

FACE RECOGNITION USING NEURAL NETWORKS

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

MASTER OF TECHNOLOGY

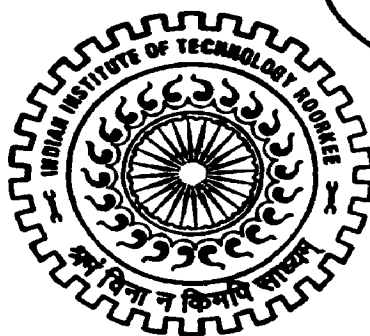
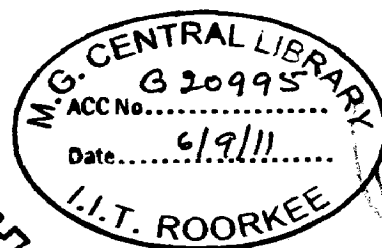
in

ELECTRONICS AND COMPUTER ENGINEERING

(With Specialization in Control and Guidance)

By

VIJAYENDRA KUMAR



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247 667 (INDIA)

JUNE, 2011

CANDIDATE'S DECLARATION

I hereby declare that the work, which is presented in this dissertation report entitled, **"FACE RECOGNITION USING NEURAL NETWORKS"** towards the partial fulfillment of the requirements for the award of the degree of **Master of Technology** with specialization in **CONTROL AND GUIDANCE**, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2009 to June 2010, under the guidance of **Dr. M. J. Nigam, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other Degree.

Date: 27.06.2011

Place: Roorkee

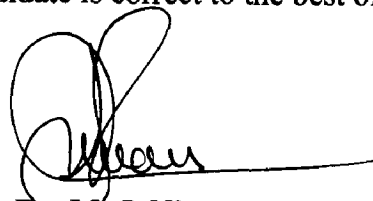

VIJAYENDRA KUMAR

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 27.06.2011

Place: Roorkee



Dr. M. J. Nigam,

Professor, E&C Department,

IIT Roorkee,

Roorkee – 247667 (India).

ACKNOWLEDGEMENTS

I would like to extend gratitude to my guide, Dr. M. J. Nigam for his intellectual guidance, attention and constant encouragement that inspired me throughout my dissertation work. I am greatly indebted to him for teaching me the sweetness of self learning. Also his critical arguments in correcting the thesis made me to read in between the lines.

I would also like to thank the Lab staff of Control Systems Lab, Department of Electronics and Communication Engineering, IIT Roorkee for providing necessary facilities.

I gratefully acknowledge my sincere thanks to my family members for their inspirational impetus and moral support during course of this work.

I thank to all my friends, who have graciously applied themselves to the task of helping me with morale support and valuable suggestions. Finally, I would like to extend my gratitude to all those persons who directly or indirectly contributed towards this work.

VIJAYENDRA KUMAR

ABSTRACT

Face recognition is a computer application of recognizing persons by their facial images, which may vary with makeover, expressions, age or due to noise added during image acquisition or during transmission over a network. Work on Face recognition is going on from around last twenty years, and has got immense future scope and great commercial value. Face recognition is being used for Access Control, Protection and Security, Industrial Automation and Robotics etc.

Face recognition system is based on principals of various fields like Image Processing, Linear Programming, Statistics and Neural Networks etc. Various Image Processing Techniques like Laplacian, Laplacian of Gaussian, and Histogram Equalization etc. are used for image preprocessing stage to increase sharpness and dynamic range. Facial images contain redundancies which do not help in recognition but increase processing overhead, Techniques like Fourier Transform and Wavelet Packet Decomposition are used for feature generation or dimensionality reduction. Concepts from Linear Programming and statistics like Principal Component Analysis, Singular Value Decomposition, and Independent Component Analysis etc. are also used for generating features of reduced dimensions from original data, which has same recognition potential as original data. Neural Networks are powerful tool for recognition type of computation. They are trained on representative set of images, once trained they can recognize images they have never seen. Most important feature of such networks are that they tolerate noise extremely well.

In this dissertation an algorithm for improved Face Recognition has been developed. Laplacian has been used for preprocessing the facial images. Singular Value Decomposition has been used for feature generation dimensionality reduction. Neural Networks have been used for recognition purposes. In order to improve noise tolerance of developed face recognition system Neural Networks has been trained by deliberately generated noisy data. Further, in training Neural Networks a technique Resilient Backpropagation has been used which made the training faster.

Face Recognition system developed has an accuracy of 91% under zero random noise condition. Further a study of variation of accuracy with noise has been carried out and results are presented in chapter 6.

CONTENTS

Declaration	i
Acknowledgements	ii
Abstract	iii
Contents	iv
1. Introduction	1
1.1 Problem Statement	3
1.2 Problem Analysis	4
1.3 Literature Review	5
1.3.1 Eigenface-based Recognition	5
1.3.2 3D Face Recognition	6
1.3.3 Applications of Face Recognition	6
1.4 Organization of the Thesis	7
2. Image Preprocessing	8
2.1 Laplacian Operator	8
2.2 Unsharp Masking and Highboost Filtering	11
2.3 Laplacian of Gaussian	12
3. Dimensionality Reduction and Feature Generation	14
3.1 Basis Vectors and Images	15
3.2 Positive Definite and Symmetric Matrices	17
3.3 Covariance and Correlation	18
3.3.1 Definition of the Covariance	18
3.3.2 Interpretation of the Covariance	19
3.3.3 Correlation Coefficient	20
3.3.4 Properties of the Correlation Coefficient	21
3.3.5 Mean Vector and Covariance Matrix	21
3.3.6 Correlation Matrix Diagonalization	22
3.4 Principal Component Analysis	23
3.4.1 Mean Square Error Approximation	24
3.4.2 Steps for Calculation of Principal Component of given Data	25

3.4.3 Example Problem	26
3.5 Singular Value Decomposition	28
3.5.1 Proof of Singular Value Decomposition	29
3.5.2 Low Rank Approximation	30
3.5.3 Dimensionality Reduction	31
3.5.3 Example Problem	33
4. Artificial Neural Systems	34
4.1 Neural Processing	35
4.1.1 Perceptron	36
4.2 Learning and Adaptation	38
4.2.1 Supervised and Unsupervised Learning	38
4.3 Neural Network Learning Rules	40
4.3.1 Hebbian Learning Rule	41
4.3.2 Perceptron Learning Rule	42
4.3.3 Delta Learning Rule	43
4.3.4 Widrow-Hoff Learning Rule	44
4.3.4 Winner Takes All Learning Rule	45
4.4 Neural Network Training	46
4.4.1 Backpropagation Algorithm	47
4.4.2 Resilient Backpropagation	48
5. Software for Face Recognition	50
5.1 Algorithm	50
5.2 Program for Computation of Principal Component Analysis	51
5.3 Program for Creation of Training Set	52
5.4 Program for Creation of Noisy Training Set	52
5.5 Program for Computation of Singular Value Decomposition	53
5.6 Program for Neural Network Training	54
5.7 Program for Simulation Neural Network	55
6. Results	58
6.1 Confusion Matrix for Various Noise Levels	60

6.2 Performance Plot	70
6.3 Worst Case Accuracy Plot	71
7. Conclusion and Future Research Suggestion	72
References	73

LIST OF FIGURES

Figure	Page no.
Figure 1.1 Dummy Track	2
Figure 1.2 The basic stages involved in the design of a classification system.	2
Figure 1.3 Facial variations with expression and make over	3
Figure 1.4 Facial variations with random noise	3
Figure 1.5 Flow diagram of generalized face recognition algorithm using neural networks	5
Figure 2.1 Laplacian Mask	9
Figure 2.2 Laplacian Mask Including Diagonal Directions	10
Figure 2.3 input image	10
Figure 2.4 Laplacian image	10
Figure 2.5 input –Laplacian	10
Figure 2.6 North Pole of Moon	11
Figure 2.7 Laplacian Filtered	11
Figure 2.8 Filtered using Fig. 2.2 Mask	11
Figure 2.9 3-D Plot of LoG Filter $\sigma = 0.5$	13
Figure 2.10 Input image	13
Figure 2.11 LoG filtered image	13

Figure 2.12 Mask used for Filtering	13
Figure 4.1a Autoassociation	35
Figure 4.1b Heteroassociation	35
Figure 4.2a Classification	36
Figure 4.2b Recognition	36
Figure 4.3 A Perceptron	37
Figure 4.4 Tansig Function	37
Figure 4.5 Logsig Function	37
Figure 4.6 Thresholding Function	38
Figure 4.7a Supervised Learning	39
Figure 4.7b Unsupervised Learning	39
Figure 4.8a Distinguishable Patterns	40
Figure 4.8b Undistinguishable Patterns	40
Figure 4.9 Perceptron under Training	41
Figure 4.10 Perceptron Learning Rule	43
Figure 4.11 Delta Learning Rule	44
Figure 4.12 Winner Takes All Learning Rule	46
Figure 4.13 Neural Network Training	46
Figure 6.1 Various Noise levels	59
Figure 6.2a Training Images	59
Figure 6.2b Test Images	59
Figure 6.3 Performance Plot	70
Figure 6.4 Worst Case Accuracy Plot	71

LIST OF TABLES

Table			Page no.
Table I.	Noise Level = 0;	91% Recognition Rate	60
Table II.	Noise Level=0.05;	91% Recognition Rate	60
Table III.	Noise Level=0.1;	89% Recognition Rate	61
Table IV.	Noise Level=0.15;	86% Recognition Rate	61
Table V	Noise Level=0.2;	83% Recognition Rate	62
Table VI.	Noise Level=.25;	78% Recognition Rate	62
Table VII.	Noise Level=0.3;	78 % Recognition Rate	64
Table VIII.	Noise Level=0.35;	72 % Recognition Rate	64
Table IX	Noise Level=0.4;	72 % Recognition Rate	65
Table X	Noise Level=0.45;	67 % Recognition Rate	65
Table XI	Noise Level=0.5;	64 % Recognition Rate	66
Table XII	Noise Level=0.55;	62 % Recognition Rate	66
Table XIII	Noise Level=0.6;	54 % Recognition Rate	67
Table XIV	Noise Level=0.65;	54 % Recognition Rate	67
Table XV	Noise Level=0.7;	53 % Recognition Rate	68
Table XVI	Noise Level=0.75;	48 % Recognition Rate	68
Table XVII	Noise Level=0.8;	38 % Recognition Rate	69
Table XVIII	Noise Level=0.85;	34 % Recognition Rate	69
Table XIX	Noise Level=0.9;	27 % Recognition Rate	70
Table XX	Noise Level=0.95;	24 % Recognition Rate	71

Chapter 1: Introduction

A Face recognition system is a computer application for automatically identifying or verifying a person from a digital image which may vary with age, makeover and expressions and under noisy conditions. In recent years, many algorithms have been proposed on the problem of face recognition. Recent survey of these algorithms can be found in [1]. Face recognition system uses algorithms from fields of Image Processing, Linear Algebra, Statistics and soft computing techniques like Neural Network, Fuzzy Logic etc. Face recognition system is studied under the broad topic of Pattern Recognition.

Pattern Recognition is scientific discipline whose goal is the classification of object into number of categories or classes. Depending on application, patterns can be images or signal waveform or any measurement that need to be classified. Pattern Recognition is an integral part of most machine intelligence systems built for decision making. Machine vision is an area in which pattern recognition is of importance. Machine vision is widely used in Automatic Control areas. A machine vision system captures images via a camera and analyzes them to produce descriptions of what is imaged. A typical application of a machine vision system is in the manufacturing industry, either for automated visual inspection or for automation in the assembly line. For example, in inspection, manufactured objects on a moving conveyor may pass the inspection station, where the camera stands, and it has to be confirmed whether there is a defect. Thus, images have to be analyzed online, and a pattern recognition system has to classify the objects into the “defect” or “nondefect” class. After that, an action has to be taken, such as to reject the offending parts. In an assembly line, different objects must be located and “recognized,” that is, classified in one of a number of classes known before hand. Then a robot arm can move the objects in the right place. Besides, Control applications pattern recognition can be use in number of different applications like, Character (letter or number) recognition, Handwriting recognition, Computer-aided diagnosis, Speech recognition, Data mining and knowledge discovery [9].

Various real life problems can be formulated as pattern recognition problem. For example, the problem is to design a robot whose task is to move on parallel tracks like railway tracks and find errors in it. Errors can be in the form of broken track, bulged in or bulged out

track. Robot has to identify the errors by producing different kind of signal for different kind of errors. figure 1.1 shows the dummy track and various errors. Conventional method for solving this problem is to design hardware with sufficient number of sensors array which can distinguish various kinds of situations, such as distinguishing between turns and errors and further between various types of errors. The other way to implement this is to use pattern recognition algorithm like, using soft computing techniques i.e. Neural Networks, Fuzzy Logic etc.

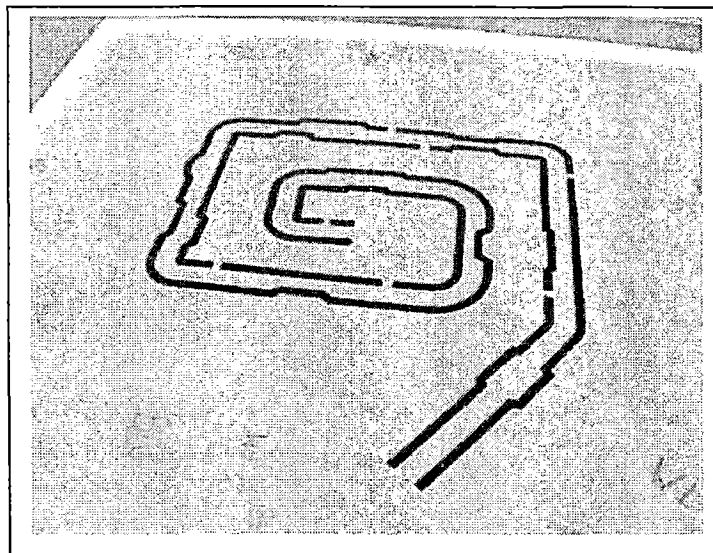


figure 1.1 Dummy Track

The robot in this case can be trained on representative set of track patterns and in recall phase it can classify, not only the training set but also the patterns it has never seen, which are similar to training pattern. In this case robot can generalize and also hardware dependencies are reduced.

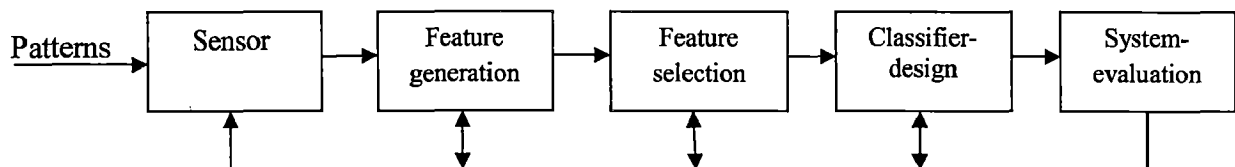


figure 1.2 The basic stages involved in the design of a classification system.

figure 1.2 shows the various stages followed for the design of a Pattern Recognition system. These stages are not independent as depicted by feedback arrows. Depending on the results, one may go back to redesign earlier stages in order to improve the overall performance [9].

Pattern recognition has a long history, but before the 1960s it was mostly the output of theoretical research in the area of statistics. As with everything else, the advent of computers increased the demand for practical applications of pattern recognition, which in turn set new demands for further theoretical developments. As our society evolves from the industrial to its postindustrial phase, automation in industrial production and the need for information handling and retrieval are becoming increasingly important. This trend has pushed pattern recognition to the high edge of today's engineering applications and research. Face recognition is one of the applications of Pattern Recognition.

1.1 Problem Statement

Face recognition is a pattern recognition task performed specifically on faces. Face recognition has become an interesting research area in vision system, image analysis, pattern recognition and biometric technology. An efficient face recognition system should recognize a person by his/her facial image which may vary with age, make ups, expressions and with noise. figure 1.3 shows facial variation of a person with expression and make over. figure 1.4 shows facial variation with noise. Noise added is normally distributed with mean zero.



figure 1.3 Facial variations with expression and make over

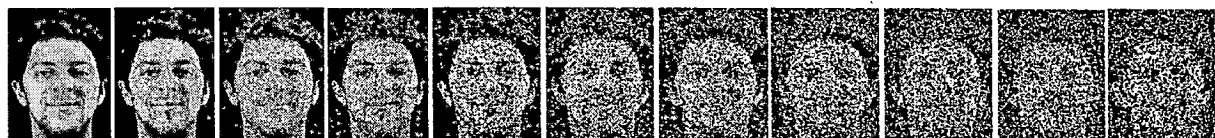


figure 1.4 Facial variations with random noise

1.2 Problem Analysis

Broadly, the algorithm of face recognition can be divided into three steps. Each step can be implemented in a number of ways:

- **Image Preprocessing:** Pre-processing of facial image is required to enhance some of its distinguishable features like moles, wrinkles and other small details [2]. Some of the algorithms that are used to perform this are Unsharp Masking [3], Laplacian of Gaussian [2], Laplacian [4] etc.
- **Feature Generation or Dimensionality Reduction:** The basic approach followed in this step is to transform a given set of measurements to a new set of features which acts as signature of original Data. Pixels of Facial images have high degree of correlation, to remove these redundancies; some of the algorithms are Principal Component Analysis [5], Wavelet Packet Decomposition [6], Singular Value Decomposition [2] etc.
- **Artificial Neural Networks:** Artificial Neural Networks function as parallel distributed computing networks. Depending on architecture, there are number of variations in neural networks which can be used for Face Recognition task, like Auto-associative Neural Networks [7], Hopfield Neural Networks [6], Feed-Forward Neural networks [2] etc. In contrast to conventional computers, which are programmed to perform specific tasks, most neural networks can be taught or trained with representative set of input and output data. Properly trained neural networks tend to give reasonable answers when presented with inputs that they have never seen. Typically, a new input leads to an output similar to the correct output for input vectors used in training that are similar to the new input being presented [8].

In the algorithm proposed, Laplacian has been used for image pre-processing; Singular Value Decomposition has been used for dimensionality reduction; and Artificial Neural Networks for classification. Reliability of the method was tested against Cambridge ORL face database. Ten different Faces with ten different expressions each, was picked .The proposed method was found to recognize the Faces with high recognition rate and under noisy conditions. figure 1.5 shows flowchart general algorithm using neural networks of face recognition.

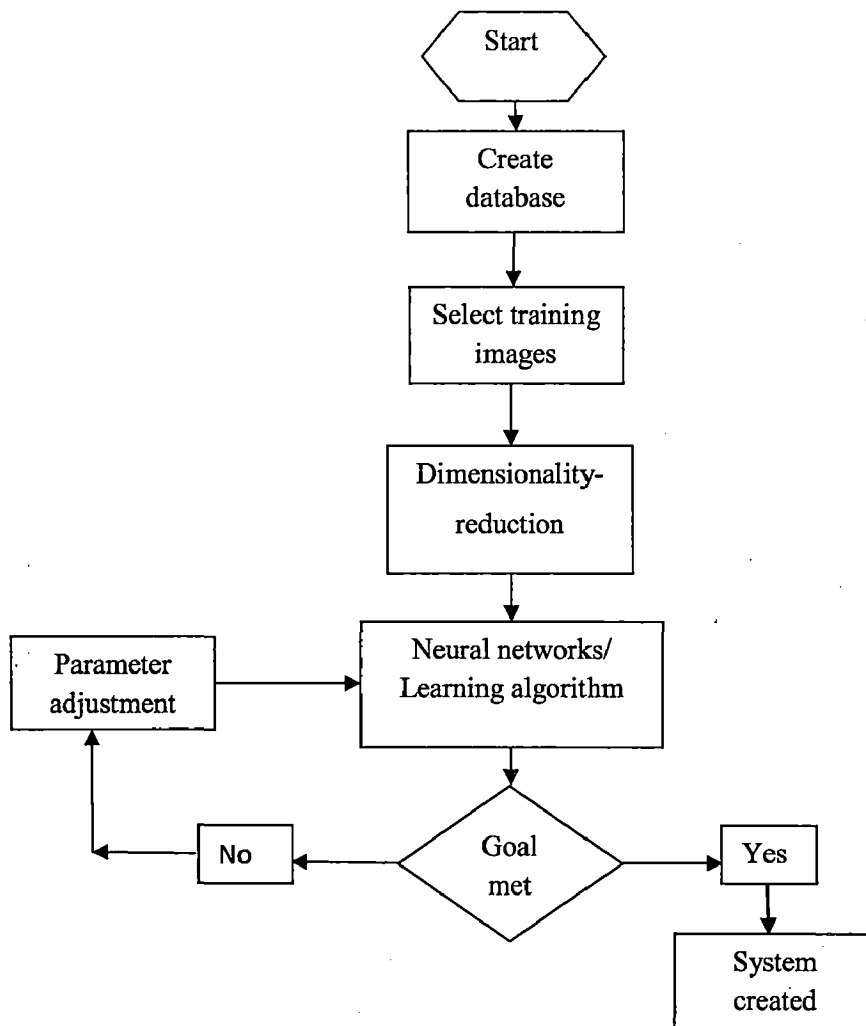


figure 1.5 Flow diagram of generalized face recognition algorithm using neural networks

1.3 Literature Review

1.3.1 Eigenface-based Recognition

2D face recognition using eigenfaces is one of the oldest types of face recognition. Turk and Pentland published the revolutionary “Face Recognition Using Eigenfaces” in 1991. The method works by analyzing face images and computing eigenfaces which are faces composed of eigenvectors. The comparison of eigenfaces is used to identify the presence of a face and its identity. There is a five step process involved with the system developed by Turk and Pentland. First, the system needs to be initialized by feeding it a set of training images of faces. This is used

these to define the face space which is set of images that are face like. Next, when a face is encountered it calculates an eigenface for it. By comparing it with known faces and using some statistical analysis it can be determined whether the image presented is a face at all. Then, if an image is determined to be a face the system will determine whether it knows the identity of it or not. The optional final step is that if an unknown face is seen repeatedly, the system can learn to recognize it.

1.3.2 3D Face Recognition

3D face recognition is expected to be robust to the types of issues that plague 2D systems. 3D systems generate 3D models of faces and compare them. These systems are more accurate because they capture the actual shape of faces. Skin texture analysis can be used in conjunction with face recognition to improve accuracy by 20 to 25 percent. The acquisition of 3D data is one of the main problems for 3D systems.

1.3.3 Applications of Face Recognition

- **Access Control:** Face verification, matching a face against a single enrolled exemplar, is well within the capabilities of current Personal Computer hardware. Since PC cameras have become widespread, their use for face-based PC logon has become feasible, though take-up seems to be very limited.
- **Identification Systems:** Two US States (Massachusetts and Connecticut) are testing face recognition for the policing of Welfare benefits. This is an identification task, where any new applicant being enrolled must be compared against the entire database of previously enrolled claimants, to ensure that they are not claiming under more than one identity.
- **Surveillance:** The application domain where most interest in face recognition is being shown is probably surveillance. Video is the medium of choice for surveillance because of the richness and type of information that it contains and naturally, for applications that require identification.
- **Pervasive Computing:** Another domain where face recognition is expected to become very important, although it is not yet commercially feasible, is in the area of pervasive or ubiquitous computing. Many people are envisaging the pervasive deployment of information devices.

1.4 Organization of the Thesis

The report has been organized into seven chapters. Chapter 1 gives an introduction to this thesis work, Problem statement, literature survey, Applications of Face Recognition and organization of the thesis. Chapter 2 discusses about image preprocessing techniques that can be used in face recognition. Chapter 3 contains techniques used for dimensionality reduction or feature generation. Chapter 4 describes about neural networks, training algorithms and training parameters. Chapter 5 gives details of the software developed using matlab for implementing the algorithm. Chapter 6 presents the Simulation results and discussions. Chapter 7 gives Conclusions and suggestions for future work.

Chapter 2: Image Preprocessing

The principal objective of image preprocessing used in face recognition is to highlight fine details in the facial image or to enhance detail that has been blurred, either in error or particular method of image acquisition. Mostly sharpening filters are used for this purpose. Various techniques can be used to implement this step. Technique used in proposed method and related techniques are discussed below.

2.1 Laplacian Operator

Laplacian is a second order derivative operator. This has been used in proposed method in the thesis for face recognition. Derivatives of digital functions are defined in terms of differences. Second order derivative shows different responses when moved over areas of constant grey level, onset and end of discontinuities and along grey level ramps. Any definition of second order derivative must satisfy (1) must be zero in constant areas (2) must be nonzero at the onset and end of an intensity ramp or step (3) must be zero along ramps of constant slope. Basic definition of second order derivative of $f(x)$ as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2 \times f(x) \quad (2.1)$$

This definition satisfies the conditions stated above. A desirable feature of sharpening filter is that it should be isotropic i.e. rotation invariant which means rotating the image then applying the filter gives the same results as applying filter to the image and then rotating the result. It can be shown that the simplest isotropic derivative operator is the Laplacian which for a function (image) $f(x, y)$ of two variables, is defined as,

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (2.2)$$

Which is sum of second order partial derivative in x and y direction respectively. In x direction we have equation which is same as 1, in y direction we have,

$$\frac{\partial^2 f}{\partial y^2} = f(y+1) + f(y-1) - 2 \times f(y) \quad (2.3)$$

From equation 1, 2 and 3 we can write,

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4 \times f(x, y) \quad (2.4)$$

This equation can be implemented using mask of figure 2.1 by convolving it with image which gives isotropic results for rotation increments of 90 degree [11].

0	1	0
1	-4	1
0	1	0

figure 2.1 Laplacian Mask

The diagonal directions can be incorporated in the definition in the definition of digital laplacian by adding two more terms to equation 4 one for each of two diagonal directions, modified equation is given by,

$$\begin{aligned} \nabla^2 f(x, y) = & f(x+1, y) + f(x, y+1) + f(x-1, y) + f(x, y-1) + f(x-1, y-1) + f(x+1, y-1) + f(x-1, y+1) \\ & + f(x+1, y+1) - 8 \times f(x, y) \end{aligned} \quad (2.5)$$

This equation can be implemented using filter of mask shown in figure 2.2. This mask yields isotropic results in increments of 45 degree. Laplacian is a derivative operator; its use highlights intensity discontinuities in an image and deemphasizes regions with slowly varying intensity levels. This produces images with grayish edge lines and other discontinuities, all superimposed upon dark featureless background.

1	1	1
1	-8	1
1	1	1

figure 2.2 Laplacian Mask Including Diagonal Directions

Background features can be recovered while still preserving the sharpening feature of the Laplacian simply by adding Laplacian image to the original using following equation,

$$g(x, y) = f(x, y) - \nabla^2 f(x, y) \quad (2.6)$$

Results of implementing equation 6 on facial images are shown in figure below. figure 2.3 shows the input image. figure 2.4 shows Laplacian image of the input image. figure 2.5 shows input image – Laplacian image. figure 2.5 clearly shows improvement in small details in output image after applying equation 6.



figure 2.3 input image



figure 2.4 Laplacian image



figure 2.5 input –laplacian

Large section of Laplacian image is dark because laplacian contain both positive and negative pixel values which are clipped to zero by the display. A simple way to scale a Laplacian image to add minimum value to each pixel so that new minimum is zero and the to scale pixel values to range $[0, L-1]$. Another way to solve this problem is to convert integer precision image into converted to double precision before applying Laplacian. A typical example showing ability of masks shown in figure 2.1 and 2.2 is shown in figure below. figure 2.6 shows blurred image of north pole of the moon. figure 2.7 shows image sharpened using mask of figure 2.1. figure 2.8 shows image sharpened by mask of figure 2.2.

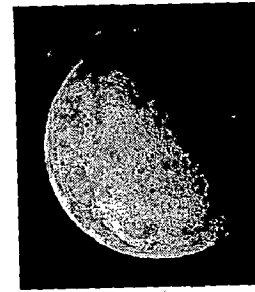
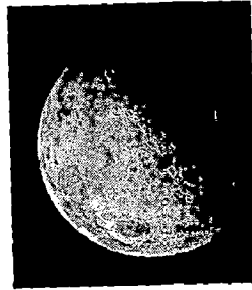
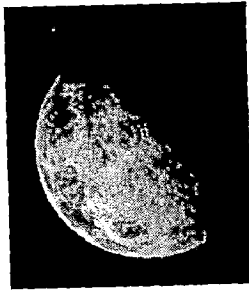


figure 2.6 North Pole of Moon

figure 2.7 Laplacian Filtered

figure 2.8 Filtered using Fig. 2.2 Mask

2.2 Unsharp Masking and Highboost Filtering

This process has been used for many years in printing and publishing industry to sharpen images consists of subtracting an Unsharp (smoothed) version of an image from the original image. This process is called Unsharp Masking. The can also be applied on faces to sharpen small details. This process consists following steps.

- Blur the original image
- Subtract the blurred image from the original (The resulting difference is called mask)
- Add the mask to the original image.

Letting $\bar{f}(x,y)$ denote the blurred image, unsharp masking is expressed as following equation,

$$g_{\text{mask}}(x,y) = f(x,y) - \bar{f}(x,y) \quad (2.7)$$

Then a weighted portion of mask is added back to the original image.

$$g(x,y) = f(x,y) + k \times g_{\text{mask}}(x,y) \quad (2.8)$$

Where k is a weight, when $k=1$ we have Unsharp Masking and when $k>1$ the process is called Highboost filtering. Choosing $k<1$ de-emphasizes the contribution of Unsharp mask [3].

2.3 Laplacian of Gaussian

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change. The Laplacian is applied to an image that has first been smoothed with a Gaussian smoothing filter in order to reduce its sensitivity to noise [2]. Additional advantage of Laplacian of Gaussian filter is that it can be tuned to act any desired scale, so that large operators can be used to detect blurry edges and small operators to detect sharply focused fine details. The derivation of mathematical equation for Laplacian of Gaussian function is given as:

Two dimensional Gaussian function is given as

$$G(x, y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (2.9)$$

Following derivatives are calculated to evaluate Laplacian of Gaussian

$$\nabla^2 G(x, y) = \frac{\partial^2 G(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y)}{\partial y^2} \quad (2.10)$$

$$\nabla^2 G(x, y) = \frac{\partial}{\partial x} \left[\frac{-x}{\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \right] + \frac{\partial}{\partial y} \left[\frac{-y}{\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \right] \quad (2.11)$$

$$\nabla^2 G(x, y) = \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right] e^{-\frac{(x^2+y^2)}{2\sigma^2}} + \left[\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2} \right] e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (2.12)$$

$$\nabla^2 G(x, y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right] e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (2.13)$$

This expression is called Laplacian of Gaussian. figure 2.9 shows 3-D plot of negative of LoG function. Because of its shape as shown in figure 2.9, it is also called Mexican hat operator. Masks of arbitrary size can be generated by sampling equation 2.13 scaling the coefficients such that they sum to zero. There are two fundamental ideas behind selection of operator $\nabla^2 G(x, y)$ for image sharpening purposes. Firstly, the Gaussian part of the operator blurs the image, thus reducing the intensity of structures at scales much smaller than σ . Secondly second order derivative is isotropic, which not only corresponds to human visual system but also responds equally to changes in intensity in any mask direction, thus avoiding having to use multiple masks to calculate strongest response at any point in the image. figure 2.10 and 2.11 shows application LoG filter with $\sigma=0.5$ on facial image. figure 2.12 shows the mask used.

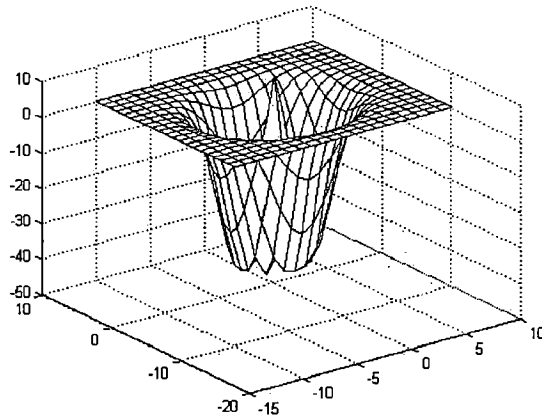


figure 2.9 3-D Plot of LoG Filter $\sigma = 0.5$



figure 2.10 Input image

figure 2.11 LoG filtered image

0.0448	0.0468	0.0564	0.0468	0.0448
0.0468	0.3167	0.7146	0.3167	0.0468
0.0564	0.7146	-4.9048	0.7146	0.0564
0.0468	0.3167	0.7146	0.3167	0.0468
0.0448	0.0468	0.0564	0.0468	0.0448

figure 2.12 Mask used for Filtering

Chapter 3: Dimensionality Reduction or Feature Generation

Feature generation is of paramount importance in any pattern recognition task. Given a set of measurements, the goal is to discover compact and informative representations of the obtained data. A similar process is also taking place in the human perception apparatus. Our mental representation of the world is based on a relatively small number of perceptually relevant features. These are generated after processing a large amount of sensory data, such as the intensity and the color of the pixels of the images sensed by our eyes, and the power spectra of the sound signals sensed by our ears[9].

The basic approach followed is to transform a given set of measurements to a new set of features. If the transform is suitably chosen, transform domain features can exhibit high information packing properties compared with the original input samples. This means that most of the classification-related information is “squeezed” in a relatively small number of features, leading to a reduction of the necessary feature space dimension. We refer to such processing tasks as dimensionality reduction techniques.

Let us take for example an image resulting from a measuring device, for example, X-rays or a camera. The pixels (i.e., the input samples) at the various positions in the image have a large degree of correlation, due to the internal morphological consistencies of real-world images that distinguish them from noise. Thus, if one uses the pixels as features, there will be a large degree of redundant information. Alternatively, if one obtains the Fourier transform, for example, of a typical real-world image, it turns out that most of the energy lies in the low-frequency components, due to the high correlation between the pixels’ gray levels. Hence, using the Fourier coefficients as features seems a reasonable choice, because the low-energy, high-frequency coefficients can be neglected, with little loss of information. Fourier transform is just one of the tools from a palette of possible transforms. Some of the ways to implement dimensionality reduction are

1. Linear Discriminant Analysis
2. Principal Component Analysis
3. Singular Value Decomposition
4. Wavelet Packet Decomposition
5. Independent Component Analysis etc.

3.1 Basis Vectors and Images

Let $x(0), x(1), x(2) \dots, x(N-1)$ be a set of input samples and \mathbf{x} be the $N \times 1$ corresponding vector.

$$\mathbf{x}^T = [x(0), x(1), \dots, x(N-1)] \quad (3.1)$$

Given a unitary $N \times N$ matrix A , we define the transformed vector \mathbf{y} of \mathbf{x} as,

$$\mathbf{y} = A^H \mathbf{x} = \begin{pmatrix} \mathbf{a}_0^H \\ \vdots \\ \mathbf{a}_{N-1}^H \end{pmatrix} \mathbf{x} \quad (3.2)$$

where H denotes the Hermitian operation, that is, complex conjugation and transposition. From equation 3.2 and the definition of unitary matrices we have

$$\mathbf{x} = A \mathbf{y} = \sum_{i=0}^{N-1} y(i) \mathbf{a}_i \quad (3.3)$$

The columns of A , \mathbf{a}_i , $i=0, 1, \dots, N-1$, are called the basis vectors of the transform. The elements $y(i)$ of \mathbf{y} are nothing but the projections of \mathbf{x} onto these basis vectors. Indeed, taking the inner product of \mathbf{x} with \mathbf{a}_j , we have

$$\langle \mathbf{a}_j, \mathbf{x} \rangle = \mathbf{a}_j^H \mathbf{x} = \sum_{i=0}^{N-1} y(i) \langle \mathbf{a}_j, \mathbf{a}_i \rangle = \sum_{i=0}^{N-1} y(i) \delta_{ij} = y(j) \quad (3.4)$$

This is due to the unitary property of A , that is, $A^H A = I$ or $\langle \mathbf{a}_i, \mathbf{a}_j \rangle = \mathbf{a}_i^H \mathbf{a}_j = \delta_{ij}$. In many problems, such as in image analysis, the input set of samples is a two dimensional sequence $X(i, j)$ where $i, j = 0, 1, 2, \dots, N-1$, defining an $N \times N$ matrix X instead of a vector. In such cases, one can define an equivalent N^2 vector \mathbf{x} , for example, by ordering the rows of the matrix one after the other (lexicographic ordering) and then transform this equivalent vector.

$$\mathbf{x}^T = [X(0,0), \dots, X(0, N-1), \dots, X(N-1,0), \dots, X(N-1, N-1)] \quad (3.5)$$

The number of operations required to multiply a $N^2 \times N^2$ square matrix A with a $N^2 \times 1$ vector \mathbf{x} is of the order of $O(N^4)$, which is prohibitive for many applications. An alternative possibility is to

transform matrix X via a set of basis matrices or basis images. Let U and V be unitary $N \times N$ matrices. Define the transformed matrix Y of X as

$$Y = U^T X V \quad (3.6)$$

Or
$$X = U Y V^T \quad (3.7)$$

The number of operations is now reduced to $O(N^3)$. Equation (3.7) can alternatively be written as,

$$X = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} Y(i, j) u_i v_j^H \quad (3.8)$$

Where u_i are the column vectors of U and v_j the column vectors of V . Each of the outer products $u_i v_j^H$ is an $N \times N$ matrix,

$$u_i v_j^H = \begin{pmatrix} u_{i0} v_{j0}^* & \cdots & u_{i0} v_{jN-1}^* \\ \vdots & \ddots & \vdots \\ u_{iN-1} v_{j0}^* & \cdots & u_{iN-1} v_{jN-1}^* \end{pmatrix} = A_{ij}$$

And (3.8) is an expansion of matrix X in terms of these N^2 basis images (matrices). The $*$ denotes complex conjugation. Furthermore, if Y turns out to be diagonal, then (3.8) becomes

$$X = \sum_{i=0}^{N-1} Y(i, i) u_i v_i^H \quad (3.9)$$

and the number of basis images is reduced to N . Inner product between two matrices is defined as

$$\langle A, B \rangle = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} A^*(m, n) B(m, n) \quad (3.10)$$

It can be shown that

$$Y(i, j) = \langle A_{ij}, X \rangle \quad (3.11)$$

In words, the (i, j) element of the transformed matrix results from multiplying each element of X by the conjugate of the corresponding element of A_{ij} and summing up all products[9].

3.2 Positive Definite and Symmetric Matrices

An $l \times l$ real matrix A is called positive definite if for every nonzero vector x the following is true:

$$x^T A x > 0 \quad (3.12)$$

If equality with zero is allowed, A is called nonnegative or positive semi-definite. It is easy to show that all eigen values of such a matrix are positive. Indeed, let λ_i be one eigen value and v_i the corresponding unit norm eigenvector ($v_i^T v_i = 1$). Then by the respective definitions

$$A v_i = \lambda_i v_i \quad (3.13)$$

$$v_i^T A v_i = \lambda_i > 0 \quad (3.14)$$

Since the determinant of a matrix is equal to the product of its eigenvalues, we conclude that the determinant of a positive definite matrix is also positive.

Let A be an $l \times l$ symmetric matrix, $A^T = A$. Then the eigenvectors corresponding to distinct eigenvalues are orthogonal. Indeed, let $\lambda_i \neq \lambda_j$ be two such eigenvalues. From the definitions we have,

$$A v_i = \lambda_i v_i \quad (3.15)$$

$$A v_j = \lambda_j v_j \quad (3.16)$$

Multiplying (3.9) on the left by v_j^T and the transpose of (3.10) on the right by v_i , we obtain

$$v_j^T A v_i - v_j^T A v_i = 0 = (\lambda_i - \lambda_j) v_j^T v_i \quad (3.17)$$

Thus, $v_j^T v_i = 0$. Furthermore, it can be shown that even if the eigenvalues are not distinct, we can still find a set of orthogonal eigenvectors. The same is true for Hermitian matrices, in case we deal with more general complex-valued matrices.

Based on this, it is now straightforward to show that a symmetric matrix A can be diagonalized by the similarity transformation

$$\phi^T A \phi = \Lambda \quad (3.18)$$

Where matrix ϕ has as its columns the unit eigenvectors ($v_i^T v_i = 1$) of A , that is,

$$\phi = [v_1 \ v_2 \ \cdots \ v_L] \quad (3.19)$$

and Λ is the diagonal matrix with elements the corresponding eigenvalues of A . From the orthonormality of the eigenvectors it is obvious that $\phi^T \phi = I$, that is, ϕ is a unitary matrix, $\phi^{-1} = \phi^T$. The proof is similar for Hermitian complex matrices as well [9].

3.3 Covariance and Correlation

As the name implies, covariance is a measure of the strength of the link between two (numerical) random variables. Given two such random variable. X_1 and X_2 , two extreme circumstances can be encountered :

- There is no link whatsoever between X_1 and X_2 . Knowing the value of X_1 gives no clue to what the value of X_2 might be. The two variables are said to be independent.
- The link is so strong that it is in fact functional. There is a completely deterministic function $y=f(x)$ such that knowing the value of X_1 then determines the value X_2 of without any uncertainty.

$$X_2 = f(X_1)$$

Most often, the link between two random variables somewhere in between knows the value taken by X_1 reduces, to a certain extent, the uncertainty about the value that X_2 will take. There is no universal way to define and measure the strength of the link in this intermediary situation. Covariance is one way to do it, and is very useful in many practical situations despite its limitations.

3.3.1 Definition of the Covariance

If X_1 and X_2 are strongly (positively) linked, then we could think of defining covariance in a way that would embody the following idea:

- Whenever $(X_1 - \mu_1)$ is positive, then $(X_2 - \mu_2)$ is likely to be positive too.
- Whenever $(X_1 - \mu_1)$ is negative, then $(X_2 - \mu_2)$ is likely to be negative too.

The product $(X_1 - \mu_1).(X_2 - \mu_2)$ is then likely to be very often positive when either because both quantities are positive, Or because both quantities are negative.

Yet, the product $(X_1 - \mu_1) \cdot (X_2 - \mu_2)$ is a random variable, and we want a fixed number. But a random variable that spends most of its time taking positive values is likely to have a positive expectation. So we will consider the expectation of $(X_1 - \mu_1) \cdot (X_2 - \mu_2)$, and call it the covariance of X_1 and X_2 .

$$\text{cov}(X_1, X_2) = E((X_1 - \mu_1)(X_2 - \mu_2)) \quad (3.20)$$

Following expression is equivalent to this other one and more convenient in practice,

$$\text{cov}(X_1, X_2) = E[X_1 X_2] - E[X_1]E[X_2] \quad (3.21)$$

3.3.2 Interpretation of the Covariance

- A large positive value of the Covariance is an indication that $(X_1 - \mu_1)$ and $(X_2 - \mu_2)$ often take large positive or large negative values simultaneously, a circumstance that strengthens our belief that the variables are indeed tightly linked.
- Where as a smaller positive value of the covariance is an indication that one of the variables has a fair chance to be close to its mean when the other takes large (positive or negative) values.
- The Covariance may be low because, indeed, the link between the two variables is weak.
- But there may exist a strong, non linear link between the two variables, the nature of this link making the Covariance low.

The argument developed leading to the definition of the covariance on the basis of a positive link between X_1 and X_2 . But it applies just as well in the case of a negative link. We can use the same line of reasoning if $X_1 - \mu_1$ taking large positive values makes it likely that $X_2 - \mu_2$ will take large negative values. In this case, the covariance is a large negative number. A drawback of covariance is that its value depends on the units used to express the values of X_1 and X_2 , whereas a practical measure of the strength of the link between two variables certainly shouldn't.

If the two random variables X and Y are independent then their covariance is 0. But the converse is not true. Two random variables may have 0 covariance, and yet not be independent. For example, let :

- X be uniformly distributed in $[-1, +1]$.
- $Y = X^2$

In this case, $\text{cov}(X, Y) = 0$.

3.3.3 Correlation Coefficient

One problem with covariance is that it is sensitive to the scales on which the values of the random variable are measured. While computing the covariance between “Height” and “Weight” in a population, Measuring weights in “Kilos” instead of “Pounds”, or heights in “Centimeters” instead of “Inches”, and the value of the covariance changes, whereas the strength of the link between “Height” and “Weight” remains the same, of course. So we would like a measure of the strength of the link between “Height” and “Weight” that does not depend on the units used to measure these quantities.

Now suppose that the unit measuring X_1 is divided by 2 (so that values of X_1 are multiplied by 2). Then the covariance $\text{Cov}(X_1, X_2)$ is also multiplied by 2. But the standard deviation of X_1 (square root of the variance) is also multiplied by 2, so the ratio

$$\frac{\text{Cov}(X_1, X_2)}{(\text{Var}(X_1))^{\frac{1}{2}}}$$

is unchanged. The same argument applies to X_2 , and, more generally, to any change of units in which X_1 or X_2 are measured. So, quite generally, the number:

$$\rho = \frac{\text{Cov}(X_1, X_2)}{\sqrt{\text{Var}(X_1)\text{Var}(X_2)}} \quad (3.23)$$

is called correlation coefficient.

3.3.4 Properties of the Correlation Coefficient

- The value of the correlation coefficient is always between -1 and +1.
- If $X_1 = X_2$ then $\text{Cov}(X_1, X_2) = \text{Var}(X_1) = \text{Var}(X_2)$. Therefore, $\rho(X, X) = +1$.
- The Correlation Coefficient is symmetrical: $\rho(X_1, X_2) = \rho(X_2, X_1)$.
- If both variables have unit variances, then their Covariance is the same as their Correlation Coefficient.
- When the two distributions are known only through a sample, the common estimate of the Correlation Coefficient is :

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 (y - \bar{y})^2}} \quad (3.24)$$

3.3.5 Mean Vector and Covariance Matrix

The first step in analyzing multivariate data is computing the mean vector and the variance-covariance matrix. Consider the following matrix:

$$X = \begin{bmatrix} 4.0 & 2.0 & .60 \\ 4.2 & 2.1 & .59 \\ 3.9 & 2.0 & .58 \\ 4.3 & 2.1 & .62 \\ 4.1 & 2.2 & .63 \end{bmatrix}$$

The set of 5 observations, measuring 3 variables, can be described by its mean vector and variance-covariance matrix. The three variables, from left to right are length, width, and height of a certain object, for example. Each row vector X_i is another observation of the three variables (or components). The mean vector consists of the mean of each variable and the variance-covariance matrix consists of the variance of the variables along the main diagonal and the covariance between each pair of variables in the other matrix positions. The Formula for evaluation of covariance matrix is given by,

$$\text{Cov} = \frac{1}{n-1} XX^T \quad (3.25)$$

Where n is number of observations

The results are,

$$\text{Mean} = [4.10 \quad 2.08 \quad .604]$$

$$\text{Covariance} = \begin{bmatrix} 0.025 & 0.0075 & 0.00175 \\ 0.0075 & 0.0070 & 0.00135 \\ 0.00175 & 0.00135 & 0.00043 \end{bmatrix}$$

Thus, 0.025 is the variance of the length variable, 0.0075 is the covariance between the length and the width variables, 0.00175 is the covariance between the length and the height variables, 0.007 is the variance of the width variable, 0.00135 is the covariance between the width and height variables and .00043 is the variance of the height variable. The mean vector is often referred to as the centroid and the variance-covariance matrix as the dispersion or dispersion matrix. Also, the terms variance-covariance matrix and covariance matrix are used interchangeably.

3.3.6 Correlation Matrix Diagonalization

Let x be a random vector in the l -dimensional space. Its correlation matrix is defined as $R = E[xx^T]$. Matrix R is readily seen to be positive semidefinite. For our purposes we will assume that it is positive definite, thus invertible. Moreover, it is symmetric, and hence it can always be diagonalized.

$$\phi^T R \phi = \Lambda \quad (3.26)$$

Where ϕ is the matrix consisting of the (orthogonal) eigenvectors and Λ the diagonal matrix with the corresponding eigenvalues on its diagonal. Thus, we can always transform x into another random vector whose elements are uncorrelated. Indeed

$$x_1 = \phi^T x \quad (3.27)$$

Then the new correlation matrix is $R_1 = \phi^T R \phi$. Furthermore, if $\Lambda^{\frac{1}{2}}$ is the diagonal matrix whose elements are the square roots of the eigenvalues of R ($\Lambda^{\frac{1}{2}} \Lambda^{\frac{1}{2}} = \Lambda$), then it is readily shown that the transformed random vector

$$x_1 = \Lambda^{-\frac{1}{2}} \phi^T x \quad (3.28)$$

has uncorrelated elements with unit variance. $\Lambda^{-\frac{1}{2}}$ denotes the inverse of $\Lambda^{\frac{1}{2}}$. That is, the transformed variables are also uncorrelated with unit variance [9].

3.4 Principal Component Analysis

Principal component analysis (PCA) is one of the most popular techniques for dimensionality reduction. Starting from an original set of l samples (features), which form the elements of a vector $x \in \mathbb{R}^l$, the goal is to apply a linear transformation to obtain a new set of samples. Let x be the vector of input samples. In the case of an image array, x may be formed by lexicographic ordering of the array elements. In order to simplify the presentation, we will assume that the data samples have zero mean. If this is not the case, we can always subtract the mean value. We have already mentioned that a desirable property of the generated features is to be mutually uncorrelated in an effort to avoid information redundancies. We begin, by first developing a method that generates mutually uncorrelated features, that is, $(E[y(i)y(j)] = 0), i \neq j$.

Let

$$y = A^T x \quad (3.29)$$

Since we have assumed that $E[x]=0$, it is readily seen that $E[y]=0$. From the definition of the correlation matrix we have

$$R_y = E[yy^T] = E[A^T x x^T A] = A^T R_x A \quad (3.30)$$

In practice, R_x is estimated as an average over the given set of training vectors. For example, if we are given n data vectors $x_k, k = 1, 2, \dots, n$, then

$$R_x = \frac{1}{n-1} \sum_{k=1}^n x_k x_k^T \quad (3.31)$$

Note that R_x is a symmetric matrix, and hence its eigenvectors are mutually orthogonal from (3.17). Thus, if matrix A is chosen so that its columns are the orthonormal eigenvectors $a_i, i=0, 1, \dots, N-1$, of R_x , then R_y is diagonal.

$$R_y = A^T R_x A = \Lambda \quad (3.32)$$

Where Λ is the diagonal matrix having as elements on its diagonal the respective eigenvalues $\lambda_i, i=0, 1, \dots, N-1$, of R_x . Furthermore, assuming R_x to be positive definite the eigenvalues are

positive from (3.14). The resulting transform is known as the Karhunen–Loève (KL) transform, and it achieves our original goal of generating mutually uncorrelated features. It has to be emphasized that the solution provided by the KL transform is not a unique one, and it was obtained by imposing an orthogonal structure on matrix A ($A^T A = I$). Also, note that for zero mean variables the correlation matrix R coincides with the covariance matrix Σ . As a matter of fact, a direct consequence of the respective definitions is that

$$\Sigma_x = R_x - E[x]E[x]^T \quad (3.33)$$

In case the zero mean assumption is not valid, the condition for uncorrelated variables becomes $E[(y(i) - E[y(i)])(y(j) - E[y(j)])] = 0, i \neq j$, and the problem results in the eigen decomposition of the covariance matrix, that is,

$$\Sigma_y = A^T \Sigma_x A = \Lambda \quad (3.34)$$

Although our starting point was to generate mutually uncorrelated features, the KL transform turns out to have a number of other important properties, which provide different ways for its interpretation and also the secret for its popularity[9].

3.4.1 Mean Square Error Approximation

From Equations (3.3) and (3.4) we have

$$x = \sum_{i=0}^{N-1} y(i)a_i \quad (3.35)$$

$$y(i) = a_i^T x \quad (3.36)$$

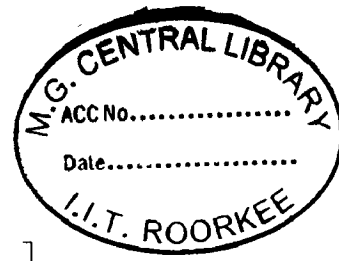
Let us now define a new vector in the m -dimensional subspace

$$\hat{x} = \sum_{i=0}^{m-1} y(i)a_i \quad (3.37)$$

where only m of the basis vectors are involved. Obviously, this is nothing but the projection of x onto the subspace spanned by the m (orthonormal) eigenvectors involved in the summation. If we try to approximate x by its projection \hat{x} , the resulting mean square error is given by

$$E \left[\left\| x - \hat{x} \right\|^2 \right] = E \left[\left\| \sum_{i=m}^{N-1} y(i)a_i \right\|^2 \right] \quad (3.38)$$

Our goal now is to choose the eigenvectors that result in the minimum MSE. From (3.38) and taking into account the orthonormality property of the eigenvectors, we have



$$E \left\| \sum_{i=m}^{N-1} y(i) a_i \right\|^2 = E \left[\sum_i \sum_j (y(i) a_i^T)(y(j) a_j) \right] \quad (3.39)$$

$$= \sum_{i=m}^{N-1} E [y(i)^2] = \sum_{i=m}^{N-1} a_i^T E [x x^T] a_i \quad (3.40)$$

Combining this with (3.38) and the eigenvector definition, we finally get

$$E \left\| x - \hat{x} \right\|^2 = \sum_{i=m}^{N-1} a_i^T \lambda_i a_i = \sum_{i=m}^{N-1} \lambda_i \quad (3.41)$$

Thus, if we choose in (3.37) the eigenvectors corresponding to the m largest eigenvalues of the correlation matrix, then the error in (3.41) is minimized, being the sum of the $N - m$ smallest eigenvalues. Furthermore, it can be shown that this is also the minimum MSE, compared with any other approximation of x by an m -dimensional vector. This is the reason that the KL transform is also known as principal component analysis (PCA). A difficulty in practice is how to choose the m principal components. One way is to rank the eigenvalues in descending order, $\lambda_0 \geq \lambda_1 \geq \dots \geq \lambda_{m-1} \geq \lambda_m$, and determine m so that the gap between the values λ_{m-1} and λ_m is large [9].

3.4.2 Steps for Calculation of Principal Component of given Data

- Estimate the covariance matrix S . Usually the mean value is assumed to be zero, $E[x] = 0$. In this case, the covariance and autocorrelation matrices coincide, $R \equiv E[x x^T] = S$. If this is not the case, we subtract the mean. Recall that, given N feature vectors, $x_i \in R^l$, $i = 1, 2, \dots, N$, the autocorrelation matrix estimate is given by,

$$R \approx \frac{1}{N-1} \sum_{i=1}^N x_i x_i^T \quad (3.42)$$

- Perform the eigen decomposition of S and compute the l eigenvalues/eigenvectors, λ_i , $a_i \in R^l$, $i = 0, 1, 2, \dots, l-1$.
- Arrange the eigenvalues in descending order, $\lambda_0 \geq \lambda_1 \geq \dots \geq \lambda_{l-1} \geq \lambda_l$.
- Choose the m largest eigenvalues. Usually m is chosen so that the gap between λ_{m-1} and λ_m is large. Eigenvalues $\lambda_0, \lambda_1, \dots, \lambda_{m-1}, \lambda_m$ are known as the m principal components.
- Use the respective (column) eigenvectors a_i , $i = 0, 1, 2, \dots, m-1$ to form the transformation matrix.

- Transform each l-dimensional vector x in the original space to an m-dimensional vector y via the transformation $y = A^T x$. In other words, the i^{th} element $y(i)$ of y is the projection of x on a_i ($y(i) = a_i^T x$) [12].

3.4.3 Example Problem

Let us take an arbitrary data matrix,

$$A = \begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \\ 2 & 14 & 7 & 23 \end{bmatrix}$$

A has 5 rows and 4 columns i.e. 4 observations with 5 dimensions each.

Mean vector, which is column vector is calculated by taking mean of each dimensions all observations, is

$$\begin{aligned} & 8.5 \\ & 8.5 \\ mn = & 8.5 \\ & 8.5 \\ & 28.5 \end{aligned}$$

Data matrix obtained after mean vector is subtracted from each column,

$$\text{data} = \begin{bmatrix} 7.5 & -5.5 & -6.5 & -4.5 \\ -3.5 & -1.5 & 2.5 & -.5 \\ .5 & -2.5 & -1.5 & 3.5 \\ -4.5 & 6.5 & 5.5 & -7.5 \\ -26.5 & -14.5 & -21.5 & -5.5 \end{bmatrix}$$

Covariance matrix of this data matrix is calculated using (3.42).

$$\text{Covariance} = \begin{bmatrix} 49.6667 & -17.6667 & 14.3333 & -46.3333 & -1.3333 \\ -17.6667 & 7.0000 & -3.6667 & 14.3333 & 6.6667 \\ 14.3333 & -3.6667 & 7.0000 & -17.6667 & 12.0000 \\ -46.3333 & 14.3333 & -17.6667 & 49.6667 & -17.3333 \\ -1.3333 & 6.6667 & 12.0000 & -17.3333 & 468.3333 \end{bmatrix}$$

Eigen values and eigen vectors are calculated. Eigen vector are also called basis vectors or principal components

Eigen values in ascending order =

-469.4782
-106.2776
-5.9108
0
0

Eigen vectors =

0.0019	0.6735	-0.4964	0.0776	-0.5477
0.0128	-0.2265	0.4985	0.5075	-0.7303
0.0275	0.2204	0.5007	-0.7824	-0.1826
-0.0422	-0.6674	-0.5028	-0.3524	-0.3651
0.9986	-0.0326	-0.0405	0.0000	0.0000

Now, reduced image is obtained by applying transformation $y = A^T x$ where A is matrix containing Eigen vectors as columns, x is data matrix. Original data can be recovered by reverse transformation i.e. $x = Ay$ and adding the mean vector to the result.

Reduced image=

-26.2914	-14.8143	-21.7243	-5.0781
9.8217	-8.4605	-8.2444	9.1003
-1.8826	-0.4554	1.8262	3.2629
0	0	0	0
0	0	0	0

Clearly, it can be observed that final out matrix of the algorithm has lesser dimensions than original data matrix, which can completely recovered from reduced image.

3.5 Singular Value Decomposition

The singular value decomposition of a matrix is one of the most elegant and powerful algorithms in linear algebra and it have been extensively used for rank and dimension reduction in pattern recognition and information retrieval applications. Given a $l \times n$ matrix X of rank r (obviously $r < \min \{l, n\}$), It can be shown that there exist unitary matrices U and V of dimensions $l \times l$ and $n \times n$, respectively, so that

$$X = U \begin{pmatrix} \frac{1}{\Lambda^2} & 0 \\ 0 & 0 \end{pmatrix} V^H$$

Or,

$$Y = \begin{pmatrix} \Lambda^{\frac{1}{2}} & \mathbf{O} \\ \mathbf{O} & 0 \end{pmatrix} = U^H X V \quad (3.43)$$

Where $\Lambda^{\frac{1}{2}}$ is the $r \times r$ diagonal matrix with elements $\sqrt{\lambda_i}$, and λ_i are the r nonzero eigenvalues of the associated matrix $X^H X$. \mathbf{O} denotes a zero element matrix. In other words, there exist unitary matrices U and V that transform X into the special diagonal structure of Y . If u_i , v_i denote the column vectors of matrices U and V , respectively, then (3.43) can be written as

$$X = \begin{bmatrix} u_0 & u_1 & \dots & u_{r-1} \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_0} & & & \\ & \sqrt{\lambda_1} & & \\ & & \ddots & \\ & & & \sqrt{\lambda_{r-1}} \end{bmatrix} \begin{bmatrix} v_0^H \\ v_1^H \\ \vdots \\ v_{r-1}^H \end{bmatrix} \quad (3.44)$$

$$X = \sum_{i=0}^{r-1} \sqrt{\lambda_i} u_i v_i^H \quad (3.45)$$

where U_r denotes the $l \times r$ matrix that consists of the first r columns of U and V_r the $r \times n$ matrix formed by using the first r columns of V . More precisely, u_i , v_j are the eigenvectors corresponding to the nonzero eigenvalues of the matrices XX^H and $X^H X$, respectively. The eigenvalues λ_i are known as singular values of X and the expansion in (3.45) as the singular value decomposition (SVD) of X or the spectral representation of X [9].

3.5.1 Proof of Singular Value Decomposition

Given a matrix X of rank r , it is known from linear algebra that the $n \times n$ matrix $X^H X$ as well as the $l \times l$ matrix XX^H is of the same rank r . Furthermore, both matrices have the same nonzero eigenvalues but different (yet related) eigenvectors.

$$XX^H u_i = \lambda_i u_i \quad (3.45)$$

$$X^H X v_i = \lambda_i v_i \quad (3.46)$$

Since both matrices are Hermitian and nonnegative (i.e., $(XX^H)^H = XX^H$), they have nonnegative real eigenvalues and orthogonal eigenvectors. The eigenvectors, given in (3.45) and (3.46), can

also be normalized to become orthonormal, that is, $u_i^H u_i = 1$ and $v_i^H v_i = 1$. Pre-multiplying (3.45) by X results in,

$$(XX^H)Xv_i = \lambda_i Xv_i \quad (3.47)$$

That is, $u_i = \alpha Xv_i$, where the scaling factor can be taken as positive and it is found from,

$$\|u_i\|^2 = 1 = \alpha^2 v_i^H X^H X v_i = \alpha^2 \lambda_i \|v_i\|^2 \Rightarrow \alpha = \frac{1}{\sqrt{\lambda_i}} \quad (3.48)$$

So,
$$u_i = \frac{1}{\sqrt{\lambda_i}} Xv_i \quad (3.49)$$

Let us now assume that $u_i, v_i, i=0, 1, \dots, r-1$, are the eigenvectors corresponding to the nonzero eigenvalues and $u_i, i=r, \dots, l-1, v_i, i=r, \dots, n-1$, to the zero ones. Then, for the latter case we have,

$$X^H X v_i = 0 \Rightarrow v_i^H X^H X v_i = 0 \Rightarrow \|Xv_i\|^2 = 0$$

Hence,

$$Xv_i = 0 \quad i=r, \dots, n-1 \quad (3.50)$$

In a similar way one can show that

$$X^H u_i = 0 \quad i=r, \dots, n-1 \quad (3.51)$$

Combining (3.49) and (3.50), it can be shown that the right-hand side of (3.45) is

$$\sum_{i=0}^{r-1} \sqrt{\lambda_i} u_i v_i^H = X \sum_{i=0}^{r-1} \sqrt{\lambda_i} \frac{1}{\sqrt{\lambda_i}} v_i v_i^H = X \sum_{i=0}^{r-1} v_i v_i^H \quad (3.52)$$

Let us now define a matrix V that has as columns the orthonormal eigenvectors v_i ,

$$V = [v_0 \quad \dots \quad v_{n-1}]$$

Orthonormality of the columns results in $V^H V = I$ that is, V is unitary and hence $VV^H = I$. Thus, it turns out that,

$$I = VV^H = [v_0 \quad \dots \quad v_{n-1}] \begin{bmatrix} v_0^H \\ \vdots \\ v_{n-1}^H \end{bmatrix} = \sum_{i=0}^{n-1} v_i v_i^H \quad (3.53)$$

From (3.52) and (3.53) we obtain

$$X = \sum_{i=0}^{r-1} \sqrt{\lambda_i} u_i v_i^H \quad (3.54)$$

And X can be written as,

$$X = U \begin{pmatrix} \frac{1}{\Lambda^2} & 0 \\ 0 & 0 \end{pmatrix} V^H \quad (3.55)$$

Where U is the unitary matrix with columns the orthonormal eigen vectors u_i .

3.5.2 Low Rank Approximation

The expansion in (3.54) is an exact representation of matrix X. A very interesting implication occurs if one uses less than r (the rank of X) terms in the summation. Let X be approximated by

$$X \approx \hat{X} = \sum_{i=0}^{k-1} \sqrt{\lambda_i} u_i v_i^H, k \leq r \quad (3.56)$$

Matrix \hat{X} , being the sum of $k \leq r$ rank-one independent $1 \times n$ matrices, is of rank k. If the k largest eigenvalues are involved, it can be shown that the squared error,

$$\epsilon^2 = \sum_{i=0}^{l-1} \sum_{j=0}^{n-1} |X(i, j) - \hat{X}(i, j)|^2 \quad (3.57)$$

is the minimum one with respect to all rank-k $1 \times n$ matrices. The square root of the right-hand side in (3.57) is also known as the Frobenius norm $\|X - \hat{X}\|_F$ of the difference matrix $X - \hat{X}$. The error in the approximation turns out to be

$$\epsilon^2 = \sum_{i=k}^{r-1} \lambda_i \quad (3.58)$$

Hence, if we order the eigenvalues in descending order, $\lambda_0 \geq \lambda_1 \geq \dots \geq \lambda_{r-1}$, then for a given number of k terms in the expansion, the SVD leads to the minimum square error. Thus, \hat{X} is the best rank k approximation of X in the Frobenius norm sense. This reminds us of the Karhunen–Loève expansion. However, in the latter case the optimality was with respect to the mean square error. This is a major difference in philosophy between SVD and KL. The former is related to a single set of samples and the latter to an ensemble of them [9].

3.5.3 Dimensionality Reduction

SVD has been used extensively for dimension reduction in pattern recognition and information retrieval, and it forms the basis of what is known as latent semantics indexing, see, for example, Adopting the notation used in (3.44) and (3.45), Eq. (3.56) can be written as[2],

$$\begin{aligned}
X \simeq \hat{X} &= [u_0 \quad u_1 \quad \dots \quad u_{k-1}] \begin{bmatrix} \sqrt{\lambda_0} v_0^H \\ \sqrt{\lambda_1} v_1^H \\ \vdots \\ \sqrt{\lambda_{k-1}} v_{k-1}^H \end{bmatrix} \\
&= U_k [a_0 \quad a_1 \quad \dots \quad a_{n-1}]
\end{aligned} \tag{3.59}$$

where U_k consists of the first k columns of U and the k dimensional vectors a_i , $i = 0, 1, \dots, n-1$, are the column vectors of the $k \times n$ product matrix $\Lambda_k^{\frac{1}{2}} V_k^H$, where V_k^H consists of the first k rows of V^H and $\Lambda_k^{\frac{1}{2}}$ is the diagonal matrix having elements the square roots of the respective k singular values. The formulation given in (3.59) suggests that each column vector, x_i of X , is approximated as,

$$x_i \simeq U_k a_i = \sum_{m=0}^{k-1} u_m a_i(m) \quad i=0, \dots, n-1 \tag{3.60}$$

where $a_i(m)$, $m=0, 1, \dots, k-1$, denote the elements of the respective vector a_i . In words, the l -dimensional vector x_i is approximated by the k -dimensional vector a_i , lying in the subspace spanned by u_i , $i = 0, 1, \dots, k-1$ (a_i is the projection of x_i on this subspace) Furthermore, due to the orthonormality of the columns u_i , $i = 0, 1, \dots, k-1$, of U_k from (3.60) it is straightforward to see that,

$$\|x_i - x_j\|^2 \simeq \|U_k(a_i - a_j)\|^2 \tag{3.61}$$

$$= (U_k(a_i - a_j))^T (U_k(a_i - a_j)) \tag{3.62}$$

$$= (a_i - a_j)^T U_k^T U_k (a_i - a_j) \tag{3.63}$$

$$= \|a_i - a_j\|^2 \quad i, j = 0, 1, 2, \dots, n-1 \tag{3.64}$$

Where $\|\cdot\|$ represents the Euclidean norm of a vector. That is, using the previous projection and assuming the approximation to be reasonably good, the Euclidean distance between x_i and x_j in the high l -dimensional space is (approximately) preserved under the projection in the lower k -dimensional subspace. The previous observation has important implications in applications such as information retrieval. For example, the simple case where we are given a set of n patterns each represented by a l -dimensional feature vector. These patterns constitute the available database. Given an unknown pattern, the goal is to search for and recover from the database the pattern that

is most similar to the unknown one, by computing its Euclidean distance from each vector in the database. When l and n are large numbers this can be a very time-consuming task. A procedure to simplify computations is the following. We form the $l \times n$ data matrix, X having as columns the n feature vectors. Perform a SVD on X and represent each feature vector, x_i , by its lower dimensional projection, a_i , as described before. Given the unknown vector, one projects it on the subspace spanned by the columns of U_k and performs Euclidean distance computations in the k -dimensional space. Since Euclidean distances are approximately preserved, one can decide about the proximity of vectors by working in a lower dimensional space. If $k \ll l$ substantial computational savings are obtained. SVD builds upon global information spread over all the data vectors in X . Indeed, a crucial part of the algorithm is the computation of the eigenvalues of $X^H X$ or XX^H , which, for zero mean data, is directly related to the respective covariance matrix. Hence, the performance of the SVD, as a dimensionality reduction technique, is most effective for cases where data can sufficiently be described in terms of the covariance matrix, for example, to be Gaussian-like distributed. In a modification of the simple SVD is suggested to account for data with a clustered structure.

Due to its optimal approximation properties, the SVD transform also has excellent information packing properties, and an image array can be represented efficiently by a few of its singular values. Thus, SVD is a natural candidate as a tool for feature generation/selection in classification. Performing SVD of large matrices is a computationally expensive task. In order to overcome this drawback, a number of computationally efficient schemes have been developed [9].

3.5.3 Example Problem

Taking same data matrix as in section 3.4.3

$$A = \begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \\ 2 & 14 & 7 & 23 \end{bmatrix}$$

Computing SVD of above matrix yields following matrices,

$$\begin{array}{r}
 -0.3929 \quad 0.5872 \quad 0.5501 \quad -0.3850 \quad 0.2236 \\
 -0.3990 \quad -0.2690 \quad 0.1160 \quad 0.5523 \quad 0.6708 \\
 U = -0.4058 \quad 0.1710 \quad 0.1950 \quad 0.5640 \quad -0.6708 \\
 -0.3724 \quad -0.7327 \quad 0.3131 \quad -0.4200 \quad -0.2236 \\
 -0.6189 \quad 0.1294 \quad -0.7402 \quad -0.2287 \quad 0.0000
 \end{array}$$

$$\begin{array}{r}
 42.1015 \quad 0 \quad 0 \quad 0 \\
 0 \quad 18.0196 \quad 0 \quad 0 \\
 S = 0 \quad 0 \quad 13.0419 \quad 0 \\
 0 \quad 0 \quad 0 \quad 2.5818 \\
 0 \quad 0 \quad 0 \quad 0
 \end{array}$$

$$\begin{array}{r}
 -0.3482 \quad 0.3839 \quad 0.8364 \quad -0.1782 \\
 -0.5191 \quad -0.5039 \quad -0.1293 \quad -0.6781 \\
 V = -0.4171 \quad -0.5516 \quad 0.2257 \quad 0.6862 \\
 -0.6598 \quad 0.5426 \quad -0.4824 \quad 0.1938
 \end{array}$$

S is a diagonal matrix, of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that

$$X = USV^T$$

Chapter 4: Artificial Neural Systems

Artificial neural systems are machines that have great potential to further improve the quality of life. Although computers outperform both biological and artificial neural systems for tasks based on precise and fast arithmetic operations, artificial neural systems represent the promising new generation of information processing networks. Advances have been made in applying such systems for problems found intractable or difficult for traditional computation. Neural networks can supplement the enormous processing power of the von Neumann digital computer with the ability to make sensible decisions and to learn by ordinary experience, as we do.

Network computation is performed by a dense mesh of computing nodes and connections. They operate collectively and simultaneously on most or all data and inputs. The basic processing elements of neural networks are called artificial neurons, or simply neurons. Often we simply call them nodes. Neurons perform as summing and nonlinear mapping junctions. In some cases they can be considered as threshold units that fire when their total input exceeds certain bias levels. Neurons usually operate in parallel and are configured in regular architectures. They are often organized in layers, and feedback connections both within the layer and toward adjacent layers are allowed. Each connection strength is expressed by a numerical value called a weight, which can be modified.

Artificial neural systems function as parallel distributed computing networks. Their most basic characteristic is their architecture. Only some of the networks provide instantaneous responses. Other networks need time to respond and are characterized by their time-domain behavior, which we often refer to as dynamics. Neural networks also differ from each other in their learning modes. There are a variety of learning rules that establish when and how the connecting weights change. Finally, networks exhibit different speeds and efficiency of learning. As a result, they also differ in their ability to accurately respond to the cues presented at the input. In contrast to conventional computers, which are programmed to perform specific tasks, most neural networks must be taught, or trained. They learn new associations, new patterns, and new functional dependencies. Learning rules and algorithms used for experiential training of networks replace the programming required for conventional computation. Neural network users do not

specify an algorithm to be executed by each computing node as would programmers of a more traditional machine. Instead, they select what in their view is the best architecture, specify the characteristics of the neurons and initial weights, and choose the training mode for the network. Appropriate inputs are then applied to the network so that it can acquire knowledge from the environment. As a result of such exposure, the network assimilates the information that can later be recalled by the user [10].

4.1 Neural Processing

Assume that a set of patterns can be stored in the network. Later, if the network is presented with a pattern similar to a member of the stored set, it may associate the input with the closest stored pattern. The process is called autoassociation. Typically, a degraded input pattern serves as a cue for retrieval of its original form. This is illustrated schematically in figure 4.1a. The figure shows a distorted square recalling the square encoded.

Associations of input patterns can also be stored in a heteroassociation variant. In heteroassociative processing, the associations between pairs of patterns are stored. This is schematically shown in figure 4.1b. A square input pattern presented at the input results in the rhomboid at the output. It can be inferred that the rhomboid and square constitute one pair of stored patterns. A distorted input pattern may also cause correct heteroassociation at the output as shown with dashed line.

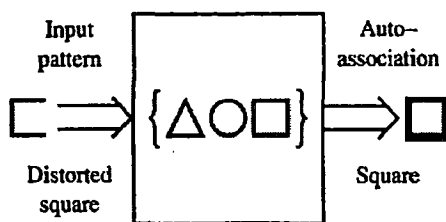


figure 4.1a Autoassociation

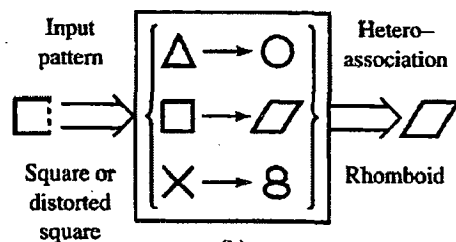


figure 4.1b Heteroassociation

Classification is another form of neural computation. Let us assume that a set of input patterns is divided into a number of classes, or categories. In response to an input pattern from the

set, the classifier is supposed to recall the information regarding class membership of the input pattern. Typically, classes are expressed by discrete-valued output vectors, and thus output neurons of classifiers would employ binary activation functions. The schematic diagram illustrating the classification response for patterns belonging to three classes is shown in figure 4.2a.

Classification can be understood as a special case of heteroassociation. The association is now between the input pattern and the second member of the heteroassociative pair, which is supposed to indicate the input's class number. If the network's desired response is the class number but the input pattern does not exactly correspond to any of the patterns in the set, the processing is called recognition. When a class membership for one of the patterns in the set is recalled, recognition becomes identical to classification. Recognition within the set of three patterns is schematically shown in figure 4.2b. This form of processing is of particular significance when an amount of noise is superimposed on input patterns.

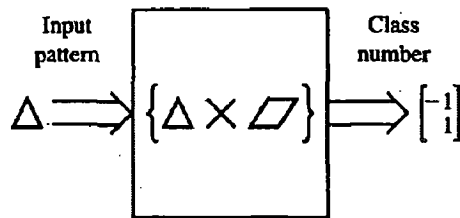


figure 4.2a Classification

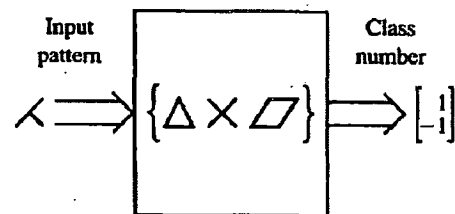


figure 4.2b Recognition

4.1.1 Perceptron

Perceptron is fundamental computational element in Neural Networks. figure 4.3 shows a Perceptron. Each external input is weighted with an appropriate weight w_{ij} , and the sum of the weighted inputs is sent to the transfer function, which also has an input of 1 transmitted to it through the bias. Depending upon transfer function used Perceptron can be discrete or continuous [9].

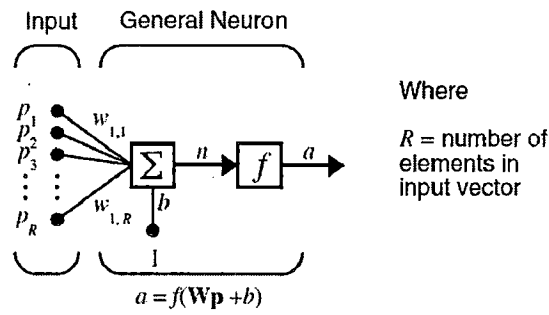


figure 4.3 A Perceptron

Most frequently used transfer functions are as follows

$$f(\text{net}) = 2 \times \left(\frac{1}{1 + e^{-\text{net}}} - \frac{1}{2} \right) \tag{4.1}$$

Its transfer characteristic is as given in figure 4.4

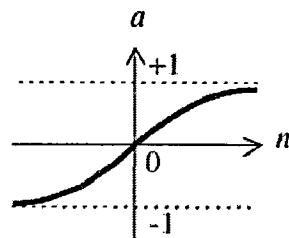


figure 4.4 Tansig Function

$$f(\text{net}) = 2 \times \left(\frac{1}{1 + e^{-\text{net}}} - \frac{1}{2} \right) \tag{4.2}$$

Its transfer characteristic is as given in figure 4.5

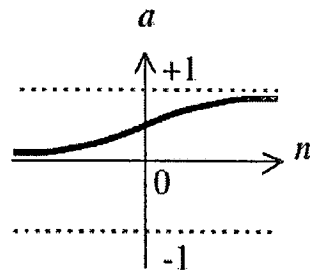


figure 4.5 Logsig Function

$$F(\text{net}) = \text{sgn}(\text{net}) \quad (4.3)$$

Its transfer characteristic is as given in figure 4.6

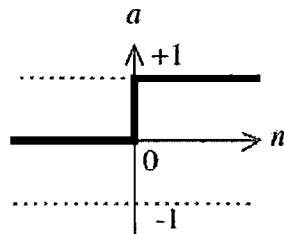


figure 4.6 Thresholding Function

4.2 Learning and Adaptation

Learning in human beings and animals is an inferred process; we cannot see it happening directly and one can assume that it has occurred by observing changes in performance. In general, learning is a relatively permanent change in behavior brought about by experience. Learning in neural networks is a more direct process, and we typically can capture each learning step in a distinct cause-effect relationship. To perform any of the processing tasks discussed in the previous section, neural network learning of an input-output mapping from a set of examples is needed. Designing an associator or a classifier can be based on learning a relationship that transforms inputs into outputs given a set of examples of input-output pairs [10].

4.2.1 Supervised and Unsupervised Learning

The majority of the neural networks require training in a supervised or unsupervised learning mode. Some of the networks, however, can be designed without incremental training. They are designed by batch learning rather than stepwise training. Batch learning takes place when the network weights are adjusted in a single training step. In this mode of learning, the complete set of input/output training data is needed to determine weights, and feedback information produced by the network itself is not involved in developing the network. This learning technique is also called recording. Learning with feedback either from the teacher or from the environment rather than a teacher, however, is more typical for neural networks. Such learning is called incremental and is usually performed in steps. figure 4.7a shows supervised learning while figure 4.7b shows unsupervised learning.

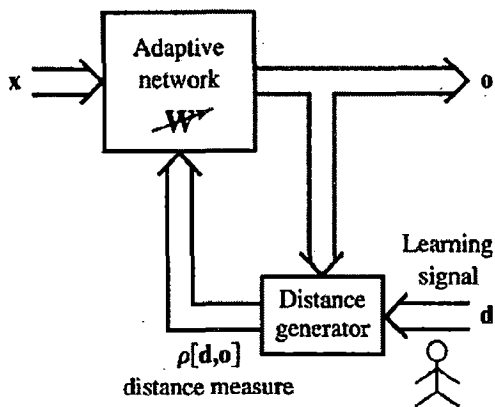


figure 4.7a Supervised Learning

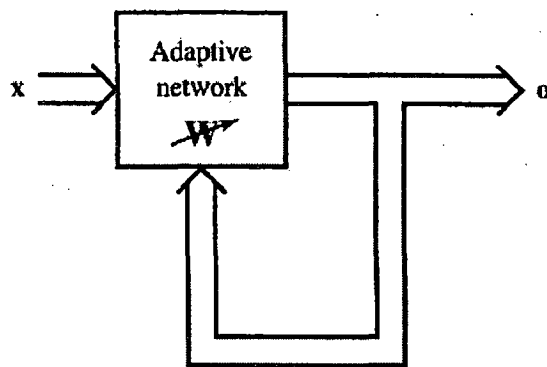


figure 4.7b Unsupervised Learning

In supervised learning we assume that at each instant of time when the input is applied, the desired response d of the system is provided by the teacher. This is illustrated in figure 4.7a. The distance between the actual and the desired response serves as an error measure and is used to correct network parameters externally. Since we assume adjustable weights, the teacher may implement a reward-and-punishment scheme to adapt the network's weight matrix W . For instance, in learning classifications of input patterns or situations with known responses, the error can be used to modify weights so that the error decreases. This mode of learning is very pervasive. Also, it is used in many situations of natural learning. A set of input and output patterns called a training set is required for this learning mode. Typically, supervised learning rewards accurate classifications or associations and punishes those which yield inaccurate responses. The teacher estimates the negative error gradient direction and reduces the error accordingly. In many situations, the inputs, outputs and the computed gradient are deterministic, however, the minimization of error proceeds over all its random realizations. As a result, most supervised learning algorithms reduce to stochastic minimization of error in multi-dimensional weight space.

figure 4.7b shows the block diagram of unsupervised learning. In learning without supervision, the desired response is not known; thus, explicit error information cannot be used to improve network behavior. Since no information is available as to correctness or incorrectness of responses, learning must somehow be accomplished based on observations of responses to inputs, that we have marginal or no knowledge about. For example; unsupervised learning can easily result in finding the boundary between classes of input patterns distributed as shown in figure

4.8a. In a favorable case, as in figure 4.8a, cluster boundaries can be found based on the large and representative sample of inputs. Suitable weight self-adaptation mechanisms have to be embedded in the trained network, because no external instructions regarding potential clusters are available. One possible network adaptation rule is: A pattern added to the cluster has to be closer to the center of the cluster than to the center of any other cluster.

Unsupervised learning algorithms use patterns that are typically redundant raw data having no labels regarding their class membership, or associations. In this mode of learning, the network must discover for itself any possibly existing patterns, regularities, separating properties, etc. While discovering these, the network undergoes change of its parameters, which is called self-organization. The technique of unsupervised learning is often used to perform clustering as the unsupervised classification of objects without providing information about the actual classes. This kind of learning corresponds to minimal a priori information available. Some information about the number of clusters, or similarity versus dissimilarity of patterns, can be helpful for this mode of learning. Finally, learning is often not possible in an unsupervised environment, as would probably be true in the case illustrated in figure 4.8b showing pattern classes not easily discernible even for a human[10].

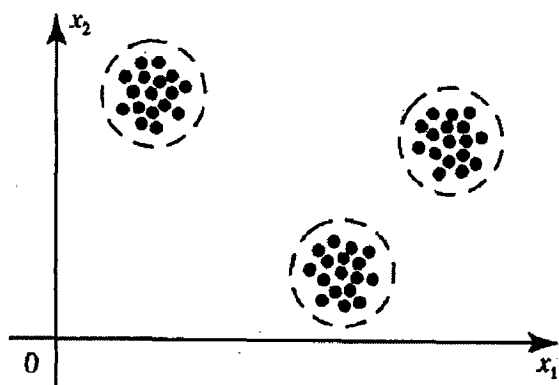


figure 4.8a Distinguishable Patterns

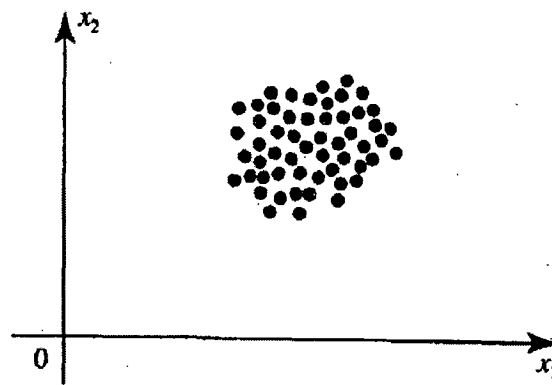


figure 4.8b Undistinguishable Patterns

4.3 Neural Network Learning Rules

A neuron is considered to be an adaptive element. Its weights are modifiable depending on the input signal it receives, its output value, and the associated teacher response. In some cases

the teacher signal is not available and no error information can be used, thus the neuron will modify its weights based only on the input and/or output. This is the case for unsupervised learning. The trained network is shown in figure 4.9.

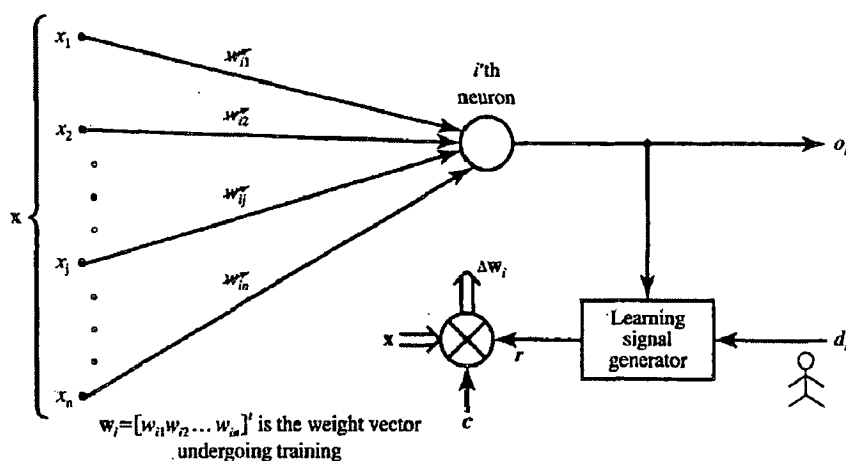


figure 4.9 Perceptron Under Training

4.3.1 Hebbian Learning Rule

For the Hebbian learning rule the learning signal is equal simply to the neuron's output. We have,

$$r = f(w_i^T x) \quad (4.1)$$

The increment Δw_i of the weight vector becomes

$$\Delta w_i = cf(w_i^T x)x \quad (4.2)$$

The single weight w_{ij} is adapted using the following increment

$$\Delta w_{ij} = cf(w_i^T x)x_j \quad (4.3)$$

This learning rule requires the weight initialization at small random values around $w_{ij} = 0$ prior to learning. The Hebbian learning rule represents a purely feed forward, unsupervised learning. The rule states that if the cross product of output and input, or correlation term $o_i x_j$ is positive, this results in an increase of weight w_{ij} otherwise the weight decreases. It can be seen that the output is strengthened in turn for each input presented. Since its inception, the Hebbian rule has evolved in a number of directions. In some cases, the Hebbian rule needs to be modified to counteract

unconstrained growth of weight values, which takes place when excitations and responses consistently agree in sign. This corresponds to the Hebbian learning rule with saturation of the weights at certain, preset level [10].

4.3.2 Perceptron Learning Rule

For the Perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response. Thus is, supervised and the learning signal is equal to,

$$r = (d_i - o_i) \quad (4.4)$$

where $o_i = \text{sgn}(w_i^t x)$, and d_i , is the desired response as shown in figure 4.10. Weight adjustments in this method, Δw_i , and Δw_{ij} , are obtained as follows,

$$\Delta w_i = c [d_i - \text{sgn}(w_i^t x)] x \quad (4.5)$$

$$w_{ij} = c [d_i - \text{sgn}(w_i^t x)] x_j \quad \text{for } j=1, \dots, n \quad (4.6)$$

This rule is applicable only for binary neuron response, and the relationships (4.5) and (4.6) express the rule for the bipolar binary case. Under this rule, weights are adjusted if and only if o_i is incorrect. Error as a necessary condition of learning is inherently included in this training rule. Obviously, since the desired response is either 1 or -1, the weight adjustment (4.5) reduces to

$$\Delta w_i = \pm 2cx \quad (4.7)$$

Where a plus sign is applicable when $d_i = 1$, and $\text{sgn}(w_i^t x) = -1$, and a minus sign is applicable when $d_i = -1$, and $\text{sgn}(w_i^t x) = 1$. The weight adjustment formula (4.7) cannot be used when $d_i = \text{sgn}(w_i^t x)$. The weight adjustment is inherently zero when the desired and actual responses agree [10].

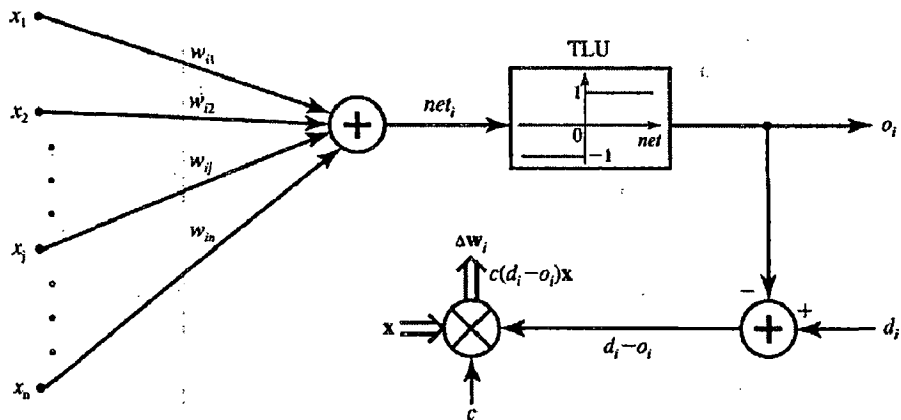


figure 4.10 Perceptron Learning Rule

4.3.3 Delta Learning Rule

The delta learning rule is only valid for continuous activation functions, and in the supervised training mode. The learning signal for this rule is called delta and is defined as follows,

$$r = [d_i - f(w_i^t x)] f'(w_i^t x) \quad (4.8)$$

The term $f'(w_i^t x)$ is the derivative of the activation function $f(\text{net})$ computed for $\text{net} = w_i x$. The explanation of the delta learning rule is shown in figure 4.11. This learning rule can be readily derived from the condition of least squared error between o_i and d_i . Calculating the gradient vector with respect to w_i of the squared error defined as,

$$E = \frac{1}{2} (d_i - o_i)^2 \quad (4.9)$$

Which is equivalent to

$$E = \frac{1}{2} [d_i - f(w_i^t x)]^2 \quad (4.10)$$

Error gradient vector value is given by

$$\nabla E = -(d_i - o_i) f'(w_i^t x) x \quad (4.11)$$

The components of the gradient vector are

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i) f'(w_i^t x) x_j \quad \text{for } j=1, \dots, n \quad (4.12)$$

Since the minimization of the error requires the weight changes to be in the negative gradient direction, we take

$$\Delta w_i = -\eta \nabla E \quad (4.13)$$

Where η is a positive constant . We then obtain from (4.11) and (4.13)

$$\Delta w_i = \eta(d_i - o_i) f'(net_i) x \quad (4.14)$$

Or, for the single weight the adjustment becomes

$$\Delta w_{ij} = \eta(d_i - o_i) f'(net_i) x_j \quad j=1, \dots, n \quad (4.15)$$

Weight adjustment as in (4.14) is computed based on minimization of the squared error.

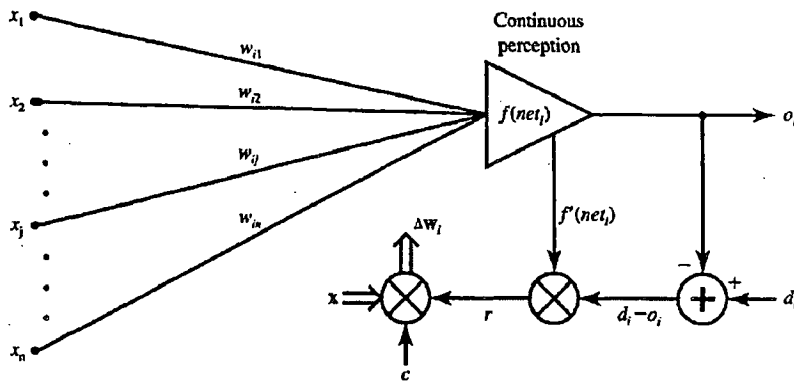


figure 4.11 Delta Learning Rule

The delta rule was introduced only recently for neural network training. This rule parallels the discrete Perceptron training rule. It also can be called the continuous Perceptron training rule. The delta learning rule can be generalized for multilayer networks [10].

4.3.4 Widrow-Hoff Learning Rule

The Widrow-Hoff learning rule is applicable for the supervised training of neural networks. It is independent of the activation function of neurons used since it minimizes the squared error between the desired output value d , and the neuron's activation value $net_i = w_i^t x$. The learning signal for this rule is defined as follows,

$$r = d_i - w_i^t x \quad (4.16)$$

The weight vector increment under this learning rule is,

$$\Delta w_i = c(d_i - w_i^t x) x \quad (4.17)$$

or, for the single weight the adjustment is,

$$\Delta w_{ij} = c(d_i - w_i^t x) x_j \quad (4.18)$$

This rule can be considered a special case of the delta learning rule. Indeed, assuming in (4.8) that $f(w_i^t x) = w_i^t x$, or the activation function is simply the identity function $f(\text{net}) = \text{net}$, we obtain $f'(\text{net}) = 1$, and (4.8) becomes identical to (4.16). This rule is sometimes called the LMS (least mean square) learning rule. Weights are initialized at any values in this method [10].

4.3.4 Winner Takes All Learning Rule

This learning rule differs substantially from any of the rules discussed so far. It can only be demonstrated and explained for an ensemble of neurons, preferably arranged in a layer of p units. This rule is an example of competitive learning, and it is used for unsupervised network training. Typically, winner takes all learning is used for learning statistical properties of inputs. The learning is based on the premise that one of the neurons in the layer Say m 'th has the maximum response due to input x , as shown in figure 4.12. This neuron is declared the winner. As a result of this winning event the weight vector w_m

$$w_m = [w_{m1} \quad w_{m2} \quad \dots \quad w_{mn}]^T \quad (4.19)$$

Weight increment is computed as follows,

$$\Delta w_m = \alpha(x - w_m) \quad (4.20)$$

or, the individual weight adjustment becomes,

$$\Delta w_{mj} = \alpha(x_j - w_{mj}) \quad \text{for } j=1, \dots, n \quad (4.21)$$

Where $\alpha > 0$ is a small learning constant, typically decreasing as learning progresses. The winner selection is based on following criterion of maximum activation among all p neurons participating in competition.

$$w_m^t x = \max_{i=1,2,\dots,P} (w_i^t x) \quad (4.22)$$

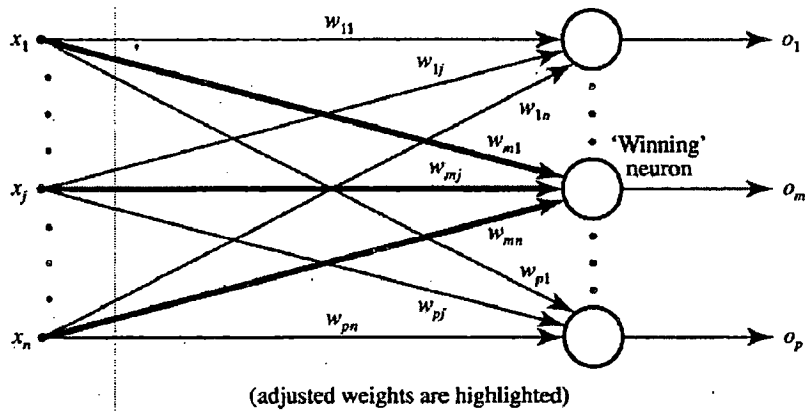


figure 4.12 Winner Takes All Learning Rule

4.4 Neural Network Training

figure 4.13 shows a Perceptron which is also called Threshold logic Unit (TLU).

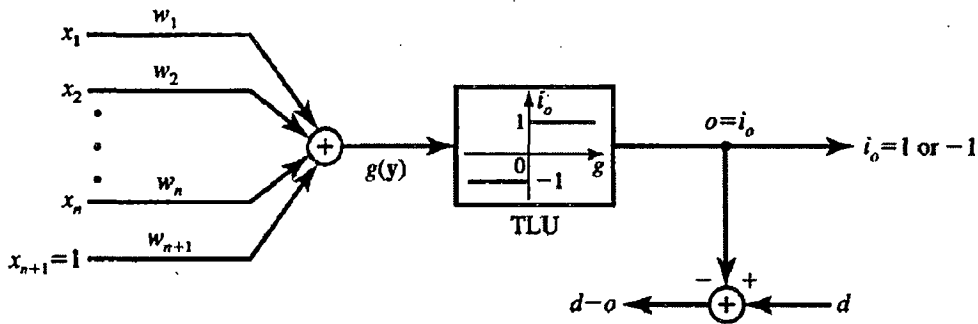


figure 4.13 Neural Network Training

Value of output at summation junction is called 'net' and is given by,

$$\text{net} = x_1 w_1 + x_2 w_2 + \dots + x_n w_n + w_{n+1} \tag{4.23}$$

This is an equation of hyperplane in n dimensional space. w_{n+1} is bias value also represented as weight with fixed input 1. With proper weight values TLU element can classify, on which side of the plane the pattern lies, 1 for positive side -1 for negative side respectively[.].

4.4.1 Backpropagation Algorithm

In this dissertation Feed-Forward two layer Neural Network has been used, hidden layer contained ten neurons. Feed-Forward Neural Networks are also called back-propagation Neural Networks because of the training method used. Back-Propagation is a gradient descent algorithm for multilayer network with non-linear differentiable transfer function, in which network weights are moved in the direction of negative gradient of performance function. Weight adjustment is repeated using (4.24) to (4.32), until performance function value decreases below a certain low value. Let z be input and o be target vectors of size $I \times 1$ and $K \times 1$ respectively and hidden layer contain J neurons so output y of hidden layer is of size $J \times 1$. Layer weight matrix W and input weight matrix V are of sizes $K \times J$ and $J \times I$ respectively, are initialized to small random values, w_{kj} denote weight connecting k^{th} output neuron to j^{th} hidden layer neuron, similarly v_{ji} denote weight connecting j^{th} hidden neuron to i^{th} input component[2].

$$y_j \leftarrow f(\text{net}_j) \quad (4.24)$$

$$\text{net}_j = v_j^t z \quad (4.25)$$

Where $j=1, 2, \dots, J$

Where v_j , a column vector, is j^{th} row of V

$$o_k \leftarrow f(\text{net}_k) \quad (4.26)$$

$$\text{net}_k = w_k^t y \quad (4.27)$$

Where $k=1, 2, \dots, K$

Where w_k , a column vector, is k^{th} row of W

Where most common choice of function f is bipolar continuous activation function given by equation:

$$f(\text{net}) = 2 \times \left(\frac{1}{1 + e^{-\text{net}}} - \frac{1}{2} \right) \quad (4.28)$$

However, it can be any nonlinear continuous differentiable function.

Error signal vectors for each layer, δ_o and δ_y of size $K \times 1$ and $J \times 1$ are computed using following equations:

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)(1 - o_k^2) \quad (4.29)$$

Where d_k is desired output of k^{th} neuron where

$k=1, 2, \dots, K$

$$\delta_{yj} = \frac{1}{2}(1 - y_j^2) \sum_{k=1}^K \delta_{ok} w_{kj} \quad (4.30)$$

for $j=1, 2, \dots, J$

Output layer weights are adjusted by equation:

$$w_{kj} \leftarrow w_{kj} + \eta \delta_{ok} y_j \quad (4.31)$$

for $k=1, 2, \dots, K$ and

$j=1, 2, \dots, J$

Hidden layer weights are adjusted by equation:

$$v_{ji} \leftarrow v_{ji} + \eta \delta_{yj} z_i \quad (4.32)$$

for $j=1, 2, \dots, J$ and

$i=1, 2, \dots, I$

Where η is learning rate.

Common choice performance function, which is optimized using (4.24) to (4.32), is mean square error, which is given by following equation [10].

4.4.2 Resilient Backpropagation

In this dissertation Resilient Backpropagation as training algorithm has been used. Multilayer networks typically use sigmoid transfer functions in the hidden layers and output layer. From (4.29) and (4.30) we find that error signal vector uses slope of transfer function for calculation, now when input is large derivative of (4.28) is small, which makes error signal vector small, as a result of which weight adjustment is slow even if weights are far from optimal value. Resilient Backpropagation eliminates these harmful effects. In this algorithm magnitude of derivative have no effect on weight adjustment, rather sign of the derivative determines the direction of weight adjustment. The size of the weight change is determined by a separate term `update_value`. The `update_value` for each weight and bias is increased by a factor `delt_inc` whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by a factor `delt_dec` whenever the derivative with respect to that weight changes sign from the previous iteration. If the derivative is zero, the update value remains the same. This algorithm leads to faster convergence.

Chapter 5: Software for Face Recognition

5.1 Algorithm

The method proposed uses Laplacian for pre-processing. Singular Value Decomposition for feature extraction and two layer Feed-Forward Neural Network for recognition and classification. Database contained Facial image of ten different persons, each person has ten different variations of facial expression.

A. Pre-Processing

Four images from each group were taken, and processed using (2.6). This step resulted in sharp image with enhanced small details as shown in figure 2.5. Application of laplacian was done by mask shown in figure 2.2. Laplacian operator results into image with some pixel values as negative, which are truncated to zero by display device. So, integer precision image was converted to double precision before applying Laplacian.

B. Training Vectors Creation

Images after pre-processing stage were arranged in the form of matrix such that, each column of matrix contains an image data. Image size we used was 92×112 so, resultant matrix was of size 10304×40 . This Matrix was replicated into three sets, second and third set were added with normally distributed noise of mean zero and standard deviation 0.05 and 0.1 respectively. This step was taken to improve noise tolerance of Neural Network. Singular Value Decomposition was applied to this matrix of size 10304×120 . As a result Feature Matrix of size 120×120 was created, which was used to train the neural network. Further, orthonormal basis matrix of size 10304×120 was also created in this step.

C. Neural Network Training

Columns of Feature Matrix created above were used to train neural network. Various Training parameters used were, number of epochs was set to 5000, performance function was sum square error, Goal was set to 0.1, Ten hidden neurons were used, Transfer function for hidden layer and output layer were tan-sigmoid and log-sigmoid respectively. Learning rate was set to 0.05. Training function used was resilient Backpropagation with `update_value=0.07`,

delt_dec=0.5 and delt_inc=1.2. Resilient Backpropagation resulted in faster convergence than conventional gradient descent algorithm.

D. Network Simulation

Face to be recognized was first pre-processed and, then its projection was taken along orthonormal basis vectors which columns of orthonormal basis matrix are obtained using Singular Value Decomposition by following equation:

$$y_{\text{test}} = U^T x_{\text{test}} \quad (4.33)$$

Where x_{test} is image data with or without noise, arranged in lexicographic form, U is basis matrix, y_{test} is feature vector of Face to be recognized. Trained network responded with a 10×1 vector with maximum value, close to one in one position, showing class to which input face belonged.

5.2 Program for Computation Principal Component Analysis

```
function [mn,signals,PC,V] = pca_fcn(data)
%data=lxn data matrix each columns are samples with l dimensions
%mn=column vector with mean of each rows
%signals=mxn dimension output matrix where m<l
%V=variance of each dimensions of output matrix
[M,N] = size(data);
mn = mean(data,2);
data = data - repmat(mn,1,N);
covariance = 1 / (N-1) * data * data';
[PC, V] = eig(covariance);
V = diag(V);
[junk, rindices] = sort(-1*V);
V = V(rindices);
PC = PC(:,rindices);
signals = PC'*data;
```

5.3 Program for Creation of Training Set

```
function [P,T]=create_db(str)
%str=cell array of strings containing image address
%P=pattern data
%T=target data
[m n]=size(str);
image=imread(str{1,1});
[x,y]=size(image(:,:,1));
P=zeros(x*y,m*n);
T=zeros(m,m*n);
%[p q]=size(P);
w=fspecial('log',[5 5],5);
for i=1:m
for j=1:n
img =imread(str{i,j});
z=img(:,:,1);
z=im2double(z);
z=z-imfilter(z,w,'replicate','same');
%size(z)
%imshow(z);
%imsave
P(:,n*(i-1)+j)=z(:);
T(i,n*(i-1)+j)=1;
end
end
%[P,ps]=mapstd(P);
```

5.4 Program for Creation of Noisy Training Set

```
function [P T]= createnoisy_db(R,G)
%R=lxn matrix each column is an log tranformed image of type double
%P=lx 3*n matrix first n columns has original image n+1 to 3n columns
% has image with normally distributed noise
[m n]=size(R);
rows=112;
cols=92;
R1= R+ randn(m,n).* .1;
R2= R+randn(m,n).* .05;
P=[R R1 R2];
T=[G G G];
img=zeros(rows,cols);
[x y]=size(P);
for i=1:y
for j=1:cols
img(:,j)=P(((j-1)*rows+1):rows*j,i);
end
%imshow(img);
%imsave
end
```

5.5 Program for Computation of Singular Value Decomposition

```
function [Y U] =svd_compute(P)
%P=log transformed image matrix in lexicographic form
%size of P is imrows*imcolumns X numImages
%Y=m X numImages size matrix
[U S V]=svd(P,'econ');
Y=S*V';
Q=U*Y;
```

```

Y1=U'*P;
rows=112;
cols=92;
img=zeros(rows,cols);
[m n]=size(Q);
for i=1:n
for j=1:cols
img(:,j)=Q(((j-1)*rows+1):rows*j,i);
end
%imshow(img);
%imsave
end

```

5.6 Program for Neural Network Training

```

function [net tr]= train_fcn(P,T,Goal,Epochs,PFcn,Trngfcn,numHneuron,tf1,tf2,lrnRate)
%various parameters in order
%P=pattern vectors
%T=target vectors
%Goal=value of sum square error
%Epochs=num of iterations
%Pfcn='sse' or 'mse' etc.
%Trngfcn=training function used
%numHneurons=number of neurons in hidden layers
%tf1=string containing transfer function of first layer
%tf2=string containing transfer function of 2nd layer
%lr=network learning rate
%net=returned trained network
net = newff(P,T,numHneuron,{tf1,tf2});
net.trainFcn=Trngfcn;
net.IW{1,1} = net.IW{1,1}*0.01;

```

```

net.outputs{2}.processParams{2}.ymin=0;
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
net.b{1} = net.b{1}*0.01;
net.divideFcn='dividerand';
net.divideParam.trainRatio=1.00;
net.divideParam.valRatio=0.00;
net.divideParam.testRatio=0.00;
net.performFcn=PFcn;
net.inputs{1}.processFcns={'fixunknowns' 'removeconstantrows' 'mapminmax','mapstd'};
%net.outputs{2}.processFcns={'mapstd'};
net.trainParam.showCommandLine=1;
net.trainParam.goal=Goal;
net.trainParam.epochs=Epochs;
net.trainParam.lr=lrnRate;
[net,tr] = train(net,P,T);

```

5.7 Program for Simulation Neural Network

```

function [O result Y]= simulate_fcn(str,U,noiseLevel,net)
%str=one row string cell array having address of images to be recognised
%U=matrix whose columns are basis vectors used for transforming input
%image into reduced subspace
%noiseLevel=amount of noise level to be added in input image
%net=network after training
[m,n]=size(str);
image=imread(str{1,1});
[x,y]=size(image(:, :, 1));
P=zeros(x*y,m*n);
w=fspecial('log',[5 5],.5);
noise=randn(x,y).*(noiseLevel);
for i=1:m

```



```

for j=1:n
img =imread(str{i,j});
z=img(:,:,1);
z=im2double(z);
z=z-imfilter(z,w,'replicate','same');
z=z+noise;
imshow(z);
%imsave;
P(:,n*(i-1)+j)=z(:);
end
end
Y=U'*P;
O=sim(net,Y);
% O = compet(O);
[rows,cols]=size(O);
result=zeros(rows,cols);
[Z,index]=max(O);
for i=1:cols
%if(Z(i)>.4)
result(index(i),i)=Z(i);
end
if(index==1)
figure,imshow('F:\face images\A\A1.jpg');
elseif(index==2)
figure,imshow('F:\face images\B\B1.jpg');
elseif(index==3)
figure,imshow('F:\face images\C\C1.jpg');
elseif(index==4)
figure,imshow('F:\face images\D\D1.jpg');
elseif(index==5)
figure,imshow('F:\face images\E\E1.jpg');

```

```
elseif(index==6)
figure,imshow('F:\face images\F\F1.jpg');
elseif(index==7)
figure,imshow('F:\face images\G\G1.jpg');
elseif(index==8)
figure,imshow('F:\face images\H\H1.jpg');
elseif(index==9)
figure,imshow('F:\face images\I\I1.jpg');
elseif(index==10)
figure,imshow('F:\face images\J\J1.jpg');
end
```

Chapter 6: Results

Neural Networks designed with the proposed method recognized images with an accuracy rate of 91% at zero noise level. Noise response of the network has been studied for twenty different noise levels. It was found that Neural Network designed by proposed method tolerated the noise extremely well. As shown in the plot of figure 6.4. Though recognition rate decreased as noise increased, it is 83%, which is appreciably high, even at random noise of standard deviation 0.2 and mean zero. figure 6.1 shows Facial images contaminated by normally distributed random noise with zero mean value and various standard deviations. The training method used is Resilient Backpropagation, which led to faster learning of the Network. It took 37 epochs to converge or to reach the goal. As shown in the performance plot of figure 6.3.

figure 6.2a and figure 6.2b shows Training and Test images respectively. Out of ten images of each person four has been used for training the network and remaining six test images which network has never seen has been used for verification.

Section 6.1 shows Confusion matrix from Table I to Table XX. A confusion matrix is analysis of Network Performance at a given noise level, in a confusion matrix ten persons are denoted by alphabets A to J, where columns denote group indicated by the network and rows indicate actual group to which person belong. For example, say in Matrix shown in Table I, a particular cell say (C, B) shows a number 2, which means that 2 images that belonged to group C were erroneously grouped into group B by the Network. Similarly a cell (C, C) which shows number 7 indicates that 7 images of group were classified correctly by the Network. Sum of diagonal elements in confusion matrix is proportional to the accuracy of the Network. Accuracy of the algorithm was found to be 91% under zero random noise condition.

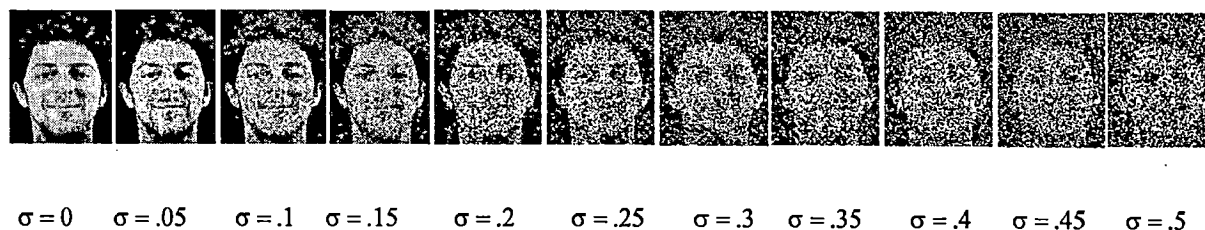


figure 6.1 Various Noise levels

Training Images

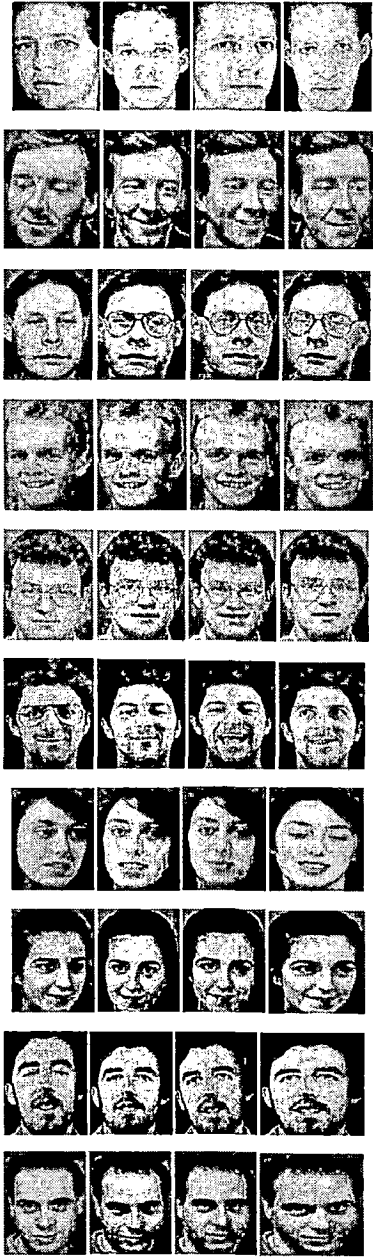


figure 6.2a

Test Images



figure 6.2b

6.1 Confusion Matrix for Various Noise Levels

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	10	0	0	0	0	0	0	0	0
C	1	2	7	0	0	0	0	0	0	0
D	0	0	0	8	2	0	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	0	0	0	10	0	0	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	0	0	0	1	0	0	9	0	0
I	0	0	0	0	0	0	1	0	9	0
J	0	1	0	0	1	0	0	0	0	8

Table I. Noise Level=0; 91% Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	10	0	0	0	0	0	0	0	0
C	1	2	7	0	0	0	0	0	0	0
D	0	0	0	8	2	0	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	0	0	0	10	0	0	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	0	0	0	1	0	0	9	0	0
I	0	0	0	0	0	0	1	0	9	0
J	0	1	0	0	1	0	0	0	0	8

Table II. Noise Level=0.05; 91% Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	10	0	0	0	0	0	0	0	0
C	1	2	7	0	0	0	0	0	0	0
D	0	1	0	7	2	0	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	0	0	1	9	0	0	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	0	0	0	1	0	0	9	0	0
I	0	0	0	0	0	0	1	0	9	0
J	0	1	0	0	1	0	2	0	0	6

Table III. Noise Level=0.1; 89% Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	8	0	2	0	0	0	0	0	0
C	0	2	8	0	0	0	0	0	0	0
D	0	0	0	8	0	0	1	0	1	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	0	0	1	9	0	0	0	0
G	0	0	1	0	1	0	8	0	0	0
H	0	0	0	0	1	0	0	9	0	0
I	0	1	0	0	0	0	0	0	9	0
J	0	0	1	0	0	0	2	0	0	7

Table IV. Noise Level=0.15; 86% Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	9	0	1	0	0	0	0	0	0
C	1	2	6	0	0	1	0	0	0	0
D	0	1	0	7	2	0	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	2	0	0	8	0	0	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	0	0	0	1	0	0	9	0	0
I	0	0	0	1	0	0	1	0	8	0
J	0	1	0	0	1	0	2	0	0	6

Table V Noise Level=0.2; 83% Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	8	0	2	0	0	0	0	0	0
C	1	2	6	0	1	0	0	0	0	0
D	0	1	0	5	2	2	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	2	1	0	7	0	0	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	1	0	0	1	0	0	8	0	0
I	0	0	0	0	0	0	1	0	9	0
J	0	1	0	2	1	0	1	0	0	5

Table VI. Noise Level=.25; 78% Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	8	0	2	0	0	0	0	0	0
C	1	2	6	0	0	1	0	0	0	0
D	1	2	0	5	2	0	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	0	0	2	0	0	7	0	1	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	1	0	0	1	0	0	8	0	0
I	0	0	0	0	0	0	1	0	9	0
J	0	1	2	0	1	0	1	0	0	5

Table VII. Noise Level=0.3; 78 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	8	0	2	0	0	0	0	0	0
C	1	2	7	0	0	0	0	0	0	0
D	0	0	0	6	2	2	0	0	0	0
E	0	0	0	0	10	0	0	0	0	0
F	2	0	2	0	0	6	0	0	0	0
G	0	0	0	2	1	1	6	0	0	0
H	0	3	0	0	1	0	0	6	0	0
I	0	0	0	1	0	0	1	0	8	0
J	0	1	2	0	1	1	0	0	0	5

Table VIII. Noise Level=0.35; 72 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	7	1	2	0	0	0	0	0	0
C	1	2	6	0	0	1	0	0	0	0
D	1	0	0	5	2	0	2	0	0	0
E	0	0	2	0	8	0	0	0	0	0
F	0	2	0	0	0	6	0	2	0	0
G	0	0	0	0	0	0	10	0	0	0
H	0	0	3	0	0	0	0	7	0	0
I	0	0	0	1	0	0	1	0	8	0
J	0	1	0	0	1	0	3	0	0	5

Table IX Noise Level=0.4; 72 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	9	0	0	0	1	0	0	0	0	0
B	0	6	0	0	0	2	2	0	0	0
C	1	2	6	0	0	0	0	0	1	0
D	0	0	0	6	2	0	0	2	0	0
E	0	0	0	0	9	0	1	0	0	0
F	0	0	2	2	0	6	0	0	0	0
G	1	2	0	0	0	0	7	0	0	0
H	0	0	3	0	1	0	0	6	0	0
I	0	0	0	1	0	0	1	0	8	0
J	0	1	0	0	1	2	0	2	0	4

Table X Noise Level=0.45; 67 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	10	0	0	0	0	0	0	0	0	0
B	0	6	2	2	0	0	0	0	0	0
C	1	2	4	0	0	3	0	0	0	0
D	0	0	2	5	2	0	0	1	0	0
E	0	1	0	0	9	0	0	0	0	0
F	0	0	2	0	0	6	2	0	0	0
G	0	0	0	3	0	0	7	0	0	0
H	0	3	0	0	1	0	0	6	0	0
I	0	0	2	0	0	0	1	0	7	0
J	0	1	2	2	1	0	0	0	0	4

Table XI Noise Level=0.5; 64 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	8	0	0	0	0	0	0	0	2	0
B	0	6	0	0	0	2	0	2	0	0
C	1	2	6	0	0	0	0	1	0	0
D	0	2	0	5	2	0	1	0	0	0
E	0	1	0	0	9	0	0	0	0	0
F	0	2	2	0	0	6	0	0	0	0
G	0	0	0	0	3	0	7	0	0	0
H	0	0	0	0	1	2	2	6	0	0
I	0	0	0	3	0	0	1	0	6	0
J	0	2	1	0	3	1	0	0	0	3

Table XII Noise Level=0.55; 62 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	8	0	0	0	2	0	0	0	0	0
B	0	6	2	2	0	0	0	0	0	0
C	1	2	6	0	0	0	1	0	0	0
D	0	0	0	5	2	0	1	2	0	0
E	0	0	0	0	8	0	2	0	0	0
F	0	0	2	0	2	6	0	0	0	0
G	0	2	0	2	0	0	3	0	3	0
H	0	2	2	0	1	2	0	3	0	0
I	0	0	0	3	0	0	1	0	6	0
J	0	1	2	0	1	2	1	0	0	3

Table XIII Noise Level=0.6; 54 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	7	0	2	0	0	1	0	0	0	0
B	0	6	0	0	2	2	0	0	0	0
C	1	2	5	0	0	0	2	0	0	0
D	0	0	0	5	2	0	2	1	0	0
E	0	0	0	0	8	2	0	0	0	0
F	0	0	0	0	0	5	4	1	0	0
G	0	0	0	0	2	0	7	0	1	0
H	0	0	3	3	1	0	0	3	0	0
I	0	0	0	2	2	0	1	0	5	0
J	0	1	4	1	1	0	0	0	0	3

Table XIV Noise Level=0.65; 54 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	8	0	0	2	0	0	0	0	0	0
B	0	6	0	0	2	2	0	0	0	0
C	1	2	4	0	0	3	0	0	0	0
D	0	0	2	4	2	0	0	2	0	0
E	0	0	2	0	6	0	2	0	0	0
F	0	2	0	3	0	5	0	0	0	0
G	0	0	2	2	0	0	6	0	0	0
H	0	3	0	2	1	0	0	4	0	0
I	0	2	2	0	0	0	1	0	5	0
J	0	1	3	0	1	2	0	0	0	3

Table XV Noise Level=0.7; 53 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	8	0	2	0	0	0	0	0	0	0
B	0	6	2	2	0	0	0	0	0	0
C	2	2	4	0	0	0	0	4	0	0
D	0	4	0	4	2	0	0	0	0	0
E	0	0	0	3	3	0	0	3	1	0
F	0	0	0	2	0	5	3	0	0	0
G	0	0	2	0	2	0	6	0	0	0
H	0	3	0	0	1	2	0	4	0	0
I	0	2	0	2	0	0	1	0	5	0
J	0	1	0	0	1	3	2	0	0	3

Table XVI Noise Level=0.75; 48 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	5	0	2	2	0	1	0	0	0	0
B	0	6	0	3	0	0	0	1	0	0
C	1	2	5	0	0	2	0	0	0	0
D	0	0	0	4	2	0	3	1	0	0
E	0	3	2	0	3	0	2	0	0	0
F	0	3	0	3	0	4	0	0	0	0
G	0	0	2	4	0	0	4	0	0	0
H	2	0	0	2	1	3	0	2	0	0
I	0	1	0	3	3	0	1	0	3	0
J	2	1	0	0	1	3	0	0	0	3

Table XVII Noise Level=0.8; 38 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	5	0	2	3	0	0	0	0	0	0
B	0	4	0	4	2	0	0	0	0	0
C	1	2	2	0	3	0	2	0	0	0
D	1	0	0	2	2	0	2	0	3	0
E	0	2	1	0	3	0	0	4	0	0
F	0	0	3	4	0	3	0	0	0	0
G	1	0	0	2	2	0	5	0	0	0
H	0	3	0	0	1	4	0	2	0	0
I	0	2	0	3	0	0	1	0	4	0
J	0	1	0	0	2	3	0	0	0	4

Table XVIII Noise Level=0.85; 34 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	6	0	2	0	0	2	0	0	0	0
B	0	4	0	3	1	2	0	0	0	0
C	2	3	2	0	0	2	1	0	0	0
D	0	0	3	1	2	0	2	1	1	0
E	1	0	3	0	2	4	0	0	0	0
F	1	3	0	5	0	1	0	0	0	0
G	0	0	0	0	0	0	2	0	0	0
H	0	2	4	0	1	0	0	3	0	0
I	0	0	3	0	3	0	1	0	3	0
J	0	1	2	1	2	1	0	0	0	3

Table XIX Noise Level=0.9; 27 % Recognition Rate

	A	B	C	D	E	F	G	H	I	J
A	6	2	0	2	0	0	0	0	0	0
B	0	4	2	0	4	0	0	0	0	0
C	1	2	1	0	3	1	2	0	0	0
D	0	2	0	2	2	0	4	0	0	0
E	0	1	4	0	1	0	4	0	0	0
F	0	0	3	0	0	1	0	4	2	0
G	0	0	0	0	0	0	2	0	0	0
H	0	3	0	5	1	0	0	2	0	0
I	0	0	2	3	0	0	1	0	4	0
J	0	2	0	3	1	0	0	0	0	4

Table XX Noise Level=0.95; 24 % Recognition Rate

6.2 Performance Plot

Performance Plot gives values of performance function as training vectors are presented to Neural Network. Epoch is measure of number of training vectors. As shown in the figure 6.3 initially Error value which in this case is sum square error was close to 300. As training proceeded error value reduced and met the dashed goal line at epoch number 37.

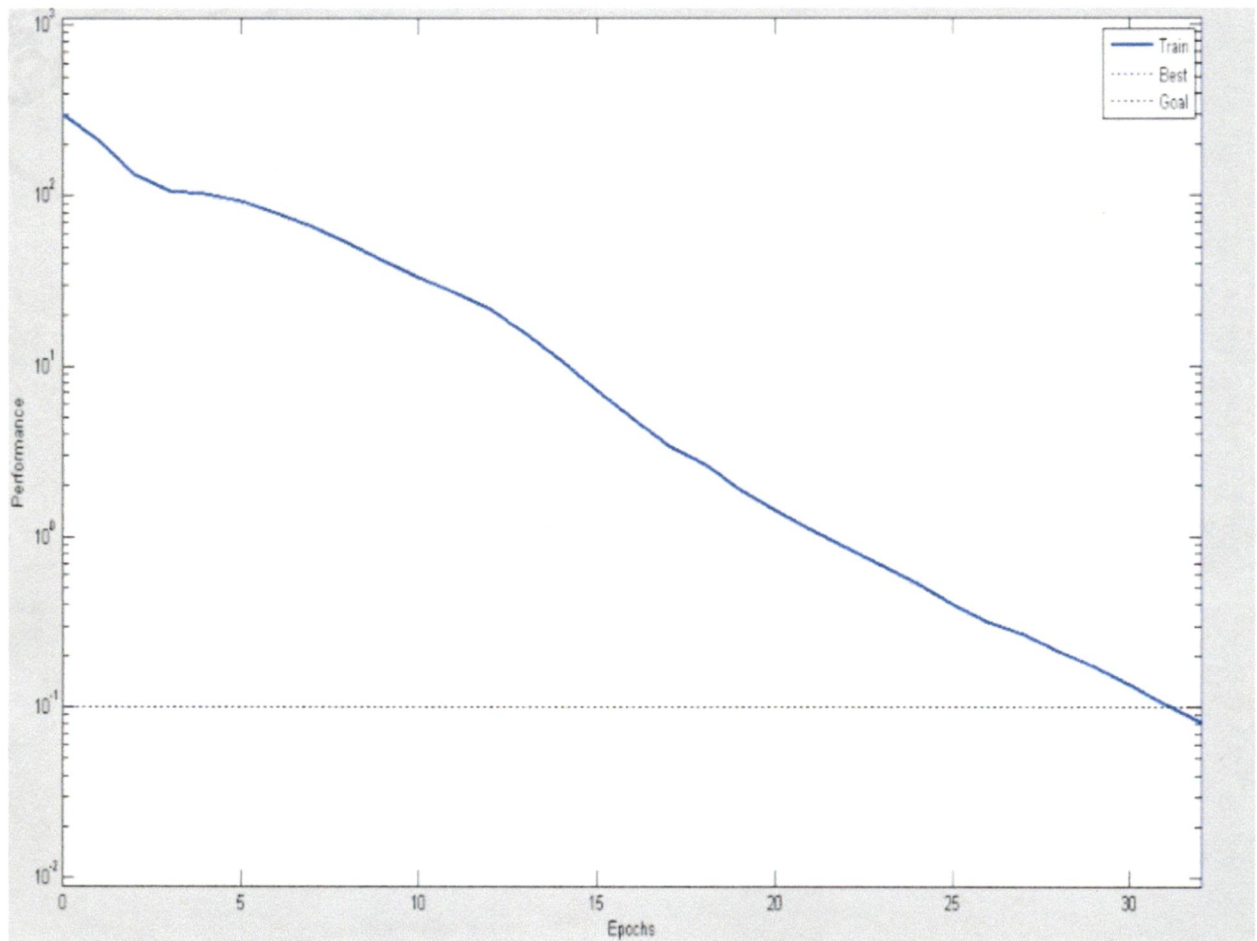


figure 6.3 Performance Plot

6.3 Worst Case Accuracy Plot

This plot is called worst case accuracy plot because, each image was tested 10 times under different noisy conditions and worst recognition of all 10 runs was used to create the plot. It was done because under noisy condition, though we know mean and standard deviation of noise, noise value at a particular pixel position were unknown. So in some runs noise values may affect , those parts which help in recognition the most, like mouth, nose, eyes etc. while in some runs noise only affect redundant information which had no effect on recognition. Hence worse case was picked.

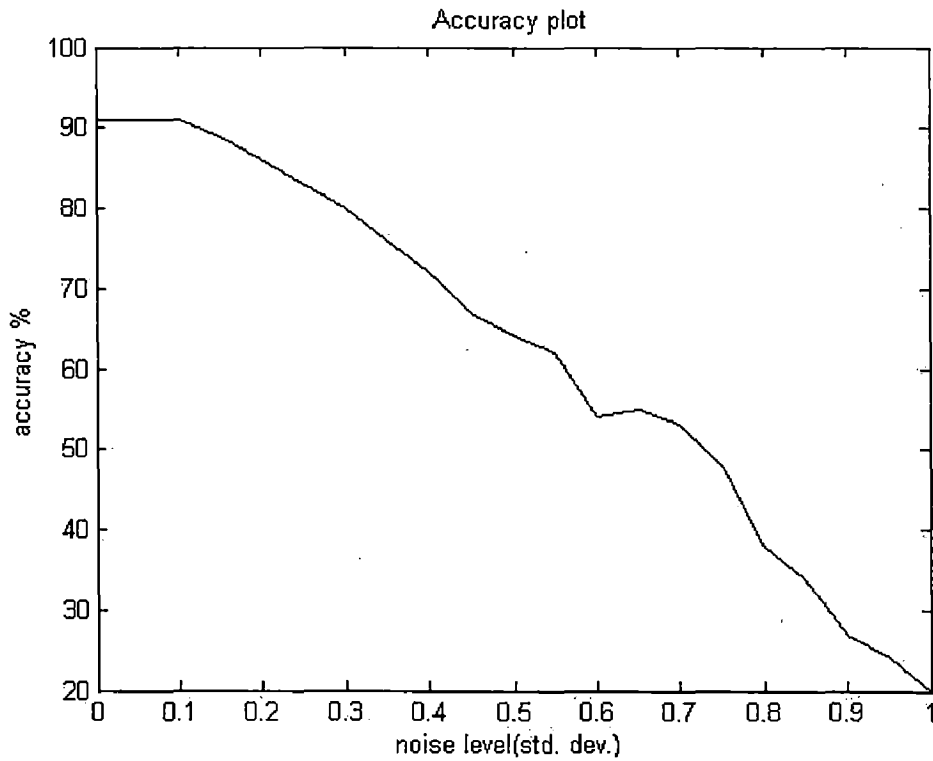


figure 6.4 Worst case Accuracy Plot

Chapter 7: Conclusions and Future Research Suggestions

In this dissertation an algorithm for face recognition using Feed Forward Neural Network with faster learning and improved noise tolerance has been developed. Faces were recognized using Neural Networks under noisy conditions and with variation in looks of a given Face. Reliability of the method was tested against Cambridge ORL face database. Ten different Faces with ten different expressions each were picked up. The proposed method was found to recognize the Faces with high recognition accuracy under noisy conditions.

It can be concluded that Neural Networks perform extremely well for computations like recognition and classification. Singular Value Decomposition is efficient method for redundancy removal (Feature generation or Dimensionality reduction). Neural Network designed with proposed method tolerated the noise extremely well and training algorithm used led to its faster learning. This approach does provide a practical solution to facial recognition with a simple algorithm under noisy condition. Resilient Backpropagation led to faster convergence. Further, with worst case performance plot shown in figure 6.4, one can conclude that noise tolerance of the Network will be at least as good as shown in the plot.

Area of Face Recognition has immense research potential and developing at a fast rate. Its scope for development lies in more advanced techniques of feature generation and dimensionality reduction. In some latest papers it has been proposed that techniques like wavelet packet decomposition and 3D Face Recognition can take the accuracy level close to 100%. Area of Neural Network has to be understood more clearly to harness its immense computational potential over conventional computing. Advanced training methods can be applied for faster learning. Reliable methods of face recognition can boost various sectors like automated surveillance, industrial automation, security systems, robotics etc.

References

- [1] W. Zhao, "Face Recognition: A Literature Survey", ACM Computing Surveys, pp. 399- 458, 2003.
- [2] Pritha, D.N., Savitha, L., Shylaja, S.S., "*Face Recognition by Feed forward Neural Network Using Laplacian of Gaussian Filter and Singular Value Decomposition*" 2010 First International Conference on Integrated Intelligent Computing (ICIIC), vol.,no.,pp.56-61,5-7Aug.2010.
- [3] Guang Deng , "*A Generalized Unsharp Masking Algorithm,*" , IEEE Transactions on Image Processing, vol.20May2007.
- [5] Fukunaga, K. Olsen, D.R., "*An Algorithm for Finding Intrinsic Dimensionality of Data,*", IEEE Transactions on Computers , vol.C- 20, no.2, pp.176-183,Feb.1971.
- [6] Jun-Ying Gan, Mengfei Liu , "*Face recognition using wavelet Packets decomposition and Hopfield neural network,*", ICWAPR 2009. International Conference on Wavelet Analysis and Pattern Recognition,vol.,no.,pp.335-339,12-15July2009.
- [7] Palanivel, S, Venkatesh, B.S, Yegnanarayana, B., "*Real time face recognition system using autoassociative neural network models,*" (ICASSP '03). IEEE International Conference on Acoustics, Speech, and Signal Processing, vol.2,no.,pp.II-833-6.2,6-10April2003.
- [8] Howard Demuth and Mark Beale, *Neural Network Toolbox™ User's Guide* . version 6 Natick :The MathWorks, Inc,2008
- [9] Sergios Theodoridis and Konstantinos Koutroumbas, *Pattern Recognition*, Fourth Edition, Burlington: Academic Press.2008.
- [10] J. M. Zuradah , *Introduction to Artificial Neural System*, Third Edition. Mumbai: Jaico Publishing House 1999.
- [11] R. C. Gonzalez and R. E. Woods, *Digital image processing*, Third Edition. New Delhi: Dorling Kindersley India Pvt. Ltd 2008.
- [12] Sergios Theodoridis and Konstantinos Koutroumbas, *An Introduction to Pattern Recognition: A Matlab approach*, Fourth Edition, Burlington: Academic Press.2008.