Cursive Script Postal Address Recognition

By

Prasun Sinha

A Thesis

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Computer Science

1997

Abstract

Cursive Script Postal Address Recognition

By

Prasun Sinha

Large variations in writing styles and difficulty in segmenting cursive words are the main reasons for cursive script postal address recognition being a challenging task. A scheme for locating and recognizing words based on over-segmentation followed by dynamic programming is proposed. This technique is being used for zip code extraction as well as city-state recognition in our system. Results have been reported based on cursive script images from the United States Postal Service(USPS) database. An overview of the cursive script postal address recognition system under development at the IBM Almaden Research Center, is presented.

Optical Character Recognition(OCR) systems are usually trained to minimize character level errors, which does not necessarily guarantee the best results for word level recognition. A scheme for combining character-level optimized classifiers, using a word-level optimization function, to achieve higher word level recognition rates is proposed. The optimum values for the parameters of the combination function are obtained using the gradient descent method. Improvements in word recognition rates on the USPS database by using word-level optimized ensembles of classifiers are reported.

To My Family

Acknowledgments

I would like to acknowledge all the people who have assisted me throughout my graduate studies at Michigan State University and my work on Cursive Script Postal Address Recognition at the IBM Almaden Research Center.

I am extremely grateful to my advisors Dr. Jianchang Mao, Dr. Anil Jain and Dr. Weng, for their guidance and encouragement throughout my Master's and for the invaluable time that they spent with me on the thesis work.

Also, I would like to acknowledge Dr. K. M. Mohiuddin, Mr. S. Gopisetty, Dr. S. Madhvanath and Mr. T. Truong for their encouragement and support during my stay at IBM Almaden.

I would like to dedicate this thesis to my family who have constantly encouraged me to achieve new heights. Without their love, patience, understanding, and support, I would not have finished this thesis. I would also like to acknowledge the help and support I have received from my friends at Michigan State University.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Postal address recognition deals with extraction of the delivery point encoding, e.g., 11-digit zip code, from scanned images of postal addresses [1]. These images have a wide variations in terms of style of writing, whether it is machine-printed, hand-printed or cursive, the kind of instrument used for writing, the size of the characters, etc. Some of the addresses are misspelled and some images are of very poor quality. These are some of the reasons why postal address recognition is a challenging task.

Once the delivery point encoding has been extracted, the letter can be routed without manual intervention. This also implies that an incorrect extraction of destination address encoding may result in a letter being delivered to a wrong address. Hence, minimization of the error rate is very critical and minimization of the reject rate is desirable. For this we need to recognize the street address, the city name, the state name and any zip code, if present. Since there are only a fixed number of values for each of these fields, lexicons of city names, state names, street names and zip codes are used for recognition.

This delivery point encoding is used by sorting machines at the post offices for an automatic sorting of mails. It has been suggested that the consideration of the whole mail chain (the components of the automated system) can lead to a coherent research program. A survey of research projects conducted at the French post office research center (SRTP) of printed and handwritten addresses for small envelopes and flat[1] mails is given in [2]. It has been noted that the aspect of mail processing which is the least developed is sorting the mail in delivery order. This is also supposed to be the part which consumes the most manpower [3]. A summary of the history of postal mechanization in Japan is provided in [4]. Srihari [5] has described a complete system for assigning 9-digit zip codes to US mail-pieces.

This thesis focuses on recognition of US postal addresses written in cursive script. The system is currently being designed to extract the 9-digit destination address encoding. It can later be extended to arrive at the 11-digit zip code, if required. The problem is related to *Off Line Cursive Script Word Recognition*. Large variations in handwritten words as well as overlaps and interconnections between neighboring characters are the major hurdles in solving this problem. Following are some prevalent techniques for off-line cursive handwriting recognition:-

- **HMM[2] on unsegmented words [6] :** One of the common approaches is the sliding window approach where features are extracted from within the window and finally an HMM is used to hypothesize the word. This is not a practical approach for large lexicons.

[1] magazines, big envelopes etc.
[2] Hidden Markov Model

- **Over-segmentation followed by HMM:** In this scheme the segments obtained after segmentation are combined using an HMM. Chen, Kundu and Srihari [7] have developed a variable duration HMM technique which has been tested on unconstrained handwritten words. Some other research groups have also proposed similar approaches [8, 9].

- **Over-segmentation followed by Dynamic Programming:** This scheme is being used by the CEDAR group at SUNY Buffalo [10] and also has been used by Gader et. al. [11]. The performance depends on the results of over-segmentation. The dynamic programming technique tries to combine the over-segmented characters and then matches those characters against the words in the available lexicon. As opposed to the *Over-segmentation followed by HMM* method, over-segments are physically combined in this scheme, and features are extracted by the underlying OCR engine.

The postal address recognition system being developed at the IBM Almaden Research Center is based on the *Over-segmentation followed by Dynamic Programming approach*. This thesis deals with several components of this system and it is related to the problem of recognition of cursive script addresses.

The design and implementation of the over-segmentation algorithm, and the dynamic programming technique for locating and recognizing a word was done as a part of this thesis. The dynamic programming technique was used successfully for zip code recognition and for city and state names recognition. Street number, street name and P.O. box number recognition is currently under testing, but they are all

based on the same technique. At the heart of this technique is the OCR engine. We discovered that optimization of the character level recognition for OCR may not guarantee higher word level recognition rates, which is our ultimate goal (City name, Street name, etc. can be all viewed as words). Hence, several ways of combining neural networks to optimize word level criteria based on word level recognition were tried. These experiments resulted in several word level optimized classifiers.

1.1 Thesis Outline

The outline of the thesis is as follows. Chapter 2 presents an overview of the postal address recognition system. The features used for character recognition and the neural networks used as classifiers are discussed in Chapter 3. The over-segmentation algorithm is described in Chapter 4. Chapter 5 presents the technique used for recognizing zip codes in US postal addresses. A similar technique for recognizing city names and state names is presented in Chapter 6. A new technique for combining the results of individual classifiers based on word level recognition is described in Chapter 7. Chapter 8 presents the summary, conclusions and some ideas for future research.

Chapter 2

System Overview

2.1 Introduction

Our postal address recognition system has images of mail-pieces coming in as inputs and it is supposed to produce the destination address encoding as output. The first step is to locate the destination address block (DAB) on the image, which is done by the Address Block Location (ABL) module.

The image obtained from the camera is a gray level image and it is passed through some pre-processing stages. Since the address is usually written as dark pixels on a light background, it is believed that a binary image, provided the image is of good quality, does not result in a significant loss of information. Hence, the image is binarized which is followed by skew correction. The lines in the image are then separated using a line separation algorithm. The address block is then analyzed to identify the address as machine printed or hand written. The machine printed text is recognized using a different technique, which is outside the scope of this thesis. This

is followed by connected components analysis and slant correction of the connected components. The large connected components are over-segmented using the over-segmentation algorithm, discussed later in the thesis. The next module extracts the 5 or 9-digit zip code from the address, if present. Currently, the module rejects the mail piece if the zip code is not extracted and sends it to the Video Coding System (VCS), which is discussed later. The ordered list of zip codes produced by this module is then compared against a lexicon of valid zip codes. This process typically eliminates more than half of the extracted zip codes. The city and state names for the corresponding zip codes is then enlisted and is matched to the address. Both the city-state recognition and the zip recognition use a dynamic programming technique for matching lexicon entries with a given sequence of over-segments. The city-state recognition module reorders the list of city-state pairs. The first two entries from the reordered list are used to generate all possible street names for those cities. Currently, the same dynamic matching scheme is under test for street level recognition. The matching scheme uses a classifier (OCR module), which in our case is a 1-hidden layer neural network. Better classifiers based on word level training have also been designed (Chapter 7). The recognition modules use various lexicons such as zip code dictionary, city-state dictionary, etc.

All the rejects from these modules are sent to the VCS where the destination address encoding is entered manually.

For every word, we define the probability of a character to be at the beginning of a word as the starting probability of that character, and the transition probability from a character c_1 to a character c_2 is defined as the probability of occurance of c_2

following c_1 in a word. These probabilities are computed separately for street names, city-states and zip-codes. Also the transition probabilities for zip codes are position dependent, which means that the transition probability from c_1 to c_2 also depends on the position of the character c_1 in the word.

In the training phase of the system, the system learns the neural network weights and computes the starting probabilities and the transition probabilities. The over-segmented characters are manually truthed. These over-segments are then combined to form characters which are then normalized in size. Features are extracted from the normalized bitmaps and the neural networks (digit-only, alphabet-only and general) are then trained using the back propagation algorithm.

The training phase of the system is shown in Figure 2.1 and the testing or the running phase of the system is depicted in Figure 2.2.

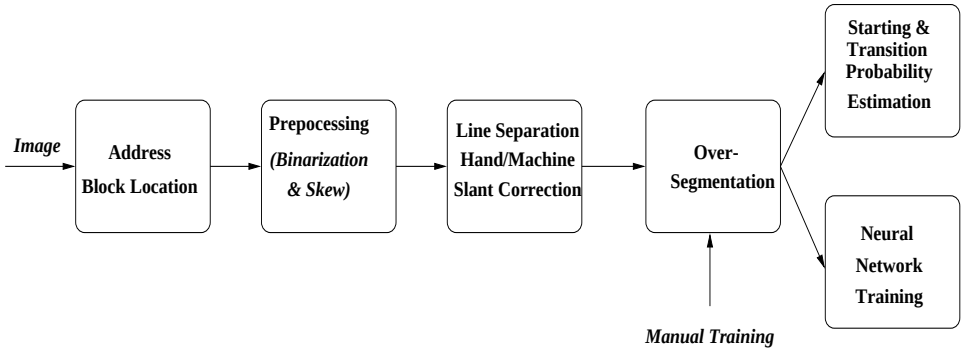


Figure 2.1: **The training phase of the system** *The System for training the Neural Network classifiers is shown here. The starting and transition probabilities of characters and digits are also estimated in the training phase.*

2.2 Modules

This section describes the modules of the system in some more detail.

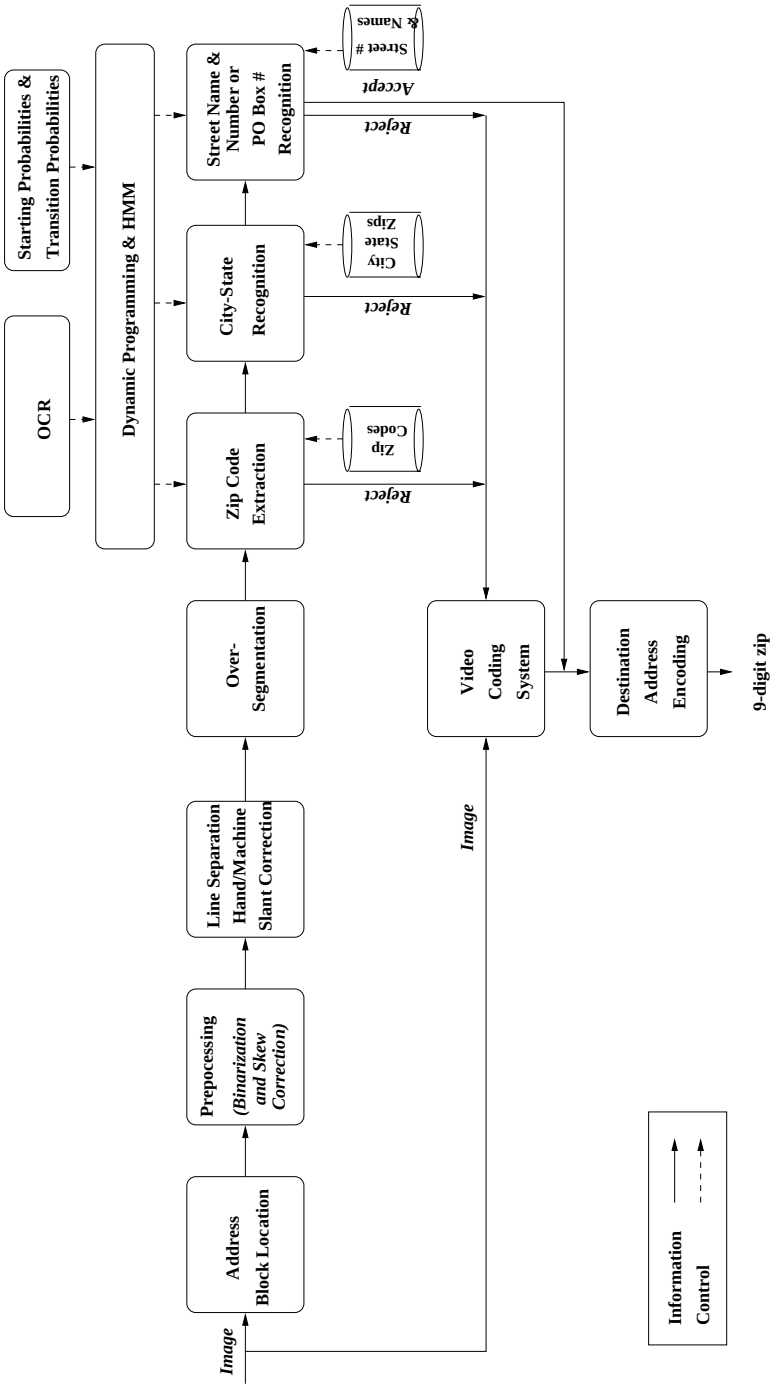


Figure 2.2: **System overview** *The input to the system is the gray scale image of the mail-piece and the output is the destination address encoding*

2.2.1 Camera Setup

A camera is used for taking an image of the mail-piece. A bar-code scanner is also attached which tries to detect if the destination address is already encoded as a bar-code on the mail-piece (which is usually the case for non-personal mails). If the bar-code is not present on the mail-piece then only the image is sent to the postal address recognition system.

2.2.2 Image Binarization

A binary image has sufficient information for extracting the destination address. An adaptive thresholding [12] technique is used for binarization. In this technique, the threshold used for binarization at each pixel is a function of the grey levels of the neighboring pixels.

2.2.3 Address Block Location (ABL)

Given an image of a mail-piece, this module locates the destination address. Though the source address is supposed to be written on the upper left corner of the mail-piece, many mail-pieces do not follow this convention strictly and hence, it may be confused with the destination address. Presence of stamps, post-marks, printed logos, advertisements and other information on the mail-piece make ABL a difficult task. The technique for ABL proposed by Yu, Jain and Mohiuddin [13] is being used in our system.

2.2.4 Skew Detection and Correction

Skew is detected in an address block using projections in several directions. It is then corrected using the classical technique of two shifts along the horizontal direction and one shift along the vertical direction.

2.2.5 Line Separation

It is also done using horizontal projection of the image. The separation of the image into lines is done by a labeling algorithm which labels the pixels where the labels denote the line numbers. All the pixels having the same label are put in the same line.

2.2.6 Hand Written or Machine Printed

There are several differences between hand written and machine printed text. In machine printed text, the baselines are usually straight, the uppercase characters are of uniform height and the lowercase characters are also of uniform height. Also, the spacing between characters or connected components is fairly uniform in machine printed text as opposed to hand written text.

Because of these differences in the two kinds of texts, different techniques are used for hand written and machine printed address recognition. Hence, we need to distinguish between the two types of text. Our system employs a simple discriminator based on the distribution of the heights and widths of connected components [14].

2.2.7 Connected Components Analysis

The connected component analysis is done by a two-pass labeling algorithm. The pixels in the same connected component are assigned the same label.

2.2.8 Slant Detection and Correction

The slant is computed by projecting black pixels along several directions. Care is taken so that only straight segments contribute to the computation of the slant. This helps in forming a sharp peak in the direction of the true slant. This slant detection step is fast as it needs to make only a single pass of the black runs.

The slant correction is performed by shifting the rows of the image in the horizontal direction. Let the detected slant angle be θ^o and the distance of the row from the center-row of that line be y, then the shift required by that row is given by $x = y \tan\theta$. Since, the image is stored in the form of horizontal black runs, shifting is done efficiently by changing the starting and ending points of the black runs.

2.2.9 Over-segmentation

As discussed before, machine printed text typically has several uniform features such as uniform spacing between characters, uniform size for lowercase characters and uniform size for uppercase characters. Since these uniformities are not present in hand written text, it is difficult to expect a perfect segmentation algorithm. Hence, in our system we use an over-segmentation algorithm, followed by a dynamic programming technique for combining the over-segments to form characters.

The goal in this step is to recognize all the segmentation points at the cost of some false alarms. The lesser the number of false alarms, the better the over-segmentation scheme.

Our over-segmentation scheme is based on the contour features of the connected components. While following the contour in the clockwise direction from the leftmost to the rightmost pixels, smooth valleys, sharp valleys and horizontal stretches are identified in a single traversal. The lowest point in the valley and the middle of the horizontal stretches are used as segmentation points. The corresponding points on the lower contour are subsequently found and then screening is done to drop some cuts[1] which can not possibly be separating two different characters. This process is explained in detail in the next chapter.

2.2.10 Optical Character Recognition (OCR)

Our system uses 1-hidden layer neural networks as character classifiers. If a given word is known to contain only digits (e.g, zip code) or only characters then the digit-only or character-only classifier is used for better recognition. All these neural networks are 1-hidden layer neural networks and have been trained using the back propagation algorithm. We have also developed and tested combinations of more than one neural networks which optimize the word level classification error. This work is presented in Chapter 7.

[1] a point on the upper contour and a point on the lower contour together represent a cut.

2.2.11 Zip-code Extraction

A zip code (5 or 9 digits) has the city or state encoded in it. We know that a lexicon-assisted recognition typically has a better performance than if a lexicon is not used. But the number of city-state pairs is so large (roughly 40,000) that it becomes difficult to directly take advantage of it. The zip extraction stage simplifies this task by producing a list of possible zip codes using a lexicon for valid zips.

A dynamic programming implementation of a Hidden Markov Model (HMM) based technique is used for zip location and for identifying the digit boundaries. We try to locate a 5-digit zip code or a 9-digit zip code in the address lines starting from the bottom most line and going upwards. If a zip code is found then a digit recognizer is used to enlist the top choices for the zip code (5-digits or 9-digits). This list of zip codes is then screened using the lexicon of valid zip codes and a lexicon of possible city-state names is generated dynamically. Currently, the system rejects the mail if no zip is recognized and it is handled by the Video Coding System (VCS), discussed later.

2.2.12 City-State Recognition

The entries in the dynamically generated city-state lexicon are matched against a set of over-segments which is hypothesized to contain the city and state names. The matching process is very similar to the one used for locating and recognizing the zip code. As a result of this dynamic matching process the lexicon of city-state names is re-ordered based on the match values.

2.2.13 Street Level Recognition

We have proposed for using the top two city names from the previous step to generate all the possible street names. Then these street names can be matched using the same technique as was used for city-state recognition.

The street number can be recognized in the same way as the city-state is recognized. If we have a database of valid street numbers for every street then it will aid the street level recognition. But, since the street number and name is present towards the middle of the address, it is difficult to locate it compared to the zip code, which is typically found at the end of the address.

Also some addresses have post box numbers but no street numbers which have to be taken care of.

2.2.14 Destination Address Encoding

The 5-digit zip code encodes the city, state, postal sectional center facility (SCF) and the post office. The four-digit add-on number identifies a geographic segment within the five-digit delivery area such as a city block, an office building, an individual high-volume receiver of mail, or any other unit that would aid efficient mail sorting and delivery. This 9-digit zip code is referred to as the Destination Address Encoding, since it encodes the destination address. USPS has added two more digits to the 9-digit zip code mainly for purposes of identifying address upto the apartment number, but our system is presently being designed for generating upto 9-digit zip code only.

2.2.15 Video Coding System

This system handles all the mail-pieces rejected by any of the earlier modules. The image of the mail-piece is displayed on the terminal and the destination address encoding is done manually. The operator has access to city, state, zip code and street level lexicons, which aid in a manual encoding of the address. Also, the system supports several image operations such as rotation, zooming and binarization to make it easier for the operator to read the address.

2.3 Contributions of the Thesis

This thesis has made contributions to several major modules of the system. In particular, the following modules have resulted from the work done in the course of this thesis.

- *Over-Segmentation:* The algorithm was designed and implemented as a part of the thesis.

- *Dynamic Programming and HMM:* This scheme was developed for matching a given lexicon of words with a given word image. A generalized version of it has been implemented and is being used for zip code, city name, state name, street name, street number and P.O. box recognition.

- *Zip Code Extraction:* When a 5-digit zip code is requested, the above technique returns several possible 5-digit zip codes. Similarly, for a 9-digit zip code, the system returns several possible choices. A lexicon of valid zip codes and the

matching confidences are used to arrive at a list of possible zip codes for the given address.

- *City-State Recognition:* The valid zip codes extracted in the previous step are used to generate a list of possible city and state names which are matched with the hypothesized subsequence containing the city-state names, using the same dynamic programming technique.

- *Word Level Training of the Classifiers:* This is implemented as a part of the OCR module to maximize word-level recognition.

Chapter 3

Character Classifier

3.1 Introduction

OCR has been a very active field of research since the mid 1950's. The two main subtasks in designing an OCR are (i) to select a feature set which has a good discrimination power, and (ii) a mechanism to use the feature set to arrive at the final classification of the input pattern. Factors like size and composition of the training data, the initial values of the parameters used for learning and the error estimation criteria also effect the design of a classifier. An excellent survey of feature extraction methods is provided in [15]. Since the character classifiers can be designed in so many different ways, there doesn't seem to be any clear winner. Researchers have suggested using combinations of classifiers to utilize the discriminant power of individual classifiers. Mao and Mohiuddin [16] have suggested using multiple classifiers for better OCR results. A study by Madhvanath and Govindaraju [17] suggests that for large lexicons, using a serial classifier is a good option.

The dynamic programming technique used for extracting zip codes or recognizing city and state names uses an OCR system to generate a posteriori probabilities for segments. The overall performance of the matching scheme, therefore, depends a lot on the performance of the OCR system. In this chapter the OCR method used in our system is discussed. The issue of combining classifiers is discussed in Chapter 7.

3.2 Features

From the input camera, we obtain a gray scale image, but at a high resolution and for good quality images, a binary image does not have significantly less information than the gray scale image. We need to extract features for a given grapheme or segment (a combination of graphemes). These graphemes are obtained by the over-segmentation scheme discussed in Chapter 4.

Various kinds of features such as those based on projection, template matching, image transform, moments, etc. have been tried for cursive script graphemes and segments. We have found that the following feature set consisting of 108 features is a good feature set for the 1-hidden layer neural net classifiers used as character classifiers in our system.

These 108 features can be categorized into the following two types.

- *88 Contour Direction Features [18]:* These features are extracted by dividing the normalized 24×16 image (input normalization reduces intra-character variability, thereby simplifying character recognition) into diagonal and rectangular zones and computing histograms of boundary directions in these zones as shown

in Figure 3.1. The vertical projection has four slices and the others have six slices (22 slices in total). The image is scanned by 2×2 masks which have between one and three black pixels to detect primitive features in each of the zones. The primitive features are grouped into four categories based on the contour directions (horizontal, vertical and two diagonal directions). Histograms are computed by summing the number of primitive features detected along the four contour directions for each of the 22 zones. This results in a total of 88 features (22 slices × 4 directions). These features are then normalized by dividing the 88 features by the maximum possible values of those features.

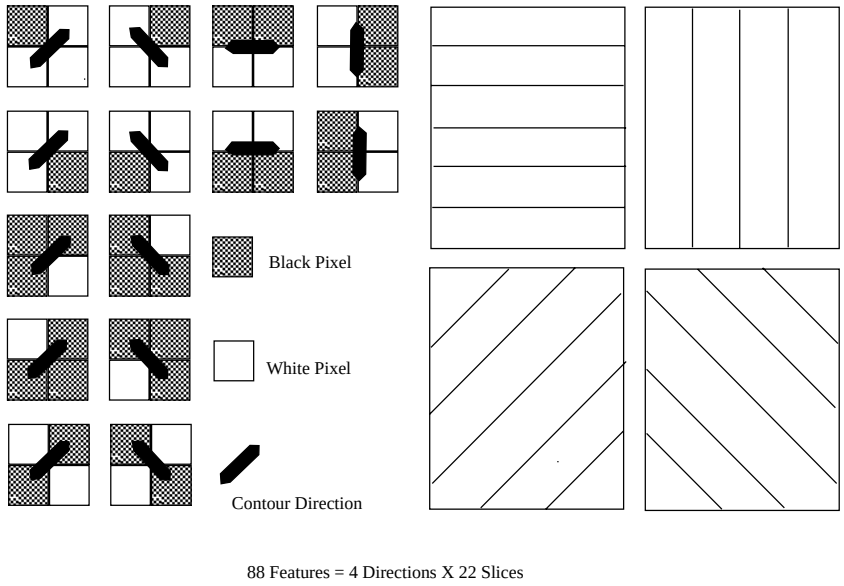


Figure 3.1: **Features based on contour directions**

- *20 Cutting Point Features:* The shape of a character is typically influenced by the preceding and the following graphemes. These 20 features which were suggested by Mao [14] add some contextual information.

  Let s_0 be the segment for which features are being computed, s_{-1} be the pre-

ceding and s_1 be the following graphemes. Let (s_i, s_j) be the segment obtained by joining s_i and s_j.

The following 4 features capture the relative size information :-

$$\frac{width(s_0)}{width(s_0, s_1)}, \frac{width(s_0)}{width(s_{-1}, s_0)}, \frac{height(s_0)}{height(s_0, s_1)}, \frac{height(s_0)}{height(s_{-1}, s_0)}$$

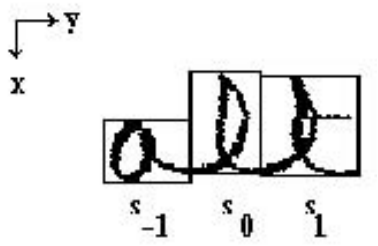


Figure 3.2: **An example of segments s_{-1}, s_0 and s_1 for the 20 cutting point features:**

Let c_1, c_2, c_3 and c_4 be the cut points, if s_0 was connected on both sides before over-segmentation. c_1 and c_2 are the points on the upper contour and the lower contour respectively at the boundary of s_{-1} and s_0. Similarly, c_3 and c_4 are the points on the upper contour and the lower contour respectively at the boundary of s_0 and s_1. Let c_i represent either c_1 or c_2. For each c_i, the following features are computed (if s_0 was not connected to the left side before segmentation, these 8 features are all set to 0) :-

- $\frac{c_i.x}{height(s_{-1},s_0)}$ and $\frac{c_i.y}{width(s_{-1},s_0)}$, where the origin is the upper left corner of (s_{-1}, s_0).
- $\frac{c_i.x}{height(s_{-1})}$ and $\frac{c_i.y}{width(s_{-1})}$, where the origin is in the upper left corner of (s_{-1}).

Similarly, let c_j represent either c_3 or c_4. For each c_j, the following features are computed (if s_0 was not connected to the right side before segmentation, these 8 features are all set to 0) :-

- $\frac{c_j.x}{height(s_0,s_1)}$ and $\frac{c_j.y}{width(s_0,s_1)}$, where the origin is the upper left corner of (s_0, s_1).
- $\frac{c_j.x}{height(s_1)}$ and $\frac{c_j.y}{width(s_1)}$, where the origin is in the upper left corner of (s_1).

3.3 Character Classifiers

Multilayer feed-forward networks have been used for classification in a large variety of applications and also have been used extensively for OCR. Several architectures with different number of hidden layers and interconnections are possible for neural networks. The feature set used as the input determines the performance of the neural network. One simple way is to use the neural network both as a feature extractor and a classifier, by giving a low resolution (scaled down) binary image of the character as input to the neural network.

A 2-layer standard feed-forward network with an arbitrarily large number of hidden units has been shown to be capable of forming an arbitrary decision boundary and approximate any continuous non-linear function. It automatically builds an internal representation of patterns although it is often difficult to interpret these representations. The neural network classifiers used in our system are standard feed-forward neural networks with one hidden layer (see Figure 3.3) trained using the back-propagation algorithm [19, 20].

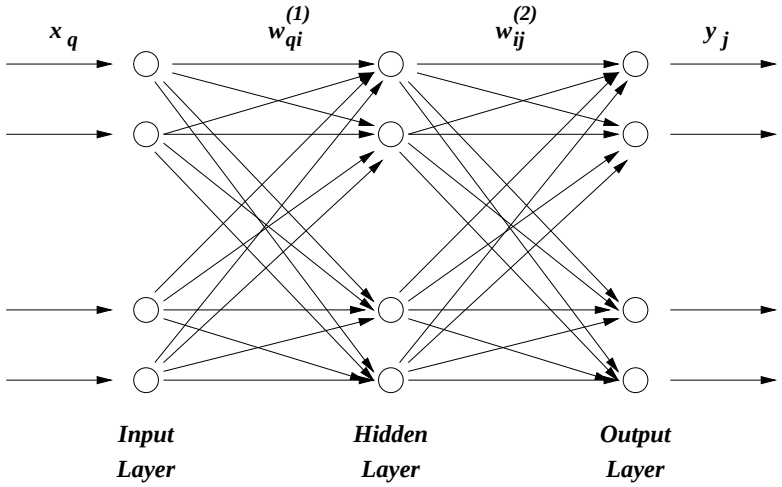


Figure 3.3: **A 1-hidden layer feed-forward neural network**

In our system we are using 1-hidden layer neural networks for OCR, mainly for its simple structure and fast trainability. It uses the feature set discussed in the previous section.

3.3.1 Cost Function

Let $\{(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \cdots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)})\}$ be a set of p training patterns (input-output pairs), where $\mathbf{x}^{(i)} \in \mathbf{R}^n$ is the input vector in the n-dimensional pattern space, and $\mathbf{d}^{(i)} \in [0,1]^m$, an m-dimensional hypercube. For classification purposes, m is the number of classes. In our case, p is the number of characters used for training. Let $\mathbf{y}^{(i)}$ be the output vector of the neural network corresponding to the input vector $\mathbf{x}^{(i)}$. These notations have been taken from [21]. The squared error cost function ($E^{(1)}$) most frequently used in the Artificial Neural Network literature is defined as :-

$$E_i^{(1)} = \| \mathbf{y}^{(i)} - \mathbf{d}^{(i)} \|^2 \tag{3.1}$$

$$E^{(1)} = \frac{1}{p}\sum_{i=1}^{p} E_i^{(1)} \tag{3.2}$$

The Kullback-Leibler distance $E^{(2)}$ is given by :-

$$E_i^{(2)} = \sum_{j=1}^{m} d_j^{(i)} \log \frac{d_j^{(i)}}{y_j^{(i)}} \tag{3.3}$$

$$E^{(2)} = \frac{1}{p}\sum_{i=1}^{p} E_i^{(2)} \tag{3.4}$$

The Kullback-Leibler distance assumes that the output of the neural network can be interpreted as an a posteriori probability for the input normalized image to belong to the corresponding class. Since the softmax function (defined later) normalizes the output probabilities to 1, it is used as the activation function at the output layer when the Kullback-Leibler function is used as the error cost function.

When the Squared Error Criteria is used as the error cost function, the activation function at the output layer in our system is the sigmoid function. In all our networks, we are using the sigmoid function as the activation function at the hidden layer units.

3.3.2 Activation Functions

Let us consider a network with no hidden layers where $\{x_1, x_2, \cdots, x_n\}$ denote the n units of the input layer and $\{y_1, y_2, \cdots, y_m\}$ denote the m units of the output layer. Let w_ij be the weight of the edge connecting x_i to y_j. The hidden layer units and the output layer units of our neural networks use the following kinds of activation

functions :-

- *Sigmoid Function:* The standard sigmoid function is given by $f(x)$ in Equation (3.6).

$$y_j = f(\sum_{i=1}^{n} w_{ij}x_i + w_0), \text{ where } w_0 \text{ is a constant} \quad (3.5)$$

$$f(x) = \frac{1}{(1+e^{(-\beta x)})}, \text{where } \beta \text{ is the slope parameter} \quad (3.6)$$

It is used as the activation function in the hidden layers in our neural networks. Since the sigmoid function lies between 0 and 1, the outputs of the neural network units using the sigmoid function can be interpreted as a probability, but the problem is that all the outputs don't sum to 1. This is taken care of by the Softmax function. In our networks we are using the Sigmoid function also at the output units when the Squared Error criteria is used.

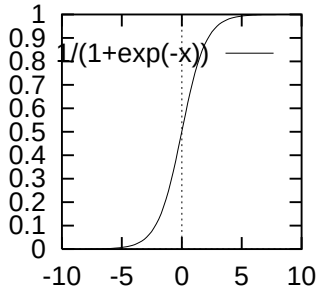


Figure 3.4: **The standard sigmoid function** $f(x) = \frac{1}{(1+e^{(-\beta x)})}$**:** This plot is for $\beta = 1$

- *Softmax Function:* The standard softmax function is given by $g(x)$ in Equation (3.9).

$$h_j = \sum_{i=1}^{n} w_{ij} x_i + w_{0j}, \text{ where } w_{0j} \text{ is a constant} \quad (3.7)$$

$$y_j = g(h_j) \quad (3.8)$$

$$g(h_j) = \frac{e^{h_j}}{\sum_{k=1}^{m} e^{h_k}} \quad (3.9)$$

From the above equations we can derive the following properties:-

$$\sum_{j=1}^{m} g(h_j) = 1 \quad (3.10)$$

$$0 < g(h_j) < 1 \quad (3.11)$$

The above two conditions make the y_j appear like probabilities. When the Kullback-Leibler function is used as the error function, it prefers the neural network outputs to be interpretable as probabilities. Hence, when the Kullback-Leibler function is used in our neural networks, the Softmax function is used as the activation function at the output units.

3.3.3 Neural Network Layout

Our system uses the following three kinds of single layer feed-forward neural networks:-

1. *General Classifier:* It has 108 inputs units, 120 hidden layer units and 43 output units. 26 of these outputs are for alphabetic characters (the lowercase and uppercase letters have been merged), 9 are for digits (0 is combined with the class o), 7 for special symbols (,-./#()) and one output for reject.

2. *Digit Classifier:* It has 108 input units, 50 hidden layer units and 11 outputs, corresponding to 10 digits and one output for reject.

3. *Alphabetic Classifier:* It has 108 input units, 80 hidden layer units and 27 outputs, corresponding to 26 alphabetic characters and one output for reject.

We have trained two neural networks corresponding to the two error functions described earlier, for each of the above three neural network configurations.

The number of hidden units for each of these networks is selected such that the network does not overfit the training data (e.g, for larger number of hidden units) and also has enough hidden units to approximate the decision boundaries. The training phase consists of several epochs (typically 40 is sufficient for all our networks). In every epoch the characters are presented to the network one by one. For every character, the 108 features, the current weight vectors of the neural network and the ideal output vector are used to compute the partial derivative of the cost function (Equation (3.1) or (3.3)) with respect to the weight vectors. These derivatives are used to update the neural network weight vectors.

For training the neural network on non-characters, adjacent two or three segments which do not make a character taken together, are combined. Since, the number of such patterns which can be generated is much larger than the true characters, we

sub-sampled the set and used it for negative training.

The neural networks are trained until the error function reaches a minima.

3.4 Character Level Classification Performance

As discussed in the previous section, six different neural networks were trained using the back-propagation algorithm. For the digit classifier, 68464 characters are used for training and 34002 characters are used for test. For the alphabetic classifier, 136647 characters are used for training and 69061 characters are used for test. For the generalized classifier, 218617 characters are used for training and 109746 characters are used for test. The performance figures for these networks on the corresponding training and test data sets are shown in Table 3.1. From the table, we observe that the digit-classifiers have higher character recognition rates compared to alphabetic or generalized classifiers. This can be explained by the clarity with which digits are written compared to other characters on mail-pieces. Moreover, the number of outputs for the digit network is 11 which is less than the number of outputs for the alphabetic classifier (27) and also less than the number of outputs for the generalized classifier (43).

3.5 Summary

The 108 features used by our classifiers were discussed in this chapter. Our system uses single layer networks for the general, digit-only as well as alphabet-only classifiers.

Table 3.1: **Character level performance:** *Results of character level recognition on a training and test set for a digit classifier, an alphabetic classifier and a generalized classifier is presented in this table for two different error functions.*

	Digit	*Alphabetic*	*Generalized*			
	Training	*Test*	*Training*	*Test*	*Training*	*Test*
Square Error	95.63%	94.32%	73.58%	73.09%	75.80%	73.36%
Kullback-Leibler	95.62%	93.58%	69.39%	68.91%	73.52%	71.16%

Two cost functions namely, Squared Error and Kullback-Leibler distance metrics were discussed. Both these distances are based on character classification errors. In the chapter on combining classifiers (Chapter 7), we will see how these character-level optimized networks can be combined so as to optimize word level criteria. Our final goal is to minimize the errors at the word-level (e.g, recognition of city, state, street name etc.) and character level optimization doesn't necessarily achieve that goal.

Chapter 4

Over-segmentation

4.1 Introduction

Segmentation refers to the determination of character boundaries for a given image of a line of text or a word. Enormous variations in writing styles for cursive handwriting make cursive handwriting recognition a difficult task and hence, a perfect segmentation can't be expected even from a good segmentation algorithm.

It has been found that the motion of the pen results from the following three types of forces:-

- Active forces (due to muscular activity).
- Viscosity (of muscles and articulations).
- Inertia of arm and wrist.

The variation of these forces across writers is one of the main reasons for variation in writing styles. Also the ligatures (pen strokes joining adjacent characters), pro-

vide the writer with another dimension of variation. These factors make the task of segmentation and recognition challenging.

Bengio et al. [22] have proposed a segmentation scheme in which cuts are generated with varying confidences. An HMM is subsequently used to select the best sequence of cuts. In the handwriting recognition system discussed in [23], the segmentation scheme finds several segmentation points and then the one which results in good character recognition for most of the characters is selected. A greedy algorithm for recognition-based segmentation has been proposed by Cohen et al. [24]. In their scheme, if a given segment is not recognized with a certain minimum confidence then it is further segmented and this process is repeated recursively till the segments have been well recognized or till they become too small to be segmented. Another recognition-aided iterative method has been proposed by Srihari [1]. It is an iterative method in which identified characters are removed at each iteration and unidentified segments are further segmented. Strathy and Suen [25] have proposed yet another recognition driven greedy method for segmentation. It separates the leftmost character from a connected component, recognizes it and then does the same recursively on the remaining string. An excellent survey of strategies in character segmentation was provided by Casey et al. [26].

Over-segmentation refers to finding a superset of the perfect segmentation points for a given image of a line of text or a word. The goal is to detect all the segmentation points at the cost of some false alarms. As the number of false alarms approaches zero, the better the over-segmentation scheme becomes. We will use the term *grapheme* for over-segments obtained after over-segmentation.

Several over-segmentation schemes have also been proposed in the literature. Chen et al. [7] have discussed an over-segmentation technique which uses morphological operators to identify segmentation points. Kim and Govindaraju [27] have proposed an over-segmentation scheme which tries to detect regular features, concavities in the upper-contour and convexities in the lower contour to find potential segmentation points. Another heuristic based over-segmentation scheme has been discussed by Gader et al. [11], where the words are segmented into primitives.

Our system uses an over-segmentation scheme discussed in this chapter. Following the over-segmentation, a Hidden Markov Model (HMM) based technique is used to select the character boundaries from the cuts generated by over-segmentation, which is discussed in later chapters. The reason for selecting over-segmentation as opposed to "just right" segmentation is that it is almost impossible to find a perfect segmentation algorithm, as pointed out earlier. Our over-segmentation scheme is mainly based on shapes of the contour of the given word or line image. The algorithm is discussed in this chapter with several examples.

4.2 Over-Segmentation Algorithm

As pointed out earlier, over-segmentation is performed after skew correction, line separation and slant correction have been performed. Then each of the connected components in these lines are over-segmented. Hence, the input to the over-segmentation algorithm is a connected component often along with some smaller connected components which may be noise or dots of i's or pieces of characters in the connected

component. The following is expected from the over-segmentation algorithm :-

- It should segment at all the true character boundaries.
- It should not find more than three graphemes per character.

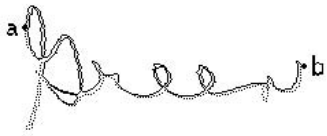


Figure 4.1: **Upper and lower contours: a** *is the leftmost point on the contour and* **b** *is the rightmost point on the contour. The part of the contour from the point* **a** *to the point* **b** *while traversing the contour in the clockwise direction is called the upper-contour and the remaining part is called the lower-contour. In the figure the upper-contour is denoted by the smooth part of the contour and the dotted part refers to the lower-contour.*

The over-segmentation is based on features of the contour. The longest connected component is the one which is segmented and each of the remaining connected components are simply joined with one or the other grapheme based on its proximity to the graphemes. The part of the contour lying between the leftmost pixel on the contour to the rightmost pixel on the contour while traversing the contour clockwise, is referred to as the *upper contour* and the remaining part of the contour would be referred to as the *lower contour* (Figure 4.1). The prospective segmentation points are found while traversing the upper-contour in the clockwise direction. Subsequently, the corresponding points on the lower contour are found. Every pair of such upper and lower contour points is referred to as a *cut*. Finally, some cuts are dropped based on various criteria. These steps are discussed in some more detail.

- **Finding segmentation points by traversing the upper contour:** While traversing the upper contour in the clockwise direction, the following features

are looked for:

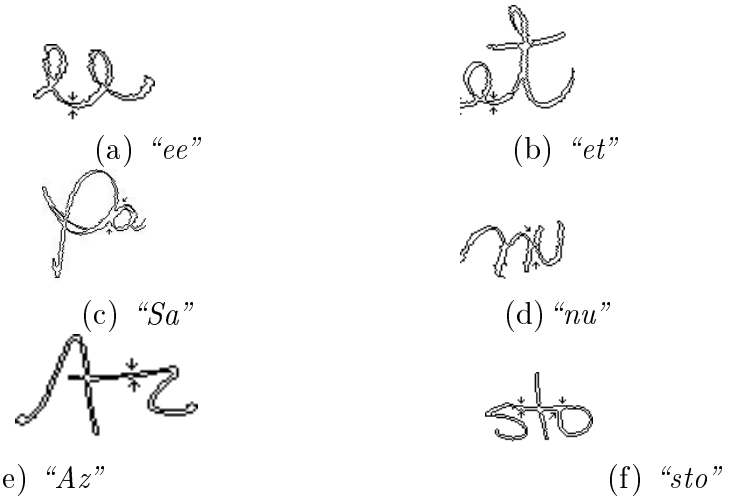
1. *A smooth valley:* Using a state machine, entering, staying and then leaving a valley on the upper contour is kept track of. Most of the cuts for well-written cursive handwriting fall in this category. The lowest point of the valley is selected as the segmentation point.

2. *A sharp valley:* Sometimes, a smooth valley is missing between consecutive characters, but a sharp valley can be found on the contour as shown in Figures 4.2(c) and 4.2(d). Once again, the lowest point in the valley is chosen as the segmentation point.

3. *A horizontal stretch:* Sometimes, characters are joined by a horizontal stretch and not a valley (as in "oo" or "tt" in cursive handwriting). Hence, using another state machine, entering, staying and leaving a fairly horizontal stretch on the upper contour is kept track of while traversing the upper contour. The point in the middle of the horizontal stretch is chosen as the segmentation point.

Figure 4.2 has examples for each of these features on the upper-contour which are used to detect over-segmentation points.

- **Finding corresponding segmentation points on the lower contour:** At this stage, the corresponding points on the lower contour are to be determined. Let us refer to a point on the upper contour found in the previous step by a. The following steps comprise the determination of the corresponding lower



(a) *"ee"* (b) *"et"*

(c) *"Sa"* (d) *"nu"*

(e) *"Az"* (f) *"sto"*

Figure 4.2: **Features on the upper-contour:** *The three main features on the upper contour used to detect cuts are shown in this figure. The relevant cuts are denoted by an arrow in the upper contour and a corresponding arrow in the lower contour. (a and b) Smooth valleys. (c and d) Sharp valleys. (e and f) Horizontal stretches.*

contour point:

1. Traverse downwards from a till a pixel on the lower contour is found. Let that pixel be b.
2. Find the point on the lower contour closest to a and to the left of b. Let that pixel be c.
3. Find the point on the lower contour closest to a and to the right of b. Let that pixel be d.
4. If the contour c-d (contour from c to d) is mostly to the right of c, then the cut a-c is selected, otherwise cut a-d is selected.

- **Screening the cuts:** This step gets rid of some redundant cuts found in the previous steps. First, the cuts are ordered from left to right and then the following inconsistencies are looked for:

1. *Close by cuts:* If two cuts are close by, then the cut with the larger cut-length[1] is dropped.

2. *Overlapping cuts:* If the cuts cross each other then also the cut with the larger cut-length is dropped.

3. *Cuts near the ends:* In cursive handwriting, at the beginning and end of words, typically there is a ligature which forms a nice valley on the upper contour. Since such cuts do not separate any characters in a connected component, they are detected based on the size of the graphemes on the extreme left and right, and are dropped.

4. *Small Graphemes:* Sometimes, noise on the contour can result in detection of a number of cuts for what should just be a single cut. Often, the graphemes resulting from these cuts are small. Hence, if a would-be grapheme bounded by two adjacent cuts is small, then the cut with the larger cut-length is dropped.

5. *Horizontal Overlap and Vertical Overlap:* For every cut, the horizontal and vertical overlaps of its left and right graphemes are computed. If there is a large horizontal overlap and a small vertical overlap, then it is likely that the two graphemes are part of a single character and hence, the cut is dropped.

[1] the length of the cut

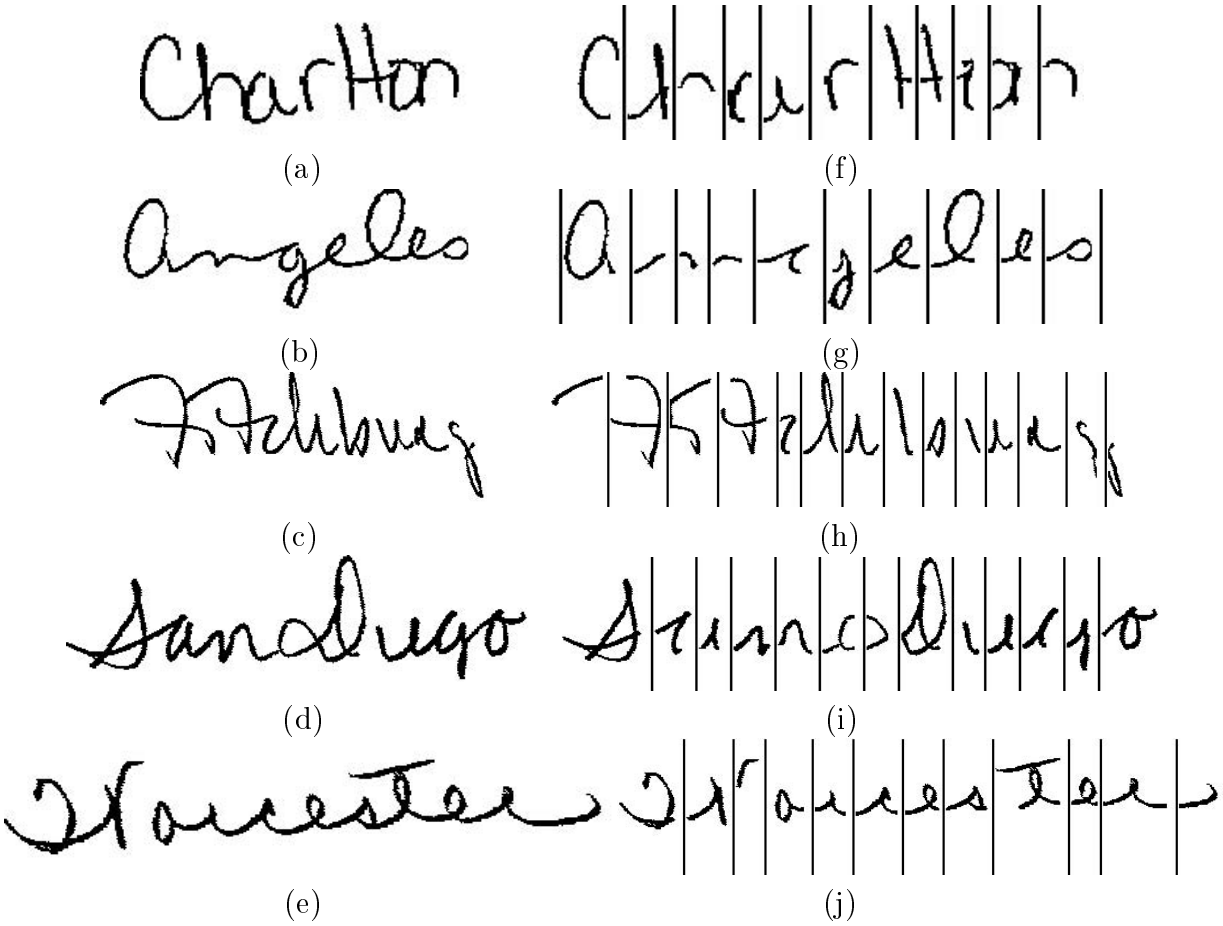


Figure 4.3: **Some examples of over-segmentation using the technique discussed in this chapter:** *The figures from (a) to (e) are the original words and the corresponding results after over-segmentation are shown from (f) to (j). These words were taken from the USPS database.*

Table 4.1: **Frequency of Graphemes per Character:** *This table shows the average number of graphemes found by our over-segmentation algorithm for each of the lower case alphabetical characters. These figures were computed using 1000 cursive postal address images taken from the USPS database.*

	Alphabetical Characters									
	a	b	c	d	e	f	g	h	i	j
# graphemes	1.37	1.48	1.06	1.49	1.12	1.22	1.38	1.72	1.08	1.20

	Alphabetical Characters									
	k	l	m	n	o	p	q	r	s	t
# graphemes	1.81	1.04	2.85	1.80	1.18	1.23	1.00	1.24	1.23	1.17

	Alphabetical Characters					
	u	v	w	x	y	z
# graphemes	1.71	1.64	2.74	1.56	1.63	1.29

4.3 Results

The algorithm discussed above met the requirements stated in Section 4.2 very well. It almost always finds the character boundaries. The average number of graphemes found by the algorithm for lower case alphabetical characters is shown in Table 4.1. The number of characters which were segmented into more than three graphemes is approximately 0.003%. All this data was obtained by performing segmentation on 1000 USPS images, which were manually truthed. The data was collected only for lower case characters. Some examples of over-segmentation produced by the algorithm discussed in this chapter are shown in Figure 4.3.

Chapter 5

Zip Code Extraction

5.1 Introduction

Given an address-block, zip code extraction deals with locating and identifying the zip code, if present in the address. Since, we are concerned with handwritten US postal addresses, and the destination addresses of most US mails contain a 5-digit or sometimes a 9-digit zip code, we are only considering recognition of 5-digit or 9-digit zip codes. The 5-digit zip code encodes the city, state, postal sectional center facility (SCF) and the post office. The 9-digit zip code is formed by adding four digits to the 5-digit zip code, which further denote the post office box number, block face [28], etc. Since, all this information is encoded in the zip code, a robust technique for zip code extraction is desirable for postal address recognition.

The information encoded in the zip code like the city name, state name, street number and P.O. Box is also present otherwise on an address block. This redundancy is beneficial for postal address recognition. In our system, we are interested in gen-

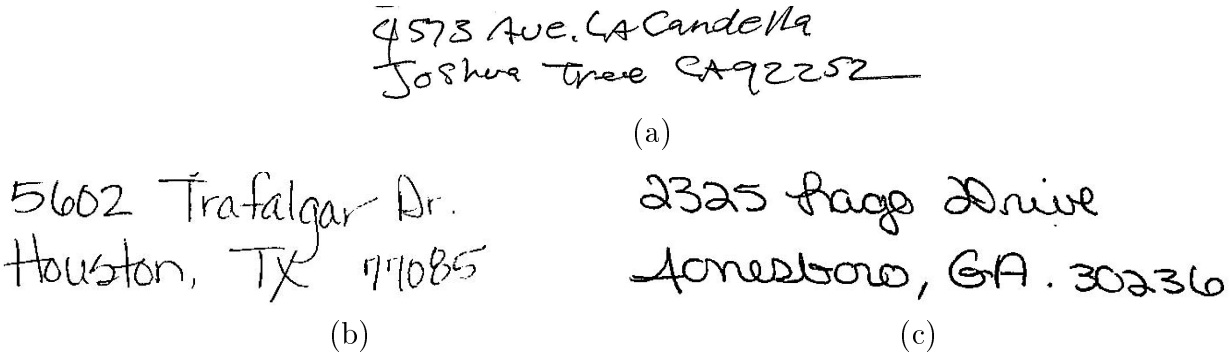
erating a list of possible zip codes, which can further be used to produce a lexicon of possible city-state pairs using dictionary lookup for the zip codes. The city-state pairs can then be verified by a matching scheme which is discussed in later chapters.

Several approaches have been proposed for zip code extraction. Strathy and Suen [26] have proposed a recognition-driven segmentation scheme for recognition of digit strings. Srihari [28] has proposed a recognition-aided iterative method for zip code recognition, which analyzes the image more finely with increasing iteration number. Both these techniques are greedy techniques and the decision for segmenting or combining broken pieces at any point is based on local information such as the proximity of the broken pieces and the recognition result of the individual pieces or their combinations, which may not give the best match for the whole 5-digits or 9-digits. Also, there is no syntax constraint used for zip extraction. Gader et al. [11] have proposed an over-segmentation into primitives, followed by a dynamic programming technique which uses neural fuzzy logic, for zip code recognition.

There are two main steps involved in zip code extraction, namely, zip code location and zip code recognition. Existing zip code extraction methods use various heuristics for locating the digit string representing the zip code in an address block. Typically, the aspect ratio of the connected components, spaces between the connected components [24] and preliminary OCR results are used for locating contiguous digits, which may represent the zip code.

We propose a dynamic programming approach to zip code location which is based on over-segmentation and a Hidden Markov Model (HMM) based zip code recognition technique. As opposed to existing methods which perform OCR first, and then locate

the most likely 5-digits, the proposed method works directly on graphemes obtained after over-segmentation. We will refer to the over-segments as graphemes and will reserve the term segment for combination of two or three graphemes. The syntax constraint is integrated in segmentation, zip code location and recognition. Given a sequence of graphemes, the zip code locator identifies a sub-sequence, along with the information if the graphemes need to be combined, for forming the zip code. The HMM-based approach for refining the zip code recognition, thereafter, identifies the possible zip codes. Both these steps, zip location and zip recognition are holistic, as a global criteria, based on the whole word, is optimized in both the cases. This approach differs from the earlier approaches since locating the zip digits and recognizing the zip string in the given address block are done in a single step by the dynamic programming approach.



(a) (b) (c)

Figure 5.1: **Some examples of addresses from real mail-stream** *(a) The zip digits are not isolated from the state name. (b) Zip digits are broken due to poor scanning or due to badly written digits. (c) The zip digits are touching.*

In handwritten addresses (hand-print or cursive), the zip code may not be isolated from the city or state name or there may be irregular spacing between digits to confuse the zip code locator, if the zip code locator is based on spacing between

connected components. Long ligatures often reduce the gaps between words, and sometimes words are connected. The initial OCR results of the zip code do not necessarily produce 5 or 9 digits because of touching and broken digits. Errors in the initial OCR results (e.g, wrongly splitting and merging) are often irrecoverable. Some interesting cases of addresses for real mail-stream which pose various problems for zip code extraction are shown in Figure 5.1.

Our proposed method locates the zip code based on the best match for 5-digits with the sequence of graphemes, which were obtained as a result of our over-segmentation algorithm. Since, our over-segmentation technique captures most of the segmentation points, our dynamic programming scheme can easily handle touching digits. Broken digits are also taken care of, since we use a global criteria, based on all the digits, to match digits with the graphemes. Since, decision about merging graphemes or broken pieces is based on finding a match for the whole of 5-digits or 9-digits and not just local information, we expect our scheme to work better than zip extraction techniques which use heuristics.

For our discussion, we will consider the last line of the address shown in Figure 5.1(c), which is also shown in Figure 5.2. The results of segmenting this line using the segmentation technique discussed earlier is shown in Figure 5.3.

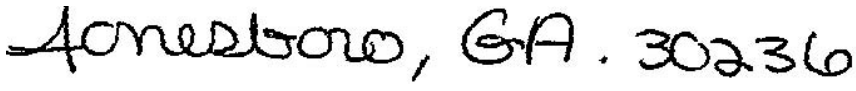


Figure 5.2: **The last line of an address:** *This is the last line of the address shown in Figure 5.1(c). We will work with this example to illustrate the concepts in this chapter.*

To keep our discussion simple, we will consider only 5 digit zip codes. The 9

Figure 5.3: **The result of over-segmentation:** *The graphemes obtained after over-segmentation of the address line shown in Figure 5.2 are shown separated by vertical lines.*

digit zip code can be recognized in a similar way as discussed towards the end of the chapter.

Typically, the zip code is present in the bottom-most line of an address-block. So, we process the address-block line by line, starting from the bottom-most line until the zip code is located with high-confidence or the zip code is not found in the address-block. The graphemes selected from our example (Figure 5.3), are shown in Figure 5.4.

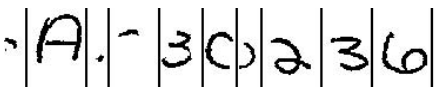


Figure 5.4: **Graphemes selected for zip extraction:** *It is found that the 5-digit zip code, if present on an address line can usually be located in the last ten graphemes of that address line. Hence, the last 10 graphemes from Figure 5.3 have been selected.*

The formalization of zip location and zip recognition problems as graph problems is presented in the next section. The dynamic programming technique for zip location and the HMM technique for zip recognition, which are essentially solutions to these graph problems, are discussed afterwards.

5.2 Zip Location and Zip Recognition as Shortest Path Problems

A sequence of s graphemes can be represented as a graph $G = < V, E >$, where V is the set of vertices and E is the set of edges, with $|V| = s + 1$. Each node can be interpreted as a segmentation point or one of the extreme points in the sequence of graphemes given as input. Let us number these vertices from 1 to $|V|$. Depending on whether we want to solve the problem in a single step or two steps, we have two different ways of interpreting the edges:

1. **One Step Model:** In this case, the solution to the graph problem is a solution to the zip location as well as zip recognition problem.

   There are 11 edges between every pair of nodes $(i, i + 1)$, corresponding to the recognition of digits (0 to 9 and one for deleting the character), where $i = 1 \cdots (|V| - 1)$. Each of the edges has a weight which is the probability of the character corresponding to that edge to represent the grapheme. These values are obtained by using an alpha-numeric classifier. Also, there are 10 edges for every pair of nodes $(j, j + 2)$, where $j = 1, \cdots, (|V| - 2)$ and for every pair of nodes $(k, k + 3)$, where $k = 1, \cdots, (|V| - 3)$, corresponding to the recognition of digits 0 to 9. The weights on these edges correspond to the probabilities obtained by the recognition of the pattern composed of two or three adjacent graphemes.

2. **Two Step Model:** In this case, the solution to the graph problem only needs

to indicate which graphemes comprise the 5-digit zip code. Recognition of these five segments (graphemes or combinations of two or three adjacent graphemes) can be refined using a digit classifier, which will comprise the second step.

For every pair of nodes $(i, i+1)$, where $i = 1, \cdots, (|V|-1)$, there are two edges in the graph; one corresponding to the probability of the grapheme representing a digit and the other corresponding to it being a deletable grapheme.

Also, there are edges $(j, j+2)$, where $j = 1 \cdots (|V|-2)$, corresponding to the recognition of the pattern formed by the combination of the j-th and $(j+1)$-th graphemes, and edges $(k, k+3)$, where $k = 1 \cdots (|V|-3)$, corresponding to the recognition of the pattern formed by the combination of the k-th, $(k+1)$-th $(k+2)$-th graphemes.

To normalize the weights of the edges with respect to the number of graphemes, weights of the edges $(j, j+2)$, where $j = 1 \cdots (|V|-2)$, are raised by a power of 2 and weights of the edges $(k, k+3)$, where $k = 1 \cdots (|V|-3)$, are raised by a power of 3.

Now, let's consider paths from node 1 to node $|V|$. Let the edges in the path be $E = (e_1, e_2, \cdots, e_n)$, where n is the path length. Let $w(e_i)$ represent the weight of the edge e_i and let e_m be the edge representing the first digit for that particular path. The normalized [1] and unnormalized weights of a path are given by the following formulae.

[1] For faster computations, all these calculations were performed in the logarithmic domain. Logarithmic probabilities were used instead of regular probabilities; multiplications became additions and so on.

$$\text{Weight}_{unnormalized} = \prod_{i=m}^{n} w(e_i) \tag{5.1}$$

$$\text{Weight}_{normalized} = (\text{Weight}_{unnormalized})^{\frac{1}{n-m+1}} \tag{5.2}$$

This weight measure ignores the deletion of graphemes from the beginning of the zip code identified by that particular path.

The problem of finding the shortest path from node 1 to node $|V|$ for the One Step Model is equivalent to the zip code extraction problem, and for the Two Step Model is equivalent to the zip code location problem.

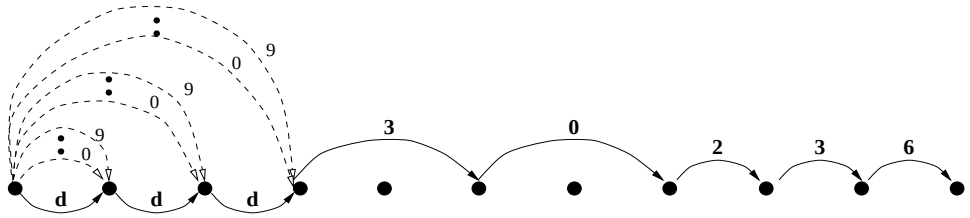


Figure 5.5: **One step model :** *The 11 vertices correspond to the 10 graphemes from Figure 5.4. For clarity, all the edges have not been shown. The best path is shown by bold lines and all the edges from the leftmost vertex are shown by dashed lines. The best path locates as well as recognizes the 5-digit zip code. The first three graphemes are deleted and are marked by* **d***'s.*

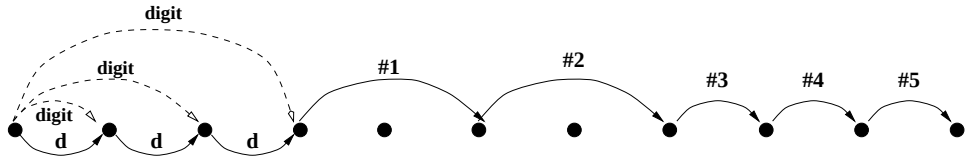


Figure 5.6: **Two step model :** *The 11 vertices correspond to the 10 graphemes from Figure 5.4. For clarity, all the edges have not been shown. The best path is shown by bold lines and all the edges from the leftmost vertex are shown by dashed lines. The best path locates the 5-digit zip code. The first three graphemes are deleted and are marked by* **d***'s and the five graphemes and segments in the best path are marked #1 through #5*

For the graphemes considered in Figure 5.4, the graph for the One Step Model is given in Figure 5.5 and the graph for the Two Step Model is given in Figure 5.6. Both these graphs have $|V| = 11$, as the number of graphemes being considered is 10. The best path and all the edges for the leftmost vertex are shown in these figures.

5.2.1 One Step vs. Two Step Model

The input to both the graph problems is a sequence of graphemes which probably contains a zip code and some extra graphemes at the beginning of the sequence. These graphemes can be part of the city name or state name preceding the zip code. Therefore, an alpha-numeric classifier must be used while locating the zip code. It is well known that alpha-numeric classifiers are not as good as numeric only classifiers (or digit classifiers) in recognizing digits because of the confusion between certain alphabets and digits. Hence, it is more desirable to use two separate classifiers; one for locating the zip code and the other one for recognizing the located zip.

Once the zip has been located, a digit classifier is preferred over an alpha-numeric classifier for the specific job of digit recognition.

Taking this into consideration, we decided to implement the Two Step Model where the zip location and zip recognition are two separate steps and hence, the character and digit classifiers could be used for zip location and recognition, respectively.

5.3 Zip Code Location

Given a sequence of graphemes, which may contain extraneous graphemes in the beginning of the sequence, the goal of zip code location is to identify the graphemes and combinations of graphemes which represent the 5-digits of the zip code, if present in the sequence. This is equivalent to solving the graph discussed under the Two Step Model. A dynamic programming scheme for solving the same graph problem is discussed in this section. This work on zip code location is based on Abayan's work [29].

Let the number of graphemes representing the 5-digit zip code be N. Since the graphemes are obtained as a result of over-segmentation, it can be assumed that $N \geq 5$. The difference of the number of graphemes and 5 represents the slack in matching.

$$\text{Slack } S = N - 5$$

Consider a table with S+1 rows and 5+1 columns. Let $P[i, j]$ denote the probability of the segment, formed by combining graphemes from i-th through j-th in the sequence, being a digit. We will consider four types of edges from one cell [2] to another cell, as shown in Figure 5.7. The top-left cell is $(1, 1)$.

1. An edge from cell $(i, j-1)$ to (i, j) denotes that the $(i+j-2)$-th grapheme corresponds to the $(j-1)$-th digit. Its weight is given by $P[(i+j-2), (i+j-2)]$.

2. An edge from cell $(i-1, j-1)$ to (i, j) denotes that the combination of the $(i+j-3)$-th and $(i+j-2)$-th grapheme corresponds to the $(j-1)$-th digit.

[2] An intersection of a row and a column defines a cell in the table.

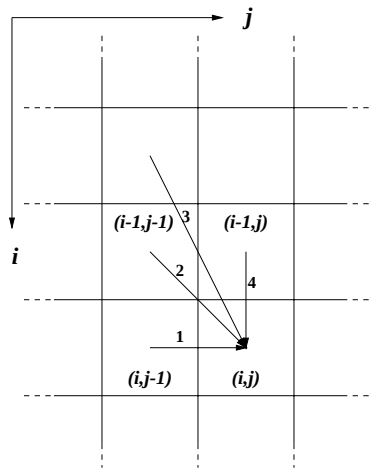


Figure 5.7: **Edges in the dynamic programming table:** *The four types of edges are numbered 1 through 4. (1) It denotes that a grapheme corresponds to a digit. (2) It denotes that two adjacent graphemes are being combined to match with a digit. (3) It denotes that three adjacent graphemes are being combined to match with a digit. (4) It represents that a grapheme is being deleted.*

Its weight is given by $P[(i+j-3),(i+j-2)]$.

3. An edge from cell $(i-2, j-1)$ to (i, j) denotes that the combination of the $(i+j-4)$-th, $(i+j-3)$-th and $(i+j-2)$-th graphemes corresponds to the $(j-1)$-th digit. Its weight is given by $P[(i+j-4),(i+j-2)]$.

4. An edge from $(i-1, j)$ to (i, j) denotes that the $(i+j-2)$-th grapheme is deleted. Its weight is given by P[$(i+j-2)$-th grapheme is a deletable grapheme].

The probability of a grapheme to represent a digit can be calculated using an alpha-numeric classifier or a classifier which distinguishes characters from digits. If we are using an alpha-numeric classifier, then the summation of probabilities for the 10 digits will give the probability of the input segment (grapheme or combination of graphemes) being a digit. Now, consider a path from cell $(1, 1)$ to cell $(S+1, 6)$ consisting of these four types of edges.

Let the edges in a path through the table be $E = (e_1, e_2, \cdots, e_n)$. Let $w(e_i)$ represent the weight of the edge e_i and let e_m be the edge representing the first digit for that particular path. The unnormalized weight of the path is given by Equation (5.1) and the normalized weight is calculated according to Equation (5.2).

The problem of locating the zip code is then equivalent to finding the path with the largest normalized weight from cell $(1, 1)$ to cell $(S + 1, 6)$.

The idea of a table and of a best path (path with the smallest normalized weight) made it simple to put everything together in a dynamic programming approach. The partial results at each of these cells contain the partial best paths. For any cell (i, j), there could be edges from the cells $(i, j-1)$, $(i-1, j-1)$, $(i-2, j-1)$ and $(i-1, j)$. Hence, the best path ending at the (i, j)-th cell requires considering at most four[3] paths coming from those four cells. We can repeat the computation of the best path ending at the cell (i, j), row-wise or column-wise till we reach the cell $(S+1, 6)$, which will contain the path representing the best match for 5 digits.

5.3.1 Dynamic Programming Algorithm

Let T be the dynamic programming table of size $(S + 1) \times 6$. Each entry $T[i, j]$ in the table stores the weight of the best path from $T[1, 1]$ to $T[i, j]$, and also a pointer to the previous cell on the best path. The steps of the algorithm are as follows:-

1. Initialize the best path to null and best path probability to zero in $T[1, 1]$.

2. for i $\leftarrow$ 2 to $(S + 1)$

[3]There would be less number of paths to some cells near the boundary of the table.

Initialize the best path to $(i, 1)$ to be from $(i-1, 1)$ and best path probability to be zero in $T[i, 1]$.

3. for j $\leftarrow$ 2 to $(5 + 1)$

   for i $\leftarrow$ 1 to $(S + 1)$

   Store information about the path from $(i, j-1)$ in $T[i, j]$

   Update $T[i, j]$ according to the path from $(i-1, j-1)$, if it is a valid cell

   Update $T[i, j]$ according to the path from $(i-2, j-1)$, if it is a valid cell

   Update $T[i, j]$ according to the path from $(i-1, j)$, if it is a valid cell

   (The updation is done to always keep track of the lowest normalized weight path (Equation 5.2) or the best path from $T[1, 1]$ to $T[i, j]$.)

4. Follow the pointers from $T[S+1, 6]$ back to $T[1, 1]$ and print the best path.

The best path for the graphemes chosen from our example (Figure 5.4), is shown in Figure 5.8. Since there are 10 graphemes, the slack is $10-5 = 5$. Hence, the size of the table is 6×6. The best path (the path with the lowest normalized weight) from the cell $(1, 1)$ to the cell $(6, 6)$ is shown in the same figure. With a small modification the best m paths from cell $(1, 1)$ to any cell (i, j) can be stored at the cell (i, j). As shown later, $m = 2$ suffices for our purposes.

5.3.2 Implementational Details

- Most of the graphemes and combinations of graphemes are present in more than one path. Hence, we precomputed the probabilities for each grapheme and each

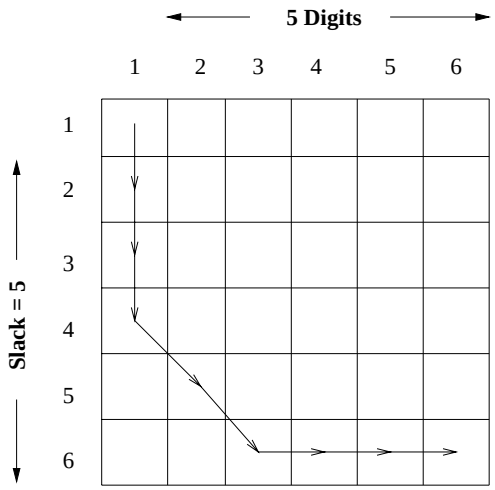


Figure 5.8: **Zip code location:** *The table for the graphemes selected in Figure 5.4 is shown in this figure. As the slack in this case is* $10 - 5 = 5$*, the table has* 6 *(= 1 + Slack) rows, and* 6 *(= 1 + #zip digits) columns for the 5 zip digits. The best path through the table from the cell* $(1, 1)$ *to the cell* $(6, 6)$ *is shown. The path shows that the first three graphemes are deleted (vertical strokes), the 4th and 5th are combined to form the first digit, the 6th and 7th are combined to form the second digit and the 8th, 9th and 10th represent the remaining three digits of the 5-digit zip code.*

segment (combinations of 2 or 3 adjacent graphemes) to represent a digit or to be a deletable grapheme.

- We found the first and the second best paths through the table to be sufficient for covering the correct path through the dynamic programming table. Each of these paths is stored at the cells using back pointers to the previous cell which lies on the best-path. After the algorithm has been executed, the back-pointers can be followed from the cell (S+1,6) back to (1,1), which would represent the best path.

5.4 Zip Code Recognition

The technique described above deals with locating the zip code in an address block. We take the best two results obtained from it and use those results to generate a list of possible zip codes.

If the zip code location algorithm locates the zip code with a certain minimum confidence then we need to identify the possible zip codes, and for that we use a digit-classifier as opposed to a more general alpha-numeric classifier that was used for locating the zip code for reasons discussed in Section 5.2.1. The numeric classifier calculates the probability for each of the graphemes or combination of graphemes to represent one of the 10 digits. To produce candidate zip codes, we use the following two methods.

1. **Hidden Markov Model (HMM) based approach:** In this approach we model the problem of recognizing the 5 digits using an HMM λ. The goal is to identify the zip codes which are the most likely candidates based on the results of the numeric classifier. The naive choice would be to pick up the best digits obtained for each of the 5 graphemes or combinations of graphemes. But the HMM approach also takes into consideration the initial probabilities and the transition probabilities.

   The Viterbi algorithm to find the best path is described in the next section. An extension has been made to it so that we can find the best n paths. We are currently using $n = 30$ for the best path obtained from the dynamic programming method and $n = 20$ for the second best path.

2. **Confusion Table:** To increase our coverage of the true zip code, we used some statistics generated by our numeric classifier.

   The Confusion Table tells if the numeric classifier returns a digit i then besides i, which other character could it have possibly mis-classified. Thus, for every digit, we have a table of three most confused-with digits. The top choice from the Viterbi algorithm was used to generate more candidate zip codes mainly in the following ways:-

   - Each of the 5 digits were replaced in turn by the most confused-with digits, to generate 5 more zip codes.
   - The pair of digits which had the lowest digit recognition confidences was replaced by several combinations of the corresponding confusing digits.

   Other similar heuristics (slight variations of the above methods) were also used to generate some more candidate zip codes. Only the two best zip codes obtained from the two best solutions to the dynamic programming technique were used to generate zip codes using the confusion table.

Using these two methods, a list of possible zip codes is obtained, which is then screened using the dictionary of zip codes. The dictionary of 5-digit zip codes contains roughly 42,000 entries. The possible number of zip codes that can be generated using 5 digits is $10^5 = 100,000$. Hence, on an average, the dictionary verification will get rid of approximately 58% of the zip codes found by the dynamic programming approach.

Table 5.1: **Confusion Table:** *The column labels stand for the results of the numeric classifier and the three rows stand for the three most confusing digits for those results. For example, the column with label "3" shows that when the numeric classifier returns "3", then besides 3, the true digit could be "5", "7" or "2" with probabilities decreasing respectively.*

Confused Digits	*Digit-Classifier Output*									
	0	1	2	3	4	5	6	7	8	9
Most	8	4	3	5	9	3	0	9	5	7
2nd most	2	2	6	7	7	6	4	2	3	4
3rd most	5	7	7	2	6	1	5	3	2	5

5.4.1 HMM based approach for zip code recognition

The recognition of the graphemes or the combinations of graphemes can be modeled as an HMM λ. As a first attempt, we thought that 10 states corresponding to the 10 digits should suffice for λ. But then we realized that in order to incorporate the position-dependent transition probability, each of the 10 states need to be replicated 5 times resulting in 50 states. The notations used are in accordance with Rabiner's tutorial paper [6] on HMM.

λ is characterized by the following elements:

- **States:** Let us denote the set of states as $S = \{S_{ij}\}$, where $i = \{1..10\}$ and $j = \{1..5\}$. Hence the number of states in λ is 50. The states are shown in Figure 5.9. We assume that every state transition takes one time unit. Time starts at $t = 1$, and the state being visited at time t is denoted by q_t.
  - The i-th dimension corresponds to an observation being classified as one of the 10 digits from 0 to 9. The j-th dimension corresponds to the 5 graphemes or segments (combinations of 2 or more adjacent graphemes)

corresponding to the 5-digit zip code, which were obtained from the dynamic programming approach discussed earlier in the chapter.

- **Observation Symbols:** The number of observation symbols per state is $2^{24\times16}$. Let's denote the individual symbols as $V = \{v_1, v_2, ..., v_M\}$, where $M = 2^{24\times16}$.

  - The image size of the character to be recognized is 24×16 after normalization. Since, we are dealing with black and white images, the total number of possible normalized character images, or observation symbols are $2^{24\times16}$.

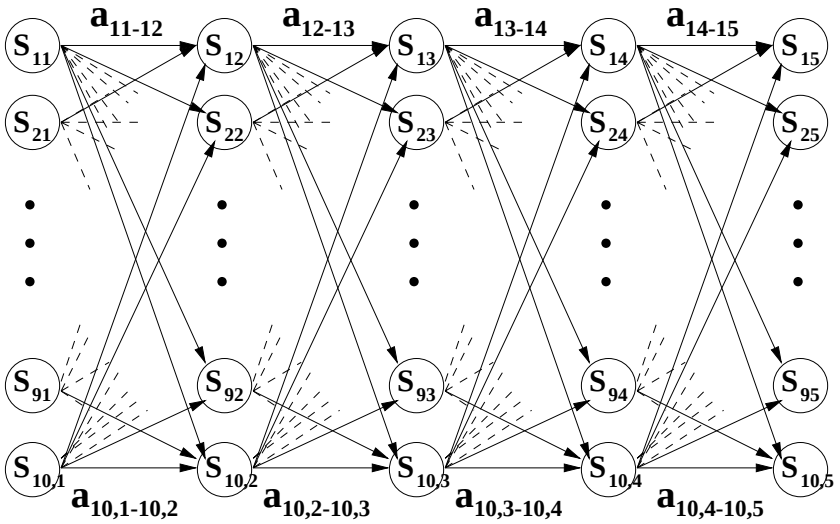


Figure 5.9: **The Hidden Markov Model λ:** *The nodes in this figure correspond to the states of the HMM and the arrows correspond to the non-zero transition probabilities. The rows of states correspond to digits from 0 to 9, and the columns correspond to the 5 observations obtained using the dynamic programming technique.*

- **Transition Probability Distribution:** It is denoted by $A = a_{ij-kl}$. It can be calculated according the following formula for $1 \leq i, k \leq 10$ and $1 \leq j, l \leq 5$.

$$a_{ij-kl} = \begin{cases} P[\text{digit pair } (i,k) \text{ at position } j \text{ in the 5-digit zip code}] & , \quad \text{if } l = j+1 \\ 0 & , \quad \text{if } l \neq j+1 \end{cases}$$

Table 5.2: **Transition Probability Table:** This table shows the transition probability from the first to the second most significant digit in a 5 digit zip code (a_{i1-k2}, where $i = \{1..10\}$ and $k = \{1..10\}$). An entry in row with header i and column with header j represents the probability of the digit pair (i, j) occuring at the beginning of the 5-digit zip code.

1st digit	*2nd digit in 5-digit zip code*									
in 5-digit zip code	0	1	2	3	4	5	6	7	8	9
0	0.02	0.13	0.15	0.08	0.15	0.10	0.12	0.12	0.12	0.00
1	0.11	0.09	0.12	0.10	0.10	0.11	0.08	0.10	0.08	0.11
2	0.08	0.09	0.09	0.09	0.11	0.11	0.08	0.10	0.13	0.12
3	0.12	0.08	0.13	0.15	0.06	0.09	0.09	0.10	0.11	0.07
4	0.08	0.09	0.07	0.10	0.10	0.11	0.10	0.10	0.12	0.12
5	0.11	0.05	0.08	0.09	0.12	0.12	0.13	0.10	0.10	0.09
6	0.11	0.12	0.15	0.11	0.09	0.09	0.09	0.10	0.12	0.03
7	0.09	0.10	0.12	0.08	0.09	0.10	0.10	0.11	0.12	0.09
8	0.15	0.09	0.07	0.14	0.12	0.14	0.04	0.10	0.06	0.07
9	0.10	0.07	0.11	0.10	0.09	0.14	0.06	0.11	0.12	0.10

The transition probabilities for the 10 digits from the first digit to the second digit in the zip code, i.e., a_{i1-k2} for $i = \{1..10\}$ and $k = \{1..10\}$ are presented in Table 5.2. The non-zero transition probabilities are shown in Figure 5.9. These probabilities are calculated from the database of zip codes. Due to lack of information about the frequency of mails to a particular zip code, the values are only a crude approximation to real mail-stream.

- **The observation symbol probability distribution in state S_{ij}:** It is denoted by $B = \{b_{ij}(O_m)\}$, where O_m is an observation at time m.

$$b_{ij}(O_m) = P[O_m \mid q_t = S_{ij}], 1 \leq i \leq 10, 1 \leq j \leq 5.$$

It may seem that $b_{ij}(O_m)$ will be expensive to calculate as there are $M = 2^{24\times16}$

Table 5.3: **Probability of a digit in the i-th place in a 5-digit zip code:** *This table shows the probability of a digit to be in the i-th place in the 5-digit zip code. The columns correspond to the 10 digits and the rows correspond to the 5 positions in the 5-digit zip code. The first row corresponds to the initial probabilities π_{ij}. For example, the probability of the 1st digit in a 5-digit zip code being "8" is 6%.*

Position in	*Digits*									
5-digit zip code	0	1	2	3	4	5	6	7	8	9
1st digit	0.08	0.12	0.10	0.11	0.11	0.10	0.10	0.11	0.06	0.10
2nd digit	0.10	0.09	0.11	0.10	0.10	0.11	0.09	0.10	0.11	0.08
3rd digit	0.15	0.10	0.09	0.10	0.10	0.10	0.10	0.10	0.09	0.07
4th digit	0.12	0.11	0.12	0.12	0.12	0.11	0.10	0.08	0.06	0.04
5th digit	0.10	0.12	0.11	0.10	0.10	0.10	0.10	0.09	0.09	0.08

different observations, but we will soon see that for our calculations we only need to consider five observations $O = \{O_1, O_2, O_3, O_4, O_5\}$.

Now,

$$P[O_m \mid q_t = S_{ij}] = \frac{P(q_t = S_{ij} \mid O_m)}{P(q_t = S_{ij})} P(O_m)$$

Later, when we discuss the algorithm, it will be clear that we only need to compute $b_{ij}(O_j)$ and hence, we only need to consider $P(q_t = S_{ij} \mid O_j)$. $P(q_t = S_{ij} \mid O_j)$ is the probability that the j-th observation is the digit i. This probability is computed using the numeric classifier, discussed in a separate chapter. Since we are interested in the best sequence of states which described the observations, $P(O_m)$ would appear in all such sequences. So, we drop this factor from our calculations. $P(q_t = S_{ij})$ is obtained from a pre-computed table (Table 5.3).

$$P[q_t = S_{ij}] = \begin{cases} T[j,i] & \text{if } j = t \\ 0 & \text{if } j \neq t \end{cases} \quad \text{for } 1 \leq i \leq 10, 1 \leq t \leq 5$$

where T refers to Table 5.3.

These probabilities are calculated from the database of zip codes. Once again, due to lack of information about the frequency of mails to a particular zip code, the values are only a crude approximation to real mail-stream.

- **The initial state probability distribution:** It is denoted by $\pi = \{\pi_{ij}\}$, where

$$\pi_{ij} = P[q_1 = S_{ij}].$$

In our model, at time t=1, only the states S_{i1} can be visited. These probabilities can be pre-computed using the zip code database. Once again, due to lack of information about the frequency of mails addressed to a zip code, the initial probabilities are only a crude approximation.

The initial probabilities π_{ij} are the same as the first row of Table 5.3. Hence,

$$\pi_{ij} = \begin{cases} T[1,i] & \text{if } j = 1 \\ 0 & \text{if } j > 1 \end{cases} \quad \text{for } 1 \leq i \leq 10, 1 \leq j \leq 5.$$

Now the problem of recognizing the zip code is equivalent to finding the best state sequence $Q = \{q_1, q_2, q_3, q_4, q_5\}$ for the given observation sequence $O = \{O_1, O_2, O_3, O_4, O_5\}$. The solution is given by the Viterbi algorithm. But in our case, some simplification can be done as far as computation and data structures are

concerned. The important point to note is that we increased the number of states from 10 to 50 in order to account for the position dependent transition probability. Hence, the initial probability is non-zero only for 10 states and the transition probability is 0 for most of the 50×50 possible transitions.

Another way of looking at it is that at $t = 1$ only 10 states can be visited, at $t = 2$ some other 10 states can be visited, etc. We have taken this into consideration in our implementation. This idea is explained in each of the steps as we discuss the algorithm.

We define the quantity

$$\delta_t(i,j) = \max_{q_1,q_2,\cdots,q_{t-1}} P[q_1, q_2, .., q_t = S_{ij}, O_1, O_2, \cdots, O_t \mid \lambda]$$

i.e., $\delta_t(i,j)$ is the best score (highest probability) along a single path at time t, which accounts for the first t observations and ends in state S_{ij}. By induction we have

$$\delta_{t+1}(k,l) = \max_{i,j}[\delta_t(i,j)a_{ij-kl}]b_{kl}(O_{t+1}).$$

To retrieve the state sequence, we need to keep track of the argument which maximized $\delta_t(i,j)$, for each t and for each state S_{ij}. We do this via the array $\psi_t(i,j)$. The complete procedure for finding the best sequence can now be stated as follows:

1. **Initialization:**

$$\delta_1(i,j) \quad = \quad \pi_{ij} b_{ij}(O_1), \text{for} 1 \leq i \leq 10, 1 \leq j \leq 5 \tag{5.3}$$

$$\psi_1(i,j) \quad = \quad 0, \text{for} 1 \leq i \leq 10, 1 \leq j \leq 5 \tag{5.4}$$

- Since $\pi_{i,j} = 0$, for $j \neq 1$, we need to compute $\delta_1(i,j)$ only for $j = 1$. Also, we will see in the following steps that $\delta_1(i,j)$ for $j \neq 1$ is not used in further computations. Similarly, $\psi_1(i,j)$ only needs to be computed for $j = 1$, as $\psi_1(i,j)$ for $j \neq 1$, is also not required in further computations.

2. **Recursion:** $\delta_t(k,l)$ and $\psi_t(k,l)$ will be calculated according to the following formulae for $1 \leq i \leq 10$, $1 \leq j \leq 5$ and $2 \leq t \leq 5$:-

$$\delta_t(k,l) \quad = \quad \max_{i,j}[\delta_{t-1}(i,j)a_{ij-kl}]b_{kl}(O_t) \tag{5.5}$$

$$\psi_t(k,l) \quad = \quad \arg\max_{i,j}[\delta_{t-1}(i,j)a_{ij-kl}] \tag{5.6}$$

- Since at time t, only states $S_{i,t}$ could be the states being visited, it can be argued that at time t, we only need to compute $\delta_t(k,t)$ and $\psi_t(k,t)$. Also for calculating the max or the arg max, we only need to consider $\delta_{t-1}(i,t-1)$ as $\delta_{t-1}(i,j)$ for $j \neq (t-1)$ is 0.

3. **Termination:**

$$P^* \quad = \quad \max_{i,j}[\delta_T(i,j)] \tag{5.7}$$

$$q_T{}^* \quad = \quad \arg\max_{i,j}[\delta_T(i,j)]] \tag{5.8}$$

where $T = 5$, since we are considering 5-digit zip codes.

- We know that for $j \neq T$, $\delta_T(i, j)$ is 0. Hence, although the formulae for P^* and ${q^*}_T$ require taking the maximum over all i and all j, we may only consider $\delta_T(i, T)$ in both the formulae.

4. **Path (state sequence) backtracking:**

$$ {q_t}^* = \psi_{t+1}({q^*}_{t+1}), \text{ for } t = T-1, T-2, \cdots, 1 \tag{5.9} $$

where $T = 5$, since we are considering 5-digit zip codes.

5.5 Experiments and Results

This section reports zip-extraction results obtained on 562 images, containing 5-digit zips. A single hidden layer neural network with 108 inputs, 50 hidden units and 43 outputs was used as a character classifier. A similar network but with 11 outputs was used as a digit only classifier. The 108 features used as input to these networks in these classifiers have been discussed in a later chapter.

- *Zip Location:* Out of 562 images, 548 (98%) were accepted, based on the confidence of the best path and the number of good and bad characters in that path found by the dynamic programming technique. A 5-digit zip is said to be located if and only if all the segments obtained by the dynamic programming technique for the 5-digit zip code are the same as the ones labeled manually. Out of the accepted images, 460 (84% of accepted) of the zips were located by

Table 5.4: **Zip Recognition:** *Out of a database of 562 addresses, 548 were accepted, out of which the zip was correctly located in 492 addresses (460 from best path and 32 from the second best path). This table represents the coverage of recognizing the correct zip code for the correctly located zips (492).*

#Top choices	*Recognized Zips' Coverage*	*Recognized Zips' Coverage (As % of located zips)*
1	420	85.37
2	445	90.45
3	453	92.07
4	455	92.48
5	461	93.70
10	466	94.72
20	471	95.73

the highest confidence path and another 32 (6% of accepted) were located by the second best path.

- *Zip Recognition:* Once a zip has been located, it is recognized by the digit-only classifier. A 5-digit zip is said to be recognized if and only if all the 5 digits are correctly recognized. The coverage of zips recognized given that it has been located correctly, is shown in Table 5.4.

- *Overall Performance:* Certain zips which are not located correctly, may be recognized correctly. The zip extraction results represent if the zip code was extracted correctly from the image of the address block. The coverage for the overall performance of zip extraction is given in Table 5.5.

Table 5.5: **Zip Extraction:** *Out of a database of 562 addresses, 548 were accepted. This table represents the coverage of extracting the correct zip code.*

#Top choices	*Extracted Zips' Coverage*	*Extracted Zips' Coverage (As % of accepted zips)*
1	424	77.37
2	463	84.49
3	473	86.31
4	481	87.77
5	484	88.32
10	497	90.69
20	505	92.15
30	509	92.88
40	511	93.25

5.6 Extension to 9-digits

The same dynamic programming approach is used for locating the 9-digit zip code, given a set of graphemes (more in number than that used for finding the 5-digit zip code). Instead of looking for 5 digits, the dynamic programming technique looks for 5 digits followed by a dash ('-'), followed by 4 more digits. After running both the 5-digit and the 9-digit zip location algorithms (the dynamic programming algorithm), one of the following decisions is made:

- The address has no zip code.
- The address has a 5-digit zip code.
- The address has a 9-digit zip code.

The decision is based on the weights of best paths, the number of good/bad digits that are obtained (a threshold for good digits, a threshold for bad digits, and the

match value of the characters in the best path are used for generating these numbers) in the 5-digit and 9-digit zip code locations. The 9-digit location is given more weightage to the 5-digit location, since if the address does contain a 9-digit zip then the 5-digit (which will include the last 4 digits of the 9-digit) recognition usually also returns a good match value. If the address contains a 9-digit zip code then the same HMM approach (one used for 5-digit zip code, discussed earlier) is used for zip code recognition and for producing a list of candidate zip codes.

Chapter 6

City-State Recognition

6.1 Introduction

In our system, city-state recognition follows the zip code extraction. As a result of zip code recognition, an ordered list of valid (screened using a zip code lexicon) zip codes is obtained. These zip codes are then mapped to corresponding city-state pairs. There may be more than one city-state pair for one zip code. These city-state names need to be matched with the one in the address, which may result in a possible reordering of the city-state pairs list and possibly a rejection because of a bad match. For example, a wrong zip code on the address, which is correctly recognized, may give a city name which matches very badly with the one on the address, and hence it may be rejected by the city-state recognition module. This reject is sent to the VCS for manual entry of destination address encoding.

As in zip code recognition, here also we are faced with the problem of locating the city and state and then recognizing those words. The city name usually starts on a

separate line, which is followed by the state name and which typically is followed by the zip code on the same line or the next line. Hence, we can match the city-state as a single concatenated word. The zip code has already been located and matched. But we will include the zip digits in the city-state strings to be matched and will consider the graphemes till the end of the zip code. This would give an overall match for the city-state-zip and also since a longer string would be matched compared to just the zip code string or the city-state string, the matching would be more robust.

Based on some heuristics, the starting grapheme of the city name is identified. Now these segments need to be matched with a list of city-state-zip strings. As we noted in Chapter 4, spaces and small characters such as a comma may be difficult to recognize. Hence, our strings only consist of alpha-numeric characters, though the matching scheme we are going to present is general enough to work for strings containing any characters, even including wild characters, which may stand for a subset of characters (e.g., a wild character δ may stand for digits 0 through 9).

Let us define a *template* as a string of characters including wild characters. It was identified that the matching scheme discussed for zip code extraction is a special case of a more general scheme for matching any template with a given sequence of graphemes. This generalized dynamic matching scheme is discussed in the next section.

6.2 Generalized Dynamic Programming Scheme

It was suggested by Gilloux [2] that for cursive script, it is difficult to expect a perfect segmentation without the knowledge of the characters themselves. Hence, the segmentation and recognition need to be integrated. Our dynamic programming scheme is based on this idea, as it has the capability of locating the word as well as recognizing it, given that the input may not have been perfectly segmented, but over-segmented.

The dynamic programming table and the algorithm was discussed in Chapter 5. If there are N segments to be matched with a word of length W, then the table is of size $(N - W + 1) \times (W + 1)$, where $(N - W)$ is referred to as the slack in matching. The top left cell in the table is cell number (1,1). We allow four kinds of edges(See Figure 5.7) in the table from one cell to another cell. The edges can be from cell $(i, j - 1)$ to (i, j) for a match of a character with a grapheme, from cell $(i - 1, j - 1)$ to (i, j) for a match of a character with two graphemes, from cell $(i - 2, j - 1)$ to (i, j) for a match of a character with three graphemes, and from cell $(i-1, j)$ to (i, j) for deleting a grapheme. These edges are the same as the ones used for the dynamic programming table used for zip code extraction. The path from cell $(1, 1)$ to the cell $(N-W+1, W+1)$ represents a match and the dynamic programming finds the path with the highest normalized cost. The cost function is discussed later.

An example of matching a city-state-zip using the dynamic programming approach is shown in Figure 6.2.

Since the zip code extraction scheme is a specific instance of the generalized

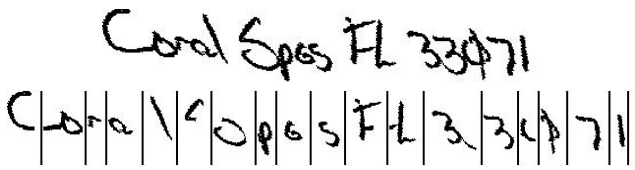


Figure 6.1: **Example of a city-state-zip image** *(a) The line containing the city name (Coral Spgs), state name (FL) and zip code (33071). (b) Results of segmentation.*

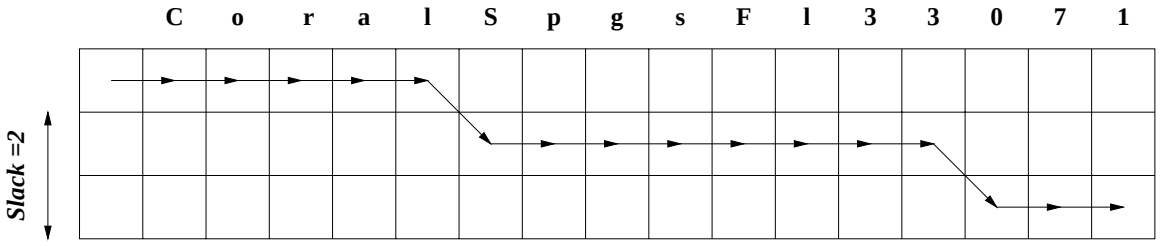


Figure 6.2: **The dynamic programming table:** *It shows the best path for the image shown in Figure 6.1*

scheme, we will only be indicating the generalizations that were done to the scheme. The reader is referred to Chapter 5 for the zip code extraction scheme.

1. *Template Representation:* A five digit template can be represented as $< \delta\delta\delta\delta\delta >$ and a 9 digit template can be represented as $< \delta\delta\delta\delta\delta - \delta\delta\delta\delta >$. The city names and state names can be represented as ordinary strings, e.g. $< SanJose >$. A template can contain digits, alphabetic characters, any special symbols or any wild characters. Currently, the system is only using one wild character δ, which stands for the set of digits 0 through 9. Other wild characters can be defined and used as and when needed.

2. *Edge weights:* In the observation graph discussed in the chapter on zip code extraction, the edge weights were based on the a posteriori probability of the segments being a digit.

   Now, the probability P_{cs} of an edge in the dynamic programming table is based

on the segment s it represents and the character c it will match in the string being matched.

$$P_{cs} = P(c/s), \text{where } P \text{ stands for the probability.}$$

For wild characters, the probability is computed by summing up the probabilities for the individual characters, which are in the set represented by that wild character. For example, if $c = \delta$, where δ is the wild character for digits 0 through 9, then $P_{\delta s} = P(\delta/s) = \sum_{i=0}^{9} P(i/s)$

3. *Cost of deletions at the ends:*

At times we are interested in matching a given word with the *best* subsequence of graphemes. Hence, we would like to allow no-cost deletions at the beginning and end of graphemes if desired. For example, while locating the 5-digit zip in the address, we can match a fixed number of graphemes (e.g, 10) from the end of the address (as the zip is typically the last item of the address) with a template representing 5-digits allowing no-cost deletions from the beginning of the sequence of graphemes.

Hence, if we are allowed deletions from the beginning, then the vertical edges (vertical edges represent deletion of graphemes) in the first column of the table are not used in calculating the weight of a path. Similarly, for allowing deletions at the end of a sequence, the vertical edges in the last column are not used in

calculating the weight of the path. Let us consider a path through the table from $(1,1)$ to the cell $(S+1, W+1)$, where the edges are $E = (e_1, \cdots, e_n)$, where n is the length of the path and e_i is the i-th edge in the path. e_p is the edge matching with the first character if no-cost deletion is allowed from the beginning or else, $e_p = e_1$. Similarly, e_q is the edge matching with the last character if no-cost deletion is allowed at the end of the sequence or else, $e_q = e_n$. Hence, we can redefine the cost as follows ($w(e_i)$ represents the weight of the edge e_i):-

$$\text{Weight}_{unnormalized} = \prod_{i=p}^{q} w(e_i) \tag{6.1}$$

$$\text{Weight}_{normalized} = (\text{Weight}_{unnormalized})^{\frac{1}{q-p+1}} \tag{6.2}$$

4. *Reusing computation for words with common prefixes:* Let us consider two words w_1 and w_2, which have a common prefix p. Hence, $w_1 = ps_1$ and $w_2 = ps_2$, where s_1 and s_2 are also strings of characters. Let us say that these words need to be matched with a sequence of N graphemes.

$$\text{Slack } L_1 \text{ for } w_1 = N - \text{length}(w_1)$$

$$\text{Slack } L_2 \text{ for } w_2 = N - \text{length}(w_2)$$

Hence, the size of the dynamic programming tables are as follows:-

$$\text{Size of table for } w_1 = (L_1 + 1) \times (\text{ length}(w_1) + 1)$$

$$\text{Size of table for } w_2 = (L_2 + 1) \times (\text{ length}(w_2) + 1)$$

Note that a path ending at a cell (i, j) can not have an intermediate node outside the rectangle $((1, 1), (i, j))$. Hence, the computation in the table in the rectangle ((1,1),(x,y)), where

$$\begin{aligned} x &= \min(L_1 + 1, L_2 + 1) \\ y &= \text{length}(p) + 1 \end{aligned}$$

is common for the computations for w_1 and w_2 (see Figure 6.3). Hence, we need to compute the entries for the cells of that part of the table only once.

In general, if there are m words $(w_1, \cdots w_m)$ with a common prefix p, then we will have m slacks $(L_1. \cdots, L_m)$, where $L_i = N - \text{length}(w_i)$. Then the common rectangle for which the computation can be used for all these words is given by $((1, 1), (x, y))$, where

$$x = \min(L_1 + 1, \cdots, L_m + 1), \text{ and}$$

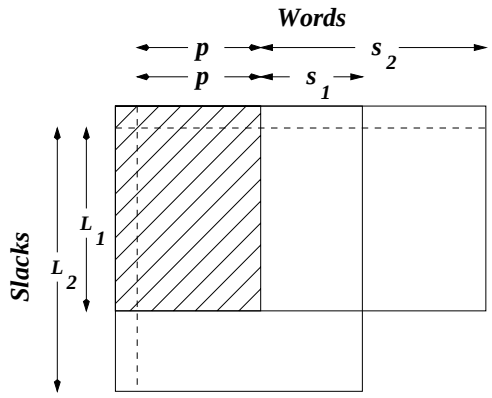


Figure 6.3: **Utilizing common prefixes to save computation in the dynamic programming scheme:** *Words w_1 and w_2 have the common prefix p. The slacks corresponding to the two words are L_1 and L_2 respectively. The first row and the column are marked by dotted lines. The shaded portion of the table represents the part of the table whose computations are common for both the words and hence can be reused.*

$$y = \text{length}(p) + 1$$

Hence, the time taken by the dynamic programming technique can be reduced.

6.3 Rejection Mechanism

The system can make recognition errors at various levels, which need to be weeded out as early as possible. Zip code extraction errors may lead to city-state lexicons which do not contain the truth in it. Even if the truth is in the lexicon, the image may match better with some other city-state-zip in the lexicon, because of poor image quality or handwriting quality.

Let us define the rank of a character in a word as the number of characters which had better match values with the segment with which this character has been matched based on overall cost. The following features were used to arrive at a recognition

mechanism based on simple thresholding :-

1. A histogram of ranks for all the characters in the city-state-zip triplet is created and the histogram is binned into 5 bins. The ranks of characters can vary from 1 through 43 as we are considering 43 characters, including the special symbols. The 1st and 2nd ranks contribute to the first bin, the 3rd and 4th to the second bin and so on till rank number 8. The rest of the ranks contribute to the last bin. A higher contribution to the 1st and 2nd bins and lower value in the last bin indicates a good match and is accepted.

2. A histogram of ranks for all the characters in the city-state pair is created and the histogram is binned into 5 bins. The binning is done exactly as in the previous step. Once again, a higher value in the 1st and 2nd bins indicate that more number of characters in the best path were good characters. Hence, the match can be accepted.

3. Total normalized match value obtained as a result of matching the city-state-zip to the address. A high match value is used as an indication of an acceptable match.

6.4 Results

The dynamic programming scheme was tested for city-state-zip level recognition using 805 cursive script images containing 5-digit zip codes from the USPS database. The average lexicon size of city-state-zip generated after zip code extraction was 28.7

Table 6.1: **Coverage for City-State-Zip recognition:***The system was tested on 805 cursive script images taken from the USPS database. 632 (78.51%) of those addresses were accepted. The numbers shown in the table are rates over the accepted addresses.*

#Top choices	*Recognized City-State-Zips' Coverage*	*Recognized City-State-Zips' Coverage (As % of accepted matches)*
1	609	96.36
2	614	97.15
3	616	97.47
4	616	97.47
5	616	97.47
6	617	97.63

words. Out of the 805 images, 632 (78.51%) were accepted by the mechanism for rejection discussed in the previous section. The coverage of the true city-state-zip by the reordered (after matching) list of city-state-zip is shown in Table 6.1. The top choice covered 96.36% of the truths. Since these city-state-zip lexicons were obtained after zip code extraction, some of the lexicons may not contain the true city-state-zip. In our case, 15 (2.37% over accepted) of the accepted addresses do not contain the truth in the lexicon.

6.5 Summary

The dynamic programming scheme was generalized so that it can work for any string of characters, even including wild characters. The total cost of a path in the dynamic programming table was redefined with slight modification to allow no-cost deletion at the beginning and end of a sequence of graphemes, if desired. This made it possible to match words with the best subsequence of graphemes. Also, a scheme for utilizing

prefixes to save computation is designed. At the city-state level, we need to reject bad matches. A mechanism for rejection based on several features extracted out of matching the city-state-zip lexicon gave an acceptance rate of 78.51% for a top choice rate of 96.36%. These figures were obtained on 805 cursive script addresses from the USPS database.

Chapter 7

Combining Classifiers to Improve Word Level Recognition

7.1 Introduction

In written text analysis, design of a high performance OCR has received a lot of attention. The goal has typically been to design OCR techniques which give high acceptance rates and low reject rates. Several ways of combining individual classifiers to obtain better overall character level performances have also been suggested [16, 30, 31, 32]. These classifier combinations have been characterized based on several schemes [33] such as (i) architecture, (ii) trainability of combiner, (iii) adaptivity of combiner and (iv) requirement on outputs of individual classifiers. Further, based on architecture these combinations have been classified as serial, parallel, gated parallel or hierarchical. Improving the character level recognition of the OCR system is the goal behind designing all these ensembles of classifiers.

However, if we take a closer look at the applications which use the OCR systems, like postal address recognition, automatic check processing and forms processing, we find that often our goal is to recognize a digit string (as in the case of zip codes) or words (like city names, state names etc.). Hence, the OCR should be designed in such a way that it maximizes the word level recognition. The word level recognition rate is a global criteria, as it takes into consideration the whole word and not just the individual characters.

Work done towards designing neural network classifiers for *digit-only*, *alphabet-only* and *general* (alpha-numeric and special) character recognition was presented in Chapter 3. This chapter discusses how these individual character level optimized classifiers can be combined to achieve higher word-level recognition rates.

Combining results from different classifiers has been shown to have a significant improvement in recognition accuracy [30, 31, 32]. It has been accepted as a good technique for exploiting the discriminant power of individual classifiers.

Word level training has been proposed and used by several researchers most notably by Bengio et. al. [22]. They have proposed a mechanism for global optimization based on a word level criteria, where all the parameters of the classifier (a neural network in their case) and the parameters of the HMM, used for matching whole words, are optimized. This optimization is done by the gradient descent method. Our work involves combining multiple character level optimized OCR systems (1-hidden layer neural networks, described in Chapter 3) using word level optimized functions. Since it involves combining classifiers as well as optimizing the combination at the word level, we expect better performance than simple classifier combinations such as aver-

aging.

This chapter reports the results of word level optimization done for several functions which have been used to combine the individual classifiers (1-hidden layer neural networks). This idea can be extended to update the neural network weights also.

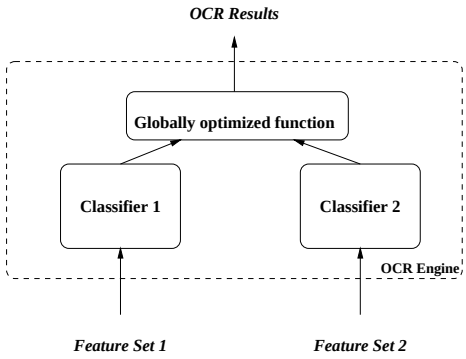


Figure 7.1: **The globally trained OCR :** *The results of the two classifiers are combined to produce the final OCR results. The parameters of the combination function will be optimized using a word level criteria.*

7.2 Word Level Cost Function

As mentioned earlier, we use the dynamic programming scheme (Section 6.2) for finding the best match for a given word with a given sequence of graphemes. The path selected through the corresponding observation graph (Figure 5.5) is the highest probability path from the source (leftmost node) to the destination (rightmost node).

Our goal is to modify the OCR engine in such a way that the weights of the edges in the observation graph are modified so that the correct path through the graph becomes the highest probability path. The cost function should be such that it approaches the optimum value as the probability of the correct path increases. We choose the following cost function:-

$$\mathrm{C} \quad = \quad \ln\left(\frac{P^{(d)}}{\sum_{i=1}^{N} P^{(i)}}\right), \tag{7.1}$$

where N is the total number of paths through the observation graph, the path-number for the correct or the desired path is d, and $P^{(i)}$ represents the total probability of the i-th path. We see that C increases as $P^{(d)}$ increases. This criteria corresponds to the mutual information between the observations and the desired interpretation. Maximizing the cost function C corresponds to maximizing the mutual information, which corresponds to maximizing the probability of the desired path.

7.3 Gradient Descent Method

In the word level training process, the aim is to find the best values for the parameters $\theta_i, i = 1 \cdots p$, of the combination function f used to combine the results of the two classifiers.

A set of words which have been over-segmented and manually truthed is used for training (details are given in the following section). These words are used to compute the partial derivatives of the cost C with respect to the parameters. These derivatives are then used to update the parameters θ_i according to Equation (7.2). Here, $\theta_i^{(t)}$ refers to the value of the parameter θ_i at time step (or iteration number) t and η_i is the learning rate for the parameter θ_i.

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \eta_i \frac{\partial C}{\partial \theta_i^{(t)}} \tag{7.2}$$

This updation is done for every word in the list. This whole process is then repeated several times till the θ_i's reach stable values.

The partial derivatives of the cost function with respect to the parameter θ_m is given in Equation (7.4). Here, n_i is the number of edges in the i-th path, N is the total number of paths and $P^{(i)}$ is the probability of the i-th path which is given in Equation (7.3). S_{ij} refers to the segment corresponding to the j-th edge on the i-th path. k_{ij} refers to the index of the classifier output for the segment S_{ij}. $y_{k_{ij}}(S_{ij})$ refers to the k_{ij}-th output of the word level trained classifier for the segment S_{ij}. We are not considering starting or transition probabilities in our experiments.

$$P^{(i)} = \prod_{j=1}^{n_i} y_{k_{ij}}(S_{ij}) \tag{7.3}$$

$$\frac{\partial C}{\partial \theta_m} = \sum_{j=1}^{n_d} \{\frac{1}{y_{k_{dj}}(S_{dj})} \times \frac{\partial y_{k_{dj}}(S_{dj})}{\partial \theta_m}\} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} \{P^{(i)} \times \frac{1}{y_{k_{ij}}(S_{ij})} \times \frac{\partial y_{k_{i,j}}(S_{ij})}{\partial \theta_m}\}}{\sum_{i=1}^{N} P^{(i)}} \tag{7.4}$$

The partial derivative $\frac{\partial y_{k_{ij}}(S_{ij})}{\partial \theta_m}$ is determined by the combination function used to combine the results of the two classifiers.

7.4 Training Setup

For our experiments on word level training, 5305 words are automatically extracted from cursive script USPS addresses. These words are over-segmented and each of the graphemes was then manually truthed. The experiments are restricted to alphabetic characters. So, each classifier has $n = 27$ outputs (26 corresponding to the alphabetic characters and 1 for the probability of the segment for deleting). The classifier outputs are approximated as a posteriori probabilities for the characters and for being a deletable character.

We used the following two neural network classifiers as the individual classifiers and tried several ways of combining the neural networks, which are discussed in the next section. The neural networks were optimized at the character level.

- *Classifier #1:* A 1-hidden layer network with 108 input units, 50 hidden layer units and 27 output units. The error function which is optimized for training is the Kullback-Leibler function (Section 3.3.1). The sigmoid function (Section 3.3.2) is used as the activation function at the hidden layer units and the softmax function is used as the activation function at the output layer.

- *Classifier #2:* Another 1-hidden layer network with 108 input units, 50 hidden layer units and 27 output units. The error function which is optimized for training is the squared error function. The sigmoid function (Section 3.3.2) is used as the activation function at the hidden layer units and the output units.

The performance of the two classifiers for word level recognition on the 5305

Table 7.1: **The Two Individual Classifiers:** *The two neural networks used for performing experiments on word level training are summarized here. The training data consists of 5305 words and was used for training the individual networks and also for estimating the parameters of the combination function. The test data consists of 4417 words. Both these sets of words were obtained from cursive script images from the USPS database.*

	Classifier #1	*Classifier #2*
Error Criteria	Kullback-Leibler function	Squared Error
Word Level Recognition(Training)	87.07%	89.41%
Word Level Recognition(Test)	88.91%	90.97%

training words is given in Table 7.1. This training data was used for training the individual classifiers as well as for estimating the parameters of the combination function. Results on a test set of 4417 words are also shown.

7.5 Different Combinations of Neural Networks

It is possible to combine the results of two or more classifiers using several functions. For example, it has been shown that averaging of results of similar classifiers achieves better character level performance than individual classifiers.

Let $\{y_{11}, \cdots, y_{1n}\}$, be the n-dimensional output of the first classifier, where the outputs correspond to the a posteriori probabilities for the n classes. Similarly, let $\{y_{21}, \cdots, y_{2n}\}$, be the n-dimensional output of the second classifier. Let $\mathbf{y} = \{y_1, \cdots, y_n\}$, be the n-dimensional output of the combination of the two classifiers such that $\mathbf{y} = f_{\theta_1, \cdots \theta_p}(y_{11}, \cdots, y_{1n}, y_{21}, \cdots, y_{2n})$, where f is the combination

function and the $\theta_1 \cdots \theta_p$ are the p parameters of the function. These are the parameters which are learned using the gradient descent method so as to improve the word recognition rate.

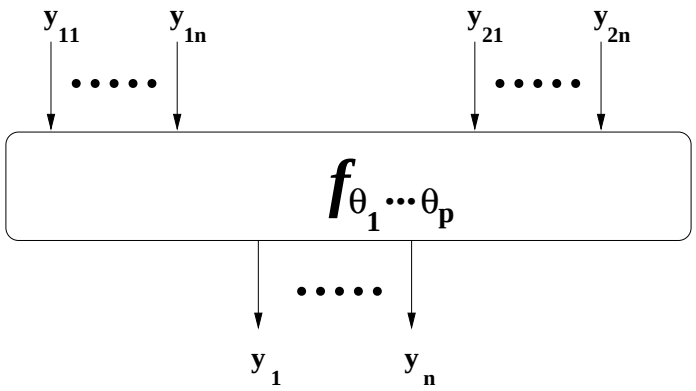


Figure 7.2: **The combination function:** *The function f combines the output of the two neural networks (classifiers in general) and produces the output vector* $\mathbf{y}$*. The parameters of the function θ_i's are obtained by word level training using the gradient descent method.*

Figure 7.2 shows the job of the combination function. An example of the function f is the averaging function which defines the vector $\mathbf{y}$ as:-

$$y_i = \frac{y_{1i} + y_{2i}}{2}, \text{ for } i = 1 \text{ to } n. \tag{7.5}$$

We tried several non-linear functions f for combining the individual neural network outputs. The functions which have been tried for training at the word level are as follows:-

1. *Softmax Function:* The softmax function is explained in detail in Section 3.3.2. Using the notations shown in Figure 7.2, the final output of the softmax ensemble of the two classifiers is given by (α and β are the parameters of the

function)

$$y_k = \frac{e^{\alpha y_{1k} + \beta y_{2k}}}{\sum_{l=1}^{n} e^{\alpha y_{1l} + \beta y_{2l}}}. \tag{7.6}$$

The partial derivatives of y_k with respect to the parameters are as follows:-

$$\frac{\partial y_k}{\partial \alpha} = y_k \times (y_{1k} - \sum_{l=1}^{n} y_l y_{1l}), \text{ and} \tag{7.7}$$

$$\frac{\partial y_k}{\partial \beta} = y_k \times (y_{2k} - \sum_{l=1}^{n} y_l y_{1l}). \tag{7.8}$$

Using Equations (7.4), (7.7) and (7.8) we get the partial derivatives of the cost function with respect to the parameters (Equations (7.9) and (7.10)). A dynamic programming technique(Section 7.6) which computes several partial results at each of the nodes of the observation graph from the leftmost to the rightmost node is used to compute these partial derivatives. These derivatives are then used in Equation (7.2) to update the values of α and β. In the following equations, $y_{1k_{ij}}(S_{ij})$ refers to the k_{ij}-th output of the first classifier for the segment S_{ij} and $y_{2k_{ij}}(S_{ij})$ refers to the same for the second classifier.

$$\frac{\partial C}{\partial \alpha} = \sum_{j=1}^{n_d} [y_{1k_{dj}}(S_{dj}) - \sum_{l=1}^{n} \{y_{k_{dj}}(S_{dj}) \times y_{1k_{dj}}(S_{dj})\}] - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} [P^{(i)} \times \{y_{1k_{ij}}(S_{ij}) - \sum_{l=1}^{n} \{y_l(S_{ij}) \times y_{1l}(S_{ij})\}\}]}{\sum_{i=1}^{N} P^{(i)}} \tag{7.9}$$

$$\frac{\partial C}{\partial \beta} = \sum_{j=1}^{n_d} [y_{2k_{dj}}(S_{dj}) - \sum_{l=1}^{n} \{y_{k_{dj}}(S_{dj}) \times y_{2k_{dj}}(S_{dj})\}] - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} [P^{(i)} \times \{y_{2k_{ij}}(S_{ij}) - \sum_{l=1}^{n} \{y_l(S_{ij}) \times y_{2l}(S_{ij})\}\}]}{\sum_{i=1}^{N} P^{(i)}} \quad (7.10)$$

After training on a data set consisting of 5305 words, we obtained $\alpha = 3.72$ and $\beta = 5.07$. The word recognition rate (using random lexicon of size 10 containing the truth) of the word level trained ensemble on the training data is 95.08%, which is 5.67% better than the best individual network used to form the ensemble. On the test data, the performance is 94.95%, which is 3.98% better than the performance of the individual best network.

2. *Sigmoid Function:* The sigmoid function is explained in detail in Section 3.3.2. Using the notations shown in Figure 7.2, the final output of the sigmoid ensemble of the two classifiers is given by (α, β and γ are the parameters of the function):-

$$y_k = \frac{e^{\alpha y_{1k} + \beta y_{2k} + \gamma}}{1 + e^{\alpha y_{1k} + \beta y_{2k} + \gamma}}. \quad (7.11)$$

The partial derivatives of y_k with respect to the parameters are as follows:-

$$\frac{\partial y_k}{\partial \alpha} = (1 - y_k) \times y_k \times y_{1k} \quad (7.12)$$

$$\frac{\partial y_k}{\partial \beta} = (1 - y_k) \times y_k \times y_{2k} \quad (7.13)$$

$$\frac{\partial y_k}{\partial \gamma} = (1 - y_k) \times y_k. \quad (7.14)$$

Using Equations (7.4), (7.12), (7.13) and (7.14) we get the partial derivatives of the cost function with respect to the parameters (Equations (7.15), (7.16) and (7.17)). A dynamic programming technique(Section 7.6) which computes several partial results at each of the nodes of the observation graph from the leftmost to the rightmost node is used to compute these partial derivatives. These derivatives are then used in Equation (7.2) to update the values of the parameters.

$$\frac{\partial C}{\partial \alpha} = \sum_{j=1}^{n_d} \{(1 - y_{k_{dj}}(S_{dj})) \times y_{1k_{dj}}(S_{dj})\} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} \{P^{(i)} \times (1 - y_{k_{ij}}(S_{ij})) \times y_{1k_{ij}}(S_{ij})\}}{\sum_{i=1}^{N} P^{(i)}} \tag{7.15}$$

$$\frac{\partial C}{\partial \beta} = \sum_{j=1}^{n_d} \{(1 - y_{k_{dj}}(S_{dj})) \times y_{2k_{dj}}(S_{dj})\} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} \{P^{(i)} \times (1 - y_{k_{ij}}(S_{ij})) \times y_{2k_{ij}}(S_{ij})\}}{\sum_{i=1}^{N} P^{(i)}} \tag{7.16}$$

$$\frac{\partial C}{\partial \gamma} = \sum_{j=1}^{n_d} (1 - y_{k_{dj}}(S_{dj})) - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} \{P^{(i)} \times (1 - y_{k_{ij}}(S_{ij}))\}}{\sum_{i=1}^{N} P^{(i)}} \tag{7.17}$$

After training on a data set consisting of 5305 words, we obtained $\alpha = 6.72$, $\beta = 9.27$ and $\gamma = -5.73$. The word recognition rate(using random lexicon of size 10 containing the truth) of the word level trained ensemble on the training data is 95.34%, which is 5.93% better than the best individual network used to form the ensemble. On the test data, the performance is 95.20%, which is 4.23% better than the performance of the individual best network.

3. *A Simple Nonlinear Function:* The k-th output of the combined network y_k is given by (α, β and γ are the parameters of the function) :-

$$y_k = \frac{\alpha y_{1k}^{\beta} + y_{2k}^{\gamma}}{\sum_{l=1}^{n} \alpha y_{1l}^{\beta} + y_{2l}^{\gamma}}. \tag{7.18}$$

The partial derivatives of y_k with respect to the parameters are as follows:-

$$\frac{\partial y_k}{\partial \alpha} = \frac{y_{1k}^{\beta} - y_k \sum_{l=1}^{n} y_{1l}^{\beta}}{\sum_{l=1}^{n} (\alpha y_{1l}^{\beta} + y_{2l}^{\gamma})} \tag{7.19}$$

$$\frac{\partial y_k}{\partial \beta} = \frac{\alpha y_{1k}^{\beta} \ln y_{1k} - y_k \sum_{l=1}^{n} (\alpha y_{1l}^{\beta} \ln y_{1l})}{\sum_{l=1}^{n} (\alpha y_{1l}^{\beta} + y_{2l}^{\gamma})} \tag{7.20}$$

$$\frac{\partial y_k}{\partial \gamma} = \frac{y_{2k}^{\gamma} \ln y_{2k} - y_k \sum_{l=1}^{n} (y_{2l}^{\gamma} \ln y_{2l})}{\sum_{l=1}^{n} (\alpha y_{1l}^{\beta} + y_{2l}^{\gamma})}. \tag{7.21}$$

Using Equations (7.4), (7.19), (7.20) and (7.21) we get the partial derivatives of the cost function with respect to the parameters (Equations (7.22), (7.23) and (7.24)). Once again, a dynamic programming technique(Section 7.6) which computes several partial results at each of the nodes of the observation graph from the leftmost to the rightmost node is used to compute these partial derivatives. These derivatives are then used in Equation (7.2) to update the values of the parameters.

$$\frac{\partial C}{\partial \alpha} = \sum_{j=1}^{n_d} \{ \frac{y_{1k_{dj}} (S_{dj})^{\beta} - y_{k_{dj}} (S_{dj}) \sum_{l=1}^{n} y_{1l} (S_{dj})^{\beta}}{y_k \times \sum_{l=1}^{n} (\alpha y_{1l} S_{(dj)}^{\beta} + y_{2l} (S_d j)^{\gamma})} \} - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} \{ P^{(i)} \times \frac{y_{1k_{ij}} (S_{ij})^{\beta} - y_{k_{ij}} (S_{ij}) \sum_{l=1}^{n} y_{1l} (S_{ij})^{\beta}}{\sum_{l=1}^{n} (\alpha y_{1l} (S_{ij})^{\beta} + y_{2l} (S_{ij})^{\gamma})} \}}{\sum_{i=1}^{N} P^{(i)}} \tag{7.22}$$

$$\frac{\partial C}{\partial \beta} = \sum_{j=1}^{n_d} [\frac{\alpha y_{1k_{dj}}(S_{dj})^{\beta} \ln y_{1k} - y_{k_{dj}}(S_{dj}) \sum_{l=1}^{n} \{\alpha y_{1l}(S_{ij})^{\beta} \ln y_{1l}(S_{ij})\}}{y_{k_{dj}}(S_{dj}) \times \sum_{l=1}^{n} (\alpha y_{1l}(S_{dj})^{\beta} + y_{2l}^{\gamma})}] - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} [P^{(i)} \times \frac{\alpha y_{1k_{ij}}(S_{ij})^{\beta} \ln y_{1k_{ij}}(S_{ij}) - y_{k_{ij}}(S_{ij}) \sum_{l=1}^{n} \{\alpha y_{1l}(S_{ij})^{\beta} \ln y_{1l}(S_{ij})\}}{\sum_{l=1}^{n} (\alpha y_{1l}(S_{ij})^{\beta} + y_{2l}(S_{ij})^{\gamma})}]}{\sum_{i=1}^{N} P^{(i)}} \tag{7.23}$$

$$\frac{\partial C}{\partial \gamma} = \sum_{j=1}^{n_d} [\frac{y_{2k_{dj}}(S_{dj})^{\gamma} \ln y_{2k} - y_{k_{dj}}(S_{dj}) \sum_{l=1}^{n} \{y_{2l}(S_{ij})^{\gamma} \ln y_{2l}(S_{ij})\}}{y_{k_{dj}}(S_{dj}) \times \sum_{l=1}^{n} (\alpha y_{1l}(S_{dj})^{\beta} + y_{2l}^{\gamma})}] - \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} [P^{(i)} \times \frac{y_{2k_{ij}}(S_{ij})^{\gamma} \ln y_{2k_{ij}}(S_{ij}) - y_{k_{ij}}(S_{ij}) \sum_{l=1}^{n} \{y_{2l}(S_{ij})^{\gamma} \ln y_{2l}(S_{ij})\}}{\sum_{l=1}^{n} (\alpha y_{1l}(S_{ij})^{\beta} + y_{2l}(S_{ij})^{\gamma})}]}{\sum_{i=1}^{N} P^{(i)}} \tag{7.24}$$

This function has a problem with convergence since the derivative has exponential and logarithmic terms which often can become too small or too large resulting in a bad approximation to the partial derivatives (Equations (7.22), (7.23) and (7.24)). Hence, for some starting values of the parameters, the values of the parameters keep diverging. Even otherwise, it takes some 20 epochs to converge as opposed to 5 epochs in the case of sigmoid or softmax functions. The sigmoid and softmax functions discussed earlier do not have any of these convergence problems.

After training on the training data consisting of 5305 words, we obtained $\alpha = 36.65$, $\beta = 1.03$ and $\gamma = 1.26$. The word recognition rate(using random lexicon of size 10 containing the truth) of the word level trained ensemble on the training data is 92.72%, which is 3.31% better than the best individual network used to form the ensemble. On the test data, the performance is 92.05%, which is 1.08% better than the performance of the individual best network.

7.6 Computation of the Partial Derivatives

The equations for partial derivatives of the cost function (Equations (7.9), (7.10), (7.15), (7.16), (7.17), (7.22), (7.23) and (7.24)) involve several summations. These are the two types of summations which appear in all these equations and can be computed much efficiently:-

$$\sum_{i=1}^{N} P^{(i)} \text{ and } \sum_{i=1}^{N} \{P^{(i)} \times \sum_{j=1}^{n_i} T(S_{ij}, k_{ij})\}$$

Here, k_{ij} is the index of the neural net output corresponding to the segment S_{ij}. and $T(S_{ij}, k_{ij})$ is a term which depends on the segment S_{ij} and the k-th output of the neural net for that segment. For example, for the partial derivative in Equation (7.9),

$$T(S_{ij}, k_{ij}) = \{y_{1k_{ij}}(S_{ij}) - \sum_{l=1}^{n} \{y_l(S_{ij}) \times y_{1l}(S_{ij})\}\} \tag{7.25}$$

Both these summations can be easily computed using dynamic programming. Let us number the vertices in the observation graph from 0 to τ, where the 0-th vertex represents the starting point and the τ-th vertex represents the destination point. The number of graphemes are τ.

The algorithm requires calculation of partial results at each of these vertices from left to right. Let us define the two partial results $A^{(t)}$ and $B^{(t)}$ at vertex number t.

$$A^{(t)} = \sum_{i=1}^{N^{(t)}} P^{(i)} \tag{7.26}$$

$$B^{(t)} = \sum_{i=1}^{N^{(t)}} \{P^{(i)} \times \sum_{j=1}^{n_i} T(S_{ij})\} \tag{7.27}$$

$N^{(t)}$ corresponds to the total number of paths from the vertex 0 to vertex t. S_{ij} corresponds to the j-th segment on the i-th path.

Hence, the final summations that we are interested in are $A^{(\tau)}$ and $B^{(\tau)}$.

The algorithm is as follows:-

1. *Initialization:* $A^{(0)} = 1$, $A^{(-1)} = 0$, $A^{(-2)} = 0$, $B^{(0)} = 0$, $B^{(-1)} = 0$ and $B^{(-2)} = 0$. $A^{(-1)}$, $A^{(-2)}$, $B^{(-1)}$ and $B^{(-2)}$ have been intialized to make the recursion step simple.

2. *Recursion:* Compute $A^{(t+1)}$ and $B^{(t+1)}$ from $A^{(t)}$ and $B^{(t)}$ for $t = 0 \cdots (T-1)$. Finally we get $A^{(\tau)}$ and $B^{(\tau)}$, the summations we were looking for. $S_{i;j}$ in the following equations refers to the segment corresponding to the edge in the observation graph from vertex i to vertex j.

$$\begin{aligned} A^{(t+1)} &= A_{(t)} \times \sum_{k=1}^{n} y_k(S_{t;t+1}) + A_{(t-1)} \times \sum_{k=1}^{n} y_k(S_{t-1;t+1}) + \\ &\quad A_{(t-2)} \times \sum_{k=1}^{n} y_k(S_{t-2;t+1}) \end{aligned} \tag{7.28}$$

$$B^{(t+1)} = \sum_{k=1}^{n} [y_k(S_{t;t+1}) \times \{B^{(t)} + A^{(t)} \times T(S_{t;t+1}, k)\}] +$$

$$\sum_{k=1}^{n} [y_k(S_{t-1;t+1}) \times \{B^{(t-1)} + A^{(t-1)} \times T(S_{t-1;t+1}, k)\}] +$$
$$\sum_{k=1}^{n} [y_k(S_{t-2;t+1}) \times \{B^{(t-2)} + A^{(t-2)} \times T(S_{t-2;t+1}, k)\}] \quad (7.29)$$

The proof of the above equations is omitted.

7.7 Summary

OCR systems are often designed for word recognition, and hence, a better word level recognition rate is more desirable than a better character level recognition rate. The word level training of an ensemble of neural networks is a promising technique for improving word level recognition rates. The function used for combining the classifier results has an effect on the trainability and generalizability of the ensemble. Three different functions were experimented with, on words obtained from the USPS cursive script addresses. The results of word level training using these functions is summarized in Table 7.2.

Based on the results on the test data, the maximum improvement in performance (Tables 7.1 and 7.2) is 95.20% - 90.97% = 4.23% for the sigmoid function.

Table 7.2: **Results of word level training:** *The training data refers to 5305 words from the USPS database. The test data refers to 4417 words also from the USPS database. The training data was used to find the values of the parameters of the combination functions.*

	Parameters	*Training Data*	*Test Data*
Softmax	$\alpha = 3.72$, $\beta = 5.07$	95.08%	94.95%
Sigmoid	$\alpha = 6.72$, $\beta = 9.27$, $\gamma = -5.73$	95.34%	95.20%
A non-linear function	$\alpha = 36.65$, $\beta = 1.03$, $\gamma = 1.26$	92.72%	92.05%

Chapter 8

Summary, Conclusions and Future Research

8.1 Summary

Cursive script postal address recognition is recognized to be a difficult problem because of large variations in writing styles. Ligatures add another dimension of difficulty to the problem by making segmentation a difficult task. A scheme based on over-segmentation followed by dynamic programming technique, for cursive script word recognition is the focus of this thesis. Given a sequence of graphemes and a lexicon of words, the dynamic programming technique locates the best subsequence of graphemes and finds the best segmentation. This scheme was tested for zip code recognition and for city and state name recognition. Results have been reported on cursive script addresses from the USPS database.

This thesis contributed to several modules of the system. The scheme for locating

and recognizing zip codes was a result of thesis work. The generalization done to the scheme to use it for word matching, and matching city-state names in particular, was an original work. The over-segmentation algorithm, which is a prerequisite for the dynamic programming, was also an original contribution to the thesis.

OCR is the heart of such a recognition system. A mechanism for combining individual classifiers, optimized at the character level which will give higher word level recognition rate, was explored. The key idea was to combine the classifiers using a function, the parameters of which are optimized to minimize a word level criteria. The parameters were trained using gradient descent method. This scheme was tested on the USPS database and a significant improvement in the word recognition rate was observed. This scheme for combining OCR systems to optimize a word level criteria is also a contribution of the thesis.

8.2 Conclusions

We draw the following conclusions from the thesis:-

1. The over-segmentation algorithm discussed in the thesis gave good results. It can be used for over-segmenting cursive script in other applications such as automatic forms processing or check recognition, etc.

2. The dynamic programming scheme presented in the thesis was used for zip extraction and city-state recognition. This is a general scheme for word level recognition and can be used in other applications dealing with cursive script.

3. The word level training of character classifiers is an effective technique for obtaining high performance classifiers, since they give better word level recognition rates. Also, since in this framework we are combining several character level optimized classifiers, we expect better results as the power of more than one classifier is being used. These ideas were supported by the experiments performed on the USPS database.

8.3 Future Research

Following are some topics on which further research can be pursued:-

1. *Reusing more intermediate computations in the dynamic programming scheme:* Our implementation currently supports words with a set of suffixes as explained in Chapter 6. We can organize the words in a tree like structure such that every node represents a substring and every leaf represents the word obtained by concatenating the strings from the root node to that leaf node. Also, the string in any node is the longest prefix for all words represented by the subtree below that node. This trie-like data structure for representing words to be matched using the dynamic programming scheme, can be used to save some intermediate computations. The dynamic programming done till a particular node in the tree can be used by all of its subtrees, hence reducing the time required for the dynamic programming scheme.

2. *Street level recognition:* The dynamic programming technique is being currently tested for street level recognition. The main problem in street level recognition

is the number of street names which need to be matched against the address. If we take the top two choices of cities, then the total number of street names is roughly 600. A reduction in the number of street names to be matched would make the street level matching more effective. For example, heuristics based on first character recognition (usually the first character in a word is capitalized and written more clearly compared to the other characters) can help in pruning the list of street names.

Bibliography

[1] S. N. Srihari. Recognition of handwritten and machine-printed text for postal address interpretation. *Pattern Recognition Letters*, 14:291–302, April 1993.

[2] Michel Gilloux. Research into the new generation of character and mailing address recognition systems at the French post office research center. *Pattern Recognition Letters*, 14(4):267–276, April 1993.

[3] Masaharu Wada. Proposal for fully automated mail processing system for the 21st century. *Pattern Recognition Letters*, 14(4):281–290, April 1993.

[4] Yukinobu Tokunaga. History and current state of postal mechanization in Japan. *Pattern Recognition Letters*, 14(4):277–280, April 1993.

[5] S. N. Srihari. High-performance reading machines. *Proc. of the IEEE*, 80(7):1120–1132, July 1992.

[6] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, February 1989.

[7] M. Y. Chen, A. Kundu, and S. N. Srihari. Variable duration hidden markov model and morphological segmentation for handwritten word recognition. *IEEE Trans. on Image Processing*, 4(12):1675–1688, December 1995.

[8] M. Y. Chen, A. Kundu, and Jian Zhou. Off-line handwritten word recognition using a hidden markov model type stochastic network. *IEEE Trans. PAMI*, 16(5):481–496, May 1994.

[9] A. Kundu. Recognition of handwritten word: First and second order hidden markov model based approach. *Pattern Recognition*, 22(3):283–297, 1989.

[10] R. M. Bozinovic and S. N. Srihari. Off-line cursive word recognition. *IEEE Trans. PAMI*, 11:68–83, Jan 1989.

[11] P. D. Gader, J. M. Keller, R. Krishnapuram, J. H. Chiang, and M. A. Mohamed. Neural and fuzzy methods in handwriting recognition. *IEEE Computer*, pages 79–85, February 1997.

[12] W. Niblack. In *An Introduction to Digital Image Processing*. Prentice Hall, 1986.

[13] B. Yu, A. Jain, and M. Mohiuddin. Address block location on complex mail pieces. *submitted to The 4th Int. Conf. on Document Analysis and Recognition, Ulm*, August 1997.

[14] J. Mao. Personal communications. IBM Almaden Research Center, 1995.

[15] O. D. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition- a survey. *Pattern Recognition*, 29(4):641–662, 1996.

[16] J. Mao and K. M. Mohiuddin. Improving OCR performance using character degradation models and multiple classifiers. *Accepted to appear in Pattern Recognition Letters.*

[17] S. Madhvanath and V. Govindaraju. Serial classifier combination for handwritten word recognition. *International Conference on Document Analysis and Recognition, Montreal, Canada*, II:911–914, August 1995.

[18] H. Takahashi. A neural net OCR using geometrical and zonal-pattern features. *Proc. 1st Intl. Conf. on Document Analysis and Recognition*, pages 821–828, 1991.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Exploration in the microstructure of cognition*, 1:318–362, 1986.

[20] Y. L. Cun. Learning processes in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240, Les Houches, France, Springer-Verlag, 1986.

[21] A. K. Jain, J. Mao, and K.M. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):31–44, March 1996.

[22] Globally trained handwritten word recognizer using spatial representation, convolutional neural networks and hidden markov models. *NIPS(Neural Information Processing Systems)*, pages 937–944, 1993.

[23] Stefan Knerr. The A2iA recognition system for handwritten checks – intercheque v1.0 and intercheque- coupon v1.0. *A2iA Technical Report*, (11), March 1996.

[24] E. Cohen, J. J. Hull, and S. N. Srihari. Control structure for interpreting handwritten addresses. *IEEE Trans. Pattern Anal. Machine Intell.*, 16(10):1049–1055, October 1994.

[25] N. W. Strathy and C. Y. Suen. A new system for reading handwritten zip codes. *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, Canada*, pages 74–77, August 1995.

[26] R. G. Casey and E. Lecolinet. Strategies in character segmentation: A survey. *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, Canada*, pages 1028–1033, 1995.

[27] G. Kim and V. Govindaraju. A lexicon driven approach to handwritten word recognition for real-time applications. *IEEE Trans. on PAMI*, 19(4):366–379, April 1997.

[28] S. N. Srihari. Recognition of handwritten and machine-printed text for postal address interpretation. *Pattern Recognition Letters*, 14:291–302, April 1993.

[29] Marlon Abayan. A system for postal address recognition. M.S. Thesis, Computer Science, MIT, 1997.

[30] L. Xu, A. Krzyzak, and C. Y. Suen. Methods for combining multiple classifiers and their applications in handwritten character recognition. *IEEE Trans. Syst., Man, Cybern.*, 22(3):418–435, 1992.

[31] T. K. Ho, J. J. Hull, and S. N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. Pattern Anal. Machine Intell.*, 16(1):66–75, 1994.

[32] J. Kittler. Improving recognition rates by classifier combination. In *Intl. Workshop on Frontiers in Handwriting Recognition*, pages 81–102, Colchester, UK, Sept. 1996.

[33] J. Mao, K. Mohiuddin, and T. Fujisaki. A two-stage multi-network OCR system with a soft pre-classifier and a network selector. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, I:78–81, August 1995.