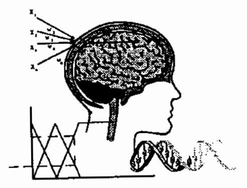
INDUSTRIAL APPLICATIONS of NEURAL NETWORKS

Edited by

Lakhmi C. Jain
V. Rao Vemuri



CRC Press
Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

Catalog record is available from the Library of Congress.

No claim to original U.S. Government works
International Standard Book Number 0-8493-9802-9
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

PREFACE

The field of artificial neural networks has come a long way. Depending on one's perspective, one can trace its origins to the research on Mach bands in visual perception and its interpretation in terms of lateral inhibition, to Hodgkin and Huxley's transmission line model of a nerve fiber, to McCulloch and Pitt's model of a neuron, to Hebb's postulate on learning, and to Rosenblatt's Perceptron. A common theme in almost all of the earlier works is the desire to build a model of the brain. The scope of this research began to widen with the publication of McCulloch and Pitt's model of a neuron. It is no accident that this model has an uncanny resemblance to the logic circuits of digital computers.

Although research in the computational capabilities of neural networks suffered a serious setback with the publication of Minsky and Pappert's book on perceptrons, the almost simultaneous development of the back propagation algorithm in the 1970s by several investigators marked the turning point in the direction of neural network research. The publication of Hopfield's paper, in 1983, in the "Proceedings of the National Academy of Sciences" (U.S.A.) almost caused a sensation in some circles. It is well over two decades now since the resurgence of neural network research. This is no longer a fringe area; it is mainstream.

This book is our way of celebrating the success of neural networks in different areas of engineering endeavor. The contributing authors come from many corners of the globe. All these chapters show how the power of neural networks can be exploited in modern engineering applications. Of the ten chapters, the first seven seem to form one group with an emphasis on image processing and industrial or manufacturing slant. Specifically, they touch on issues related to shape recognition, shape from shading, aircraft detection in SAR images, visualization of high-dimensional databases of industrial systems, 3-D object learning and recognition from multiple 2-D views, fingerprint classification and performance optimization in flexible manufacturing systems. The remaining three are applications to the communications area. Surprisingly, one of the earliest applications of neural networks was noise cancellation in telephone networks. That interest continues even today. The last three chapters included here address the issues involved in the exploding area of multimedia communications and in the area of mobile and cellular communications.

The first chapter by Ulgen, Akamatsu and Fukumi focuses on the issue of fast, on-line incremental training in the context of solving geometric shape recognition problems. It is well known that humans correctly recognize shapes independent of the variations in size, orientation and distortions caused by noise. The thesis of this chapter is that the angular difference between any two consecutive tangent vectors drawn on the borders of a shape, is important in the perceived shape of the object. The authors train a neural net to learn these angular differences and erform the required classification. The authors used some pre-rocessing to extract features to render the procedure invariant to ranslations and rotations. They also performed some heuristic fuzzy iltering to impart noise immunity prior to the submission of the raining samples to a feed-forward neural network.

he second chapter by Wei and Hirzinger deals with "shape from hading," an important problem in computer vision. The problem is to fer the surface shape of an object from a single image. This problem an be formulated either as one of solving a partial differential equation solving the minimization problem of the equivalent variational rmulation. This chapter departs from this classical approach and uses feed-forward neural network to parameterize the object surface and lves an ordinary extremum problem via a conjugate gradient method well as a radial basis function approach.

e third chapter by Filippidis, Jain and Martin is about the use of ural nets to automatic target recognition with particular emphasis on tection of aircraft from synthetic aperture radar (SAR) images. The us of this chapter is on fusion of different technologies to improve target recognition statistics. In addition to the SAR image data, the thod also uses texture data and information from the RGB bands m a color photograph and feeds this to a fuzzy "fusion" system.

Self-organizing maps (SOMs) are a powerful tool in clustering, dimensionality reduction where there is a need to preserve the topological relationships in the original data and compressed data. In Chapter 4, Simula, Vasara, Vesanto and Helminen explore its uses in visualizing higher-dimensional data, using the forest industry as a vehicle.

In the fifth chapter, Grossberg and Bradski develop a family of self-organizing neural architectures, called VIEWNET, for learning to recognize 3-D objects from sequences of their 2-D views. VIEWNET architectures use View Information Encoded With NETworks to accomplish this task. VIEWNET incorporates a preprocessor that generates a compressed but 2-D invariant representation of an image, a supervised incremental learning system (Fuzzy ARTMAP) that classifies the pre-processed representations into 2-D view categories whose outputs are combined into 3-D invariant object categories, and a working memory that makes a 3-D object prediction by accumulating evidence over time from 3-D object category nodes as multiple 2-D views are experienced. Fuzzy ARTMAP was modified to enable learning of the object classes. VIEWNET was modified to enable probability learning of object classes. VIEWNET was benchmarked on an MIT Lincoln Lab database of 128x128 2-D views of aircraft, including small frontal views, with and without additive noise. A recognition rate of up to 90% was achieved with one 2-D view and up to 98.5% correct with three 2-D views. The properties of 2-D view and 3-D object category nodes are compared with those of cells in monkey inferotemporal cortex.

Halici, Erol and Ongun, in the sixth chapter, discuss applications of hierarchical networks with particular reference to character recognition and fingerprint classification. Drawing from rich sources pertaining to pre-attentive and attentive levels of human cognition, the authors fashion a hierarchical system where the pre-attentive level is modeled by a self-organizing feature map that makes a rough decision of the eventual outcome by crudely clustering the input patterns. The attentive level is modeled by dedicating to each cluster a recognition module, which in turn is comprised of a feature extraction stage and a classifier stage. Feature extraction is performed using Principal Component Analysis and classification is performed using multilayer feed-forward networks. This method is successfully used for fingerprint classification.

Chapter 7 marks a turning point in the emphasis by leaving behind issues related to classification and focusing on optimization. Identifying the task of maximizing the throughput (i.e., the number of jobs done in a time unit) as the crux of optimization in a flexible manufacturing system, Cavalieri proceeds to formulate the problem in terms of event graphs (a special case of Petri Nets) and an integer linear programming problem. Then, instead of solving this using classical optimization procedures, the author solves the problem using a variation of the Hopfield network.

Wang and Ansari take a turn on the theme in Chapter 8 and address a problem in wireless digital communications, an exciting area of research. The channel assignment problem in mobile communication networks is known to be NP-complete. This problem is solved using mean field annealing (an idea borrowed from statistical mechanics) on an energy function of the Hopfield type.

Sheu, Wang and Young in Chapter 9 continue the theme by addressing a problem on the use of neural networks as baseband maximum likelihood sequence detectors. This technology is likely to play a progressively important role in the burgeoning field of multimedia communications. The emphasis here is on the so-called cellular compact neural networks with potential VLSI implementation possibilities.

Finally, in Chapter 10, Cavalieri and Mirabella talk about the use of Hopfield networks for process scheduling in communication systems, with an emphasis on scheduling for access to a physical channel in a computer network.

Although commendable progress has been made in the use of neural networks in a variety of application areas, the progress is not as rapid in the theoretical underpinnings of the methods. The trend in the use of hybrid methods as a way of improving performance attests to the plausible validity of this observation.

This book will be useful to researchers, practicing engineers and students who wish to develop successful applications of neural networks.

The editors are grateful to the authors for preparing such interesting and diverse chapters. We would like to express our sincere thanks to Berend Jan van der Zwaag, Ashlesha Jain, Ajita Jain and Sandhya Jain for their excellent help in the preparation of the manuscript. Thanks are due to Gerald T. Papke, Josephine Gilmore, Jane Stark, Dawn Mesa, Mimi Williams, Lourdes Franco and Suzanne Lassandro for their editorial assistance.

L.C. Jain, Australia
R. Vemuri, U.S.A.

Contents

Chapter 1:

On-Line Shape Recognition with Incremental Training Using a Neural Network with Binary Synaptic Weights

ON-LINE SHAPE RECOGNITION WITH INCREMENTAL TRAINING USING A NEURAL NETWORK WITH BINARY SYNAPTIC WEIGHTS

F. Ulgen
Motorola, Lexicus Division
3145 Porter Drive, Palo Alto, CA 94304, U.S.A.

N. Akamatsu
Department of Information Science, The University of Tokushima
2-1 Minami-Josanjimacho, Tokushima-shi 770 Japan

M. Fukumi
Department of Information Science, The University of Tokushima
2-1 Minami-Josanjimacho, Tokushima-shi 770 Japan

Recognition of hand-drawn shapes is beneficial in drawing packages and automated sketch entry in handheld computers. Fast incremental training, which is performed on-line, is accomplished by the use of the Binary Synaptic Weights algorithm, a one-pass, feed-forward neural network training algorithm. Incremental training offers the advantage of adjusting the recognition capability of the system to the user's drawings. In this chapter, we propose a new approach to on-line geometric shape recognition with incremental training which utilizes a fuzzy function for filtering angular differences and a neural network for classification and on-line training. Instead of recognizing segments of a drawing and then performing syntactical analysis to match with a predefined shape, which is weak in terms of generalization and dealing with noise, we examine the shape as a whole. The main concept of the recognition method is derived from the fact that the angular difference between any two consequent tangent vectors of a shape is important in the perceived shape of outlines. Our application's aim is to recognize elliptic, rectangular, and triangular shapes in a way similar to human cognition of these shapes. Human beings recognize such basic shapes regardless of the variations in size, noise on the shape border, rotation, and in the case of triangles, regardless of the type of the triangle. The key concept is that the neural network learns the angular difference between any two consequent tangent vectors of a shape; therefore, only a few training samples that represent the class of the shape are sufficient. The results are very successful in the sense that the neural network correctly classified shapes that did not have any resemblance to the shapes in the initial training set.

1 Introduction

Neural networks, the focus of connectionist artificial intelligence, are characterized by a large number of simple processing elements connected as a network with highly parallel and distributed control. The neural network architectures and training algorithms vary according to the problem they are applied to. Here, we propose a new training algorithm that does not suffer from the drawbacks of the Backpropagation (BP) algorithm. The Binary Synaptic Weights (BSW) algorithm and its application for geometric shape recognition is the focus of this chapter.

By using state-of-the-art MOS fabrication processes, thousands of neurons and a large number of binary synapses can be realized in a single IC chip. Through the learning procedure, the binary synaptic weights can be easily modified and this provides adaptability to a variety of applications. Using the programmable BSW neural chips reconfigurable neural systems with learning capability can be produced. Besides being easily implementable, the BSW algorithm can provide fast and guaranteed learning.

2 The Binary Synaptic Weights (BSW) Algorithm for Neural Networks

2.1 Motivation

The drawbacks of the Backpropagation algorithm drove researchers to look for new algorithms that do not suffer from the local minima trapping, offer fast training and do not require a trial and error approach to the parameters of the network topology. A number of learning algorithms utilizing binary synaptic weights has been proposed [1].

The BSW algorithm [2], which is implemented on a three layer network, determines the thresholds for the hidden and output layer nodes and the weights of the synaptic links between the layers, in addition to the required number of hidden layer nodes in one feed-forward pass. The main concept of the algorithm can be explained as the separation of globally intermingled patterns within an n-dimensional space through the formation of hyperplanes that separate different classes of patterns at a local region in the space. This in presented in Figure 1.

Each hyperplane, created to distinguish different patterns in the input space, corresponds to a hidden layer node and therefore the number of hidden layer nodes are automatically determined during training. The equation of a hyperplane in n-dimensional space is

$$\sum_{j=1}^{n} w_j i_j = c \qquad (1)$$

where w_j is interpreted as the synaptic link weight between the jth coordinate of the input vector $[i_1, i_2, \cdots, i_n]$ and a hidden layer node, and the constant c of the equation becomes the threshold of the node corresponding to the hyperplane. Therefore, in a network trained using the BSW algorithm, the synaptic link weights and thresholds of the nodes correspond to the pattern space separated by the hyperplanes.

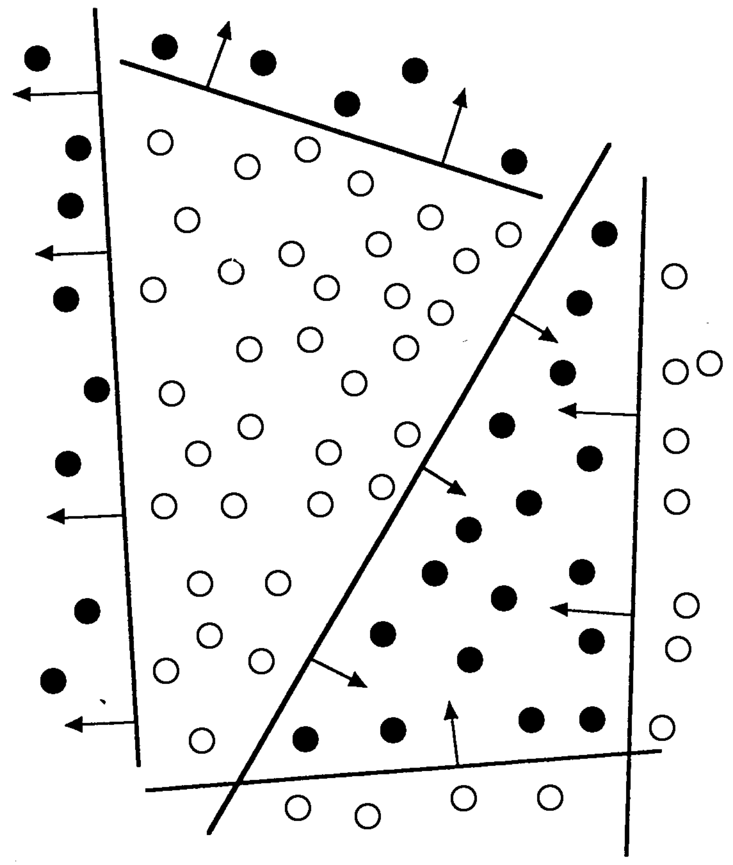


Figure 1. Separation of different patterns with multiple planes.

The activation functions of the nodes are hardlimiter thresholds, and thus a node is activated depending on whether the input pattern is on the positive or negative side of the hyperplane that represents that node.

The input and output pairs presented to the network are binary valued while the weights of the synaptic links between the input and hidden layer nodes are either 1 or and output layers, the weights of the synaptic links are either 1 or 0 depending on the activation level of the connected hidden layer nodes. If a hidden layer node is activated, the corresponding synaptic link to the output node has a weight of 1. The hidden layer node thresholds are always the constant c of equation (1). The output layer node thresholds are determined by the number of hyperplanes that enclose a pattern type. If there are P such planes, then a threshold of $(P - 0.5)$ is assigned to the output node associated with that pattern. The binary nature of the synaptic weights and the simplicity of the activation function make BSW a very promising candidate for hardware implementation.

2.2 Separation of distinct patterns by BSW algorithm

In the separation of distinct patterns using the BSW algorithm, a representative input pattern vector, corresponding to a particular class, is chosen. In this discussion we will use the pattern types ● and ○, which were separated by hyperplanes in Figure 1. The sample input pattern vector, which we will refer to as $\vec{x}^{key}$, is a vector that is approximately at the center of all the other input pattern vector in the hyperspace for the same class of output. Formally, the calculation of $\vec{x}^{key}$ is as follows: Let I be the $(m \times n)$ input matrix where m is the number of training samples and n is the input pattern vector size, and O be the $(m \times k)$ output matrix where k is the output pattern vector size:

$$I = \begin{bmatrix} i_{11} & \cdots & i_{1n} \\ & \cdots & \\ i_{m1} & \cdots & i_{mn} \end{bmatrix} \qquad O = \begin{bmatrix} O_{11} & \cdots & O_{1k} \\ & \cdots & \\ O_{m1} & \cdots & O_{mk} \end{bmatrix}$$

Consider the separation of the pattern type ● in Figure 1. Initially an average vector, $\vec{x}^{av} = \{x_1^{av}, x_2^{av}, \cdots, x_n^{av}\}$ of type ● is calculated:

$$x_q^{av} = \left(\sum_{p=1}^{m} i_{pq} \times O_{pj} \right) \Big/ \sum_{p=1}^{m} O_{pj} \tag{2}$$

where q = 1, .., n, and j represents a column of O, $1 \leq j \leq k$. If $\vec{x}^{av}$ exists among the ● patterns, it becomes $\vec{x}^{key}$. It is likely that $\vec{x}^{av}$ does not exist among the ● patterns, and if so the next step is to search for $\vec{x}^{key}$ among all ● patterns, to select the ● pattern which has the minimum Hamming distance to $\vec{x}^{av}$.

$$\vec{x}^{key} = \arg\min_{i=1}^{m} \left(HamDist\left[\vec{x}^{ave}, \vec{P}_i \right] \right) \tag{3}$$

where $P_i = [i_{i1}, i_{i2}, \cdots, i_{in}]$ is of type ● and *HamDist* represents the Hamming distance between two vectors. Again, using the Hamming distance formula, we determine $\vec{x}^{yes}$, the furthest input pattern vector with the same output classification type as $\vec{x}^{key}$, and $\vec{x}^{no}$, the nearest input pattern vector with an output classification type different than $\vec{x}^{key}$, which is of type ○ in this context.

$$\begin{aligned} \vec{x}^{yes} &= \arg\max_{l=1}^{m} \left(HamDist\left[\vec{x}^{key}, \vec{P}_l \right] \right) \\ \vec{x}^{no} &= \arg\min_{l=1}^{m} \left(HamDist\left[\vec{x}^{key}, \vec{P}_l \right] \right) \end{aligned} \tag{4}$$

If HamDist[$\vec{x}^{key}$,$\vec{x}^{yes}$] < HamDist[$\vec{x}^{key}$,$\vec{x}^{no}$], then the ● and ○ patterns are linearly separable. Otherwise we start the search for a Hamming distance D, initially zero, which will ensure that the majority of the ● type patterns will be included in the separation if we perform the separation at a distance (D + 0.5) from $\vec{x}^{key}$. In other words, at distance (D + 1) from $\vec{x}^{key}$, the number of (●) patterns is less than the number of (○) patterns. We then search for the ○ type patterns which might have been enclosed together with the ● type patterns, and perform further separation using the same procedure, this time starting with ○ type patterns as $\vec{x}^{key}$. This procedure is repeated until all of the patterns have been isolated, as is illustrated in Figure 1.

The hyperplane equation in n-dimensional space given by equation (1), assumes that $\vec{x}^{key}$ is at the origin of the coordinate system. Equation (1) takes the following form when there are components of $\vec{x}^{key}$ which are non-zero;

$$\sum_{q=1}^{n} y_q = k, \quad where \ y_q = \begin{cases} (1 - w_q i_q), iff \ x_q^{key} = 1 \\ w_q i_q, iff \ x_q^{key} = 0 \end{cases} \tag{5}$$

With reference to equation (5), the linear equation for the separation hyperplane to be formed at distance (D + 0.5) from $\vec{x}^{key}$ is:

$$\sum_{q=1}^{n} y_q = (D + 0.5), \quad where \ y_q = \begin{cases} (1 - w_q i_q), iff \ x_q^{key} = 1 \\ w_q i_q, iff \ x_q^{key} = 0 \end{cases} \tag{6}$$

where (D + 0.5) is the constant c of equation (1). Collecting the constants on the right-hand side of the equation (6), we obtain (D + 0.5) – α which becomes the threshold of the node that represents the newly created hyperplane and where x_q^{key} = 1. The synaptic link weights are set as follows:

$$w_{uv} = \begin{cases} -1 & iff \ x_q^{key} = 1 \\ 1 & iff \ x_q^{key} = 0 \end{cases} \tag{7}$$

where q = 1,$\cdots$, n and w_{uv} is the synaptic link that connects the input node u to the newly created vth hidden layer node.

2.3 Algorithm

The BSW algorithm is presented in pseudo-code below:

Let I be the $(n \times m)$ input matrix where n is the number of training samples and m is the input vector size, and O be the $(n \times k)$ output matrix where k is the output vector size:

$$I = \begin{bmatrix} i_{11} & \cdots & i_{1m} \\ & \cdots & \\ i_{n1} & \cdots & i_{nm} \end{bmatrix} \qquad O = \begin{bmatrix} O_{11} & \cdots & O_{1k} \\ & \cdots & \\ O_{n1} & \cdots & O_{nk} \end{bmatrix}$$

For each column, j of O, $(1 \leqq j \leqq k)$ do:

```
Step 1:
    1.1: Calculate the vector $Ave = [a_1, a_2, \dots, a_m]$, where $a_q$ is given by
         $a_q = \frac{\left(\sum_{p=1}^{n} i_{pq} \times o_{pj}\right)}{r}$   where $\begin{cases} 1 \le q \le m \\ r = \sum_{p=1}^{n} o_{pj} \end{cases}$
    1.2: Calculate the vector $key = [ky_1, ky_2, \dots, ky_m]$, where $key$ is given by
         $key = \min_{l=1}^{n}\left(Ham[Ave, R_l]\right)$   where $\begin{cases} R_l \textit{ is the } l^{th} \textit{ row of } I \\ \mathrm{Ham}[X,Y] \textit{ is the} \\ \textit{Hamming distance} \\ \textit{between vectors } X \,\&\, Y \end{cases}$
    1.3: Calculate the vector $\mathrm{Yes.dis} = [u_1, u_2, \dots, u_m]$, where $Yes.dis$ is given by
         $\mathrm{Yes.dis} = \max_{l=1}^{n}(\mathrm{Ham}[key, R_l])$   iff $O_{lj} = 1$
    1.4: Calculate the vector $\mathrm{No.clo} = [c_1, c_2, \dots, c_m]$, where $No.clo$ is given by
         $\mathrm{No.clo} = \min_{l=1}^{n}(\mathrm{Ham}[Ave, R_l])$   iff $O_{lj} = 0$
    1.5: $Dist := 0$
Step 2:
    if $(\mathrm{Ham}[key, \mathrm{Yes.dis}] < \mathrm{Ham}[key, \mathrm{No.clo}])$
    then goto Step 4
    else{ $Dist := 1;$
          goto Step 3}
```

```
Step 3:
    if $\left(\sum_{p=1}^{n} o_{pj} < \sum_{r=1}^{n} o_{rj}\right)$ or $(Dist > \mathrm{Ham}[key, \mathrm{Yes.dis}])$
        iff $\begin{cases} o_{pj} = 1 \,\&\, o_{rj} = 0 \\ \& \\ \mathrm{Ham}[key, R_p] = \mathrm{Ham}[key, R_j] = Dist \end{cases}$
    then goto Step 4;
    else {$Dist := Dist + 1;$
          goto Step 3}
Step 4:
    Separation_Plane_Creation[Dist, Dist+1]
    if (in created hyperplane there exists $R_l$ where $o_{lj}$ <> NN(key), $(1 \le l \le n)$)
    then { $key := R_l$;
           $Dist := 1$ ;
           goto Step 5
         }
    else exit;
Step 5:
    if $\left(\sum_{p=1}^{n} o_{pj} < \sum_{r=1}^{n} o_{rj}\right)$   iff $\begin{cases} o_{pj} = 1 \,\&\, o_{rj} = 1 \\ \& \\ \mathrm{Ham}[key, R_p] = \mathrm{Ham}[key, R_j] = Dist \end{cases}$
    then { $Dist := Dist + 1;$
           goto Step 5
         }
    else { Separation_Plane_Creation[Dist, Dist+1];
           goto Step 6
         }
Step 6:
    if(in the created hyperplane there exists $R_l$ where $o_{lj} = 0$, $(1 \le l \le n)$)
    then { $Dist := Dist + 1;$
           goto Step 4;
         }
    else exit;
```

```
Separation_Plane_Creation
    if $(key = [ky_1, ky_2, \ldots, ky_m])$ where $(1 \le s \le m \,\&\, ky_s = 0)$
    then $\left\{ separation\_plane = \sum_{t=1}^{m} x_t = (Dist + Dist + 1)/2 \right\} (*)$
    else $\left\{ separation\_plane = \sum_{t=1}^{m} (1 - x_t) + \sum_{q=1}^{m} x_q = (Dist + Dist + 1)/2 \right\} (**)$
                        iff $(ky_t = 1 \,\&\, ky_q = 0 \text{ in the vector } key)$
    Threshold of a hidden node = constant RHS of (*) or (**)
    (each hidden node represents one plane)
$wuv$ := weight of the synaptic link between input node $u$ and hidden node $v$.
    $:= \begin{cases} 1, iff\ ky_u = 0 \text{ in } key \\ -1, iff\ ky_u = 1 \text{ in } key \end{cases}$
$zkl$ := weight of the synaptic link between hidden node $k$ and output node $l$.
    $:= \begin{cases} 1, iff \text{ node } k \text{ gets activated} \\ 0, iff \text{ node } k \text{ is not activated} \end{cases}$
```

2.4 Example for the BSW algorithm

In this simple example, the different patterns (●,○) in 3-dimensional space will be separated. The input to the neural network is as in Table 1.

Table 1. Training data for the example problem

Input Patterns	Output Patterns
$P_1=(0,0,0)$	1 (●)
$P_2=(1,1,0)$	1 (●)
$P_3=(0,0,1)$	0 (○)
$P_4=(1,0,0)$	0 (○)
$P_5=(1,1,1)$	0 (○)

These patterns can be represented as in Figure 2(a), coinciding with the corners of a hypercube. As explained in Section 2.2, initially $\vec{x}^{av}$ will be determined. Since there are only two ● patterns, we calculate the x_j^{av} to be the average of the jth components of

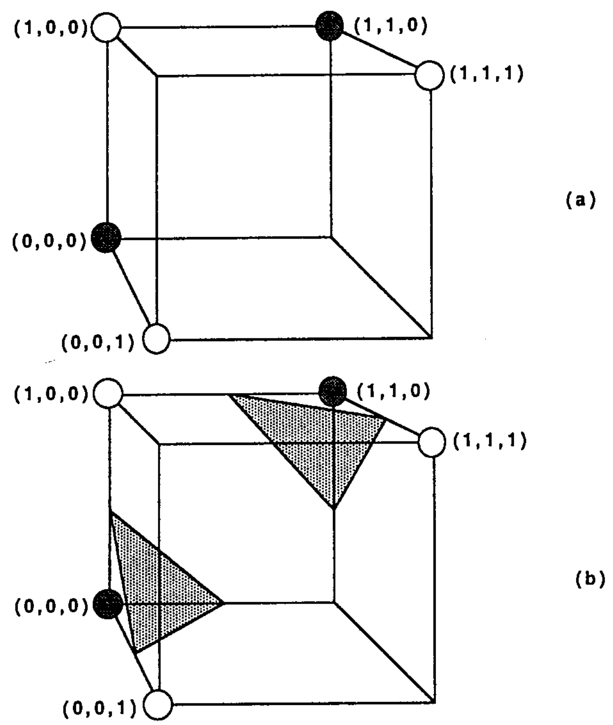


Figure 2.(a) Patterns to be separated; (b) Separation by BSW.

pattern vectors P_1 and P_2, where $0 \leqq j \leqq 2$. Thus $x_1^{av} = (0+1)/2$, $x_2^{av} = (0+1)/2$, $x_3^{av} = (0+0)/2$. As 0.5 falls between 0 and 1, either (0,0,0) or (1,1,0) can become $\vec{x}^{av}$. Let us choose $\vec{x}^{av}$ to be (0,0,0), then *HamDist*[(0,0,0),(0,0,0)] = 0 and *HamDist*[(0,0,0),(1,1,0)] = 2 and according to equation (3) $P_1 = (0,0,0)$ becomes $\vec{x}^{key}$.

At this point, we classify all patterns in the pattern space according to their Hamming distances to $\vec{x}^{key}$. This is presented in Table 2.

Table 2. Hamming distance classifications of training patterns according $\vec{x}^{key}$

Ham. Dist.= 0	Ham. Dist.= 1	Ham. Dist.= 2	Ham. Dist.= 3
(0,0,0) ●	(0,0,1) ○	(1,1,0) ●	(1,1,1) ○
	(1,0,0) ○		

From this representation, we can see that $\vec{x}^{yes} = (1,1,0)$, HamDist[$\vec{x}^{key}$,$\vec{x}^{key}$] = 2 and $\vec{x}^{no} = (0,0,1)$or(1,0,0), HamDist[$\vec{x}^{key}$,$\vec{x}^{no}$] = 1. Since HamDist[$\vec{x}^{key}$,$\vec{x}^{yes}$] is greater than HamDist[$\vec{x}^{key}$,$\vec{x}^{no}$], we can say that the patterns are not linearly separable. At distance

D = 0, there is only one pattern $\overrightarrow{x^{key}}$, and at D = 1, the number of ○ patterns is 2, while the number of ● patterns is 0. Therefore, as at distance D = 1, the number of ○ patterns is greater than number of ● patterns, we perform separation at D = 0 and the equation of the separation hyperplane becomes x1+x2+x3 = (0+0+1)/2 = 0.5. The side of the hyperplane which encloses (0,0,0) is represented by x1+x2+x3 $\leq$ 0.5. With respect to Figure 2(b), in order to represent this in a form suitable for threshold logic, we multiply both sides of the inequality by −1 and obtain −x1−x2−x3 $\geqq$−0.5. The threshold of the newly created hidden layer node in −0.5 and the synaptic link weights connecting this node to the input nodes are the coefficients of the x-components in this equation (−1,−1,−1).

The next step is to separate the remaining ● pattern and to accomplish this, P2 = (1,1,0) is selected as $\overrightarrow{x^{key}}$. Classification of all patterns according to their Hamming distances to $\overrightarrow{x^{key}}$, we determine that the separation plane will be formed at D = 0. With reference to equation (6), the separation plane's equation becomes $(1-x_1)+(1-x_2)+x_3 = (0+0+1)/2 = 0.5$ and if we accumulate all constants on one side the equation becomes $-x_1-x_2+x_3 = -1.5$. Since the side of the plane which encloses (1,1,0) is desired, we set the synaptic links as (1,1,−1) and the threshold as 1.5. Finally, the threshold of the output layer node is determined. The output pattern vector has only one column, which requires only one output node. The number of planes created to enclose either ● pattern was 1; therefore, the threshold of the output node is (1− 0.5) = 0.5. The resultant network topology is shown in Figure 3.

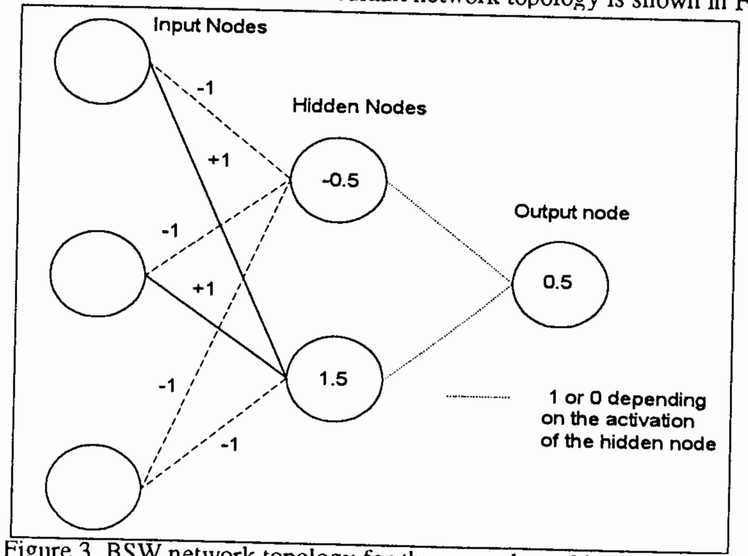


Figure 3. BSW network topology for the example problem of Section 2.4.

3 On-Line Geometric Shape Recognition with Fuzzy Filtering and Neural Network Classification

3.1 Geometric Shape Recognition

The aim of this section is to present a simple method to recognize and correctly redraw hand-drawn regular geometric shapes, such as circles, ellipses, squares, rectangles, and triangles. This would be beneficial in automated sketch entry in software packages to handle drawings in computers. Each shape is recognized as a whole, regardless of size, translation, rotation, or choice of starting point, instead of recognizing the individual segments and then performing syntactic analysis as discussed in [3]. The main concept of the recognition method presented in this section, is derived from the fact that angles are very important in the perception of shapes. Although Davis has also used this concept in his work [4], his method of syntactically analyzing the angles is weak in terms of dealing with the noise along the boundary of the shape. To overcome previous works' demerits and accomplish fast recognition with a high success rate, we have employed scaling/rotation/translation-invariant feature extraction, noise reduction through fuzzy function filtering and classification using a neural network.

Feature extraction is performed on the captured points along the boundary of the shape. Since the captured data for the boundary of a hand-drawn shape invariably includes noise, the fuzzy function is beneficial in elimination of most of the noise and also in detecting portions of the boundary of the shape that exhibit significant angular differences. The features of a shape are represented as a parsable string in terms of significant internal angles and this string is then input to an appropriate type of classifier [5]. The feature extraction and fuzzy filtering that we present in this chapter, prepares data that is invariant in terms of translation, scaling, rotation and in the case of triangles, even invariant of the particular class of the triangle, with a reduced noise level for input to the neural network. Previous work employing neural networks in the classification process usually concentrated on teaching a particular shape to the network and then measuring how well the network performs on noisy versions of that particular shape [6–8]. However, our purpose is to teach the network the definition of a class of shapes, such as triangles or ellipses. In our method, the training set is not necessary to include all kinds of possible shapes, but the training can be accomplished using only a small representative set. The speed with which BSW can be trained enabled us to add an incremental training module to the application.

3.2 Feature Extraction

Preprocessing of the data for the purpose of feature extraction is performed prior to the application of recognition algorithms. In other words, the purpose of preprocessing is the creation of an intermediate representation of the input data, in which information that does not help in the discrimination between different classes is reduced and desired or 'useful' information is enhanced [9]. In the case of handwritten shapes, undesirable information includes noise due to the limiting accuracy of the tablet, noise introduced in the digitizing process, variations in the capture speed of the mouse and erratic hand motion while drawing with the stylus or dragging the mouse. In the presence of undesirable information in the input during shape recognition, we have to distinguish between essential and nonessential curvature changes along the boundary. In order to accomplish this without a significant loss in useful information, our feature extraction process involves a number of steps. We are proposing two approaches to feature extraction designated as Method1 and Method2.

3.2.1 Method1

Method1 involves the steps of resampling the input data, calculation of the center of the shape and extraction of significant points.

3.2.1.1 Resampling

The raw input data obtained from the contact between the stylus with the tablet, or the mouse with the pad, which is the input media used for the application purposes of this chapter, has enormous variations in drawing speed and therefore the points that represent the shape are not equally spaced. In order to obtain equally spaced points along the trajectory of the stylus or the mouse, we need to resample the input data [9].

Simple linear interpolation between captured points within every pen-down/mouse button-down and pen-up/mouse button-up portions of the drawing serves the purpose of resampling. Linear interpolation is the interpolation between given two points (x_0,f_0) and (x_1,f_1) through the formulae [10]:

$$P_1(x) = f_0 + (x - x_0) f[x_0, x_1]$$

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0} \tag{8}$$

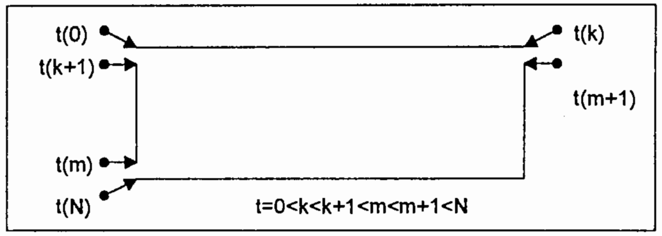


Figure 4. Rectangle drawn with unordered strokes; N is the total number of boundary points.

The application of interpolation is feasible only between subsequent pen-down/pen-up portions of the capture data because of the possible variations in the order of strokes while drawing a shape as seen in Figure 4.

3.2.1.2 Calculation of center of the shape

In the case of handwritten character recognition, in order to obtain a representation which is scale and translation invariant, the center point (x_c, y_c) of the characters is calculated with the following formulae:

$$x_c = \frac{x_{\max} + x_{\min}}{2} \quad and \quad y_c = \frac{y_{\max} + y_{\min}}{2} \tag{9}$$

However, in shape recognition, calculation of the center of the shape with this method does not always bring the correct center of the shape, as can be seen in Figure 5. Therefore, we chose to calculate the center point of a shape through the center of gravity method which defines:

$$x_c = \frac{\sum_{i=0}^{N-1} x_i}{N} \quad and \quad y_c = \frac{\sum_{i=0}^{N-1} y_i}{N} \tag{10}$$

where N is the total number of captured data points. The drawing speed, i.e., the speed at which the stylus is moved or the mouse is dragged, makes a difference in the number of points captured in a certain part of the shape, which may lead to shifts in the location of the center of gravity. For this reason, the calculation of (x_c, y_c) is performed after resampling of the captured data points.

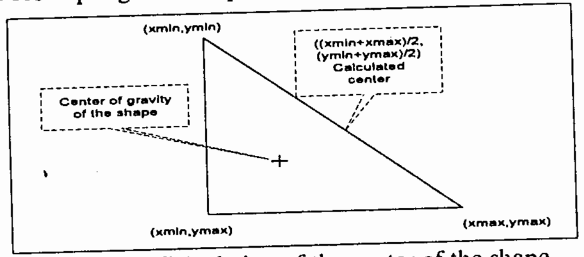

(xmin,ymin)
((xmin+xmax)/2, (ymin+ymax)/2) Calculated center
Center of gravity of the shape
(xmin,ymax)
(xmax,ymax)

Figure 5. Calculation of the center of the shape.

3.2.1.3 Extraction of Significant Points

After resampling and the calculation of the center of the shape (x_c, y_c), we would like to extract L points, which will sufficiently represent the shape, out of the total of N input data points. Figure 6 depicts L quantized vectors that are angularly equispaced, meaning there is an angular difference of ($2\pi/L$) between every two successive vectors. The L sample points $\{[p_x[i], p_y[i]), i = 1, \cdots, L\}$, will be chosen to be the intersections of the shape boundary with L quantized direction vectors originating from (x_c, y_c) as seen in Figure 7. These sample points possess the circularity property as $P_{L+k} = (p_x[L+k], p_y[L+k]) = P_k = (p_x[k], p_y[k])$ for any integer k, $1 \leqq k \leqq L$. This method of sample point extraction makes the resultant shape representation a candidate for the circular auto-regressive (CAR) model [10]. This is desirable as the parameters of CAR models are invariant to transformations on the boundary, such as translation, choice of starting point and rotation over angles that are multiples of $(2\pi/L)$.

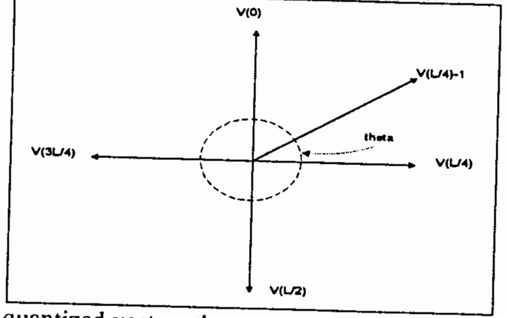


Figure 6. L quantized vectors that are angularly equispaced (theta = $2\pi/L$).

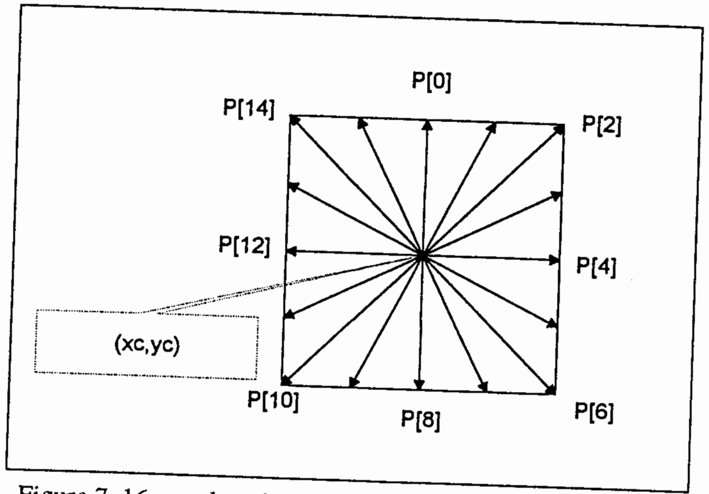


Figure 7. 16 sample points are extracted on a shape (L = 16).

There are cases where a significant point in a shape falls between two sample points as depicted in Figure 8. For the purposes of this method of recognition, any point (x), which stands for the significant point, must be taken into consideration. In addition to this, we must also keep the number of sample points a constant, as the sample points will become the input to a neural network classifier after further manipulation.

The method we use in obtaining significant points such as (x) in Figure 8, is based upon the method used for obtaining line approximations of the boundary of a shape in [11]. This method involves finding the longest line segments that have the minimum sum of absolute errors along the digitized shape boundary. The equation of a line that originates in $P_i = (x_i, y_i)$ and ends in $P_j = (x_j, y_j)$ is:

$$(y_j - y_i)x - (x_j - x_i)y + x_i y_j - x_j y_i = 0 \tag{11}$$

The points that lie in between P_i and P_j add towards an error term depending on how well they fit the equation (11). This error term (E = sum of absolute errors) for each line segment $[P_i, P_j]$ is calculated as follows;

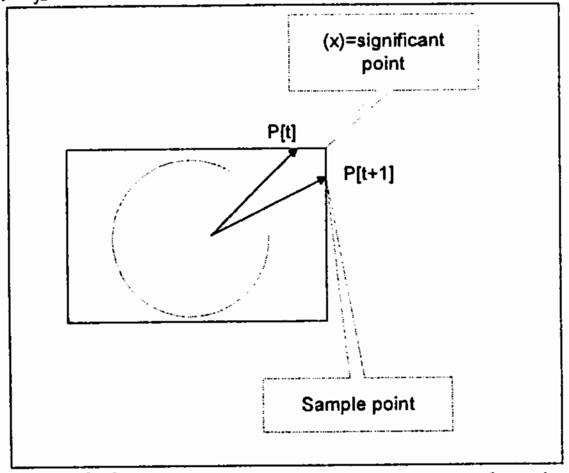


Figure 8. Significant point between two sample points.

$$E = \frac{\sum_{k=i+1}^{j-1} \left| (y_j - y_i)x_k - (x_j - x_i)y_k + x_j y_i - x_i y_j \right|}{\alpha \left\{ (x_j - x_i)^2 + (y_j - y_i)^2 \right\}^{1/2}} \tag{12}$$

where $k = i+1, \cdots, j-1$ are the points along the boundary of the shape that lie in between P_i and P_j and α is the constant associated with the sensitivity of the input device. Since our purpose is to find the longest line segment with the minimum E, we maximize $(LS - E)$ where LS is the length of the line segment under consideration.

$$LS - E = \left\{ (x_i - x_j)^2 + (y_i - y_j)^2 \right\}^{1/2} - E \tag{13}$$

Maximizing the equation (13) corresponds to maximizing LS and minimizing E. We select a sample point on the boundary and assign it to P_i. We must then normalize the coordinates of the remaining points such that P_i becomes the origin. Each consecutive point on the boundary is tested until the first line segment that satisfies the maximization of the equation (13) is found. This point becomes the new P_i and the process is repeated until the boundary has been traversed. Eventually we obtain a list of significant points in addition to the previously calculated sample points, which need to be merged to maintain a constant number of representative points. The merging process involves shifting the sample points toward the nearest significant point in the case where a significant point falls in between two sample points as illustrated in Figure 8.

3.2.2 Method2

In the second method of feature extraction, we aim to eliminate crossovers, both internal and external, and finally obtain L sample points as in Method1. The L sample points, which are the beginning and end points of the vectors $\{(P_x[i], P_y[i]), i = 1, \cdots, L\}$, are chosen to be the intersections of the input shape boundary with the L vectors originating from (x_c, y_c), as shown in Figure 9(a). Figures 9(b) and (c) present a rectangle and circle, respectively, that have been drawn by hand. As can be seen in the figures, these hand-drawn shapes contain a large amount of undesirable information such as internal and external extensions, and concave segments, which should be removed. In addition to removing the undesirable information, we would like to detect information which will aid in recognition, such as intersection points or corners, and also those input points which contribute significant information to the class of the shape. In order to produce the intermediate representation of the input data as shown in Figure 9(a), from hand-input shapes such as those of Figures 9(b) and (c), without losing significant information, the feature extraction process involves a number of steps. These include the extraction of a set of sample points which contribute significant information to the shape, calculation of the center point of the shape, detection of intersection points, removal of internal and external extensions, determining the convex hull of the shape, determining sample points along the shape boundary, and the formation of tangent vectors between these points. These steps are discussed in the following subsections.

3.2.2.1 Extraction of Significant Sample Points

Of the total N input data points, we would like to extract the J points, which capture only the significant changes along the shape boundary, without losing any significant information. Also, we wish to process each point on-line, as it is received from the mouse or stylus. This will reduce the total amount of data which must be passed to later stages of the preprocessor and also allows the overlapping of the input and processing of the received data. As in Method1, we have adapted the method proposed in [11], which determines the optimal polygon from a digital curve using the L_1 norm, for use in an on-line recognition environment. This method determines the longest line segments that have the minimum sum of absolute errors along the digitized shape boundary. The equation of a line that originates at $P_i = (x_i, y_i)$ and ends in $P_j = (x_j, y_j)$ is the same as in equation (11). The points (x_k, y_k), $k = i{+}1, i{+}2, \cdots, j-1$, which are on the boundary of the shape between points P_i and P_j contribute to the sum of the absolute errors E, for the line segment representing the boundary $[P_i, P_j]$, as in equation (12). The variable α is the sensitivity parameter that is determined as a function of the input device type and the speed with which the input device is moved while data is being captured, such as $\alpha = f(\phi, t)$, where t is a real number heuristically determined for each type of input device and ϕ is a real number determined according to the acceleration and deceleration in the strokes of the drawing. Since our purpose is to find the longest line segment with the minimum E, we maximize $(L_s - E)$, where L_s is the length of the line segment under consideration:

$$\begin{aligned} M_s &= L_s - E \\ &= \left\{(x_j - x_i)^2 + (y_j - y_i)^2\right\}^{1/2} - E \\ &= \left\{(x_j - x_i)^2 + (y_j - y_i)^2\right\}^{1/2} - \frac{\sum_{k=i+1}^{j-1} \left|(y_j - y_i)x_k - (x_j - x_i)y_k + x_j y_i - x_i y_j\right|}{\alpha\left\{(x_j - x_i)^2 + (y_j - y_i)^2\right\}^{1/2}} \end{aligned} \tag{14}$$

Maximizing M_s in equation (14) corresponds to maximizing L_s and minimizing E. Given the set of points representing a stroke, $P = \{p_0, p_1, \ldots, p_{n-1}\}$ input by mouse or stylus, where n is the number of points, our on-line approach to significant point detection proceeds as follows: The first point p_0 is accepted as a significant point and is assigned to (x_i, y_i) of equation (14). Each time a new point, $p_R \in \{p_1, \ldots, p_{n-1}\}$, is received from the mouse or stylus, it is assigned to (x_j, y_j) and M_R is calculated using equation (14). If $M_R \geqq M_{R-1}$, then p_{R-1} is not significant and the process will be repeated when a new p_R is received. If $M_R < M_{R-1}$, then p_{R-1} is significant, and therefore it is saved and it is also assigned to be the new (x_i, y_i). This process is repeated until all of the input points for the stroke have been received, after which the end point p_{n-1} is saved as a significant point.

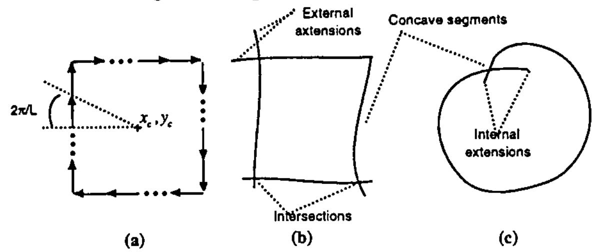


Figure 9.(a) Shape represented by equispaced vectors; (b) hand-drawn rectangle; (c) hand-drawn circle.

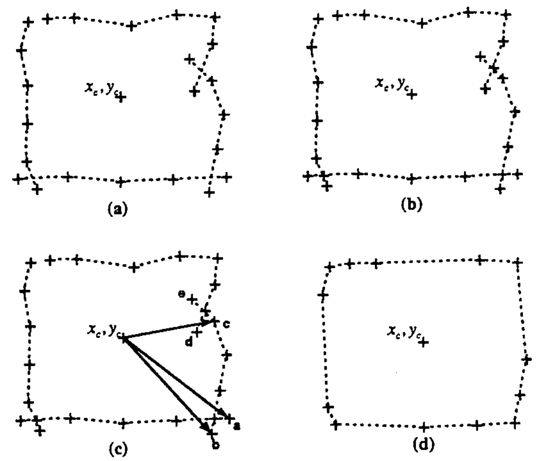


Figure 10.(a) Significant points and stroke segment input data. (b) Detection of intersection points. (c) Removal of extensions. (d) Shape after application of convex hull algorithm.

3.2.2.2 Calculation of the Center of Gravity

As explained in the section on resampling for Method1, due to the variations in the drawing speed, the points which represent the shape will not be equally spaced. In addition, the distance between significant points obtained in the previous section will exhibit significant variation in spacing. Those areas of the figure in which the rate of change in the direction of the line segments, formed by the points, is small, will result in far less significant points than in those areas in which the rate of change is high. According to the center of gravity method, the center point (x_c, y_c) is determined by equation (10) as in Method1. By considering the effects of capture speed and significant point spacing, the center of gravity calculation becomes:

$$x_c = \frac{\sum_{i=0}^{M-1} \text{length}(segment_i) \times \text{X}_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)} = \frac{\sum_{i=0}^{M-1} \text{X}_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)}$$

$$y_c = \frac{\sum_{i=0}^{M-1} \text{length}(segment_i) \times \text{Y}_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)} = \frac{\sum_{i=0}^{M-1} \text{Y}_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)} \qquad (15)$$

where M is the number of line segments, and the functions length( ), X_{ave}( ) and Y_{ave}( ) return, respectively, the length, average x-value, and average y-value of a segment. As each new significant point, p_R is added to the data set, the cumulative length, cumulative X_{weight}, and cumulative Y_{weight} are updated for the segment $[p_{R-1}, p_R]$, further reducing the processing which must be carried out once input is complete.

3.2.2.3 Residual Preprocessing of the Input

Once the strokes which define the input shape have been entered, we further process the significant point data. The center of gravity is calculated using the cumulative length, cumulative X_{weight} and cumulative Y_{weight} according to (15). Intersections between any of the segments which form the strokes that define the shape are determined and saved. Figure 10(a) presents a shape defined by its significant points and segments between consecutive points in each of four strokes. In Figure 10(b), the intersection points of the segments have been detected. To remove the extensions present in the shape, a ray is fired from the center of gravity to each significant point as is shown in Figure 10(c). If the ray intersects with any other segment, then the point lies outside of the shape and can be discarded; otherwise, the point is saved. With respect to Figure 10(c), this is correct for points a and b; however, the presence of the internal extensions d and e will result in point c in being erroneously discarded, while points d and e are saved. Taking this into consideration, the convex hull

algorithm is applied to the points of the shape, points d and e are removed, and the resultant shape is shown in Figure 10(d).

3.3 Formation of Tangent Vectors Along Shape Boundary

A constant number of sample points that can sufficiently represent the shape of the boundary is necessary for input to the neural network. In order to obtain the L representative points along the boundary of the shape for the formation of the tangent vectors illustrated in Figure 9(a), rays are fired from the center of gravity to the boundary of the shape at an angular spacing of $(2\pi/L)$. However, in the case where a significant point falls between two of the equispaced rays, as in Figure 8, this may result in the loss of a corner point from a shape. To ensure that this does not occur, the J significant points, determined by the significant points process discussed in Sections 3.2.a and 3.2b, are merged with the ray intersection points by shifting the ray intersection points toward the nearest significant point. Once L representative points along the boundary are obtained, we calculate the tangent vectors $t\nu_i$ between the ith and the $(i+1)$th points $i = 0, \cdots, L-1$. To form a closed shape, we also calculate $t\nu_L = \frac{y_0 - y_L}{x_0 - x_L}$. After L tangent vectors are obtained, we want to apply an approximation to these tangent vectors to match each vector with a quantized direction vector. The quantized direction vectors are equally spaced to be $(2\pi/G)$ degrees apart for any number of vectors G. In our implementation, we have chosen G to be equal to L. As G increases, the presence of noise on the shape becomes more accented, whereas very low values of G result in a significant loss in information. Figure 11 depicts quantized vector differences of boundary tangent vectors.

3.4 Fuzzy Function Filtering

We propose that the information provided by the tangent vectors be analyzed by one of the following two methods:

- **The quantized vector differences**: The difference between any two consequent tangent vectors along the boundary of the shape is the clockwise difference between the orderings of the vectors as seen in Figure 11. The difference $d_i = q\nu_d[t\nu_i, t\nu_{i+1}] = k$, where k is an integer.

- **The angle between two tangent vectors**: This method actually utilizes the outcome of the first method and obtains the angle between $t\nu_i$ and $t\nu_{i+1}$ by

$$angle[i] = qvd[tv_i, tv_{i+1}] \times (2\pi/L) \qquad (16)$$

Figure 12 depicts the angular difference between any two consequent tangent vectors of a shape. The second method is utilized as input to our fuzzy function depicted in Figure 13. Since angular differences along the tangents of the handwritten shape boundary are susceptible to noise, we devised a fuzzy function and tuned it by a heuristic analysis of a large number of sample shape boundaries. This fuzzy function helps reduce the effect of noise by attenuating the effect of small angular deviations which it introduces. It also brings out the portions of the boundary that exhibit significant changes in the internal angles between consecutive tangents.

The angle between two consecutive tangent vectors is assigned a degree of membership between 0 and 1, for each of the fuzzy sets illustrated in Figure 13. These fuzzy sets broadly classify an angle into a category such as straight, obtuse, right, acute, or reflex. For further detail, we have divided obtuse angle set into two; right-obtuse, and wide-obtuse, depending on how close the angle is to 90°. In the cases where an angle has more than one non-zero membership value, for example, right and right-obtuse, the maximum membership value determines the winner [12]. Once we have only a few meaningful categories to classify the angles into, rather than 360 individual values, we can count the frequency of their occurrence along the shape boundary. The frequency count of significant angles will be the input to the neural network.

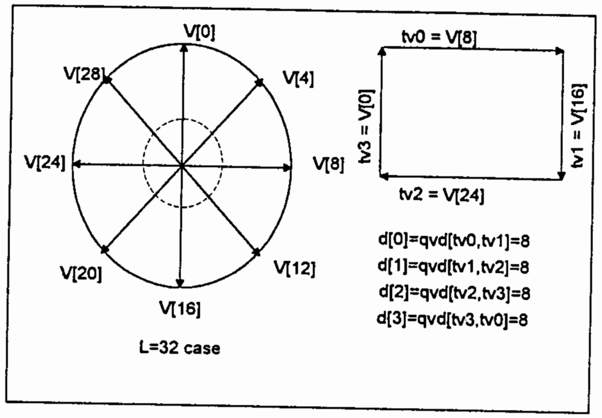


Figure 11. Quantized vector differences of boundary tangent vectors.
$q\,\nu_d$ = quantized vector difference function
V[i] = i-th direction vector, $i = 0, \ldots, L = 31$
d[j] = j-th quantized vector difference, $j = 0, \cdots, M-1$,
M = #tangent vectors
$t\,\nu\,k$ = tangent vector k, $k = 0, \cdots, M-1$

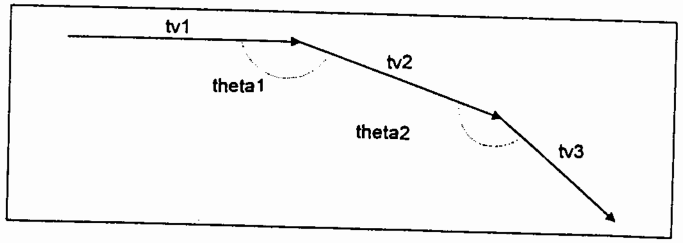


Figure 12. Angular differences between tangent vectors.

3.5 On-line Shape Classification and Training by a BSW Network

The fuzzy function supplies the information on the detected angles. The four most significant types of angles that determine a shape (with reference to Figure 13) are wide-obtuse, right-obtuse, right, and acute. This information is supplied as input to the neural network. Formally, the input that the neural network receives is as follows:

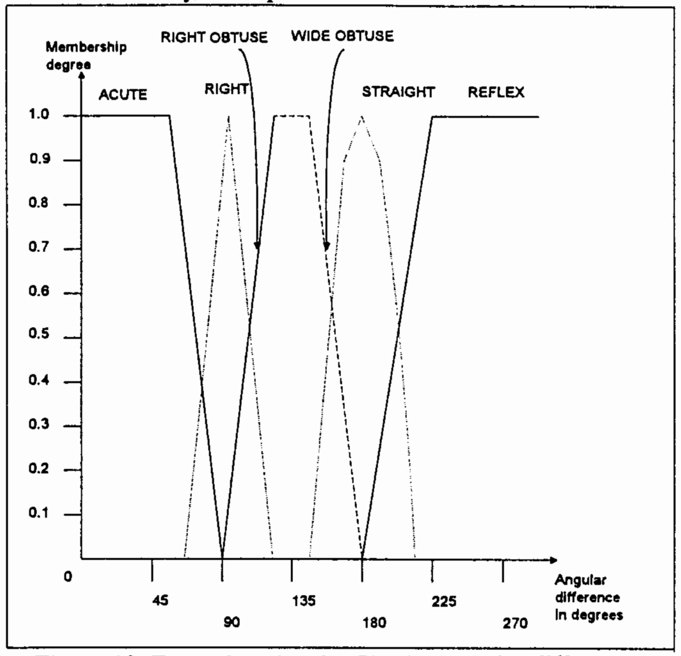


Figure 13. Fuzzy function for filtering angular differences.

$$\#atype = \sum_{i=0}^{L} \max\left(fuzzy\ membership\left(angle[i]\right)\right) \tag{17}$$

where *angle*[*i*] is determined according to equation (16) and #*atype* represents the sum of a specific type of angle, such as #wide-obtuse, #right-obtuse, #right, or #acute, as shown in Figure 13.

The neural network's task is to find the relationships between the fuzzy filtered internal angles of a geometric shape and its desired classification, while maintaining its generalization ability for recognizing noisy shapes. The desired classifications are circular shapes (circle, ellipse), rectangular shapes (square and rectangle), and triangular shapes (any triangle). The speed with which a neural network can be trained by BSW algorithm makes it a perfect candidate for including on-line training in the system, where the user can retrain the system to classify a shape into a certain category.

In classification of shapes, the output of the fuzzy function supplies the information on the detected angles, and the input that the neural network receives is prepared as in equation (17). BSW networks accept binary input; therefore, we convert these integer values to binary digits. However, since the BSW distinguishes different patterns according to their Hamming distances, the conventional binary-radix integer representation would create conflicts, as can be observed in Table 3. In order to make the Hamming distances of patterns equal to their Euclidean distances, we use Serial Coding, which is depicted in Table 4. In a system where L tangent vectors are utilized, the range of values that *#atype* can assume is 0 to (L-1). Therefore, each node that represents an integer valued *#atype* is expanded into L nodes after serial coding. In this experiment, 31 nodes are required for each *#atype* to be represented using Serial Coding. Therefore, the input layer consists of $31 \times 4 = 124$ nodes.

Table 3. Conventional Binary Coding

Integer	Binary vector	Hamming Distance From 0	Euclidean Distance From 0
0	00000	0	0
1	00001	1	1
2	00010	1	2
3	00011	2	3
4	00100	1	4
5	00101	2	5
.	.	.	.
.	.	.	.
.	.	.	.

Table 4. Serial Coding

Integer	Binary vector (Serially Coded)	Hamming Distance From 0	Euclidean Distance From 0
0	00000	0	0
1	00001	1	1
2	00011	2	2
3	00111	3	3
4	01111	4	4
5	11111	5	5
.	.	.	.
.	.	.	.
.	.	.	.

The network was initially trained with a total of 7 sample shapes drawn by mouse which are presented in Figures 14(a) through (c). The resultant network topology had three hidden nodes created during initial training, as shown in Figure 15. The initial training set is kept small on purpose, so that the user can incrementally train the system according to his/her style of drawing. Therefore, initially, the network is given enough samples to roughly classify elliptic, rectangular, and triangular shapes. For testing, geometric shapes are again drawn by mouse and on-line recognition is performed. Four different users were given the same initial configuration and told to

draw a total of 50 figures. Each set of figures was to include elliptic, triangular, and rectangular shapes. They were told that they could add samples that are not recognized by the network and train the network on-line. It was left to the discretion of the user whether the unrecognized figure should be included in the training set as a representative of that particular class of shapes. Thus, it was possible that the user may add a noisy figure which may create conflicts, such as the new input vector being the same as a previously included vector that had a different output classification. However, there is a simple verification mechanism, so that after the user adds a training sample and performs on-line training, he/she can test the network for the recognition of the newly added sample. If more than one classification is chosen, the user is informed and has the option of removing the conflicting data from the training set.

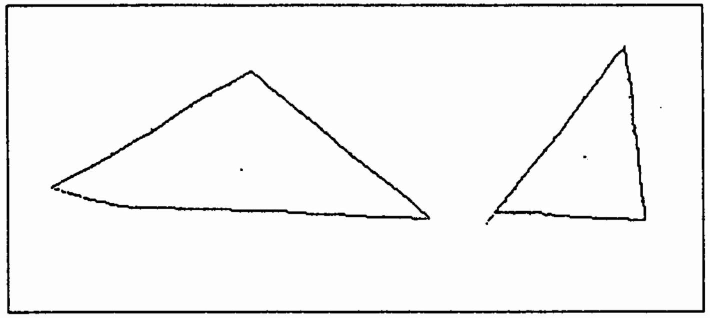


Figure 14(a). Initial training samples for triangular shapes

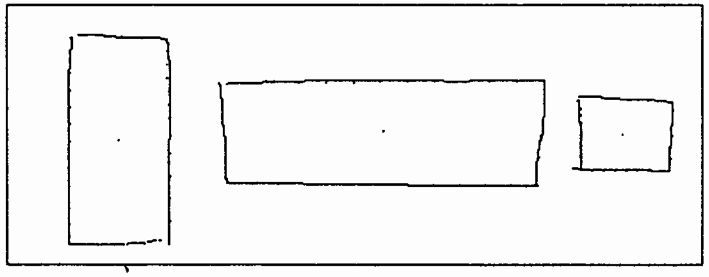


Figure 14(b). Initial training samples for rectangular shapes

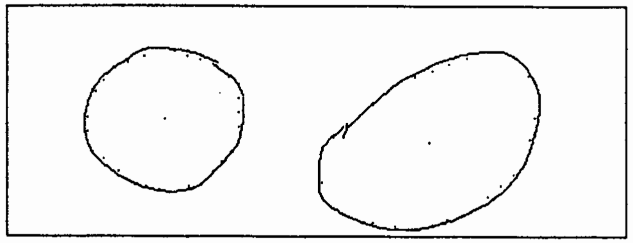


Figure 14(c). Initial training samples for circular shapes

Table 5 displays the additions to the training set made by the four users and the resultant number of hidden layer nodes. For reference, we are including the figures drawn by users *User*1 and *User*3, in Figures 16(a),(b),(c) and 17(a),(b),(c). Also the network topology at the end of the *User3*'s session is given in Figure 18. With reference to Figures 16 and 17, the order in which the shapes were entered into the system was from left to right and top to bottom. The shapes enclosed in boxes are the additional samples which were added to the system on-line. In this experiment, none of the additional shapes input by the users caused a classification conflict. The initial network exhibited very good performance by recognizing shapes that were not in the training set and had noise along the shape boundary. Thus, in this example, our network has learned the underlying properties of the class of triangular shapes, rather than a set of specific triangles. Variety rather than the number of shapes recognized clearly illustrates this point. For the addition of a new shape to the training set, the desired category information was interactively supplied by the user and the network was retrained, which took less than 10 seconds on a 200MHz-clock PC. The resultant networks had varying configurations depending upon the input vectors of the additional training samples.

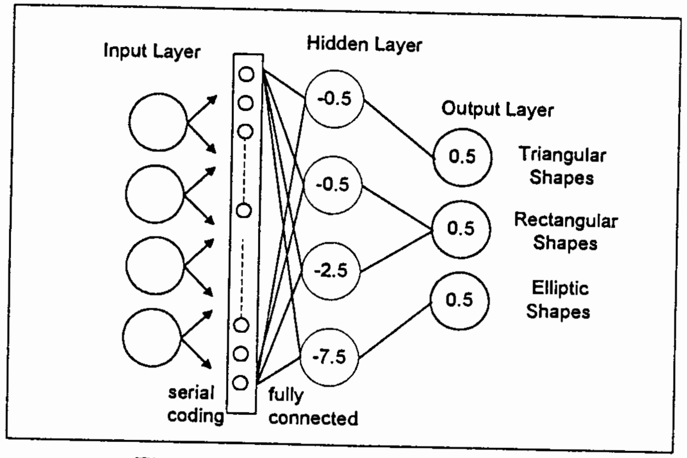


Figure 15. Initial network topology.

Table 5. Results of training sessions by four users

	Number of additional triangular training samples	Number of additional rectangular training samples	Number of additional elliptic training samples	Total number of additional training samples	Number of resultant hidden layer nodes
User1	2	4	4	10	13
User2	2	2	3	7	8
User3	3	3	1	7	8
User4	0	1	1	2	5

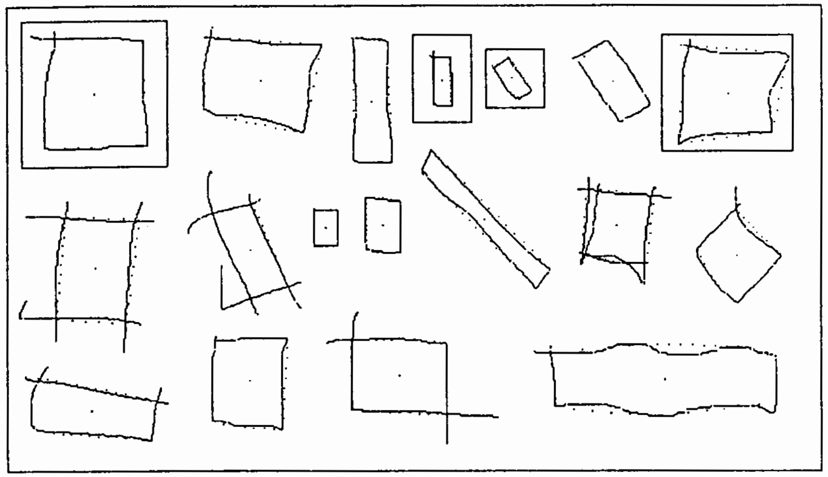


Figure 16(a). Rectangular shapes drawn by *User*1.

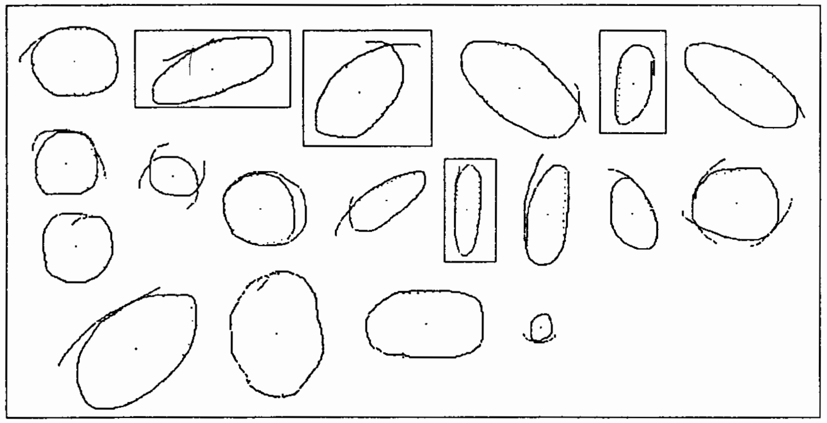


Figure 16(b). Elliptic shapes drawn by *User*1.

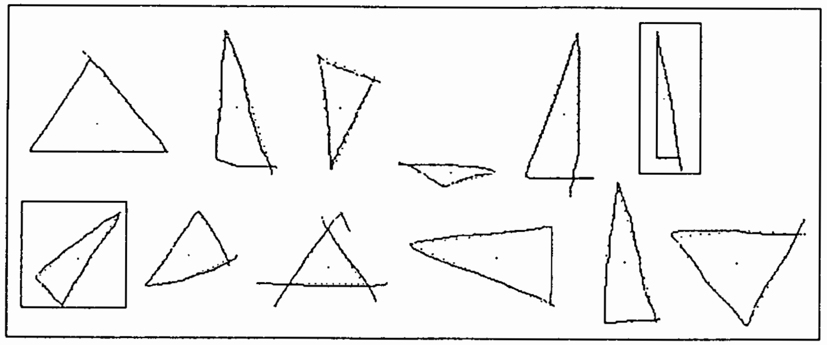


Figure 16(c). Triangular shapes drawn by *User*1.

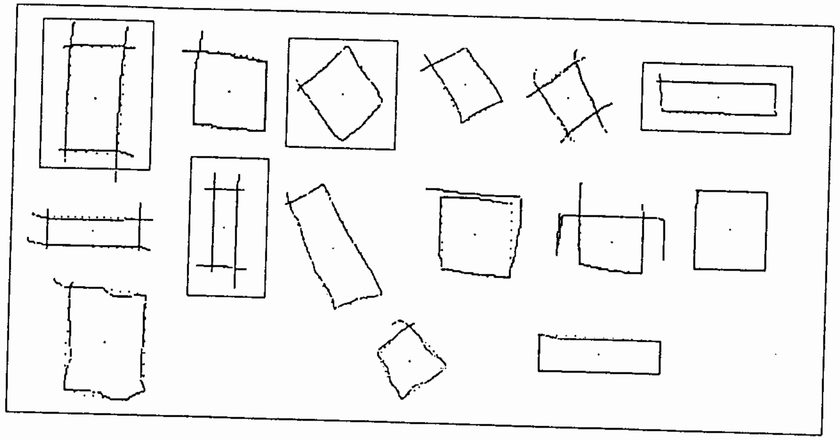


Figure 17(a). Rectangular shapes drawn by *User3*.

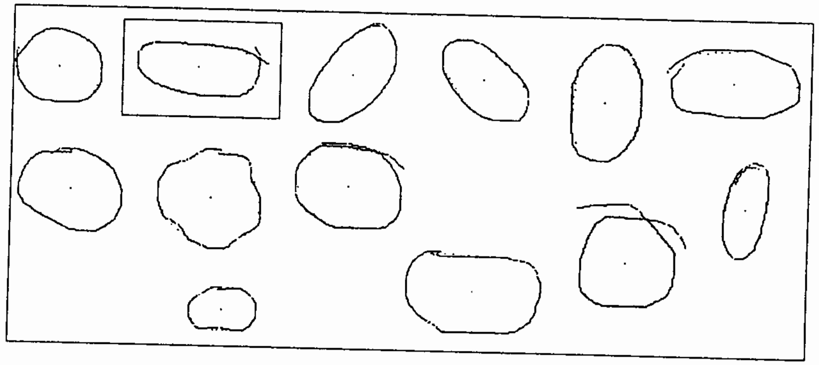


Figure 17(b). Elliptic shapes drawn by *User3*.

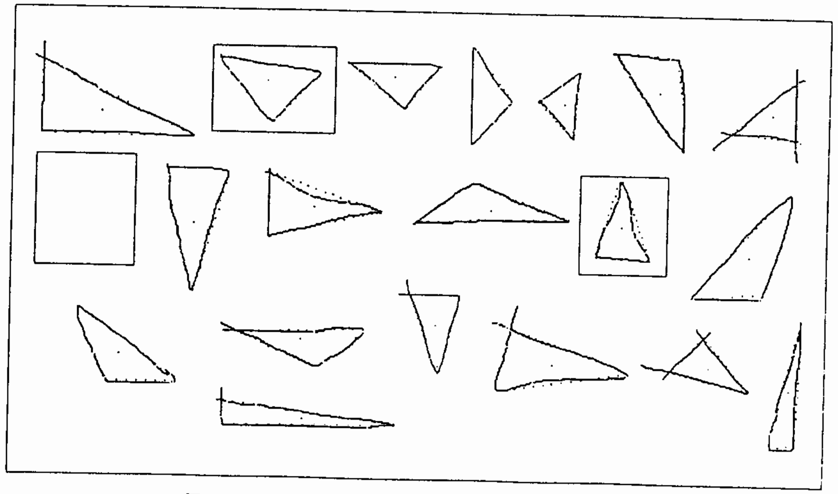


Figure 17(c). Triangular shapes drawn by *User3*.

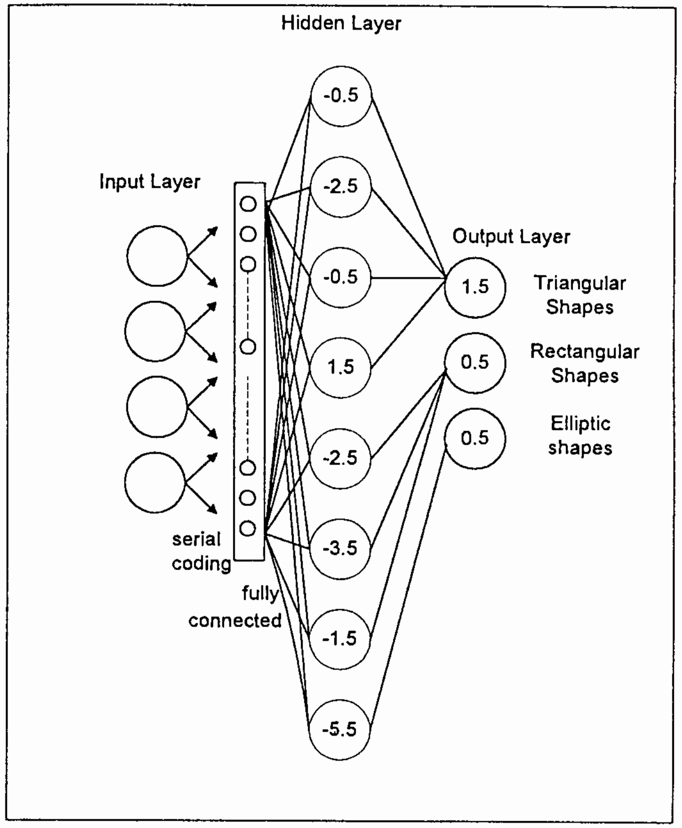


Figure 18. The network configuration after User3's session.

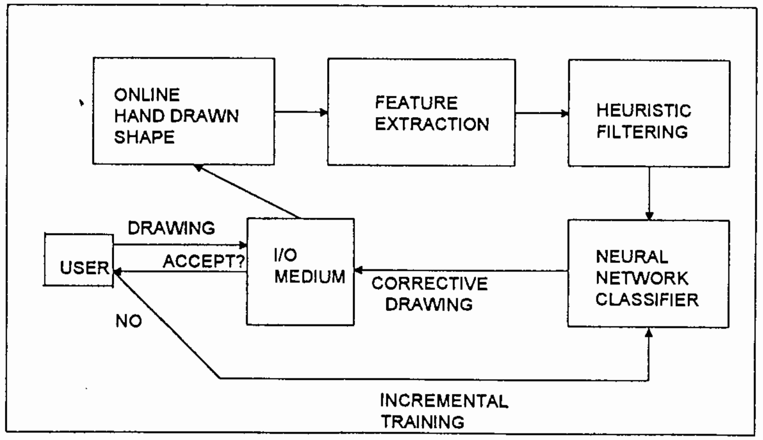


Figure 19. Schematic representation for the customizable figure recognition system Binary Synaptic Weight Neural Network).

3.6 Results

The total recognition time taken from the completion of shape input, until the corrected shape is redrawn, is less than 0.03 seconds on a Pentium PC (clock: 100MHz). The process of correction is illustrated in Figures 20(a) through (c) which present an input shape, the results of feature extraction, and the redrawn figure, respectively.

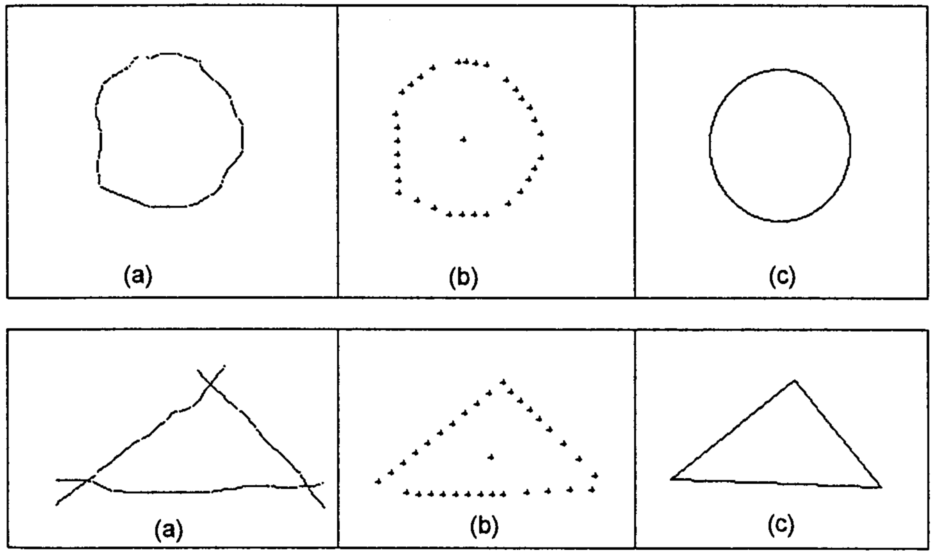


Figure 20.(a) Shape drawn by mouse;
(b) Feature extraction and preprocessing results;
(c) Recognized and redrawn shape.

The BSW algorithm, used for the training purposes of the neural network, was fast enough to allow us to include on-line training in the system. With this type of system, the user can customize the recognition of shapes to his/her drawing style by initially training the system with a few samples and then incrementally training as unrecognized shapes emerge in drawings. However, caution must be taken in order not to teach a shape that cannot be clearly judged to be of a specific category, as this would confuse the recognition ability of the system.

The result for the application of the presented BSW algorithm has proved that this algorithm is an effective pattern classifier. It has a number of desirable merits such as fast and guaranteed learning and it can be easily implemented in hardware. Therefore, it is recommended for use as an alternative neural network training algorithm for many problems.

4 Conclusion

In this chapter, we have presented a new method to recognize basic geometric shapes and provide on-line training through the use of feature extraction, a heuristic filtering function, and a fast training neural network. The overall structure of our current system, depicted schematically in Figure 19, takes an on-line hand-drawn shape, input mouse, and performs feature extraction, heuristic filtering, and neural network classification, presenting a corrected redrawing of the recognized shape on the output device. The process of correction is illustrated in Figures 20(a) through (c), which present an input shape, the result of feature extraction, and the redrawn figure, respectively.

The feature extraction process is important since it prepares input that is invariant in terms of scaling, translation, and rotation, which is a major problem in computer recognition of images. The heuristic function is very beneficial in reducing the noise and accenting the significant curvature maxima. The neural network brings the ability of dynamically expanding the knowledge of the system for the classification of shapes. Shape recognition by most neural network application is performed by training the system with a particular shape and trying to recognize its noisy or rotated versions rather than trying to capture the definition of that class of shapes. With this approach, our objective was to mimic the flexibility of the human eye in the recognition of three basic geometric shapes. In order words, the neural network learned the underlying definition of a category of three classes of geometric shapes in terms of their internal angles instead of learning to recognize individual shapes. The neural network, therefore, performed the task of extracting the relationship between the significant internal angles of a shape and its classification. The initial training set supplied to the neural network was only a few representative shapes for each category and the network's ability to recognize a wide variety of shapes is enhanced on-line by the user addition of new samples and retraining if necessary.

The BSW algorithm, used for the training purposes of the neural network, provides fast training and this enabled us to include on-line training in the system. With this type of a system, the user can customize the recognition of shapes to his/her drawing style by initially training the system with a few samples and then incrementally training as unrecognized shapes emerge in drawings. However, caution must be taken in order not to teach a shape that cannot be clearly judged to be of either category, as this would result in confusing the recognition ability of the system. Our future work will involve increasing the variety of shapes which the system can manipulate.

References

[1] Andree, H.M.A., et.al., (1993), "A Comparison Study of Binary Feedforward Neural Networks and Digital Circuits," Neural Networks, vol.6, pp.785-790.

[2] Ulgen, F., (1992), Akamatsu, N, "A Fast Algorithm with Guarantee to Learn: Binary Synaptic Weight Algorithm on Neural Networks, " Proceedings of SIMTEC'92, Houston, Texas.

[3] Pavlidis, T., (1978), "A Review of Algorithms for Shape Analysis," Computer Graphics and Image Processing, vol.7, pp.243-258.

[4] Davis, L.S., (1977), "Understanding Shape: Angles and Sides," IEEE Trans. on Computers, vol.C-26, pp.125-132.

[5] Montas, J., (1987), "Methodologies in Pattern Recognition and Image Analysis-A Brief Survey," Pattern Recognition, vol.20, pp.1-6.

[6] Gupta, L. et. al., (1990), "Neural Network Approach to Robust Shape Classification," Pattern Recognition, vol.23, pp.563-568.

[7] Perantonis, S.J., Lisboa, P.J.G., (1992), "Translation, Rotation, Scale Invariant Pattern Recognition by Higher-Order Neural Networks and Moment Classifiers," IEEE Trans. on Neural Networks, vol.3, pp.243-251.

[8] Khotanzad, A., Lu, J., (1990) " Classification of Invariant Image Representations Using a Neural Network," IEEE Trans. on Acoustics, Speech and Signal Processing, vol.38, pp.214-222.

[9] Guyon, I., et. al., (1991), "Design of a Neural Network Character Recognizer for a Touch Terminal," Pattern Recognition, vol.24, pp.105-119.

[10] Kashyap, R.L.(1981), Chellappa, R., "Stochastic Models for Closed Boundary Analysis: Representation and Reconstruction," IEEE Trans. on Information Theory, vol.IT-27, pp.627-637.

[11] Ray, B.K., (1993), Ray, K.S., "Determination of Optimal Polygon from Digital Curve Using L1 Norm," Pattern Recognition, vol.26, pp.505-509.

[12] Kosko, B., (1992), "Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence," Prentice Hall.

Chapter 2:

Neural Network Approaches to Shape from Shading