_i)'\mathbf{S}_{cluster}^{-1}(\mathbf{X}-\eta_i)} + e^{-0.5(\mathbf{X}+\eta_i)'\mathbf{S}_{cluster}^{-1}(\mathbf{X}+\eta_i)} \right)$$

The data was generated with the same random variable generator, and scaled between 0 and 1.

In the 3 dimensional case, we simply added one more dimension, centered at 0.5. The data then is shaped like a 3 dimensional ring.

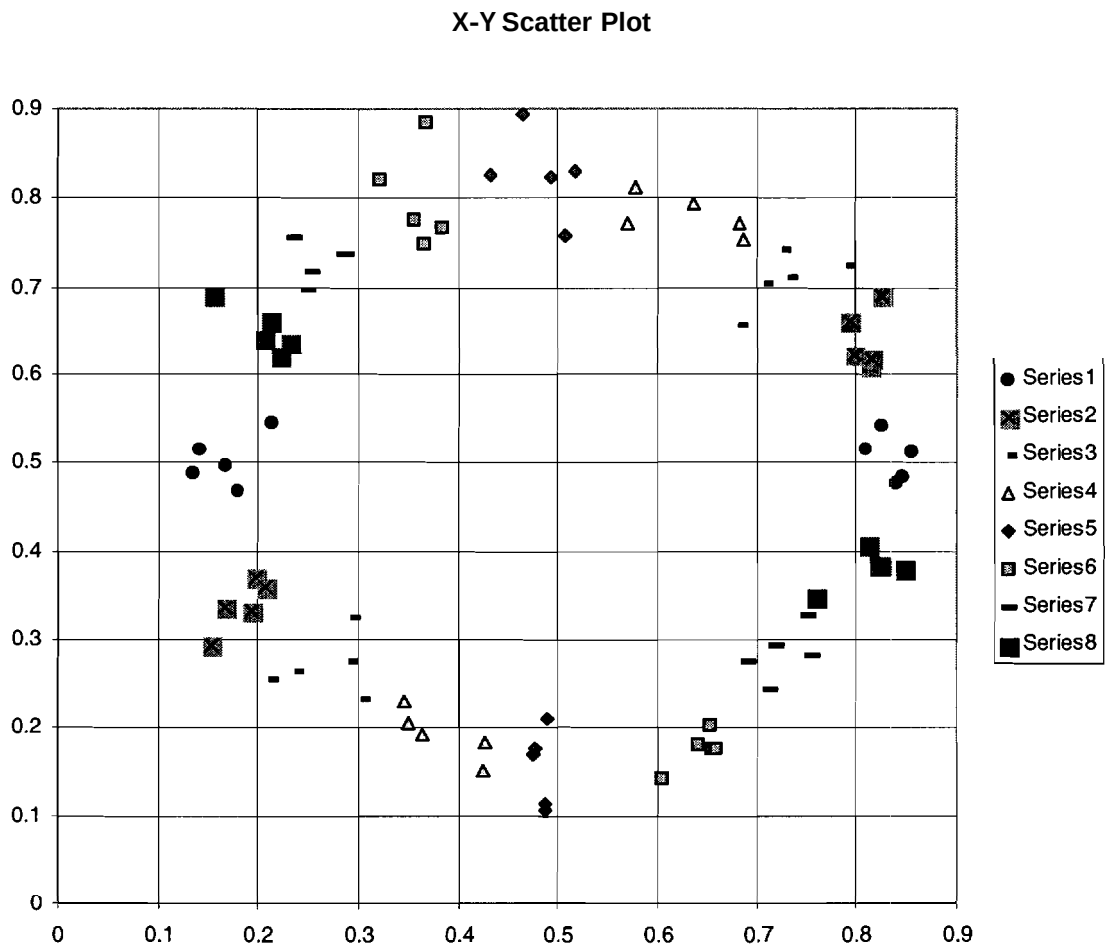

Fig. 2.1 Input data clustering for the 2-D case

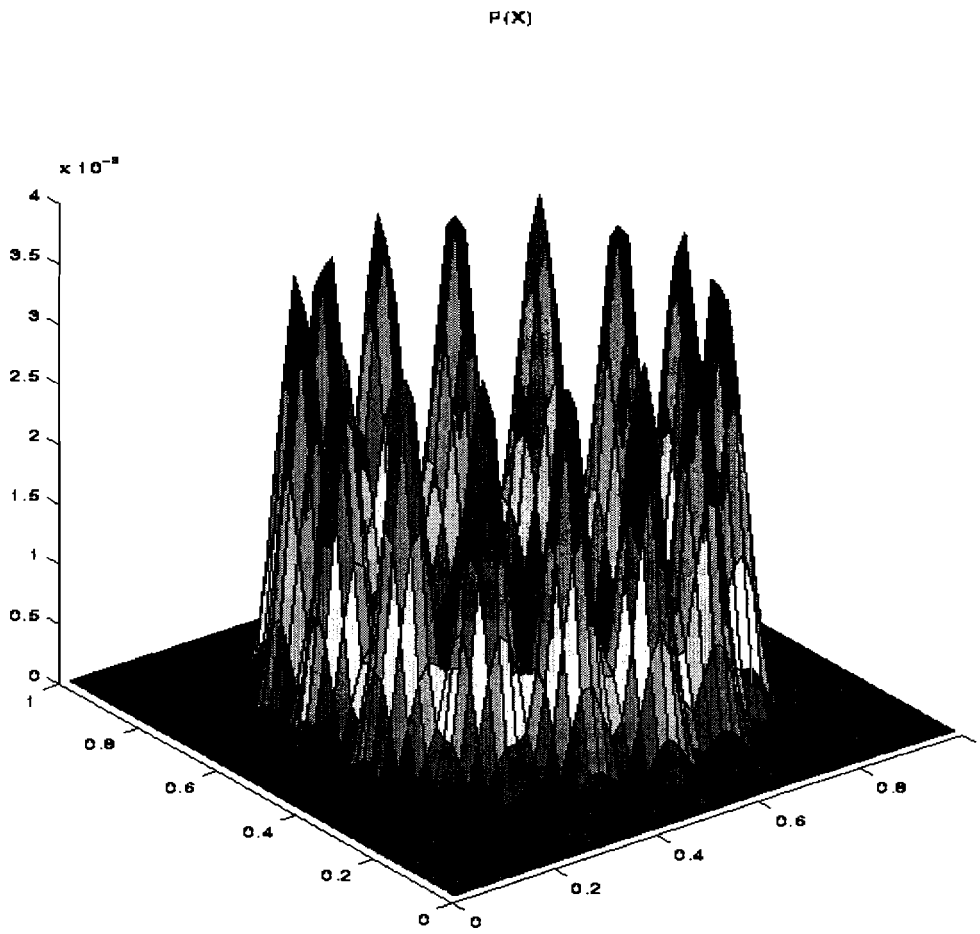

Fig. 2.2 Probability distribution of the input vectors

2.3.2 True a posteriori probabilities

The use of synthetic input data allows us to compute $P(C_j \mid Xi)$ directly, using simply the Bayesian rule:

$$P(C_j \mid \mathbf{X_i}) = P(\mathbf{X_i} \mid C_j) \frac{P(C_j)}{P(\mathbf{X_i})} \qquad (2.21)$$

$P(\mathbf{X_i}|C_j)$ is known, and $P(\mathbf{X_i})$ is given by total probability as

$$P(\mathbf{X_i}) = \sum_{j=1}^{N} P(\mathbf{X_i}|C_j) \tag{2.22}$$

Usually the a priori probability of each class is known. In the above case, $P(C_j)$ equals 1/8, so we can compute the a posteriori probabilities, using equation (21):

$$P(C_j|\mathbf{X_i}) = \frac{1}{4\pi N\sqrt{|\mathbf{S}_{cluster}|}\sum_{j=1}^{N} P(X_i|C_j)} \left(e^{-0.5(\mathbf{X}-\eta_i)'\mathbf{S}_{cluster}^{-1}(\mathbf{X}-\eta_i)} + e^{-0.5(\mathbf{X}+\eta_i)'\mathbf{S}_{cluster}^{-1}(\mathbf{X}+\eta_i)} \right)$$

2.3.3 Experimental results

We first ran a series of examples with 100 training vectors per class. For classification problems with nonseparable classes, the mean and the sum squared error do not converge to zero [1], [2], [3]. This results from the estimation of the a posteriori probabilities, which are not necessarily close to the desired output values. Since the sum squared error reaches high values, one cannot be sure whether a local or a global minimum is reached. Hence, we have to use a different criterion to measure the training success of the neural network. In our case, we decided to measure the number of correctly classified training patterns every 50 sweeps to show the progress of learning. Figure 2.3 shows a typical learning curve.

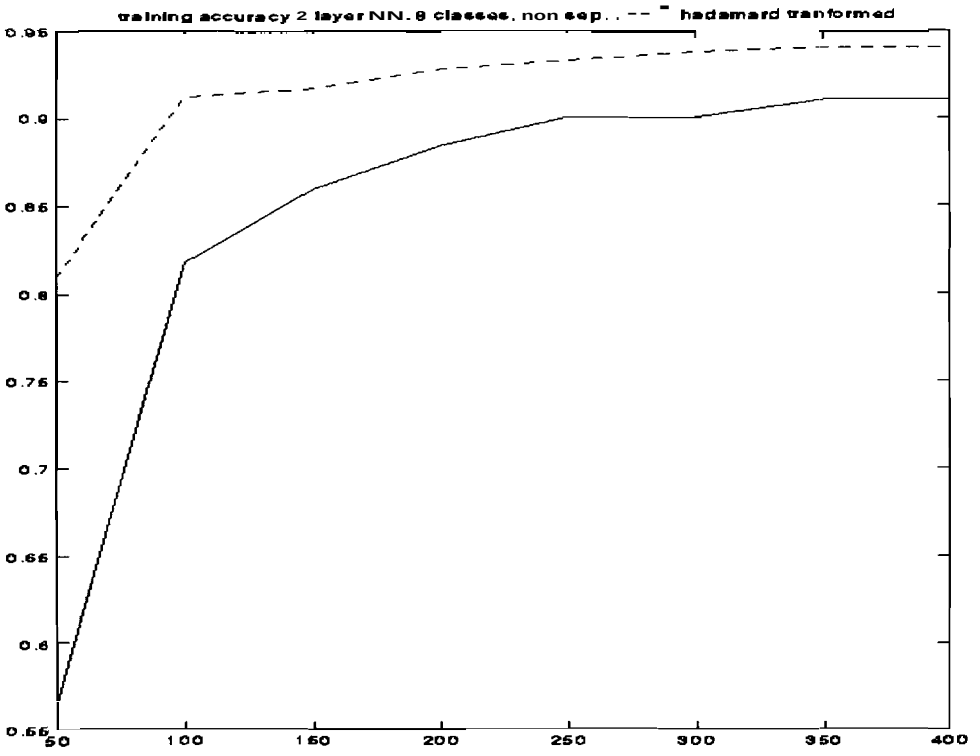

Fig. 2.3 Number of correctly classified training pattern

Tables 2.1, - 2.2 show the results for the variance of the estimation error and the mean of estimation error.

Table 2.1
Average variance of estimation error, 100 training vectors
2 - dimensional input data, 1000 vectors for testing:

output node	1	2	3	4
binary	0.01 409	0.01 410	0.02 185	0.02 981
Hadamard	0.00 918	0.00 969	0.02 102	0.01 773
binary	0.02 239	0.02 175	0.02 246	0.02 911
Hadamard	0.01 998	0.00 664	0.00 824	0.01 761
binary	0.03 193	0.02 758	0.02 014	0.02 409
Hadamard	0.02 358	0.01 500	0.00 774	0.00 803
binary	0.05 230	0.04 652	0.06 227	0.05 195
Hadamard	0.03 267	0.03 169	0.02 805	0.02 649
binary	0.03 947	0.04 480	0.02 825	0.04 111
Hadamard	0.02 779	0.02 691	0.02 813	0.02 663
binary	0.03 573	0.03 724	0.02 865	0.03 856
Hadamard	0.03 298	0.03 139	0.02 035	0.02 426

output node	5	6	7	8	average variance
binary	0.04 196	0.04 576	0.02 601	0.02 135	0.02 687
Hadamard	0.02 727	0.04 539	0.01 718	0.01 593	0.02 042
binary	0.03 552	0.01 765	0.01 808	0.02 367	0.02 383
Hadamard	0.02 074	0.02 074	0.01 409	0.01 094	0.01 487
binary	0.03 378	0.03 975	0.05 221	0.05 055	0.03 500
Hadamard	0.01 246	0.01 589	0.03 883	0.03 635	0.01 973
binary	0.05 348	0.02 965	0.02 359	0.03 516	0.04 437
Hadamard	0.02 591	0.01 208	0.02 812	0.03 521	0.02 753
binary	0.04 272	0.03 249	0.03 583	0.03 280	0.03 718
Hadamard	0.02 916	0.02 266	0.02 412	0.03 368	0.02 739
binary	0.03 261	0.02 625	0.02 349	0.03 346	0.03 200
Hadamard	0.03 516	0.03 348	0.02 037	0.01 619	0.02 677
			average binary		0.03 321
			average Hadamard		0.02 279

Table 2.2
Average mean of estimation error, 100 training vectors
2 - dimensional input data, 1000 vectors for testing

output node	1	2	3	4
binary	0.00 349	-0.00 523	0.00 110	0.00 447
Hadamard	-0.00 260	-0.00 185	0.00 286	0.00 513
binary	0.00 490	-0.00 068	-0.00 385	-0.00 055
Hadamard	0.00 056	0.00 765	-0.01 332	0.00 899
binary	0.00 334	-0.00 306	-0.01 186	-0.00 256
Hadamard	0.00 386	0.00 187	-0.00 148	-0.00 170
binary	-0.00 071	0.00 217	0.00 221	-0.00 435
Hadamard	-0.02 314	0.05 612	-0.02 779	-0.02 740
binary	0.00 346	-0.01 080	0.03 816	-0.01 832
Hadamard	-0.00 090	-0.00 079	0.00 564	0.00 229
binary	-0.00 143	0.00 317	-0.00 537	-0.00 124
Hadamard	0.00 503	-0.00 031	-0.00 320	-0.00 099

output node	5	6	7	8	average mean
binary	-0.00 193	0.00 213	0.00 165	-0.00 698	-0.00 016
Hadamard	-0.00 139	0.00 122	-0.00 377	0.00 094	0.00 007
binary	-0.00 106	0.00 746	-0.00 336	-0.00 280	0.00 001
Hadamard	0.00 038	0.00 155	0.00 299	-0.00 637	0.00 030
binary	0.03 042	-0.02 309	-0.00 894	0.01 647	0.00 009
Hadamard	0.00 231	-0.00 428	0.00 069	-0.00 179	-0.00 006
binary	-0.00 617	0.00 591	0.00 111	-0.00 200	-0.00 023
Hadamard	0.03 791	-0.04 018	0.03 365	-0.01 537	-0.00 078
binary	-0.00 494	0.02 307	-0.02 754	0.01 369	0.00 210
Hadamard	0.00 380	-0.00 473	-0.00 671	0.00 220	0.00 010
binary	0.00 659	-0.00 185	0.00 257	-0.00 342	-0.00 012
Hadamard	-0.00 270	0.00 382	0.00 180	-0.00 408	-0.00 008
				average binary	0.00 028
				average Hadamard	-0.00 007

The predicted results are not reached. However, the error variance and the mean square error do reduce by approximately 30-40 %. Each node though achieves a different value. Also, the mean error does not vanish. Figure 2.4 shows a detailed plot of the output error

distribution at each node. Apparently, the output error at the output of the Hadamard-transformed network before taking the inverse Hadarnard-transform is larger than the error at the output of the binary output.

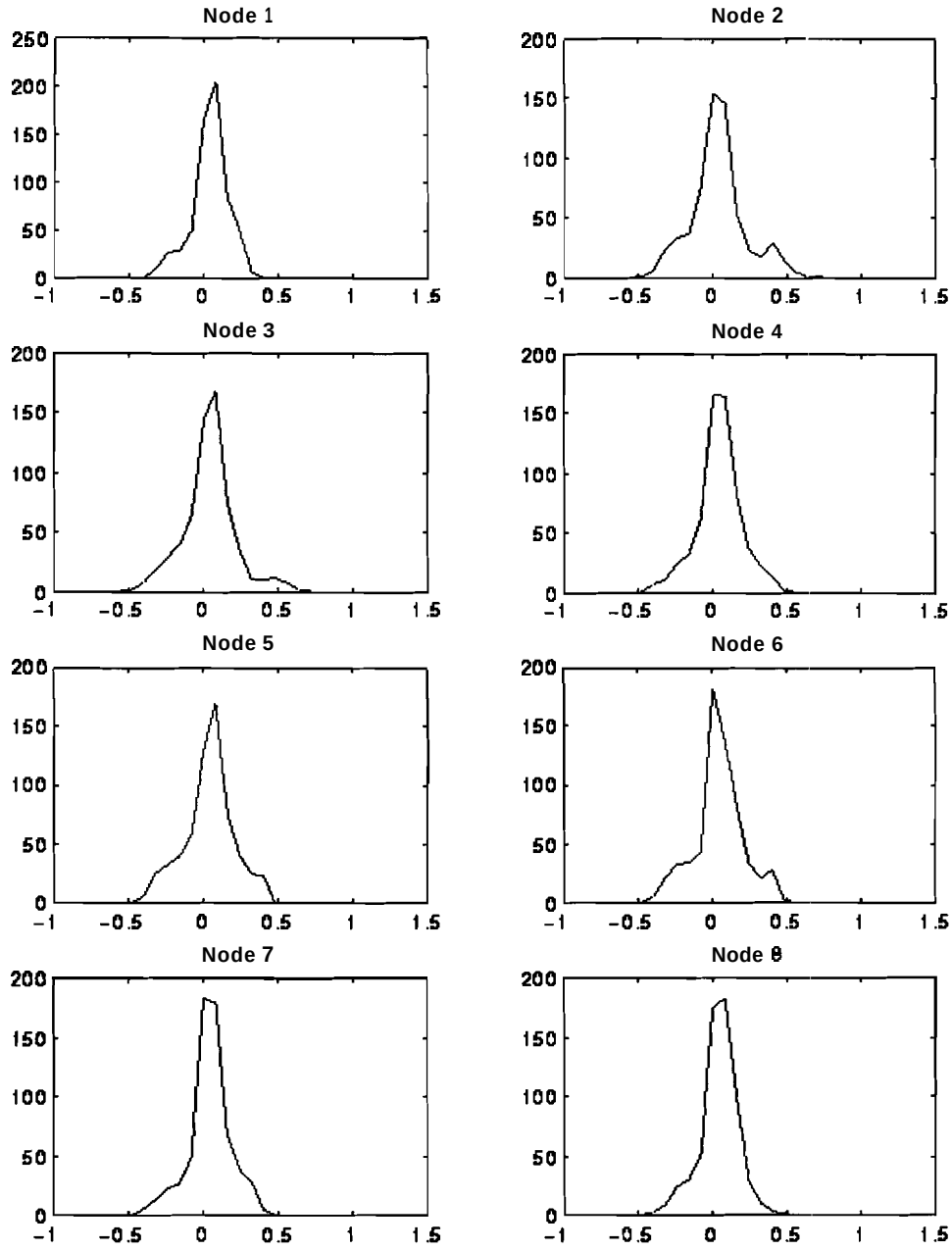

Fig. 2.4 **(a)** Error distribution at the output nodes for the binary output representation

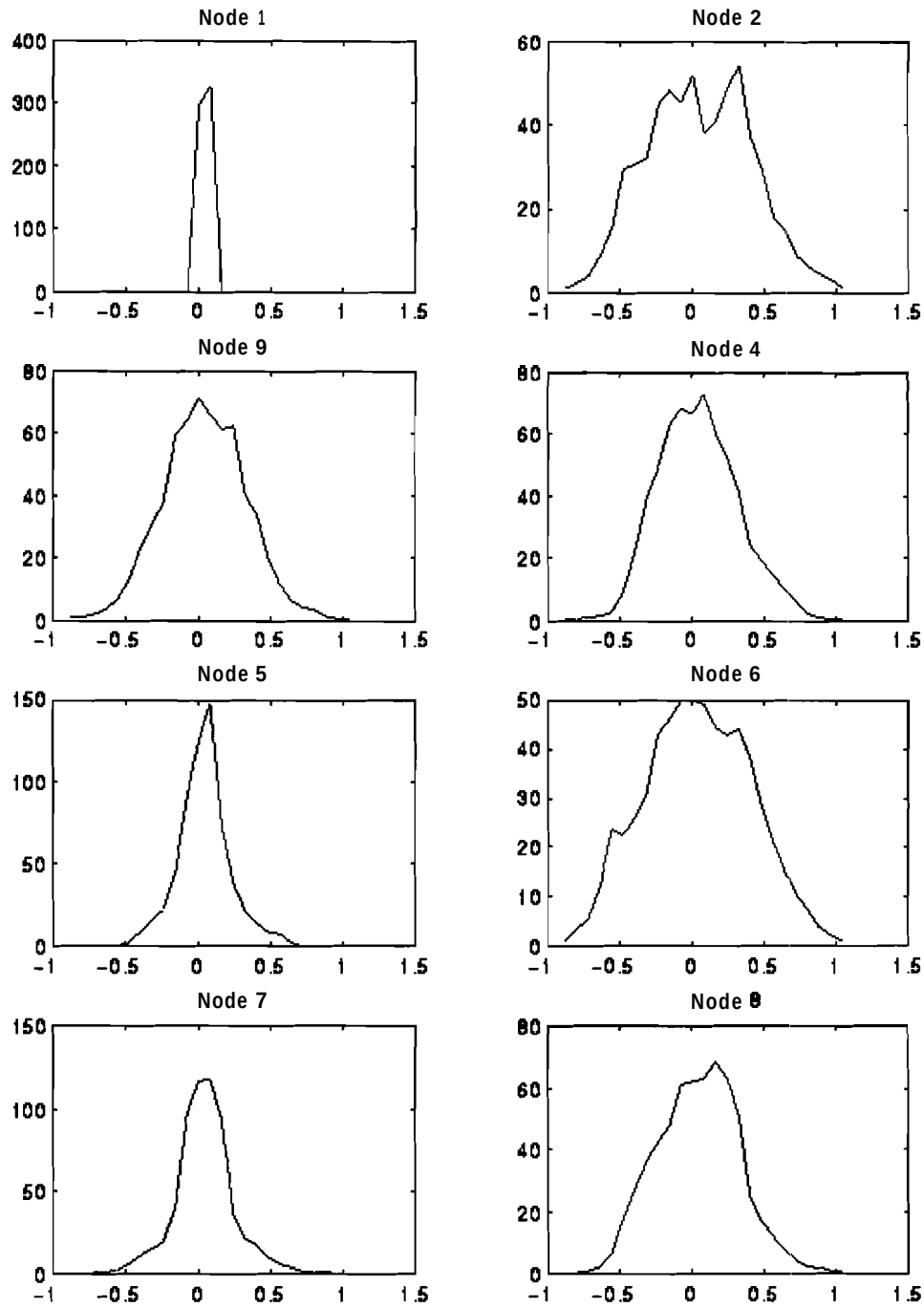

Fig. 2.4 (b) Error distribution at the output nodes before the inverse Hadamard-transform

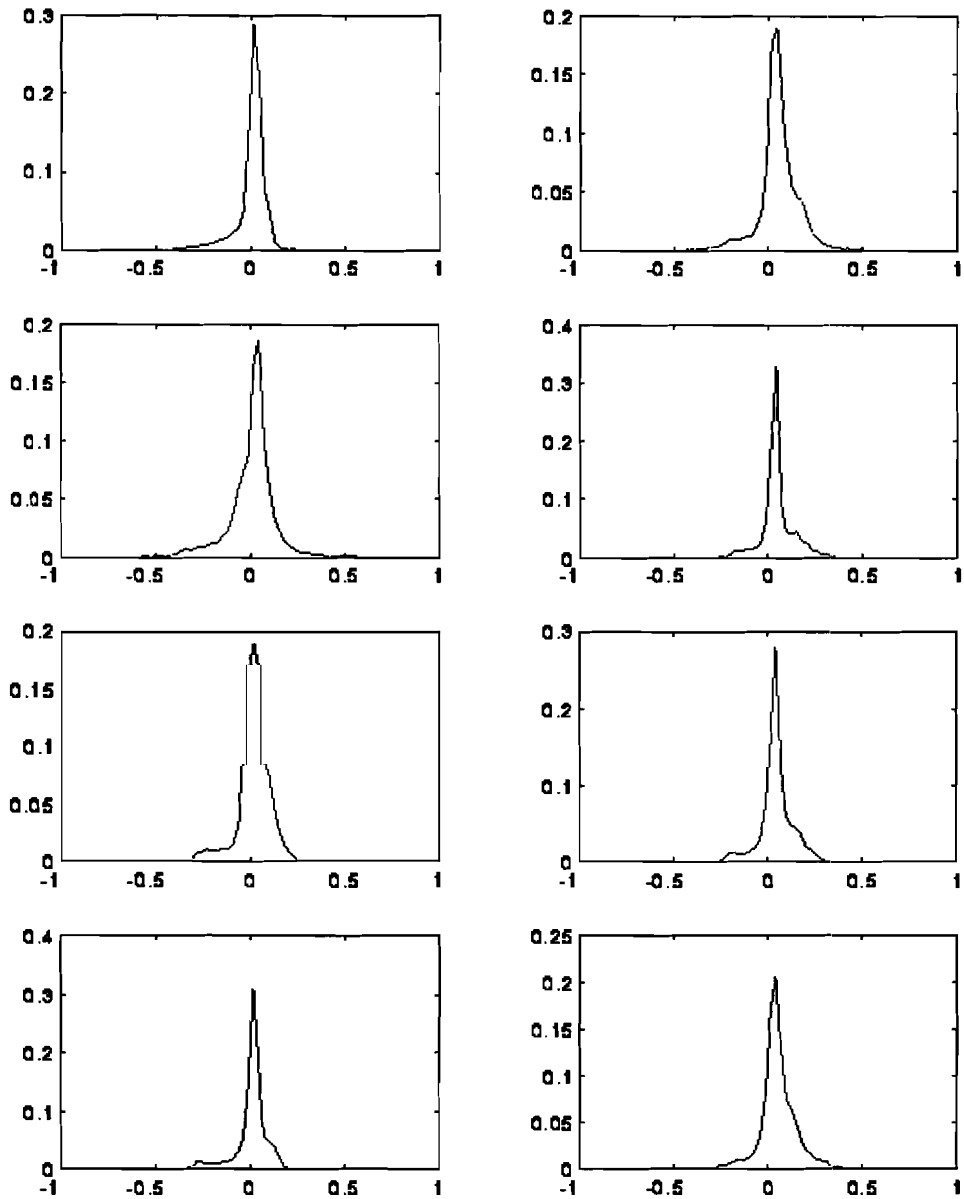

Fig. 2.4 (c) Output error distribution after the inverse Hadamard-transform

2.3.4 Conclusions

Figure 2.5 shows the Hadamard-transformed network and the measurement points for the 2 different errors.

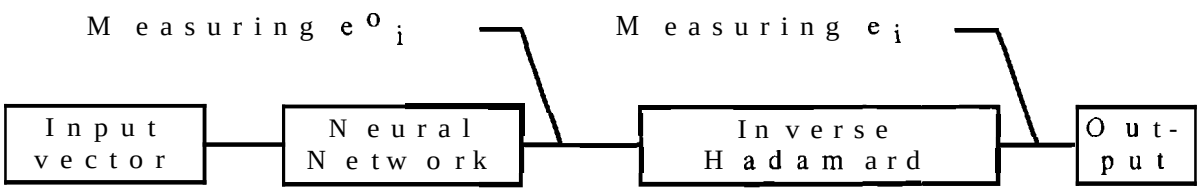

Fig. 2.5: The measured errors at different points

The probability P(C (Xi) was computed for each vector. The estimated $\hat{P}(C|X_i)$ was then compared with the correct one, and the error

$$e_i = \hat{P}(C|X_i) - P(C|X_i) \qquad (2.26)$$

was calculated. In the binary case, using 0 and 1 as desired outputs, the values of the output nodes are the a posteriori probability estimations. The estimation error made then equals e^o_i. Figure 2.4 (a) shows the sampled error distributions at the output nodes of the binary output neural network.

For the binary network, the output error e^o_i is equal to the probability error e_i since the inverse Hadamard-transform is missing.

For the Hadamard-transformed output, we compared the calculated inverse Hadamard-transform of the outputs, using equations (2.9) and (2.11):

$$P(C|Xi) = H^{-1} O(X_i) - H^{-1} e^o_i = H^{-1} O(X_i) - e_i \qquad (2.27)$$

$$\mathbf{e_i = H^{-1}\ O(X_i)} - P(C \mid Xi) \tag{2.28}$$

The sampled error distributions of all e_{ji} are shown in Figure 2.4 (c)

We also Hadamard-transformed the computed a posteriori probabilities and compared them directly with the neural network output (2.3):

$$\mathbf{e^o_i = O(X_i)} - H\ P(C \mid Xi) \tag{2.3}$$

However, $\mathbf{O(X_i)}$ now ranges from -1 to 1, so to obtain the error between the estimation of the probability of the output at node j equaling one ($\hat{P}(o_{ji}=1 \mid \mathbf{X_i})$) and the true probability $\mathbf{P}(o_{ji}=1 \mid X)$, we have to scale [2] by 0.5 and shift by adding 0.5, giving us:

$$\mathbf{e^o_{ip} = 0.5\ (O(X_i) + 1) - P(O_i=1 \mid X)} \tag{2.29}$$

where $\mathbf{P(O_i=1} \mid X)$ can be obtained by

$$\mathbf{P(O_i=1} \mid X) = 0.5\ (H\ P(C \mid Xi) + 1) \tag{2.30}$$

with $P(C \mid Xi)$ being the computed a posteriori probability in our specific example.
This implies:

$$\mathbf{2\ e^o_i = e^o_{ip}} \tag{2.31}$$

$\mathbf{e^o_{ip}}$ is the probability estimation error at the output nodes of the Hadarnard-transformed neural network and is shown in Figure 2.4 (b).

Assuming the relative error e^{o}_{ip} to be of the same range as the output error of the binary network we expect the mean to double and the variance to increase by a factor of 4 before taking the inverse Hadarnard-transform. The inverse Hadamard-transform uses the absolute error at the output, which is two times the relative error. Hence we can only expect a reduction of the variance by N/4, in our example 50%.

The distributions of these errors are similar to those of the binary representation, and but the variances are different. The variance for the output before the inverse Hadamard-transform is higher than the variance of the binary network. This shows, that like in our model, the reduction of the error variance is a result due to the inverse Hadamard-transform of the output, and not of better learning done by the Hadamard-transformed network..

2.4 Modification of the Model

The sampling of the density function of the error shows that one cannot really use the approximation of a zero mean error over all vectors. The distributions are approximately Gaussian. Also it is a rough approximation to assume the same variances for all output nodes. One would have to include the mean error in a more detailed model, since it does not totally vanish. It is usually higher for the Hadamard-transformed-output representation.

The experimental results show that we cannot justify all the assumptions we made in Section 2. Clearly, the limited sample size will produce a sample slightly different from the original distribution. For each component, the sample expectation and its variance are given by [12]:

$$E\{\bar{x}\} = E\{x\} = 0.5 \tag{2.32}$$

$$Var\{\bar{x}\} = \sigma_x^2 = \sigma^2/M \tag{2.33}$$

where M is the sample size, $\bar{x}$ the sample mean, σ_x^2 the variance of the sample mean and σ^2 is the variance of each component of the input data. In our case, the input data consists of 8 independent classes, each of them with 2 different clusters. This gives us 16 clusters of vectors. Each cluster has a covariance matrix of dimension 2. The covariance matrix for each cluster of our synthetic data is given by

$$\mathbf{S_{cluster}} = 1/900\ \mathbf{I} \tag{2.34}$$

where $\mathbf{I}$ is the identity matrix and $16 \geq k$.

Then, due to independence, the overall covariance matrix becomes [7]

$$\mathbf{S_{all}} = \sum_{k=1}^{L} \mathbf{S_{bk}} + \mathbf{S_{wk}} \tag{2.35}$$

where $\mathbf{S_b}$ is the in between scatter matrix , $\mathbf{S_w}$ is the within scatter matrix and L is the total number of clusters, equal to 2 N. N is the number of different classes.

According to [7], the within cluster scatter matrix is defined by:

$$\mathbf{S_{wk}} = \sum_{k=1}^{L} P(\text{Cluster}_k)\, E\{(\mathbf{X}-\mathbf{m_k})\,(\mathbf{X}-\mathbf{m_k})^t\} \mid \text{Cluster}_k\} \tag{2.36}$$

with $\mathbf{m_k}$ as the mean of each cluster. For our data, the cluster probability equals

$\frac{1}{2N} = \frac{1}{L}$ for all classes. This yields:

$$\mathbf{S_{wk}} = \frac{1}{L} \sum_{k=1}^{L} \mathbf{S_{cluster}} \tag{2.37}$$

With L = 16 and $\mathbf{S_{cluster}} = 11900\,\mathbf{I}$, we obtain:

$$\mathbf{S_{wk}} = \mathbf{S_{cluster}} = 1/900\ \mathbf{I} \tag{2.38}$$

The in between class scatter is $\mathbf{S_{bk}}$ is defined by [7]

$$\mathbf{S_{bk}} = \sum_{k=1}^{L} P(\text{Cluster}_k)\,(\mathbf{m_k} - \mathbf{m_0})\,(\mathbf{m_k} - \mathbf{m_0})^t \tag{2.39}$$

$$= \frac{1}{L} \sum_{k=1}^{L} (\mathbf{m_k} - \mathbf{m_0})\,(\mathbf{m_k} - \mathbf{m_0})^t$$

And, with the means used for our random data:

$$\mathbf{S_{bk}} = 0.8711\,\mathbf{I} \tag{2.40}$$

The overall covariance matrix then becomes:

$$\mathbf{S_{all}} = 0.9067\,\mathbf{I} \tag{2.41}$$

We can now compute the standard deviation for each component. For the 100 vector case we obtained $\sigma = 0.0181$, for the 1000 vector case $a = 0.00181$. This suggests that we have to expect some bias at the output as well, due to the limited sample mean.

The neural network is a highly nonlinear system. Hence, we cannot propagate the sample mean through it and expect the output to equal the observed mean.

We will now drop the assumption of an unbiased error. Using a biased estimation error and keeping up the assumption of independence, Equations (2.4) - (2.6) for the expected error, the output error covariance matrix and the expectations of the product of 2 different vectors and 2 different components, all before taking the inverse Hadamard-transform, become

$$E\{e^o\} = \eta_o$$

$$S^o = Var\{e^o_i\} = E\{(e^o_i-\eta_o)\,(e^o_i-\eta_o)^t\} \tag{2.42}$$

$$E\{e^o_i\ e^o_k\} = E\{e^o_i\}\ E\{e^o_k\} = \eta_o^2 \qquad i\neq k \tag{2.43}$$

$$E\{e^o_{ji}\ e^o_{mi}\} = E\{e^o_{ji}\}\ E\{e^o_{mi}\} = \eta^o_j\ \eta^o_m \qquad j\neq m \tag{2.44}$$

Eq. (2.11), the mean error after the inverse Hadamard-transform then becomes

$$E\{e_i\} = E\{H^{-1}\ e^o_i\} = H^{-1}\ E\{e^o_i\} = (H/N)\ \eta_o = \eta \tag{2.45}$$

where η^o_j η^o_m are the means of e^o_{ji} e^o_{mi}, respectively. Does our assumption of independence hold? Assuming independence, but using the biased estimate, we obtain the covariance matrix **S** after the inverse Hadamard-transform as

$$\begin{aligned} S &= E\{(e_i - \eta)\,(e_i - \eta)^t\} \\ &= [\,E\{[H\,(e^o_i - \eta_o)]\ [H\,(e^o_i - \eta_o)]^t\}\,]\,/\,N^2 \end{aligned} \tag{2.46}$$

We are now interested in the variance for each output value after the inverse Hadarnard-transform, since this is the important term for the accuracy of the probability density estimation. In order to obtain the error variance of each output value explicitly, we will use the scalar notation. We obtain with N = 4:

$$\begin{aligned} e_{1i} &= (e^o_{1i} + e^o_{2i} + e^o_{3i} + e^o_{4i})/4 \\ e_{2i} &= (e^o_{1i} - e^o_{2i} + e^o_{3i} - e^o_{4i})/4 \\ e_{3i} &= (e^o_{1i} + e^o_{2i} - e^o_{3i} - e^o_{4i})/4 \end{aligned} \tag{2.47}$$

$$e_{4i} = (e^o_{1i} - e^o_{2i} - e^o_{3i} + e^o_{4i})/4$$

$$e_{1i}^2 = (e^o_{1i} + e^o_{2i} + e^o_{3i} + e^o_{4i})^2)/16 \qquad (2.48)$$

$$= (\sum_{j=1}^{4} e^o_{ji}{}^2 + e^o_{1i}\, e^o_{2i} + e^o_{1i}\, e^o_{3i} + e^o_{1i}\, e^o_{4i} + e^o_{2i}\, e^o_{1i} + e^o_{2i}\, e^o_{3i} +$$

$$e^o_{2i}\, e^o_{4i} + e^o_{3i}\, e^o_{1i} + e^o_{3}e^o_{2i} + e^o_{3i}\, e^o_{4i} + e^o_{4i}\, e^o_{1i} + e^o_{4i}\, e^o_{2i} +$$

$$e^o_{4i}\, e^o_{3i})/16$$

We will use a constant K_{ri} for the respective product terms of each component now:

$$e_{1i}^2 = (\sum_{j=1}^{4} e^o_{ji}{}^2 + K_{1i})/16$$

$$e_{2i}^2 = (e^o_{1i} - e^o_{2i} + e^o_{3i} - e^o_{4i})^2/16$$

$$= (\sum_{j=1}^{4} e^o_{ji}{}^2 - e^o_{1i}\, e^o_{2i} + e^o_{1i}\, e^o_{3i} - e^o_{1i}\, e^o_{4i} - e^o_{2i}\, e^o_{1i} - e^o_{2i}\, e^o_{3i} +$$

$$e^o_{2i}\, e^o_{4i} + e^o_{3i}\, e^o_{1i} - e^o_{3}e^o_{2i} - e^o_{3i}\, e^o_{4i} - e^o_{4i}\, e^o_{1i} + e^o_{4i}\, e^o_{2i} -$$

$$e^o_{4i}\, e^o_{3i})/16$$

$$= (\sum_{j=1}^{4} e^o_{ji}{}^2 + K_{2i})/16$$

$$e_{3i}^2 = (e^o_{1i} + e^o_{2i} - e^o_{3i} - e^o_{4i})^2/16 = (\sum_{j=1}^{4} e^o_{ji}{}^2 + K_{3i})/16$$

$$e_{4i}^2 = (e^o_{1i} - e^o_{2i} - e^o_{3i} + e^o_{4i})^2)/16 = (\sum_{j=1}^{4} e^o_{ji}{}^2 + K_{4i})/16$$

Now, if we take the expectation of the equations (2.47) we obtain

$$E\{e_{1i}\} = (E\{e^o_{1i}\} + E\{e^o_{2i}\} + E\{e^o_{3i}\} + E\{e^o_{4i}\})/4 \qquad (2.49)$$
$$= (\eta^o_1 + \eta^o_2 + \eta^o_3 + \eta^o_4)/4$$

$$E\{e_{2i}\} = (E\{e^o_{1i}\} - E\{e^o_{2i}\} + E\{e^o_{3i}\} - E\{e^o_{4i}\})/4$$
$$= (\eta^o_1 - \eta^o_2 + \eta^o_3 - \eta^o_4)/4$$

$$E\{e_{3i}\} = (E\{e^o_{1i}\} + E\{e^o_{2i}\} - E\{e^o_{3i}\} - E\{e^o_{4i}\})/4$$
$$= (\eta^o_1 + \eta^o_2 - \eta^o_3 - \eta^o_4)/4$$

$$E\{e_{4i}\} = 9E\{e^o_{1i}\} - E\{e^o_{2i}\} - E\{e^o_{3i}\} + E\{e^o_{4i}\})/4$$
$$= (\eta^o_1 - \eta^o_2 - \eta^o_3 + \eta^o_4)/4$$

Squaring yields:

$$E^2\{e_{1i}\} = (\eta^o_1 + \eta^o_2 + \eta^o_3 + \eta^o_4)^2/16 \qquad (2.50)$$
$$= (\sum_{j=1}^{4} \eta^o_j{}^2 + \eta^o_1\eta^o_2 + \eta^o_1\eta^o_3 + \eta^o_1\eta^o_4 + \eta^o_2\eta^o_1 + \eta^o_2\eta^o_3 + \eta^o_2\eta^o_4 +$$
$$\eta^o_3\eta^o_1 + \eta^o_3\eta^o_2 + \eta^o_3\eta^o_4 + \eta^o_4\eta^o_1 + \eta^o_4\eta^o_2 + \eta^o_4\eta^o_3)/16$$

Similar to the term K_r for the product terms in Eq.(2.48), we use constant A_r for the product terms now:

$$[E\{e_{1i}\}]^2 = (\sum_{j=1}^{4} \eta^o_j{}^2 + A_1)/16$$

$$[E\{e_{2i}\}]^2 = (\eta^o_1 - \eta^o_2 + \eta^o_3 - \eta^o_4)^2/16$$

$$= (\sum_{j=1}^{4} \eta^o_j{}^2 - \eta^o_1\eta^o_2+\eta^o_1\eta^o_3-\eta^o_1\eta^o_4-\eta^o_2\eta^o_1-\eta^o_2\eta^o_3+\eta^o_2\eta^o_4+$$

$$\eta^o_3\eta^o_1-\eta^o_3\eta^o_2-\eta^o_3\eta^o_4-\eta^o_4\eta^o_1+\eta^o_4\eta^o_2-\eta^o_4\eta^o_3)/16$$

$$= (\sum_{j=1}^{4} \eta^o_j{}^2+A_2)/16$$

$$[E\{e_{3i}\}]^2 = (\eta^o_1 + \eta^o_2 - \eta^o_3 - \eta^o_4)^2/16$$

$$= (\sum_{j=1}^{4} \eta^o_j{}^2+A_3)/16$$

$$[E\{e_{4i}\}]^2 = (\eta^o_1 - \eta^o_2 - \eta^o_3 + \eta^o_4)^2/16$$

$$= (\sum_{j=1}^{4} \eta^o_j{}^2+A_4)/16$$

The expectations of equations (2.48) are:

(2.51)

$$E\{e_{1i}{}^2\} = [E\{(e^o_{1i} + e^o_{2i} + e^o_{3i} + e^o_{4i})^2\}]/16 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{1i}\}]/16$$

$$E\{e_{2i}{}^2\} = [E\{(e^o_{1i} - e^o_{2i} + e^o_{3i} - e^o_{4i})^2\}]/16 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{2i}\}]/16$$

$$E\{e_{3i}{}^2\} = [E\{(e^o_{1i} + e^o_{2i} - e^o_{3i} - e^o_{4i})^2\}]/16 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{3i}\}]/16$$

$$E\{e_{4i}{}^2\} = [E\{(e^o_{1i} - e^o_{2i} - e^o_{3i} + e^o_{4i})^2\}]/16 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{4i}\}]/16$$

Now, for independent errors e_{jk}, $E\{K_r\}$ equals A_r

With $E\{K_r\} = A_r$ we obtain the variance of each output node as

$$\sigma_r^2 = [E\{\sum_{j=1}^{4} e^o{}_{ji}^2\} - \sum_{j=1}^{4} \eta_j^2]/16 \qquad (2.52)$$

Our experimental results obtained with the training set are not equal to the value obtained with this formula, see Table **2.9.**

Hence, we have to drop the assumption of independence of the error over the nodes as well. We will still assume independence for the errors of different input vectors, since the system has no memory. The new variance of the r^{th} output value after the inverse Hadamard-transform is given by

$$\sigma_r^2 = E\{(e_{ri}-\eta_r)^2\} \qquad (2.53)$$

Since we no longer assume independence, the expectation of equations **(2.48)** becomes:

$$E\{e_{1i}^2\} = [E\{(e^o{}_{1i} + e^o{}_{2i} + e^o{}_{3i} + e^o{}_{4i})^2]/16 = [E\{\sum_{j=1}^{4} e^o{}_{ji}^2\} + E\{K_{1i}\}]/16 \qquad (2.54)$$

$$= [E\{\sum_{j=1}^{4} e^o{}_{ji}^2\} + E\{ e^o{}_{1i}\, e^o{}_{2i} + e^o{}_{1i}\, e^o{}_{3i} + e^o{}_{1i}\, e^o{}_{4i} + e^o{}_{2i}\, e^o{}_{1i} +$$

$$e^o{}_{2i}\, e^o{}_{3i} + e^o{}_{2i}\, e^o{}_{4i} + e^o{}_{3i}\, e^o{}_{1i} + e^o{}_{3i} e^o{}_{2i} + e^o{}_{3i}\, e^o{}_{4i} + e^o{}_{4i}\, e^o{}_{1i} +$$

$$e^o{}_{4i}\, e^o{}_{2i} + e^o{}_{4i}\, e^o{}_{3i} \}]/16$$

The correlation between two nodes is defined by

$$s^o_{jk} = E\{ e^o_{ji}\, e^o_{ki}\} - E\{ e^o_{ji}\}\, E\{ e^o_{ki}\} \tag{2.55}$$

This can be written as:

$$E\{ e^o_{ji}\, e^o_{ki}\} = s^o_{jk} + \eta^o_j \eta^o_k \tag{2.56}$$

This yields

$$\begin{aligned} E\{e_{1i}^2\} = [E\{ \sum_{j=1}^{4} e^o_{ji}{}^2\} &+ s^o_{12} + \eta^o_1\eta^o_2 + s^o_{13} + \eta^o_1\eta^o_3 + s^o_{14} + \eta^o_1\eta^o_4 + s^o_{21} + \\ &\eta^o_2\eta^o_1 + s^o_{23} + \eta^o_2\eta^o_3 + s^o_{24} + \eta^o_2\eta^o_4 + s^o_{31} + \eta^o_3\eta^o_1 + s^o_{32} + \\ &\eta^o_3\eta^o_2 + s^o_{34} + \eta^o_3\eta^o_4 + s^o_{41} + \eta^o_4\eta^o_1 + s^o_{42} + \eta^o_4\eta^o_2 + s^o_{43} + \\ &\eta^o_4\eta^o_3]/16 \end{aligned}$$

Using the previously defined constant A_1 and defining a constant T_1 for the sum of all s^o_{jk} yields

$$E\{e_{1i}^2\} = [E\{ \sum_{j=1}^{4} e^o_{ji}{}^2\} + A_1 + T_1]/16 \tag{2.57}$$

And, similarly for the other components, we get

$$E\{e_{2i}^2\} = [E\{(e^o_{1i} - e^o_{2i} + e^o_{3i} - e^o_{4i})^2]/16 = [E\{ \sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{2i}\}]/16$$

$$\begin{aligned} = [E\{ \sum_{j=1}^{4} e^o_{ji}{}^2\} &- s^o_{12} - \eta^o_1\eta^o_2 + s^o_{13} + \eta^o_1\eta^o_3 - s^o_{14} - \eta^o_1\eta^o_4 - s^o_{21} - \\ &\eta^o_2\eta^o_1 - s^o_{23} - \eta^o_2\eta^o_3 + s^o_{24} + \eta^o_2\eta^o_4 + s^o_{31} + \eta^o_3\eta^o_1 - s^o_{32} - \end{aligned}$$

$$\eta^o_3\eta^o_2 - s^o_{34} - \eta^o_3\eta^o_4 - s^o_{41} - \eta^o_4\eta^o_1 + s^o_{42} + \eta^o_4\eta^o_2 - s^o_{43} - \eta^o_4\eta^o_3]/16$$

$$= [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + A_2 + T_2]/16$$

$$E\{e_{3i}{}^2\} = [E\{(e^o_{1i} + e^o_{2i} - e^o_{3i} - e^o_{4i})^2\}]/16 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{3i}\}]/16$$

$$= [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + A_3 + T_3]/16$$

$$E\{e_{4i}{}^2\} = [E\{(e^o_{1i} - e^o_{2i} - e^o_{3i} + e^o_{4i})^2\}]/16 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + E\{K_{4i}\}]/16$$

$$= [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} + A_4 + T_4]/16$$

Now, using equations (2.57) and equations (2.56) with equation (2.53), we obtain

$$\sigma_r{}^2 = [E\{\sum_{j=1}^{4} e^o_{ji}{}^2\} - \sum_{j=1}^{4} \eta_j{}^{o\,2} + T_r]/16 \qquad (2.58)$$

The average variance is the sum of the variances over all the output nodes. Since $\sum_{j=1}^{N} T_j = 0.$, the average gain over all the output values after the inverse Hadamard-transform, is given by

$$\sigma_a{}^2 = \frac{1}{N}\sum_{r=1}^{N} \sigma_r{}^2 \qquad (2.59)$$

$$= \frac{1}{N^3} \sum_{r=1}^{N} [E\{ \sum_{j=1}^{4} e^{o}_{ji}{}^2 \} - \sum_{j=1}^{4} \eta_j{}^2]$$

$$= \frac{1}{N^2} [E\{ \sum_{j=1}^{4} e^{o}_{ji}{}^2 \} - \sum_{j=1}^{4} \eta_j{}^2]$$

The $\sigma_r{}^2$ can also be obtained using the matrix notation. Taking the inverse Hadamard-transform is a linear transform, where vector $\mathbf{e^o_i}$ is multiplied with $\mathbf{H}/N$. According to [7], and using the symmetry of the Hadamard matrix, we obtain:

$$\mathbf{S = (H\, S^o\, H\,) / N^2} \qquad (2.60)$$

where S is the covariance matrix after the inverse Hadamard-transform and $\mathbf{S^o}$ is the covariance matrix at the actual neural network output. Now in order to obtain the elements on the diagonal, $\sigma_r{}^2$ we would have to write Eq. (2.60) in component form, which will then yield the same results as Eq. 2.59.

2.5 Experiments with the More Detailed Model

Our experimental results, shown in Table 2.3, agree with Eqs. (2.58) and (2.59). Eq. 2.59 shows the variance of each output node, pertinent to individual classes. Since the dependency terms drop out, and the results for the average terms is similar to the case with independence but nonzero means.

Table 2.3
Predicted variance vs. actual variance

nodes	binary output	Output at node, e_i^o	Hadamard output e_i	**Eq.** (2.14)	**Eq.** (2.46)	Eq. (2.58)	constant T_j of Eq. (2.58)
1	0.01 409	0.00 043	0.00 918	0.02 043	0.02 042	0.00 918	-0.01 124
2	0.01 410	0.52 280	0.00 969	0.02 043	0.02 042	0.00 969	-0.01 073
3	0.02 185	0.10 613	0.02 102	0.02 043	0.02 042	0.02 102	0.00 060
4	0.02 981	0.11 102	0.01 773	0.02 043	0.02 042	0.01 773	-0.00 270
5	0.04 196	0.05 035	0.02 727	0.02 043	0.02 042	0.02 727	0.00 685
6	0.04 576	0.19 593	0.04 539	0.02 043	0.02 042	0.04 539	0.02 496
7	0.02 601	0.06 238	0.01 718	0.02 043	0.02 042	0.01 718	-0.00 325
8	0.02 135	0.25 810	0.01 593	0.02 043	0.02 042	0.01 593	-0.00 449
average	0.02 687	0.16 339	0.02 042	0.02 043	0.02 042	0.02 042	0.00 000

sample 2

nodes	binary output	Output at node, e_i^o	Hadamard output e_i	Eq. (2.14)	Eq. (2.46)	Eq. (2.58)	constant Tj of Eq. (2.58)
1	0.02 239	0.00 079	0.01 998	0.01 492	0.01 487	0.01 998	0.00 511
2	0.02 175	0.34 997	0.00 664	0.01 492	0.01 487	0.00 664	-0.00 824
3	0.02 246	0.09 681	0.00 824	0.01 492	0.01 487	0.00 824	-0.00 663
4	0.02 911	0.10 407	0.01 761	0.01 492	0.01 487	0.01 761	0.00 274
5	0.03 552	0.05 437	0.02 074	0.01 492	0.01 487	0.02 074	0.00 587
6	0.01 765	0.13 492	0.02 074	0.01 492	0.01 487	0.02 074	0.00 587
7	0.01 808	0.04 768	0.01 409	0.01 492	0.01 487	0.01 409	-0.00 079
8	0.02 367	0.16 319	0.01 094	0.01 492	0.01 487	0.01 094	-0.00 393
average	0.02 383	0.11 898	0.01 487	0.01 492	0.01 487	0.01 487	0.00 000

sample 3

nodes	binary output	Output at node, e_i^o	Hadamard output e_i	Eq. (2.14)	Eq. (2.46)	Eq. (2.58)	constant T_j of Eq. (2.58)
1	0.03 193	0.00 112	0.02 358	0.01 974	0.01 973	0.02 358	0.00 384
2	0.02 758	0.46 190	0.01 500	0.01 974	0.01 973	0.01 500	-0.00 474
3	0.02 014	0.09 153	0.00 774	0.01 974	0.01 973	0.00 774	-0.01 200
4	0.02 409	0.11 480	0.00 803	0.01 974	0.01 973	0.00 803	-0.01 171
5	0.03 378	0.05 625	0.01 246	0.01 974	0.01 973	0.01 246	-0.00 727
6	0.03 975	0.19 303	0.01 589	0.01 974	0.01 973	0.01 589	-0.00 384
7	0.05 221	0.05 276	0.03 883	0.01 974	0.01 973	0.03 883	0.01 909
8	0.05 055	0.29 160	0.03 635	0.01 974	0.01 973	0.03 635	0.01 661
average	0.03 500	0.15 787	0.01 973	0.01 974	0.01 973	0.01 973	0.00 000

sample 4

nodes	binary output	Output at node, e_i^o	Hadarnard output e_i	Eq. (2.14)	Eq. (2.46)	Eq. (2.58)	constant T_j of Eq. (2.58)
1	0.05 230	0.00 079	0.03 267	0.02 873	0.02 753	0.03 267	0.00 514
2	0.04 652	0.72 637	0.03 169	0.02 873	0.02 753	0.03 169	0.00 416
3	0.06 227	0.10 920	0.02 805	0.02 873	0.02 753	0.02 805	0.00 053
4	0.05 195	0.12 603	0.02 649	0.02 873	0.02 753	0.02 649	-0.00 104
5	0.05 348	0.08 665	0.02 591	0.02 873	0.02 753	0.02 591	-0.00 162
6	0.02 965	0.22 056	0.01 208	0.02 873	0.02 753	0.01 208	-0.01 544
7	0.02 359	0.10 203	0.02 812	0.02 873	0.02 753	0.02 812	0.00 059
8	0.03 516	0.39 005	0.03 521	0.02 873	0.02 753	0.03 521	0.00 768
average	0.04 437	0.22 021	0.02 753	0.02 873	0.02 753	0.02 753	0.00 000

sample 5

nodes	binary output	Output at node, e_i^o	Hadamard output e_i	Eq. (2.14)	Eq. (2.46)	Eq. (2.58)	constant Tj of Eq. (2.58)
1	0.03 947	0.00 039	0.02 779	0.02 740	0.02 739	0.02 779	0.00 041
2	0.04 480	0.69 640	0.02 691	0.02 740	0.02 739	0.02 691	-0.00 048
3	0.02 825	0.14 963	0.02 813	0.02 740	0.02 739	0.02 813	0.00 074
4	0.04 111	0.12 975	0.02 663	0.02 740	0.02 739	0.02 663	-0.00 076
5	0.04 272	0.08 617	0.02 916	0.02 740	0.02 739	0.02 916	0.00 178
6	0.03 249	0.26 044	0.02 266	0.02 740	0.02 739	0.02 266	-0.00 472
7	0.03 583	0.10 940	0.02 412	0.02 740	0.02 739	0.02 412	-0.00 326
8	0.03 280	0.32 056	0.03 368	0.02 740	0.02 739	0.03 368	0.00 629
average	0.03 718	0.21 909	0.02 739	0.02 740	0.02 739	0.02 739	0.00 000

sample 6

nodes	binary output	Output at node, e_i^o	Hadamard output e_i	Eq. (2.14)	Eq. (2.46)	Eq. (2.58)	constant Tj of Eq. (2.58)
1	0.03 573	0.00 034	0.03 298	0.02 678	0.02 677	0.03 298	0.00 620
2	0.03 724	0.71 044	0.03 139	0.02 678	0.02 677	0.03 139	0.00 462
3	0.02 865	0.09 579	0.02 035	0.02 678	0.02 677	0.02 035	-0.00 642
4	0.03 856	0.13 301	0.02 426	0.02 678	0.02 677	0.02 426	-0.00 252
5	0.03 261	0.07 426	0.03 516	0.02 678	0.02 677	0.03 516	0.00 839
	0.02 625	0.32 130	0.03 348	0.02 678	0.02 677	0.03 348	0.00 671
	0.02 349	0.05 920	0.02 037	0.02 678	0.02 677	0.02 037	-0.00 640
	0.03 346	0.31 917	0.01 619	0.02 678	0.02 677	0.01 619	-0.01 058
average	0.03 200	0.21 419	0.02 677	0.02 678	0.02 677	0.02 677	0.00 000

average variance over all samples

	0.03 321	0.18 229	0.02 279	0.02 300	0.02 279	0.02 279	0.00 000

2.6 Conclusions

In comparison to the 0-1 representation network our gain is only 30-45 %. Both networks learn towards a similar probability error. The Hadamard representation is shifted and scaled compared to the probability error. The error equals the probability error scaled by 2. Due to that, the mean of the error doubles, and the variance has to be

multiplied by a factor of 4. However, taking the inverse Hadamard-transform reduces the average variance by 1/N. For our experimental results N = 8, a maximum reduction of 50 % of the error variance can be expected.

One can expect that for problems with more classes than 8, the variance reduction gain will be larger, i.e. a reduction by 75 % for a 16 class problem.

Our experiments do not reach 50 %. Hadamard-transformed outputs force the output neurons to learn several decision borders, since the output has to be 1 for N/2 classes and -1 for the other N12 classes. This explains why we usually reach only 30-45 % reduction. On the other hand, the experimental mean error did not double like expected but only increased about 30 - 45 % .

2.7 Error Estimation of the Classifier without known A Posteriori Probabilities

2.7.1 Sum of all output values

The output of the binary network and the Hadamard-transformed network both estimate the a posteriori probabilities. So far, we could compute the estimation error of each input vector $\mathbf{X_i}$, since we knew the underlying data distributions. The input data always belongs to one of the classes. Then, since we estimate the :probability of the vector belonging to each of the possible 8 classes, the sum over all output values has to sum up to one.

$$\sum_{j=1}^{N} P(C_j | \mathbf{X_i}) = 1 \qquad (2.61)$$

Now, one measure for the accuracy of the density estimation will be if our probability estimates will sum up to 1 or not. Figure 2.6 will show two samples, where, for 80 testing vectors, we show the overall output value.

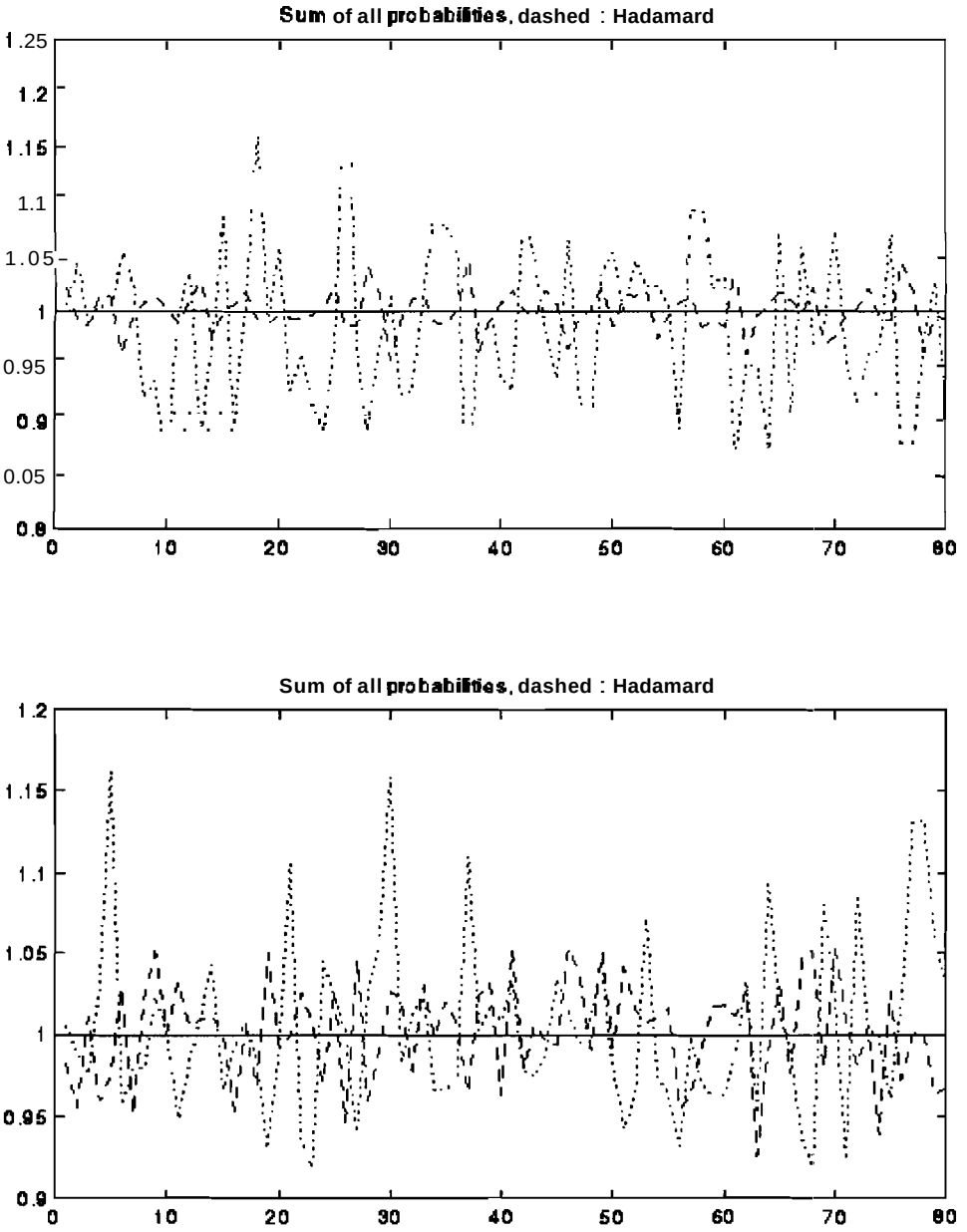

Fig. 2.6 Summation over all output nodes

dotted : binary network

dashed : Hadamard-transformed network

Clearly, the sum over all output nodes for the Hadamard-transformed neural network is much closer to 1, and oscillates less. Hence, the probability estimation is better.

2.7.2 Estimation of the mean of the output error

A short calculation will show that even without knowing the a posteriori probabilities of our testing set we can still obtain sum measurement for the mean estimation error. Let us first consider the output of the neural network to be the exact a posteriori probability. Then by taking the expectation over all testing vectors, we obtain:

$$E\{P(C_j|X_i)\} = \frac{1}{M}\sum_{i=1}^{M} P(C_j|X_i) = P(C_j) \tag{2.61}$$

Hence, for an ideal neural network estimator, we obtain the probability of the class $P(C_j)$ as expectation for the output node j. For a non-ideal neural network estimator, Equation (2.61) one changes to:

$$E\{\hat{P}(C_j|\mathbf{X}_i)\} = \frac{1}{M}\sum_{i=1}^{M} \hat{P}(C_j|\mathbf{X}_i) = \frac{1}{M}\sum_{i=1}^{M} [P(C_j|\mathbf{X}_i) + e_{ji}] \tag{2.62}$$

$$= \frac{1}{M}\sum_{i=1}^{M} P(C_j|\mathbf{X}_i) + \frac{1}{M}\sum_{i=1}^{M} e_{ji}$$

$$= P(C_j) + \eta_j$$

This gives

$$\eta_j = E\{\hat{P}(C_j|\mathbf{X})\} - P(C_j) \tag{2.63}$$

where $P(C_j)$ is the known probability of class j and the estimation is computed from the output of the neural network. If $P(C_j)$ is not known, we are not able to estimate each mean separately. However, we can estimate $E\{\sum_{j=1}^{N}\eta_j\}$. Taking the expectation of the sum of all equations (2.63), we obtain:

$$E\{\sum_{j=1}^{N} P(C_j) + \eta_j\} = 1 + E\{\sum_{j=1}^{N}\eta_j\} \tag{2.64}$$

Summing over all the output nodes yields

$$E\{\sum_{j=1}^{N}\eta_j\} = -1 + \sum_{j=1}^{N} E\{\hat{P}(C_j|\mathbf{X})\} \tag{2.65}$$

Table 2.4 compares the predicted mean error with the actual mean error for the 100 vector case.

Table 2.4
Approximated average mean error versus real mean error

sample #	1		2		3	
node	real	estimated	real	estimated	real	estimated
1	0.00 349	0.00 351	0.00 490	0.00 510	0.00 334	0.00 334
2	-0.00 523	-0.00 578	-0.00 068	-0.00 178	-0.00 306	-0.00 343
3	0.00 110	0.00 215	-0.00 385	-0.00 291	-0.01 186	-0.01 189
4	0.00 447	0.00 346	-0.00 055	0.00 110	-0.00 256	-0.00 279
5	-0.00 193	-0.00 004	-0.00 106	-0.00 209	0.03 042	0.03 034
6	0.00 213	0.00 215	0.00 746	0.00 637	-0.02 309	-0.02 222
7	0.00 165	0.00 092	-0.00 336	-0.00 213	-0.00 894	-0.00 987
8	-0.00 698	-0.00 770	-0.00 280	-0.00 360	0.01 647	0.01 723

3. COMPARISON BETWEEN STATISTICAL METHODS AND NEURAL NETWORKS

3.1 Introduction

There exist two major statistical non parametric probability density estimation techniques - histogramming and Parzen density estimation. Histogramming is probably the easiest, but the Parzen density estimation is more accurate. In this chapter, we will first introduce the two methods and provide the approximations formulas for adjusting the respective parameters. When we refer to nonparametric density estimator this means that, instead of assuming a certain distribution and estimating its parameters like variance, mean etc., we estimate the whole function numerically and generate a lookup table in which we store the estimated distributions. There are no assumptions made of the underlying probability.

In Section **3.4,** we compare the results achieved by the binary neural network, the Hadamard-transformed neural network, histogramming and the Parzen Density estimation. Also, training times, testing times and memory needs of the different algorithms are investigated. Section 3.5 provides the conclusions.

3.2 Histogramming

Histogramming is the oldest known method for probability density estimation. Classical histograms consist of nonoverlapping intervals, the bins. The density function of the histogram is then obtained by dividing the number of points fallen in one bin by the total number of points. The actual probability mass is then the product of the binwidth with the binvalue.

The problem of the appropriate binwidth selection is treated well in [8]. Clearly, if we choose the binwidth to be large, we get only a very rough approximation of the density function. Small features will be oversmoothed. On the other hand, for a small binwidth, we will obtain arbitrary oscillations in regions with few points. The problem of

binwidth selection is equal to the problem of the number of bins to use, since the binwith for the classical histogram used here is 1/number of bins.

[8] derives the following formula for the optimal number of bins for each dimension, which minimizes the asymptotic mean integral square error, AMISE:

$$\text{number of bins} = \sqrt[3]{2M} \tag{3.1}$$

A histogram using this formula, where M is the total number of vectors, should be optimally smoothed.

3.3 Parzen Density Estimation

In the Parzen density estimation, the estimate at one point is obtained not by simply counting the number of points but by averaging over the neighboring points as well. The value at one point is obtained from the kernel function of the region

$$\hat{f}(x) = \frac{1}{hL}\sum_{l=1}^{L} K\left(\frac{x - x_l}{h}\right) \tag{3.2}$$

where K is the Kernel function and h is the smoothing parameter. L is the number of points within the Kernel. In our case, we used normal Kernel, which has infinite support. Hence L equals the total number of vectors M of the training set. There have been proposed many different Kernel function, like a uniform normal, triangular or a combination of several functions [7], [8]. In this section, we will restrict to a normal Kernel with variance equal to one [8]. For the one dimensional case, an optimal kernel can be derived. For higher dimensions, where a product Kernel is used, we can only estimate an optimal h_i for each dimension i. For the normal Kernel used in our experiments, an approximation formula minimizing the AMISE is given by

$$h_j \approx \hat{\sigma}_j M^{-\frac{1}{d+4}} \tag{3.3}$$

M is the training sample size and d is the number of dimensions [Scott 92]. $\hat{\sigma}_j$ is the estimated standard deviation of the training data. It is estimated from the sample by computing the following for each dimension of the input data:

$$\hat{\sigma}_j 2 = \frac{\sum X^2}{M} - \left(\frac{\sum X}{M}\right)^2 \tag{3.4}$$

If we cannot assume an underlying distribution, it will become very difficult to derive an estimation formula from the AMISE. Heuristic approaches are equally different, since the underlying distributions are not known. As an illustration we applied the Parzen density estimation with several different smoothing parameters h to the data used in Chapter **2.3.** The resulting **P(X)** is shown in Figure **3.1.**

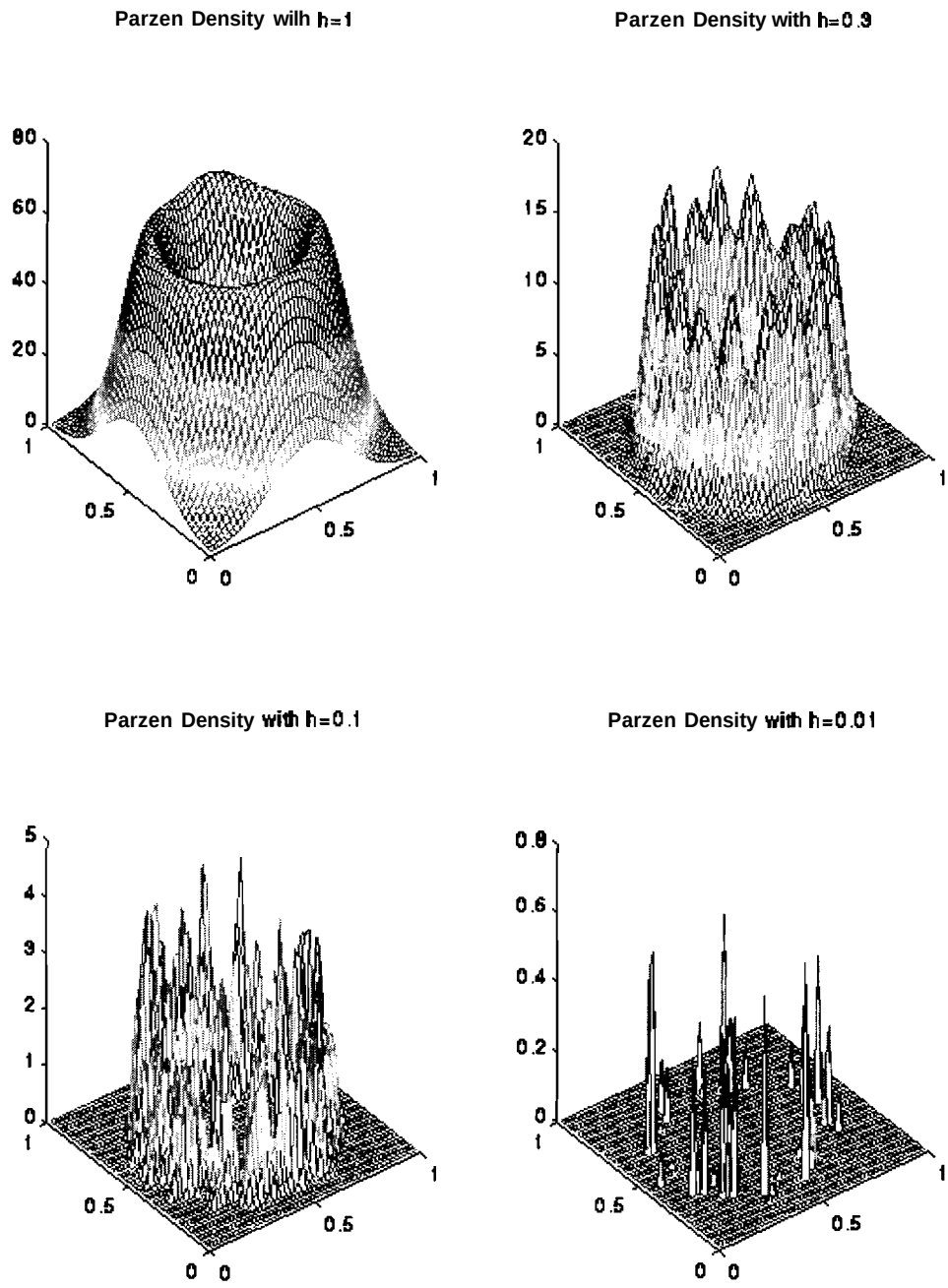

Fig. 3.1. Estimation of $P(\mathbf{X})$, using the Parzen method with different smoothing parameters.

3.4 Experiments

We first ran a series of examples with 1000 training vectors per class. We then reduced the size of the training set to 100 vectors per class for a second set of simulations. The results for classification and probability estimation with the present method were studied comparatively with the methods of histogramming and the Parzen density estimation.

In a second set of experiments, we used 3-dimensional data as input with 300 vectors per class as data set. We had to increase the number of training sweeps from 1000 to 3000, and the number of hidden neurons from 15 to 25, since the data was more complicated.

Since there are quite a lot of bins in the histogram where no vectors occurred during the estimation, we set them to -1 and counted every testing vector falling in such bins as misclassified. We excluded those vectors for the calculation of the mean error and the variance of the estimation, since we could not assign a specific error to them. Those regions would be very large in the 3-D case. Hence we restricted histogramming to the 2 - D case.

In the 2 and dimensional case we obtained $n_{opt} = 676$ bins for the 1000 vector case. Tables 1, 2 and 3 show the achieved classification, the mean error and the variances for the 2-D case, respectively. The results are shown for 3 different histograms, with 100 bins, n_{opt} and 2500 bins.

The histogramming method performed the worst in classification, especially when using the small data set for density estimation. The neural networks performed better than the histograrnming method, especially if we use the smaller training set of 100 vectors per class. In this case, histogramming is useless, since the distribution is sampled inaccurately, and there are not enough samples in the regions of low probability to sample them accurately. The Parzen method performed better then histogramming, but it did not reach the classification performance of the Hadamard-transformed neural network. For the average probability estimation performance, the Parzen method with h_{opt} performs the best, yielding a smaller bias and a much smaller mean variance than the neural networks. The Parzen density estimations though depend highly on the choice

of the smoothing parameter h. For a non-optimal h, it yields results inferior to those of the neural network.

The result of a better classification despite a higher estimation error may be related to the fact that neural networks approximate the decision boundaries continuously, whereas for the statistical methods we had to use the method of bilinear interpolation. Another reason is that the mean is more influenced by a small number of vectors which are misclassified with a huge error than by small errors. However, when the small errors are made in regions of high vector density and near a decision boundary, the classification performance is affected quite strongly. So, the regions where the errors occur becomes equally important to the error itself. The region of error does not influence the mean error or the mean variance.

Neural networks using the mean square error as error function do not approximate the a posteriori probabilities in regions with low probability well either [3]. If there is special interest in those regions, one can use importance sampling [13].

In the 3-dimensional case, the estimation errors of the Parzen density with h_{opt} and the Hadarnard-transformed neural network perform approximately equally, as shown in Tables 3.1, 3.4 and 3.5.

Table 3.1 (a)
Correct testing classification, using 100 vectors for training

sample no.	1	2	3	4	5	6
binary repr.	91.96%	92.38%	93.89%	92.66%	89.79%	92.65%
Hadamard repr.	92.76%	93.98%	94.23%	93.75%	92.89%	93.16%
Parzen density h = 1	56.14%	47.28%	38.35%	42.28%	54.26%	39.13%
Parzen density $h = h_{opt}$	91.80%	91.53%	91.38%	91.65%	92.10%	91.26%
Parzen density h = 0.0 1	90.20%	90.54%	90.78%	91.06%	91.38%	90.63%
Histogram 100 bins	52.85%	56.01%	54.38%	53.50%	56.29%	54.63%
Histogram 676 bins	65.16%	65.30%	61.21%	66.89%	64.95%	64.75%
Histogram 2500 bins	89.56%	87.74%	88.76%	89.28%	87.95%	88.11%
max. possible class.	95.20%	94.91%	95.05%	94.96%	94.91%	94.90%

Table 3.1 (b)
Correct testing classification, using 1000 vectors for training
2-dimensional input data

sample no,	1	2	3	4	5	6
binary repr.	93.04%	92.76%	92.45%	93.06%	93.80%	93.79%
Hadamard repr.	94.50%	90.90%	92.50%	94.60%	94.20%	94.60%
Parzen density h = 1	62.85%	72.39%	72.29%	67.10%	70.91%	71.33%
Parzen density h = h_{opt}	91.90%	91.99%	92.40%	92.48%	91.99%	92.56%
Parzen density h = 0.0 1	91.68%	91.91%	92.23%	92.43%	91.83%	92.53%
Histogram 100 bins	68.19%	67.04%	66.89%	67.29%	66.95%	67.86%
Histogram 676 bins	88.84%	88.55%	88.30%	87.84%	88.40%	88.49%
Histogram 2500 bins	92.73%	92.48%	92.94%	92.93%	92.64%	92.84%
max. possible class.	95.20%	94.91%	95.05%	94.96%	94.91%	94.90%

Table 3.1 (c)
Correct testing classification, using 300 vectors for training
3-dimensional input data

sample no,	1	2	3	4	5	6
binary repr.	87.60%	84.20%	89.40%	88.90%	90.80%	88.00%
Hadamard repr.	91.89%	90.75%	90.96%	93.14%	92.83%	90.83%
Parzen density h = 1	67.12%	---	---	---	---	---
Parzen density h = h_{opt}	81.86%	---	---	---	---	---
Parzen density h = 0.0 1	90.80%	---	---	---	---	---
max. possible class.	95.30%	94.90%	94.50%	95.00%	94.30%	94.70%

Table 3.2 (a)
Mean of the estimation error for the 100 vector case

sample 1

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	-0.04 099	-0.00 519	-0.00 030	-0.00 594	-0.00 647	0.05 138	0.08 995	0.35 012
2	0.02 693	0.00 663	0.00 065	0.00 190	0.00 027	0.12 996	0.13 333	0.36 016
3	0.01 494	-0.00 213	-0.00 013	0.00 360	0.00 503	0.09 926	0.13 840	0.36 406
4	-0.03 853	-0.00 003	0.00 050	-0.00 544	-0.00 689	0.11 137	0.12 703	0.36 062
5	0.02 136	-0.00 320	-0.00 192	-0.00 655	-0.00 722	0.09 268	0.08 395	0.35 337
6	0.01 293	0.00 200	0.00 034	0.00 812	0.00 753	0.10 005	0.14 195	0.35 935
7	-0.02 683	0.00 452	0.00 085	0.00 037	0.00 192	0.10 732	0.15 252	0.36 489
8	0.03 056	-0.00 226	0.00 002	0.00 394	0.00 5821	0.12 762	0.11 9371	0.35 561

sample 2

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	-0.00 247	0.00 987	0.00 125	0.00 564	0.00 457	0.17 620	0.14 037	0.36 955
2	0.00 444	0.00 173	-0.00 163	-0.00 079	-0.00 158	0.19 175	0.17 788	0.37 379
3	0.00 358	0.00 129	0.00 173	0.00 239	0.00 301	0.17 709	0.21 669	0.37 307
4	-0.00 647	-0.00 559	-0.00 153	-0.00 489	-0.00 622	0.19 569	0.16 265	0.36 151
5	0.00 543	0.00 580	0.00 202	0.00 581	0.00 798	0.15 387	0.14 425	0.36 790
6	-0.00 042	-0.00 571	-0.00 056	-0.00 256	-0.00 225	0.17 774	0.17 557	0.36 669
7	-0.00 880	0.00 306	-0.00 099	-0.00 011	-0.00 066	0.17 595	0.18 828	0.36 215
8	0.00 490	-0.01 084	-0.00 029	-0.00 550	-0.00 485	0.18 907	0.17 195	0.35 724

sample 3

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	-0.01 091	-0.00 599	-0.00 029	-0.00 339	-0.00 463	0.19 295	0.08 513	0.37 344
2	0.00 738	0.00 302	-0.00 091	0.00 183	0.00 132	0.23 722	0.16 392	0.36 832
3	-0.00 113	0.00 393	0.00 035	0.00 142	0.00 158	0.21 123	0.17 535	0.38 041
4	-0.00 080	-0.00 469	-0.00 027	0.00 109	0.00 319	0.23 623	0.15 785	0.36 345
5	0.00 224	0.00 232	0.00 051	-0.00 445	-0.00 824	0.19 229	0.10 410	0.36 403
6	-0.00 665	-0.00 415	-0.00 045	-0.00 085	0.00 031	0.22 922	0.14 820	0.36 912
7	-0.00 004	0.00 223	-0.00 006	0.00 360	0.00 565	0.19 973	0.17 410	0.36 828
8	0.00 978	0.00 384	0.00 111	0.00 074	0.00 082	0.23 245	0.15 774	0.37 119

sample 4

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 1676 bins	histogr. 2500 b.
1	0.00 535	0.00 069	0.00 110	0.00 417	0.00 363	0.13 114	0.05 333	0.36 586
2	-0.00 060	0.00 352	-0.00 034	0.00 065	0.00 122	0.18 739	0.08 861	0.36 263
3	-0.00 121	-0.00 493	-0.00 085	-0.00 048	-0.00 064	0.13 833	0.08 663	0.36 860
4	-0.00 364	0.00 351	0.00 078	-0.00 727	-0.00 693	0.15 085	0.08 491	0.37 183
5	-0.00 138	-0.01 128	0.00 060	0.00 600	0.00 688	0.14 534	0.04 235	0.37 404
6	0.00 784	0.01 106	0.00 045	-0.00 121	-0.00 418	0.16 660	0.09 568	0.36 568
7	-0.00 331	0.00 098	-0.00 063	0.00 178	0.00 381	0.14 900	0.08 587	0.37 006
8	-0.00 318	-0.00 317	-0.00 111	-0.00 363	-0.00 377	0.16 205	0.08 627	0.36 931

sample 5

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.00 688	0.01 229	-0.00 133	-0.00 112	-0.00 228	0.14 138	0.10 902	0.37 293
2	-0.00 379	-0.00 689	-0.00 179	-0.00 237	-0.00 339	0.18 386	0.15 174	0.37 128
3	-0.00 888	-0.00 979	-0.00 117	-0.00 132	-0.00 081	0.16 695	0.14 947	0.37 324
4	0.00 386	0.01 241	-0.00 406	-0.00 493	-0.00 512	0.18 244	0.13 651	0.36 415
5	0.00 384	-0.01 710	0.00 528	0.00 476	0.00 489	0.16 790	0.11 724	0.37 172
6	-0.00 292	0.01 921	0.00 142	0.00 275	0.00 358	0.20 557	0.15 955	0.37 676
7	-0.00 259	-0.00 819	-0.00 560	-0.00 590	-0.00 731	0.15 186	0.19 972	0.36 596
8	0.00 349	-0.00 168	0.00 723	0.00 813	0.01 043	0.19 300	0.15 629	0.37 362

sample 6

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.00 688	-0.00 297	0.00 068	-0.00 161	-0.00 093	0.12 156	0.05 961	0.35 974
2	-0.00 379	0.00 306	-0.00 023	-0.00 095	-0.00 251	0.17 434	0.10 840	0.36 549
3	-0.00 888	-0.00 151	-0.00 205	0.00 128	0.00 276	0.14 546	0.11 048	0.37 204
4	0.00 386	-0.00 024	-0.00 032	0.00 111	-0.00 076	0.18 123	0.10 148	0.36 660
5	0.00 384	0.00 517	0.00 226	0.00 217	0.00 194	0.12 968	0.06 770	0.35 849
6	-0.00 292	-0.00 305	-0.00 022	-0.00 175	-0.00 078	0.15 547	0.09 912	0.36 663
7	-0.00 259	0.00 033	-0.00 017	-0.00 090	0.00 113	0.15 469	0.08 930	0.36 289
8	0.00 349	-0.00 035	0.00 005	0.00 066	-0.00 085	0.15 568	0.09 568	0.35 645

average mean

-1.04E-05	3.29E-05	8.33E-13	5.21E-12	-8.75E-12	0.11611	0.07711	0.24550

Table3.2 (b)
Mean of the estimation error for the 1000 vector case
2 - dimensional input data

sample 1

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 1676 bins	histogr. 2500 b.
1	0.00 220	0.00 231	-0.00 014	0.01 075	0.01 112	0.03 215	0.01 835	0.02 119
2	-0.00 137	-0.00 016	0.00 047	-0.00 495	-0.00 502	0.07 777	0.01 483	0.02 372
3	-0.00 164	0.00 018	-0.00 013	-0.00 230	-0.00 246	0.06 256	0.01 081	0.02 463
4	0.00 228	-0.00 100	0.00 041	0.00 140	0.00 140	0.07 316	0.02 024	0.02 121
5	-0.00 272	0.00 010	-0.00 179	-0.00 220	-0.00 218	0.03 072	0.01 298	0.02 203
6	0.00 233	-0.00 095	0.00 026	0.00 171	0.00 181	0.07 648	0.01 427	0.02 406
7	0.00 002	0.00 119	0.00 082	-0.00 093	-0.00 123	0.06 560	0.01 288	0.02 007
8	-0.00 179	-0.00 150	0.00 011	-0.00 348	-0.00 344	0.06 671	0.01 164	0.02 446

sample 2

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.00 033	0.00 282	0.00 127	0.00 112	0.00 125	0.03 272	0.01 778	0.02 782
2	-0.00 037	-0.00 082	-0.00 169	-0.00 013	-0.00 044	0.07 964	0.01 617	0.02 822
3	0.00 276	0.00 191	0.00 141	-0.00 039	-0.00 033	0.06 863	0.01 385	0.02 690
4	-0.00 320	-0.00 142	-0.00 139	-0.00 088	-0.00 087	0.07 350	0.02 262	0.02 914
5	0.00 105	-0.00 199	0.00 202	0.00 044	0.00 037	0.03 450	0.01 204	0.02 875
6	-0.00 171	0.00 184	-0.00 042	-0.00 137	-0.00 137	0.07 461	0.01 649	0.02 705
7	0.00 031	-0.00 016	-0.00 102	0.00 102	0.00 124	0.07 169	0.01 719	0.02 767
8	0.00 023	-0.00 209	-0.00 018	0.00 020	0.00 014	0.07 428	0.01 932	0.02 661

sample 3

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 1676 bins	histogr. 2500 b.
1	0.00 035	-0.00 115	-0.00 001	0.00 040	0.00 070	0.09 096	0.01 392	0.02 298
2	-0.00 084	0.00 069	-0.00 091	-0.00 116	-0.00 152	0.09 216	0.01 187	0.02 257
3	0.00 111	0.00 103	0.00 018	0.00 131	0.00 160	0.06 756	0.01 253	0.02 188
4	-0.00 134	-0.00 176	-0.00 040	-0.00 161	-0.00 196	0.10 183	0.01 142	0.02 447
5	0.00 252	0.00 261	0.00 073	0.00 388	0.00 448	0.09 399	0.01 436	0.02 471
6	-0.00 098	0.00 034	-0.00 032	-0.00 153	-0.00 195	0.09 882	0.01 242	0.02 421
7	-0.00 109	-0.00 240	-0.00 012	-0.00 151	-0.00 165	0.06 501	0.00 934	0.02 451
8	0.00 026	0.00 072	0.00 084	0.00 023	0.00 030	0.10 081	0.01 311	0.02 440

sample 4

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.00 015	0.00 200	0.00 101	0.00 030	0.00 011	0.03 194	0.01 401	0.02 385
2	-0.00 091	-0.00 285	-0.00 040	-0.00 060	-0.00 016	0.06 688	0.01 204	0.02 372
3	-0.00 005	0.00 001	-0.00 081	-0.00 076	-0.00 101	0.06 333	0.01 618	0.02 577
4	-0.00 029	0.00 096	0.00 081	0.00 081	0.00 079	0.06 630	0.01 244	0.02 299
5	0.00 043	-0.00 095	0.00 042	0.00 001	0.00 009	0.02 659	0.01 340	0.02 472
6	-0.00 009	-0.00 014	0.00 033	-0.00 023	-0.00 028	0.06 771	0.01 701	0.02 466
7	0.00 050	0.00 136	-0.00 051	0.00 108	0.00 112	0.06 042	0.01 604	0.02 457
8	0.00 026	-0.00 043	-0.00 085	-0.00 061	-0.00 066	0.06 704	0.01 722	0.02 450

sample 5

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	-0.00 007	-0.00 121	0.00 076	0.00 024	0.00 045	0.02 378	0.02 329	0.01 499
2	0.00 080	0.00 061	0.00 001	-0.00 003	-0.00 020	0.06 443	0.01 901	0.00 802
3	-0.00 149	-0.00 224	-0.00 010	-0.00 123	-0.00 146	0.05 399	0.02 066	0.01 117
4	0.00 079	0.00 029	-0.00 162	0.00 036	0.00 064	0.06 200	0.02 179	0.01 168
5	0.00 018	0.00 307	0.00 058	0.00 043	0.00 038	0.01 888	0.01 985	0.01 243
6	0.00 064	-0.00 216	0.00 166	0.00 090	0.00 086	0.06 440	0.02 183	0.01 169
7	-0.00 109	-0.00 139	-0.00 147	-0.00 078	-0.00 072	0.05 677	0.02 045	0.01 106
8	0.00 024	0.00 313	0.00 017	0.00 011	0.00 005	0.05 340	0.01 920	0.00 941

sample 6

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.00 095	0.00 242	0.00 063	0.00 109	0.00 103	0.02 286	0.02 393	0.01 392
2	0.00 000	0.00 008	-0.00 002	0.00 082	0.00 100	0.06 212	0.02 184	0.01 658
3	0.00 059	0.00 116	-0.00 194	-0.00 090	-0.00 114	0.05 538	0.02 195	0.01 286
4	-0.00 230	-0.00 214	-0.00 023	-0.00 078	-0.00 059	0.06 165	0.02 279	0.01 186
5	0.00 112	-0.00 114	0.00 206	-0.00 057	-0.00 049	0.02 169	0.02 385	0.00 983
6	0.00 143	0.00 375	-0.00 030	0.00 294	0.00 283	0.06 194	0.02 142	0.01 676
7	-0.00 171	-0.00 261	-0.00 019	-0.00 316	-0.00 304	0.05 314	0.02 373	0.01 019
8	-0.00 008	-0.00 161	0.00 000	0.00 056	0.00 040	0.05 962	0.02 340	0.01 327

average mean

-1.93E-04	-4.70E-04	1.67E-04	-1.27E-04	5.31E-05	0.03848	0.01726	0.00938

Table 3.3 (a)
Variance of the estimation error for 1000 training vectors

sample 1

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.02 842	0.00 684	0.09 901	0.00 666	0.00 729	0.09 685	0.02 207	0.02 119
2	0.02 454	0.01 547	0.10 006	0.00 518	0.00 562	0.10 081	0.02 390	0.02 372
3	0.02 231	0.01 699	0.09 995	0.00 362	0.00 385	0.06 502	0.02 201	0.02 463
	0.02 085	0.00 675	0.10 031	0.00 428	0.00 449	0.10 176	0.02 729	0.02 121
	0.02 441	0.00 754	0.09 891	0.00 551	0.00 575	0.09 208	0.02 300	0.02 203
	0.02 481	0.00 638	0.10 030	0.00 671	0.00 714	0.10 524	0.02 319	0.02 406
	0.02 700	0.01 092	0.10 087	0.00 741	0.00 800	0.06 644	0.02 208	0.02 007
	0.02 971	0.01 006	0.10 001	0.00 713	0.00 774	0.09 908	0.01 978	0.02 446
average variance	0.02 526	0.01 012	0.09 993	0.00 581	0.00 623	0.09 091	0.02 291	0.02 267

sample 2

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 1676 bins	histogr. 2500 b.
1	0.02 753	0.01 789	0.10 056	0.00 530	0.00 532	0.09 617	0.02 568	0.02 782
2	0.03 402	0.02 721	0.09 792	0.00 373	0.00 383	0.10 542	0.02 236	0.02 822
3	0.04 076	0.04 803	0.10 050	0.00 350	0.00 376	0.06 696	0.02 558	0.02 690
4	0.04 203	0.03 999	0.09 863	0.00 434	0.00 472	0.10 397	0.02 835	0.02 914
5	0.04 083	0.00 918	0.10 161	0.00 574	0.00 618	0.09 398	0.02 327	0.02 875
6	0.04 759	0.01 122	0.09 967	0.00 684	0.00 711	0.10 587	0.02 489	0.02 705
7	0.04 641	0.02 338	0.09 867	0.00 699	0.00 716	0.06 597	0.02 488	0.02 767
8	0.04 614	0.02 944	0.09 958	0.00 663	0.00 675	0.10 139	0.02 701	0.02 661
average variance	0.04 067	0.02 579	0.09 964	0.00 538	0.00 560	0.09 247	0.02 525	0.02 777

sample 3

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.01 890	0.00 782	0.09 960	0.00 457	0.00 457	0.09 096	0.02 238	0.02 298
2	0.03 161	0.01 126	0.09 890	0.00 444	0.00 469	0.09 216	0.02 155	0.02 257
3	0.03 456	0.01 670	0.09 970	0.00 371	0.00 393	0.06 756	0.02 192	0.02 188
4	0.02 881	0.04 595	0.09 994	0.00 364	0.00 375	0.10 183	0.01 971	0.02 447
5	0.02 578	0.04 229	0.10 113	0.00 459	0.00 486	0.09 399	0.01 966	0.02 471
6	0.01 848	0.02 154	0.09 939	0.00 625	0.00 666	0.09 882	0.02 118	0.02 421
7	0.01 390	0.01 551	0.09 930	0.00 765	0.00 814	0.06 501	0.02 284	0.02 451
8	0.01 869	0.01 569	0.10 023	0.00 620	0.00 642	0.10 081	0.02 304	0.02 440
average variance	0.02 384	0.02 210	0.09 977	0.00 513	0.00 538	0.08 889	0.02 153	0.02 371

sample 4

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.04 030	0.00 659	0.10 033	0.00 510	0.00 517	0.09 151	0.02 258	0.02 385
2	0.03 359	0.00 712	0.09 962	0.00 389	0.00 397	0.10 213	0.02 447	0.02 372
3	0.03 152	0.00 965	0.09 933	0.00 314	0.00 321	0.06 484	0.02 595	0.02 577
4	0.03 979	0.00 897	0.10 059	0.00 335	0.00 354	0.10 087	0.02 176	0.02 299
5	0.03 640	0.01 071	0.09 990	0.00 461	0.00 484	0.09 740	0.02 495	0.02 472
6	0.05 305	0.00 720	0.09 996	0.00 621	0.00 629	0.09 922	0.02 403	0.02 466
7	0.03 583	0.01 122	0.09 919	0.00 757	0.00 754	0.06 900	0.02 448	0.02 457
8	0.03 400	0.00 712	0.09 860	0.00 696	0.00 696	0.09 239	0.02 434	0.02 450
average variance	0.03 806	0.00 857	0.09 969	0.00 510	0.00 519	0.08 967	0.02 407	0.02 435

sample 5

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 1676 bins	histogr. 2500 b.
1	0.02 476	0.00 707	0.10 027	0.00 670	0.00 707	0.08 638	0.02 329	0.02 506
2	0.02 692	0.00 533	0.10 006	0.00 478	0.00 517	0.10 127	0.01 901	0.02 352
3	0.02 104	0.01 127	0.09 986	0.00 356	0.00 387	0.05 927	0.02 066	0.02 653
4	0.01 925	0.01 857	0.09 834	0.00 398	0.00 409	0.09 973	0.02 179	0.02 798
5	0.03 475	0.02 004	0.09 970	0.00 518	0.00 523	0.08 977	0.01 985	0.02 677
6	0.04 983	0.01 818	0.10 026	0.00 631	0.00 633	0.09 089	0.02 183	0.02 540
7	0.04 296	0.01 125	0.09 842	0.00 636	0.00 649	0.06 335	0.02 045	0.02 396
8	0.03 528	0.01 323	0.09 987	0.00 677	0.00 709	0.08 462	0.01 920	0.02 855
average variance	0.03 185	0.01 312	0.09 960	0.00 545	0.00 567	0.08 441	0.02 076	0.02 597

sample 6

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.04 448	0.01 164	0.10 036	0.00 540	0.00 561	0.09 140	0.02 393	0.02 408
2	0.02 413	0.00 551	0.10 011	0.00 368	0.00 379	0.09 607	0.02 184	0.02 666
3	0.01 155	0.00 456	0.09 826	0.00 312	0.00 326	0.05 841	0.02 195	0.02 467
4	0.01 394	0.00 863	0.09 936	0.00 358	0.00 383	0.09 512	0.02 279	0.02 696
5	0.02 030	0.01 480	0.10 131	0.00 490	0.00 525	0.08 280	0.02 385	0.02 437
6	0.01 793	0.01 513	0.09 937	0.00 587	0.00 610	0.08 598	0.02 142	0.02 722
7	0.02 219	0.01 087	0.10 019	0.00 585	0.00 588	0.06 151	0.02 373	0.02 482
8	0.03 486	0.00 626	0.10 043	0.00 614	0.00 628	0.09 454	0.02 340	0.02 539
average variance	0.02 367	0.00 967	0.09 992	0.00 482	0.00 500	0.08 323	0.02 287	0.02 552

average variance over all samples

	0.03 056	0.01 4891	0.09 9761	0.00 5281	0.00 551	0.08 8261	0.02 290	0.02 500

Table **3.3** (b)
Variance of the estimation error for the **100** vectors case

sample 1

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = **0.01**	histogr. **100** bins	histogr. **1676** bins	histogr. 2500 b.
1	0.01 409	0.00 680	0.00 833	0.00 571	0.09 902	0.12 664	0.14 001	0.17 184
2	0.01 410	0.01 143	0.00 760	0.00 470	0.10 006	0.12 362	0.13 009	0.16 506
3	0.02 185	0.01 629	0.00 717	0.00 483	0.09 995	0.13 683	0.13 000	0.17 956
4	0.02 981	0.04 165	0.00 797	0.00 562	0.10 031	0.13 395	0.13 152	0.18 431
5	0.04 196	0.03 134	0.01 071	0.00 762	0.09 891	0.11 903	0.12 822	0.17 329
6	0.04 576	0.01 600	0.01 293	0.00 933	0.10 030	0.09 644	0.13 548	0.16 719
7	0.02 601	0.01 336	0.01 064	0.00 791	0.10 087	0.13 581	0.13 687	0.17 163
8	0.02 135	0.01 454	0.00 832	0.00 635	0.10 001	0.13 420	0.13 130	0.16 404
average variance	0.02 687	0.01 893	0.00 921	0.00 651	0.09 993	0.12 581	0.13 294	0.17 211

sample 2

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = **0.01**	histogr. 100 bins	histogr. **1676** bins	histogr. 2500 b.
1	0.02 239	0.01 174	0.00 831	0.00 637	0.10 057	0.17 143	0.15 361	0.18 200
2	0.02 175	0.00 815	0.00 901	0.00 558	0.09 792	0.16 286	0.15 453	0.18 303
3	0.02 246	0.01 208	0.00 659	0.00 504	0.10 050	0.17 232	0.18 856	0.19 317
4	0.02 911	0.01 086	0.00 593	0.00 455	0.09 864	0.15 826	0.15 807	0.18 565
5	0.03 552	0.01 102	0.00 722	0.00 589	0.10 161	0.15 783	0.15 095	0.17 747
6	0.01 765	0.00 723	0.00 861	0.00 744	0.09 967	0.15 778	0.14 957	0.18 141
7	0.01 808	0.01 244	0.00 871	0.00 703	0.09 867	0.16 891	0.14 554	0.17 385
8	0.02 367	0.00 894	0.00 737	0.00 681	0.09 958	0.16 918	0.14 805	0.16 767
average variance	0.02 383	0.01 031	0.00 772	0.00 609	0.09 964	0.16 482	0.15 611	0.18 053

sample 3

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.03 193	0.00 721	0.00 873	0.00 628	0.09 960	0.18 158	0.14 395	0.18 064
2	0.02 758	0.00 619	0.00 643	0.00 444	0.09 890	0.19 857	0.14 758	0.17 541
3	0.02 014	0.01 408	0.00 408	0.00 347	0.09 970	0.17 290	0.14 373	0.19 004
4	0.02 409	0.02 057	0.00 543	0.00 426	0.09 993	0.18 975	0.14 347	0.16 866
5	0.03 378	0.02 566	0.00 918	0.00 696	0.10 113	0.18 973	0.14 429	0.17 534
6	0.03 975	0.02 147	0.01 209	0.00 942	0.09 939	0.18 085	0.14 423	0.16 767
7	0.05 221	0.00 939	0.01 130	0.00 972	0.09 930	0.17 625	0.13 744	0.17 527
8	0.05055	0.01 272	0.00 944	0.00 838	0.10 022	0.18 083	0.14 372	0.18 517
average variance	0.03 500	0.01 466	0.00 833	0.00 662	0.09 977	0.18 381	0.14 355	0.17 728

sample 4

nodes	binary output	Hadam. output	Parzen h = 1	Parzen h = h_{opt}	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
	0.05 230	0.01 324	0.00 661	0.00 576	0.10 033	0.13 903	0.11 250	0.17 665
	0.04 652	0.00 596	0.00 594	0.00 458	0.09 962	0.15 259	0.11 688	0.17 153
	0.06 227	0.00 443	0.00 637	0.00 439	0.09 934	0.16 130	0.07 897	0.17 750
	0.05 195	0.01 188	0.00 794	0.00 607	0.10 059	0.13 744	0.11 359	0.18 910
	0.05 348	0.02 367	0.00 856	0.00 694	0.09 990	0.15 015	0.11 205	0.18 240
6	0.02 965	0.01 609	0.00 970	0.00 764	0.09 996	0.14 981	0.11 256	0.18 086
7	0.02 359	0.01 082	0.01 072	0.00 893	0.09 919	0.15 142	0.07 323	0.17 790
8	0.03 516	0.00 594	0.00 868	0.00 772	0.09 860	0.14 355	0.11 146	0.17 714
average variance	0.04 437	0.01 150	0.00 807	0.00 650	0.09 969	0.14 816	0.10 391	0.17 914

sample 5

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.03 947	0.02 549	0.00 792	0.00 608	0.10 027	0.16 696	0.14 757	0.18 579
2	0.04 480	0.00 938	0.00 625	0.00 530	0.10 006	0.14 853	0.14 521	0.19 456
3	0.02 825	0.00 441	0.00 509	0.00 385	0.09 986	0.16 927	0.10 524	0.17 835
4	0.04 111	0.00 737	0.00 631	0.00 432	0.09 834	0.16 131	0.14 246	0.18 622
5	0.04 272	0.01 150	0.00 577	0.00 450	0.09 970	0.16 986	0.14 017	0.17 235
6	0.03 249	0.01 011	0.00 629	0.00 518	0.10 026	0.18 137	0.15 189	0.18 258
7	0.03 583	0.02 941	0.00 938	0.00 702	0.09 842	0.16 776	0.17 314	0.17 846
8	0.03 280	0.03 209	0.00 923	0.00 663	0.09 987	0.15 797	0.15 385	0.17 804
average variance	0.03 718	0.01 622	0.00 703	0.00 536	0.09 960	0.16 538	0.14 494	0.18 204

sample 6

nodes	binary output	Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01	histogr. 100 bins	histogr. 676 bins	histogr. 2500 b.
1	0.03 573	0.03 596	0.00 720	0.00 564	0.10 035	0.15 035	0.11 781	0.16 300
2	0.03 724	0.02 963	0.00 989	0.00 660	0.10 012	0.15 513	0.11 730	0.17 515
3	0.02 865	0.01 486	0.00 788	0.00 496	0.09 826	0.16 082	0.12 285	0.18 877
4	0.03 856	0.01 092	0.00 797	0.00 604	0.09 936	0.16 647	0.11 687	0.17 726
5	0.03 261	0.01 118	0.01 064	0.00 872	0.10 131	0.15 756	0.11 263	0.16 445
6	0.02 625	0.00 614	0.01 008	0.00 859	0.09 937	0.13 815	0.11 979	0.17 046
7	0.02 349	0.02 024	0.00 989	0.00 849	0.10 019	0.15 962	0.08 143	0.17 021
8	0.03 346	0.02 733	0.00 773	0.00 656	0.10 043	0.13 285	0.11 743	0.15 907
average variance	0.03 200	0.01 9531	0.00 891	0.00 6951	0.09 9921	0.15 262	0.11 3261	0.17 105

average variance over all samples

	0.03321	0.01519	0.00821	0.00634	0.09976	0.15677	0.13245	0.17702

Table 3.4
Mean of the estimation error for the 300 vector case

sample 1

nodes	15 hidden neurons binary output	15 hidden neurons Hadam. output	25 hidden neurons binary output	25 hidden neurons Hadam. output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01
1	-0.00 284	0.00 099	-0.00 621	0.00 116	-0.00 049	0.00 316	-0.00 289
2	-0.00 365	-0.00 208	-0.00 745	-0.00 282	-0.00 184	-0.01 260	-0.00 285
3	0.00 506	0.00 376	0.00 816	0.00 776	0.00 206	0.01 329	0.00 441
4	-0.00 070	-0.00 007	0.00 293	-0.00 303	-0.00 066	0.01 271	0.00 114
5	-0.00 285	-0.00 419	-0.00 453	-0.00 374	0.00 029	-0.01 809	-0.00 491
6	0.00 014	0.00 495	-0.00 555	0.00 316	-0.00 186	0.01 748	0.00 423
7	0.00 402	-0.00 261	0.00 709	-0.00 085	0.00 088	-0.01 213	-0.00 165
8	0.00 012	-0.00 073	0.00 439	-0.00 204	0.00 161	-0.00 016	0.00 251

sample 2

nodes	binary output	Hadam. output	binary output	Hadam. output
1	-0.00 286	-0.00 489	0.00 194	-0.00 531
2	0.00 457	0.00 018	-0.00 088	0.00 366
3	0.00 084	0.00 067	-0.00 332	-0.00 118
4	-0.00 444	-0.00 105	0.00 167	-0.00 053
5	-0.00 080	-0.00 001	0.00 352	0.00 245
6	0.00 342	0.00 017	-0.00 186	-0.00 570
7	0.00 301	0.00 185	-0.00 124	0.00 164
8	-0.00 284	0.00 378	0.00 116	0.00 474

average absolute errors over all samples:

3.17E-06	1.17E-05	3.71E-05	3.50E-05	3.12E-11	4.57E-04	-1.25E-06

Table 3.5

Variance of the estimation error for 300 training vectors,
3-dimensional input data

sample 1 15 hidden neurons 25 hidden neurons

nodes	binary output	Hadamard output	binary output	Hadamard output	Parzen h = 1	Parzen $h = h_{opt}$	Parzen h = 0.01
1	0.05 674	0.04 847	0.04 293	0.01 686	0.09 970	0.02 300	0.00 690
2	0.04 888	0.03 621	0.05 509	0.01 314	0.09 900	0.02 540	0.00 667
3	0.05 565	0.01 400	0.05 024	0.03 049	0.10 200	0.02 700	0.00 691
4	0.05 078	0.01 067	0.05 716	0.03 477	0.09 980	0.02 300	0.00 677
5	0.04 823	0.02 088	0.05 483	0.03 870	0.10 000	0.02 230	0.00 821
6	0.05 296	0.02 071	0.04 651	0.03 774	0.09 840	0.02 690	0.00 905
7	0.04 599	0.02 110	0.04 830	0.01 396	0.09 950	0.02 480	0.00 911
8	0.04 624	0.02 850	0.04 405	0.01 599	0.10 000	0.02 450	0.00 867
average variance	0.05 068	0.02 507	0.04 989	0.02 521	0.09 980	0.02 461	0.00 779

sam le 2

nodes	output	output	output	output
	0.04 847	0.02 089	0.05 528	0.01 890
2	0.05 045	0.02 932	0.04 733	0.01 708
3	0.05 091	0.02 219	0.05 022	0.01 604
4	0.04 889	0.04 147	0.05 311	0.01 832
5	0.05 105	0.05 026	0.04 870	0.01 915
6	0.04 261	0.03 004	0.04 770	0.03 923
7	0.04 758	0.01 047	0.05 589	0.04 559
8	0.04 942	0.01 559	0.05 225	0.03 113
average variance	0.04 867	0.02 753	0.05 131	0.02 568

average variance over all samples

	0.04 9681	0.02 630	0.05 060	0.02 5451	0.09 980	0.02 461	0.00 7791

In the Parzen case, store the distribution in a 50 by 50 by 50 lockup table took a lot of memory, about 7.75 MB for each class. In contrast, the weights and biases of the neural network can be stored in 150 kb. The training time for the Parzen density estimation for the 3-D case was 32 hours. Hence we restricted it to one sample. The training times for the neural network were smaller as well, and the time needed to compute the probability of an input vector during testing was about 2 times smaller than that of the Parzen density estimation. These experimental results are shown in Tables 3.6-3.8.

Table 3.6
Memory needed to store the weights

	2-D case	3-D case 15 hidden neurons	3-D case 24 hidden neurons
neural network	8.44 kb	11.7 kb	19.7 kb
parzen density	939 kb	45 700 kb	

Table 3.7 (a)
Training times, using 100 vectors for training

sample no.	1	2	3	4	5	6	average
binary repr.	189	194	189	188	191	190	190
Hadamard repr.	192	190	190	188	190	193	190
Parzen density	36	36	36	36	36	36	36

Table 3.7 (b)
Training times, using 1000 vectors for training
2-dimensional input data

sample no,	1	2	3	4	5	6	average
binary repr.	3733	3 914	3 766	3 556	3 890	3 750	3 768
Hadamard repr.	3 734	3 743	3 618	3636	3917	3740	3731
Parzen density	221	226	222	232	217	225	224

Table 3.7 (c)
Training time for the classification 300 vectors for training
3-dimensional input data

sample no,	1	2	3	4	5	6	average
binary repr. 15 hidden nodes	2 200	2 240	2 190	2200	2 180	2 160	2 195
Hadamard repr. 15 hid. nod.	2 190	2 250	2 190	2210	2 190	2210	2 207
binary repr. 25 hidden nodes	3414	3 419	3 431	3 397	3419	3416	3416
Hadamard repr. 25 hid. nod.	3 422	3 427	3 480	3 378	3 439	3 382	3 421
Parzen density	3 634	---	---	---	---	---	3 634

Apparently, CPU time is not equivalent to real time. The time needed to compute the Parzen density estimate was 32 hours, compared to 5 hours for the slowest neural network.

Table 3.8 (a)
Testing time using 1000 vectors
2-D

sample no.	1	2	3	4	5	6	average
binary repr.	99	96	109	94	95	93	98
Hadamard repr.	94	94	93	94	94	93	93
Parzen density	228	228	226	225	226	228	227

Table 3.8 (c)
Testing time using 1000 vectors
3-D

sample no,	1	2	3	4	5	6	average
binary repr. 15 hidden no.	105	104	106	107	103	104	105
Hadamard repr. 15 hidd. no.	103	101	104	105	103	104	103
binary repr. 25 hidden no.	109	117	137	161	156	153	139
Hadamard repr. 25 hidd. no.	102	105	103	108	109	107	106
Parzen density	670	---	---	---	---	---	670

3.5 Conclusions

We have introduced a new output representation, which reduces the variance of the estimation error by up to 50 % for an 8 class problem. We have then compared the performance of the neural networks to the performance of mathematical classifiers. Our research shows that already for the 3 dimensional case the neural networks become much faster, while yielding estimation results similar to those of the Parzen density estimation. The neural networks need less memory, and, for the 3 dimensional case, perform faster in both training and testing. The biggest advantage of the neural network is the complete absence of assumptions of the underlying data. For the Parzen density estimation, one has to adjust the smoothing parameter correctly. The existing formula for the estimation of an optimal smoothing parameter assumes an underlying distribution and depends on the estimated variance of the training set.

4. ADDITION OF REDUNDANT HIDDEN NODES

4.1 Introduction

The model we set up in Chapter 2 shows a decrease of the error variance by $1/N$ for the Hadamard-transformed output representation. N is the number of output nodes, which was equal to the number of classes in Chapter 2. This suggests that, by increasing the number of output nodes, and consequently, the size of the Hadamard matrix, we can decrease the variance further.

In order to increase the number of output nodes, we added components with zero value for all input classes to the desired outputs. We will refer to this new binary output vectors as zero padded vectors. The output nodes which are trained against the components of the desired vectors equaling zero for all input classes will be referred to as redundant nodes. They are referred to by redundant nodes, since their values have no direct effect on classification. They values will be discarded during testing.

When taking the Hadamard-transform of these vectors, we obtain new desired output vectors which equal the first rows of the Hadamard matrix. These new Hadamard-transformed outputs will have no constant component except the first one. Hence the learning with these vectors will be simply completed like learning a Hadamard-transformed representation of a problem with a number of different classes.

The effect of increasing the size of the Hadamard matrix might be opposed by an increase of the error at the output nodes, before taking the inverse Hadamard-transform. We show in Section 4.2 that the model of Chapter 2 can be used for neural networks with redundant output nodes as well. Section 4.3 contains the experiments with redundant nodes. We show that for small and medium amounts of redundant nodes we do not encounter significant increases in the error at the output. Hence, the performance improves drastically. As in Chapter 3, we compare the results to those of statistical density estimators. Section 4.4 contains the discussion. In Section 4.5 we apply these techniques to the 3-dimensional case. Section 4.6 is conclusions.

4.2 Theoretical Treatment of Redundant Nodes

In Chapter 2, we have shown that by using the inverse Hadamard-transform, we can decrease the output error by 1/N. We assumed the number of classes to be equal to the matrix size N of the Hadamard matrix.

Introducing redundant nodes can be seen as zero padding of the desired output vectors. For example, desired output vectors for a two class problem, which is zero padded to size 4, are given by

$$\mathbf{D} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.1}$$

Using equation (2.1), we get:

$$\mathbf{D_h} = \mathbf{H}\,\mathbf{D} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.2}$$

Thus, $\mathbf{D_h}$ simply becomes the first two columns of the Hadamard matrix of size 4. We now train the neural network with the first column as desired output for class 1 and the second column as desired output for class 2. In Chapter 2, we have explained that the output of the network then learns the probability of the desired output component j of the desired output $\mathbf{D_h}$ equaling 1. Of course, to obtain that actual probability, we would have to shift and scale the actual output. This property of the neural network is not changed by using only the first 2 columns as desired output vectors.

As in Chapter 2 the actual output is given by

$$\mathbf{O_i = H\ \hat{P}(C|X_i)} \tag{4.3}$$

Hence we can solve equation (4.3) to obtain the probability estimates. Using $\mathbf{H^{-1} = H/N}$, we obtain:

$$\mathbf{\hat{P}(C|X_i) = H\ O_i\ /N} \tag{4.4}$$

Since **H** is a square matrix of size N, and $\mathbf{O_i}$ is a vector of length N, $\hat{P}(C|Xi)$ is a vector of length N. We can truncate all output values with j>(N-M). The first N-M components contain the a posteriori probabilities of the (N-M) classes we have.

Now we will investigate if using only the first N-M columns of the Hadamard matrix for training affects the model variance set up in Chapter 2. We have shown, that for the average variance reduction, we can use our simple model set up in Section 2.2. Using the assumptions (2.4)-(2.7), Equation (2.12) is still given by

$$\mathbf{S = E\{(H^{-1}\ e^{o}_{i})^2\}} \tag{4.5}$$

Using $\mathbf{H^{-1} = H^t/N}$ and the independence of the components of $\mathbf{e^{o}_{i}}$ we obtain the average variance to be

$$\sigma^2 = \frac{1}{N^2}\sum_{i=1}^{n} E\{e_i^{o2}\} = \frac{1}{N^2}\sum_{i=1}^{n}\sigma_{oi}^2 \tag{4.6}$$

Like we have shown at the beginning of this section, $\mathbf{H^{-1}}$ is not affected by using only the first N-M columns for training. Also, the error at the output nodes before the inverse

Hadamard-transform still minimizes the error of that node equaling 1. Hence the gain in variance is not affected by the number of different output patterns used for training.

This result enables us to add redundant nodes and expect a reduction of the variance by the size N of the zero padded desired outputs. Clearly N still has to be 2^k since Hadamard matrices only exist for 2^k, where k is an integer. However, since we no longer use all columns of H, we can now use any number L<N of the columns of H as desired output. This means, we are no longer restricted to problems with 2^k classes.

By adding redundant nodes, we increase the network complexity. This might lead to an increase of the output error before the inverse Hadamard-transform. We now experimentally test if the output error increases, and if the addition of redundant notes yields the theoretical benefits. The results are discussed in the next section.

4.3 Experiments with the Redundant Nodes and Comparison with the Statistical Methods

We used the same artificially generated 2 dimensional 100 vector training set used in Chapter 2. The same 1000 vector testing set like in Chapter 2 was used as well. Only the testing results are shown, since there is limited interest in training values. We restricted to the use of the. For comparison, we always trained a binary output and a Hadamard-transformed output network. We show the results for 0, 8, 24 and 56 redundant nodes, giving a total of 8, 16, 32 and 64 output nodes, respectively. For the 24 redundant node case we, trained networks with 25 and 45 hidden nodes. For the 56 redundant node case, we trained one network with 45 hidden nodes and one with 75 hidden nodes. We increased the number of hidden neurons, since the redundant nodes increase the complexity of the output. Like before, we computed the variance, mean square and the mean of the estimation error. In this section, we will only show the average testing results for each sample, and the average over all samples. For comparison, we included the results of the best statistical method, namely the Parzen density estimation with estimated optimal smoothing parameter h_{opt}. In Table 1 we also included the optimal

Bayesian classifier, giving the maximal possible performance. Tables **2** through **4** will show the variance of the estimation error, the mean square error and the mean error, respectively. Figures **4.1** and **4.2** display how the classification accuracy increases the mean square error decreases, as the number of hidden and output nodes increases.

Table 4.1
Classification percentage 100 vectors as training set
2 - dimensional input data, 1000 vectors for testing

100 vectors

sample #	1	2	3	4	5	6	average
no redundant nodes, 15 hidden nodes							
binary output	91.20%	93.50%	90.40%	87.40%	90.60%	91.40%	90.75%
Hadamard output	92.30%	94.10%	92.30%	91.10%	92.80%	90.10%	92.12%
8 redundant nodes, 25 hidden nodes							
binary output	91.70%	93.00%	92.90%	93.20%	94.00%	91.80%	92.77%
Hadamard output	94.14%	94.03%	93.85%	94.25%	94.74%	94.21%	94.20%
24 redundant nodes, 25 hidden nodes							
binary output	89.50%	93.20%	93.60%	93.70%	93.60%	92.70%	92.72%
Hadamard output	93.90%	94.50%	94.10%	94.90%	94.50%	94.70%	94.43%
24 redundant nodes, 45 hidden nodes							
binary output	92.68%	93.71%	92.64%	91.78%	92.13%	93.24%	92.69%
Hadamard output	94.25%	94.64%	93.99%	94.59%	94.59%	94.71%	94.46%
58 redundant nodes, 45 hidden nodes							
binary output	91.90%	91.50%	91.50%	91.60%	91.20%	89.00%	91.12%
Hadamard output	94.24%	94.81%	94.44%	94.68%	95.04%	94.34%	94.59%
58 redundant nodes, 75 hidden nodes							
binary output	93.00%	93.00%	92.80%	92.80%	92.30%	91.70%	92.60%
Hadamard output	94.10%	94.50%	94.40%	94.60%	94.90%	94.60%	94.52%
Parzen density h_{opt}	91.50%	91.80%	91.00%	91.70%	92.10%	90.90%	91.50%
maximal possible classification	94.63%	94.99%	94.86%	94.84%	95.10%	95.11%	94.92%

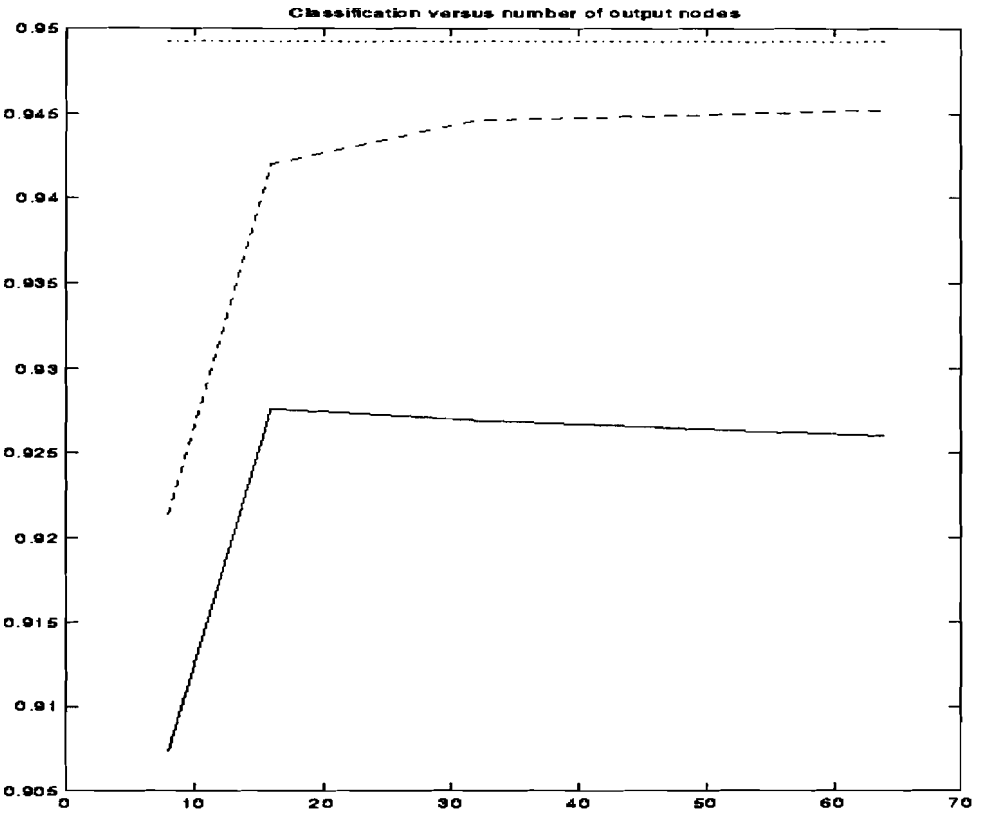

Fig. 4.1 Classification vs. number of output/hidden nodes
dotted: maximum possible classification
dashed: Hadamard-transformed network
solid: binary network

Table 4.2
Average variance of estimation error, 100 training vectors
2 - dimensional input data, 1000 vectors for testing

100 vectors

sample #	1	2	3	4	5	6	average
no redundant nodes, 15 hidden nodes							
binary	0.02 690	0.02 380	0.03 500	0.04 440	0.03 720	0.03 200	0.03 322
Hadamard	0.02 040	0.01 490	0.01 970	0.02 750	0.02 740	0.02 680	0.02 2781
8 redundant nodes, 25 hidden nodes							
binary	0.02 480	0.01 970	0.02 150	0.02 250	0.01 660	0.02 670	0.02 197
Hadamard	0.00 747	0.00 947	0.00 892	0.00 897	0.00 711	0.00 771	0.00 828
24 redundant nodes, 25 hidden nodes							
binary	0.03 540	0.01 960	0.02 170	0.02 500	0.02 530	0.02 440	0.02 523
Hadamard	0.00 755	0.00 726	0.00 808	0.00 592	0.00 7341	0.00 588	0.00 701
24 redundant nodes, 45 hidden nodes							
binary	0.01 800	0.02 200	0.01 740	0.02 690	0.02 120	0.02 080	0.02 105
Hadamard	0.00 5591	0.00 5261	0.00 5621	0.00 547	0.00 6811	0.00 5151	0.00 5651
58 redundant nodes, 45 hidden nodes							
binary	0.03 510	0.03 000	0.03 080	0.03 890	0.02 580	0.04 040	0.03 350
Hadamard	0.00 5531	0.00 4941	0.00 5661	0.00 529	0.00 5321	0.00 7451	0.00 570
58 redundant nodes, 75 hidden nodes							
binary	0.01 240	0.01 630	0.01 640	0.01 860	0.01 570	0.01 740	0.01 613
Hadamard	0.00 412	0.00 412	0.00 407	0.00 451	0.00 395	0.00 513	0.00 432
Parzen h_{opt}	0.00 625	0.00 556	0.00 672	0.00 629	0.00 558	0.00 707	0.00 625

Table 4.3
Average mean square of estimation error, 100 training vectors
2 - dimensional input data, 1000 vectors for testing

100 vectors

sample #	1	2	3	4	5	6	average
no redundant nodes, 25 hidden nodes							
binary	0.02 690	0.02 380	0.03 530	0.04 440	0.03 760	0.03 200	0.03 333
Hadamard	0.02 040	0.01 490	0.01 970	0.02 870	0.02 740	0.02 680	0.02 298
8 redundant nodes, 25 hidden nodes							
binary	0.02 490	0.01 980	0.02 170	0.02 250	0.01 670	0.02 670	0.02 205
Hadamard	0.00 748	0.00 952	0.00 8951	0.00 9191	0.00 713	0.00 8241	0.00 8421
24 redundant nodes, 25 hidden nodes							
binary	0.03 540	0.01 970	0.02 170	0.02 500	0.02 530	0.02 440	0.02 525
Hadamard	0.00 769	0.00 7281	0.00 811	0.00 5941	0.00 757	0.00 5881	0.00 708
24 redundant nodes, 45 hidden nodes							
binary	0.01 800	0.02 200	0.01 740	0.02 690	0.02 120	0.02 080	0.02 105
Hadamard	0.00 590	0.00 529	0.00 564	0.00 548	0.00 682	0.00 515	0.00 571
58 redundant nodes, 45 hidden nodes							
binary	0.03 510	0.03 000	0.03 080	0.03 900	0.02 580	0.04 040	0.03 352
Hadamard	0.00 554	0.00 496	0.00 584	0.00 531	0.00 534	0.00 753	0.00 575
58 redundant nodes, 75 hidden nodes							
binary	0.01 240	0.01 650	0.01 650	0.01 860	0.01 570	0.01 740	0.01 618
Hadamard	0.00 413	0.00 414	0.00 410	0.00 453	0.00 397	0.00 517	0.00 434
Parzen h_{opt}	0.00 627	0.00 5581	0.00 6731	0.00 631	0.00 560	0.00 707	0.00 626

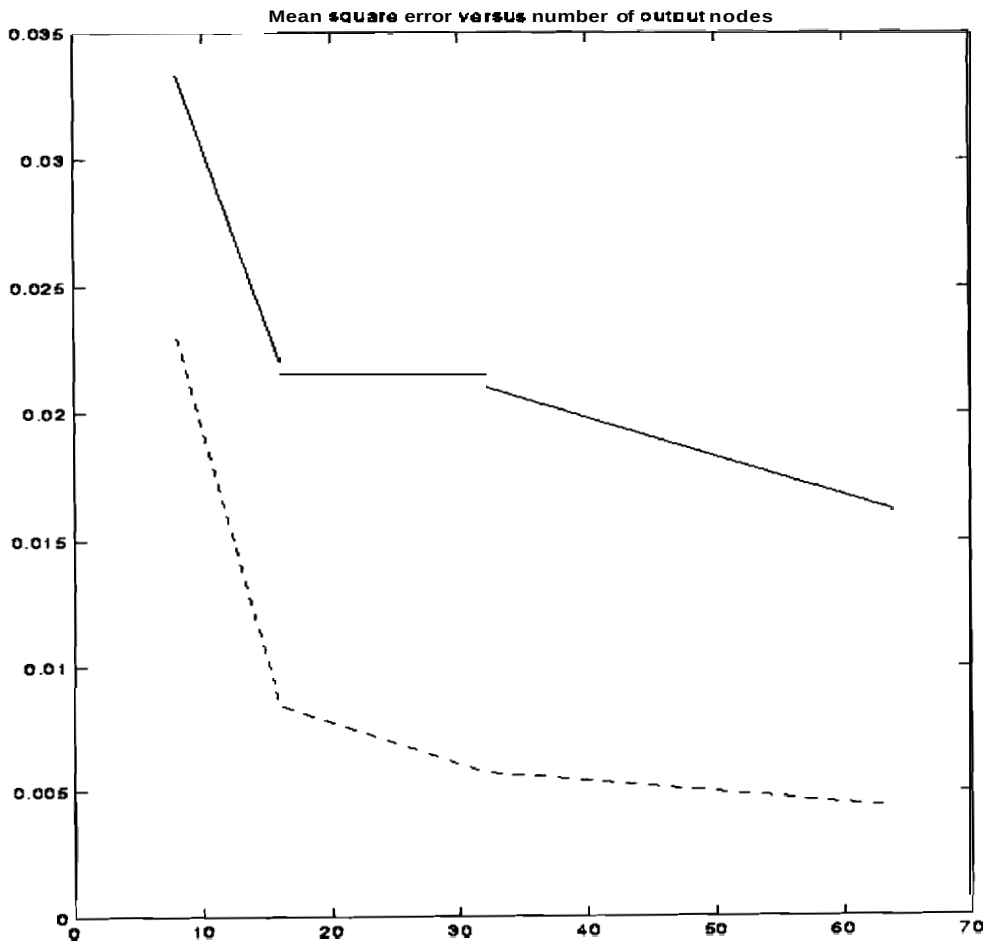

Fig. 4.2 Mean square vs. number of output/hidden neurons
solid : binary network
dashed : Hadarnard-transformed network

Table 4.4
Average mean of the estimation error, 100 training vectors
2 - dimensional input data, 1000 vectors for testing

sample #	1	2	3	4	5	6	average
no redundant nodes, 15 hidden nodes							
binary	-0.000 1651	0.000 008	0.000 089	-0.000 231	0.002 100	-0.000 121	0.00 028
Hadam.	0.000068	0.000 304	-0.000 064	-0.000 776	0.000 099	-0.000 081	-0.00 007
8 redundant nodes, 25 hidden nodes							
binary	-0.004 130	0.000 375	0.000 184	0.000 016	0.000 384	0.000 100	-0.00 051
Hadam.	0.000 120	0.000 066	0.000 086	-0.000 065	0.000 2431	-0.000 291	0.00 003
24 redundant nodes, 25 hidden nodes							
binary	-0.001 050	0.000 294	-0.000 043	-0.000 401	-0.000 037	-0.003 080	-0.000 720
Hadam.	0.000 005	0.000 086	-0.000 071	-0.000 043	-0.000 018	-0.000 258	-0.000 050
24 redundant nodes, 45 hidden nodes							
binary	0.000 783	0.000 106	0.001 590	0.000 160	-0.000 848	-0.000 517	0.000 212
Hadam.	-0.000 091	0.000 077	0.000 016	0.000 120	-0.000 114	0.000 097	0.000 018
58 redundant nodes, 45 hidden nodes							
binary	-0.000 054	0.000 044	-0.000 219	-0.000 167	0.000 062	-0.000 606	-0.000 157
Hadam.	-0.000 023	0.000 030	-0.000 026	0.000 006	0.000 005	-0.000 007	-0.000 002
58 redundant nodes, 75 hidden nodes							
binary	-0.0000371	0.0058201	0.0000571	0.000046	-0.0005701	-0.0015301	0.000631
Hadam.	0.000002	-0.000107	-0.000080	-0.000142	-0.000003	-0.000055	-0.000064
Parzen h_{opt}	6.25E-12	1.74E-12	-5.62E-12	1.05E-11	-1.92E-11	2.23E-11	2.66E-12

Interestingly, the average estimation error for the binary network decreases as well with the addition of 8 and 16 redundant notes. The experiments also indicated that the number of hidden neurons should always be a larger than the number of output nodes to achieve optimal results.

The classification increases and reaches almost the optimal Bayesian classification result. The classification accuracy achieved with the redundant neural network is significantly better than that achieved by Parzen density estimation.

The variance and the mean square error of the Hadarnard-transformed neural network

are also reduced by using redundant output nodes. In comparison to the non-redundant Hadamard-transformed neural network, the reduction in error variance for 8 redundant nodes is 64 %, more then the 50 % we expected. For 16 redundant nodes and 45 hidden neurons, we reduced it by about 75%, like expected. For the 56 vector case with 75 hidden neurons, the improvement is about 81 %, 6.5 % less than the expected 87.5 %. The maximal achieved reduction of network error in comparison to the binary network with no redundant notes is 87%. This means that we could reduce the variance of the estimation error to about 1/8 of its previous value. The lowest variance and mean square error achieved were about 31 % lower than that of the optimal Parzen density estimation method.

On the other hand, increasing the network complexity increases the time needed for training, and testing. Tables 4.5 and 4.6 show the respective CPU times in seconds. Table 4.5 shows the increase in memory needed to store the additional weights.

Table 4.5
Training time, using 100 vectors for training
2-dimensional input data

sample no.	1	2	3	4	5	6	average
no redundant nodes, 25 hidden nodes	99	96	109	94	95	93	98
8 redundant nodes, 25 hidden nodes	189	194	189	188	191	190	190
24 redundant nodes, 25 hidden nodes	716	721	734	727	731	737	728
24 redundant nodes, 45 hidden nodes	1 280	1 290	1 270	1 280	1 290	1 270	1 280
58 redundant nodes, 45 hidden nodes	1 580	1 440	1 390	1 320	1 320	1 410	1 410
58 redundant nodes, 75 hidden nodes	3 730	3 610	3 660	3 720	3 480	3 790	3 665
Parzen density	35	35	34	35	34	36	35

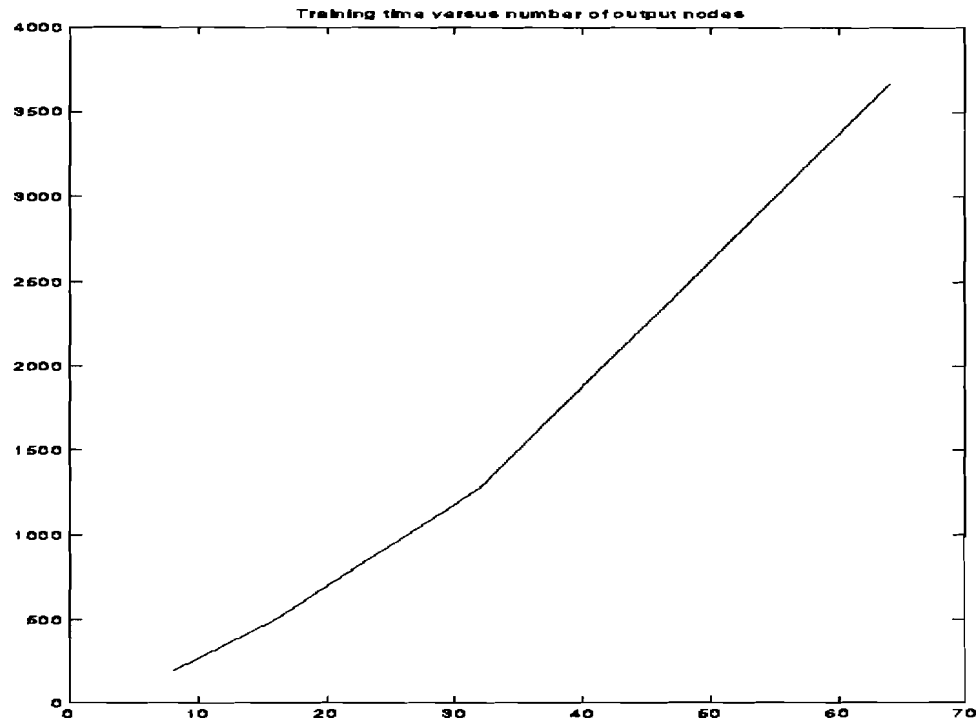

Fig. 4.3 Training time versus number of output/hidden nodes

Table 4.6
Testing time, using 1000 vectors for testing
2-dimensional input data

sample no.	1	2	3	4	5	6	average
no redundant nodes, 25 hidden nodes	106	113	100	112	105	103	107
8 redundant nodes, 25 hidden nodes	104	104	144	148	140	161	134
24 redundant nodes, 25 hidden nodes	209	177	185	198	211	234	202
24 redundant nodes, 45 hidden nodes	153	197	227	221	230	225	209
58 redundant nodes, 45 hidden nodes	190	463	493	540	439	424	425
58 redundant nodes, 75 hidden nodes	242	451	726	698	739	640	583
Parzen density	228	228	226	225	226	228	227

Table 4.6
Memory needed to store the mapping

	neural network						Parzen
no. redundant neurons	0	8	24		58		---
no. hidden neurons	25	25	25	45	45	75	---
memory / kb	2.72	7.66	14.13	25.13	54.33	77.09	939.00

4.4 Discussion of the Results

The most surprising result is the reduction of error variance and the mean square error of the binary network with redundant nodes. The most likely explanation is that with the addition of redundant nodes we increased the number of hidden nodes. When we left the number of hidden nodes constant, but increased the number of redundant output nodes, the output error increased slightly. This can expected, since we allow more total output noise, but do not use those additional output nodes.

To show this effect we set the number of hidden neurons constantly to 75 for the 3 dimensional case, discussed in Section 4.5.

In the previous Chapters, we had used 15 - 25 hidden neurons. This number was experimentally determined by [3]. However, they used one dimensional input data. We used 2 or 3 dimensional input data. One can think of each hidden neuron trying to learn small piece of the distribution function, similar to the function learning presented by [14]. Now, the accuracy of this mapping will depend not only on how accurate one neuron learns its piece of the distribution, but also on how many hidden neurons we use. This is a problem seemingly similar to over -and under smoothing in statistical estimators. In our case, the lowest mean square error was achieved with the highest number of hidden neurons. We can expect to reach some saturation point, after which the addition of more hidden neurons will increase the error. For the 2 - and 3 dimensional input data, we stopped before, due to the excessive times necessary. In the one dimensional case, [3] show that there is a certain number of hidden neurons, which gives

minimum output error. The increase of hidden neurons might be thought of as "curse of dimensionality", analog to the curse of dimensionality for statistical methods. [8]

The effects for the Hadamard-transform are quite different. As discussed in Chapter 2, the error at the output of the Hadamard-transform depends on the absolute error at the output of the network, and not on the relative error, the probability error. In Chapter 2, the output error of the Hadamard-transformed network was found to be approximately twice the size of the error of the binary network. Hence the error variance of the Hadamard-transformed network is reduced by 4/N instead 1/N. Still, our results do not show a reduction by 4/N. 4/N seems to be the upper bound of reduction, achieved in some cases, while others achieve lower reductions. This must be a consequence of an increase of the output error at the output nodes of the Hadamard-transformed network.

In Chapter to we found out that the dependency terms T, caused by taking the inverse Hadarnard-transform of not independent output values, summed up to 0 over all output values. However, when we use redundant nodes, we will only use the first 8 output nodes for the a posteriori probabilities. The dependency terms no longer have to cancel out, like they do in Section 2.5. In Section 4.5, the 3 - dimensional case, we will measure the average of the actual dependencies components T for the first 8 output nodes.

Another disadvantage is that training and testing times increase significantly, and so does the needed memory.

4.5 Further Experiments

We used a set of 100 training vectors. The structure of the set is similar to that of the 300 vector per class 3-dimensional set used in Chapter 3.

We again trained a binary and a Hadamard network. Each of them had 75 hidden neurons. Table 4.8 will show the testing classification, using 1000 testing vectors. We ran 4 samples, each of them with 0, 8, 24 and 56 redundant output nodes. the achieved classification, and the mean square error are shown in Table 4.8, 4.9 and 4.10. Figure 4.4 and 4.5 will show the classification and the mean square error versus the number of

output nodes, respectively.

Table 4.8
Testing classification with the 3 - D input data

sample #	1	2	3	4	average
8 output nodes					
binary	88.00%	87.20%	88.10%	85.90%	87.30%
Hadamard	191.20%	91.30%	91.00%	91.40%	91.23%
16 output nodes					
binary	86.60%	86.90%	87.20%	87.90%	87.15%
Hadamard	91.00%	91.90%	92.70%	92.10%	91.93%
32 output nodes					
binary	87.08%	85.73%	85.96%	84.19%	85.74%
Hadamard	93.80%	92.80%	92.10%	93.20%	92.98%
64 output nodes					
binary	83.80%	85.90%	85.00%	82.60%	84.33%
Hadamard	94.00%	93.80%	93.60%	93.70%	93.78%
max. possible classification	95.25%	94.88%	94.46%	95.03%	94.90%

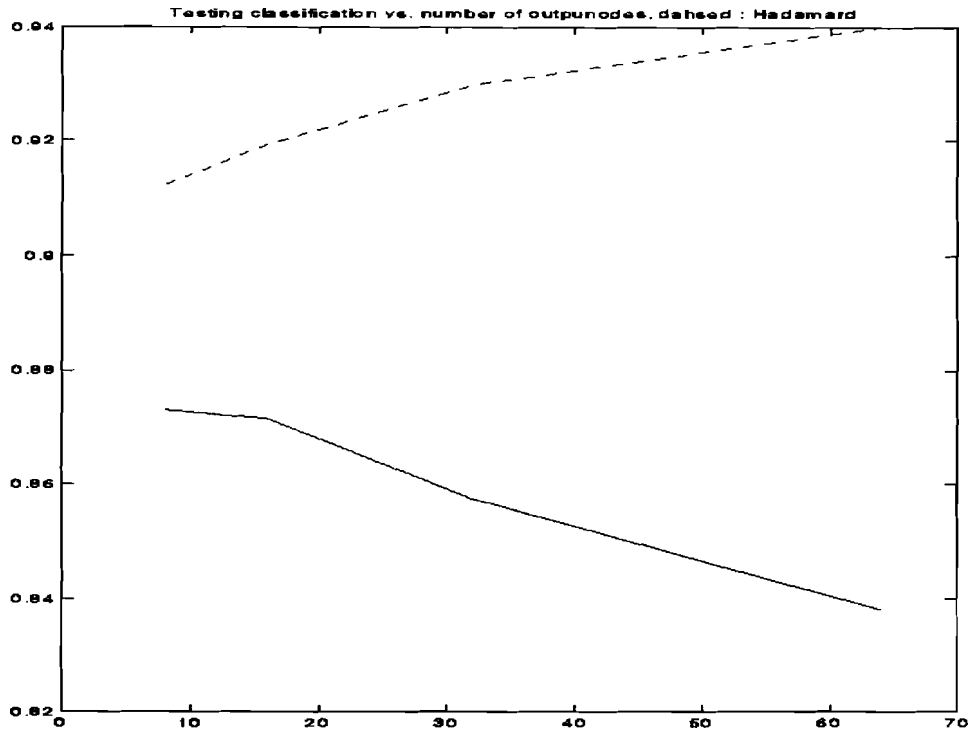

Fig. 4.4 Testing classification versus number of output neurons

Table 4.9
Mean square error in the 3 - D case, 75 hidden neurons

sample #	1	2	3	4	average
8 output nodes					
binary	0.03 904	0.03 956	0.03 453	0.04 057	0.03 842
Hadamard	0.02 680	0.02 570	0.02 250	0.02 520	0.02 505
16 output nodes					
binary	0.04 590	0.04 245	0.03 591	0.03 806	0.04 058
Hadarnard	0.02 100	0.02 200	0.01 590	0.02 590	0.02 120
32 output nodes					
binary	0.03 941	0.04 780	0.04 643	0.04 806	0.04 542
Hadamard	0.01 123	0.01 486	0.01 617	0.01 377	0.01 401
64 output nodes					
binary	0.05 059	0.04 540	0.04 642	0.05 023	0.04 816
Hadamard	0.01 084	0.01 110	0.00 807	0.01 040	0.01 010

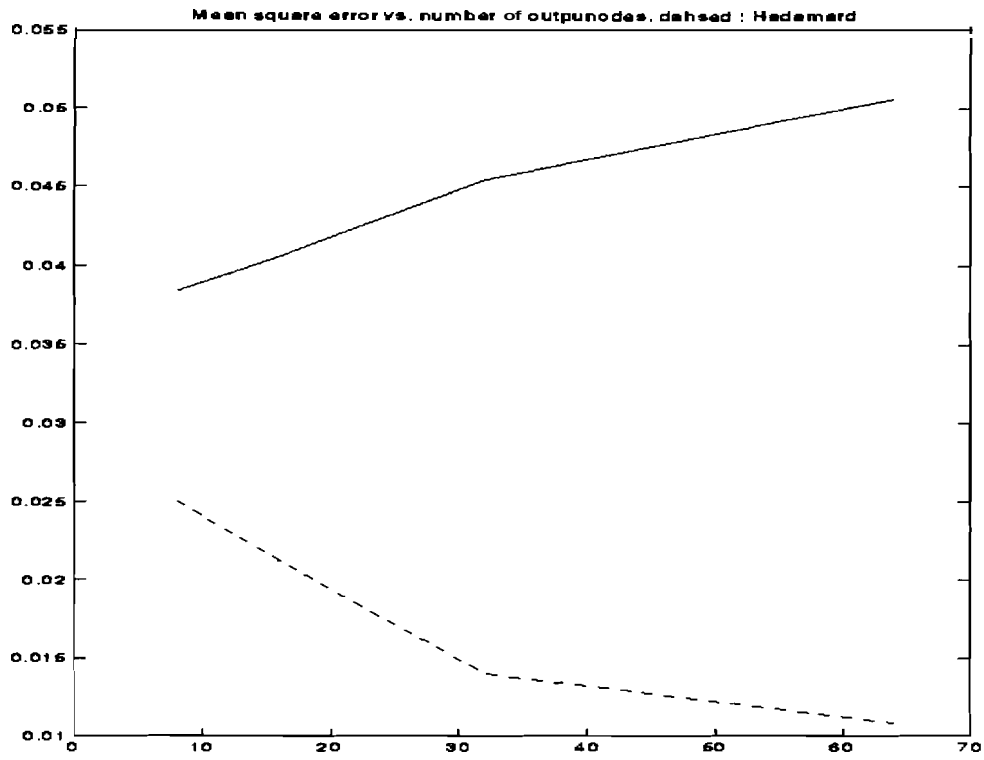

Fig. 4.5 Mean square error versus number of output neurons

With a constant number of hidden neurons, the binary classification decreases and the mean square error increases with the number of output neurons, while the classification of the Hadamard-transformed network increases. The mean square error of the best Hadamard network is approximately 4 times lower than the one of the best binary network.

We will now compute the dependency terms, like in Section 2.4. the interest will be if the first 8 nodes, which are now relevant, will have large dependency terms or not, and if they sum up to 0. Table 4.9 shows the dependency terms for all output values, for the 8 redundant node Hadamard-transformed case.

Table 4.10
Dependency terms for the 8 redundant nodes cases

dependency at node :	sample 1	sample 2	sample 3	sample 4
1	0.00 582	0.01 310	0.01 250	0.01 110
2	0.00 585	0.02 380	-0.00 072	0.01 160
3	0.00 583	0.01 920	0.00 163	0.01 610
4	0.02 160	0.01 050	0.01 030	0.02 190
5	0.02 410	0.00 204	0.00 993	0.01 600
6	0.00 741	0.00 276	0.00 575	0.00 790
7	0.00 270	0.00 493	0.00 869	0.00 683
8	0.00 741	0.00 887	0.01 360	0.00 956
9	-0.01 010	-0.01 050	-0.00 775	-0.01250
10	-0.01 020	-0.01 070	-0.00 773	-0.01270
11	-0.01 010	-0.01 080	-0.00 784	-0.01270
12	-0.01 000	-0.01 090	-0.00 753	-0.01270
13	-0.00 997	-0.01 050	-0.00 755	-0.01 250
14	-0.01 000	-0.01 070	-0.00 760	-0.01 270
15	-0.01 010	-0.01 070	-0.00 788	-0.01 260
16	-0.01 010	-0.01 050	-0.00 781	-0.01 260
average dependencies	0.00 000	0.00 000	0.00 000	0.00 000

Now, the dependency terms over all output values cancel out. However, the terms for the first 8 values, which contain the a posteriori probabilities, are all positive. This explains, why the decrease of the mean square error between the 0 and the 8 redundant node case is much less then the expected 50 %. Table 4.10 will now show the dependency terms for the first 8 nodes for 0, 8, 24 and 56 redundant nodes. Figure 4.7 shows how the average dependency of the first 8 nodes over the 4 samples is effected by the increase of the hidden nodes.

Table 4.11
Average dependencies of the first 8 nodes

sample #	1	2	3	4	average dependency
0 redundant nodes	0.00 000	0.00 000	0.00 000	0.00 000	0.00 000
8 redundant nodes	0.01 010	0.01 070	0.00 771	0.01 260	0.01 028
24 redundant nodes	0.00 822	0.01 070	0.01 190	0.01 010	0.01 023
(56redundant nodes	0.00 932	0.00 954	0.00 694	0.00 889	0.00 867

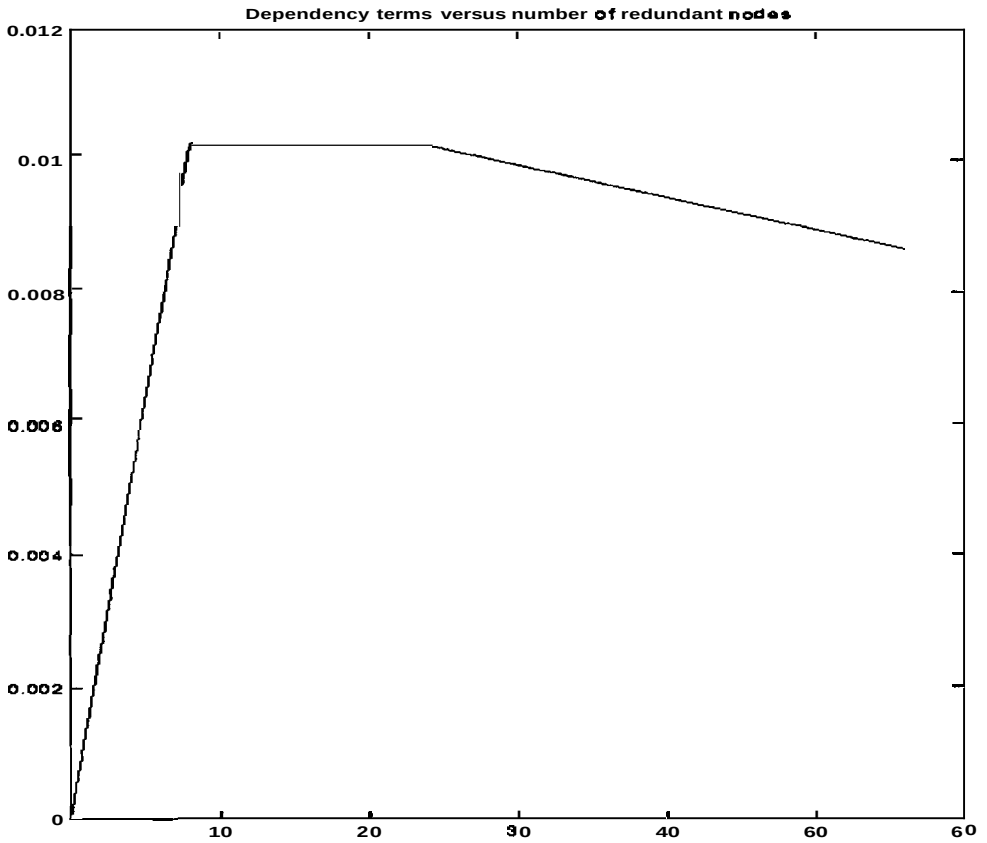

Fig. 4.7 Dependency terms versus number of output nodes

Apparently, the dependency terms are the highest for only 8 redundant nodes, and decrease slowly with the addition of more redundant hidden nodes. the magnitude of the dependency terms in Table 4.10 accounts for most of the observed output mean square error. For the average variance σ^2, the reduction is no longer given by 1/N, in comparison with the variance before the inverse Hadamard-transform. 1/N marks the lowest bound, while the actual variance reduction is much lower, due to the dependency terms. Still, the average performance improves with the addition of the redundant nodes, especially when a large number is added.

4.6 **Conclusions**

In comparison with the Parzen density estimation, our network now performed better in both classification and the mean square error. The only remaining advantages of the 2-dimensional Parzen density classifier are the extremely fast training time and a smaller mean error. To store the lookup tables we still need significantly more memory than for the neural network weights.

Previously we discussed advantages of the neural networks, like the absence of a smoothing parameter, the possibility to use high dimensional input data, etc. Now, with the introduction of the redundant nodes, we obtain a classifier that outperforms the statistical methods both in classification and mean square error, that is nonparametrical, has no smoothing parameters to adjust, and can be used for higher dimensions easily.

The only remaining advantages of the Parzen density classifier are the shorter training times, and for low mean square error, the faster testing times as well.

5. CONVERGENCE ISSUES

5.1 Introduction

We have introduced a new output representation and added redundant nodes for the neural network learning. So far we examined the results with respect to testing performance in classification and a posteriori probability estimation. In this chapter we will discuss speed of convergence and, the likelihood of getting stuck in a local minimum when the initialization used is imperfect.

In the backpropagation algorithm used here, the gradient, and therefore the weight adjustment in the output layer, are proportional to the error at the output, D - Y, where Y is the actual output vector. This suggests that the speed of learning not only depends on the chosen input representation, but also on the chosen output representation.

When we add redundant output nodes, the error surfaces for the output layer remain the same. However, the weight changes in the hidden layer depend on the backpropagated error of the output layer. Now, if we change the structure of that layer, we are quite likely to change the error backpropagated to the hidden layer and therefore the weight adaptation in the hidden layer itself. This will be discussed in Section 5.3.

5.2 Convergence of Hadamard-Transformed Output Networks

5.2.1 Effects of the Hadamard-transformed output representation

The Hadamard-transformed output representation is orthogonal. This means the inner product of two different columns is zero. The Hamming distance is N/2 for each column. compared to a Hamming distance of 2 for the binary output vectors.

Hence, the spheres in the hyperspace are further apart for Hadamard-transformed

outputs than for binary outputs.

The weight adaptation for the backpropagation is given by [15]

$$\Delta w_{jkl}(i) = \alpha\, \delta_{jl}(i)\, y_{kl}(i) \tag{5.1}$$

where 1 is the layer index, j is the j^{th} neuron in layer 1, and k is the k^{th} neuron in the next layer. $\delta_j(i)$ is the local gradient, and $y_{kl}(i)$ is the input of neuron k to neuron j. α is the learning rate, and i stands for the i^{th} input vector.

Since we use a linear output layer, the gradient at the output layer (layer number 3) becomes

$$\delta_{j2}(i) = e_j(i) = d_j(i) - y_j(i) \tag{5.2}$$

For the hidden layer, δ_{j1} is given by

$$\delta_{j1}(i) = \varphi(v_j(i)) \sum_k \delta_m(i) w_{mj}(i) \tag{5.3}$$

where $\varphi(v_j(i))$ is the derivative of the activation function of the hidden layer.

Clearly, using an orthogonal output representation instead of a binary will effect the error in Eq.(5.2). therefore, the gradient in Eq. (5.3) will be changed as well, and hence the weight adjustments in both the hidden and the output layer will be effected. Weather the effects will improve or slow convergence has to be determined experimentally.

5.2.1 Experimental Results

As discussed in Section 2.3, we cannot use the mean squared error to show the progress of learning. We will instead display the number of correctly classified vectors of the training set. Figure 5.1 shows 3 typical training patterns with both the Hadamard-

transformed and the binary output representation, using the 2 - dimensional training data. Figure *5.2* shows **3** typical training patterns with the **3** - dimensional input data. We decided not to average over a sum of different networks, since the practical interest is to evaluate the performance of a specific classifier and not of a set. Using the average would smooth out some important characteristics.

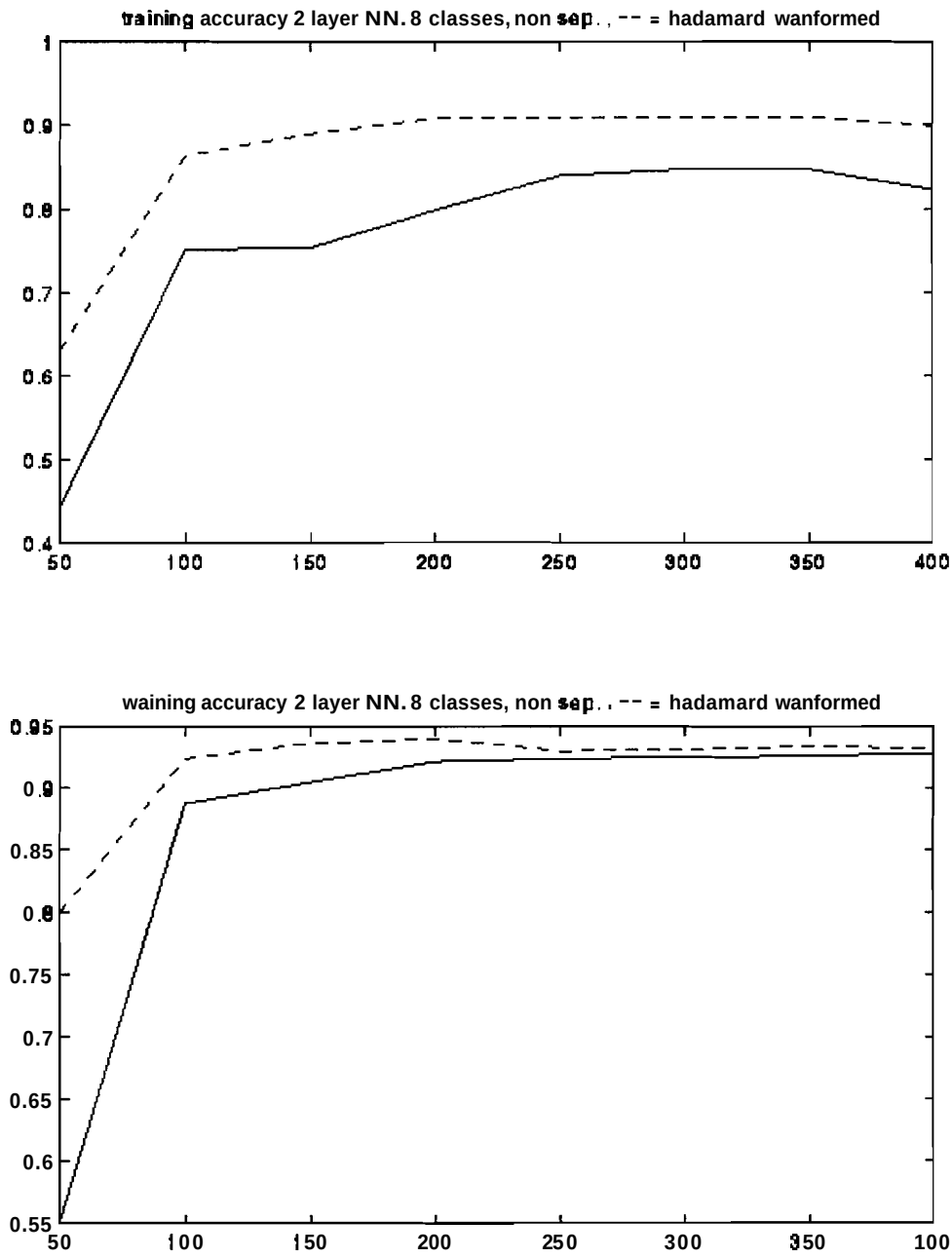

Fig. 5.1 Training classification versus number of sweeps

2 - D case

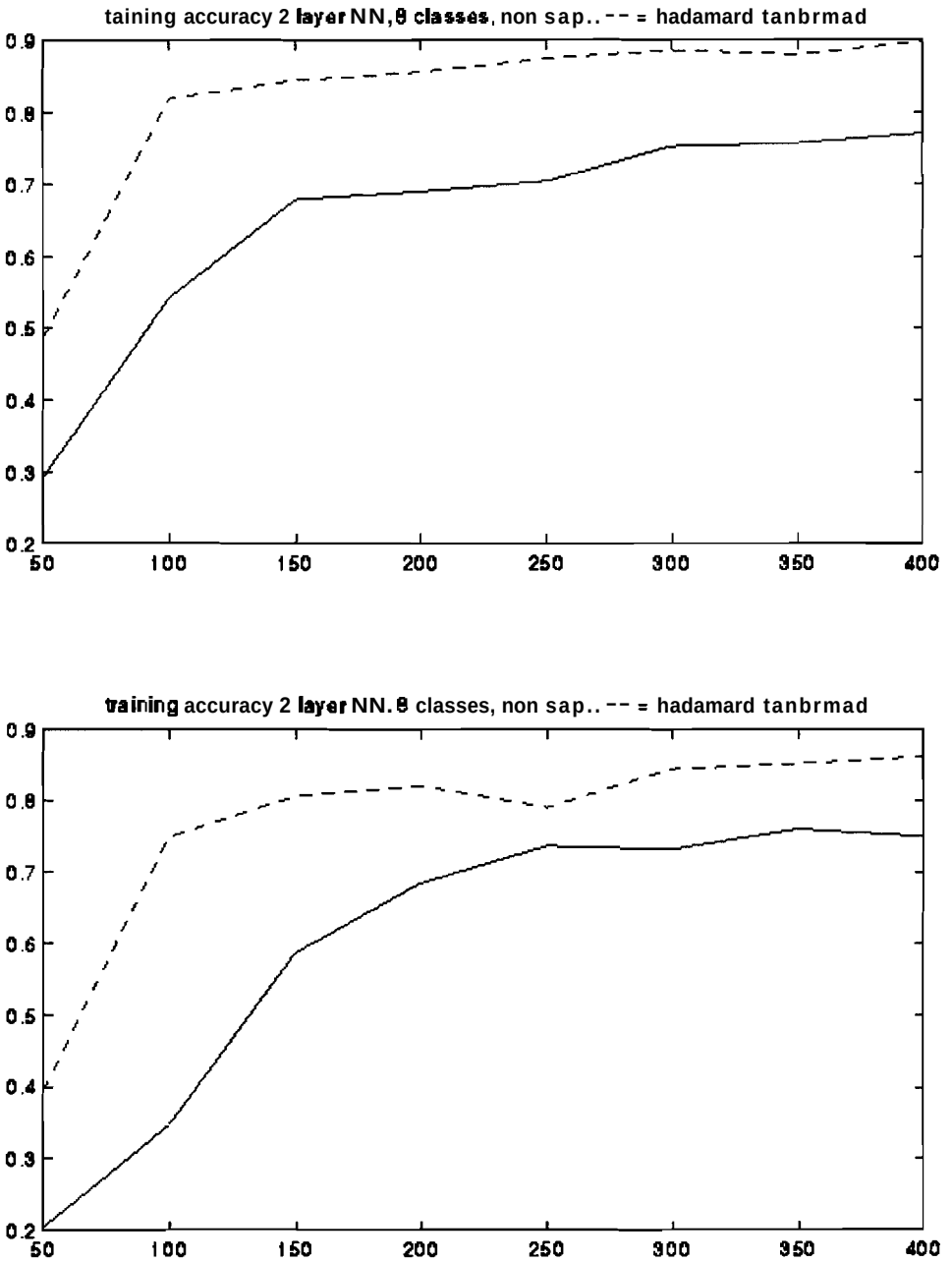

Fig. 5.2 Training classification versus number of sweeps, 3 - D case

Clearly, the Hadamard-transformed neural network converges faster then the binary network, especially during the first sweeps. The effects are more obvious in the 3 - dimensional case. This suggests that the benefits of the Hadamard-transformed convergence will be stronger with more complex problems.

Tables 5.1 (a), 5.1(b) show the average number of iterations of six samples and CPU-time needed until the network reached 70, 80 and 85 % correctly separated patterns in the 3 dimensional case. This time we decided to use the average instead of the single cases, in order to show the average gains. Due to computational limitations, we only show the results after 600 sweeps. Our previous findings show that the binary network will eventually reach the 85 % classification later.

Table 5.1 (a)
Training results achieved within the first 600 sweeps

0 redundant nodes, 55 hidden nodes

training classification	70%	80%	85%
binary	280	400	560
Hadamard	120	220	360
binary	360	---	---
Hadamard	100	180	200
binary	380	520	---
Hadamard	180	280	360
binary	220	440	---
Hadamard	100	120	200
average			
binary	310	340	560
Hadamard	125	200	280

Table 5.1 (b)
Training times needed to reach the classification percentage

0 redundant nodes. 55 hidden nodes

training classification	70%	80%	85%
binary	1 110	1 620	2 290
Hadamard	422	848	1 450
binary	1 450	---	---
Hadamard	336	676	761
binary	1 520	2 120	---
Hadamard	670	1 100	1 440
binary	840	1 780	---
Hadamard	337	423	764
average			
binary	1 230	1 380	2 290
Hadamard	441	762	1 104

Clearly, the Hadarnard-transformed networks converge faster, reaching 70% correct classification within 113 of the time of the binary network, and 85 % at about less then 1/2 of the time needed for the binary network.

The comparison between the 2-D case and the 3-D case shows that the convergence speedup is larger for the 3-D case. Hence, the benefits of using Hadamard-transformed output representations can be expected to be larger for more complex problems.

5.3 Convergence of networks with redundant nodes

We introduced redundant hidden nodes in Chapter 4. Now, equation (5.3) shows that the weight adaptation in the hidden layer is proportional to the sum of the error of all output nodes connected with the hidden node. by adding redundant output terms, we add more terms to the sum in equation (5.3). This will affect the adjusting of the weights in the hidden layer. However, it is impossible to give a theoretical treatment if this will affect the network convergence speed in a positive or negative way.

Figure **5.3** will show the effects for two samples, in the **2** dimensional case. In each sample we trained a neural network with no, 8, 16 and **32** redundant nodes. The number of hidden nodes was set a little larger than the number of output nodes, like discussed in Chapter 4. We used the previously used **2** dimensional 100 vector training set.

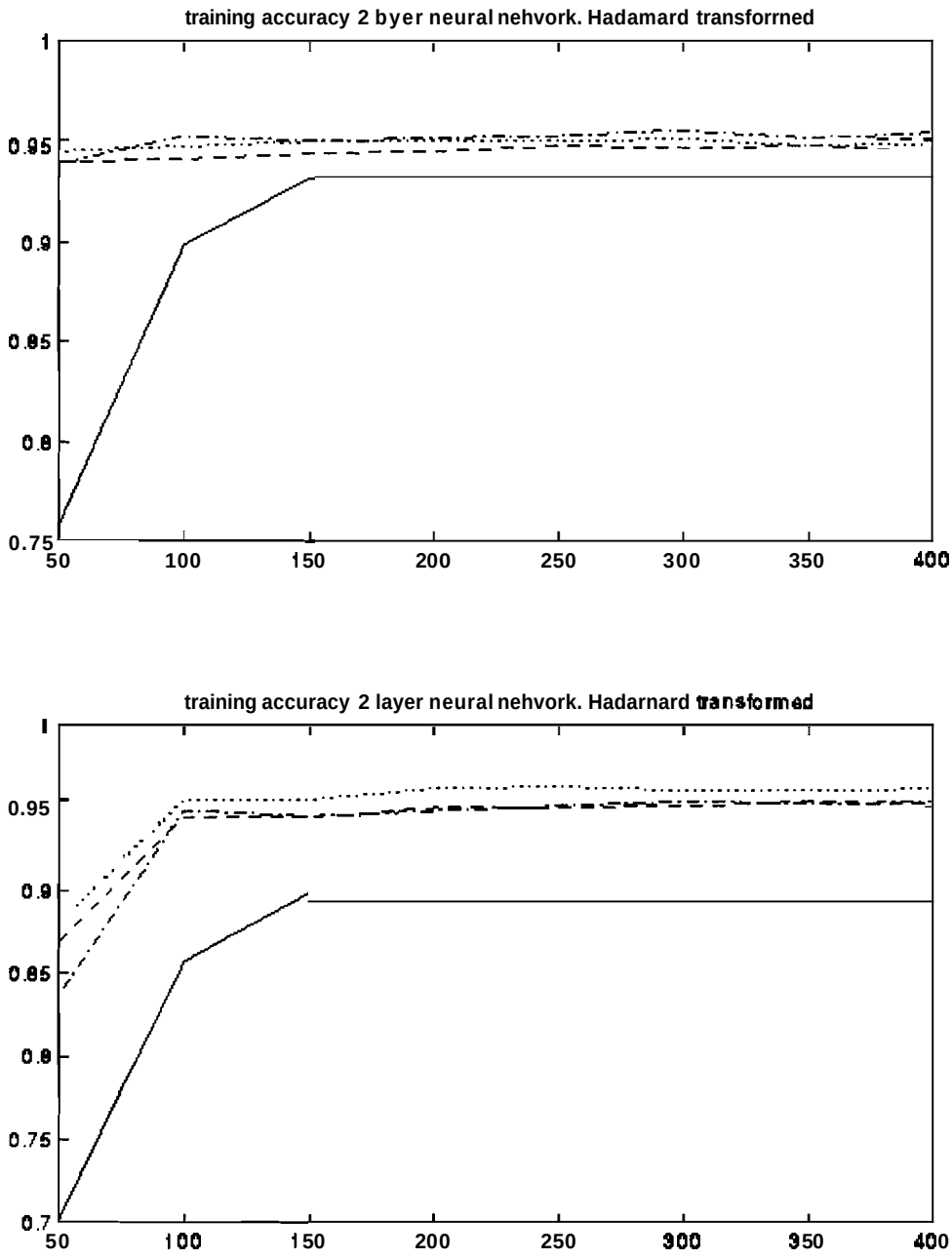

Fig. 5.3 Training accuracy versus number of sweeps

solid = 0, dashed = 8, dash-dot = 16 and dotted = 32 redundant nodes

In order to verify if the improved convergence is the result of the addition of the hidden nodes or of the redundant nodes, we then ran 3 samples with a constant amount of hidden nodes, 55. The results for 0, 8, and 16 redundant nodes are shown in Table 5.2.

Table 5.2 (a)
Training results achieved within the first 600 sweeps

24 redundant nodes, 55 hidden nodes

training classification	70%	80%	85%
binary	500	---	---
Hadamard	140	200	---
binary	---	---	---
Hadamard	140	180	500
binary	---	---	---
Hadamard	180	180	440
binary	---	---	---
Hadamard	120	140	380
average			
binary	500	n. a.	n. a.
Hadamard	145	175	330

Table 5.2 (b)
Training results achieved within the first 600 sweeps

8 redundant nodes, 55 hidden nodes

training classification	70%	80%	85%
binary	300	600	---
Hadamard	100	180	280
binary	400	---	---
Hadamard	100	120	300
binary	420	560	0
Hadamard	140	220	240
binary	340	500	0
Hadamard	100	120	160
average			
binary	365	415	n. a.
Hadamard	110	160	245

Table 5.2 (c)
Training results achieved within the first 600 sweeps

0 redundant nodes, 55 hidden nodes

training classification	70%	80%	85%
binary	280	400	560
Hadamard	120	220	360
binary	360	---	---
Hadamard	100	180	200
binary	380	520	---
Hadamard	180	280	360
binary	220	440	---
Hadamard	100	120	200
average			
binary	310	340	560
Hadamard	125	200	280

Hence, the addition of more hidden nodes reduces the number of iterations needed. The addition of redundant output nodes has no or little effect on the number of iterations needed for convergence in the Hadamard-transformed case. This can be explained by the fact that the desired output of the redundant nodes is constant for all vectors. It slows the convergence of the binary case.

5.4 Initialization Problems

So far we used an initialization procedure based on the algorithm by [Nguyen Widrow]. Instead of initializing with small random variables, the hidden neurons are initialized to cover the whole range of the input data. First all the weights are set to uniform random values between the minimum and the maximum of the input data. Then the weight magnitudes are readjusted so that each hidden neuron is linear only over a small interval. Then, the magnitude of W_i is set to:

$$|W_i| = f\, H^{\frac{1}{P}} \tag{5.4}$$

where H is the number of hidden nodes and P is the number of input dimensions. f is a factor smaller than one, to give some overlap between the intervals of the different hidden nodes. The bias for the hidden node is then set to a uniform random variable between $-|W_i|$ and $+|W_i|$. This distributes the hidden nodes over the whole input space equally. The output layer weights are set to small random variables.

Another initialization scheme is to choose the weights randomly in the range

$$\left(-\frac{2.4}{F_j}, +\frac{2.4}{F_j}\right) \tag{5.5}$$

where Fj is the fan-in, the total number of inputs to neuron j in the network [15].

W also ran an initialization where the initial weights were distributed gaussian between 0 and 1. The idea of setting all weights to a positive value is to complicate the learning process and increase the likelihood of it to get stuck in a local minimum. This will then show whether any of the changes proposed in this thesis will enable the neural network to avoid local minima, or decrease the ability to overcome local minima.

Table 5.3 shows the iterations needed to reach 70, 80, 85 and 90 % in the 3-dimensional classification problem. 25 hidden nodes were used in all cases.

In the case of poor initialization, both networks get stuck in a local minimum twice. However, there are 2 more cases were the binary network fails to reach acceptable classification accuracy within a reasonable number of training iterations. The Hadamard network was performing better, though not as good as with both the heuristic range and the Nguyen Widrow algorithm. Figure 5.4 show 2 training cases.

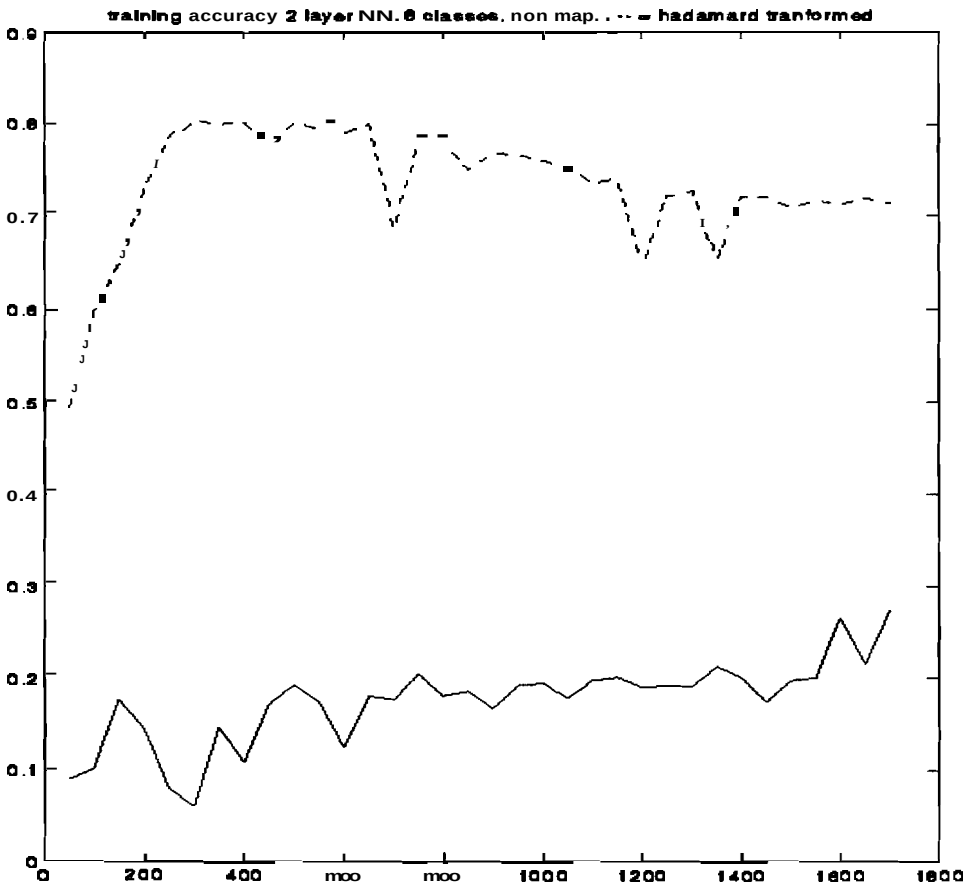

Fig. 5.4 Backpropagation with binary representation stuck in a local minimum

In general, even if the local minimum is overcome, the learning curve is no longer asymptotic. It instead shows slow increase which gets halted occasionally.

Using Hadamard-transformed output representations worked better with all the 3 different initializations. Of the 3 different initializations, the Nguyen/Widrow initialization and the uniform random initialization give the best results.

Table 5.3 (a)
Iterations needed to reach certain training performance
Nguyen Widrow initialization

train. class	70%	80%	85%	90%
binary	150	450	500	---
Hadamard	100	200	400	1 250
binary	150	400	400	---
Hadamard	150	200	550	---
binary	150	500	500	---
Hadamard	100	200	200	850
binary	250	250	250	---
Hadamard	100	300	550	1 300
binary	250	450	550	1 350
Hadamard	100	150	550	1 650
binary	200	200	350	---
Hadamard	200	700	850	---
average				
binary	175	342	492	not computed
Hadamard	142	325	450	not computed

Table 5.3 (b)
Iterations needed to reach certain training performance
Uniform Random initialization

train. class	70%	80%	85%	90%
binary	400	450	500	1 000
Hadamard	300	400	400	1 200
binary	500	550	550	750
Hadamard	250	300	350	1 850
binary	400	500	500	500
Hadamard	200	250	250	300
binary	450	450	500	---
Hadamard	200	200	200	1 350
binary	450	450	550	---
Hadamard	200	200	350	450
binary	450	450	550	550

Hadamard	200	250	300	650
average				
binary	442	475	525	n.a. (700)
Hadamard	225	267	308	967

Table 5.3 (c)
Iterations needed to reach certain training performance
Positive gaussian values as initialization

train. class	70%	80%	85%	90%
binary	---	---	---	---
Hadamard	---	---	---	---
binary	1 750	---	---	---
Hadamard	400	500	550	---
binary	1 850	---	---	---
Hadamard	450	600	1 300	1 700
binary	---	---	---	---
Hadamard	500	600	---	---
binary	1 450	1 600	---	---
Hadamard	450	550	900	---
binary	---	---	---	---
Hadamard	---	---	---	---

6. CONCLUSIONS

6.1 Summary of the Results

We have introduced an output representation which reduces the mean square estimation error of a neural network classifier drastically. Our experimental results confirmed the model we have set up for the expected reductions. We have then modified the new output representation by enlarging its size. This yields the expected benefits, up to the point where the computational overhead becomes excessive.

The modifications we introduced reduced the mean square error of a neural network classifier to about 1/8 of its previous value. This is accompanied be classification results which almost reach the maximum possible classification. This is especially remarkable, since it is usually possible to overcome the last 2 - 3 percent for maximum classification at great cost and effort.

The proposed changes make neural networks a powerful, completely non-parametric a priori probability density estimator. It needs little more memory to store the distributions than the classic backpropagation, there are no smoothing parameters to adjust, and the density can be estimated more accurately than with the Parzen density estimation. It can easily be used for higher dimensional probability density estimation as well.

The only advantage remaining for the Parzen density estimation is the shorter training time in the 1- and 2- dimensional case to generate the lookup table. However, already for the 3-dimensional case, the product Kernel becomes so excessive to compute that almost no practical use is known. The testing times differ. For small an medium network complexity, the network performs faster, while for high complexity, i.e. many redundant nodes, the Parzen density estimation is faster.

6.2 Possible Applications

Lower mean square estimation error and better classification are interesting for all classification problems. However, there might be special interest in applying these techniques to problems where misclassifications produce great cost, i. e. in signature verification, medical engineering etc. In such problems classification confidence is of great importance. Hence, by reducing the error variance, we would be able to raise the rejection borders significantly, allowing more input data to be classified, and less to be rejected.

The ability of the Hadamard-transformed neural network to overcome local minima and to speed up convergence will be especially beneficial for complex problems. Those problems will both benefit from the reduction of the variance of the output error and the effects of the orthogonalty of the desired output.

Application of the Hadamard-transformed representation to problems outside of classification, i. e. time series prediction, functional approximation or image compression might be difficult. In those cases, when we take the Hadamard-transform of the desired output, we would get a higher range. The desired outputs are analog between some values, and taking the Hadamard-transform would produce large output values. This will then eat up the benefits of taking the inverse Hadamard-transform.

6.3 Direction for Further Research

Further research might bring down the level of the output noise even further. One possible place to look for that will be the "saturation phase", where the neural network basically has found its minimum, but keeps oscillating slightly, since the desired output representation is fairly different and produces an error affecting the learning as discussed in Chapter 5.

Other interesting areas are the parallel implementation of redundant networks. This would enable us to find the point where increasing the number of redundant output neurons further might be overcome by the general error increase due to complexity. Another interesting field will be the effects of Hadamard-transformed output representations to supervised, but not backpropagation, learning algorithms, like the

PSNN presented in [4].

Concerning the initialization problems, investigations into other algorithms which are more likely to get stuck in a local minimum then our approach with an adaptive learning rate might show the effects of initialization, and output transformations overcoming a local minimum better. Investigations of the sources of noise inside a neural network might be promising as well.

LIST OF REFERENCES

[1] M. D. Richard, R. P. Lippman, "Neural Network Classifiers Estimate Bayesian a posteriori probabilities" *Neural Computations* MIT Press 1991.

[2] E. A. Wan, "Neural Network Classification: A Bayesian Interpretation" *IEEE Transactions on Neural Networks,* Vol. 1. No4, December 1990.

[3] H. White, "Learning in Artificial Neural Networks: A Statistical Perspective" *Neural Computations* vol. 1, no. 4, pp. 425-464 MIT Press 1989.

[4] O. K. Ersoy, D. Hong, "Parallel, Self-Organizing, Hierarchical Neural Networks - 11" *IEEE Transactions on Industrial electronics,* Vol. 40 No.2 April 1993.

[5] M. Harwit N. J. A. Sloane, *Hadamard-transform Optics* Academic Press, Inc., New York 1979.

[6] S. S. Agaian, *Hadamard Matrices and Their Applications* Springer Verlag, Berlin 1985.

[7] Keinosuke Fukunaga, *Introduction to Statistical Pattern Recognition* Academic Press, San Diego, California, second edition 1990, chapters 6 and 10.

[8] David W. Scott, *Multivariate Density Estimation* John Wiley & Sons, New York 1992, chapters 3 and 6.

[9] J. Nazari, "Development of Algorithms for Generalization, Convergence and Parallelization in Neural Networks" *Chapter 6, Output representations* Ph. D. dissertation, Purdue University 1994.

[10] J. Nazari and O.K. Ersoy, "Utilization of Hadamard Matrices for Output Representation in Neural Networks" Proceedings of *European Conference on Circuit Theory and design,* ECCTD-95, pp. 691-694, Istanbul, Turkey, August 1995.

[11] J. Nazari and O.K. Ersoy, "Application of Error Control Codes and Hadamard Matrices in Solving Classification Problems with Neural Networks" ***Proceedings of ICEE-95 conference,*** pp. 210-217, Tehran, May 1995.

[12] A. Papoulis, ***Probability, Random Variables and Stochastic Processes*** McGraw Hill, third edition 1991.

[13] D.J. Munro, O.K. Ersoy, M.R. Bell, J. S. Sadowski, "A Weighted Least Squares Algorithm for Neural Network Learning in Recognition of Low Probability Events" ***in press.***

[14] D. Nguyen, B. Widrow, "Improving the Learning Speed of 2 - Layer Neural Networks by choosing Initial Values of the Adaptive Weights".

[15] S. Haykin, ***Neural Networks*** Macmillan New York 1994.