M. Tech (C.S) Dissertation Series

Connectionist Schemes for Image Segmentation

A dissertation submitted towards partial fulfilment of the requirements for the **M.Tech. (Computer Science)** degree of Indian Statistical Institute

By

Tapas Chakraborty

Under the supervision of

Prof. N.R Pal and Dr. S. Pal

INDIAN STATISTICAL INSTITUTE

203, Barrackpore Trunk Road

Calcutta-700035

Certificate of Approval

This is to certify that the thesis entitled *Connectionist Schemes for Image Segmentation* submitted by *Tapas Chakraborty*, towards partial fulfilment of the requirement for *M.Tech.* in *Computer Science* degree of the *Indian Statistical Institute, Calcutta*, is an acceptable work for the award of the degree.

Date : July 18, 2000

19/7/2000
(Supervisors)

19.7.2000
(External Examiner)

Acknowledgement

My sincerest gratitude goes to *Prof. N.R. Pal* and *Dr. S. Pal* for their guidance, advice, enthusiasm and criticisms throughout the course of this dissertation

I would also like to take this oppurtunity to thank *Dr. B. Chanda, Dr. P. Sarkar, Dr. D.P. Mukherjee, Dr. B.K. Roy and Dr M. Mitra* for their excellent courses they have offered which helped me in this work.

Finally I would like to thank Mr. Debrup Chakraborty, JRF, Mr. Anil Ghosh, JRF and all my class mates, without whose co-operation and support this work would not have been a success.

Tapas Chakraborty

Connectionist Schemes for Image Segmentation

Tapas Chakraborty

July 18, 2000

Chapter 1

Introduction

The technique of extracting information from an image is referred as image analysis. Generally, the first step of image analysis is to segment the image into its constituent parts or objects. Autonomous segmentation is one of the most difficult tasks in image processing. Fu and Mui [7] categorized segmentation techniques into three classes. (1) feature based clustering, (2) edge detection and (3) region extraction. Segmentation is a process of partitioning the image into some non intersecting regions such that each region is nearly homogeneous. Formally it can be defined [8] as follows : if F is the set of all pixels and $p(\ )$ is the uniformity (homogeneity) predicate defined on groups of connected pixels, then the segmentation is a partitioning of the set of connected subsets or regions $(s_1 \cdots s_n)$ such that

$$\cup_{i=1}^{n} s_i = F$$

with

$$s_i \cap s_j = \phi, \quad i \neq j$$

The uniformity predicate $p(s_i)$ = true for all regions s_i and $p(s_i \cap s_j)$ is false, when s_i is adjacent to s_j. A large number of segmentation techniques are present in the literature, but there is no single method which can be considered good for all images, also all methods are not equally good for a particular type of image. There are many challenging issues like, the development of a unified approach to image segmentation which can (probably) be applied to all kinds of images. Till now, there is no universally accepted method of quantification of segmented output. Authentication of edges is also a very important task. Different edge operators like Sobel, Prewitt, Mar-Hildreth, etc. compute a numerical value using its edge operator(s) at every pixel location to indicate whether an edge is present or not at that location. Here this numerical value may be termed as a measure of edginess at that pixel location. However, all of them are not valid candidates for edges. Normally a threshold needs to be applied on the edginess to get the edge pixels. The selection of a suitable threshold is very important and a difficult task. It is crucial because for some part of the image, low intensity variation may correspond to edges of interest, while the other part may require the high intensity variation. Adaptive thresholding [9][10] [11] often is taken as a solution of this. Obviously it cannot eliminate the problem of threshold selection. A good strategy to produce meaningful segments would be to fuse region segmentation results and edge outputs.

One may attempt to extract the segments in a variety of ways. Broadly, there are two approaches namely, classical approach and fuzzy mathematical approach. Under the classical approach we have segmentation techniques based on histogram thresholding, edge detection, relaxation, and semantic and syntactic approaches. In addition to these there are some methods

which do not fall clearly in any one of the above classes. Similarly, the fuzzy mathematical approach also has methods based on edge detection, thresholding, and relaxation. Some of these methods, particularly the histogram based methods are not all suitable for noisy images. Several attempts have also been made to develop image processing algorithm using Artificial Neural Network (ANN) models, particularly Hopfiled and Kohonen Neural Network models. These algorithms work well even in a highly noisy environment and they are capable of producing outputs in real time applications. Since the proposed methods are neural net based we provide a brief summary of some NN based segmentation techniques.

1.1 Neural network based approaches

In an artificial vision system, one desires to achieve robustness of the system with respect to random noise and failure of processors. Moreover, a system can probably be made artificially intelligent if it is able to emulate some aspect of human information processing system. Another requirement is to produce outputs for real time applications. Neural network based approaches are attempts to achieve these goals. Blanz and Gish [12] use a three-layer feed forward neural network for image segmentation, where the number of nodes in the input layer depends on the number of input features and number of nodes in the output layer is equal to the number of classes. Babaguchi et al. [13] used a multilayer network trained with back propagation for thresholding an image. The input of the network is the histogram of an image and the output of the network is the desirable threshold. At the time of learning, this method needs a large set of sample images with known thresholds which produce visually suitable outputs. But for practical applications it is very difficult to get many sample images. Shah [14] formulated the problem of edge detection in the context of edge minimizing model. The method is capable of eliminating weak boundaries and small regions. Cortes and Hertz [15] proposed a neural network based model to identify potential edges in different orientation. The performance of the system has been investigated through simulation studies using simulated annealing and mean field annealing. Srinivasan et al. [3] proposed a two stage encoder-detector network model for edge detection. The single layer encoder stage, trained in a competitive mode, compresses data from an input receptive field and drives a back-propagation-trained detector network whose two outputs represent components of an edge vector. In case of step edges of a noisy image, the experimental results show that the performance of the neural edge detector is comparable to that of the Canny [1] edge detector.

1.1.1 Proposed Work

In this thesis we report two connectionist schemes for two different types of problems in image segmentation. The first scheme deals with edge detection. There are many methods of edge detection. Here we have chosen two widely used edge detection operators namely Canny's operator and Mar-Hildreth's operator. Neural networks are well known for their property of universal approximation of arbitrary nonlinear functions and subsequent generalization. These properties of a neural network has been adequately exploited in this scheme.

The next scheme deals with segmentation of "homogenious" textured regions. Texture classification has also been attempted in numerous ways. Gabor filters are known to detect features appropriate for the texture classification. Texture classification with the use of Gabor filters has been reported to be successful in the literature [6][5]. But a crucial problem with these methods involving Gabor filters lies in the selection of the filters. We have tried to address this problem to a certain extent. In our scheme, a feedforward architecture extracts the features in the image using Gabor filters and classifies the image. Moreover, it learns the optimal parameters of the

filters. Our study involving this scheme is not yet complete.

The thesis is organized as following. Chapter 2 deals with a brief introduction to neural networks, the detailed learning methods for a feedforward type of network is also discussed there. Chapter 3 deals with our edge detection schemes. It contains description of some important edge detection schemes along with the details of our scheme and the simulation results. In Chapter 4 we discuss the methodology and results of our of texture segmentation scheme. Finally the thesis is concluded in Chapter 5.

Chapter 2

Introduction to Neural Networks

This chapter gives a brief introduction to neural networks in general and the training scheme of a multilayered perceptron in particular.

2.1 Neural Networks

The concept of artificial neural networks (NN) has been inspired by biological neural networks, but the heart of this emerging technology is rooted in different disciplines. Biological neurons are believed to be the structural constituents of the brain and they are much slower than silicon logic gates. But inferencing in biological NN is faster than the fastest computer available today. Brain compensates for the relatively slower operation by a really large number of neurons with massive interconnections between them. Biological Neural Networks enjoy the following characteristics:

- It is a nonlinear device highly parallel, robust and fault tolerant.
- It has a built-in capability to adapt its synaptic weights to changes in the surrounding environment.
- It can handle easily imprecise, fuzzy, noisy and probabilistic information.
- It can generalize from known tasks or examples to unknown ones.

Artificial neural network (in short NN) is an attempt to mimic some or all of these characteristics [16], [17], [18]. This NN paradigm is different from programmed instruction sequence. Here information is stored in the synaptic connections. A neuron is an elementary processor with primitive types of operations, like summing the weighted inputs coming to it and then amplifying or thresholding the sum. The computational neuron model proposed by Mc-Culloch-Pitts is a simple binary threshold unit. The i th neuron computes the weighted sum of all its inputs from other units and outputs a binary value, zero or one, depending on whether this weighted sum is greater than equal or less than a threshold θ_i.

$$\begin{array}{lrcl} \text{Thus} & y_i & = & f(\sum_i w_{ij}\, x_j - \theta_i) \\ \text{where} & f(z) & = & 1 \quad \text{if} \quad z \geq 0 \\ & & = & 0 \quad \textit{otherwise.} \end{array}$$

If the synaptic weight $w_{ij} > 0$, then it is called an excitatory connection; if $w_{ij} < 0$, it is viewed as an inhibitory connection. A simple generalization of Mc-Culloch-Pitts neuron by replacing the threshold function f with a more general non-linear function enhances the power of the networks

built from such neurons. Even a synchronous assembly of McCulloch-Pitts neurons is capable, in principle, of universal computation for suitably chosen weights [16]. Such an assembly can perform any computation that an ordinary digital computer can. Neural networks are naturally parallel computing devices. Although the development of neural networks is inspired by models of brains, the purpose is not just to mimic biological neuron, but to use principles from nervous systems to solve complex problem in an efficient manner.

A neural network is characterized by the network topology, connection strength between pairs of neurons (weights), node characteristics and the status updating rules. The updating or learning rules may be for weights and/or states of the processing elements (neurons). Normally an objective function is defined which represents the complete status of the network, and its set of minima corresponds to different stable states of the network. The adaptability of a neural network comes from its capability of learning from "environments".

2.2 Training a Multilayered Perceptron

The Multilayered Perceptron (MLP), is the most popular of the feedforward type neural networks. Both of our schemes use this type of architecture. Hence for the sake of completeness we describe the structure and the training equations of an MLP here.

Figure 2.1: A Conventional MLP

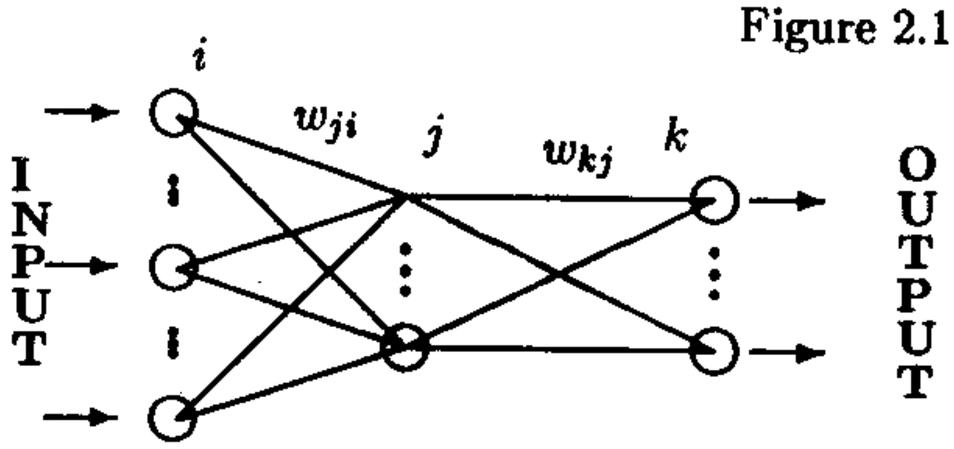

Figure 2.1. shows a conventional MLP network. The first layer is the input layer and the last layer is output layer. All the remaining layers are the hidden layers. Each node in a hidden layer is connected to all the nodes of its immediately following and preceding layers via weights. At each node, all incoming signals are summed and transformed by a non-linear activation function to give the output of the node.

We use the following symbols,

- x_i : i^{th} component of an input vector $\mathbf{x}$ in the training set.
- y_i : i^{th} component of the target vector $\mathbf{y}$ in the training set.
- f : activation function.
- n : No. of hidden layers.
- o_i^k : Output of the i^{th} node in k^{th} hidden layer; $o_i^0 = x_i$ and $o_i^{n+1} = y_i$.
- $f_i'^k$: Derivative of the activation function of i^{th} node in the k^{th} hidden layer.
- w_{ij}^k : Weight connecting the j^{th} node in the k^{th} hidden layer to i^{th} node in the $k+1^{th}$ hidden layer.

- ϵ_x : Error rate corresponding to the pair $(\mathbf{x}, \mathbf{y})$.

- η : learning rate or step size.

The back propagation algorithm comprises a forward and a backward pass, the latter being the weight adjustment pass. The forward
pass is given by,

$$o_i^k = \begin{array}{l} f\left(\sum_j o_j^{k-1} w_{ij}^k\right) \quad when, \quad k = 1, ..., n+1 \\ x_i \quad when, \quad k = 0. \end{array} \tag{2.1}$$

In the backward pass, weights are adapted with a view to minimizing $\sum \epsilon_x$ over the entire data set S, using gradient decent on each ϵ_x in the training set where,

$$\epsilon_x = \frac{1}{2} \sum_i \left(y_i - o_i^{n+1}\right)^2 \tag{2.2}$$

Following gradient decent technique we obtain the expression for the incremental changes as
:

$$\begin{array}{lcl} \Delta w_{ij}^k & = & \eta \delta_i^{k+1} o_i^k \\ where, & & \\ \delta_i^k & = & \begin{array}{l} f_i'^k \sum_j \left(\delta_j^{k+1} w_{ji}^k\right), when \quad k \neq n+1 \\ \left(y_i - o_i^{n+1}\right) f_i'^k, \quad when \quad k = n+1. \end{array} \end{array} \tag{2.3}$$

The incremental changes Δw_{ij}^k may be summed up over the entire data set S and w_{ij}^ks updating by the resulting sums or the updates may be done separately for each pattern in S. The former method is called the *batch* method while the latter is called the *on-line* method of training . One update cycle using the entire data set is known as epoch. The training is continued for several epochs, and it is stopped when the connection weights stabilize or decrease in the error is insignificant.

Chapter 3

Edge Detection by Neural Network

As stated earlier, it has been proved that neural networks can act as universal approximators for a large class of non-linear fuctions [17]. Hence it is justified that neural networks can be used to learn the complex functional relationship between an image F an its edge image I. In this chapter we first give an overview of some of the well known edge detection techniques, then we describe our methodology along with some simulation results.

3.1 Edge Detection

Edge detection is the most common approach for detecting meaningful discontinuities in a gray level image. An edge is the boundary between two regions with relatively distinct gray level properties. In edge detection problem, we generally assume the region in question to be homogeneous so that transition between two regions can be determined on the basis of gray-level discontinuities alone. Basically, the idea underlying most edge detection techniques is the computation of a local derivative operator. The magnitude of the first derivative can be used to detect the presence of an edge in an image, and the sign of the second derivative can be used to determine whether an edge pixel lies on the dark or the light side of an image.

3.1.1 Gradient Operators

The gradient operator of an image $f(x, y)$ at location (x, y) is the vector

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.1}$$

It is well known that the gradient vector points in the direction of maximum rate of change of f at (x, y). In edge detection the magnitude of ∇f is an important quantity and is denoted by $mag(\nabla f)$

$$mag\,(\nabla f) = \left[G_x^2 + G_y^2\right]^{1/2}, \tag{3.2}$$

where $G_x = \frac{\partial f}{\partial x}$ and $G_y = \frac{\partial f}{\partial y}$. This quantity equals the maximum rate in increase of $f(x, y)$ per unit distance in the direction of ∇f. Common practice is to approximate the gradient with absolute values :

$$\nabla f \approx |G_x| + |G_y|, \tag{3.3}$$

which is much simpler to implement, particularily with dedicated hardware. The direction of the gradient vector also is an innportant quantity. Let $\alpha(x, y)$ represent the direction angle of the vector ∇f at (x, y). Then,

$$\alpha(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right). \tag{3.4}$$

The gradient image is obtained by the computatining $mag(\nabla f)$ at every pixel location. The places of gray level discontinuity may be detected from the gradient image. The gradients are sensitive to noise present in the image. Usually computation of the gradient can be done by some discrete approximation of the derivatives. Many such approximation are available in literature, egg the Sobel, Robert, Prewitt etc. As a representative we describe Sobel operator.

The *Sobel operator* is a type of gradient operator which has the advantage of providing both a differencing and smoothing effect. The smoothing effect is particularly an attractive feature of the sobel operator, as differencing enhances noise. Figure 3.1(a) shows a typical 3×3 window in an image. Considering the window in Figure 3.1(a), the derivatives based on the Sobel operator are

$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{3.5}$$

and

$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7). \tag{3.6}$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Figure 3.1 (a) : 3×3 regions

To calculate the gradient of the image by the above equations the gray level values in a 3×3 window (as in Figure 3.1(a)) are overlapped with the masks shown in Figure 3.1(b) and 3.1(c). Computation of the gradient at the location of the center of the masks is then done by Eqs. (3.5) or (3.6). To get the next value, the masks are moved to the next pixel location and the procedure is repeated. Thus, after the procedure has been completed for all posible locations, the result is a gradient image of the same size as the original image. As usual mask operations on the border of an image are implemented by using the appropriate partial neighborhoods.

-1	-2	-1
0	0	0
1	2	1

Figure 3.1 (b) : mask used to compute G_x at center point of the 3×3 region

-1	0	1
-2	0	2
-1	0	1

Table 3.1 (c) : mask used to compute G_y at that point

The other gradient operators use the same philosophy but different approximation schemes. Although Sobel operator has some smoothing effect, it is still sensitive to noise. To get better noise immunity several other approaches have been designed. We next discuss two such methods.

3.1.2 Laplacian of a Gaussian Method - Marr - Hildreth Approach

The laplacian of a 2-D function $f(x, y)$ is defined by its second-order derivative as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.7}$$

Equation (3.7) may be implemented in digital form in various ways.For a 3×3 region, the most frequently encountered implementation is

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8) \tag{3.8}$$

where the z's are been defined in Fig (3.1(a)). The basic requirement in defining the digital Laplacian is that the coefficient associated with the center pixel be positive and the coefficients with the outer pixels can be negative . Because the Laplacian is a derivative , the sum of the coefficients has to be zero. Hence the response is zero, whenever the point in question and its neighbors have the same value.

0	-1	0
-1	4	-1
0	-1	4

Figure 3.2 Mask used to compute the Laplacian.

Figure (3.2) shows a spatial mask that can be used to implement Eq(3.8). Although, as indicated earlier, the Laplacian responds to transitions in intensity, it is seldom used in practice for edge detection for several reasons. As a second order derivative, the Laplacian typically is unacceptably sensitive to noise. Moreover, the Laplacian produces double edges and is unable to detect edge direction. For these reasons, the Laplacian usually plays the secondary role of detector for establishing whether a pixel is on the dark or light side of an edge. A better use of the Laplacian is in finding the location of an edge using its zero crossing property.To reduce noise sensitivity, the image usually smoothed by a Gaussian filter and Then the zero crossing of the Laplacian is used as an edge point. More specifically we convolve an image with the Laplacian of a 2-D Gaussian function of the form

$$h(x, y) = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \tag{3.9}$$

where σ is the standard deviation(Marr and Hildreth [1980]). Let $r^2 = x^2 + y^2$. Then, from Eq (3.7) the Laplacian of h (that is second derivative of h with respect to r) is

$$\nabla^2 h = \left(\frac{r^2 - \sigma^2}{\sigma^4}\right) e^{-\frac{r^2}{2\sigma^2}} \tag{3.10}$$

The zero crossing of (3.10) are detected as edges of the image. The amount of smoothing depends on σ and hence the quantity of edges obtained seriously depends on σ.

3.1.3 Canny's Method - A Summary of Some Inportant Feature

This method also consists of two steps : a smoothing of the gray label image and a procedure for detecting the maximum gradient in the smoothed image. The optimal smoothing operator has been discussed in the literature for some time, for example, Marr-Hildreth(1980) and Canny

(1983). Both authors came to the conclusion that the Gaussian average operator is near optimal. Canny used a derivation of optimality which is linked with the concept of an edge. The search of the maximum gradient is done differently in Canny(1983) and Marr-Hildreth[7]. Canny used the derivative in the gradient direction whereas Marr used the zero crossings of a 2nd derivative (Laplace). Canny proved that his procedure is more efficient with respect to localization. Marr also dicussed the possibility of using the derivative in the gradient direction but used Laplacian -operator to simplfy the algorithm. Canny's operator tends to "push away" noise (insignificant edges) around a strong edge. Canny's edge dector in-general consists of two steps namely : smoothing and finding the maxima of the first derivative

Let the smoothed image be denoted by $g(x, y)$:

$$g(x,y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} f(x+i, y+j)h(i,j),$$

where $f(x, y)$ is the original image and $h(i, j)$'s are weights.The $h(i, j)$ has the same form of (3.9) with (x, y) replaced by (i, j)

The strength of the smoothing filter is apparently determined by the σ parameter - the bigger σ the more smothing.For the 2nd step, edge detection, we need the first derivatives in the X and Y -directions. Instead of using the differences $g(x+1, y) - g(x, y)$ and $g(x, y+1) - g(x, y)$ one can use

$$g_x = \sum_{i=-n}^{n} \sum_{j=-n}^{n} f(x+i, y+j)h_x(i,j)$$

and

$$g_y = \sum_{i=-n}^{n} \sum_{j=-n}^{n} f(x+i, y+j)h_y(i,j)$$

where h_x is the derivative of $h(x, y)$ with respect to x and h_y is the derivative of $h(x, y)$ with respect to y. The size of the filter should be linked with the strength of the filter ($= \sigma$). For higher σ we get higher values farther away from the local origin (i, j) = (0, 0). Therefore, we need some rule for cutting off the series of coefficients coming from h_x and h_y. We have chosen the following rule:
n = the size of the filter ((2n+1) $\times$(2n+1) matrix)
$n = 2.5\sigma$, $\sigma > 1$ (n is rounded to integer)
$n = 3$, $\sigma < 1$

This rule ensures that if $|h_x(i, j)|$ or $|h_y(i, j)| > 0.01$ then these values are always included.

3.1.4 Non Maximal Supression

The 2nd step of edge detection algorithm involves locating the edges of the smoothed image. The position where the slope, - the derivative, of the gray level function $f(x, y)$ has its maximum is declared as the location of the edge point. To determine the steepest slope, not in the X or Y direction but for a curve running in gradient direction, we maximize the first derivative in the gradient direction. Due to the risk of getting many insignificant edges it is necessary to look the edges in smoothed image $g(x, y)$ instead of $f(x, y)$. So we estimate of the gradient direction and gradient slope in the smoothed image $g(x, y)$ with g_x and g_y being the (approximate) difference quotient of g in the X and Y direction. The gradient direction is $(g_x g_y)$ and the gradient slope

gradient magnitude is equal to $\sqrt{g_x^2 + g_y^2}$. In principle, for every pixel, it is necessary to find the maximum gradient direction on the gradient surface. To reduce computational overhead one can check for the maximum in the local gradient directions in a number of points in the grid. For each point the gradient slope in (x, y) which ditto in $(x + \frac{gx}{gy}, y + 1)$ $(x - \frac{gx}{gy}, y - 1)$
In this fashion compare three gradient slope G, $G1$, $G2$. If $G > G1$ and $G > G2$ then a maximum is obtained and consequently (x, y) is edge point. G1 and G2 is estimated by linear interpolation between grid points.

3.1.5 Hysterisis

Canny used two threshold values $T1$ (lower threshold) and $T2$ (higher threhold). If the gradient $G < T1$ then he concluded that there is no edge point at that pixel. If gradient $G > T2$ then accept the pixel as an edge pixel. If $(T1 < G < T2)$ then accepted that pixel as an edge pixel if there is atleast edge point within 4N or 8N . Here 4N and 8N stands for 4-neighbourhood and 8-neighbourhood respectively which are defined in the following Figure.

8	4,8	8
4,8	P	4,8
8	4,8	8

In the above Figure 8 denotes the 8N and 4 denotes the 4N of P.

3.2 The Network Structure

Here we propose a 2-stage feed forward multilayered network very similar to the conventional multilayer perceptron(MLP) accomplish the task of edge detection. The network is 4 layered as shown in Figure (3.1). The first layer is called the input layer.

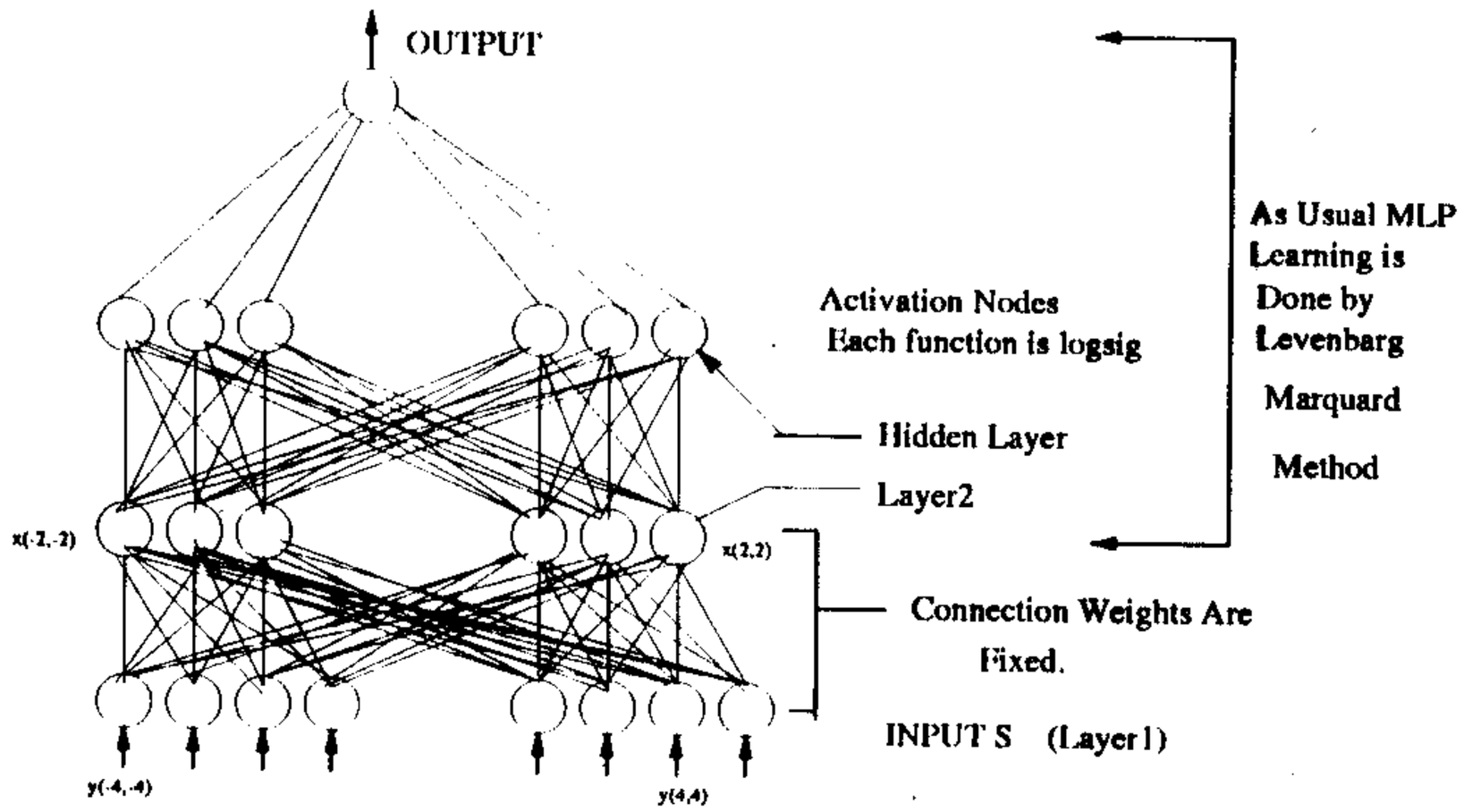

Figure 3.1: Multi layer perceptron network.

Edge is a local property of an image. Let us assume that the edge at pixel (x, y) is determined by a window of $s \times s$ around (x, y) and for the smoothing (or feature extraction) suppose the pixels in a window of size $p \times p$ around (x, y) are enough. So for the smoothing of all pixels in a

window of size $s \times s$ about (x, y) a window of size $(s+p-1)$ around (x, y) is needed. The layer 1 (input layer) of our network will have $(s+p-1) \times (s+p-1)$ neurons and layer 2 will have $s \times s$ neurons, If s = 5 and p = 5 then layer 1 will have $9 \times 9 = 81$ neurons and layer 2 will have 25 neurons. In our simulation we use $p = 5$ and $s = 5$. In the first layer we take 81 nodes and in the 2nd layer we take 25 nodes. We denote the nodes in the first layer $y_{(-4,-4)}, \ldots\ldots, y_{(4,4)}$ and in the second layer nodes are denoted by $x_{(-2,-2)}, \ldots\ldots, x_{(2,2)}$. We first smoothed the image to extract the feature values by considering a 5×5 window around each pixel. The connection weights between first two layers are determined by $w_{(i_1,i_2),(j_1,j_2)}$ where $w_{(i_1,i_2),(j_1,j_2)}$ = weight between $x_{(i_1,i_2)}$ and $y_{(j_1,j_2)}$, $-2 \leq i_1, i_2 \leq 2$ and $-4 \leq j_1, j_2 \leq 4$;

$$\begin{aligned} w_{(i_1,i_2),(j_1,j_2)} &= 0 \quad if \ \ |i_1 - i_2| > 2 \quad or \ \ |i_2 - j_2| > 2 \\ &= g(|i_1 - j_1|, |i_2 - j_2|) \quad otherwise. \end{aligned}$$

Here $g \sim N_2(0, 0, \sigma^2, \sigma^2)$. $N_2(.)$ stands for a bivariate normal distribution. The connection weights between the first two layers are fixed because the output of layer 2 is the feature value . Nodes in layer 2 computes the weighted sum of inputs and thereby it finds the Gaussian smoothed subimage in layer 2. The connection weights between all other layers are not predetermined, they are learnt. These weights are updated by Levenberg-Marquardt method which we describe next. In the 3rd layer we took 35 nodes and a single node in the output layer . Note that, between layer 2 and output layer there could be several hidden layers. The output node determines whether an edge is present or not. All nodes, except nodes in layer 1 and laye 2 use sigmoidal activation function.

3.3 The Training Scheme

Generally the feed forward networks are trained by the backpropagation(BP) algorithm. The backpropagation algorithm through widely used has certain problems. The problem of getting stuck at a local minima is often encountered while using backpropagation algorithm. The choice of the learning coefficient used in the BP algorithm is very important and usually it takes too many epochs for convergence. Here we have used a less popular scheme of gradient descent type optimization procedure called Levenbarg and Marquardt method. This algorithm is computationally more complex than the BP algorithm. But the number of iterations required for convergence is generally much lower than that of the BP algorithm. Moreover, the problem of local minima is not as serious as in the BP algorithm. Next we discuss the details of Levenbarg-Marquardt method.

3.3.1 Levenbarg-Marquardt Method

To discuss the method we pose the problem in a simple way as : find θ to minimize

$$\begin{aligned} s(\theta) &= \sum_{i=1}^{n} (y_i - f_i(\theta))^2 \qquad (3.11) \\ &= \|\mathbf{y} - \mathbf{f}\|^2. \end{aligned}$$

Where y_i is the target value. $f_i(\theta)$ is the functional output, θ is the vector of parameters to be identified.

Linear Taylor series approximation to $f_i(\theta)$ about $\theta^{(a)}$ obtains

$$f_i(\theta) = f_i\left(\theta^{(a)}\right) + \frac{\partial f_i\left(\theta^{(a)}\right)}{\partial \theta'}\left(\theta - \theta^{(a)}\right). \tag{3.12}$$

Using the approximation (3.12) in (3.11) the minimization problem can be converted to a linear least square problem

$$minimize \qquad \|\mathbf{r}^{(a)} - \mathbf{F}^{(a)}\left(\theta - \theta^{(a)}\right)\| \tag{3.13}$$

where

$$\mathbf{r}^{(a)} = \mathbf{y} - \mathbf{f}\left(\theta^{(a)}\right)$$

and $\mathbf{F}^{(a)}$ is $\frac{\partial f}{\partial \theta'}$ evaluted at $\theta^{(a)}$. Now (3.13) has the solution

$$\theta - \theta^{(a)} = \left(F^{(a)\prime}F^{(a)}\right)^{-1} F^{(a)\prime}r^{(a)}$$

leading to the Gauss-Newton algorithm

$$\theta^{(a+1)} = \theta^{(a)} + \delta^{(a)}$$

where

$$\delta^{(a)} = \left(F^{(a)\prime}F^{(a)}\right)^{-1} F^{(a)\prime}r^{(a)}. \tag{3.14}$$

Let θ^* be the desired optimal solution. So $s'(\theta^*) = 0$. If θ_0 be a trial value,by the Taylor series expansion we get

$$s'(\theta^*) - s'(\theta_0) = (\theta^* - \theta_0)\mathbf{s}''(\theta_0)$$

$$(\theta^* - \theta_0) = -\left[\mathbf{s}''(\theta_0)\right]^{-1}\mathbf{s}'(\theta_0)$$

Taking θ^* as $\theta^{(a+1)}$ Gauss-Newton iteration is repeated..

This $s'(\theta_0)$ is now the gradient g and $s''(\theta_0)$ is the Hessian H . the gradient $s(\theta)$ are, respectively

$$\begin{aligned} g(\theta) &= \frac{\partial s(\theta)}{\partial \theta} \\ &= 2\sum_{i=1}^{n} r_i(\theta)\frac{\partial r_i(\theta)}{\partial \theta} && (3.15) \\ &= 2J'\mathbf{r} && (3.16) \end{aligned}$$

and the Hessian is

$$\begin{aligned} H(\theta) &= \frac{\partial^2 s(\theta)}{\partial \theta \partial \theta'} \\ &= 2\sum_{i=1}^{n}\frac{\partial r_i(\theta)}{\partial \theta}\frac{\partial r_i(\theta)}{\partial \theta'} + 2\sum_{i=1}^{n} r_i(\theta)\frac{\partial^2 r_i(\theta)}{\partial \theta \partial \theta'} \\ &= 2\left(J'J + A\right), && (3.17) \end{aligned}$$

where $r_i(\theta) = y_i - f_i(\theta)$.

Here J is the matrix

$$\begin{aligned} J = J(\theta) &= [(J_{ij})] \\ &= \left[\left(\frac{\partial r_i}{\partial \theta_j}\right)\right] \\ &= \frac{\partial r}{\partial \theta'} \\ &= -F.(\theta). \end{aligned} \tag{3.18}$$

(3.19)

and

$$A = A(\theta) = \sum_{i=1}^{n} r_i(\theta) \frac{\partial^2 r_i(\theta)}{\partial \theta \partial \theta'} \tag{3.20}$$

Thus Gauss-Newton step from equation (3.14) becomes

$$\begin{aligned} \delta^{(a)} &= -H^{(a)^{-1}} g^{(a)} \\ &= -\left(J^{(a)'} J^{(a)} + A^{(a)}\right)^{-1} J^{(a)'} r^{(a)} \end{aligned} \tag{3.21}$$

since $J^{(a)} = -F^{(a)}$. We have $F^{(a)'} F^{(a)} = J^{(a)'} J^{(a)}$ and (3.20) becomes

$$\delta^{(a)} = -\left(J^{(a)} J^{(a)}\right)^{-1} J^{(a)'} r^{(a)}. \tag{3.22}$$

Using ideas due to Levenbarg(1944) and Marquart(1963), Levenbarg-Marquart algorithm enables one to use ill conditioned matrices $J^{(a)'} J^{(a)}$ by modifying the Gauss newton step (3.21) to

$$\delta^{(a)} = -\left(J^{(a)'} J^{(a)} + \eta^{(a)} D^{(a)}\right)^{-1} J^{(a)'} r^{(a)} \tag{3.23}$$

Here $D^{(a)}$ is a diagonal matrix with positive diagonals elements. Often for simplicity $D^{(a)} = I_p$, the identity matrix.

When $D^{(a)} = I_p$, the Levenbarg-Marquardt direction of $\delta^{(a)}$ in (3.22) interpolates between the Gauss -Newton direction ($\eta^{(a)} \to 0$) and steepest desent direction ($\eta^{(a)} \to \infty$) Also, as the direction tends to the steepest desent,the step length $\|\delta^{(a)}\|^2$ tends to zero.It is Known that a direction **d** is descent direction at θ iff there exits a positive definite matrix R such that $d = -Rg$ [4]. For $\eta^{(a)} > 0$. $J^{(a)'} J^{(a)} + \eta^{(a)} D^{(a)}$ is positive definite, as $D^{(a)}$ is positive definite. Hence $\delta^{(a)}$ (3.12) in defines a descent direction. We note that as $\eta^{(a)} \to \infty$. the step the length $\delta^{(a)}$ tends to zero. Thus by choosing $\eta^{(a)}$ large enough we can reduce $s(\theta) = \sum_i^n r_i(\theta)^2$. However if $\eta^{(a)}$ is too large for too many iterations the algorithm will take too many small steps and make little progress.

Levenberg-Marquardt algorithm differs in how to chose and update η^a.Initially a small positive value is taken, egg. $\eta^{(1)} = .001$. If, at the a-th iteration, the step $\delta^{(a)}$ in (3.22) reduces $s(\theta)$, then set $\theta^{(a)} = \theta^{(a)} + \delta^{(a)}$ and divides η by a factor, egg $\eta^{(a+1)} = \eta^{(a)}/10$, to push the algorithm closer to Gauss-Newton [and to bigger steps for the next iteration, by (3.22)] If within the a-th iteration the step $\delta^{(a)}$ does not reduce $s(\theta)$, they progressively increase $\eta^{(a)}$ by a factor, egg $\eta^{(a)} \to 10\eta^{(a)}$, each time recomputing $\delta^{(a)}$ until a reduction in $s(\theta)$ is achieved.

3.4 Simulation results

To train the network we need some training data. Let F be the original image. and I be the edge image obtained by some edge detetor (in this case LoG or Canny's operator). We randomly partion (F, I) into (F_{tr}, I_{tr}) and (F_{te}, I_{te}) such that $I_{tr} \cup I_{te} = I$, $I_{tr} \cap I_{te} = \phi$ and $F_{tr} \cup F_{te} = F$, $F_{tr} \cap F_{te} = \phi$, where F_{tr} and I_{tr} correspond to the same pixel sets. We use (F_{tr}, I_{tr}) for training the network and then use (F_{te}, I_{te}) to test the generalization capability of the network. The trained network can also be tested on a different image $F' \neq F$. We use the F = Lenna (Fig 3.2) image (256×256) with 256 gray level to generate the training data. The LoG operator with $\sigma = \sqrt{5}$ is applied on F, with a 3×3 mask to obtain the target image I. F (or I) has $256 \times 256 = 2^{12}$ pixels of which we randomly selected 8000 pixel (4000 edge and 4000 nonedge pixels) for training, i.e $|I_{tr}| = 8000$. The number of nodes in layer 1 is 25, 9 nodes are used layer 2 and 20 nodes in layer 3 The net is trained for 300 iterations. The trained net is tested on the entire Lenna. The network produced output is shown in Fig(3.3). Comparing Fig(3.3) with the edge image directly obtained by the LoG operator in Fig(3.5), we find that the net can learn the non-linear edge detection operation by LoG.

In our next experiment we tried to learn the Canny's edge detector. Fig 3.6 shows the original output with $\sigma = \sqrt{5}$ and window size of 5×5. Fig 3.4 shows the network produced output when the training data consists of 8000 pixels. The edges are little thicker than the Canny's operator. So we enhanced I_{tr} to contain 15000 pixels (5000 edge and 10000 non-edge pixels) and trained for 250 iterations. The number of nodes in layer 1 = 81, number of nodes in layer 2 = 25, and number of nodes in the hidden layer = 35. This time the result improved a lot Fig 3.7. To test the generalizing ability of the trained network we used an image of brain Fig 3.8. Fig 3.9 shows the edges of the brain image by Canny's operator and Fig(3.10) shows the edge image computed by the network.

Figure 3.2: Lenna Image

Figure 3.3: Edge detection neural learning when target was edge image by LoG operator.

Figure 3.4: Edge detection by neural learning when target was edge image by Canny operator.

Figure 3.5: Edge image of Lenna by Log operator.

Figure 3.6: Edge image of Lenna by Canny operator.

Figure 3.7: Edge image by neural learning.

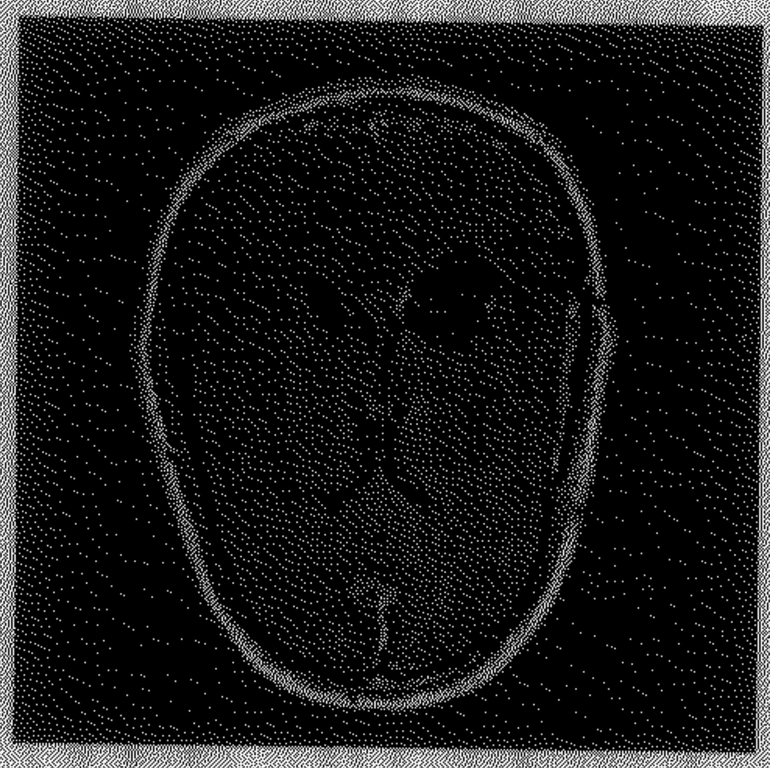

Figure 3.8: Brain Image.

Figure 3.9: Edge image of Fig(3.8) by Canny operator.

Figure 3.10: Edge image of Fig(3.8) by neural learning.

Chapter 4

A Neural Network based Method for Texture Segmentation

4.1 Introduction

Inspite of the importance of texture and its presence in many real and synthetic images, it is hard to find a reasonable quantitative definition for texture that is general enough. One definition for texture describes it as repetition of some primitive patterns in space. But such a definition may be appropriate only while referring to deterministic patterns. In contrast, man made and natural images (for example clouds, ocean surface, etc.) posses stochastic structure. Furthermore, it must be noted that the interval between repetitions of the primitive patterns need not be constant and can be space variant in one or more directions. Thus diversity of natural and artificial texture makes it very difficult to give a universal definition of texture. We prefer to adopt the definition suggested by Sklansky(1979), because of its generality: *"a region in an image has a constant texture if a set of local statistics or other local properties of the picture are constant, slowly varying, or approximately periodic"*.It must, however be noted that texture has both local as well as global meanings - it is characterized by invariance of certain local attributes of the texture that is used in the identification of a texture type and its global property. In image segmentation using textures , 2D Gabor filters are being widely used for extraction of local texture information. The merit of employing Gabor filter is that it provides maximum spatial resolution in characterization, while keeping the filters as narrowband as possible for discrimination between the spectrally neighbouring features of different textures. It is possible to use a bank of Gabor filters so as to represent a texture with feature vectors whose elements are the amplitudes of each filter response. This approach is known as multichannel filtering. Selection of filter for efficient characterization of the textures in an image is one of the major issues in multichannel filtering. Selection of set of Gabor filters bands for efficient characterization of the textures within the image is one of the major issues in Multichanel filtering. Bovik et al. have used one Gabor filter per texture class, each tuned to peak frequency [5]. Raghu and Yagnarayana [6] proposed a texture classification framework based on two stage neural network model comprising Self Organizing Map (SOM) and Multilayer Perceptron (MLP) for the problem of texture classification . The texture features are extracted using a multichannel approach. The chanels comprise a set Gabor filters having different orientations and frequencies to constitue N dimensional feature vectors. SOM acts a clustering mechanism to map these N -dimensional feature vector into its M dimensional output space, where they used $M = 2$. This in turn, forms the features space are then fed into an MLP for training and subsequent classification.

In this chapter we introduce a neural network model which integrates the convolution filter

and classifier in one. The network also learns the filter parameters along with the rules for classification, online. Thus our strategy involves finding the correct global filters which will produce the appropriate features required for the classification task along with the classification itself. In the subsequent sections we discuss about the network architecture and the training schemes.

4.2 Network Architecture

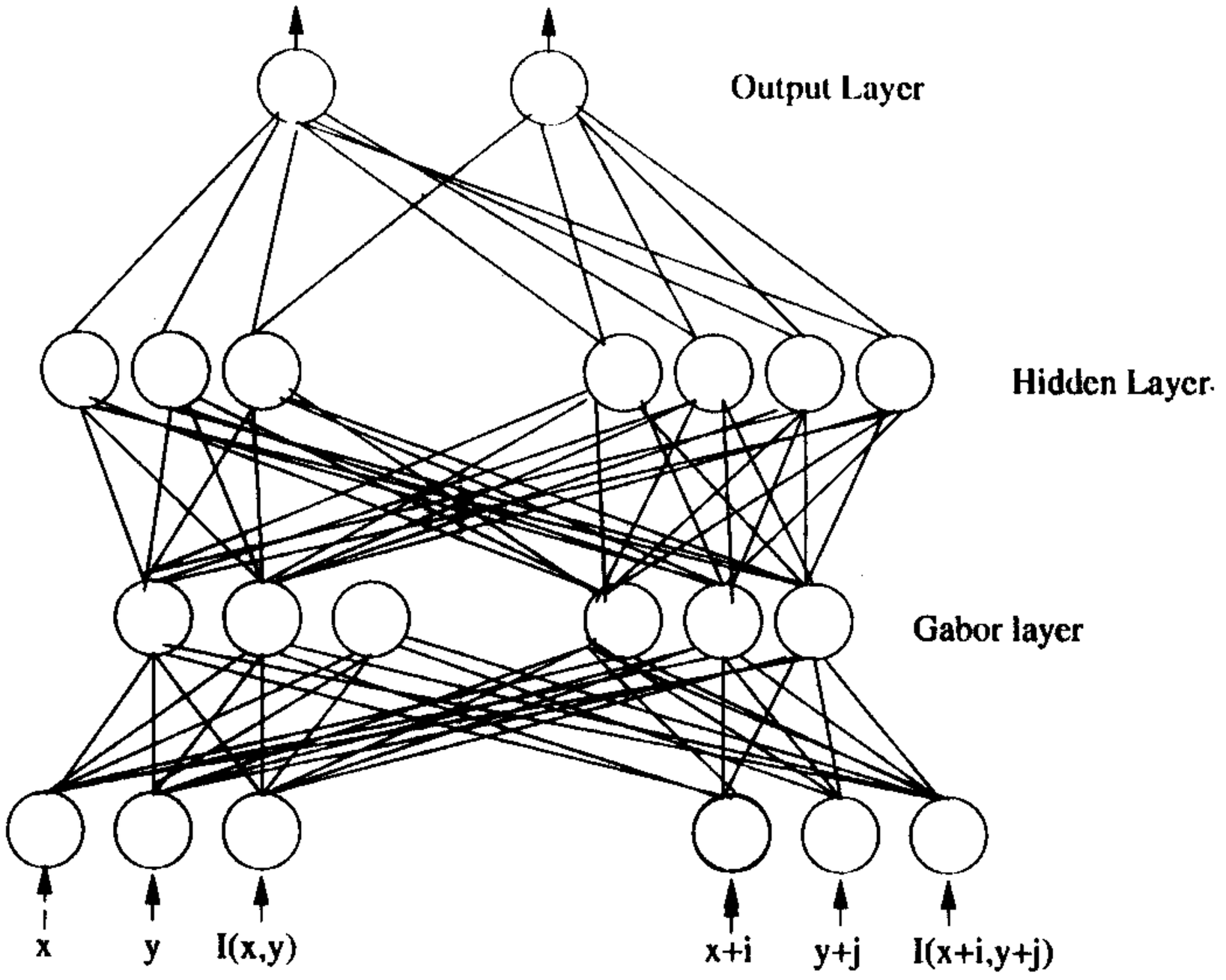

Figure 4.1: Neural Network.

The architecture of the proposed network is illustrated in Fig 4.1.The network has layered architecture of a feed forward type. The first layer is the input layer, the second layer is called Gabor layer. The General form of 2-D Gabor function is given by

$$h\left(x,y,\sigma_x,\sigma_y,k_x,k_y\right)=\frac{1}{2\times\pi}e^{\frac{-1}{2}\left[\left(\frac{x}{\sigma_x}\right)^2+\left(\frac{y}{\sigma_y}\right)^2\right]}e^{jk_xx+jk_yy}$$

The spatial extent of the Gabor function is defined by (σ_x,σ_y). The orientation of the filter is given by $\tan^{-1}\left(\frac{k_y}{k_x}\right)$ and its frequency is specified along the x and y co-ordinates by k_x and k_y respectively. If $I(x,y)$ is the image then the feature value at (x,y) is given by

$$\begin{aligned} f(x,y) &= |I(x,y)*h(x,y)|^2 \qquad (4.1)\\ &= \left[\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}I(x+\alpha,y+\beta)h(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)\right]^2 \end{aligned}$$

for a filter $g(x, y)$ with given $k_x, k_y, \sigma_x, \sigma_y$, where * denotes the convolution operator and d defines a window of size $2d+1$. This $f(x, y)$ is the output of the Gabor layer. Thus each processing unit in the Gabor layer acts as a Gabor filter with parameters σ_x, σ_y, k_x, k_y. Thus at end of successful training we obtain the set of optimal Gabor filters which can do the classification task. The subsequent network layers are called hidden layers and output layer. The units in these two layers use sigmoidal function, taking the weighted sum of the outputs of the units in the preceding layer as inputs. Thus layers two and three are identical to that of an conventional MLP. We start with a fixed number of nodes in the Gabor layer(i.e., we start with a fixed number of Gabor filters). In the course of training the parameters take optimal values as required for the classification task.

4.3 Training

Let Y_h (h = 1, $\cdots$, n) be the target output for the h-th node and Z_h be the computed output of node h in the output layer; where

$$Z_h = f^o\left(\sum_{i=1}^{k} s_{ih}\right).$$

The instantaneous training error is defined as

$$L = \sum_{h=1}^{n} [Y_h - Z_h]^2$$

Here $s_{ih} = w'_{ih} f(x_{ih})$, f^o is sigmoidal function i,e.

$$f^o(x) = \frac{1}{1+e^{-x}},$$

$$x_{ih} = \sum_{j=1}^{t} w_{ji} f_j \qquad i = 1(1)k$$

where

$$f_i = |g_i(x, y) * h_c(x, y)|^2 + |g_i(x, y) * h_s(x, y)|^2 \qquad i = 1(1)t;$$

$$h_c(x, y, \sigma_x, \sigma_y, k_x, k_y) = e^{\frac{-1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} cos(k_x x + k_y y);$$

$$h_s(x, y, \sigma_x, \sigma_y, k_x, k_y) = e^{\frac{-1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)} sin(k_x x + k_y y);$$

and

$$f(x_{ih}) = \frac{1}{1+e^{-x_{ih}}} \qquad i = 1(1)k.$$

The connection weights between hidden layer and output layer is denoted by w'_{ih} and the connection weights between Gabor layer and hidden layer is denoted by w_{ji} These connection weights and the filter parameters are updated using gradient desent. To write the updation law explicitly,

$$w'_{ih} = w'_{ih} - \eta_w \frac{\partial L}{\partial w'_{ih}}$$

where

$$\frac{\partial L}{\partial w'_{ih}} = -2\left(Y_h - f^o\left(\sum_{i=1}^{k} s_{ih}\right)\right) \times \frac{e^{-s_{oh}}}{(1+e^{-s_{oh}})^2}\frac{1}{(1+e^{x_{ih}})} f_j$$

$$s_{oh} = \sum_{i=1}^{k} w_{ih} f(x_{ih}) = \sum_{i=1}^{k} w_{ih}\frac{1}{1+e^{x_{ih}}}.$$

Similarly,

$$w_{ji} = w_{ji} - \eta_w \frac{\partial L}{\partial w_{ji}}$$

where

$$\frac{\partial L}{\partial w_{ji}} = -2\sum_{h=1}^{n}\left(Y_h - f^o\left(\sum_{i=1}^{k} s_{ih}\right)\right) \times \frac{e^{-s_{oh}}}{(1+e^{-s_{oh}})^2} w'_{ih}\frac{e^{-x_{ih}}}{(1+e^{x_{ih}})^2} f_j.$$

The parameters for the Gabor filters, i.e, the parameters associated with the second layer nodes are also updated by gradient decent technique. The update equations will be as

$$\sigma_{x_j} = \sigma_{x_j} - \eta_{\sigma_x}\frac{\partial L}{\partial \sigma_{x_j}}$$

$$\sigma_{y_j} = \sigma_{y_j} - \eta_{\sigma_y}\frac{\partial L}{\partial \sigma_{y_j}}$$

$$k_{x_j} = k_{x_j} - \eta_{k_x}\frac{\partial L}{\partial k_{x_j}}$$

$$k_{y_j} = k_{y_j} - \eta_{k_y}\frac{\partial L}{\partial k_{y_j}}.$$

Here

$$\frac{\partial L}{\partial \sigma_{x_j}} = -2\sum_{h=1}^{n}\left(Y_h - f^o\left(\sum_{i=1}^{k} s_{ih}\right)\right) \times \frac{e^{-s_{oh}}}{(1+e^{-s_{oh}})^2}\sum_{i=1}^{k} w'_{ih}\frac{e^{-x_{ih}}}{(1+e^{x_{ih}})^2}\frac{\partial f_j}{\partial \sigma_{x_j}}$$

and

$$\frac{\partial f_j}{\partial \sigma_{x_j}} = 2[I(x,y) * h_c(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\frac{\alpha^2}{\sigma_x^3} I(x+\alpha, y+\beta) h_s(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)] + 2[I(x,y) * h_s(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\frac{\alpha^2}{\sigma_x^3} I(x+\alpha, y+\beta) h_c(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)].$$

Similarly,

$$\frac{\partial L}{\partial \sigma_{y_j}} = -2\sum_{h=1}^{n}\left(Y_h - f^o\left(\sum_{i=1}^{k} s_{ih}\right)\right) \times \frac{e^{-s_{oh}}}{(1+e^{-s_{oh}})^2}\sum_{i=1}^{k} w'_{ih}\frac{e^{-x_{ih}}}{(1+e^{x_{ih}})^2}\frac{\partial f_j}{\partial \sigma_{y_j}};$$

and

$\frac{\partial f}{\partial \sigma_{y_j}} = 2[I(x,y) * h_c(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\frac{\beta^2}{\sigma_y^3}I(x+\alpha,y+\beta)h_s(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)] + 2[I(x,y) * h_s(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\frac{\beta^2}{\sigma_y^3}I(x+\alpha,y+\beta)h_c(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)];$

$$\frac{\partial L}{\partial k_{x_j}} = -2\sum_{h=1}^{n}\left(Y_h - f^o\left(\sum_{i=1}^{k}s_{ih}\right)\right) \times \frac{e^{-s_{oh}}}{(1+e^{-s_{oh}})^2}\sum_{i=1}^{k}w'_{ih}\frac{e^{-x_{ih}}}{(1+e^{x_{ih}})^2}\frac{\partial f_j}{\partial k_{x_j}};$$

and

$\frac{\partial f}{\partial k_{x_j}} = -2[I(x,y) * h_c(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\alpha I(x+\alpha,y+\beta)h_s(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)] + 2[I(x,y) * h_s(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\alpha I(x+\alpha,y+\beta)h_c(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)];$

$$\frac{\partial L}{\partial k_{y_j}} = -2\sum_{h=1}^{n}\left(Y_h - f^o\left(\sum_{i=1}^{k}s_{ih}\right)\right) \times \frac{e^{-s_{oh}}}{(1+e^{-s_{oh}})^2}\sum_{i=1}^{k}w'_{ih}\frac{e^{-x_{ih}}}{(1+e^{x_{ih}})^2}\frac{\partial f_j}{\partial k_{y_j}};$$

and

$\frac{\partial f}{\partial k_{y_j}} = -2[I(x,y) * h_c(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\beta I(x+\alpha,y+\beta)h_s(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)] + 2[I(x,y) * h_s(x,y)][\sum_{\alpha=-d}^{d}\sum_{\beta=-d}^{d}\beta I(x+\alpha,y+\beta)h_c(\alpha,\beta,\sigma_x,\sigma_y,k_x,k_y)]$.

the updating of the weights and parameters are continued in an online fashion till they stabilize.

4.4 Experiments and results

We tested the proposed net to classify two classify two types of textures. The texture image used for this experiment is computer generated. The size of the image is 64×128 containing two textures. We took a sample of size 800 and try to learn. The weights and filter parameters are randomly initialized. For two texture we took 5 input nodes, 10 nodes in the hidden layer and 2 nodes in the output layer. The results, we got are not satisfactory. We noticed that the results are influenced very much by the initial choice of parameters. Random initialization of the filter parameters is not a feasible scheme for this problem. We believe a good initialization scheme will improve the results to a great extent.

Chapter 5

Conclusion

In this thesis we have tried to address the problem of image segmentation in a neural paradigm. We have attempted two different problems. The first one is of edge detection, which is central to many complicated tasks in image processing, artificial vision etc. The second one is on texture segmentation. Many real life images like satellite images, variety of medical images etc. are generally textured, hence the segmentation of such images requires proper methodology of texture segmentation.

Edge detection operators basically trasforms an image F to its edge image say I. Neural networks can be employed to learn this functional relationship which maps F to I. We have made a feedforward neural network to learn the well known edge operators of Canny and Mar-Hildreth, our results showed that the neural edge detector is quite successful in learning these operators. Though our experiments involved only two edge operators but we believe that a feedforward type network can be easily made to learn any other type of more involved operators.

Segmentation of textured images were attempted by using Gabor filters. Literature shows that the problem of selection of a proper set of Gabor filters which can extract relevant features from an image for good segmentation has not been adequately addressed. Here we proposed an online selection procedure for the parameters of the Gabor filters used. Our network generates features from the pixel values through a bank of Gabor filters, whose parameters are learned by the network along with the connection weights through training. Our study in this regard is not yet complete. We have not been able to train the network adequately so that it does a useful segmentation. We have noticed that the output of segmentation highly depends on the initialization of the filter parameters. We tried with a random initialization scheme, which failed to give good segmentation. But we believe that the methodology developed will be useful enough once a proper initialization scheme is developed for the filter parameters before the onset of learning. We are presently exploring possibilities for a good initialization scheme.

Bibliography

[1] J. Canny, A computational approach to edge detection, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **PAMI-8**(6), 679-698 (1986)

[2] D. Marr and E. Hildreth, Theory of edge detection, *Processing of the Royal society London,* **B207**, 187-217(1980)

[3] V. srinivasan, P. Bhatia and S.H Ong, Edge detection using neural network. *Pattern Recognition* **27**(12), 1653-1662(1994)

[4] Non Linear regression, *Seber and Wild*

[5] A.C Bovik, M.clark and W.S Geisler, Multichannel Texture analysis Using Localized filters *IEEE Trans* **PAMI**, vol. 12 no 1, pp 55-73, Jan 1990

[6] P.P Raghu, R Poongodi and B Yegnarayana, A Combined Neural Network Approach For Texture Classification *Neural Networks*, vol. 8 , No 6, pp 975-987, 1995

[7] K.S. Fu and J.K. Mui A survey of image segmentation, *Pattern Recognition* 13, 3-16(1981).

[8] S.L Horowitz and T. Pavlidis, Picture segmentation by directed split and merge procedure, *Proc. 2nd Int. Joint Conf Pattern Recognition* pp. 424-433(1974).

[9] C.K Chow and T.Kaneko, Automatic boundary detection of the left-ventricle from cineangiograms, *Comput. Biomed. Res.* 5, 388-410 (1972).

[10] Y. Nakagawa and A Rosenfield, some experiments on variable thresholding, *Pattern Recognition* 11 , 191-204 (1979).

[11] S.D Yanowitz and A.M. Bruckstein, A new method for image segmentation, *Comput. Vision Graphics Image Process* 46, 82-95 (1989).

[12] W.E. Blanz and S.L Gish, A Connectionist classifier architecture applied to image segmentation, *Proc. 10th ICPR*, pp. 272-277

[13] N. Babaguchi, K Yamada, K. Kise and T Tezuka, Connectionist model binarization, *Proc. 10th ICPR*, pp. 51-56(1990).

[14] J.Shah, Parameter estimation, multiscale representation and algorithms for energy-minimizating segmentation, *Proc. Int Conf. Pattern Recognition* pp. 815-819 (1990).

[15] C.Cortes and J.A Hertz, A neural network system for image segmentation, *Proc. Int Conf. Neural Network*, vol 1, pp. 121-125 (1989).

[16] J. Hertz, A. Krogh and R. G. Palmer, Introduction to the Theory of Neural Computation, *Addison-Wesley*, Redwood City, CA, 1991.

[17] S. Haykin, Neural networks-a comprehensive foundation, *Macmillan College*, Proc. Con. Inc, NY, 1994.

[18] R. Rosenfeld and J. Anderson, (Ed.) Neuro Computing, *MIT Press*, 1988.

[19] J. C. Bezdek and S. K. Pal, Fuzzy Models for Pattern Recognition, *IEEE Press*, 1992.