



National Library  
of Canada

Bibliothèque nationale  
du Canada

Acquisitions and  
Bibliographic Services Branch

Direction des acquisitions et  
des services bibliographiques

395 Wellington Street  
Ottawa, Ontario  
K1A 0N4

395, rue Wellington  
Ottawa (Ontario)  
K1A 0N4

You're... *Microfilms*

Vous... *Microfilms*

## NOTICE

## AVIS

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

If pages are missing, contact the university which granted the degree.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

# **HYBRID RECOGNITION OF HANDWRITTEN WORDED AMOUNTS ON CHEQUES**

Jean-Pierre Dodel

A Thesis in the Department of  
Computer Science

Presented in Partial Fulfillment of the Requirements  
for the Degree of Master of Computer Science at  
Concordia University  
Montreal, Quebec, Canada

August 1995

© Jean-Pierre Dodel, 1995



National Library  
of Canada

Acquisitions and  
Bibliographic Services Branch

395 Wellington Street  
Ottawa, Ontario  
K1A 0N4

Bibliothèque nationale  
du Canada

Direction des acquisitions et  
des services bibliographiques

395, rue Wellington  
Ottawa (Ontario)  
K1A 0N4

*Your file* *Votre référence*

*Our file* *Notre référence*

**The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.**

**L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.**

**The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.**

**L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.**

ISBN 0-612-10840-6

**Canada**

# Abstract

## Hybrid Recognition of Handwritten Worded Amounts in Cheques

Jean-Pierre Dodel

A hybrid classifier combining symbolic and neural routines to recognize unconstrained cursive script indicating the worded amount in bank cheques is proposed. The symbolic routines extract features outside the body of words, and classify letters at the beginning of words. Features extracted by the symbolic classifier are such as ascenders, descenders and loops. When these features provide insufficient information to classify an unknown word, the neural classifier is used to complete the recognition started by the symbolic classifier. Therefore, depending on the features extracted by the classifiers, some words are recognized entirely symbolically, some words entirely neurally, and the remaining, both symbolically and neurally. Results of experiments at the word level are provided.

keywords: Handwriting recognition, symbolic and neural classifiers, cheques, worded amount.

## Acknowledgments

I wish to express my sincere thanks to my supervisor, Dr. R. Shinghal of Concordia University for his direction, patience and encouragement, specially during my difficult times. My gratitude goes out to Ms. C. Nadal working at CENPARMI (Center for Pattern Recognition and Machine Intelligence, at Concordia University), for her advice and informative discussions with me on OCR (Optical Character Recognition) projects, at the beginning of my research. I wish to thank my friend Ms. Nina Hoffle, Ph.D. student at the Montreal Neurological Institute, for her help in collecting the specimens used for training the system. I would also like to thank Dr. Opatrny from the Computer Science Department at Concordia University, for helping me develop the regular grammar shown in Figure 6.2. Finally, I would like to deeply thank my parents for their invaluable moral support, advice, and encouragement throughout these years.

# Table of Contents

	<u>Page</u>
List of Figures .....	ix
<b>1. Introduction .....</b>	<b>1</b>
1.1 Manual processing of bank cheques .....	1
1.2 Automated processing of bank cheques.....	1
1.3 Current Advances in Cursive Handwriting Recognition .....	5
1.3.1 Simon .....	5
1.3.2 Cohen .....	8
1.3.3 Barrière and Plamondon .....	9
1.3.4 Gorsky.....	10
1.4 Scope of the thesis .....	10
<b>2. Preprocessing and Overall Architecture of the System .....</b>	<b>12</b>
2.1 The vocabulary .....	12
2.2 Finding the body-ceiling .....	13
2.2.1 The body-ceiling window .....	15
2.2.2 Adjusting the window height .....	16
2.3 Text segmentation .....	19
2.4 The classifiers: overall view.....	24
<b>3. The Symbolic Classifier .....</b>	<b>40</b>
3.1 Identifying ascenders and descenders .....	40
3.2 Identifying notable letters at the beginning of words .....	43

3.3	Finding the separation point between the notable letter and the remaining part of the word to its right .....	46
3.4	Categorizing notable letters .....	55
3.5	Words recognized entirely by the symbolic classifier (word category 1) .....	76
3.5.1	Recognizing "eight" .....	76
3.5.2	Recognizing "eighty" .....	80
3.5.3	Recognizing "and" .....	81
3.5.4	Recognizing "One" .....	81
3.5.5	Recognizing "Hundred" .....	82
3.5.6	Recognizing "Thousand" .....	82
3.5.7	Recognizing "dollars" .....	83
3.5.8	Recognizing "forty" .....	91
3.5.9	Recognizing "fifty" .....	91
4.	The Neural Classifier .....	93
4.1	Input Patterns .....	95
4.2	Output Patterns .....	96
4.3	Neurodes and Connections .....	98
4.4	Words recognized entirely by the neural classifier (word category 2) .....	100
4.5	Preparing the input to the neural network .....	100
4.5.1	Finding the start and end points of word segments .....	100
4.5.2	Normalizing the word segments .....	102
4.5.3	Anchoring the input pattern.....	110
4.6	Feeding word segments through the neural network .....	110
5.	Symbolic/Neural Classification .....	111

5.1	Words recognized jointly by the symbolic and neural classifiers (word category 3) .....	111
5.2	Preparing word segments from Word category 3, to be sent to the neural network.....	114
5.2.1	Finding the start and end points of word segments .....	114
5.2.2	Size-normalization .....	120
5.3	Classifying word segments and patching-up the final word .....	120
5.4	Setting an output mask.....	122
6.	Experimental Results .....	126
6.1	The Grammar checker .....	126
6.1.1	Examples of grammatical forms accepted by the grammar checker .....	128
6.1.2	Examples of grammatical forms rejected by the grammar checker .....	131
6.2	Examples of classification .....	132
6.2.1	Word category 3 (type 1 word) .....	132
6.2.2	Word category 3 (a different type 1 word example) .....	132
6.2.3	Word category 3 (type 2 word) .....	132
6.2.4	Word category 3 (type 1 word) .....	132
6.2.5	Word category 2 (words recognized entirely by the neural network) .....	132
6.3.1	Training set.....	132
6.3.2	Testing set .....	132
6.3.3	Recognition speed and memory requirements .....	144
6.4	Integration with a digit classifier .....	147
6.4.1	Brief description of the digit classifier .....	147



6.4.2	The resolution strategy .....	147
6.4.3	Results of integration .....	153
7.	Concluding Remarks .....	157
	Suggestions for future research.....	158
	References .....	160
	Appendix .....	164

# List of Figures

<u>Figure number</u>	<u>Page</u>
1.1 : The three handwriting styles most often found .....	2
1.2 : An example of worded and numeric amounts on a cheque .....	4
1.3 : Terminology used for word components.....	6
1.4 : Morphological simplicity of ascenders compared to body area strokes .....	7
2.1 : Dividing the text image into windows .....	14
2.2 : Finding the body-ceiling window .....	16
2.3 : Increasing window size to find body-ceiling window .....	18
2.4 : Each scanned row returns an estimate of where the current word ends (P1) and where the next word begins (P2).....	22
2.5 : Segment of scanned amount "seven thousand" .....	23
2.6 : An example of incorrect segmentation by the algorithm <i>Find_Word_Limits</i> discussed in section 2.3 .....	25
2.7 : Two main components of the word classifier.....	27
2.8 : Terminology and graphic symbols used in the decision tree of Figures 2.9 to 2.18.....	29
2.9 : First steps in the traversal of the decision tree .....	30
2.10 : Leaves obtained from node <i>B01</i> .....	31
2.11 : Leaves obtained from node <i>B02</i> .....	32
2.12 : Leaves obtained from node <i>B03</i> .....	33
2.13 : Two steps down the tree from node <i>B04</i> .....	34
2.14 : Two steps down the tree from node <i>B05</i> .....	35

2.15 :	Three steps down the tree from node <i>B06</i> .....	36
2.16 :	Leaves obtained from node <i>B07</i> .....	37
2.17 :	Leaves obtained from node <i>B08</i> .....	38
2.18 :	Leaves obtained from node <i>B09</i> .....	39
3.1 :	Ascender and descender windows .....	41
3.2 :	Identifying the presence of notable letter "t" in "twenty" .....	44
3.3 :	Coming across a dot when looking for a notable letter .....	47
3.4 :	Tracing around the edge of the black pixel cluster to find its perimeter .....	48
3.5 :	Skipping the dot and resuming the search for the notable letter .....	49
3.6 :	Example of inaccurate separation point between a notable letter and the letter following it .....	51
3.7 :	Finding estimates for the separation point between the notable letter and the remaining of the word (example 1).....	52
3.8 :	Finding estimates for the separation point between the notable letter and the remaining of the word (example 2).....	53
3.9 :	Problems finding the separation point between the notable letter and the remaining of the word in words starting with "th" .....	55
3.10 :	Separation point between notable letter and the remaining of the word for words starting with "th" .....	57
3.11 :	Example of word with lowercase "f" .....	58
3.12 :	Algorithm (1) <i>Notable_Category_2</i> , applied to recognize an "S" .....	60
3.13 :	Algorithm (2) <i>Notable_Category_2</i> , applied to recognize an "S" .....	61
3.14 :	Uncommon case of disproportionned "S" where algorithm <i>Notable_Category_2</i> fails to recognize the notable letter as an "S" .....	63
3.15 :	Identifying the first letter in "eleven" as lowercase "e" .....	65
3.16 :	First feature to be found in a capital "O" (notable category 4) .....	67

3.17	: Third feature to be found in a capital "O" (notable category 4) .....	70
3.18	: More common and less common types of "h" in "th" .....	72
3.19	: Common feature in "E" (notable category 5), "th" (notable category 5) and "O" (notable category 4) .....	73
3.20	: Words (category 1) recognized entirely by the symbolic classifier .....	77
3.21	: Search boundaries inside the ascender/descender windows for ascenders and descenders.....	78
3.22	: Recognizing "eight" .....	79
3.23	: Tracing over the right "l" .....	85
3.24	: Recognizing different types of "l"s .....	87
3.25	: Identifying loops by approximating the right "l" loop with an ellipse .....	90
4.1	: Typical 1-hidden layer backpropagation network .....	94
4.2	: Example of pattern ("nine") in input matrix .....	97
4.3	: Overview of the neural network architecture .....	99
4.4	: Words recognized entirely by the neural classifier (category 2 words).....	101
4.5	: Normalizing word segments while preserving aspect ratios.....	103
4.6	: Normalizing wide word segments .....	104
4.7	: Example of a word segment to be scaled down to fit inside the neural network's input matrix .....	105
4.8	: Selecting the 22 uniformly distributed rows to be deleted .....	107
4.9	: Selecting the 51 uniformly distributed columns to be deleted.....	108
4.10	: Scaled down image now fits inside the input matrix .....	109
5.1	: Type 1 words in word category 3, where one segment is recognized by the neural classifier .....	112

5.2	: Type 2 words from word category 3, where 2 segments are recognized by the neural classifier .....	113
5.3	: Tracing the right side vs tracing the left side of the "t", to find the word segment start point.....	117
5.4	: Different segments for the word "thirty" sent to the neural classifier....	119
6.1	: Truth table for grammar flags <i>flag_10_90</i> , <i>flag_100</i> , and <i>flag_1000</i> .....	129
6.2	: State diagram of finite automaton representing the regular grammar suggested by Dr. Opatrny .....	130
6.3	: Input word yielding output shown in Figure 6.3 .....	133
6.4	: Example of neural net output when the input is Figure 6.2.....	134
6.5	: Type 1 word of category 3: "sixty" without a notable letter at the beginning .....	135
6.6	: Output when input is Figure 6.4 .....	136
6.7	: Type 2 word of category 3 .....	137
6.8	: The output when input is Figure 6.6.....	138
6.9	: Type 1 word from category 3 with a notable letter at the beginning .....	140
6.10	: The output when input is Figure 6.8.....	141
6.11	: Recognizing a word in word category 2.....	143
6.12	: The output when input is Figure 6.10.....	144
6.13	: Training specimens.....	145
6.14	: Testing specimens.....	146
6.15	: Other examples of reliable and unreliable mismatches .....	150
6.16	: Confusion matrices for the digit and word classifiers .....	151
6.17	: Example of mismatch resolution when the word classifier is assumed to be wrong .....	154
6.18	: Example of a mismatch resolution when the digit classifier is assumed to be wrong .....	155

6.19 : Table showing the recognition rates of the word classifier on the  
training and testing sets ..... 156

# Chapter 1

## Introduction

### 1.1 Manual processing of bank cheques

Today, charter banks in Canada process over one billion cheques every year (Gordon, 1995). For example, from November 1994 to January 1995 the Royal Bank processed 348,801,000 cheques from which 70,632,203 were either personal cheques or cheques written on savings accounts with chequing privileges; these 70,632,203 cheques were all handwritten except for rare specimens (1% approximately) which were typed or printed (King, 1995). Manual cheque processing is a slow and expensive process, requiring a bank department to collect all cheques received during the day, and manually type in the amount written on each cheque into a computer (Gordon, 1995). Automating cheque processing should increase the speed of processing, and yet reduce costs.

### 1.2 Automated processing of bank cheques

In the pursuit of better and faster services, banks have partially or totally automated many of their tasks, for example, developing the Automated Teller Machine (ATM). However, these tasks were automated efficiently due to their relative mechanical simplicity. Automating the process of reading a handwritten cheque, however, is far from being a simple mechanical task; human writing has so many styles and variations that creating a well defined model for recognizing handwriting is a complex task requiring expensive hardware and software. Figure 1.1 shows the three cursive handwriting styles often found (Tappert, 1984).

*Sixteen thousand and Forty five*

Handwriting where  
letters in a word are mostly joined

*seven hundred twelve*

Handwriting where  
letters in a word are mostly separated

*Six thousand and nine hundred*

Handwriting where letters in a word are sometimes  
joined and sometimes separated

Figure 1.1: The three handwriting styles most often found.



Nonetheless, the growing need for automation in industrialized societies, has resulted in the ever growing emergence of faster processors, which have brought the execution of complex algorithms such as cursive handwriting recognition, within the realm of real-time applications.

This thesis proposes a method to recognize cursive handwriting that indicates the *worded amount* on a bank cheque (see Figure 1.2). The recognition process can be divided into three phases: During the first phase (called preprocessing), the digital image of the worded amount will be segmented into the individual words. The second phase consists of extracting features from each word, and the last phase will analyze the extracted features and identify (that is, classify) each word. During the last phase, two types of classifiers are used to recognize a word: a *symbolic classifier* and a *neural classifier*. A symbolic classifier is one that manipulates symbols such as numbers 0 to 9 or non-numeric symbols such as procedures and mathematical logic operators used for decision making (Shinghal, 1992). A neural classifier, however, recognizes objects much like the human brain, using a network of neurons to learn and recognize the name and shapes of certain objects, forming a "general" idea of how each shape looks like.

In Figure 1.2, we also define a key term called the *baseline*, which will be used extensively in future discussions. The baseline is the horizontal line above which all words are written. This line is present in most cheques to guide a person writing the worded amount. The *numeric amount* on a cheque, also shown in Figure 1.2, is the amount written in digit form. Our method takes advantage of the strengths of each of the symbolic and neural approaches (Sun and Bookman, 1993; Nagy, 1992; Shinghal, 1992) to focus each technique on sections of words which it can classify better. Unlike the other approaches we will discuss shortly,

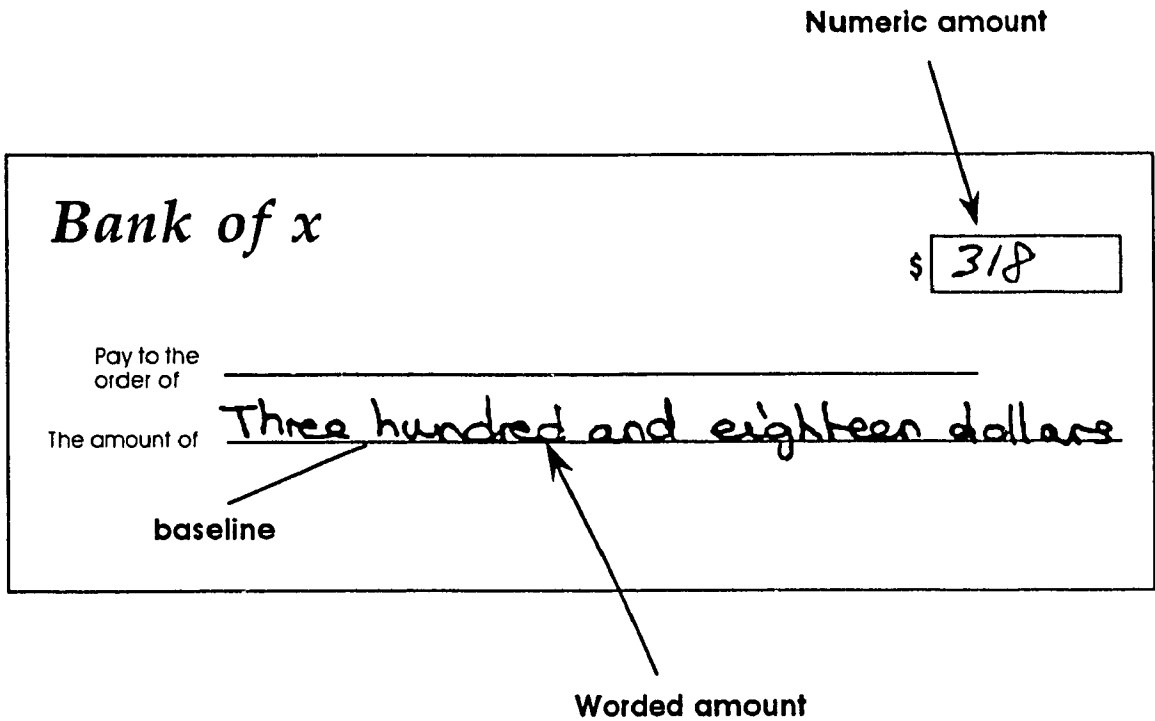


Figure 1.2: An example of worded and numeric amounts on a cheque.

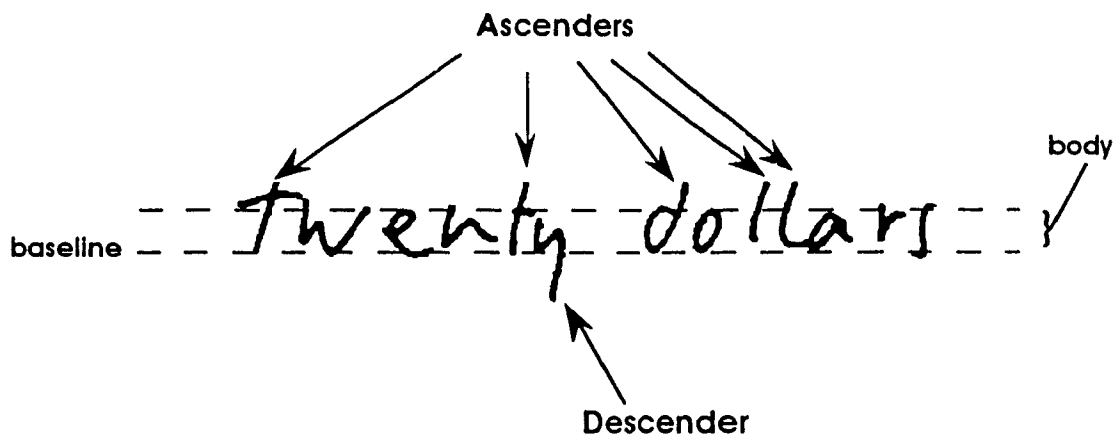
our classifier distinguishes sections which typically experience character distortions such as the body, from more consistent sections such as those having *ascenders* and *descenders* (Figure 1.3). This distinction can be made due to the fact that ascenders and descenders are, in general, morphologically simpler and better isolated shapes than those inside the *body*. For instance, the ascenders on both letters "d" and the "h" shown in Figure 1.4, stand out from the rest of the word, and the simplicity of the vertical stroke makes it relatively easier to identify symbolically compared to other letters in the word.

Therefore the symbolic classifier will analyze the more consistent parts of words, as algorithmic (symbolic) classification performs best when features are consistent from one sample to another. The less consistent parts words will be classified with the help of the neural classifier. The ability of neural networks to learn and classify even from noisy and distorted data, makes them better suited to deal with letter distortions.

### **1.3 Current Advances in Cursive Handwriting Recognition**

In this section, we will cover some of the work done in recent years in this field, showing only the important characteristics of each system such as the classification technique, size of the vocabulary, writing constraints, number of writers involved in training and testing, and recognition rate. Most of these works deal with the recognition of the worded amount on cheques. We shall refer back to this section at the end of this thesis to compare our results to these.

**1.3.1** Simon (1992) worked on the classification of the following French words: "un", "deux", "trois", "quatre", "cinq", "six", "sept", "huit", "neuf", "dix", "onze", "douze", "treize", "quatorze", "quinze", "seize", "vingt", "trente",



**Figure 1.3:** Terminology used for word components.

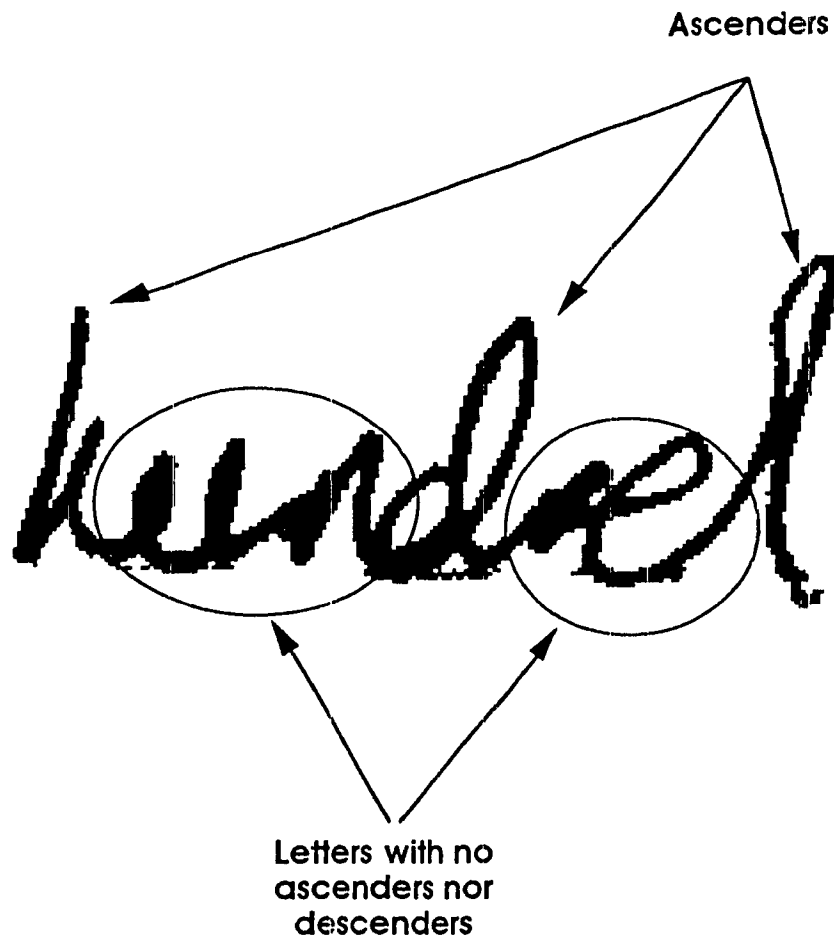


Figure 1.4: Morphological simplicity of ascenders compared to body area strokes.

“quarante”, “cinquante”, “soixante”, “cent”, “mille”, “million”, and “milliard”. His technique consisted of representing words with an abstract graph built from features found in words. These features were such as line forks or crossings, and loops in a word. By selecting robust anchor features, candidate words were selected from the vocabulary and examined to see which one fit best the description given by the graph. When samples were collected for testing and training, the writers were constrained to write without capital letters. Each writer wrote a set of 25 words twice: once on a sheet of paper used for training the system, and the second time on another sheet of paper used for testing. In other words, their tests were done with the same writers used for training. Only 8 writers were used for training and testing. Results showed an average of 76% correct recognition of words in the test set.

1.3.2 Cohen (1994) worked on the recognition of bank cheque amounts. His approach focused on the recognition of the numeric amount, relying very little on the worded amount. Although the recognition of the numeric amount is beyond the scope of this thesis, we shall compare his cheque-level recognition rate to ours in a future chapter. When collecting the test samples, 66 cheques were written without imposing any constraints to the writers. At the cheque-level, his results showed 14% correct recognition of the test data, using only the digits to recognize the amount. He obtained between 18% and 46% correct recognition when the expected amounts and the worded amount were used to add contextual information to the numeric recognition. The expected amount is, for instance, the amount which a phone company would expect to receive from each customer, based on the last bill sent to him or her. In the 18% to 46% case, however, the digit recognizer used the worded amount only 10% of the time due to “poor quality handwriting in worded amounts and the quality of the word

recognition algorithm used". Cohen did not describe how the classifier was trained.

1.3.3 Barrière and Plamondon (1992) worked on an on-line system which recognized words by first identifying its letters (letter-based recognition). On-line systems use an electronic tablet to enter the handwriting. Off-line systems such as ours, takes the input image from a sheet of paper which is scanned to be put in a digital format. On-line systems therefore have significant advantages over off-line recognition, such as being able to time each stroke, obtain the order in which the strokes were drawn and measure pressure changes on the pen. Barrière and Plamondon's used as vocabulary the French dictionary "Larousse de Poche", containing approximately 30,000 common nouns. Their recognition technique consisted of slowly moving a window across the word, identifying letter sequences. From the letter sequences obtained, the word length was obtained and candidate letters for each position within the word would be proposed. Training consisted of displaying each word in the vocabulary on a screen and having a human tutor indicate where each letter began and ended. From this information the system was able to make models for each pattern in the set of features. When the window moved across the word, it could sometimes be on top of an entire letter or covering the end of a letter and the beginning of the next one. In the latter, the system still managed to extract features; having previously learned what each features looks like, it could also know what the end of a feature and the beginning of another looked like. When samples were collected for testing and training the following constraints were imposed upon the writers:

- Lowercase letters only.
- No component could be added to a letter after another letter had been started. In other words, if we were writing the word "piéton"

and we were up to letter "o", we would not be allowed to go back to letter "t" to add an extra stroke such as the horizontal crossbar.

Only six writers were used to train the system. The same writers were used for testing. Therefore, as with Simon's (1992) system, the classifier is writer-dependent. Results showed 79% correct recognition of letters in a test set of 250 words chosen randomly from the French dictionary.

**1.3.4** Gorsky (1994) worked on the recognition of worded amounts on bank cheques. His vocabulary, however, was smaller than ours, it contained 23 words. His recognition technique consisted of approximating words by a set of segments (similar to a skeletonization process). Each segment was mapped into a parameter space called *hollograph*. The final classification was done by comparing the obtained hollograph with the closest prototype generated during training for each word in the vocabulary. The hollographic prototypes were created from a wide variety of writing styles, and at training and testing time, writers were not given any writing constraints. His results showed a 74% correct word recognition rate on a test set of 500 cheques. His recognition speed, however, was slow, taking an average of 20 seconds for each 4-word cheque, excluding preprocessing (word segmentation). These tests were performed on an IBM RT-6500.

## **1.4 Scope of the thesis**

In this thesis, we propose a hybrid approach to classifying unconstrained handwritten worded amounts on cheques. Our objective is to develop two complementary classifiers that can, together, recognize a wide variety of cursive handwriting styles for the words in the 31-word vocabulary, which will be covered in section 2.1.



In the following chapter, we will look at how the worded amount is prepared for recognition; this is the preprocessing stage. We will also have, in the same chapter, an overview of the classifiers to have an intuitive idea of how they work together. In Chapter 3, we describe the symbolic classifier, and in Chapter 4, the neural classifier. Chapter 5 will describe how and when both classifiers work together to complement each other. In Chapter 6, we give our experimental results at the word level and at the cheque level. Finally, we give our concluding remarks in Chapter 7.

# Chapter 2

## Preprocessing and Overall Architecture of the System

### 2.1 The vocabulary

In this thesis, we are focusing on the automated reading of only worded integer amounts. Our working domain is therefore the following 31-word vocabulary: *one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, hundred, thousand, and, dollars*. We thus assume the maximum alphabetic amount to be “nine hundred and ninety nine thousand nine hundred and ninety nine dollars”. We consider this to be a reasonable assumption since cheques for higher amounts are usually corporate cheques that are typed or printed. Around this vocabulary, we built a grammar checker which will help us minimize cheque-level misrecognitions and reject ambiguous grammatical forms of writing worded amounts. The grammar checker will be covered thoroughly in a future chapter.

Following our introductory discussion of Chapter 1, we give below the definitions of word components and key terms which will be used in this thesis (Figures 1.2 to 1.4):

- **text image:** digitized version of the original text (worded amount) written on the cheque. The image is a black and white picture of the worded amount scanned at a 300 dots per inch resolution.
- **baseline:** The baseline is the horizontal line above which all words are written.

- **ascender:** Section of letters such as in *d* or *t* which rise above lowercase letters such as *x* or *u*.
- **descender:** Section of letters such as *y* or *g* which extend below the baseline.
- **body:** Section of the word which is between the ascenders and descenders sections. The same term *body* can be used for a letter when referring to the section of the letter which is between the ascender and descender section.
- **body-height:** Defines the height in rows of the body for all words in the worded amount. For example, in the word "forty", the body-height is the height of letters having neither ascenders nor descenders, in other words, the height of letters "o" and "r".
- **body-ceiling:** top row in the body.

## 2.2 Finding the body-ceiling

The first objective in preprocessing is to locate the baseline and give an estimate of the body-ceiling. The baseline is assumed to be at a fixed place, as all cheques from the same bank have that line at the same location.

When a person writes the worded amount on a cheque, it is inevitable to find height variations amongst letters. For instance, if someone wrote "eighteen dollars" (see Figure 2.1), there would be height differences between letters "e", "i", "n", "o", "a", "r" and "s". The body-ceiling will therefore be the result of an approximation, an estimate of the average height characterizing the letters with neither ascenders nor descenders.

The first step in finding the body-ceiling consists in dividing the text image from top to bottom into windows. Each window will initially contain 5 adjacent rows. Therefore if, for instance, the whole text image has 71 rows, there will be

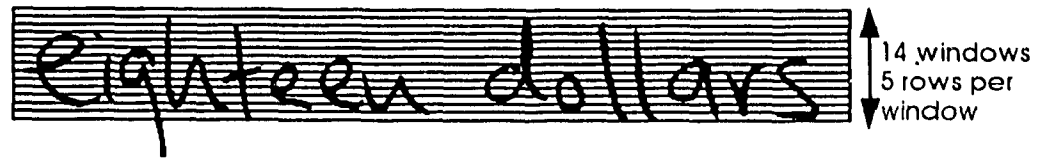


Figure 2.1: Dividing the text image into windows.

$\lfloor \frac{71}{5} \rfloor = 14$  windows (see Figure 2.1). It is important to point out that flooring  $71/5$  leaves 1 row out from the 14 windows, as  $14 \times 5 = 70$  rows; generally speaking, all rows left out of the windows will be in the bottom section of the text image and will be ignored as they are in a region where the body-ceiling is least expected to be found. The idea behind this technique is to enclose within one of the windows, the top section of the body of words, where there is a drastic increase in black pixel concentrations compared to the space above that top section.

The next step is to count the number of black pixels (also referred to as black pixel density) inside each window, starting from the top window and proceeding downwards (lines [3], [4] and [5] in algorithm *Find\_Body\_Ceiling* provided shortly below). The body-ceiling is found in a window having a number of black pixels greater than the window immediately above it by a heuristic threshold  $\Delta$  defined as follows:

$$\Delta = \frac{\text{number of pixels in window}}{3}$$

In algorithm *Find\_Body\_Ceiling*, these adjacent windows are called *window\_2* and *window\_1* respectively. This difference in black pixel densities (line [6]) is a function of the window height (initially set to 5 rows) and the length (in columns) of the text image.

**2.2.1 The body-ceiling window:** We define the *body-ceiling window* as the window where the  $\Delta$  threshold is reached. Figure 2.2 shows how the text image is divided into windows, and where the body-ceiling window is located. Once this body-ceiling window is found, the exact body-ceiling value is defined to be the middle row inside the body-ceiling window. The body-ceiling is therefore a

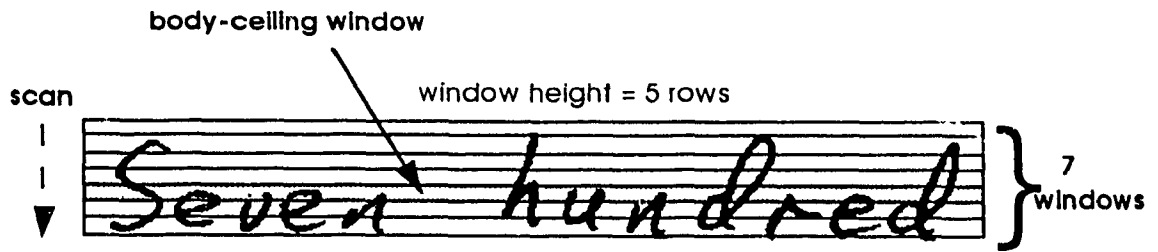


Figure 2.2: Finding the body-ceiling window.

row defining the upper limit of the body. So, if for instance, the window having the  $\Delta$  difference is the 10th window from top to bottom, and each window has 5 rows, the body-ceiling would correspond to row number  $9 \times 5 + \left\lfloor \frac{5}{2} \right\rfloor = 47$ , from top to bottom; the  $9 \times 5$  factor counts the rows in the first 9 windows, and  $\left\lfloor \frac{5}{2} \right\rfloor$  counts half the rows in the 10th window. From this approximation, it is clear that the number of rows assigned for each window is directly related to the accuracy of the body-ceiling; the larger the number of rows per window, the less accurate the body-ceiling will be.

**2.2.2 Adjusting the window height (when necessary):** All windows have the same number of rows which is initially set to a heuristic value of 5 (step [1] in algorithm *Find\_Body\_Ceiling* below). It was found that for values lower than 5, in most cases the windows would not contain enough black pixels to make a good approximation of the body-ceiling. The initial value of 5 rows, if it suffices, will optimize the accuracy of the body-ceiling since the approximation error will, in the worst case, be 4 rows. However, 5 rows per window is sometimes too small for the  $\Delta$  difference to be found. For instance, as shown in Figure 2.3, there are cases where the body of letters varies greatly in height and the body-ceiling window needs to hold a greater number of rows in order to contain the majority of black pixels present in the top section of the body. For example, looking closer at the top portion of Figure 2.3, we can see that none of the windows can really claim to be the body-ceiling window, the difference in heights of letters with no ascenders nor descenders is too important, and there cannot be a drastic black pixel density difference  $\Delta$ , from one window to the next one. In this case, the algorithm will increase the window height by 1 row and try to find  $\Delta$  from top to bottom of the text image, once more. It will keep increasing the window height,

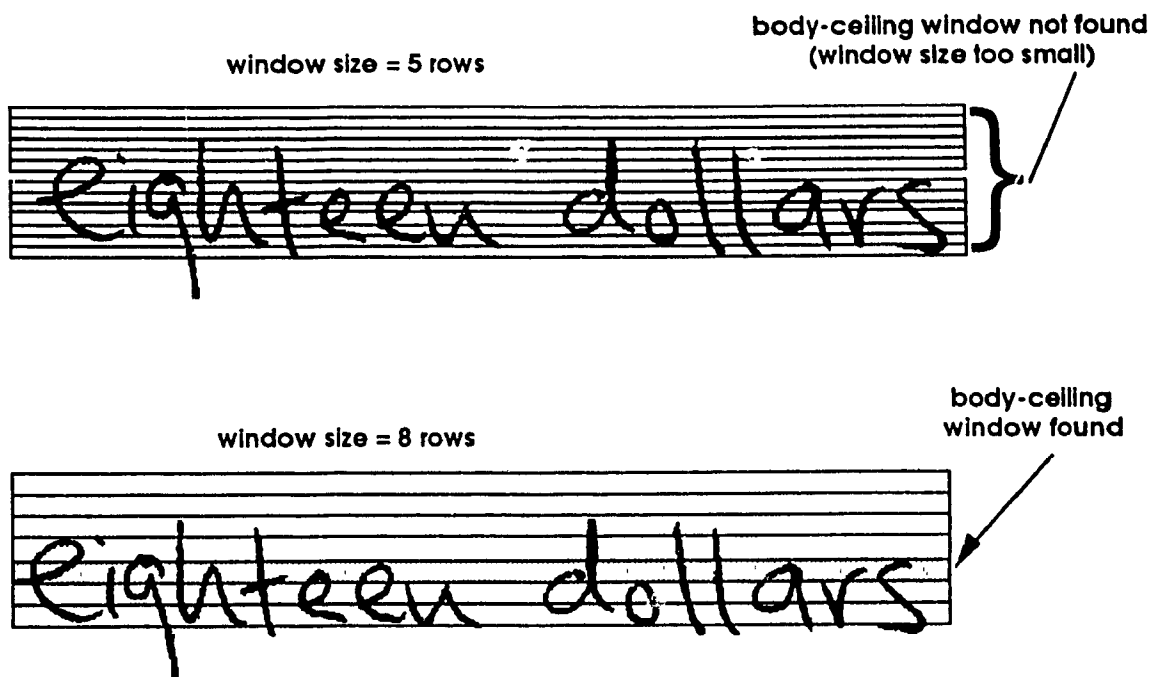


Figure 2.3: Increasing window size to find body-ceiling window.



one row at a time (lines [2] and [7]), until  $\Delta$  is found or until the window size exceeds 15 rows in which case it will abort the search (line [8]). In most cases, the window height will not exceed 9 rows. As shown in the bottom portion of Figure 2.3, a window height of 8 rows was sufficient to find  $\Delta$ . The following pseudo-code outlines the main idea behind the algorithm.

*Algorithm Find\_Body\_Ceiling;*

```

[1]   window_height = 5 rows
      bottom_window = last window in the bottom section of the text image
[2]   loop until window height exceeds the maximum size of 15 rows
[3]       window_1 = top window
[4]       window_2 = window immediately below window_1
[5]       loop until window_2 = bottom_window
[6]           if ( Pixel_Density(window_2) - Pixel_Density(window_1) )  $\geq$ 
               $\Delta$  then
                  return body-ceiling = middle row of window_2
              else
                  window_1 = window_2
                  window_2 = window immediately below window_1
              endloop
[7]       increment window_height by 1 row
      endloop
[8]   return Error (cannot find the body-ceiling)
endAlgorithm

```

The above algorithm was developed using original techniques. All algorithms described in this thesis were developed by the author, unless otherwise specified.

## 2.3 Text segmentation

The second objective in preprocessing is to estimate where each word starts and finishes on the worded amount. Known as text segmentation, a recent reference on

it is by Seni and Cohen (1994). Having obtained the body-ceiling, word segmentation will consist of looking for blank spots inside the body, which typically indicate a separation between two words. These blank spots are defined as areas with a high density of white pixels within the body. The beginning of the blank spot (or word separation) marks the end of the word to its immediate left, and the end of the blank spot indicates the beginning of the word to its immediate right. The following pseudo-code outlines the main steps involved in finding these blank spots.

Definition of some important notation:

*current\_word(beginning)* is the column in the text image where the current word to be recognized starts.

*Minimum\_word\_length* = (number of rows in the body) x 2

*column(P)* = column in the text image where a pixel P is located.

**Algorithm Find\_Word\_Limits(current\_word)**

[1] Choose 10 equally distant rows starting from the top of the body and reaching the baseline. Choosing these rows is done as follows:

$$\text{space between adjacent rows} = \left\lfloor \frac{\text{rows in body}}{10} \right\rfloor$$

row #0 = body-ceiling

row #n = (  $\left\lfloor \frac{\text{rows in body}}{10} \right\rfloor \times n$  ) rows down from row #1. (0 ≤ n ≤ 9)

[2] P2 = current\_word(beginning)

[3] for each row selected do

loop

[4] scan row from P2 to end, stopping when a sequence of white pixels at least 1/2 the length of the previous word separation<sup>1</sup> is found.

[5] P1 = first pixel in the sequence found.

[6] P2 = last pixel in the sequence found (previous P2 gets replaced by this new value)

[7] P3 = white pixel in the middle between P1 and P2.

---

<sup>1</sup> Since the first word in the worded amount has no word preceding it, the "previous word separation" is initially set by default to 1.5 times the body-height.

[8] Let us define the rectangle  $R$  with the following columns and rows:

- leftmost column = (column of  $P3$ ) - 2
- rightmost column = (column of  $P3$ ) + 2

The heuristic values “-2” and “2” are used to set the left and right margins respectively, of  $R$  centered around  $P3$ .

- top row = baseline - body\_height - 8
- bottom row = baseline + 8

The heuristic values “-8” and “8” are used to set the top and bottom margins respectively, of  $R$  centered around  $P3$ .

Roughly speaking,  $R$  is a 5-column rectangle (there are 5 columns between the leftmost and the rightmost columns) centered horizontally at  $P3$ . Its height is variable depending on the body-height.

[9] if all pixels inside  $R$  are white

we have found a potential word separation starting at  $P1$  and finishing at  $P2$ , according to the current row being scanned.

exit loop

endloop

endfor

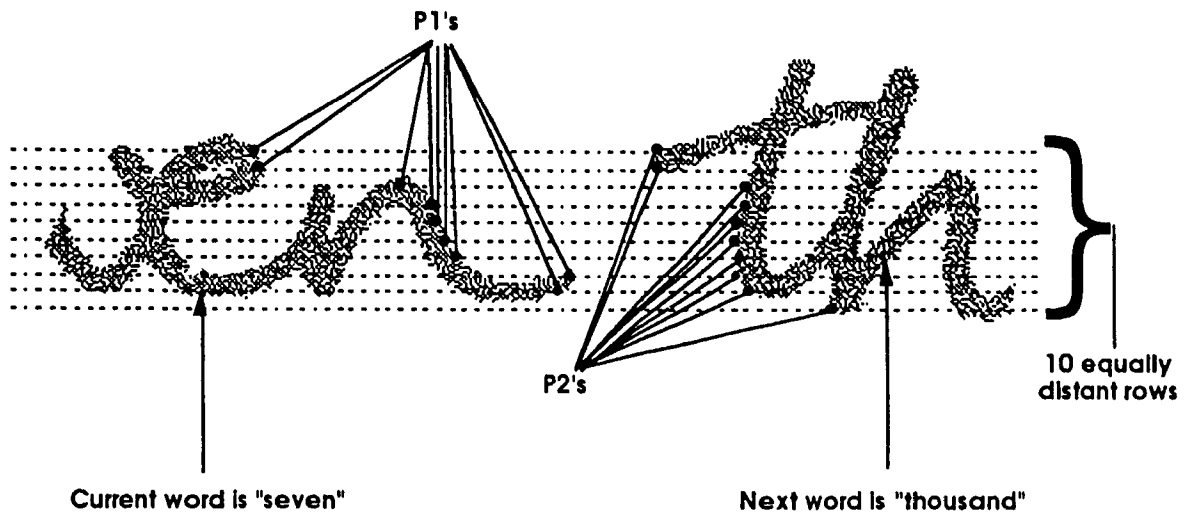
At this point, each row has returned its estimate on where the current word ends ( $P1$ ) and where the next word begins ( $P2$ ) (see Figure 2.4). We must now choose the optimal  $P1$  and  $P2$ .

[10]  $current\_word(end) = optimal\ P1 = rightmost\ P1$  in area of high density of  $P1$ 's. Delimiting this area (call it  $A$ ), will be discussed shortly. See left side of Figure 2.5.

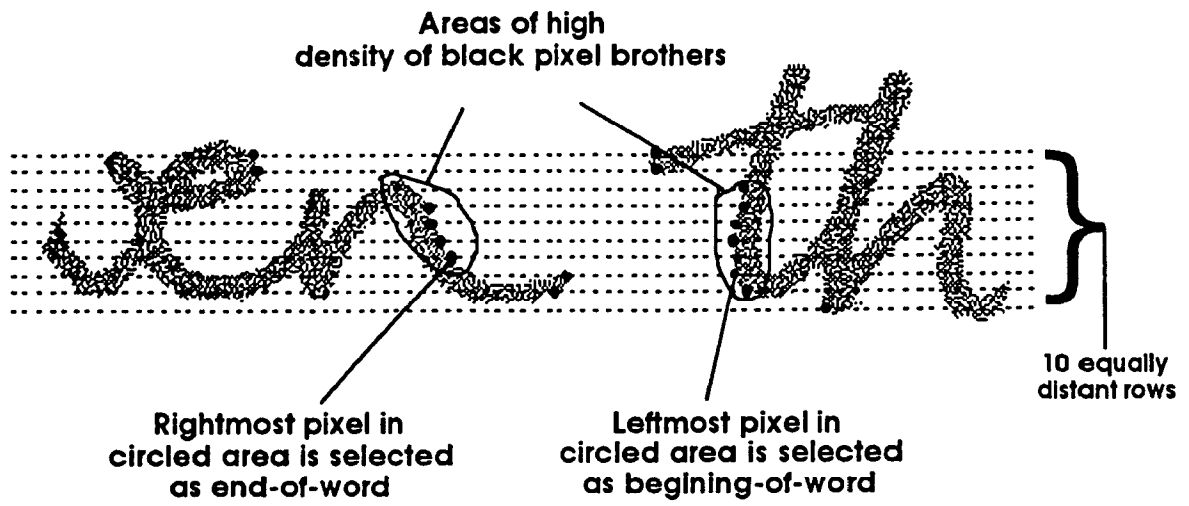
[11]  $next\_word(begin) = optimal\ P2 = leftmost\ P2$  in area of high density of  $P2$ 's. This area is defined analogously to the one with  $P1$ 's. See Figure 2.5.

**endAlgorithm**

Let us now explain more specifically how this area  $A$  is defined: We call a *brother*, a pixel  $P1$  which is 20 columns or less apart from another  $P1$ . If we call these two  $P1$ 's  $P1^*$  and  $P1^{**}$  respectively, we can therefore say that  $P1^*$  and  $P1^{**}$  are



**Figure 2.4:** Each scanned row returns an estimate of where the current word ends (P1) and where the next word begins (P2).



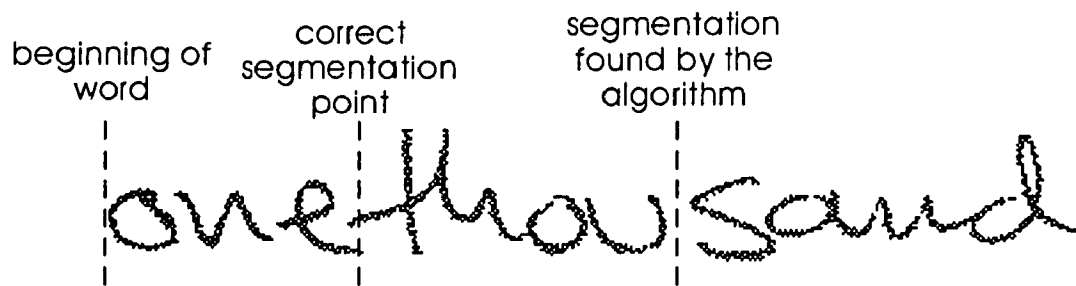
**Figure 2.5:** Segment of scanned amount "seven thousand". Finding the end and the beginning words.

brothers. For every P1, the segmentation algorithm will compute the number of brothers P1 has; we shall retain the P1's having the highest and the next-to-highest number of brothers. The area *A* will be delimited by the retained P1's. Therefore if, for instance, 7 and 5 are the highest and next-to-highest number of brothers respectively found amongst P1's, those P1's having 7 or 5 brothers will constitute area *A*. Finding the corresponding *A'* for P2's is done analogously to finding *A* for P1's. In Figure 2.5, both areas *A* and *A'* are illustrated as the "Areas of high density of black pixel brothers".

The above heuristic algorithm fails to separate complex cases such as the one shown in Figure 2.6. In this figure, the lower stroke from the "e" in "one" stretches to the right, beyond the leftmost column of the "t", leaving no white pixel columns between the "e" and the "t". Between the "u" and the "s" in "thousand", however, the separation reveals white pixel columns which, in this case, is interpreted as a word separation. As defined in algorithm *Find\_Word\_Limits*, a rectangle *R* (five adjacent columns) with only white pixels must be present for that region to be considered as a word separation.

## 2.4 The classifiers: Overall view

The classification system to read worded amounts, described in subsequent chapters, consists of two classifiers: A symbolic classifier and a neural classifier. In general terms, the symbolic classifier can be seen as having two major tasks: the first one is to detect features corresponding to ascenders and descenders of specific letters in a word, the second task is to identify the presence of *notable letters* (letters with ascenders present at the beginning of words), and classify each into one of the 9 categories defined for notable letters. A notable letter can



**Figure 2.6:** An example of incorrect segmentation by the algorithm *Find\_Word\_Limits* discussed in section 2.3.

be, for example, a capital (uppercase) "E" at the beginning of word "Eighty", or it could also be both letters "t" and "h" from the word "thousand". In more formal terms, if we divide the image of a word from left to right in 3 equal sections (left section, middle section and right section), a letter is defined as "notable" if it has an ascender in the leftmost section of the word. This 1/3 ratio may vary depending on the features found in the word. Details will be given in a future chapter.

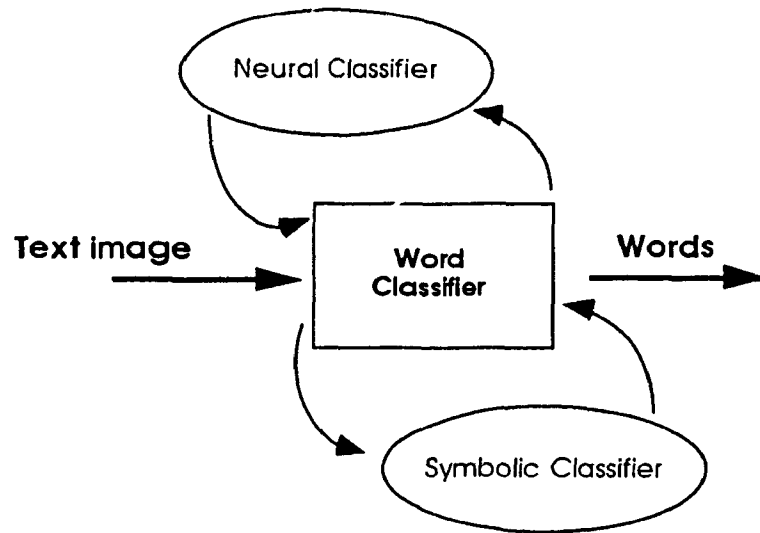
The neural classifier, is a network of neurodes that takes an image as input and produces an output from it. The network will be used whenever the symbolic classifier is unable to find enough features to classify the entire word. The neural network will therefore receive its input from the symbolic classifier, and return its results. In this way, the symbolic and neural classifiers work together to recognize a word (see Figure 2.7).

From our 31-word vocabulary described earlier, each word will be classified through one of the three different schemes of symbolic/ neural interactions.

- Word category 1: Words classified entirely by the symbolic classifier (will be covered in detail in Chapter 3).
- Word category 2: Words classified entirely by the neural classifier (will be covered in detail in Chapter 4)
- Word category 3: Words jointly classified by the symbolic and neural classifiers (will be covered in detail in Chapter 5)

This categorization will depend on the features found (if any) by the symbolic classifier. Generally speaking, there are words which are classified entirely from





**Figure 2.7:** Two main components of the word classifier

ascenders and descenders features (word category 1). Other words have no ascenders nor descenders; these will be classified entirely by the neural classifier (word category 2). Finally, some words have little ascender/descender information, needing some parts to be classified by the neural classifier (word category 3).

In order to get an intuitive understanding of how the overall classification scheme works, we will look at a decision tree having as root node, the unknown word to classify. Figure 2.8 shows the legend used for representing *not-leaf*, *clustered-leaf* and *single-leaf nodes* in the tree. As is customary, a leaf in a decision tree is a terminal point, and in this case it corresponds to the classifier's output to the given unknown word. Each word in the vocabulary is therefore contained in either a clustered-leaf node or a single-leaf node. The tree was broken up into several sections due to its large size. Figures 2.9 to 2.18 make up the set of diagrams illustrating the different sections of the tree. These figures should be read from left to right as opposed to the more common top-down form.



*Non-leaf node:* all nodes in the tree, except for single-leaf nodes and clustered-leaf nodes. Each of these nodes will represent a test or an action made on the unknown word



*Clustered-leaf node:* These round-cornered rectangles group many leaves in the decision tree. One of the leaves is selected according to the output given by the neural network.



*Single-leaf node:* Node containing only one leaf from the decision tree.

**Figure 2.8:** Terminology and graphic symbols used in the decision tree of Figures 2.9 to 2.18.

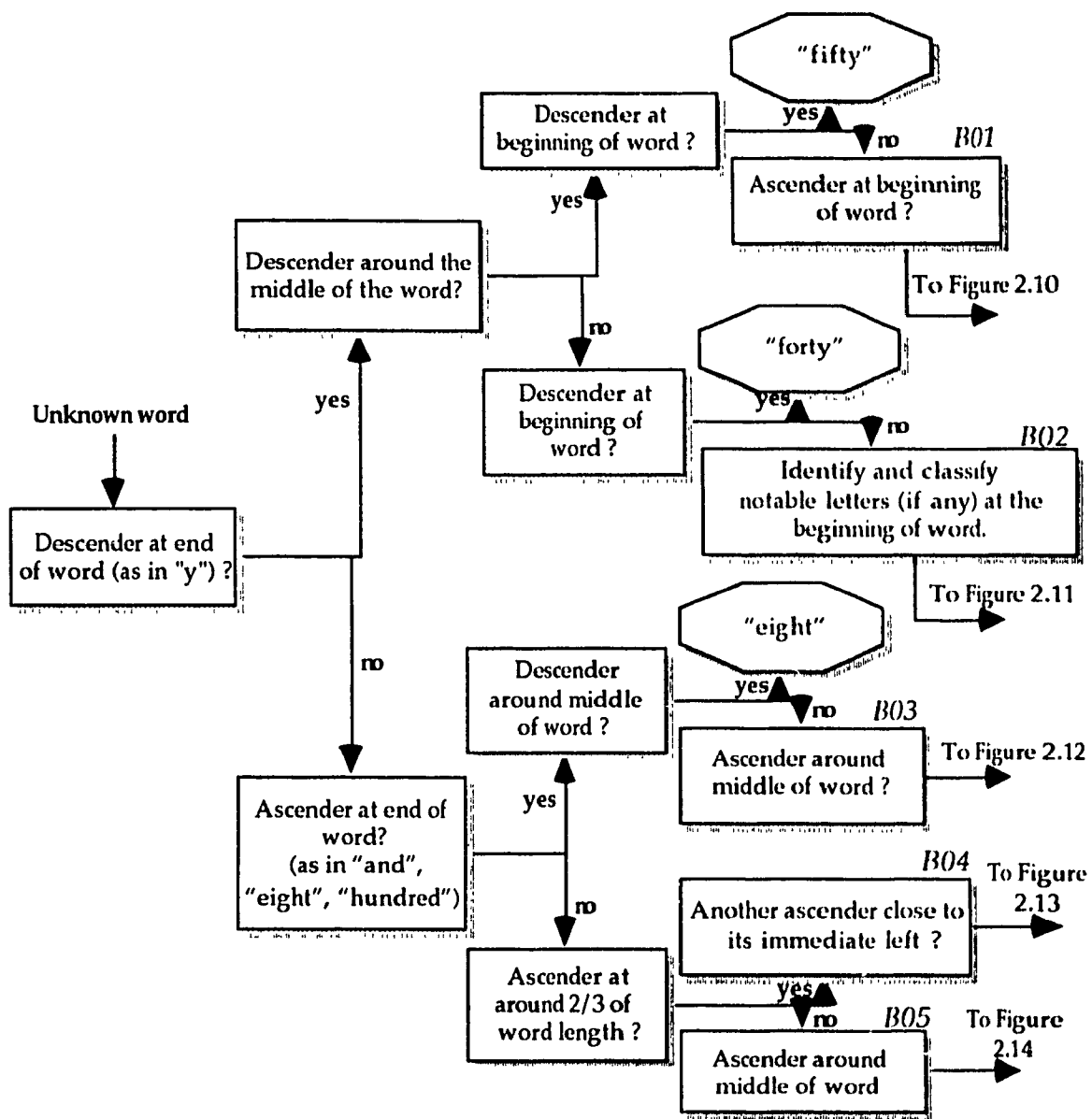


Figure 2.9: First steps in the traversal of the decision tree. Further steps beyond nodes with codes *B01* to *B05* are shown in Figures 2.10 to 2.14 respectively.

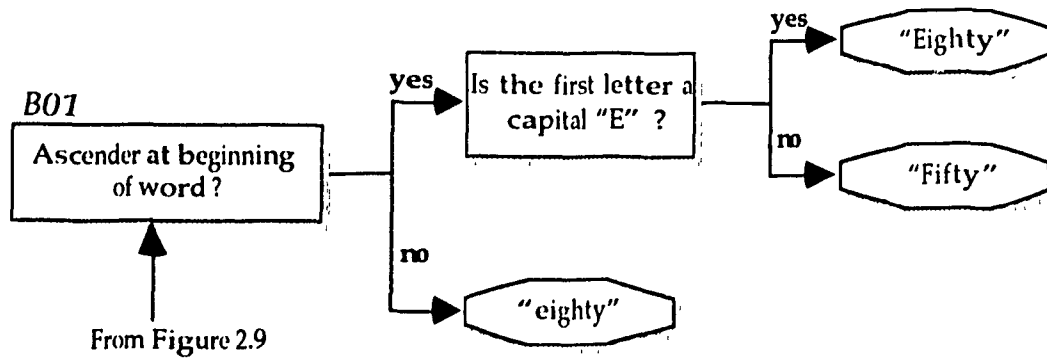
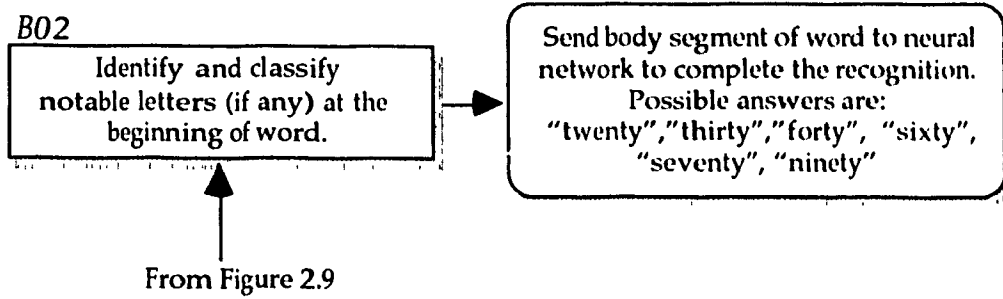


Figure 2.10: Leaves obtained from node *B01*.



**Figure 2.11:** Leaves obtained from node *B02*.

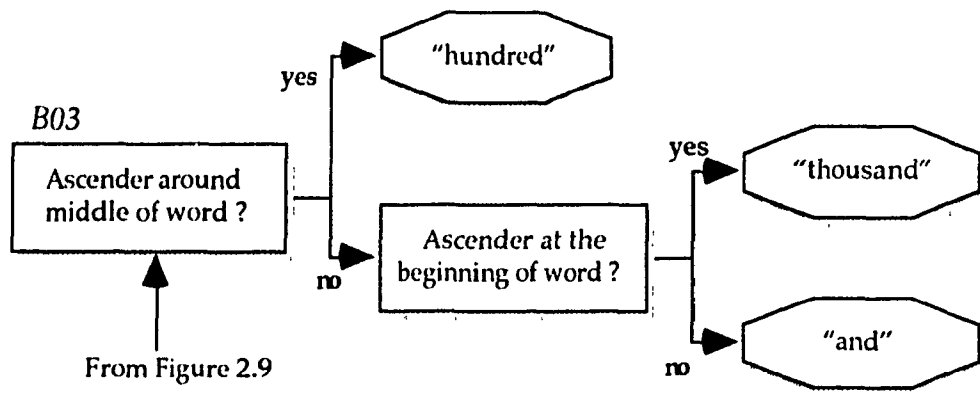


Figure 2.12: Leaves obtained from node *B03*.

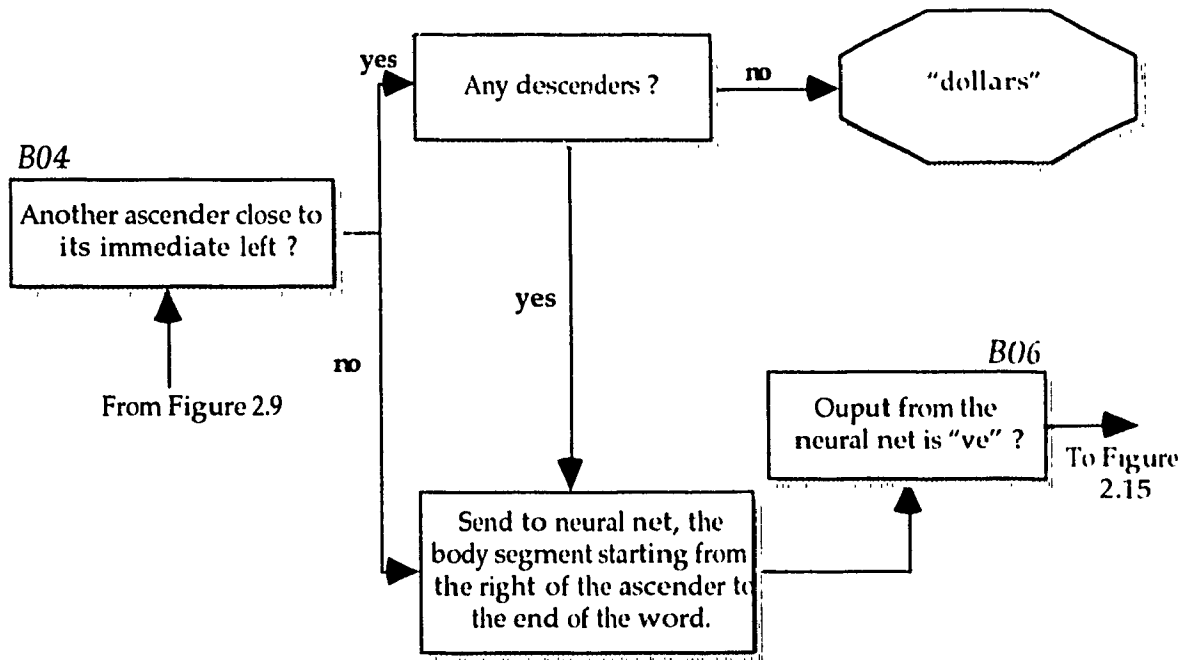
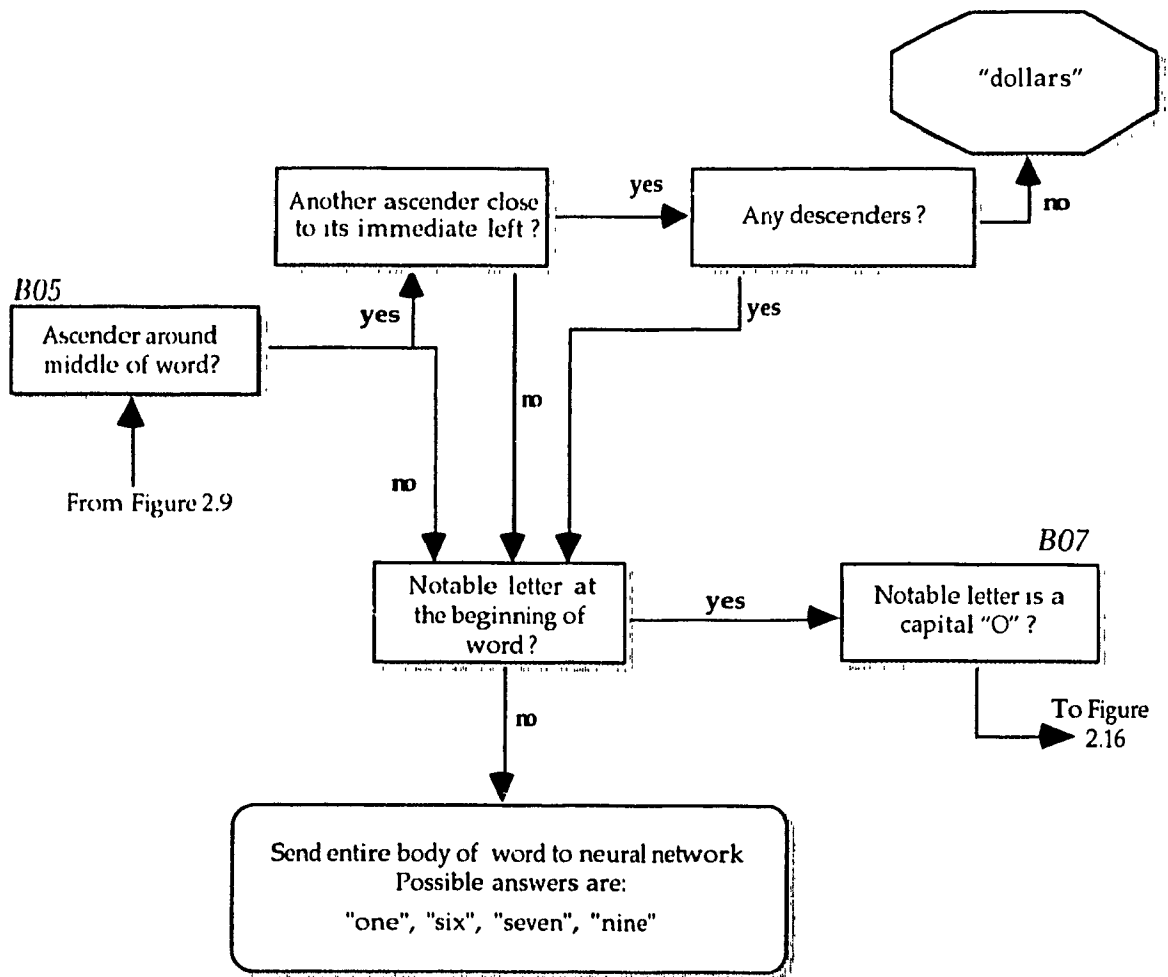


Figure 2.13: Two steps down the tree from node *B04*. Further steps beyond node *B06* are expanded in Figure 2.15.





**Figure 2.14:** Two steps down the tree from node *B05*. Further steps beyond node *B07* are expanded in Figure 2.16.

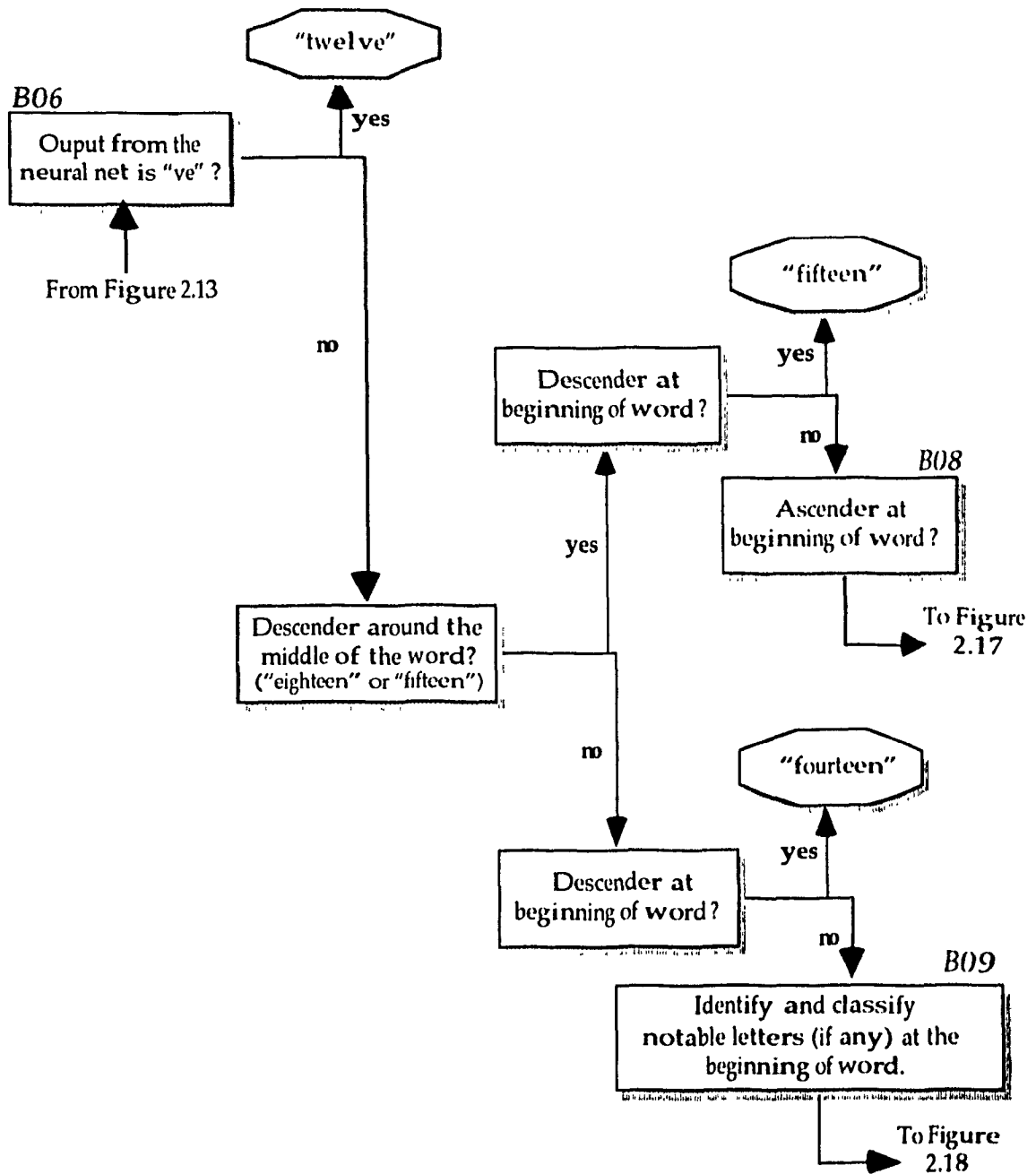


Figure 2.15: Three steps down the tree from node B06. Further steps beyond nodes B08 and B09 are expanded in Figures 2.17 and 2.18 respectively.

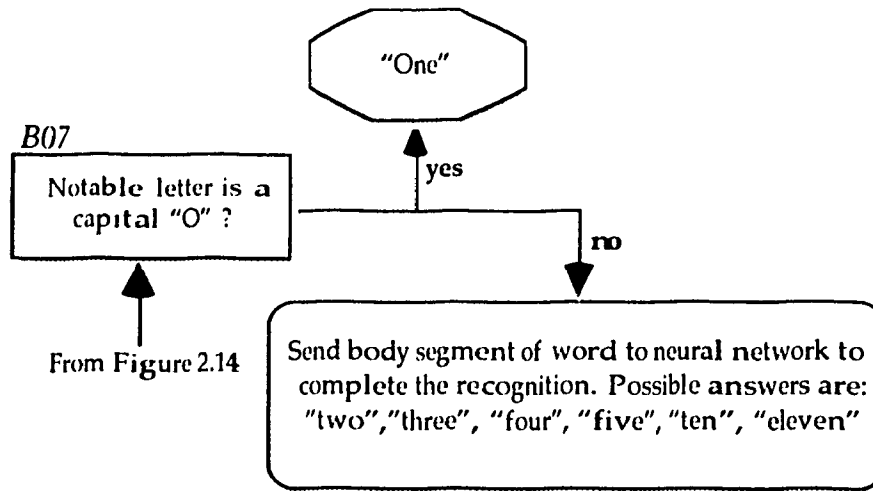


Figure 2.16: Leaves obtained from node B07.

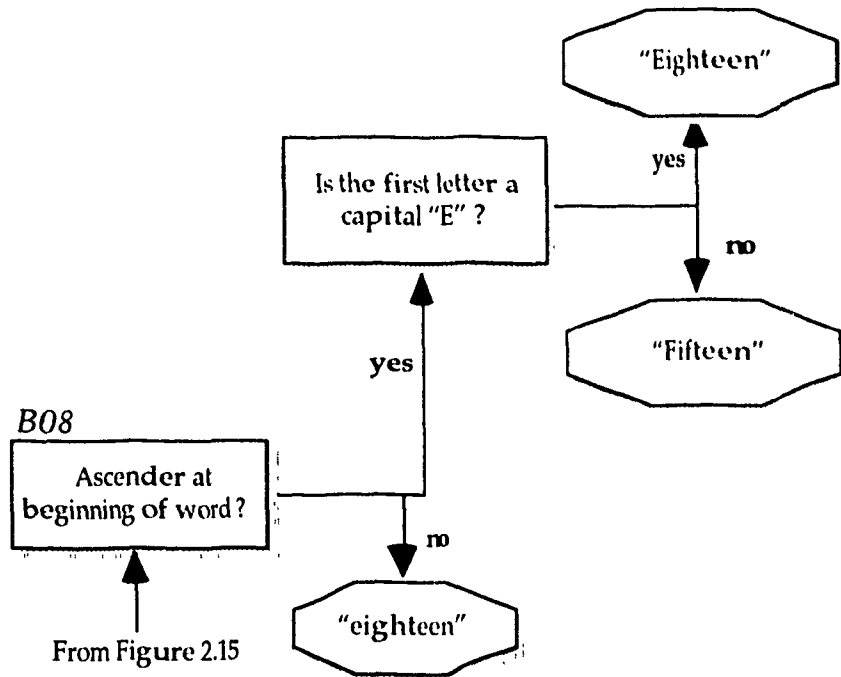
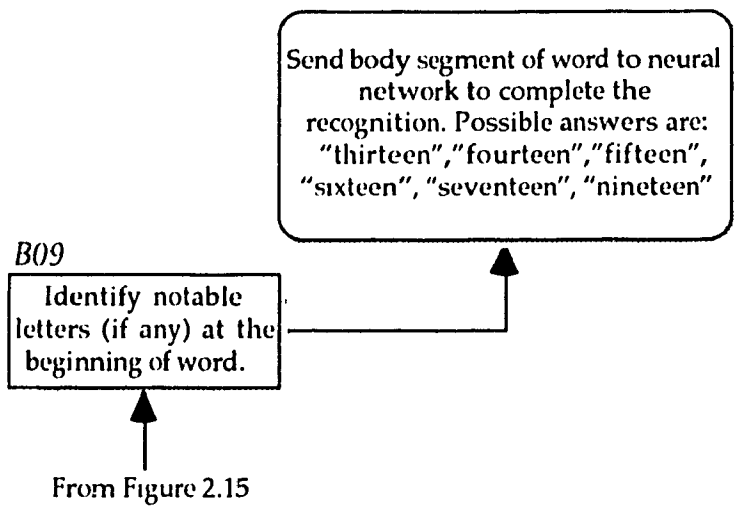


Figure 2.17: Leaves obtained from node B08.



**Figure 2.18:** Leaves obtained from node *B09*.

## Chapter 3

# The Symbolic Classifier

When an unknown word is to be classified, it is the symbolic classifier which is called first to extract features. For each word in the worded amount, the symbolic classifier looks for specific features such as ascenders, descenders, loops such as those found sometimes in letter “l”, notable letters at the beginning of words, and length/body-height ratio of words. Moreover, it performs the following tasks:

- classifies notable letters in order to have a better context for recognition;
- finds points separating notable letters from the rest of the word.
- traces the contour of certain letters in order to find their widths, heights, slopes, and positions within the word;

In the tree shown previously in Figure 2.9, we see that the first feature looked for, when an unknown word is given, is a descender at the end of the word. Following is the description of how ascenders and descenders are identified.

### 3.1 Identifying ascenders and descenders

We define two areas called the ascender and the descender windows as follows: The ascender window, is a horizontal strip of 5 adjacent rows, stretching from the beginning of the word to the end. It sits  $\left\lfloor \frac{1}{3} \times body\_height \right\rfloor$  rows above the body-ceiling. This strip will contain the segments of ascenders which will be traced for feature extraction. Similarly, the descender window is also 5 rows tall and lies at a distance of  $\left\lfloor \frac{1}{3} \times body\_height \right\rfloor$  rows below the baseline (see Figure 3.1). The

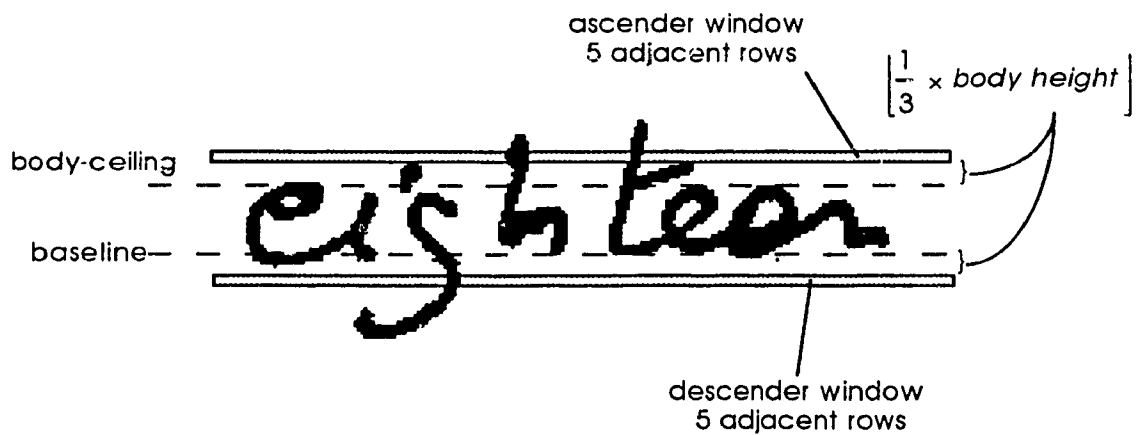


Figure 3.1: Ascender and descender windows.

reason for keeping the ascender/descender windows a certain distance above the body-ceiling and below the baseline is to avoid body segments stretching slightly above the body-ceiling or below the baseline. In Figure 3.1, for instance, the first “e” in “eighteen” stretches slightly under the baseline; by having the descender window  $\left\lfloor \frac{1}{3} \times \text{body\_height} \right\rfloor$  rows below the baseline, we avoid that stroke segment from the “e” which extends under the baseline and could be misinterpreted as a descender.

Detecting ascenders and descenders consists of scanning the image of a word from right to left inside the ascender/descender windows within a region where the feature is expected to be found. In a future section, we will see a detailed table showing how each feature is mapped onto a specific area inside the ascender/descender windows. A descender is detected in the descender window when a near-vertical stroke is found. We define a near-vertical stroke as one with a slope within the following values:

$$1 \leq \left| \frac{\delta y}{\delta x} \right|$$

where  $\delta y$  is the projection of the stroke inside the window, on the y-axis, and  $\delta x$  is the projection of the stroke inside the window, on the x-axis. In other words, the absolute value of the slope inside the descender window must be greater or equal to 1. This wide range of slopes is to accommodate inclined writing. The same applies for detecting ascenders inside the ascender window. Specific examples of ascender/descender detection will be shown in later sections.

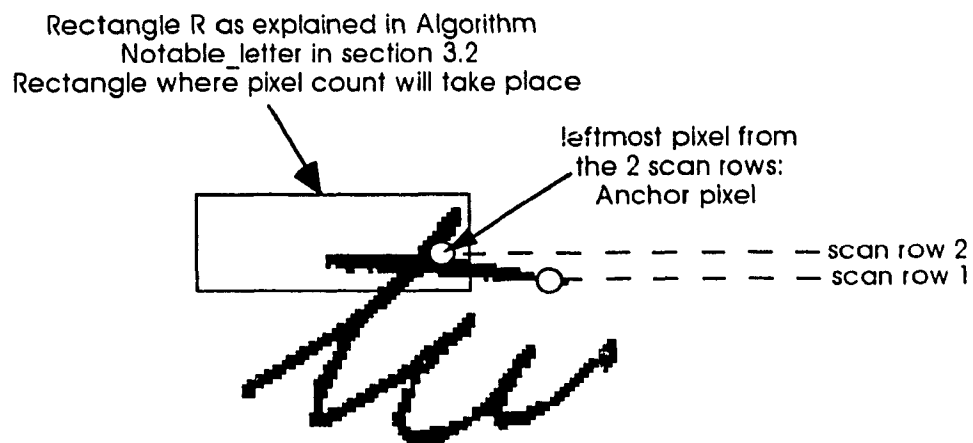


### 3.2 Identifying notable letters at the beginning of words

Identifying the presence of a notable letter is, in many cases, necessary not only for context information, but also because in the design of the neural classifier, notable letters were to be analyzed symbolically to improve the neural network's performance: Notable letters varying greatly in width and shapes make it difficult for the neural classifier to learn and classify accurately all word patterns. The identification of notable letters is used in situations such as shown in box *B02* appearing in Figures 2.9 and 2.11, and box *B09* appearing in Figures 2.15 and 2.18. Not all words, however, require looking for notable letters, their length-height ratio and ascender/descender windows contain enough information to classify them without further analysis.

In order to detect the presence of a notable letter at the beginning of a word, first, two rows (called *scan-rows*) are selected above the body of the word. The first scan-row is situated 3 rows above the ascender window and the second scan-row is 5 rows above the first scan-row (steps [1] and [2] in algorithm *Notable\_letter* below). We start scanning along the scan-rows from right to left starting at  $\lfloor 1/3 \times \text{word\_length} \rfloor$  and up to the beginning of the word. Scanning halts when each row reaches a black pixel (steps [3] and [4]). The leftmost pixel of the two found is selected and called *anchor pixel* (step [5]). In the example shown in Figure 3.2, we can see that it is the pixel found in scan-row 2 which is selected as anchor pixel, being the leftmost pixel found from the two scan-rows.

Following, a black pixel count is done inside a box *R* defined as follows: The left boundary is the middle of the blank before the word, the right boundary is set to a heuristic 15 pixels to the right of the anchor pixel. The top boundary is the top of the word's highest ascender, and the bottom boundary is the top row inside



**Figure 3.2:** Identifying the presence of notable letter "t" in "twenty".

the ascender window (step [7]). Figure 3.2 illustrates these concepts. If either the pixel count inside the box exceeds a threshold limit of 40 black pixels, or the slope of the stroke at the anchor pixel is a near-vertical stroke, then a notable letter is said to be found (step [8]).

The following pseudo code outlines the main steps involved in identifying notable letters.

*Algorithm Notable\_letter: returns TRUE when a notable letter is found, FALSE otherwise.*

- [1] *scan\_row\_1 = 3 rows above the ascender window*
- [2] *scan\_row\_2 = 5 rows above scan\_row\_1*
- [3] *P1 = First black pixel found in scan\_row\_1, starting from  $\lfloor 1/3 \times \text{word\_length} \rfloor$  and up to the beginning of the word.  
if P1 is on a letter "i"'s dot, update P1's value. {the algorithm that checks this will be given shortly below}*
- [4] *P2 = First black pixel found in scan\_row\_2, starting from  $\lfloor 1/3 \times \text{word\_length} \rfloor$  and up to the beginning of the word.  
if P2 is on a letter "i"'s dot, update P2's value. {the algorithm that checks this will be given shortly below}*
- [5] *anchor\_pixel = leftmost amongst P1 and P2.*
- [6] *last\_blank\_middle = in the last word separation (blank) found, select the middle column.*
- [7] *define the rectangle R with the following columns and rows:*
  - *leftmost column of R = last\_blank\_middle*
  - *rightmost column of R = column of anchor\_pixel + 15*  
*The "+ 15" is there to extend the right boundary of the box a little further than the anchor point in order to make sure most of the notable letter's black pixels are enclosed inside this box R.*
  - *top row of R = top row in text image*
  - *bottom row of R = top row in ascender window**see Figure 3.2*
- [8] *if (black pixel count inside R is greater than 40) or (the slope of the stroke at the anchor pixel is a near-vertical stroke)*

```

        return TRUE
    else
        return FALSE
endAlgorithm

```

As shown in steps [3] and [4], special precaution is taken for handling the dots over an "i" which could be on the scan path, as exemplified in Figure 3.3; therefore, when  $P1$  is found on scan-row1, we trace around the black pixel cluster starting from  $P1$  (see Figure 3.4 and step [1] in algorithm *P\_is\_on\_a\_dot* shown below); if the perimeter found for the black pixel cluster is under 60 pixels, we conclude it is a dot and hence we skip the black pixel cluster and continue scanning left for the next black pixel  $P1'$  along scan-row1 (as shown in Figure 3.5 and steps [2], [3], and [4]).  $P1$  is then rejected and  $P1'$  will be renamed  $P1$  (step [5]). These steps are also applied on scan-row2 when  $P2$  is found. The following algorithm outlines the steps discussed in this paragraph:

*Algorithm P\_is\_on\_a\_dot; (checks if pixel P is on a letter "i"'s dot. If it is, the algorithm skips the dot to find the right P)*

```

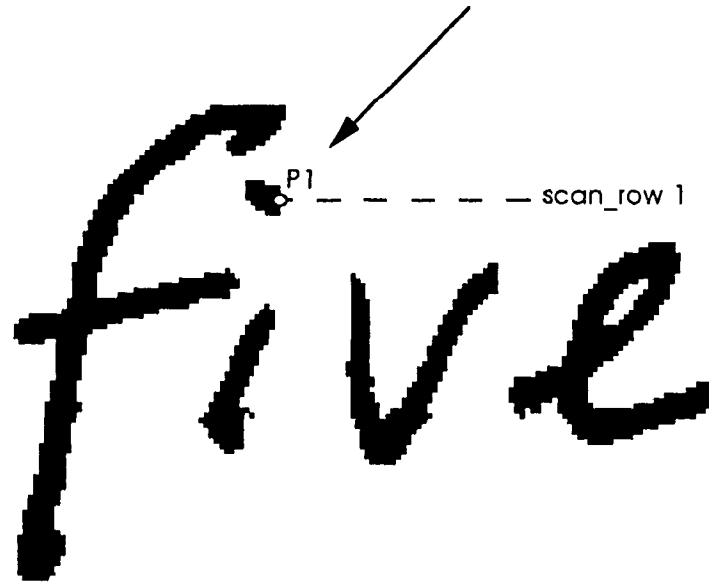
[1] Trace around the black pixel region where P was found. The trace starts from P
    and stops when the trace has come full circle back to P.
[2] if the number of pixels traced in step [1] is  $\leq 60$  pixels
[3]     Scan to the left, from P until a white pixel is reached.
[4]     Continue scanning to the left until a black pixel is reached
        Call this black pixel P'
[5]     P = P'
[6] return P
endAlgorithm

```

### 3.3 Finding the separation point between the notable letter and the remaining part of the word to its right

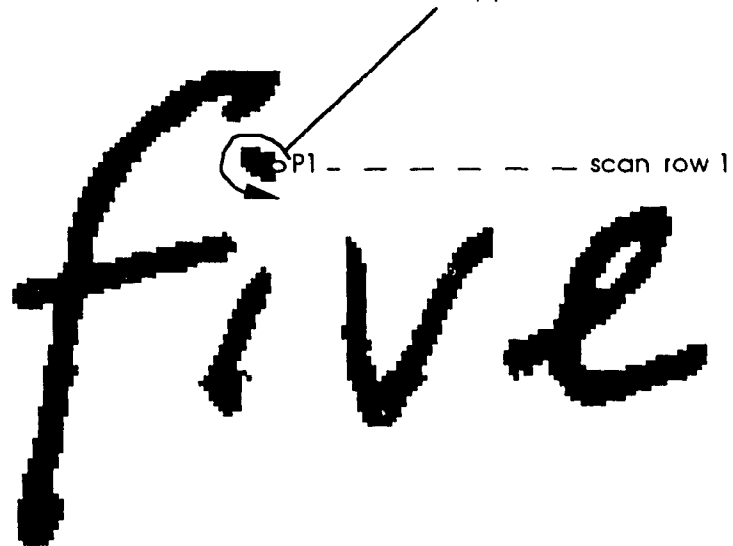
As we mentioned before, the neural classifier will recognize only word segments with letters having neither ascenders nor descenders. Therefore when a notable

scanning along scan\_row 1, one sees the letter "i"'s dot



**Figure 3.3:** Coming across a dot when looking for a notable letter (see section 3.2).

Tracing around the black pixel cluster starting from P1, gives a perimeter of less than 60 pixels. Therefore it is a dot and must be skipped.



**Figure 3.4:** Tracing around the edge of the black pixel cluster to find its perimeter (in pixels traced).

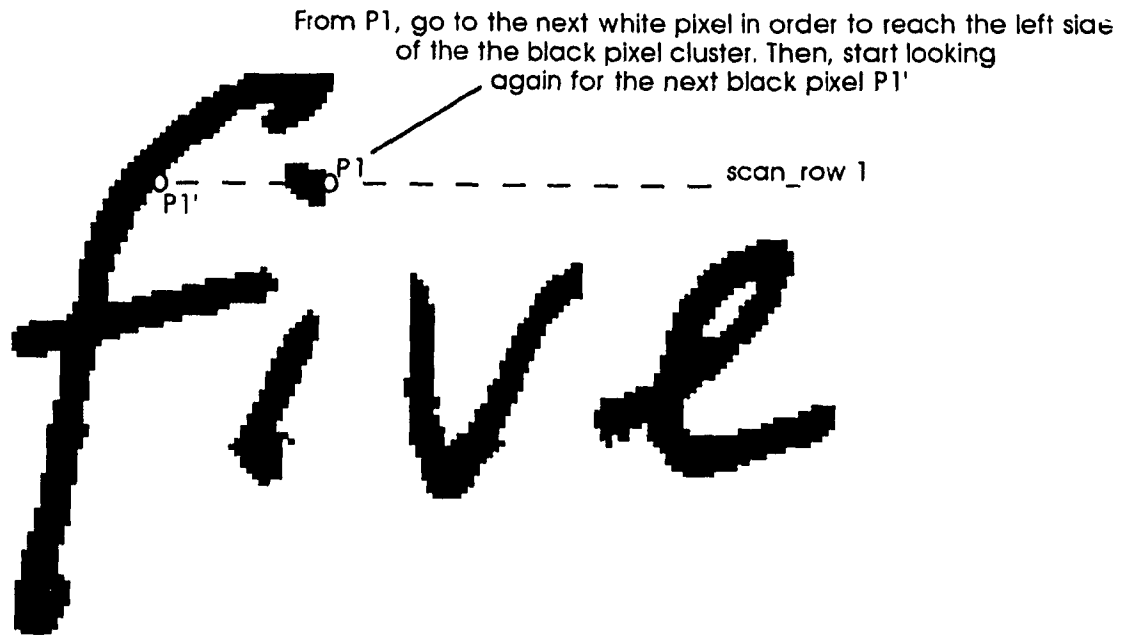
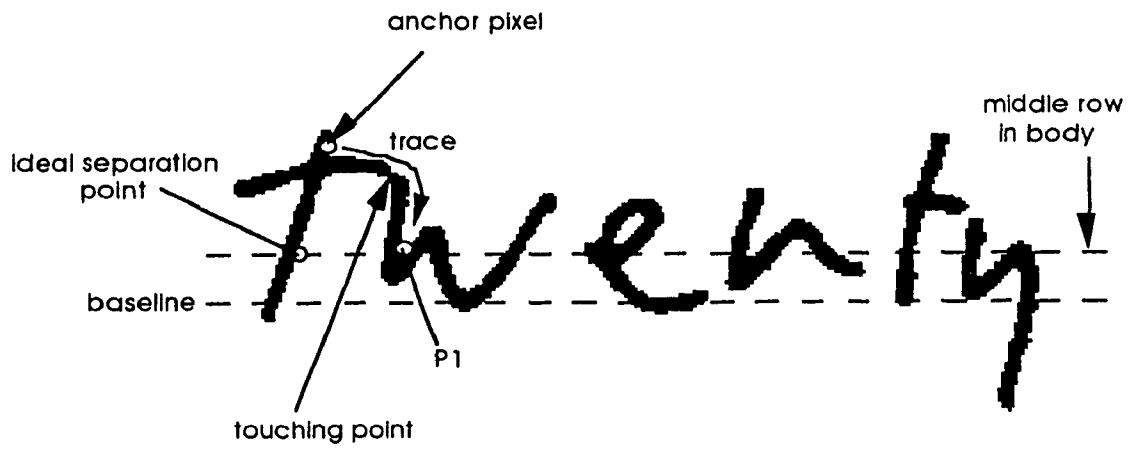


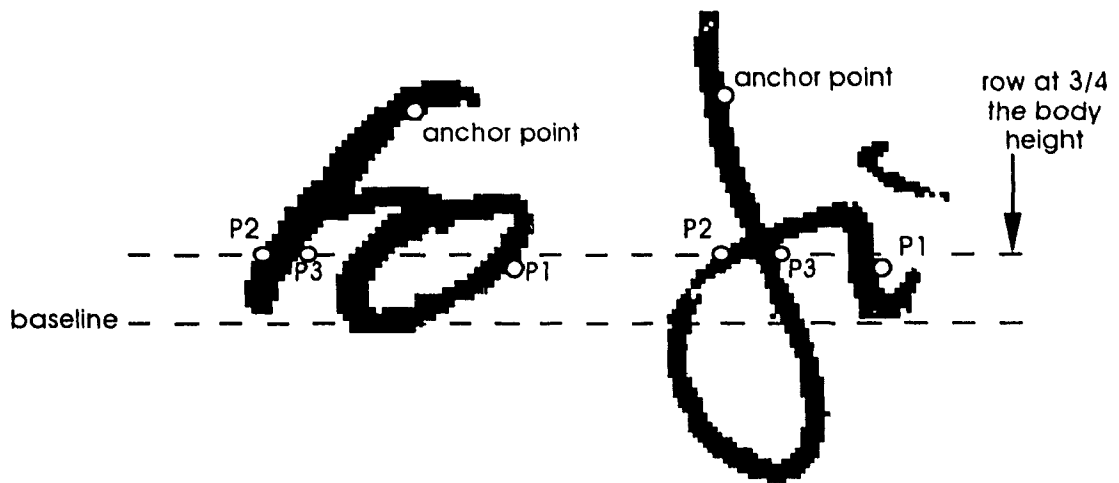
Figure 3.5: Skipping the dot and resuming the search for the notable letter.

letter is found at the beginning of the word, it is necessary to skip it by finding an accurate separation point between the notable letter and the remaining section of the word to its right. This process is done before invoking the neural network, in situations such as shown in the clustered-leaf nodes appearing in Figures 2.11 and 2.16. The algorithm starts by scanning down the notable letter from the anchor pixel found, until the middle row inside the body is reached. Let us call that point  $P1$  (step [1] in algorithm *Find\_Separation\_Pixel* below). In an ideal situation where letters never touched,  $P1$  would always be selected as the separation point that we are trying to find. In reality, however,  $P1$  could either be on the left edge of the notable letter, or if the letters were touching, it could be on another edge in the letter next to it (see Figure 3.6). We therefore need to find other estimates for the separation point. Starting from the anchor pixel once more, we now trace the letter upwards, stopping when we have come down the other side of the letter and reached the row at  $\lfloor 3/4 \times \text{body\_height} \rfloor$  up from the baseline. Let us call that point  $P2$  (step [2]). As we can see from the examples in Figure 3.7, the ideal separation point is between  $P1$  and  $P2$ . Hence, from  $P2$  we resume the trace downwards, stopping once more when we have reached the same row where  $P2$  was found. Let us call that point  $P3$  (step [3]). If  $P3$  is between  $P1$  and  $P2$ , and  $P1$  isn't along the tracing path from  $P2$  to  $P3$ , the optimal separation point is said to be found. Otherwise, we resume the trace until we find a  $Pn$  which is either between  $P1$  and  $P2$  or to the right of  $P1$  (see Figure 3.7). When a  $Pn$  is found, if it is between  $P1$  and  $P2$ , and  $P1$  isn't along the tracing path from  $Pn-1$  to  $Pn$ , then  $Pn$  is selected as the optimal separation point (step [4]). Otherwise, if  $P1$  was found along the path from  $Pn-1$  to  $Pn$ , then  $P1$  is selected as the optimal separation point (step [5]) (Figure 3.8 (a)). If  $Pn$  is found to the right

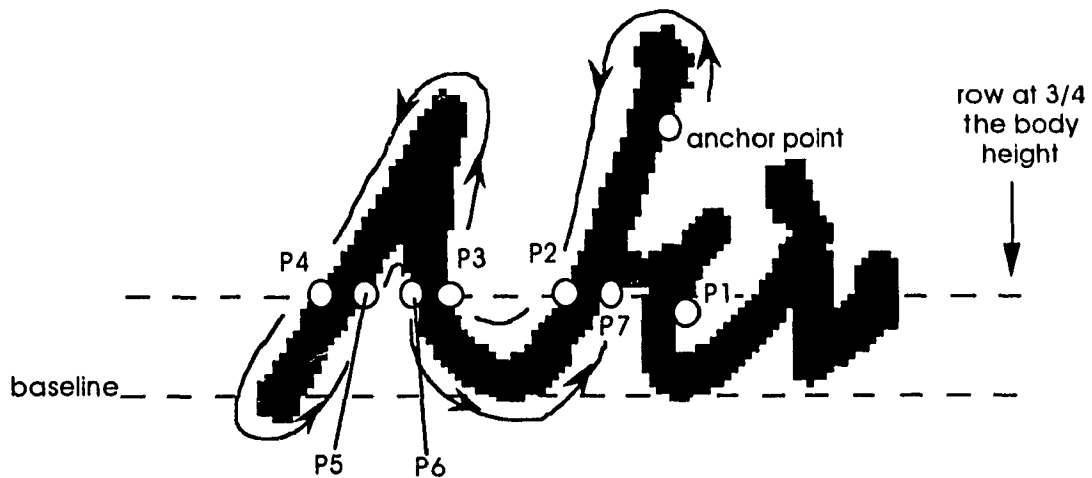




**Figure 3.6:** Example of inaccurate separation point between a notable letter and the letter following it.

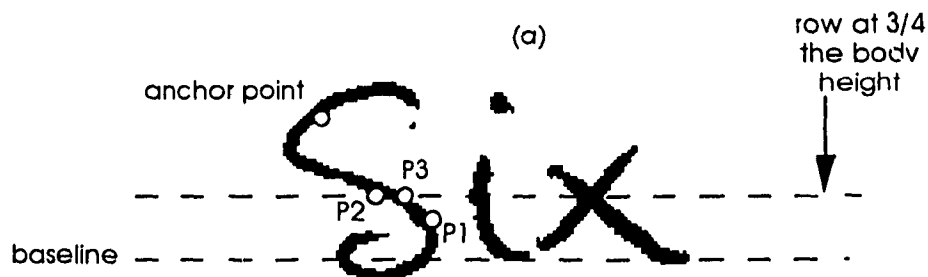


In these two cases ("fo" as in "four", and "fi" as in "five"), P3 is the last P<sub>n</sub> since it was found between P1 and P2. P3 is returned as the separation point

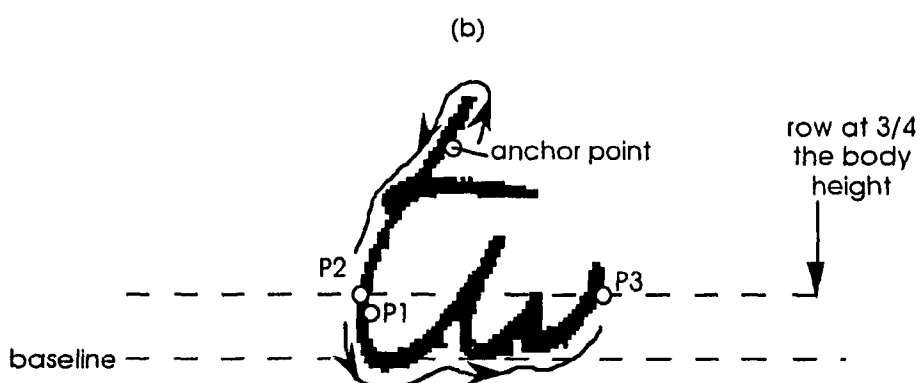


In this section of the word "Nine", P7 is the last P<sub>n</sub> since it was found between P1 and P2. P7 is returned as the separation point

**Figure 3.7:** Finding estimates for the separation point between the notable letter and the remaining of the word: Algorithm *Find\_Separation\_Pixel* in section 3.2 is applied to a capital "F", a lowercase "f", and a capital "N".



P3 was discarded since P1 was found along the path from P2 to P3. Therefore P1 is selected as separation point



P3 was found to the right of P1, therefore P1 is selected as separation point

**Figure 3.8:** Finding estimates for the separation point between the notable letter and the remaining of the word: Algorithm *Find\_Separation\_Pixel* in section 3.2 is applied to a capital "S", and a lowercase "t".

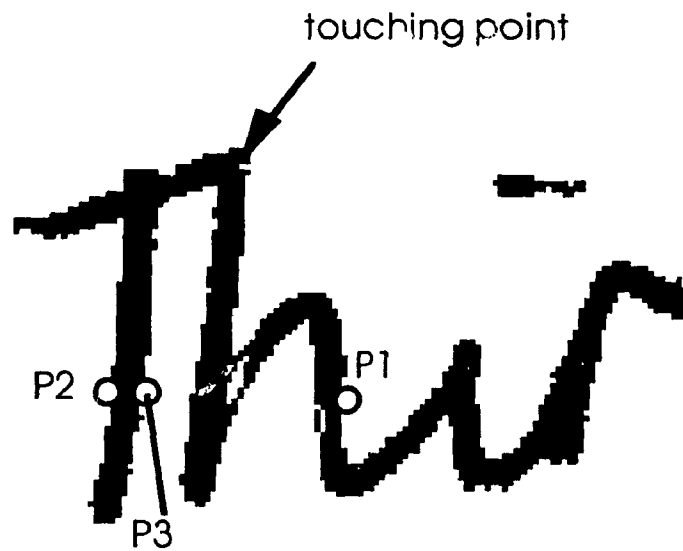
of  $P1$ , this means there isn't a better estimate for the separation point between  $P1$  and  $P2$ , therefore  $P1$  is selected as the optimal separation point (step [5]) (see Figure 3.8 (b)).

The following pseudo code outlines the main steps involved in finding the separation point between the notable letter and the remaining section of the word to its right:

**Algorithm Find\_Separation\_Pixel;**

```
[1] From the anchor pixel, start tracing downwards. Stop when we reach row =  
middle row in the body. Call this pixel  $P1$   
[2] From the anchor pixel, start tracing upwards. Stop when we reach row =  
 $\lfloor 3/4 \times \text{body\_height} \rfloor$  above the baseline. Call this pixel  $P2$ .  
[3] Resume trace stopped in step [2]. Stop when we reach once more row =  
 $\lfloor 3/4 \times \text{body\_height} \rfloor$  above the baseline. Call this pixel  $P3$ .  
   $n = 3$   
  loop  
[4]   if ( $Pn$  is between  $P1$  and  $P2$ ) and ( $P1$  isn't on the path from  $P_{n-1}$  to  $Pn$ )  
      return  $Pn$   
[5]   if ( $Pn$  is to the right of  $P1$ ) or ( $P1$  was found on the path from  $P_{n-1}$  to  $Pn$ )  
      return  $P1$   
[6]    $n = n + 1$   
[7]   Resume trace stopped at  $P_{n-1}$ . Stop when we reach once more row =  
       $\lfloor 3/4 \times \text{body\_height} \rfloor$  above the baseline. Call this pixel  $Pn$ .  
  endloop  
endAlgorithm
```

Although this technique works well in most cases of notable letters, there is a problem when the first two letters are "th", the "t" touching the "h" at the top and disjoint from it at the bottom (see Figure 3.9). In order to solve this problem, we shall treat all words starting with "th" differently. As we shall see shortly, notable letters are classified into different categories, therefore identifying the



P3, being between P1 and P2, is selected as optimal separation point

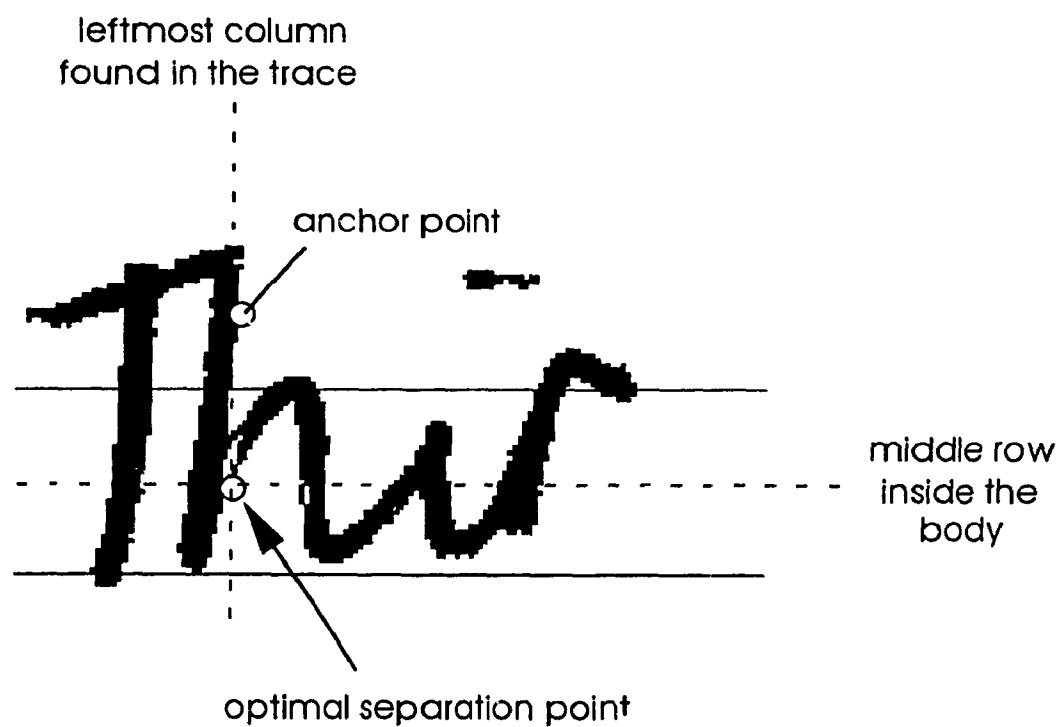
**Figure 3.9:** Problems finding the separation point between the notable letter and the remaining of the word in words starting with "th".

words starting with "th" prior to finding the separation point will not be a problem. The alternative algorithm to find the optimal separation point consists in scanning down the notable letter from the anchor pixel, until the body-ceiling is reached. On the way down, the leftmost column reached along the path is retained as the separation point is defined to be at the intersection of that leftmost column found, and the middle row inside the body (see Figure 3.10).

### 3.4 Categorizing notable letters

Owing to the wide variety of capital letter writing styles, notable letters are separated into 9 categories, each holding 1 or 2 notable letters. Notable letters are categorized immediately after being found (section 3.2). Therefore the following algorithms are invoked in situations as shown in box *B02* appearing in Figures 2.9 and 2.11, and box *B09* appearing in Figures 2.15 and 2.18. Each category described below will start by stating which types of words fall into the category, and it will then explain which features the classifier looks for in an unknown word in order to classify it under this category.

- *Notable category 1*: Word starts with a lowercase "f" as in "forty": This is a special case where the first letter of the word has an ascender and a descender (see Figure 3.11). It is actually the only letter in this vocabulary that has this particular feature; therefore, recognizing it will consist of looking for a descender at the beginning of the word.
- *Notable category 2*: Word starts with "S" as in "Sixty": To recognize an "S", the first algorithm traces down the left side of the notable letter, starting from a row which is 30% down from the top of the letter, and finishing at 65% down from the top. Within those 35%, the leftmost and rightmost black pixels along the path are kept in memory as these roughly represent the locations where the



**Figure 3.10:** Separation point between notable letter and the remaining part of the word for words starting with "th".

lowercase "f"s have ascenders and descenders

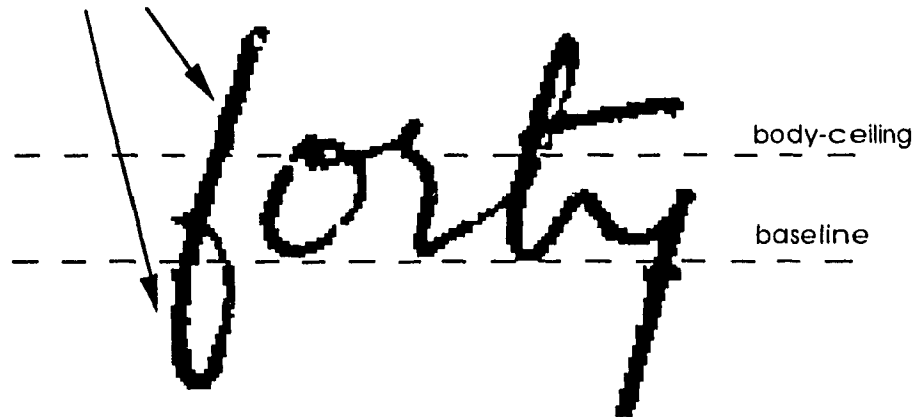


Figure 3.11: Example of word with lowercase "f".



horizontal tracing direction switches from left to right or vice-versa (see Figure 3.12) (steps [1] to [4] in algorithm *Notable\_Category\_2* below). If the slope of the straight line going through these two turning points satisfies:  $-1 \leq \text{slope} < 0$ , then the slope is considered to be sufficiently oblique to characterize an "S" curvature (steps [5] and [6]). If this condition isn't satisfied, the notable letter is rejected from this category; otherwise, a second algorithm will be executed to further confirm the answer from the first one. Before looking at the second algorithm we will look at the following pseudo-code outlining the first algorithm used to recognize a capital "S".

*Algorithm (1) Notable\_Category\_2;*

- [1] *notable\_letter\_height = height in rows of the notable letter*
- [2] *start\_row = [0.3 × notable\_letter\_height] rows down from the top of the notable letter.*
- [3] *end\_row = [0.65 × notable\_letter\_height] rows down from the top of the notable letter.*
- [4] *Trace down the left edge of the notable letter from start\_row down to end\_row, storing in leftmost\_pix the leftmost pixel found during the trace, and in rightmost\_pix the rightmost pixel found during the trace.*
- [5] *L = line going through leftmost\_pix and rightmost\_pix*
- [6] *if (slope (L) < 0) and (slope (L) ≥ -1) then*  
           *return TRUE*  
       *else*  
           *return FALSE*

*endAlgorithm*

The second algorithm looks at the left side of the top 30% portion of the letter. It traces it upwards while verifying the slope of the stroke at every black pixel traced. If the slope is positive and less than or equal to a near-horizontal threshold value of 1, then we have probably found the parabolic top section of an "S" (see Figure 3.13). It has been noted that most capital "S" have a sufficiently

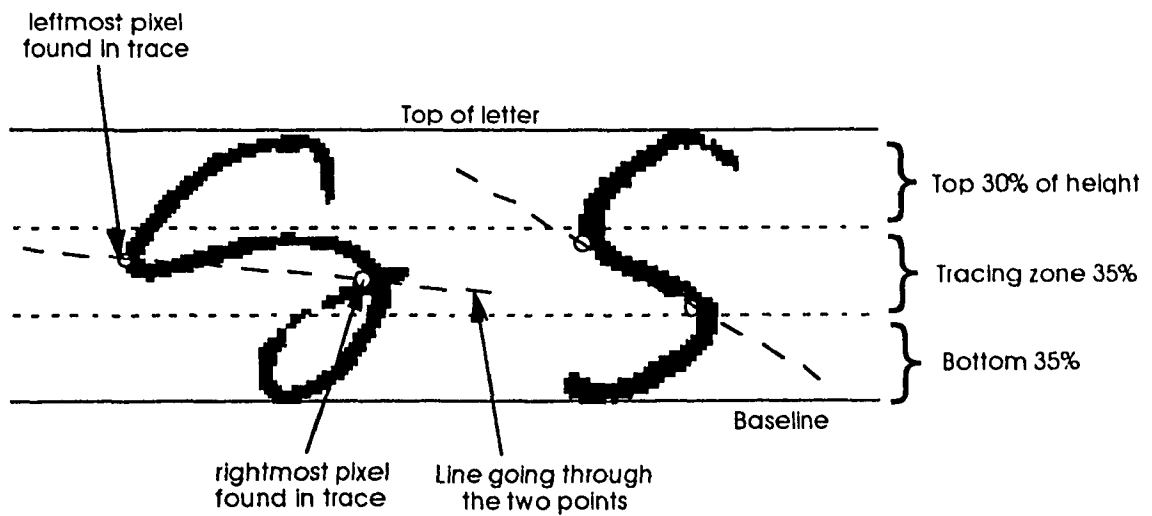
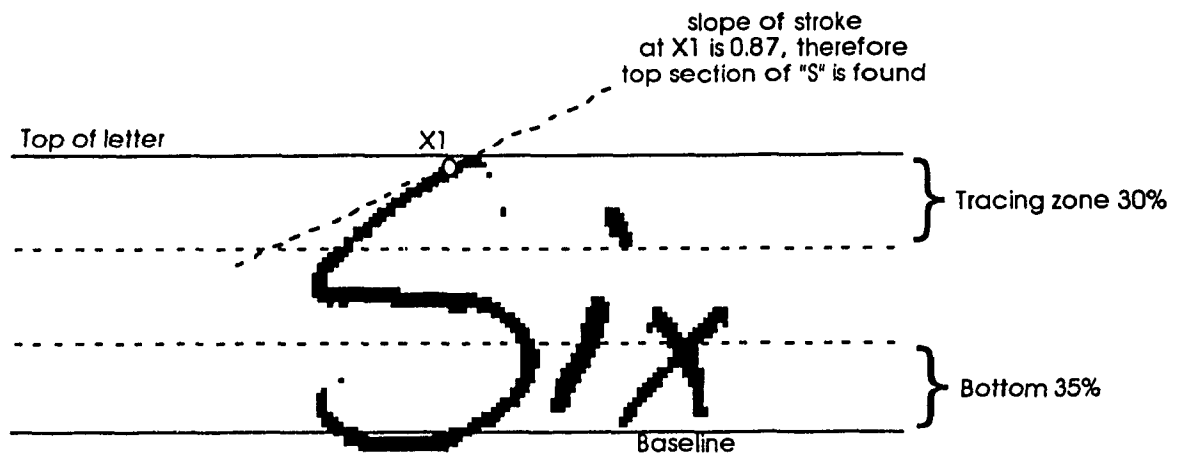


Figure 3.12: Algorithm (1) *Notable\_Category\_2*, described in section 3.4, applied to recognize an "S".



**Figure 3.13:** Algorithm (2) *Notable\_Category\_2*, described in section 3.4, applied to recognize an "S": Identifying the parabolic top section of an "S".

large horizontal separation between the two turning points to satisfy the slope thresholds defined. There are unusual cases, however, of disproportionned "S" where the height is much larger than its width, and consequently the turning points will be horizontally much closer to each other which will generate a near-vertical slope between them, leading to the failure of the first algorithm (see Figure 3.14). The following pseudo code outlines the main idea behind this second algorithm:

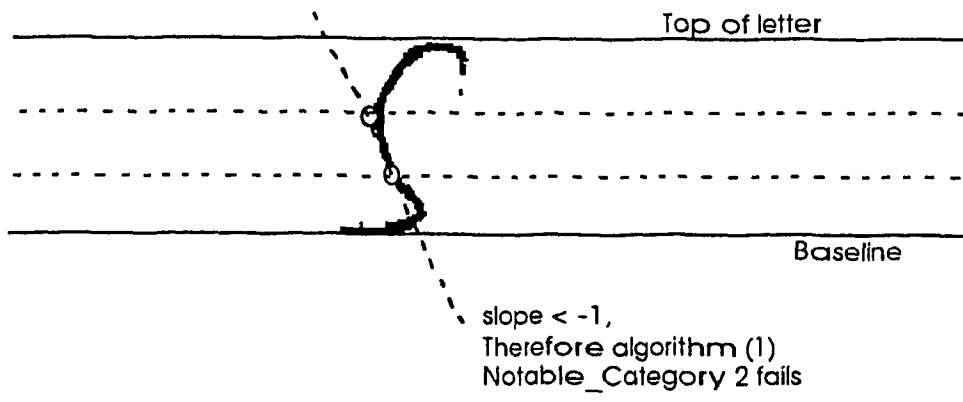
*Algorithm (2) Notable\_Category\_2;*

```

[1]  notable_letter_height = height in rows of the notable letter
[2]  start_row =  $\lfloor 0.3 \times \text{notable\_letter\_height} \rfloor$  rows down from the top of the notable
      letter.
[3]  end_row = top of the notable letter.
[4]  current_row = start_row
[5]  Trace the left edge of notable letter from start_row to end_row, checking the slope
      of the stroke at every pixel traced:
      while current_row  $\neq$  end_row
          currentPixel = black pixel on left edge of notable letter, at current_row.
          T = tangent to left edge of notable letter, at currentPixel
          if slope(T) > 0 and slope(T)  $\leq$  1 then
              return TRUE
          current_row = row immediately above
      endwhile
      return FALSE
endAlgorithm

```

- *Notable category 3:* Word starts with "E" as in "eleven": For this category, the first occurrence of letter "E" is actually a lowercase "e", and it is through the location of the "l" within the word that the presence of this "e" is deduced. Formally speaking, we scan from left to right the leftmost 1/4 of the word, at a row 5 rows above the ascender window. Scanning stops when a black pixel is

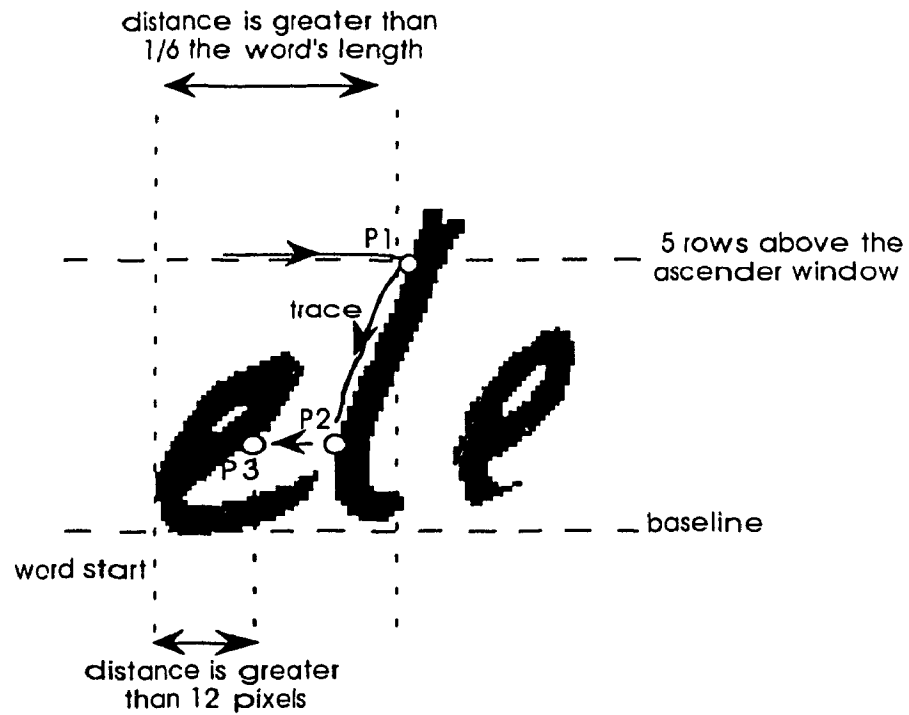


**Figure 3.14:** Uncommon case of disproportionned "S" where algorithm *Notable\_Category\_2* shown in section 3.4 fails to recognize the notable letter as an "S".

found. If the unknown word is actually “eleven”, this pixel is either the left edge of a capital “E”, or the “l” if the “E” is lowercased (steps [1] and [2] in algorithm *Notable\_Category\_3* below). To determine that the first letter is a lowercased “e”, the following steps are done: The pixel found during the scan must be at a distance greater than or equal to  $\lfloor 1/6 \times \text{word\_length} \rfloor$ , from the beginning of the word (step [3]). This minimum distance would correspond to the space occupied by the lowercase “e” at the beginning of the word. If this condition is satisfied, we trace down the left edge of the “l” until we reach the middle row inside the body (step [4]). From there, we move west until we reach the next black pixel which should be part of the right side of the lowercase “e” (step [5]). If no black pixel is found, then the “l” didn’t have any letter to its left which is impossible, therefore the letter is rejected from this notable category (step [6]). If a black pixel is found, however, we then verify that its distance from the beginning of the word is greater than the maximum stroke width set at 12 pixels (steps [7] and [8]). This minimum distance ensures the presence of a letter between the black pixel found and the beginning the word. Figure 3.15 illustrates the overall algorithm, and the following pseudo-code is provided to outline the main steps in classifying a letter into notable category 3:

**Algorithm *Notable\_Category\_3*:**

- [1] *scan\_row = 5 rows above the ascender window*
- [2] *scan along scan\_row from the beginning of the word to  $\lfloor 1/4 \times \text{word\_length} \rfloor$ . Stop when a black pixel is found. Call this pixel P1 (Figure 3.15).*
- [3] *If distance between P1 and beginning of the word is  $< \lfloor 1/6 \times \text{word\_length} \rfloor$ , return FALSE*
- [4] *Trace downwards from P1, the left edge of the notable letter. Stop when the middle row inside body is reached. Call this pixel P2 (Figure 3.15).*
- [5] *From P2, stay on the same row and scan left until a black pixel is found or until beginning of word is reached.*



**Figure 3.15:** Identifying the first letter in “eleven” as lowercase “e”. Refer to notable category 3 in section 3.4.

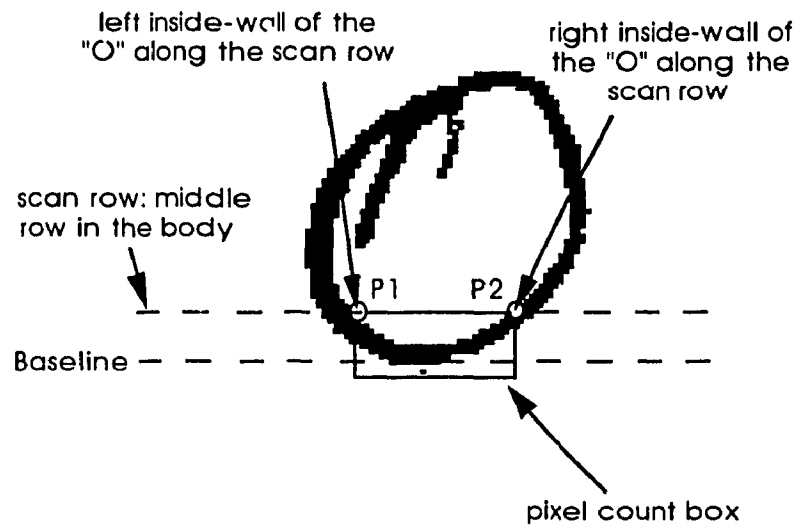
```

[6]   if no black pixel was found during [5]
        return FALSE
      else
[7]   Call this black pixel P3 (Figure 3.15).
[8]   if distance between beginning of word and P3 is > 12 columns
        return TRUE
      else
        return FALSE
endAlgorithm

```

- *Notable category 4: Word starts with a capital "O".* The first feature that would come to mind when trying to identify a capital "O" would be to look for a loop, however, expecting a closed loop in every capital "O" is unrealistic. Therefore, the first feature looked for is a high concentration of black pixels at the bottom of the letter (see Figure 3.16), corresponding to the lower portion of an "O" sitting on top of the baseline. So the approach is to scan the letter from left to right at the middle row inside the body. Scanning stops when a black pixel is found, indicating the presence of the left edge of the letter. Then, we continue scanning through that left edge until we reach a white pixel. At this point, we would be on the left side of the inside loop of the presumed "O". We must now locate the right side of the inside loop by resuming the scan and stopping when the next black pixel is reached (see Figure 3.16). A black pixel count is done in a box sitting 3 rows below the baseline, its left boundary at the column where the left side of the inside loop was found, its right boundary at the right side of the inside loop, and its top boundary at the middle row inside the body (see Figure 3.16). If the pixel count is greater than the heuristic threshold value of 70 pixels, then the feature is defined to be present. In general, other capital letters such as "T" in "Ten" or "F" in "Five", do not have this feature since most of the time, they are detached from the rest of the word, however, a lowercase "t" as in "ten",





**Figure 3.16:** First feature to be found in a capital "O" (notable category 4).

and lowercase "th" as in "three", are also considered to be notable letters and will sometimes have that feature since the "t" can be connected from its lower curve, to the next letter. Therefore we must look for another feature present in a capital "O" in order to narrow down the choices to only one. The following algorithm outlines the steps discussed in this paragraph:

*Algorithm (1) Notable\_Category\_4;*

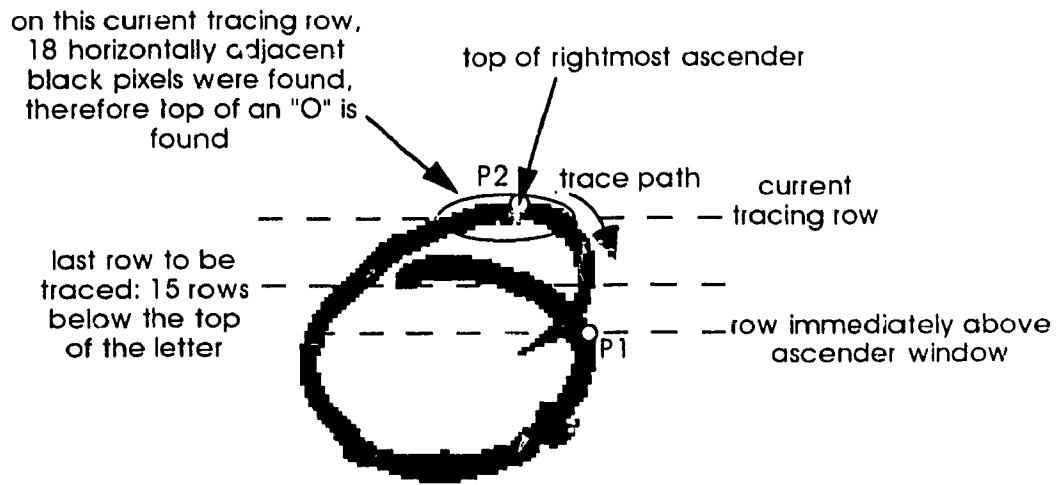
- [1] *scan\_row = middle row in body*
  - [2] *scan along scan\_row from the beginning of the word to  $\lfloor 1/2 \times \text{word\_length} \rfloor$ .  
Stop when a black pixel is found.*
  - [3] *Resume scanning until a white pixel is found. Call this point P1 (see Figure 3.16)*
  - [4] *From P1, resume scanning until a black pixel is found. Call this point P2 (see Figure 3.16)*
  - [5] *Define the rectangle R with the following columns and rows:*
    - *top-left corner = P1*
    - *bottom-right corner = (column,row) = (column of P2, 3 rows below the baseline)*
  - [6] *if black pixel count inside R is > 70*
    - return TRUE*
    - else*
      - return FALSE*
- endAlgorithm*

The second feature requires that the optimal separation point (Figure 3.7 and 3.8) must be at least 20 column to the right of the beginning of the word. If this condition isn't satisfied, the letter is very narrow in the top section of the body, therefore it is more likely to be a "t" as in "ten" than a "O". If the condition is satisfied, however, a third feature must differentiate the "O" from a "th" who's optimal separation point is along the right edge of the "h" and therefore would not be within the 20 column boundary from the beginning of the word.

Before going into details about how we find this third feature, it is important to know that when the notable letter classifier is invoked, prior analysis is done to detect the presence of any other ascender aside from the notable letter's. If this prior analysis finds an ascender, the word cannot be "One" since the only ascender in this word is in letter "O". Having said this, we therefore know that there are no other ascenders in the word. The first requirement to identify this third feature is to place a starting scan-point at the top of the rightmost ascender. To do this we scan the word from right to left at the row immediately above the ascender window. Starting from column at  $\lfloor 0.6 \times \text{word\_length} \rfloor$ , we stop scanning when a black pixel is found, indicating the presence of the left edge of the rightmost ascender (steps [1] and [2] from algorithm *Notable\_Category\_4* below). From there, we trace the ascender upwards until the tracing direction starts going downwards. The topmost row reached during the trace is defined to be the top of the rightmost ascender in the word (step [3]). Now, finding the third feature in a "O" consists of tracing down from the top of the rightmost ascender, down 15 rows along the left edge. At each row, if the left edge counts more than 14 horizontally adjacent black pixels, the third feature is found. This feature represents the characteristic top curve of an "O" (see Figure 3.17, and steps [4], [5] and [6]). The letter is then classified as an "O". The following pseudo-code outlines the steps discussed above.

**Algorithm (2) *Notable\_Category\_4*;**

- [1] *scan\_row = row immediately above the ascender window*
- [2] *scan along scan\_row from:  $\lfloor 0.6 \times \text{word\_length} \rfloor$  to the beginning of the word. Stop when a black pixel is found. Call this pixel P1 (see Figure 3.17)*
- [3] *From P1, trace upwards the left edge of the notable letter until top of letter is reached. Call this point P2 (see Figure 3.17)*
- [4] *From P2 trace back down the same path:*
- [5] *current\_row = P1's row position.*



**Figure 3.17:** Third feature to be found in a capital "O" (notable category 4).

```

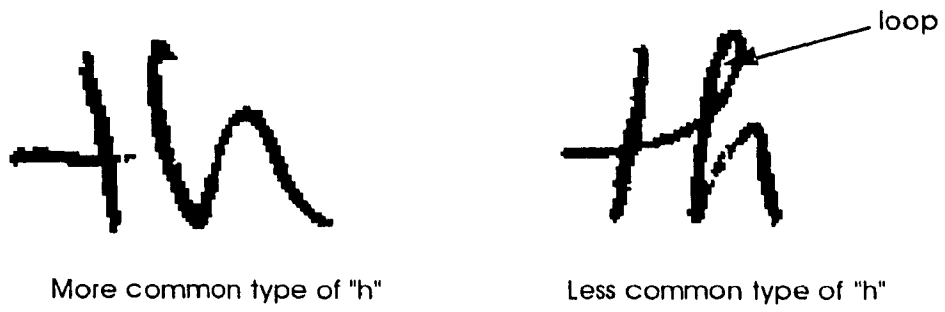
[6]  while (current_row ≠ 15 rows down from P1)
      current_pixel = black pixel currently being traced on the left edge of the
      notable letter
      if current_pixel has ≥ 14 horizontally adjacent black pixels
          return TRUE
      current_row = row immediately under current_row
    endwhile
    return FALSE
endAlgorithm

```

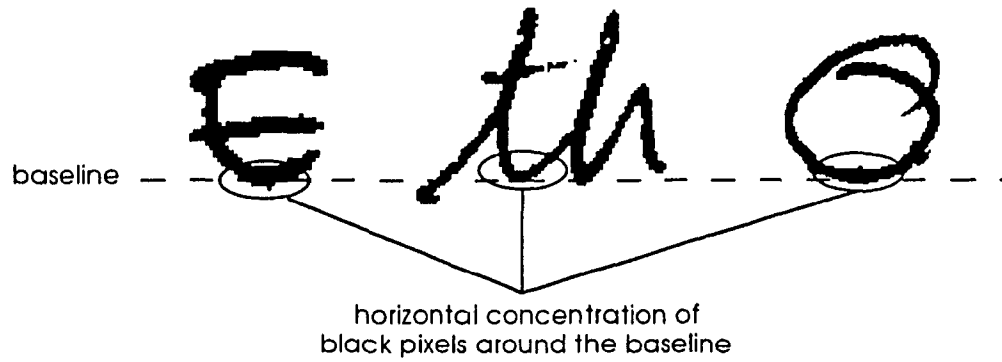
We could think that if instead of an “O”, we had a “th”, and the “h” could have a loop in its ascender (see Figure 3.18) which would have this feature at its top, however, we have observed that loops in a “h” are far less common than those without loops, and when loops are present, they are usually much more narrow than the one in a capital “O”. This difference in loop widths makes a difference in the width at the top of the loop, therefore finding 14 horizontally adjacent black pixels at the top of an “h” is rather rare. This remark regarding the “h” also applies for letter “l” in “Eleven”, where looped “l”s are less common than those without loops.

- *Noible category 5: Word starts with “th” as in “three”, or “E” in “Eleven”:* In this category the “th” and the “E” share a common feature with the “O” in the previous notable category. It is the first feature shown in Figure 3.16 that is present in the three cases. In the “E”, the feature corresponds to the lower horizontal stroke sitting on top of the baseline, and on the “th”, it would correspond to the bottom curve of the “t” connecting to the “h”. Figure 3.19 illustrates these three cases.

The “E” and the “th” also have their optimal separation point beyond the 20 pixel boundary from the beginning of the word, since for the “E”, it would be



**Figure 3.18:** More common and less common types of "h" in "th".



**Figure 3.19:** Common feature in "E" (notable category 5), "th" (notable category 5) and "O" (notable category 4).

along the right edge of the "l", and for the "th" it would be along the right edge of the "h". We can immediately say that if the word has other ascenders aside from the "El" in "Eleven", the "th" as in "three" and the "O" in "One", then the word cannot be "One", and hence it is classified under this notable category. Otherwise, recalling the third feature found in a "O", if we do not find those 14 horizontally adjacent black pixels, we conclude that the letter is not likely to be a "O", therefore we classify the letter in the notable category having the closest resemblance to "O", namely this category.

- *Notable category 6:* Word starts with lowercase "t" as in "ten". In this category the "t" has its bottom stroke connected to the next letter as shown, for instance, in Figure 3.8 (b). Therefore it shares a common feature with "O", namely the horizontal concentration of black pixels along the baseline. The distinguishing characteristic of this "t" will be the location of its optimal separation point, which is, in the great majority of cases, found within the 20 column distance from the beginning of the word (Figure 3.8 (b)).
- *Notable category 7:* Word starts with uppercase "N" as in "Nine", or "th" as in "three". In this category we find again "th", however, in this case, the "t" can be lowercase or uppercase and is not connected through the lower stroke to the "h". Therefore, the "th" and the "N" share a common feature which is the absence of the horizontal concentration of black pixels along the baseline. This is the feature which distinguishes this notable category from all previous ones. The "N" and "th" also share a common feature with notable category 5 and 4, namely the location of the optimal separation point, being beyond the 20 column threshold distance from the beginning of the word. For the "N", the bottom



section of Figure 3.7 shows the distant position of the optimal separation point with respect to the beginning of the word.

- *Notable category 8:* Word starts with "t" as in "ten", or uppercase "F" as in "Four". In this category, the "t" can be either uppercase or lowercase; however, it does not have its lower stroke connected to the next letter. Therefore the "t" and the "F" share a common feature with the previous notable category, namely the absence of the horizontal concentration of black pixels along the baseline. The feature that distinguishes this notable category from the previous one, however, is the position of the optimal separation point. Like in notable category 6, the point is found, in the great majority of cases, within the 20 column distance from the beginning of the word. The top portion of Figure 3.7 shows the location of the optimal separation point in uppercase letter "F".

- *Notable category 9:* Word starts with uppercase "N" as in "Nine" or "E" in "Eleven". This is actually the first category checked when classifying a notable letter. If the optimal separation point is beyond 25 columns to the right of the beginning of the word, and the word's length/body-height ratio is greater than 6.5 (corresponding to a word categorized as "long"), and finally, if there are no other ascenders aside the "N" or the "E", then the word is classified under this notable category. A 6.5 length/body-height ratio is indeed a large ratio for the word "Nine", however, the uppercase "N" is in most cases, the widest notable letter found in this particular vocabulary, therefore adding substantial length to the word. This notable category basically tries to filter in words categorized as "long" and without other ascenders except the ones present in the notable letters.

### 3.5 Words recognized entirely by the symbolic classifier (word category 1)

This word category consists of nine words (see Figure 3.20), which are recognized entirely by the symbolic classifier. The symbolic features discussed up to now, are sufficient to recognize the words. In the following pseudo-codes, we will make references to ascenders and descenders in different parts of the word to be recognized. The specific sections of the word in which each ascender or descender is searched will be given in the table shown in Figure 3.21, and referred to in the code as "<table item  $x$ >", where  $1 \leq x \leq 10$ .

**3.5.1 Recognizing "eight":** The following pseudo-code and Figure 3.22 illustrate the algorithm to recognize the word "eight". We can also visualize this algorithm by looking at Figure 2.9 and tracing along the path from the starting box in the tree (where the unknown word is given) and up to the leaf node "eight".

*Algorithm Eight;*

```
    if (ascender at end of word <table item 1>)
        if (descender around middle of word <table item 2>)
            if (length/body-height ratio < LONG_WORD) and
                (no other descenders)
                return "Eight"
```

*endAlgorithm*

In Figure 3.22, letters "t" and "g" provide all the necessary ascender/descender information to classify the word. Therefore the ascender in letter "h" will be ignored as it is dispensable to the classifier. The length/body-height threshold ratio for a word to be categorized as a "long" word is 6.5. In the example shown in Figure 3.22, the ratio is about 5.5.

Capital "O"  
One eight Eighty  
and hundred dollars  
thousand forty fifty  
Lowercase "f"  
(has a descender)

Figure 3.20: Words (category 1) recognized entirely by the symbolic classifier.

Item	Feature	window	left boundary	right boundary
1	<i>t</i> or <i>d</i> as in "eight" or "and"	ascender	$[5 / 6 \times \text{word\_length}]$	middle of blank after word.
2	<i>g</i> or middle <i>f</i> , as in "eight" or "fifty" respectively	descender	$[1 / 3 \times \text{word\_length}]$	$[2 / 3 \times \text{word\_length}]$
3	<i>y</i> as in "sixty"	descender	$[3 / 4 \times \text{word\_length}]$	end of word
4	<i>t</i> as in "ninety"	ascender	$[3 / 5 \times \text{word\_length}]$	end of word
5	<i>t</i> , right <i>l</i> , or <i>l</i> , in "...teen" "dollars", and "twelve" respectively	ascender	$[1 / 2 \times \text{word\_length}]$	$[0.85 \times \text{word\_length}]$
6	middle <i>d</i> as in "hundred"	ascender	$[5 / 12 \times \text{word\_length}]$	$[3 / 4 \times \text{word\_length}]$
7	right <i>l</i> in "dollars"	ascender	$[1 / 2 \times \text{word\_length}]$	$[3 / 5 \times \text{word\_length}]$
8	lowercase <i>f</i> as in "forty"	descender	middle of blank before word	$[1 / 4 \times \text{word\_length}]$
9	<i>h</i> in "thousand"	ascender	beginning of word	$[1 / 4 \times \text{word\_length}]$
10	<i>l</i> in "eleven"	ascender	beginning of word	$[1 / 2 \times \text{word\_length}]$

Figure 3.21: Search boundaries inside the ascender/descender windows for ascenders and descenders. *Left boundary* and *right boundary* are the leftmost and the rightmost column limits respectively used to focus the search of the corresponding feature to a specific word section. Section 3.5 uses this table in its algorithms to explain in detail how words in word category 1 are classified.

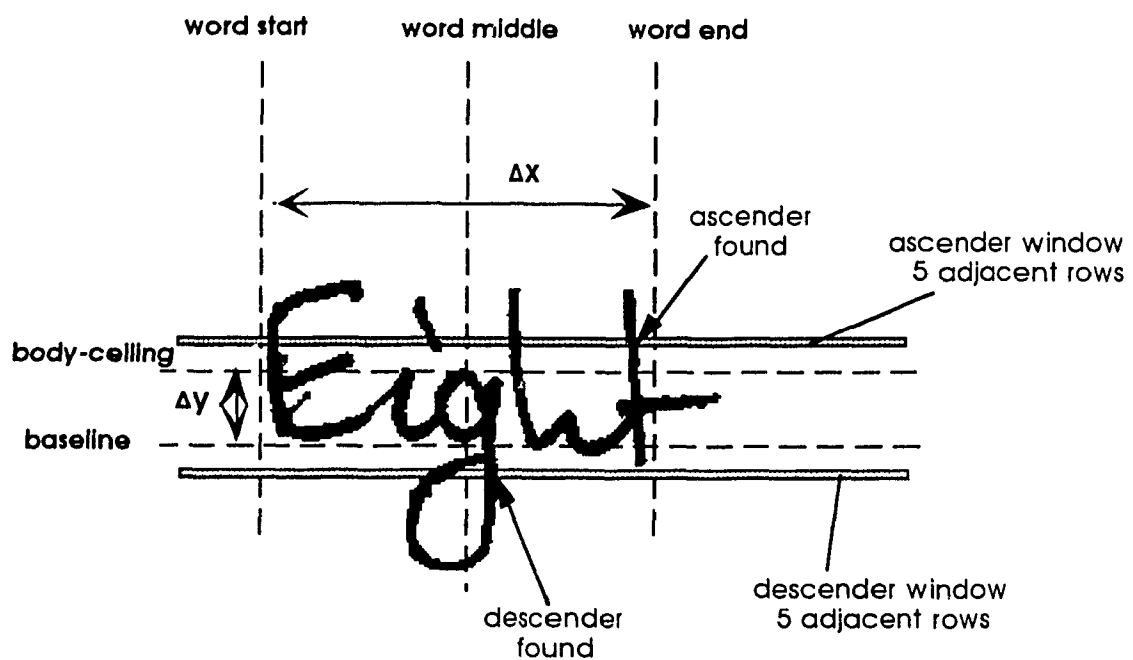


Figure 3.22: Recognizing "eight". Refer to algorithm *Eight* in section 3.5.1.

**3.5.2 Recognizing “eighty”:** If the “e” in the word “eighty” is lowercased, the algorithm is very similar to the one for “eight” except that it must find a descender at the end of the word, specifically item 3 from the table of Figure 3.21. This descender is part of the “y” in “eighty”. It must also check that there isn’t a notable letter at the beginning of the word. The following pseudo-code outlines the algorithm used to recognize the word “eighty” with a lowercase “e”:

*Algorithm eighty;*

```

    if descender at end of word <table item 3> and
      ascender at end of word <table item 4>
      if (descender around middle of word <table item 2>)
        if (no ascenders at the beginning of the word)
          if (length/body-height ratio < LONG_WORD) and
            (no other descenders)
            return “eighty”

```

*endAlgorithm*

If the “e” is capitalized, however, then it is a notable letter, and the information given by the ascenders and descenders positions is insufficient to classify the word. Some deeper analysis must be done to the notable letter at the beginning of the word. The pseudo-code in this case is the following:

*Algorithm Eighty;*

```

    if descender at end of word <table item 3>
      if descender around middle of word <table item 2>
        if there is a notable letter at the beginning of the word and
          notable letter is “E”
          if length/body-height ratio < LONG_WORD and
            no other descenders
            return “Eighty”

```

*endAlgorithm*

We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node “Eighty” (with a capital “e”) shown in Figure 2.10. Classifying the notable letter uses the information given by the ascenders and descenders to narrow down the possible letters to a minimum number. The only other word which has similar ascender/descender features as “Eighty” is “Fifty” written with a capital “F”; they both have a descender at the end, another descender at the middle, and a notable letter at the beginning. So, to distinguish these two words, we recall what was said previously regarding the difference between capital “E” and “F”; we look for a horizontal concentration of black pixels around the notable letter’s baseline, indicating the presence of a horizontal stroke typically present in a “E”. The algorithm to do this is identical to the one used to identify the first feature in the notable category 4 (that is “O”, described in section 3.4).

### 3.5.3 Recognizing “and”:

*Algorithm And;*

*if ascender at end of word <table item 1>  
if (length/body-height ratio < MEDIUM\_WORD) and  
(no other ascenders or descenders)  
return “and”*

*endAlgorithm*

The minimum length/body-height ratio corresponding to a “MEDIUM\_WORD” is 5.0. We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node “and” shown in Figure 2.12.

**3.5.4 Recognizing “One”:** It is important to recall that in this case we are recognizing “One” with a capital “o”. This is because the symbolic classifier

looks mainly for features outside the body, and that is where a capital "o" would reveal information to the classifier. Pseudo code follows:

*Algorithm One;*

*if length/body-height ratio < LONG\_WORD and no descenders  
if there is a notable letter at the beginning and  
notable letter is a "O" (notable category 4)  
return "One"*

*endAlgorithm*

We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node "One" shown in Figure 2.16.

### 3.5.5 Recognizing "Hundred":

*Algorithm Hundred;*

*if ascender at end of word < table item 1> and no descenders  
if ascender at beginning and middle of word <table items 9 and 6> and  
if length/body-height ratio > SHORT\_WORD  
return "hundred"*

*endAlgorithm*

The maximum length/body-height ratio for words categorized as "short" is: SHORT\_WORD = 4.3. We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node "hundred" shown in Figure 2.12.

### 3.5.6 Recognizing "Thousand":

*Algorithm Thousand;*

*if ascender at end of word < table item 1> and no descenders  
if ascender at beginning of word <table item 9> and no other ascenders  
if length/body-height ratio > SHORT\_WORD*



*return "thousand"*

*endAlgorithm*

We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node "thousand" shown in Figure 2.12

**3.5.7 Recognizing "dollars":** This word is slightly more complex to recognize since the position of the ascender in the rightmost "l" is found inside the same region where the "t" in words such as "sixteen", is found. Words ending with "teen", however, have a length/body-height ratio heuristically greater than 5.0, and the word "dollars" may sometimes be above that threshold or below it, depending on the writer. Therefore depending on the word's length/body-height ratio, different algorithms will be used. In the code below, a word categorized as "medium" has a minimum length/body-height ratio of MEDIUM\_WORD = 5.0.

*Algorithm Dollars*

```
if length/body-height ratio > MEDIUM_WORD and no ascenders  
    if ascender around 3/4 of word <table item 5> and ascender is right "l" as  
    in "dollars"(will be explained shortly)  
        return "dollars"  
else if length/body-height ratio > SHORT_WORD and no descenders  
    if ascender around 3/4 of word <table item 7> and the ascender isn't part  
    of a capital "O" as in "One"  
        return "dollars"
```

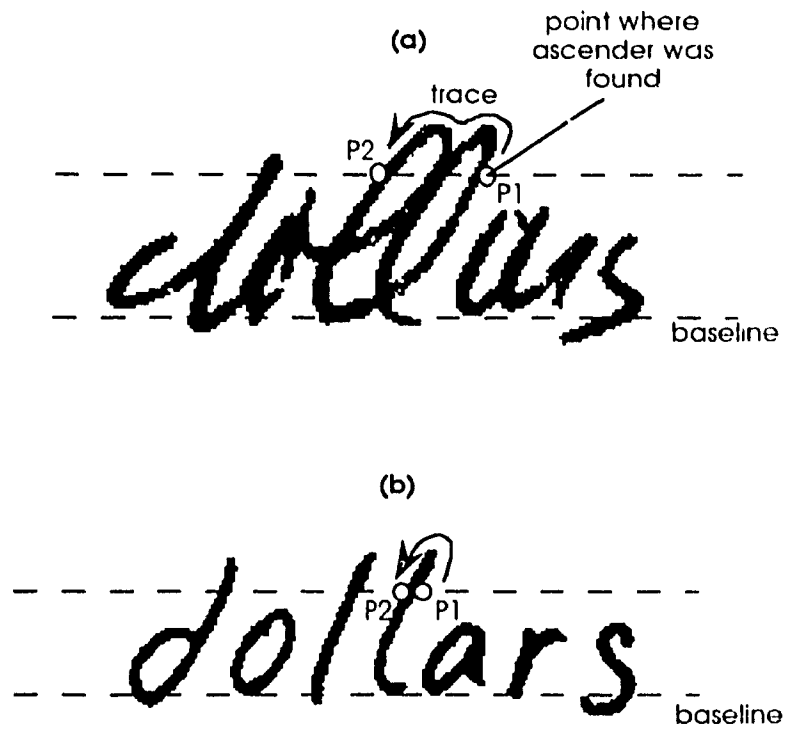
*endAlgorithm*

We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node "dollars" shown in

Figures 2.13 and 2.14. There are indeed two paths to reach the same leaf node "dollars".

In the second line of the above pseudo-code, it is mentioned that the ascender found was recognized as the right "l" in "dollars". The recognition consists of the following steps: First, we start tracing upwards the ascender, starting from the row and column where it was found. Tracing stops when the initial row is reached again (steps [1] to [3] in algorithm *Is\_letter\_L* below). At this point, if the unknown word is "dollars", we should be either on the left edge of the right "l" (Figure 3.23 (b)) or on the left edge of the left "l" (Figure 3.23 (a)) if the two "l" touch at a row above the trace starting row. In the latter, it is usually because the two "l"s have overlapping loops (see Figure 3.23 (a)). Let us call *P1*, the starting point of the trace, and *P2* the end-point of the trace, as shown in Figure 3.23.

Since the word "dollars" is the only word in the vocabulary which has no descenders and two ascenders around the middle of the word, we can say the following: If the distance between *P1* and *P2* is greater than a threshold value of  $\lfloor 1/6 \times \text{word\_length} \rfloor$ , then there most probably is another ascender next to the first one found, and the two are most probably touching since the distance between *P1* and *P2* is so important (Figure 3.23 (a)). Therefore if this condition is satisfied, the ascenders are recognized as double "l", as shown in Figure 3.23 (a) (step [4]). However, if the condition is not satisfied, *P2* is most probably on the left edge of the right "l", as shown in Figure 3.23 (b). If the distance from *P1* and *P2* is greater than a threshold value of 12 pixels (step [5]), then the right "l" is suspected to have a loop and is immediately checked for it (step [6]. The specifics regarding the detection of a loop will be discussed shortly). If no loop is found, the ascender is rejected as an "l" (step [10]). If a loop is present, however, we scan horizontally from *P2*, towards the left until we reach a black pixel *P3* (see Figure



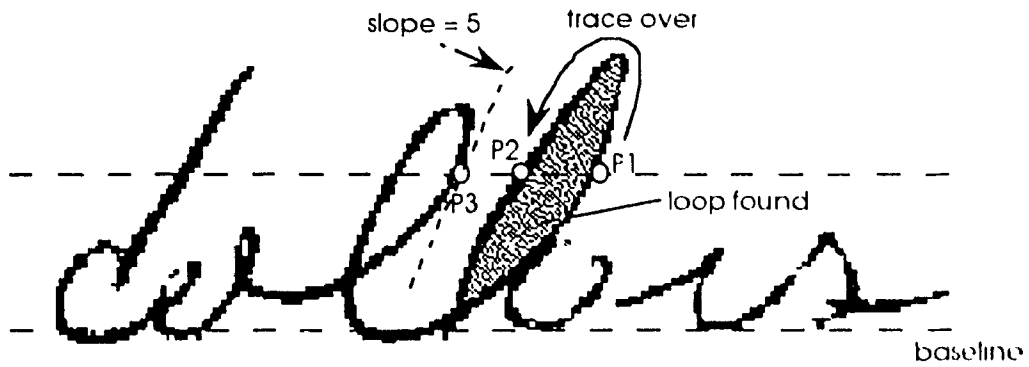
**Figure 3.23:** Tracing over the right “l”. Steps [1] to [3] in algorithm *ls\_letter\_l*, described in section 3.5.7.

3.24 (a)) (step [7]). If the distance between  $P2$  and  $P3$  is greater than a threshold value of  $\lfloor 1/6 \times \text{word\_length} \rfloor$ , the distance is too important to be the separation between two "l"s in "dollars" (step [8]). Therefore the ascender found at  $P1$  is rejected as an "l". However, if the distance between  $P2$  and  $P3$  is less than or equal to  $\lfloor 1/6 \times \text{word\_length} \rfloor$ , and the stroke at  $P3$  is categorized as near-vertical (absolute value of the slope is greater or equal to 1), the ascenders at  $P1$  and  $P3$  are recognized as two "l"s (step [9]). See Figure 3.24 (a).

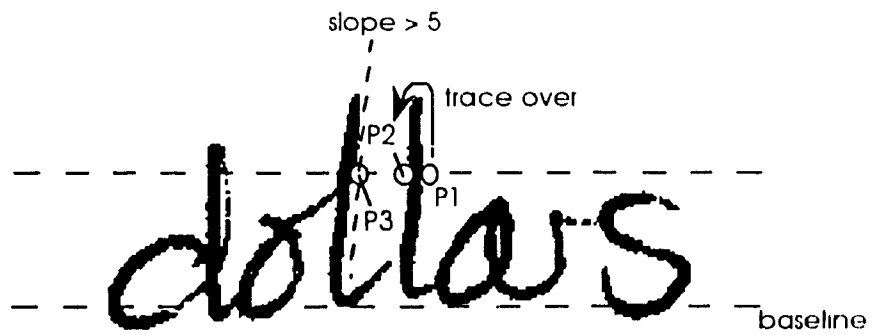
If the distance between  $P1$  and  $P2$  is below the threshold value of 12 pixels, the stroke between the two points is most probably a single-stroke letter such as a non-looped "l" (Figure 3.24 (b)). We must now, however, find the second "l". In the same way described before, we scan horizontally from  $P2$ , towards the left until we reach a black pixel  $P3$  (step [11]). If the distance between  $P2$  and  $P3$  is greater than a threshold value of  $\lfloor 1/6 \times \text{word\_length} \rfloor$ , the distance is too important to be the separation between two "l"s in "dollars" (step [12]). Therefore the ascender found at  $P1$  is rejected as an "l". However, if the distance between  $P2$  and  $P3$  is less than or equal to  $\lfloor 1/6 \times \text{word\_length} \rfloor$ , and the stroke at  $P3$  is categorized as near-vertical (absolute value of the slope is greater or equal to 1), the ascenders at  $P1$  and  $P3$  are recognized as two "l"s (step [13]). See Figure 3.24 (b). The following algorithm outlines the algorithm described in these 3 last paragraphs: recognizing letter "l" from word "dollars" given an unknown word.

*Algorithm Is\_letter\_L*

- [1]  $P1$  = pixel where <table item 5> was found.
- [2]  $\text{start\_row}$  = row where <table item 5> was found
- [3] Start tracing upwards from  $P1$ . Stop when  $\text{start\_row}$  is reached again. Call this pixel  $P2$ .
- [4] if  $(P1 - P2) > \lfloor 1/6 \times \text{word\_length} \rfloor$   
           return TRUE



(a) looped "l"s were found uncommon



(b) one stroke "l" were commonly found

**Figure 3.24:** Recognizing different types of "l"s (algorithm *Is\_letter\_l* in section 3.5.7).

```

[5]   if ( $P1 - P2 > 12$ )
[6]       if there is a loop between  $P1$  and  $P2$ 
[7]           Scan left from  $P2$  until a black pixel is found. Call this pixel  $P3$ 
[8]           if ( $P2 - P3 > \lfloor 1/6 \times \text{word\_length} \rfloor$ )
[9]               return FALSE
[10]          if absolute value of stroke at  $P3$  is  $\geq 1$ 
[11]              return TRUE
[12]          return FALSE
[13]       else
[14]           return FALSE
[15]   else
[16]       scan left from  $P2$  until a black pixel is reached. Call this pixel  $P3$ 
[17]       if ( $P2 - P3 > \lfloor 1/6 \times \text{word\_length} \rfloor$ )
[18]           return FALSE
[19]       if absolute value of stroke at  $P3$  is  $\geq 1$ 
[20]           return TRUE
[21]       return FALSE
[22] endAlgorithm

```

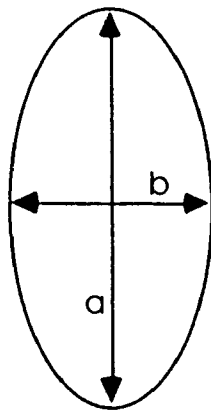
In the *else* part of algorithm *Dollars*, it is mentioned that the ascender found should not be part of a capital "O" as in "One". The algorithm that checks for this condition is almost identical to the one used to identify the first feature in notable category 4 ("O"), namely the horizontal concentration of black pixels. Instead of checking for the horizontal concentration of black pixels, we measure the distance between the left side and the right side of the inside loop, and if it is less than  $\lfloor 0.22 \times \text{word\_length} \rfloor$ , the ascender is not part of a capital "O".

**Detecting the presence of a loop:** Searching for loops is mostly used when analyzing the word "dollars", where the "l" could feature loops instead of simple straight vertical bars. The algorithm to detect loops is as follows: Suppose we wanted to search for a loop in the right "l" of the word "dollars" shown in Figure 3.24(a). Starting from  $P2$  shown in the same figure, we scan horizontally to the

right until we reach a white pixel, indicating the inside region of the potential loop. We then give an estimate of the loop perimeter in pixels by approximating it with an elliptic shape (Figure 3.25).

The perimeter of an ellipse is formally given by:

$$P = 4a \int_0^{\frac{\pi}{2}} \sqrt{1 - k^2 \sin^2 \theta} d\theta$$

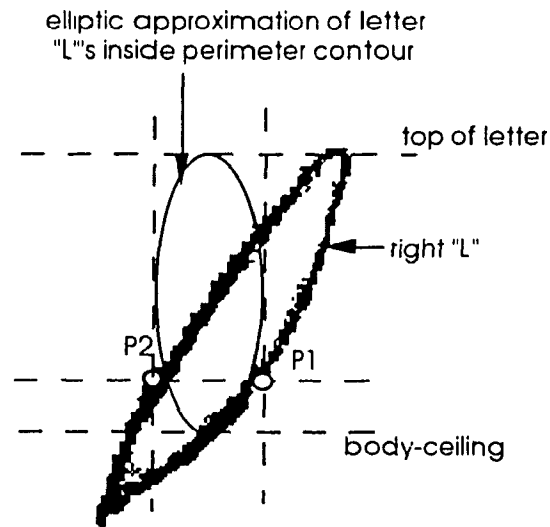


$$\text{where } k = \sqrt{\frac{a^2 - b^2}{a}}$$

However, these integral calculations are quite heavy and time consuming. Furthermore the extreme accuracy given by the integral is not necessary and can be approximated by the much faster version of  $P$  described below:

$$P = 2\pi \sqrt{\frac{1}{2}(a^2 + b^2)}$$

$a$  will be equal to the distance between the upper body-limit and the top of the "L",  $b$  will be the distance between  $P1$  and  $P2$ . We can now estimate the number of pixels to be traced around the inside of the loop. In practice the number of pixels traced in a loop never exceeds  $P$  due to the nature of the approximation, therefore  $P$  will be the threshold number of pixels allowed to be traced along the inside of the loop. If the trace exceeds this threshold before coming back to the starting point, the trace is most probably going around the entire letter in which



**Figure 3.25:** Identifying loops by approximating the right "L" loop with an ellipse (last paragraph in section 3.5.7).



case the loop is said to be absent. Otherwise, we say a loop is present.

**3.5.8 Recognizing "forty":** In this algorithm, we are only recognizing the word "forty" with a lowercase "f" which has a descender. Words with uppercase "F" will be processed differently and will be discussed later. Although the correct spelling for this word is f-o-r-t-y, it has been observed that many people mistakenly spell it as f-o-u-r-t-y. We decided to handle and accept these spelling mistakes as the percentage was too high to be ignored. Since we are allowing one extra letter in this relatively short word, the length/body-height ratio for this word becomes significantly less accurate, hence we will disregard it, counting on the descender/ ascender information.

*Algorithm forty;*

*if descender at end of word <table item 3>*

*if descender at beginning of word <table item 8> and no other descenders*

*return "forty"*

*endAlgorithm*

We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node "forty" shown in the same figure.

**3.5.9 Recognizing "fifty":** In this algorithm, the word "fifty" is processed differently depending on the capitalization of the first letter "f". If the "f" is lowercased, then the pseudo code is the following:

*Algorithm fifty;*

*if descender at end of word <table item 3>*

*if descender at beginning of word <table item 8>*

*if descender around middle of word <table item 2>*

*return "fifty"*

*endAlgorithm*

We can visualize the above algorithm by tracing along the path between the beginning of the tree shown in Figure 2.9 up to leaf node "fifty" shown in the same figure.

If the "f" is uppercased, however, the information given by the ascenders and descenders positions is insufficient to classify the word. It is actually the same situation as we saw before with "Eighty" written with an uppercase "E". And as it was mentioned for "Eighty", the only other word which has similar ascenders/descender features as "Eighty" is "Fifty" written with a capital "F"; they both have a descender at the end, at the middle and a notable letter at the beginning. Therefore, to distinguish "Fifty" from "Eighty", we will use the same distinguishing feature used for "Eighty", namely, if we DO NOT find a horizontal concentration of black pixels around the baseline of the notable letter, the word is classified as "Fifty".

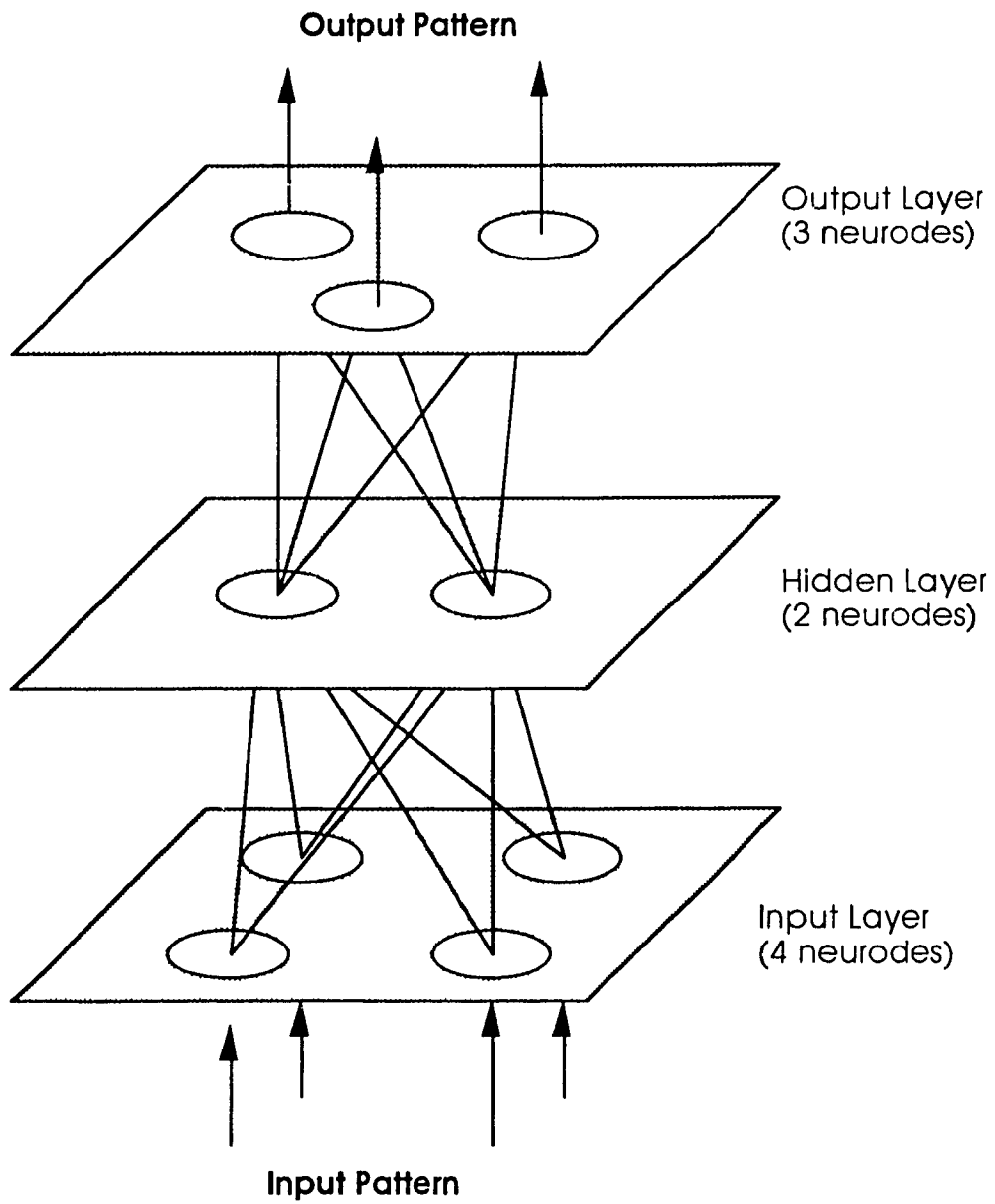
In future chapters, we will see categories of words requiring deeper analysis in order to classify them. The information collected by the symbolic classifier needs to be completed with the information provided inside the body of words. This area will be the working domain of the neural classifier, which is described in the next chapter.

## Chapter 4

### The Neural Classifier

During the recognition of an unknown word, the neural classifier is used (when necessary) to complement the information extracted by the symbolic classifier. In this chapter, we will cover only words which are recognized entirely by the neural classifier; to understand better where the neural classifier is used to recognize these words, we can look back at the clustered-leaf node appearing in Figure 2.14.

A neural network is, in this case, an algorithm, whose design was motivated by the design and functioning of human brains and components thereof. It is a network of many very simple processors called *neurodes* ("artificial neurons"), each having a small amount of local memory. These neurodes are connected by unidirectional communication channels ("connections"), which carry numeric (as opposed to symbolic) data. In our thesis, we have chosen a specific type of neural network called a backpropagation neural network which is simply a neural network having 3 layers or more of neurodes and uses a special learning rule called backpropagation, or Delta Rule (Zurada, 1992). In these networks, the neurodes are grouped by layers and operate only on their local data and on the inputs they receive from the previous layer via the connections. Neurodes are not connected with other neurodes within the same layer. For a typical 1-hidden layer backpropagation network like the simplified one shown in Figure 4.1, each layer is denominated as follows:



**Figure 4.1:** Typical 1-hidden layer backpropagation network.

- *The input layer:* layer receiving the input pattern. In our case, we are classifying words coming from black and white digital images; therefore each neurode will receive a pixel value from the image. For simplicity, Figure 4.1 shows only 4 neurodes in the input layer; architectural details of our specific implementation will be given shortly.
- *The hidden layer:* In Figure 4.1, there are 2 neurodes in this layer. Every neurode from the input layer is connected to every neurode in the hidden layer; the input layer is said to be *fully connected* to the hidden layer.
- *The output layer:* Figure 4.1 shows 3 neurodes in the output layer. As we can observe, the hidden layer is fully connected to the output layer. The conjunction of the individual outputs from the output neurodes yields the final result returned by the neural network.

The one-hidden-layer neural network (Goodman, 1993) used in our system works interactively with the symbolic classifier. As learning model, the network uses a combination of the Backpropagation algorithm (Zurada, 1992) and its accelerated version, Quickpropagation written by Scott Fahlman (1988). The net's function is to recognize word segments (Plamondon, 1991) in order to complete the recognition started by the symbolic classifier. In other words, the symbolic classifier uses the neural network when it needs to read the body of a word.

## 4.1 Input Patterns

Designing the architecture of the neural classifier was crucial, since it was necessary for it to be fast, accurate, and require the least possible number of training samples (Bichsel and Seitz, 1992). The input matrix to the neural network

has 50 columns and 10 rows. Therefore, the total number of neurodes in the input layer is 500. As an example, Figure 4.2 shows the input pattern for "nine".

## 4.2 Output Patterns

There are 19 possible output patterns. These are segments of words, which need partial or total neural recognition. These segments are always at the beginning of a word, or they are preceded by a letter containing an ascender. The output patterns (19 word segments) are the following:

one	: entire word
wo	: from "two"
ree	: from "three"
hree	: from "three"
our	: from "four" and "fourteen"
ive	: from "five"
six	: entire word
seven	: entire word
nine	: entire word
en	: from "ten"
ve	: from "twelve"
ir	: from "thirty" and "thirteen"
hir	: from "thirty" and "thirteen"
or	: from "forty"
wen	: from "twenty"
een	: as in "nineteen" or other words ending with "een"
ix	: from "Six"
even	: from "Seven"
ine	: from "Nine"

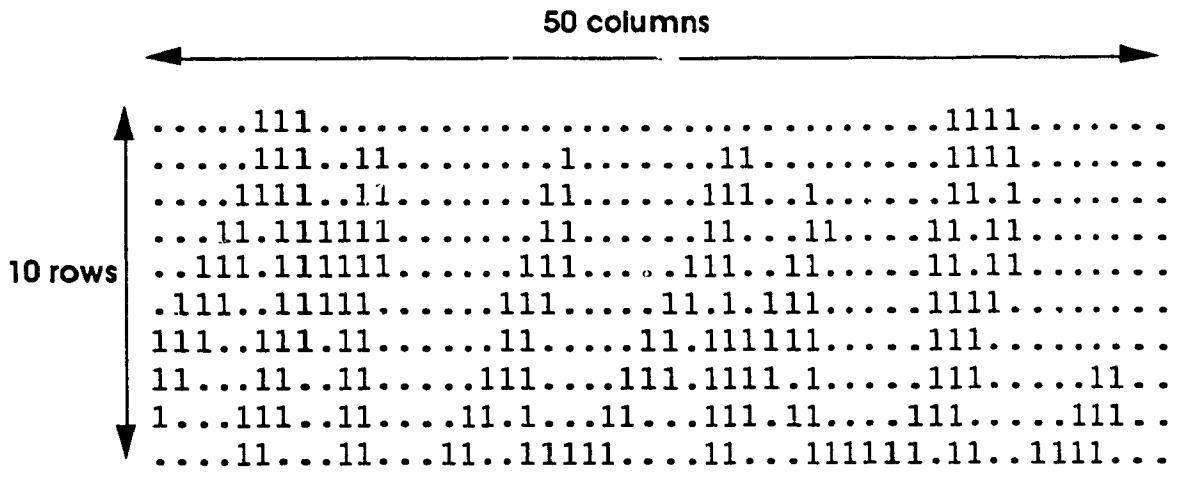


Figure 4.2: Example of pattern ("nine") in input matrix.

As shown above, each output pattern belongs to only one word in the vocabulary with the exception of output pattern "een". We might wonder why there are two word segments "ree" and "hree" for the word "three" and two word segments "ir" and "hir" for words "thirty" and "thirteen". The reason behind it will be given in the next sections.

### 4.3 Neurodes and Connections

The neurodes in the network have real-valued weights. The activation function used is the sigmoidal function:

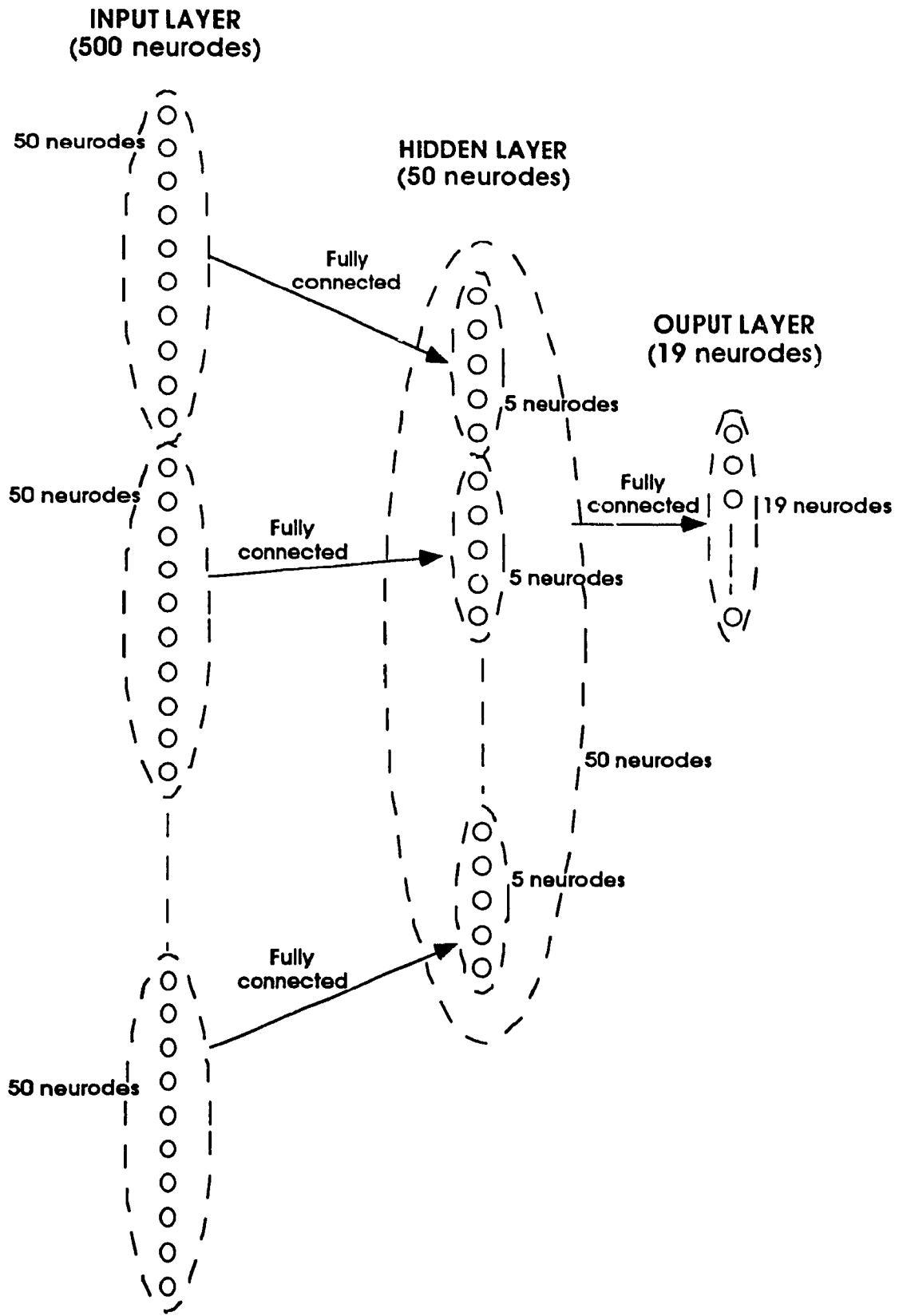
$$S(x) = \frac{1}{1 + e^{-\alpha x}}$$

where  $\alpha = 1$ .  $S(x)$  values range from -0.5 to +0.5. The output function is the identity function. In order to maximize the neural network's response time without losing much accuracy, connections between layers were carefully designed (Le Cun, Boser, Denker, Henderson, Howard, Hubbard and Jackel, 1992). From the input layer to the hidden layer, each group of 50 adjacent neurodes is fully connected to 5 adjacent neurodes in the hidden layer. Hence there are ten groups of 50 neurodes in the input layer connected to ten groups of 5 neurodes in the hidden layer. The hidden layer and the output layer are fully connected. Biasing is absent in this implementation. Therefore the total input to each neurode  $u_i$  in the hidden or the output layer will be

$$net_i = \sum_j w_{ij} \cdot inp_j$$

where  $w_{ij}$  is the weight assigned to  $inp_j$ , the input received by  $u_i$  from neurode  $u_j$  of the previous layer. Figure 4.3 shows an overview of the network's architecture.





**Figure 4.3:** Overview of the neural network architecture.

#### **4.4 Words recognized entirely by the neural classifier (word category 2)**

In this chapter, we will be concerned only with words who are classified entirely by the neural classifier (word category 2). In the next chapter we will show the rest of the words which were not covered up to now. Words in category 2 have no ascenders nor descenders. The only information about them is contained inside the body. All the symbolic classifier can provide is the length/ height ratio of these words. For this reason, these words will be recognized entirely by the neural classifier. Figure 4.4 shows the set of words in this category. Let us recall what was stated at the beginning of this chapter: for words in word category 2, the neural classifier is used as shown in the clustered-leaf node appearing in Figure 2.13. If we trace along the path from the start box in the tree (Figure 2.8) where the unknown word is given, and up to the clustered-leaf node in Figure 2.13, we notice that all the features searched by the symbolic classifier were absent, which further illustrates the nature of words in word category 2.

#### **4.5 Preparing the input to the neural network**

This phase uses a symbolic procedure. These routines are responsible for finding the start and end points of all word segments to be sent to the neural network. They must also normalize the word segments in order to scale them to the neural network's input matrix.

**4.5.1 Finding the start and end points of word segments:** Since words in category 2 are recognized entirely by the neural classifier, these words will have only one word segment to be sent to the network. The segment is the entire

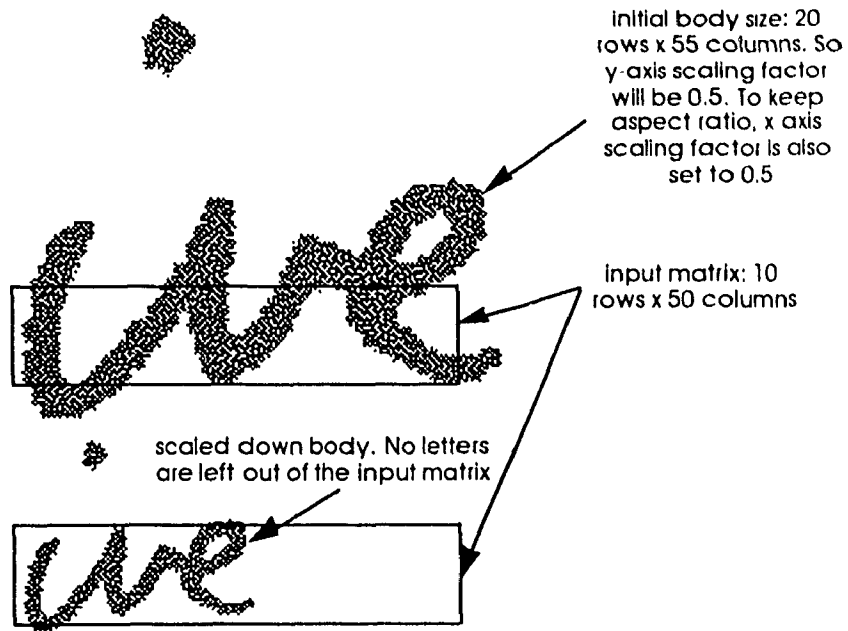
one six  
seven nine

**Figure 4.4:** Words recognized entirely by the neural classifier (category 2 words).

word, therefore the start and end points will be the beginning and the end of the word respectively.

**4.5.2 Normalizing the word segments:** Since the words are scanned with a resolution of 300 dots per inch, and the input matrix to the neural network is 10 rows by 50 columns, the bodies of words will have to be scaled down to 10 rows in height, in order to feed them into the matrix. Ideally, to preserve the aspect ratio (that is, the ratio of the horizontal and vertical proportions) in this scaling process, the y-axis scaling factor should be the same as the x-axis scaling factor (see Figure 4.5). However, sometimes the word "seven", for instance, can be written very wide and not very tall, so applying the y-axis scaling factor to the x-axis could lead to losing letters at the end of the segment. Therefore, before scaling down, we verify that the calculated y-axis scaling factor will make the word segment fit in height as well as in length inside the matrix. In the case where the y-axis scaling factor is too small to make the word segment fit in length, we change the x-axis scaling factor to make all letters fit (see Figure 4.6).

The actual scaling is done by a common technique of removing equally spaced rows and columns from the body, in order to uniformly reduce the number of pixels in the image. For example, let us take a word segment such as the one shown in Figure 4.7, in order to make the height of the segment fit inside the input matrix, the y-axis scaling factor must be equal to  $10/32 = 0.31$ . This means that  $\lfloor (1 - 0.31) \times 32 \rfloor = 22$  rows must be deleted from the segment. Similarly,  $\lfloor (1 - 0.31) \times 75 \rfloor = 51$  columns must be deleted to preserve the aspect ratio. We can see that in this case, using the y-axis scaling factor for the x-axis scaling factor does indeed make the segment fit in length inside the input matrix:



**Figure 4.5:** Normalizing word segments while preserving aspect ratios.

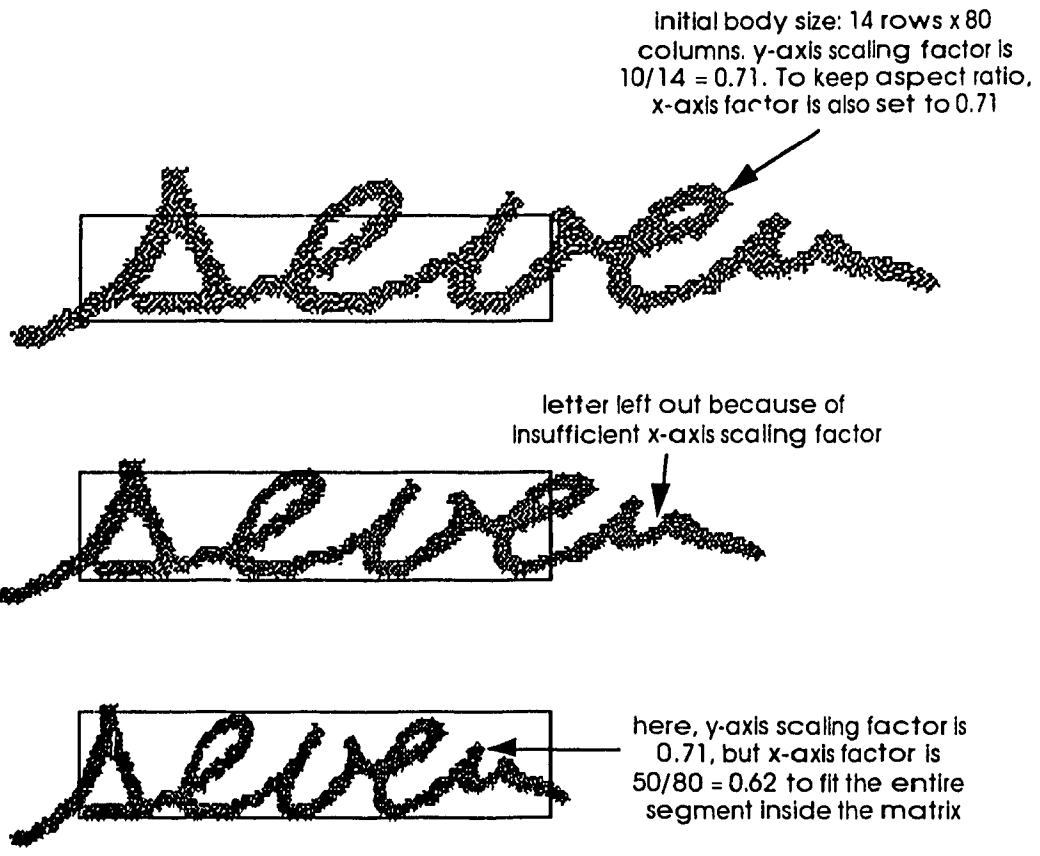
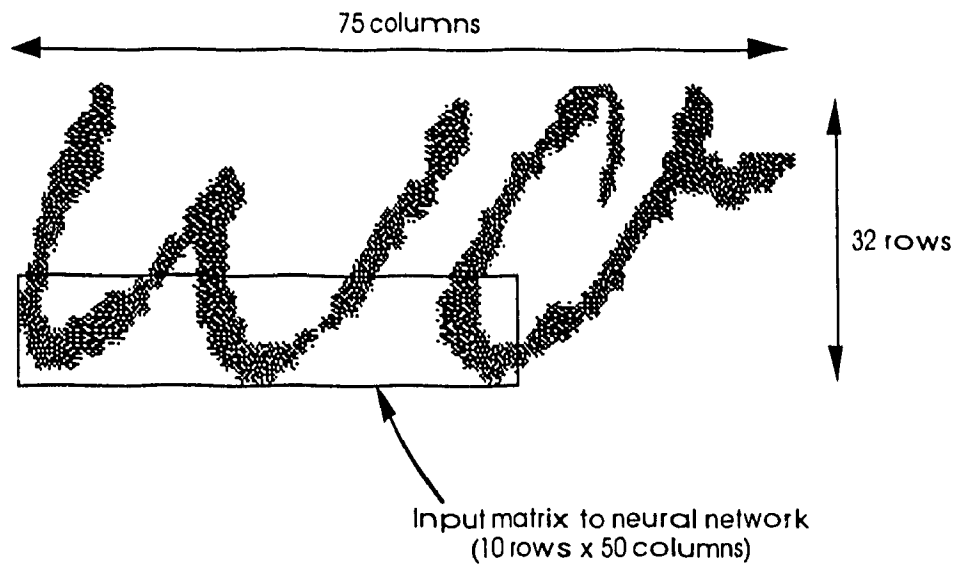


Figure 4.6: Normalizing wide word segments.



**Figure 4.7:** Example of a word segment to be scaled down to fit inside the neural network's input matrix.

75 columns - 51 columns < 50 columns. We must now select 22 uniformly distributed rows across the segment (see Figure 4.8), as well as 51 uniformly distributed columns (Figure 4.9). Therefore we will start scanning the rows from the top of the segment to the bottom, removing a row every  $(32 / 22) = 1.45$  rows. It seems odd to talk about 1.45 rows since there is no such thing as a .45 row, however, 1.45 can neither be floored or ceilinged as it would dramatically change the spacing between deleted rows. To resolve this problem of removing every 1.45 rows, we will just say that to delete the  $n^{th}$  row ( $n = 1$  to 22), we delete row number  $\lfloor 1.45 \times n \rfloor$ . The same technique is applied for deleting the 51 columns. The scaled down image is shown in Figure 4.10. To fully understand the steps discussed above, the following pseudo-code outlines the main steps:

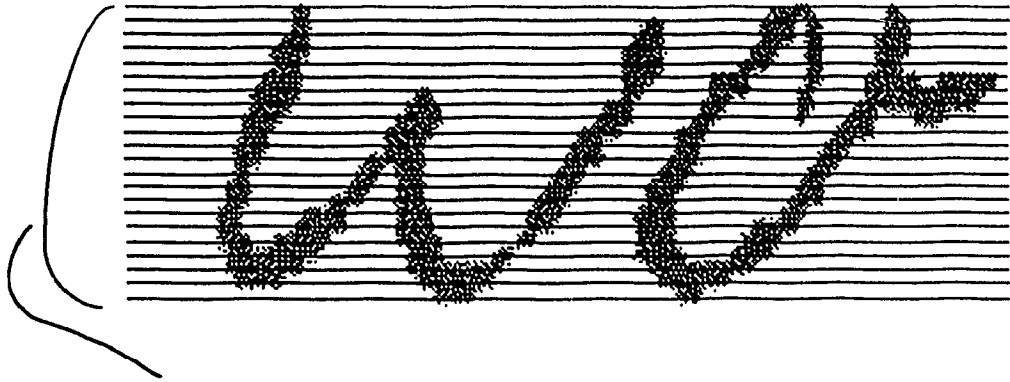
*Algorithm Normalize;*

```

[1]   y_scaling_factor = 10 (input matrix height) / word_segment_height
[2]   if (y_scaling_factor × word_segment_length) ≤ 50 (input matrix length)
        x_scaling_factor = y_scaling_factor (preserves the aspect ratio)
    else
        x_scaling_factor = 50 / word_segment_length (compresses more the x-
        axis than the y-axis to make all letters fit inside the input matrix)
[3]   rows_to_delete = (1 - y_scaling_factor) × word_segment_height
[4]   columns_to_delete = (1 - x_scaling_factor) × word_segment_length
[5]   deleted_row_spacing = spacing between rows to be deleted =
        word_segment_height / rows_to_delete
[6]   if x_axis_scaling_factor = y_axis_scaling_factor
        deleted_column_spacing = spacing between columns to be deleted =
        deleted_row_spacing
    else deleted_column_spacing = word_segment_length / columns_to_delete
[7]   for (row = 1 to rows_to_delete)
        delete row #  $\lfloor \text{deleted\_row\_spacing} \times n \rfloor$ 
[8]   for (column = 1 to columns_to_delete)
        delete column #  $\lfloor \text{deleted\_column\_spacing} \times n \rfloor$ 
endAlgorithm

```

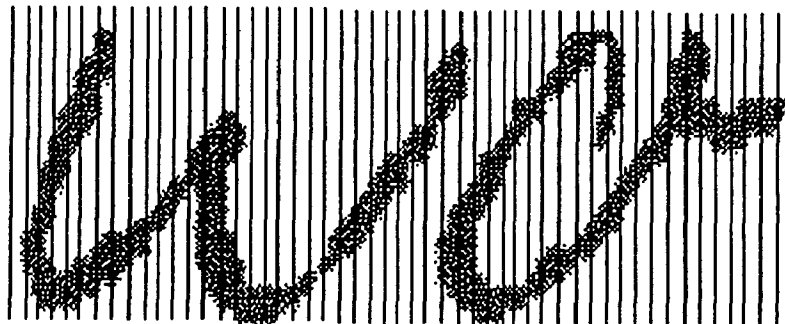




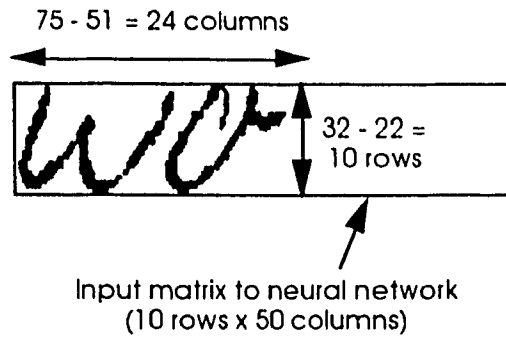
22 rows to be deleted. space  
between rows = 1.45 rows

**Figure 4.8:** Selecting the 22 uniformly distributed rows to be deleted.

51 columns to be deleted. Space between columns = 1.45 columns



**Figure 4.9:** Selecting the 51 uniformly distributed columns to be deleted.



**Figure 4.10:** Scaled down image now fits inside the input matrix.

In general, a 10 x 50 matrix accommodates comfortably all word segments.

**4.5.3 Anchoring the input pattern:** Anchoring the input pattern resulting from a normalization, consists of shifting the word segment inside the matrix, towards the bottom left corner; horizontal shifting is allowed whenever a blank column separates the segment from the leftmost column in the matrix. Similarly, vertical shifting is allowed whenever blank rows separate the segment from the bottom row inside the matrix. The idea behind this is to optimize the consistency of patterns inside the input matrix. The more consistent they are, in terms of location within the matrix, the more accurate the learning and recognition.

## **4.6 Feeding word segments through the neural network**

For each word from word categories 2 and 3 (covered in the next chapter), the following process is done: each word segment is fed through the neural network, each time producing 19 output neurode values. Discarding the output neurodes corresponding to output patterns disabled by a masking technique (will be discussed in the next chapter), the neural network returns the output pattern corresponding to the highest valued output neurode. Since words from category 2 are fed entirely through the neural network, the answer from the neural network will immediately yield the answer to the classification of the unknown word. Examples classification of words in category 2 will be shown in Chapter 6.

# Chapter 5

## Symbolic/Neural Classification

In this chapter, we will cover the last set of words to be classified. These are classified jointly by the symbolic and neural classifiers. To understand better where the neural classifier is used to partly classify these words, we can look at the clustered-leaf nodes shown in Figures 2.11, 2.16 and 2.18. It is also used in the non-leaf node appearing in Figure 2.13.

### **5.1 Words recognized jointly by the symbolic and neural classifiers (word category 3)**

The words in this category cannot be recognized fully by the symbolic classifier due to insufficient information in the ascender and descender windows. To complete the recognition, information inside their body must be extracted. This word category consists of 24 words, which can be separated into two groups. The first group (called type 1) consists of nineteen words, each containing one segment recognized by the neural net (Figure 5.1). The circled areas indicate the word segments to be sent to the neural network. The second group of five words (called type 2) are such that each word contains two segments recognized by the neural net (Figure 5.2). In both groups, words can be written with or without a notable letter at the beginning except for those pointed to by arrows.

Adding up the number of words in the three word categories (word category 1 with 9 words, word category 2 with 4 words and word category 3 with 24 words)

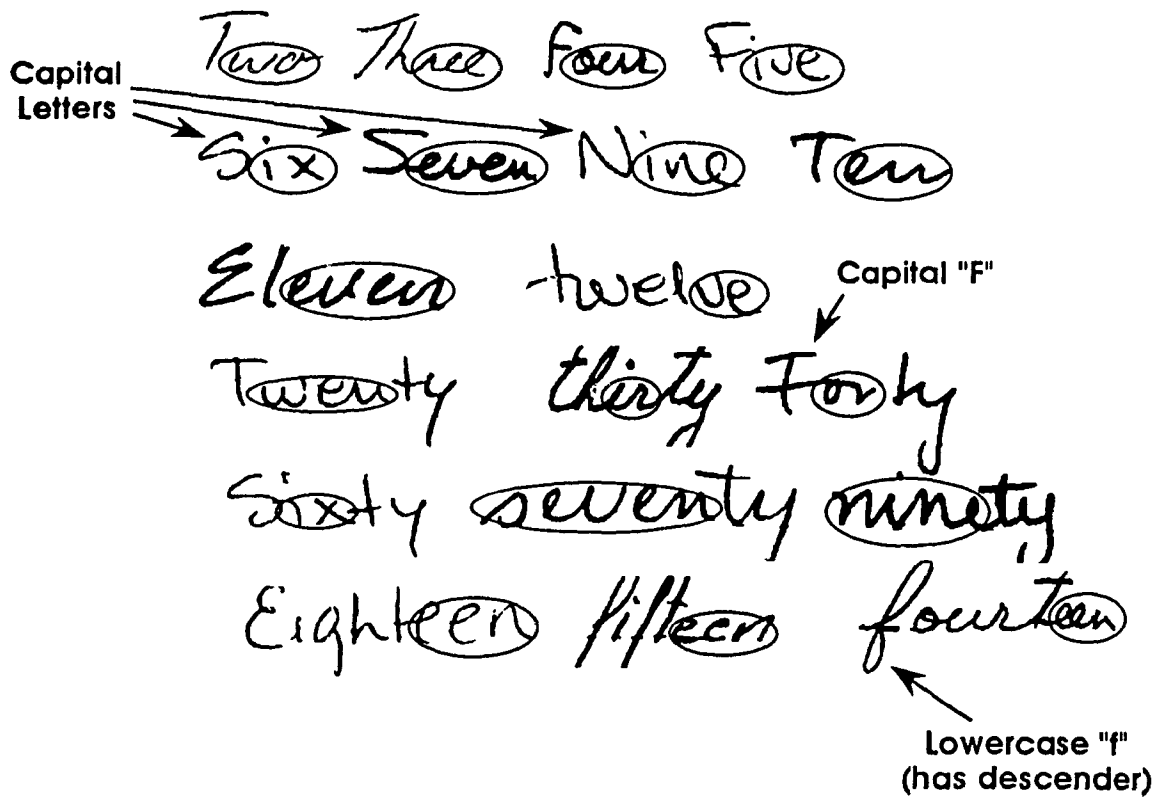


Figure 5.1: Type 1 words in word category 3, where one segment is recognized by the neural classifier. The remaining part of the word is recognized by the symbolic classifier.

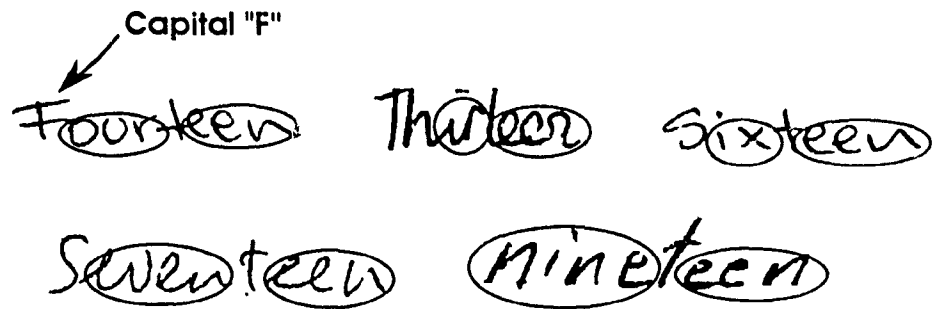


Figure 5.2: Type 2 words from word category 3, where 2 segments are recognized by the neural classifier. The remaining part of the word is recognized by the symbolic classifier.

gives us a total of 37 words which is higher than the number of words in our vocabulary (31 words). This is because the following words belong to more than one word category:

- “one”: If the “o” is capitalized, then it is a notable letter and the word belongs to word category 1; otherwise, it belongs to word category 2.
- “six”, “seven”, “nine”: When the first letter is capitalized, then it is a notable letter and the words belong to word category 3; otherwise, they belong to word category 2.
- “forty”: When the “f” is capitalized, it has no descenders and the word belongs to word category 3; otherwise, it belongs to word category 1.

In addition, although “fourteen” always belongs to word category 3, it is treated as a type 1 word when “f” is not capitalized, and as a type 2 word otherwise.

## **5.2 Preparing word segments from word category 3, to be sent to the neural network**

The process of preparing word segments to be sent to the neural network takes place before invoking the neural network. Therefore, in Figures 2.10, 2.15, 2.17 and 2.12, this process is done right before the nodes send the segments to the neural network.

**5.2.1 Finding the start and end points of word segments:** Finding the start and end points of category 3 words is complex due to the division of words into multiple regions to be analyzed by different classifiers.

For words ending with “ty” such as “sixty”, the “t” (feature item #4 from Figure 3.21) and the “y” (item #3 from the same figure) will be identified by the



symbolic classifier, hence the end of the word segment to be sent to the neural network will be immediately to the left of the "t". It is important to recall that the symbolic classifier is providing the location where the "t" was found; therefore to find the end of the word segment, we trace letter "t" upwards starting from where it was found in the ascenders window, and stopping once we have gone over its top and back down to the middle row inside the body. The start-point of the word segment will depend on whether the word has a notable letter at the beginning. If it does, then optimal separation point discussed in the previous chapter, will be taken as start-point. Otherwise, the segment will start at the beginning of the word. The following pseudo-code outlines the main steps discussed in this paragraph.

*Algorithm (1) Find\_start\_end\_points;*

- [1] *start\_point = Find\_Separation\_Pixel(); {call algorithm Find\_Separation\_Pixel covered in section 3.3}*
- [2] *feature\_pixel = pixel inside the ascenders window where item #4 in Figure 3.21 was found*
- [3] *starting from feature\_pixel, trace upwards the right edge of the "t". Stop when the trace goes down the left edge of the "t" and down to the middle row inside the body.*
- [4] *end\_point = last pixel in the trace.*

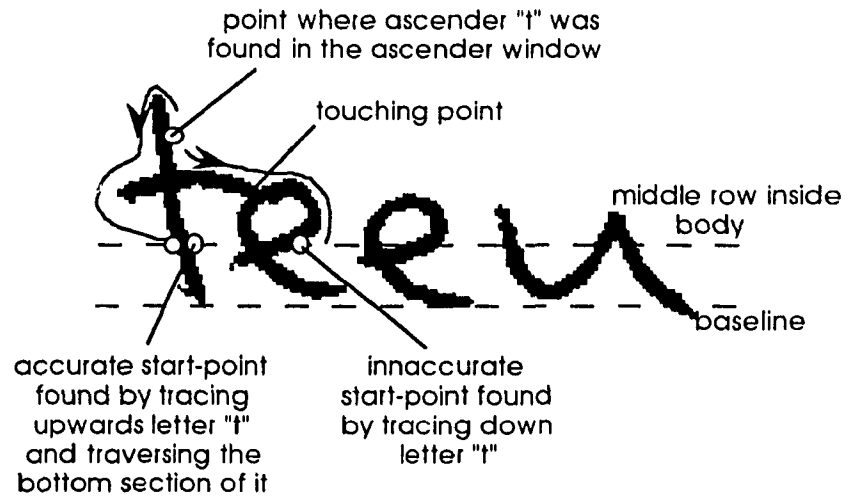
*endAlgorithm*

For words having feature item #5 in Figure 3.21 ("twelve", and all words ending with "teen"), the first word segment to be sent to the neural net is the one immediately to the right of the feature found; in other words, the word segment will either be "ve" or "een". Finding the start-point will not consist of tracing down the found letter (either a "t" as "sixteen" or a "l" in "twelve") along its left edge, as this can often generate inaccurate results for words ending with "teen" due to the presence of the horizontal bar in the "t" which often stretches to touch

the top of letter "e" to its right (see Figure 5.3). Interestingly enough, we observed that the horizontal bar in letter "t" would usually stretch out more on the right side of the "t" than on the left side (Figure 5.3 illustrates this). For this reason, we chose to trace the letter in the same way as described in the previous paragraph, namely, upwards from where it was found, in order to trace down its more reliable left side. Tracing stops when the middle row inside the body is reached. From this point, we scan horizontally towards the right until we find a white pixel indicating the presence of a white space separating letter "t" from letter "e". At this point, if the distance travelled to the right, to find the white pixel, is less than a threshold value of 18 pixels, the white pixel is selected as start-point (Figure 5.3). If the distance, however, is greater than 18 pixels, the "t" and the "e" are probably touching, leaving no white space between them. In this extreme situation, the pixel where the threshold value was reached during the last scan, is selected as start-point of the word segment. The end-point will simply be the end of the word. The following pseudo-code outlines the main steps discussed: finding the start-point and end-point for the word segment to the right of feature item #5 in Figure 3.21.

*Algorithm (2) Find\_start\_end\_points;*

- [1] end\_point = end of word*
- [2] feature\_pixel = pixel inside the ascenders window where item #5 in Figure 3.21 was found.*
- [3] starting from feature\_pixel, trace upwards the right edge of the feature item #5. Stop when the trace goes down the left edge and down to the middle row inside the body. Call this point P1.*
- [4] From P1, scan towards the end of the word on the same row. Stop when a white pixel is found or when distance between P1 and the current pixel being scanned is greater than 18 pixels.*



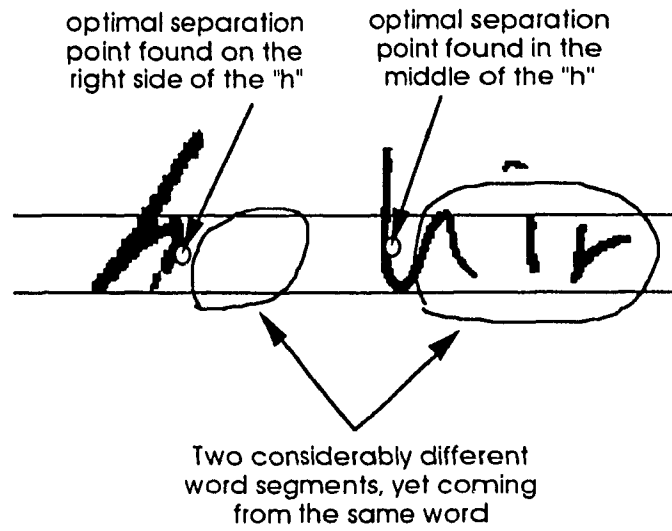
**Figure 5.3:** Tracing the right side vs tracing the left side of the "t", to find the word segment start point.

[5] *start\_point = last pixel scanned in step [4]. (see Figure 5.3)*

*endAlgorithm*

Most words ending with "teen", have a second segment to be sent to the neural network. This second word segment will have its end-point to the immediate left of the "t" in "...teen". The algorithm to find this end-point is identical to the one used on words ending with "ty". The algorithm to find the start-point is also identical to the one used on words ending with "ty". For all other words, such as "Seven", "Nine", and "Six", the start point is the separation point between the notable letter and the rest of the word. The end point will simply be the end of the word.

*Note regarding notable letters "th" as in "thirty"*: Let us recall the point made previously in section 4.2 when discussing output patterns, regarding the two output patterns for each word "thirty" (or "thirteen") and "three". The reason for having two almost identical segments per word is the way the optimal separation point is found in notable letters. As we saw for the symbolic classifier, the optimal separation point in words starting with "th" is on the middle row inside the body, on the right edge of letter "h" (see Figure 5.4). Depending on the writer, however, the optimal separation point could be found to the right of the "h" or in the middle of it (see Figure 5.4). From the neural network's point of view, the two resulting word segments are considerably different, and learning from them as one single output pattern becomes difficult and sometimes yielding unacceptable performance. Therefore the two types of segments were put in separate classes to maximize consistency in the learning set of patterns. At recognition time, both output patterns "ree" and "hree" will translate into "three", "ir" and "hir" into "thir".



**Figure 5.4:** Different segments for the word "thirty" sent to the neural classifier.

**5.2.2 Size-normalization:** The same normalization algorithm discussed in the previous chapter (algorithm *Normalize* in section 4.5.2) is used for size-normalizing word segments from word category 3.

### **5.3 Classifying word segments and patching-up the final word**

Word segments from category 3 are fed through the neural network in the same way as for word category 2. The difference is that, since words from category 3 are recognized only partly by the neural network, the answers from the neural network will not immediately yield the answer the entire classification of the unknown word. All pieces collected from the symbolic and neural classifiers must be put together to make up the final word.

*Patching up words "twelve", "eleven" and those ending with "teen":* In some cases, these words can all share the same feature, namely, an ascender between the middle and three fourths the length of the word. Therefore, when this feature is found, we will first send to the neural network the word segment to the right of their common feature. Thus the three words segments are "een", "ve", and "even". If the output pattern sent back from the neural network is "ve" and the notable letter is in a notable category containing a "t", the word is recognized as "twelve". If the output pattern received is "even" and the notable letter is in notable category 3, 5, or 9, the word is recognized as "eleven". And finally if the output pattern received is "een", then the following words can be recognized immediately:

- "fifteen": The algorithm is similar to the one used for "fifty":

*Algorithm fifteen;*

*if no descender at end of word and*

*if ascender towards the end of the word <table item 5>*

```

    if descender at beginning of word <table item 8>
      if descender around middle of word <table item 2>
        return "fifteen"

```

*endAlgorithm*

- "eighteen": The algorithm is similar to the one used for "eighty":

*Algorithm eighteen;*

```

    if no descender at end of word
      if ascender towards end of word <table item 5>
        if (descender around middle of word <table item 2>)
          if (no ascenders at the beginning of the word)
            if (length/body-height ratio < LONG_WORD) and
              (no other descenders)
              return "eighteen"

```

*endAlgorithm*

- "fourteen": The "f" must be lowercased. Again, the algorithm is similar to the one used for "forty". If the "f" is uppercased, the word must send another segment to the neural network. Below is the algorithm for recognizing the word "fourteen" with a lowercase "f":

*Algorithm fourteen;*

```

    if no descender at end of word
      if ascender towards the end of the word <table item 5>
        if descender at beginning of word <table item 8> and
          no other descenders
          return "fourteen"

```

*endAlgorithm*

All other words ending with "teen" require a second word segment to be sent to the neural network. We therefore prepare the new input to the neural net by fetching the word segment to the left of the common feature mentioned before. The following pseudo-code illustrates the patch-up process.

### *Algorithm Patch\_Up;*

```
if WordEnd = last 2 or 3 letters of word = "ty" or "teen"  
final word is the concatenation of the notable letter (if any), the  
output-pattern name and "ty".  
ex: "S" + "even" + "ty" = "Seventy", "th" + "ir" + "teen" = "thirteen"  
elsif WordEnd = "other" = neither "ty" nor "teen"  
if output pattern returned is "even"  
if (word has no notable letter) or (notable letter is "S")  
return "seven"  
if notable letter is from notable category 3, 5, or 9  
return "eleven"  
elsif output pattern returned is "ine" or "ive"  
if there is a descender at the beginning of the word <table item 8>  
return "five"  
elseif output pattern is "ine"  
return "Nine"  
else return "five"  
elsif output pattern returned is "ix" or "six"  
return "six"  
elsif output pattern returned is "en" return "ten"  
elsif output pattern returned is "wo" return "two"  
elsif output pattern returned is "hree" or "ree" return "three"  
elsif output pattern returned is "one" return "one"  
elsif output pattern returned is "our" return "four"  
elsif output pattern returned is "nine" return "nine"
```

*endAlgorithm*

## **5.4 Setting an output mask**

When the input matrix to the neural network is ready, it is important to guide the neural net's output decision based on the features found by the symbolic classifier. A single 32-bit integer number is used for fast masking; each output segment is given a unique bit position in the 32-bit number. Bits set to 0 eliminate the corresponding output patterns from the possible outputs of the neural



network. Therefore, when choosing the highest output values from the output neurodes, the neural network will ignore neurodes corresponding to those output patterns having their bit set to 0.

The following pseudo-code is the best way to see how the masking is done. Two parameters are given to the masking routine:

- The word termination (*WordEnd*): "teen", "ty", or "other"
- The notable letter at the beginning of the word. If there isn't one, "no Notable\_Letter" is given.

Before sending the word segment to the neural network, the 32-bit mask is set to 0 to disable all output patterns' bits. When reading the pseudo-code, you may want to refer to the list of output patterns given in section 4.2. This list will remind you of which segment belongs to which word.

*WordEnd* = ending of the word, e.g. it could be "ty", "teen", or "other" if it's not "ty" nor "teen".

*Algorithm Mask(Notable\_Letter, WordEnd)*

*case Notable\_Letter is in*

*notable category 8: if WordEnd = "other"*  
*bits are set to "1" for output patterns "wo", "en",*  
*"our", and "ive"*  
*elsif WordEnd = "ty"*  
*bits are set to "1" for output patterns "wen", "or",*  
*and "our"*  
*elsif WordEnd = "teen or lve"*  
*bits are set to "1" for output patterns "our" and*  
*"or"*

*notable category 7: if WordEnd = "other"*  
*bits are set to "1" for output patterns "ine", "ree",*  
*and "hree"*  
*else bits are set to "1" for output patterns "ine", "ir"*

*and "hir"*

*notable category 2: bits are set to "1" for output patterns "ix" and "even"*

*notable category 6: if WordEnd = "other"*

*bits are set to "1" for output patterns "wo" and "en"*

*elsif WordEnd = "ty"*

*bits are set to "1" for output pattern "wen"*

*else (WordEnd = "teen")*

*all bits are set to "0" because no word starts with this notable category and ends with "teen". This word is therefore rejected.*

*notable category 1: bits are set to "1" for output patterns "our" and "ive"*

*note: Words starting with this notable category and ending with ":y" or "teen" are recognized entirely by the symbolic classifier, therefore we ignore those words.*

*notable category 9: if WordEnd = "other"*

*bits are set to "1" for output patterns "ine" and "even"*

*else bits are set to "1" for output pattern "ine"*

*notable category 3: if WordEnd = "other"*

*bits are set to "1" for output pattern "even"*

*else all bits are set to "0" since there are no words in this vocabulary which start with an "E" aside from "eleven". The word is therefore rejected.*

*notable category 5: if WordEnd = "other"*

*bits are set to "1" for output patterns "ree", "hree" and "even"*

*else bits are set to "1" for output patterns "ir" and "hir"*

```
no Notable_Letter:  if WordEnd = "ty" or "teen"  
                    bits are set to "1" for output patterns "six",  
                    "seven", and "nine"  
                    else bits are set to "1" for output patterns "one", "six",  
                    "seven", and "nine"
```

*endAlgorithm*

For example, given an unknown word, suppose the symbolic classifier recognizes a "ty" (items 3 and 4 from the table in Figure 3.21) at the end of the word, and finds a notable letter in notable category 7 (a capital "N" or a "th" at the beginning of the word). The words which have these features are only "Ninety" and "thirty"; therefore, before sending the word segments to the neural network, the mask bits for word segments "ine", "ir", and "hir" are enabled (set to 1) and the rest are disabled (set to 0). The experimental results from our classification system are given in the next chapter.

# Chapter 6

## Experimental Results

### 6.1 The Grammar checker

A context free grammar for worded amounts was developed to minimize cheque-level misrecognition and reject ambiguous forms of writing alphabetic amounts. In the current implementation, the grammar checker is used to accept cheques which are grammatically correct, and reject all other ones. For example, if we have the following amount: "Ten thousand and twenty dollars", and the second word "thousand" is misrecognized as "twenty" then the cheque is rejected since "Ten twenty..." is assumed to be grammatically incorrect.

The grammar consists of 16 production rules (shown below) and a set of flags. Formally speaking, the grammar G is defined as follows:

**Start symbol:** *S*

**Non-terminal symbols:** {*A,B,C,D,E,E2,F,G,a,b,c,d,e,f,g*}

**Terminal symbols:** {*one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen, hundred, thousand, and, dollars, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, ø\*, eol\*\**}

\* *ø* represents the empty string.

\*\* *eol* represents the "end-of-line" character.

```
S ->  A | B | G
A ->  aC | aD | aF
B ->  bC | bD | bF
```

C -> cE  
 D -> dE  
 E -> E2 A | E2 B | F  
 E2 -> e |  $\emptyset$   
 F -> f | eol  
 G -> gA | gD | gF  
 a -> "one" | "two" | "three" | "four" | "five" | "six" |  
 "seven" | "eight" | "nine"  
 b -> "ten" | "eleven" | "twelve" | "thirteen" | "fourteen" |  
 "fifteen" | "sixteen" | "seventeen" | "eighteen" |  
 "nineteen"  
 c -> "hundred"  
 d -> "thousand"  
 e -> "and"  
 f -> "dollars"  
 g -> "twenty" | "thirty" | "forty" | "fifty" | "sixty" |  
 "seventy" | "eighty" | "ninety"

Some flagging techniques were added to the grammar checker in order to simplify the grammar definition. These flags are:

**flag\_1000** : is TRUE when the token (word) recognized is "thousand"


**flag\_10\_90** : is TRUE when the token (word) recognized is between "ten" and "twenty" or one of "thirty", "forty"... "ninety".


**flag\_100** : is TRUE when the token (word) recognized is "hundred".

The way they these flags work together is whenever *flag\_1000* is set to TRUE, it sets the other flags to FALSE in order to allow any other word to occur next in the worded amount. However, the word "thousand" can only appear once in the worded amount, therefore if *flag\_1000* is TRUE and there is another word "thousand" found in the amount, the grammar will reject the cheque. The word "hundred" can appear once, or twice if after its first instance, the word

"thousand" was found (as in "two hundred thousand four hundred dollars"). Moreover, the word "hundred" should not appear if *flag\_1000* is TRUE and *flag\_10\_90* is TRUE; otherwise we would get amounts such as "one thousand twenty hundred dollars" which is not only odd but also ambiguous. In the table shown in Figure 6.1, we can see the truth table for the three flags; for each combination of truth values, we give the possible words which are accepted.

It should be noted that the above grammar is equivalent to the regular grammar shown in Figure 6.2 (Opatrny, 1995). For the state diagram in the figure, we use the following symbols:

 : A final state.

 : Initial state.

Symbols "a", "h", "t", "1-9", "10-19", "20-90" and "d", stand for "and", "hundred", "thousand", "one...nine", "ten...nineteen", "twenty...ninety", and "dollars" respectively.

**6.1.1 Examples of grammatical forms accepted by the grammar checker:** All common grammatical forms are accepted. There are variations such as in the use of the word "and": some write, for instance, "three thousand and fifty dollars", others might write "three thousand fifty dollars", omitting the word "and". Both forms are acceptable by the grammar checker. The same applies for the word "dollars". There is also a common spelling mistake amongst some writers who tend to confuse the spelling of "fourteen" and "forty", writing sometimes "fourty" and "forteen". Any of these four spellings are accepted as being a common

<b>flag_1000</b>	<b>flag_100</b>	<b>flag_10_90</b>	<b>Words accepted</b>
<i>FALSE</i>	<i>FALSE</i>	<i>FALSE</i>	"one", "two"..."thousand"
<i>FALSE</i>	<i>FALSE</i>	<i>TRUE</i>	"one"..."nine", "hundred", "thousand"
<i>FALSE</i>	<i>TRUE</i>	<i>FALSE</i>	"one"..."ninety", "thousand"
<i>FALSE</i>	<i>TRUE</i>	<i>TRUE</i>	"one"..."nine", "thousand"
<i>TRUE</i>	<i>FALSE</i>	<i>FALSE</i>	"one"..."hundred"
<i>TRUE</i>	<i>FALSE</i>	<i>TRUE</i>	"one"..."nine"
<i>TRUE</i>	<i>TRUE</i>	<i>FALSE</i>	"one"..."ninety"
<i>TRUE</i>	<i>TRUE</i>	<i>TRUE</i>	"one"..."nine"

**Figure 6.1:** Truth table for grammar flags *flag\_10\_90*, *flag\_100*, and *flag\_1000*.

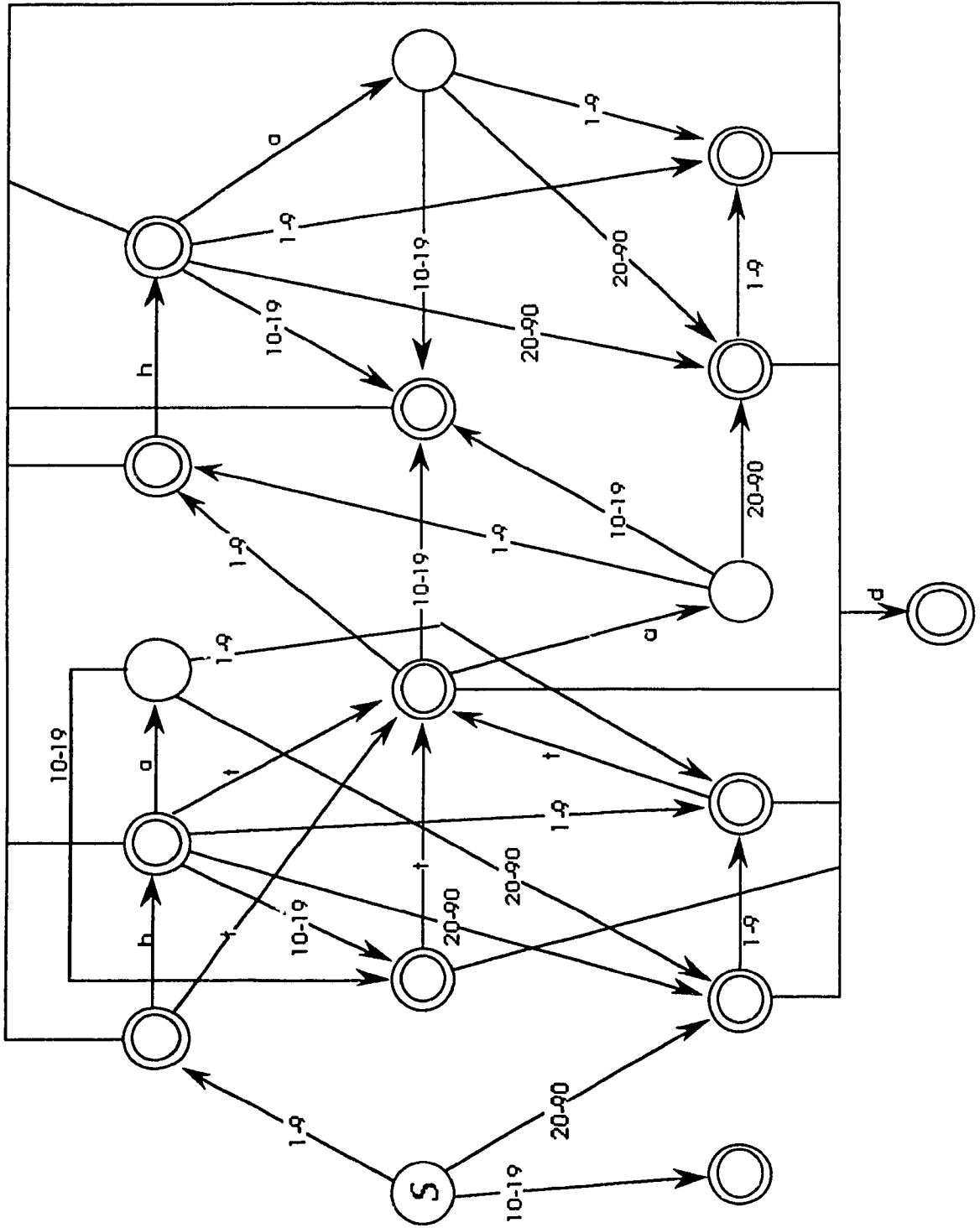


Figure 6.2: State diagram of the regular grammar (Opatrny, 1995).



misspelling. This feature, however, is not directly implemented in the grammar definition nor in the flags, but rather it is dealt with by the classifier which must output properly spelled words. To see how it does this, let us refer back to the masking technique discussed in section 5.4; looking at algorithm *Mask*, we notice that in the case where the notable letter is classified under notable category 8 and the word ends with "ty" or "teen", both segments "or" (from "forty") and "our" (as in "fourteen") are allowed to be selected as output from the neural network. This means that we can have "fourteen", "forteen", "forty", and "fourty".

We have also taken into account two ways of writing amounts which are above 1,000 dollars. For example, to write the amount of \$1,600, most writers would write "*one thousand six hundred dollars*", a less common form would be "*sixteen hundred dollars*". Therefore our grammar checker will accept forms such as "*twenty five hundred and sixty five dollars*", "*thirty hundred*", etc. up to "*ninety nine hundred and ninety nine dollars*". It will reject, however, forms such as "*sixty eight hundred thousand...*", which is not ambiguous mathematically speaking; however, it is too rare to take the risk to accept it. A common form for this amount is "*Six million eight hundred thousand*". So for this last form, the highest number preceding the words "hundred thousand" will be "nine".

**6.1.2 Examples of grammatical forms rejected by the grammar checker:** All odd or ambiguous forms of grammatical structures are rejected, such as "*one fifty dollars*" for \$150, or "*twenty thousand and three thousand dollars*" for \$23,000.

## **6.2 Examples of classification**

**6.2.1 Word category 3 (type 1 word):** Let us look at the example shown in Figure 6.3, when given as input to the neural network, the net's output is shown in Figure 6.4.

**6.2.2 Word category 3 (A different type 1 word example):** When the input shown in Figure 6.5 is given to the neural network, the output is shown in Figure 6.6.

**6.2.3 Word category 3 (type 2 word):** Now, let us look at the example shown in Figure 6.7, where two word segments must be sent to the neural net. The output is shown in Figure 6.8.

**6.2.4 Word category 3 (type 1 word):** This example features a notable letter at the beginning. Figure 6.9 is the input and Figure 6.10 shows the output.

**6.2.5 Word category 2 (words recognized entirely by the neural network):** In the following example, the word has no ascenders nor descenders, the distinguishing characteristic of words in this category. Figure 6.11 is given as input and Figure 6.12 shows the output of the classifiers.

## **6.3 Word level recognition and experimental results**

**6.3.1 Training set:** For our experiments, we used 700 words taken from 60 cheques and 450 individual words, to train the symbolic and neural classifiers. The data was collected from over 80 volunteers from Concordia and McGill University students. For the neural classifier, once the architecture of the neural network was finalized, training it took about one hour on a Sun Sparkstation 2.

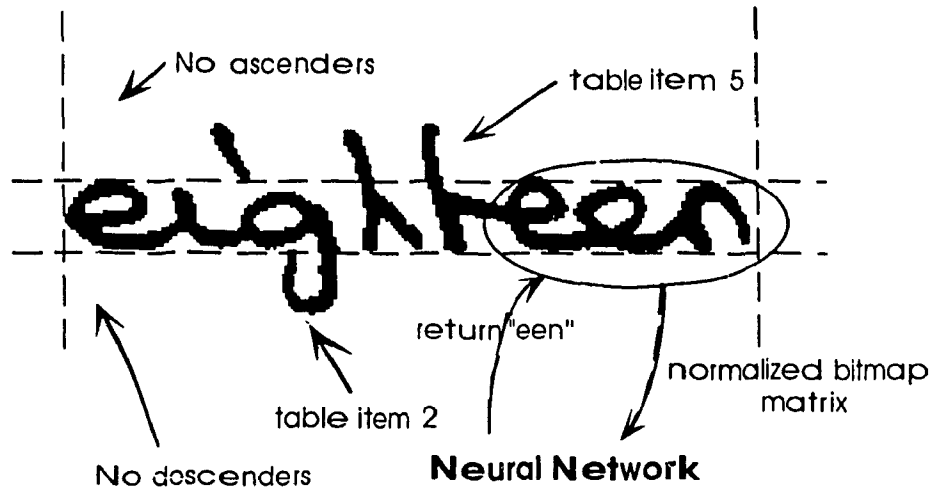


Figure 6.3: Input word yielding output shown in Figure 6.4. Table items are from Figure 3.21.

Word has L/H ratio: 7.935484. Hence it is a "long" word.

Word has a ascender at around 2/3 of its length.

Checking to see if the ascender found could be double "L" as in "dollars"...

Second "L" was not found therefore cannot be "dollars" !  
Sending end-of-word to neural-net, with possible output patterns "ve", "een" and "even"

Input to the neural network is the following bitmap matrix:

```
...11.....1.....  
.1111.....1111.....1.1.....  
.1..1....11.11.....1111.....  
11..1....1..11.....1111.....  
1111....11..1.....11.1.....  
111....1..11.....11..1.....  
1.....1.11....111...1.....  
1.....11....11.1...1.....  
1.....1....111..1...11.....  
11....111....11....1...11.1.....
```

Neural net guess is "een".

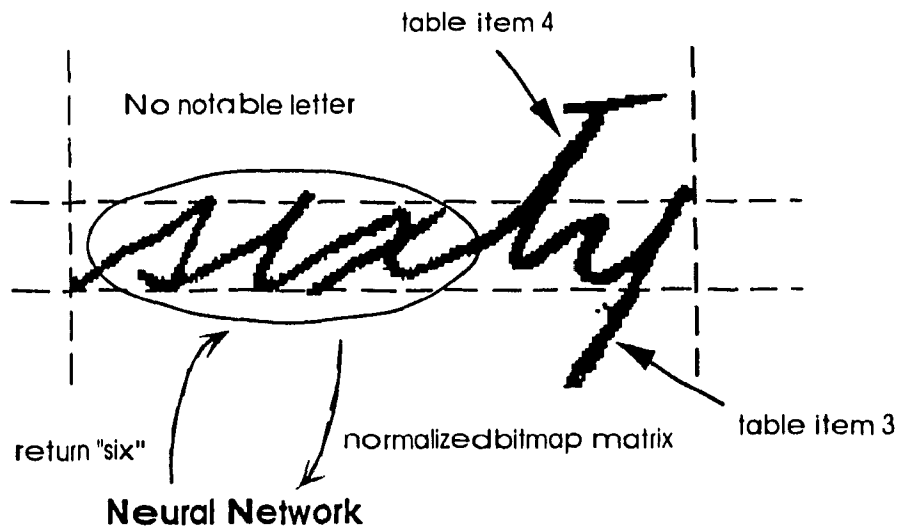
So we now know the word ends with "teen".

Word has a descender at around 1/3 of its length (could be g as in "eighteen" or an "f" as in "fifteen")

First letter has no ascenders nor descenders, therefore

Word was recognized as 'eighteen'

Figure 6.4: Example of neural net output when the input is Figure 6.3.



**Figure 6.5:** Type 1 word of category 3: "sixty" without a notable letter at the beginning. Table items are from Figure 3.21.

```
Potential DESCENDER at row 71, col 203. Dumping local
bitmap...
```

```
00000111111000000000000000000000
00000111111000000000000000000000
00000111100000000000000000000000
00000111100000000000000000000000
00001111100000000000000000000000
00011111100000000000000000000000
```

```
Slope = 2.50. Therefore word has a 'y' at the end
```

```
Potential ASCENDER at row 26, col 186. Dumping local
bitmap...
```

```
00000001111111000000000000000000
00000011111111000000000000000000
00000011111111000000000000000000
00000111111110000000000000000000
00001111111000000000000000000000
00111111110000000000000000000000
```

```
Slope = 1.67.
```

```
...above slope is for 't (as in 'ninety')', in Word
```

```
Looking for a notable letter at the beginning of the
word...None Found!
```

```
Possible output patterns: 'six', 'seven', 'nine',
```

```
Input to the neural network is the following bitmap matrix:
```

```
0000000011110000000001100000000000000000000000000
0000000111100000000011000000000011000000000000000
0000001101000000000110000000001111001100000000000
0000111011000000001100000000011111110000000000000
00011000100000000111000000001110011110000000000000
01100001100000011110000001110000111000000000000000
11000001000001110100000011100011110000000100000000
00000010000111001100001100001110100000011100000000
00000010001100001000011000011100100011110000000000
00100110110000011011000011100001111100000000000000
```

```
Net guessed: 'six'
```

```
Word was recognized as 'sixty'
```

Figure 6.6: Output when input is Figure 6.5.

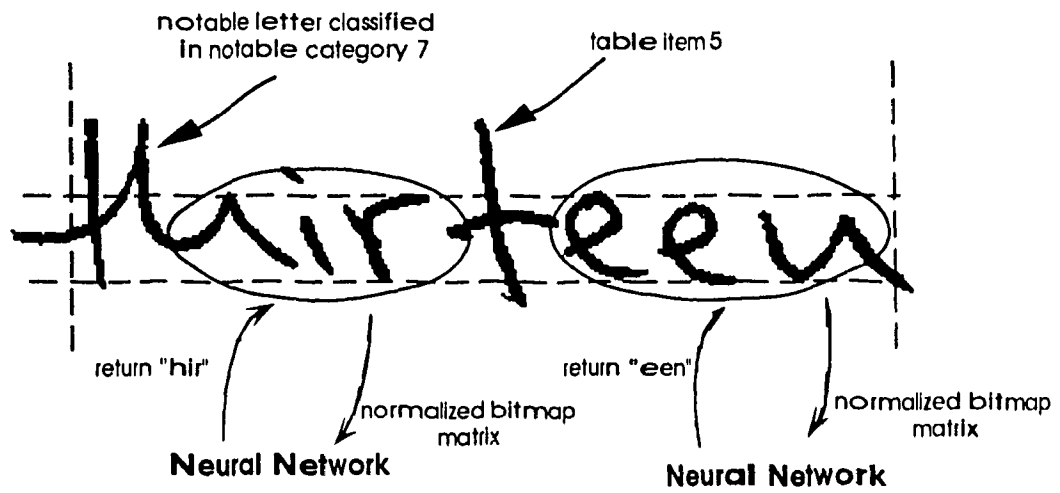


Figure 6.7: Type 2 word of category 3. Table items are from Figure 3.21.





(...continued from previous page)

Notable letter is 'N\_th'.

Possible output patterns: 'ir', 'ine', 'hir',

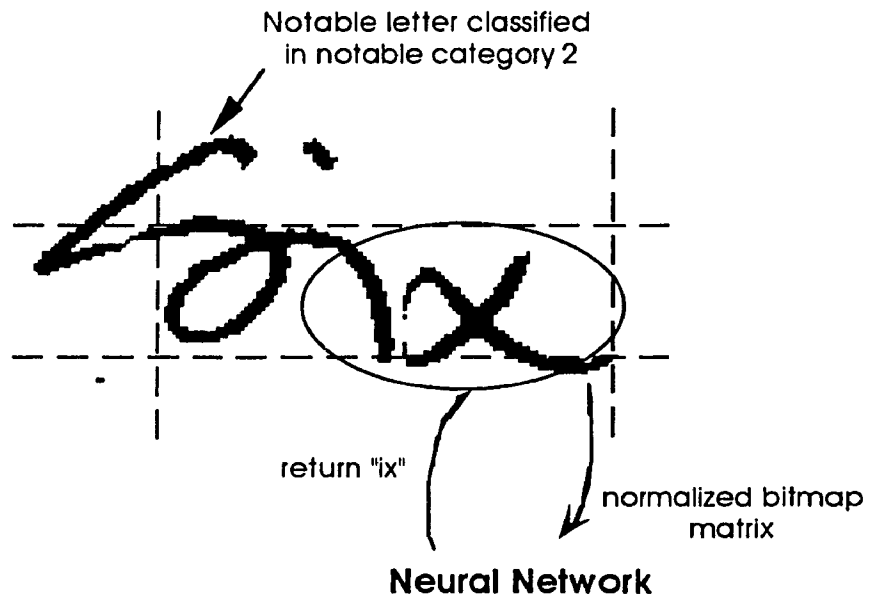
Input to the neural network is the following bitmap matrix:

```
0000000111100000000000010111000000000000000000000
00000001001100000010000011110000000000011100000000
100000110001100000110000011100000000001111100000000
10001110000011000011000000110000000001110000000000
111111000000010000001000000111000000000000000000000
001000000000001100011000000100000000000000000000000
000000000000001100010000001100000000000000000000000
000000000000000001100110000011000000000000000000000
000000000000000000001000000110000000000000000000000
000000000000000000000000000100000011000000000000000000
000000000000000000000000000000110000000000000000000
```

Net guessed: hir.

Word was recognized as 'thirteen'

Figure 6.8: The output when input is Figure 6.7.



**Figure 6.9:** Type 1 word from category 3 with a notable letter at the beginning.



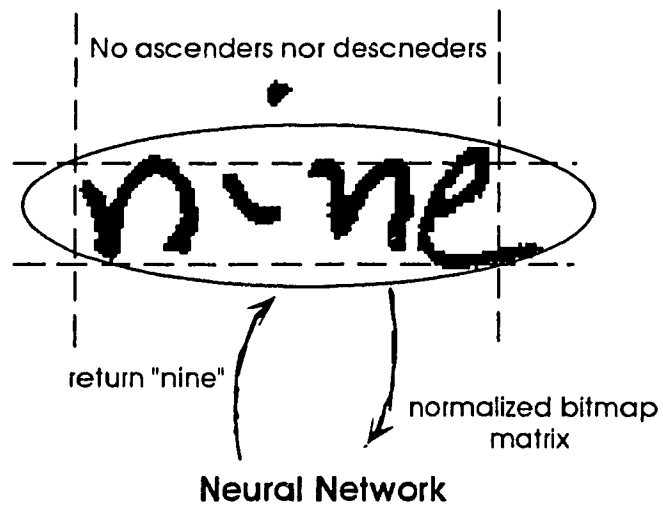


Figure 6.11: Recognizing a word in word category 2.

Word has L/H ratio: 5.333333. Hence it is a "medium" word.

Looking for a notable letter at the beginning of the word...None Found!

Possible output patterns: 'one', 'six', 'seven', 'nine',

Input to the neural network is the following bitmap matrix:

```
00000111011100000110000000011111001111110000110010
10000110000110000110000000000011011100110001100110
11001100000110000111000000000011011000110001001110
11101100000010000011000000000011111000110011011110
11111000000010000011100000000111110000110011110000
01111000000011000001111100000011100000110111100000
01111000000011000000111100000011100001110110000000
01111000000111000000000000000011100001100110000000
00110000000110000000000000000011000001100100000000
011100000011100000000000000000000000001100100000000
```

Net guessed: 'nine'.

Word was recognized as 'nine'

Figure 6.12: The output when input is Figure 6.11.

On this training data, the system delivered a 85% correct word recognition rate, an error rate of 11%, and a rejection rate of 4%. The rejected were misrecognized words which yielded a grammatically incorrect worded amount. Figure 6.12 shows some specimens from the training set.

**6.3.2 Testing set:** We tested on 500 words from over 40 volunteers. All words come from a set of 153 cheques (see Figure 6.13 for examples of such worded amounts) and 100 individual words. On these words, the system delivered 70.47% correct word level recognition rate, a 21.16% error rate, and a 8.37% rejection rate. The testing set and training set are disjoint. Moreover, in order to emphasize even more the writer-independence of the system, the set of writers from the training set is disjoint from the set of writers in the testing set.

For a cheque to be recognized correctly, all words in the cheque must be recognized correctly. At the cheque level, the recognition rate on the testing set is 17%; however, we will see shortly that with the addition of a digit classifier and a resolution strategy, cheque-level results become better.

**6.3.3 Recognition speed and memory requirements:** For a typical 5-word example such as the one shown in Figure 1.2, the system takes approximately 1.5 seconds to recognize all words on a Sun Sparkstation 2. This includes preprocessing time and grammar checking time. The amount of memory required to run the system is of approximately 2.5Mb of RAM. The system was designed to be 100% portable to any computer platform having an ANSI C compiler, making it easy to work with it in different environments. All routines are original code written in C, with the exception of the neural network code which was adapted from a generic neural network C code provided as shareware by Goodman (1993) from the University of Nevada.

One Thousand four hundred and ninety five

sixty dollars

Eight Hundred and eighty one

twenty eight dollars

Seventeen hundred and Forty

Figure 6.13: Training specimens.

One hundred sixty five dollars

Nine thousand nine hundred

two hundred and seventy five

three hundred

**Figure 6.14:** Testing specimens.



## **6.4 Integration with a digit classifier**

In order to apply the system to a cheque level application, experiments were done with the integration of a digit classifier designed by Hum (1995). At the cheque level, both numeric and alphabetic amounts are read and compared; if they match, the cheque is accepted, otherwise, we apply a resolution strategy (will be discussed shortly) to compute alternative classifications. If the resolution strategy fails, the cheque is rejected, otherwise, it is accepted.

**6.4.1 Brief description of the digit classifier:** The symbolic digit classifier extracts features from skeletonized images of the input character. The features extracted from the skeletons are such as the number of holes, upward openings, downward openings, left openings and right openings of an image. If a hole or an opening has been found, its location is also recorded. The number of horizontal, vertical, left, and right diagonal crossings of the image is also extracted as features. The classifier was trained and tested on the CEDAR database (Hull, 1994) of digits. The database consists of totally unconstrained handwritten numerals taken from actual post office envelopes mailed in the Buffalo, New York area. As such, there were no restrictions in the way the digits were written. Hum reported a correct recognition rate of 93% on segmented digits.

**6.4.2 The resolution strategy:** The strategy uses statistical information from the confusion matrices of both word and digit classifiers to try to resolve mismatches between the two classifiers. A mismatch is detected when the number returned by the word classifier (corresponding to the worded amount), does not match the

number returned by the digit classifier (corresponding to the digit amount). We will try to resolve only those mismatches that have the following conditions:

- The number of digits in the amount returned by the word classifier must be the same as the one returned by the digit classifier. For example "1647" and "245" do not have the same number of digits, therefore the condition is not satisfied.
- There is only one mismatched digit from the two numbers. For example, "658" and "659" only differ on the third digit, therefore they satisfy the condition. "750" and "860", however, do not satisfy the condition.

The reason for imposing conditions is to resolve only mismatches which were close, leaving out the riskier cases which could lead to accepting a faulty cheque. For this reason, the mismatches which satisfy the above conditions will be called "reliable mismatches", and those who do not satisfy them are called "unreliable mismatches". In the table shown in Figure 6.15, we can see other examples of reliable and unreliable mismatches.

Two confusion matrices were generated from the training sets for both digit and word classifiers (see Figure 6.16). The digit classifier's matrix can be seen as a 1-dimensional array of size 10 (for digits 0 to 9). Each slot in the array contains the highest confusion digit corresponding to the digit represented by the slot index. In other words, if slot #1 = "2", this means that digit "1" is most often confused with digit "2". In general terms, slot # $n$  =  $m$  means that digit " $n$ " is most often confused with digit " $m$ ". The word classifier's confusion matrix can be seen as a 2-dimensional array (10 rows for digits 0 to 9, and 2 columns). The first column contains the list of the highest confusion digits, and the second column contains

the list of the second highest confusion digits. For instance, word "four" is most often confused with word "five", but it is also commonly confused with word "two". It is important to point out that the word classifier's confusion matrix does not have a highest confusion digit for every word in the vocabulary; the reason is that some words such as "and" and "dollars" do not translate into any number. Moreover, other words such as "hundred", "fifty", and "seventeen" are confused most often with words (such as "twenty", "seventy", "twelve") which cannot be translated into a single digit, as required for our resolution strategy. The reason for having two choices for the word classifier is due to its less accurate distinction between word classes, compared to the digit classifier. We notice from the word classifier's confusion matrix that for words "zero" and "eight", no alternative words are given. This is because the word "zero" is never present on a cheque, and for "eight" the highest and second highest confusion words are "and" and "thousand" respectively, which do not translate into a single digit.

Since the digit classifier has a better recognition rate than the word classifier, we start by assuming that in a mismatch, the word classifier has the wrong answer. When the mismatched digit is located, we check in the word classifier's confusion matrix if any of the alternative choices (highest confusion digit and second highest confusion digit) for its erroneous digit matches the digit found by the digit classifier (step [6] in algorithm *Resolve\_Mismatch* shown below). If a match is found then the mismatch is resolved and the cheque is accepted. If no match is found, however, we take the reverse situation where we assume it is the digit classifier which misrecognized the mismatched digit. In this case, we check in the digit classifier's confusion matrix if the highest confusion digit for its

Reliable mismatches		Unreliable mismatches	
Worded amount	Digit amount	Worded amount	Digit amount
"1629'	"1529"	"157"	"258"
"342"	"442"	"157"	"15"

**Figure 6.15:** Other examples of reliable and unreliable mismatches.

(a)

*Digit classifier's confusion matrix*

Slot #	Highest confusion digit
0	2
1	7
2	6
3	7
4	9
5	6
6	2
7	5
8	2
9	5

(b)

*Word classifier's confusion matrix*

Slot #	Highest confusion digit	Second highest confusion digit
0 ("zero")	<i>none</i>	<i>none</i>
1 ("one")	6	9
2 ("two")	4	5
3 ("three")	9	2
4 ("four")	5	2
5 ("five")	4	2
6 ("six")	5	7
7 ("seven")	6	9
8 ("eight")	<i>none</i>	<i>none</i>
9 ("nine")	7	3

**Figure 6.16:** Confusion matrices for the digit and word classifiers.

erroneous digit matches the digit found by the word classifier (step [7]). If a match is found, then the mismatch is resolved and the cheque is accepted; otherwise, the cheque is rejected. The following pseudo code outlines the main steps in the resolution strategy.

**Algorithm Resolve\_Mismatch;**

- [1] *digit\_number* = number returned by the digit classifier corresponding to the digit amount.
  - [2] *words\_number* = number returned by the digit classifier corresponding to the worded amount.
  - [3] *Verify both conditions for mismatch reliability:*  
*if (length(digit\_number) ≠ length(words\_number) ) or*  
*(digit\_number and words\_number differ by more than one digit)*  
*return CHEQUE\_REJECTED*
  - [4] *digits\_mismatch* = mismatched digit in number returned by the digit classifier
  - [5] *words\_mismatch* = mismatched digit in number returned by word classifier.
  - [6] *Assume the word classifier is wrong, check for alternative digits in the word classifier's confusion matrix:*  
*if (words\_confusion\_matrix(row\_1, words\_mismatch) = digit\_mismatch) or*  
*(words\_confusion\_matrix(row\_2, words\_mismatch) = digit\_mismatch)*  
*return CHEQUE\_ACCEPTED*
  - [7] *Assume the digit classifier is wrong, check for alternative digits in the digit classifier's confusion matrix:*  
*if digits\_confusion\_matrix(digits\_mismatch) = words\_mismatch*  
*return CHEQUE\_ACCEPTED*
  - [8] *return CHEQUE\_REJECTED*
- endAlgorithm**

Despite the efforts put into this strategy, there is one flaw which will need to be addressed in the future in case of a commercial application. For example if, by distraction, a person wrote 5,000 in the digit amount and "Four thousand" in the worded amount, the resolution strategy would resolve the conflict by assuming the word classifier was wrong and using the fact that "Five" is indeed often

confused with "Four" (according to the word classifier's confusion matrix). The cheque is therefore accepted where it should have been rejected for inconsistency between the digit and worded amounts. Although this situation is quite rare in real applications, the potential danger exists and should be dealt with prior to a commercial use.

**6.4.3 Results of integration:** We have tested the integrated system with 153 cheques and obtained a 29% correct cheque level recognition rate, and 71% rejection rate. It is important to differentiate the notion of rejection at the cheque level from the one at the word level; at the cheque level, a specimen is rejected if any of the words in the worded amount yields a grammatical error, or if the resolution strategy discussed in section 6.4.2 fails. Therefore it is much more difficult to obtain high recognition rates at the cheque level than it is at the word level. The digit classifier takes on average 63ms to classify one digit (on a Sun Spark 2), including a preprocessing stage involving thinning and smoothing. Therefore we can neglect the time taken by the digit classifier as it is insignificant next the word classifier's 1.5 seconds (average) per 5-word cheque. The time complexity of the resolution strategy is fairly small and can also be neglected for the overall performance of the system. Thus, the average classification time for a 5-word cheque with 3 or 4 digits in the numeric amount, is 1.5 seconds on a Sun 2. Figures 6.17 and 6.18 show examples of outputs generated from the integrated system, featuring the resolution strategy and the numbers returned by the digit and word classifiers. The results found in this chapter are gathered in the table shown in Figure 6.19.

**Bank of x**

\$ 500

Pay to the order of \_\_\_\_\_

The amount of Five hundred dollars

*Output from the system:*

~~~~~

...

Worded amount: 400. Digit amount: 500.

Digit #1 is different.

Highest confusion digit for "4" in word classifier's confusion matrix matches "5"

Cheque accepted. Amount recognized as "500".

~~~~~

**Figure 6.17:** Example of mismatch resolution when the word classifier is assumed to be wrong.



**Bank of x**

\$ 79

Pay to the order of \_\_\_\_\_

The amount of Seventy Nine \_\_\_\_\_

*Output from the system:*

~~~~~

...

Worded amount: 79. Digit amount: 74.  
 Digit #2 is different.  
 Highest confusion digit for "4" in digit classifier's confusion matrix matches "9"  
 Cheque accepted. Amount recognized as "79"

~~~~~

**Figure 6.18:** Example of a mismatch resolution when the digit classifier is assumed to be wrong.

	Without the Digit Classifier			With the Digit Classifier
	Word Level		Cheque Level	Cheque Level
	Training	Testing	Testing	Testing
Number of individual words	450	0	0	0
Number of words from cheques	250	500	500	500
Correct recognition rate	85%	70.47%	17%	29%
Error rate	11%	21.16%	73%	0%
Rejection rate	4%	8.37%	10%	71%

Figure 6.19: Table showing the recognition rates on the training and testing sets.

## Chapter 7

### Concluding Remarks

Our system to read the handwritten worded amount on a cheque is a hybrid classifier combining symbolic and neural approaches. The symbolic routines take care of extracting features outside the body of words. The neural classifier is used to complete the recognition by reading word segments within the body of words. A grammar checker was added to optimize the reliability of the system at the cheque level, as well as to provide further context information for future improvements of the classification scheme. All routines were designed and written from scratch except for the basic neural network routines, which were obtained from public domain software.

The main strength of this approach is the specialization of each classifier. The symbolic classifier concentrates on features which tend to be consistent from one sample to another, whereas the neural classifier concentrates on the more variable parts of the word which are usually found inside the body.

One of the weaknesses of the system is the classification of notable letters: We defined 9 categories for notable letters, when there are only 6 possible notable letters at the beginning of a word ("T" as in "Ten", "F" as in "Four", "S" as in "Seven", "N" in "Nine", "O" in "One", and "E" in "Eleven"). As discussed in Chapter 3, this is because some notable letters can appear in more than one notable category. The ideal would be to have one notable letter per category; however, the diversity in writing styles is so large, specially in notable letters, that trying to reduce the 9 categories to 6 will be very challenging. The

second weakness is the limited context information provided by the grammar checker; since our thesis focused mainly on word-level recognition, the grammar checker was used to simply accept or reject worded amounts. For the cheque level implementation the grammar should have been used dynamically during the recognition of the worded amount, giving for each unknown word in the worded amount, the list of the possible words which it can be classified as, according to the grammar. The reason for the absence of this feature is because cheque level recognition is a much larger problem than word-level recognition and is beyond the scope of this thesis; implementing this feature would have required substantially more coding as the recognition process would have been modified to dynamically change its decision tree as words are being classified. Some improvements to the overall performance of the system may be achieved by adding the following routines:

- Salt and pepper removal could improve the performance of the symbolic classifier.
- Baseline adjustment for words far above or below the baseline,
- Body-ceiling calculated for individual words, instead of having a single body-ceiling for all words in the text image.
- Slant correction in order to optimize symbolic feature extraction and add consistency to the training and testing patterns for the neural network.

## **Suggestions for future research**

Throughout the development of this system, there were many unexpected changes which ultimately led to this final version. Although some of the design decisions were risky due to their unknown outcome, fortunately none led to dead ends. However, having spent many hours studying the human writing

process and the possible ways of reading it by a machine, we would like to give our suggestions for future research in this area.

In a reduced vocabulary such as the one for banking cheques, contextual information is just as important as the classifier; this is especially true for unconstrained handwriting, where sometimes entire words are so deformed that they cannot be recognized unless put in context with the rest of the text. Therefore, prior to the detailed design of a classification scheme, extensive research and development should be done on tools for extracting contextual information, such as Automated Transition Networks, semantic and pragmatic analyzers, and dictionaries (Shinghal, 1992). Context information can be extracted and used at different levels:

- **Word-level:** For example, in our thesis, extracting features from notable letters to reduce the number of possible outcomes from the neural network.
- **Sentence-level:** When recognizing a word, a grammar (or ATN) could provide conditional probabilities for what the current word could be. A grammar could also be used to check the validity of each word in the sentence, rejecting those sentences which are ambiguous or grammatically incorrect.

For future research specifically in the recognition of bank cheques worded amounts, we would give importance to notable letters (specially capital letters) and all letters having ascenders or descenders as they provide valuable information which can be isolated and retrieved more easily than features inside the body of words. It is important to study the features to be extracted and rank them in order of heuristic reliability, putting at the top of the list the features which tend to be more consistent (less distorted) amongst writers in general. The bottom of the list would then consist of less reliable features.

## References

1. Barrière, C. and Plamondon, R. (1992). Recognizing Sequences of Letters in Mixed-Script Handwriting. In *Proceedings of the Vision Interface Conference*, Vancouver, B. C., pp. 83-91.
2. Bichsel, M. and Seitz, P. (1992). Object Recognition with Optimum Neural Networks. In P. G. J. Lisboa, editor, *Neural Networks: Current Applications*, Chapman & Hall, London, U. K., pp. 163-184.
3. Cohen, E. (1994). Computational Theory for Interpreting Handwritten Text in Constrained Domains. *Artificial Intelligence*, vol. 67, no. 1, pp. 1-31.
4. Fahlman, S. (1988). Faster-Learning Variations on Back-Propagation: An Empirical Study. In *Proceedings of the 1988 Connectionist Models Summer School*, pp. 38-51.
5. Goodman, P. (1993). Neural network code developed by the University of Nevada. Available by ftp, contact [goodman@unr.edu](mailto:goodman@unr.edu).
6. Gordon, B. (1995). Manager of Operations at the Toronto Operations Service Center, Bank of Montreal. Tel: (416) 867-4679, personal communication (attached).

7. Gorsky, N. D. (1994). Experiments with handwriting recognition using holographic representation of line images. *Pattern Recognition Letters*, vol. 15, pp. 853-858.
8. Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1992). Handwritten Digit Recognition with a Back-Propagation Network. In P. G. J. Lisboa, editor, *Neural Networks: Current Applications*, Chapman & Hall, London, U. K., pp. 185-195.
9. Hull, J. (1994). A Database for Handwritten Text Recognition Research. In *IEEE Transactions on Patterns Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550-560.
10. Hum, T. (1995). Graduate student at Concordia University finishing his major report on handwritten numerals recognition. Contact [tonyhum@cs.concordia.ca](mailto:tonyhum@cs.concordia.ca).
11. King, J. (1995) Service and Operations officer for Document Processing at the Royal Bank's Quebec Processing and Operations Center. Tel: (514) 874-3296, personal communication (attached).
12. Nagy, G. (1992). At the Frontiers of OCR. *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1093-1098.
13. Opatrny, J. (1995). Department of Computer Science, Concordia University, Montreal. *Personal communication*.

14. Parizeau, M. and Plamondon, R. (1993). Allograph Adjacency Constraints for Cursive Script Recognition. In *Pre-Proceedings of the Third Workshop on Frontiers in Handwriting Recognition, Buffalo, N. Y.*, pp. 252-261.
15. Plamondon, R. (1991). Development Stages of an Electronic Notepad. In *Proceedings of the First International Conference on Document Analysis and Recognition, Saint-Malo, France*, pp. 361-371.
16. Schürmann, J., Bartneck, N., Bayer, T., Franke, J., Mandler, E., and Oberländer, M. (1992). Document Analysis – From Pixels to Contents. *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1101-1118.
17. Seni, G. and Cohen, E. (1994). External Word Segmentation of Off-Line Handwritten Text Lines. *Pattern Recognition*, vol. 27, no. 4, pp 41-52.
18. Shinghal, R. (1992). *Formal Concepts In Artificial Intelligence*, Chapman & Hall, London, U. K., co-published in the U. S. with Van Nostrand, New York.
19. Simon, J. C. (1992). Off-Line Cursive Word Recognition. *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1150-1160.
20. Srihari, S. N. (1992). High-Performance Reading Machines. *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1120-1131.



21. Sun, R. and Bookman, L. (1993). How Do Symbols and Networks Fit Together: A Report from the Workshop on Integrating Neural and Symbolic Processes. *AI Magazine*, summer 1993, pp. 20-23.
22. Tappert, C. (1984). Adaptive On-Line Handwriting Recognition. In *Proceedings of the 7th International Conference on Pattern Recognition*, Montreal, Canada, July-August 1984, pp. 1004-1007.
23. Zurada, J. M. (1992). *Artificial Neural Systems*, West Publishing Company, St. Paul, MN.

# Appendix