

On-Line Handwriting Recognition Using Hidden Markov Models

by

Han Shu

S. B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology (1996)

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February, 1997

Copyright © 1997 by Han Shu. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author.....
Department of Electrical Engineering and Computer Science
August 15, 1996

Certified by.....
John Makhoul
Chief Scientist, Speech and Language Department, BBN Corp.
Thesis Supervisor

Certified by.....
Victor W. Zue
Senior Research Scientist, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

On-Line Handwriting Recognition Using Hidden Markov Models

by

Han Shu

Submitted to the
Department of Electrical Engineering and Computer Science

February 1, 1997

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

New global information-bearing features improved the modeling of individual letters, thus diminishing the error rate of an HMM-based on-line cursive handwriting recognition system. This system also demonstrated the ability to recognize on-line cursive handwriting in real time. The BYBLOS continuous speech recognition system, a hidden Markov model (HMM) based recognition system, is applied to on-line cursive handwriting recognition. With six original features, delta x , delta y , writing angle, delta writing angle, *PenUp/PenDown* bit, and $sgn(x - \max(x))$, the baseline system obtained a word error rate of 13.8% in a 25K-word lexicon, 86-character set, writer-independent task. Four new groups of features, a vertical height feature, a space feature, hat stroke features, and substroke features, were implemented to improve the characterization of vertical height, inter-word space, and other global information. With the new features, the system obtained a word error rate of 9.1%, a 34% reduction in error. Additionally, the space feature and the substroke features each reduced the word error rate approximately 15%. In addition, we demonstrated real-time, large vocabulary, writer-independent, on-line cursive handwriting recognition without sacrificing much recognition accuracy of the baseline system by implementing minor modifications to the baseline handwriting recognition system. The details of the on-line cursive handwriting recognition system, the feature experiments, and the real-time demonstration system are presented.

Thesis Supervisors: Dr. John Makhoul and Dr. Victor W. Zue

Titles: Chief Scientist, Speech and Language Department, BBN Corp. and
Senior Research Scientist, Dept. of EECS, MIT

Acknowledgments

First of all, I would like to thank my advisors, Dr. John Makhoul, Richard Schwartz, and Dr. Victor Zue for the opportunity to study with them. I have learned a great deal from their six-day weeks, their enthusiasm and tenacity in research, and their ability to explain difficult concepts with clarity.

Second, I would like to thank all of the people who were a part of BBN's Speech and Language Department and HARK Department. Special thanks go to Mike Vande Weghe, George Yu, and Bruce Musicus for introducing me to speech recognition and the groups, and to George Zavaliagkos, Long Nguyen, Chris LaPre, John McDonough, and Francis Kubala for giving me valuable help with the BYBLOS speech recognition system.

I would like to thank all of the people who have contributed to the BYBLOS on-line cursive handwriting recognition project over last four years. Without their effort, this project would not have been possible: to Thad Starner, James Yang, and George Chou for collecting the on-line handwriting data and establishing the baseline system, and to Greg Spurrier, for implementing the Winsock interface on the PC for the real-time cursive handwriting recognition demo.

Special thanks to my best friend Richard Diaz for editing my thesis many times over, sometimes even into the early morning hours. Also, thanks to Eugene Choi, who is a Biology major, but knows more about setting up computers than I do, for his wizardry in solving some very difficult Windows problems. In addition, many other people and other factors¹ have contributed to the completion of this thesis. Here, I would especially like to take this opportunity to thank Trey Reyher, Erik Miller, John McDonough, Jennifer Ng, Chris Sandoval, and Joel Ford.

I also want to thank both my Chinese and American families for the continuing support and unconditional love they have given me. Without them, the last five years of MIT would have been very difficult.

I am most indebted to Prof. Sanjoy Mitter, my present advisor, for being so understanding and patient during the writing phase of this thesis. I would also like to thank Chairman Paul Gray for his constant support and encouragement during my undergraduate years at MIT.

Last of all, I would like to thank Dr. John Makhoul, Richard Schwartz, and Dr. Bruce Musicus for giving me freedom, guidance, and financial support throughout my 6-A Co-op assignments. They have made the experience invaluable.

¹I would like to thank the New England Patriots for not winning the 31st Superbowl. Had they done so, I would have had no other recourse but to celebrate extravagantly, and thus completion of this thesis would have been delayed.

Table of Contents

1. Introduction	11
<i>1.1 Issues of On-line Handwriting Recognition.....</i>	<i>12</i>
1.1.1 Styles of Handwriting: Printed vs. Cursive	12
1.1.2 Writer-Dependent vs. Writer-Independent	13
1.1.3 Closed-Vocabulary vs. Open-Vocabulary	13
1.2 Background	14
1.3 Summary and Research Goals	15
2. Hidden Markov Models for On-line Handwriting Recognition	17
2.1 Model Parameters of Hidden Markov Model	17
2.1.1 Discrete-State Markov Process	17
2.1.2 Hidden Markov Models: An Extension of Discrete-State Markov Processes .	19
2.2 Three Basic HMM Problems.....	20
2.2.1 A Solution to the Evaluation Problem – The Forward Algorithm	21
2.2.2 A Solution to the Decoding Problem – The Viterbi Algorithm	22
2.2.3 A Solution to the Training Problem – The Baum-Welch Algorithm	23
2.3 Using HMMs for On-line Handwriting Recognition	25
2.3.1 Modeling Letters, Words, and Sentences with HMMs	25
2.3.1.1 Modeling Letters	25
2.3.1.2 Modeling Words.....	26
2.3.1.3 Modeling Sentences	27
2.3.2 Recognition of Handwritten Isolated Letters Using HMMs	29
2.3.3 Recognition of Cursively Handwritten Sentences Using HMMs.....	29
2.4 Summary.....	30
3. BBN On-line Cursive Handwriting Data Corpus	31
3.1 Overview	31
3.2 Building BBN On-line Cursive Handwriting Data Corpus	31
3.2.1 Data Collection.....	31
3.2.2 Data Preparation.....	33
3.3 BBN On-line Cursive Handwriting Data Corpus and UNIPEN.....	34
3.4 Training Data Set and Testing Data Set.....	34
4. The Baseline BYBLOS On-line Cursive Handwriting Recognition System.....	36

4.1 Overview	36
4.2 Modules	36
4.2.1 Front-end	36
4.2.1.1 Preprocessing	37
4.2.1.2 Computing Feature Vectors.....	37
4.2.1.3 Calculating Observation Symbols.....	39
4.2.2 Trainer	40
4.2.3 Decoder	41
4.3 Results	44
5. Features Experiments	46
5.1 Introduction.....	46
5.2 The Four Feature Experiments	47
5.2.1 The Vertical Height	47
5.2.2 Space Feature	49
5.2.3 Hat Stroke Features	52
5.2.4 Substroke Features	54
5.3 Summary of Results.....	56
6. Real-time On-line Cursive Handwriting Recognition Demonstration	58
6.1 Introduction.....	58
6.2 Design & Implementation	58
6.2.1 Real-time Data Sampling Module.....	59
6.2.2 Real-time Front-end	61
6.2.3 Real-time Decoder.....	62
6.2.4 Results Module.....	63
6.3 Summary & Results	63
7. Summary & Future Directions	66
7.1 Summary.....	66
7.2 Suggestions for Future Work.....	66
Appendix A: List of 89 Letters of BBN On-line Cursive Handwriting Data Corpus and the BYBLOS Symbols	69
Appendix B: A Sampling of the Vocabulary of the BBN On-line Cursive Handwriting Data Corpus.....	70
Appendix C: A Sampling of the Decoding Results for the Substroke Feature Experiment.....	82

List of Figures

Figure 1-1: Types of English writing styles [1].....	12
Figure 2-1: Illustration of a Markov chain with 3 states (labeled S_1 , S_2 , and S_3).	18
Figure 2-2: Illustration of a hidden Markov model with 3 states.....	19
Figure 2-3: A 7-state HMM for a letter.....	25
Figure 2-4: Illustration of contextual effects on a cursively written "i". The left "i" is written after an "m"; the right "i" is written after a "v".....	26
Figure 2-5: Illustration of an HMM modeling of the word "pen", consists of three 7-state HMMs for the trigraph "[p]e", "p[e]n", and "e[n] ".....	27
Figure 2-6: Illustration of a simple bigram grammar for sentences composed of the words "It", "I", "is", "am", "orange", and "young".....	28
Figure 3-1: Sample handwriting from experimental corpus. (Transcription: Pierre Vinken, 61 years old, owes AT&T \$30,981 by Nov. 21, 1989.).....	31
Figure 3-2: Screen caption of the GRiD™ LCD during a data collection session.....	32
Figure 4-1: Illustration of invisible strokes. The highlighted dash lines are the invisible strokes.....	37
Figure 4-2: The writing angle and the delta writing angle features.....	38
Figure 4-3: Calculation of the $sgn(x-\max(x))$ bit for the word "the" only. The highlighted portions are encoded with 1 and the gray portions are encoded with 0, for the $sgn(x-\max(x))$ bit.....	39
Figure 4-4: Block diagram of the training process.....	44
Figure 4-5: Block diagram of testing process.....	44
Figure 5-1: Definition of the x and y directions.	47
Figure 5-2: Illustration of three zones, lower, middle, and upper zones, of English handwriting.....	48
Figure 5-3: HMMs with <i>OptionalSpace</i> modeling the sentence, "Lassie was rewarded". The dark ellipses represent HMMs modeling a word and the white circles represent	

HMMs modeling an <i>OptionalSpace</i> . Skipping transitions connect the two neighboring words directly.....	50
Figure 5-4: Calculation of the variable <i>gap</i> for the highlighted dash invisible stroke.....	51
Figure 5-5: Scaling function of the variable <i>gap</i> for the space feature.....	51
Figure 5-6: Total word error rates of four feature experiments.....	57
Figure 6-1: Block diagram of the real-time cursive handwriting recognition system.....	59
Figure 6-2: Screen capture of the real-time data sampling module.....	60
Figure 6-3: Block diagram of the real-time front-end.....	61
Figure 6-4: Block diagram of the real-time preprocessing sub-module.....	61
Figure 6-5: Screen capture of the real-time data sampling module and the results module during a real-time handwriting recognition demonstration. The writer wrote “ <i>Intel cut prices less than expected on its most powerful computer chip in a sign that it can flex its.</i> ” On the real-time data sampling module. The current best match is “ <i>Price cut prices less than expected on its most powerful computer chips in a sign that ice . 5 0 in flex its</i> ” on the results module.....	64

List of Tables

Table 1-1: Summary of HMM-based on-line handwriting recognition systems.....	15
Table 3-1: Summary of handwriting data of the six writers.....	34
Table 3-2: Summary of training data set and testing data set.....	35
Table 4-1: Baseline BBN on-line cursive handwriting recognition system.	44
Table 5-1: Results of the feature experiment with vertical height.	49
Table 5-2: Results of the feature experiment with space.	52
Table 5-3: Results of the feature experiment with hat stroke.....	54
Table 5-4: Results of the feature experiment with substrokes.	55
Table 5-5: Summary of total word error rate of all four feature experiments.	56

1. Introduction

Handwriting recognition can be divided by its input method into two categories: off-line handwriting recognition and on-line handwriting recognition. For off-line recognition, the writing is usually captured optically by a scanner. For on-line recognition, a digitizer samples the handwriting to time-sequenced pixels as it is being written. Hence, the on-line handwriting signal contains additional time information which is not presented in the off-line signal. This thesis addresses the problem of on-line handwriting recognition.

On-line handwriting recognition technology has great potential for improving human-machine interaction. For example, without a keyboard, portable computers could be smaller; writing down a mathematical equation by hand would be more natural; and taking notes with a pen-based portable computer would be free of keyboard clicking. These are just a few of the possible benefits. Products including the Newton™ from Apple and personal digital assistants from AT&T and Hewlett Packard have already started to incorporate on-line handwriting recognition technology.

About ten years ago, advances in tablet digitizer technology brought on a wave of interest in the on-line handwriting recognition research arena. However, after ten years, some of the toughest problems of on-line handwriting recognition remain. A few on-line handwriting recognition systems have demonstrated highly accurate recognition results by imposing constraints such as printed writing style, writer-dependence, and small vocabulary size. Each of these issues will be discussed in detail in the next section. These constraints can make on-line handwriting recognition considerably easier, but for on-line handwriting recognition technology to have the full benefits mentioned above, the systems must be independent of these constraints.

The focus in this thesis will be an on-line handwriting recognition system that is mostly free of these constraints. Namely, a system performs cursive, large vocabulary, and writer-independent (WI) on-line handwriting recognition, as well as the ability to perform this task in real time.

1.1 Issues of On-line Handwriting Recognition

1.1.1 Styles of Handwriting: Printed vs. Cursive

The difficulty of handwriting recognition varies greatly with different writing styles. Figure 1-1 illustrates different writing styles in English. The writing style of the first three lines is commonly referred to as printed or discrete handwriting, in which the writer is told to write each character within a bounding box or to separate each character. The writing style of the fourth line is commonly referred to as pure cursive or connected handwriting, in which the writers are told to connect all of the lower case characters within a word. Most people write in a mixed style, a combination of printed and cursive styles, similar to the writing on the fifth line.

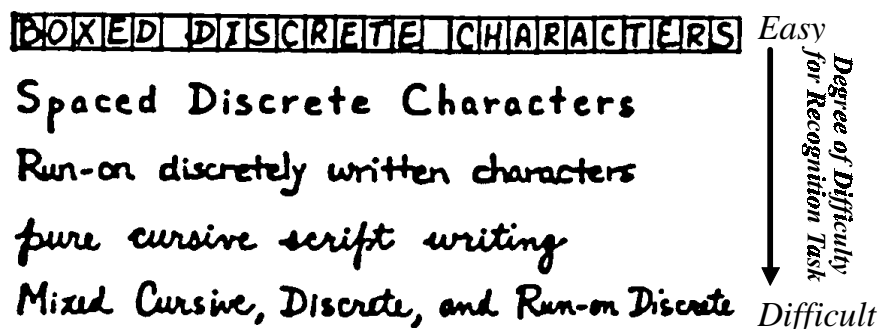


Figure 1-1: Types of English writing styles [1].

Both printed and cursive handwriting recognition are difficult tasks because of the great amount of variability present in the on-line handwriting signal. The variability is present both in time and signal space. Variability in time refers to variation in writing speed, while variability in signal space refers to the shape changes of the individual characters. It is rare to find two identically written characters. The difficulty of recognizing handwriting lies in constructing accurate and robust models to accommodate the variability in time and feature space.

In addition to these two types of variability in time and signal space, cursive handwriting has another type of variability in time which makes this task even more difficult. This additional type of variability is due to the fact that no clear inter-character boundaries (where one character starts or ends) exist. In printed handwriting, a pen-lift defines these boundaries between characters. However, in cursive handwriting the pen-

lift cues simply do not exist. Cursive-style handwriting recognition is more difficult because the recognizer has to perform the error-prone step of character segmentation, either explicitly or implicitly. In this project, only cursive handwriting will be directly investigated.

1.1.2 Writer-Dependent vs. Writer-Independent

A writer-independent (WI) system is capable of recognizing handwriting from users whose writing the system has not seen during training. In general, WI systems are much more difficult to construct than writer-dependent (WD) ones. Humans are capable of WI recognition. However, we are better at WD recognition than WI recognition tasks, *i.e.*, generally we can recognize our own handwriting better than a stranger's handwriting.

The WI systems are more difficult to construct because the variability of handwriting across writers is much greater than the handwriting of a writer. For WD tasks, the system is only required to learn a few handwriting styles. On the other hand, for WI tasks, the system must learn invariant and generalized characteristics of handwriting.

1.1.3 Closed-Vocabulary vs. Open-Vocabulary

Vocabulary is also a major factor in determining how difficult a handwriting recognition task is. *Closed-vocabulary* tasks refer to recognition of words from a predetermined dictionary. The dictionary size is arbitrary. *Open-vocabulary* tasks refer to recognition of any words without the constraint of being in a dictionary. Closed-vocabulary tasks are easier than open-vocabulary ones because only certain sequences of letters are possible when limited by a dictionary.

Closed-vocabulary tasks using a small dictionary are especially easy because: 1) a small vocabulary size can mean a smaller number of confusable word pairs; 2) a small vocabulary size enables the direct modeling of individual words, whereas a large vocabulary size necessitates the modeling of letters, which is due to the computational complexity of modeling words directly; 3) with the usage of letters for large vocabulary tasks, the search space of all possible sentences is usually much larger due to an increase in the number of nodes in the search graph. When letters are used for modeling instead of

words, the number of nodes is $m \times n$ instead of n where n is the number of words, and m is the average number of letters per word (generally between three and ten).

As the vocabulary size increases, the occurrence of *out-of-vocabulary* words is less frequent. Thus, the performance of the large vocabulary tasks is approximately the same as of the performance of the open-vocabulary tasks.

1.2 Background

Since the improvement of tablet digitizer technology about ten years ago, many researchers around the world have developed an interest in on-line handwriting recognition. Initially, research focused on template matching approaches. While these approaches gave reasonable results on printed or discrete handwriting, they had very limited success with cursive handwriting due to difficulty in accurately and reliably locating character boundaries. Tappert gives a complete review of this approach [2].

Recently, several research sites have started to focus on statistical approaches, especially hidden Markov models (HMMs), for on-line handwriting recognition. HMMs' success in the speech recognition domain has motivated the use of HMMs in the on-line handwriting domain. HMMs have proven themselves to be effective mathematical models for characterizing the variance both in time and signal space presented in speech signals [3, 4]. Several research sites have built on-line handwriting recognition system using HMMs. Below is a summary of their work with emphasis on the features used:

- Nag *et al.* [5] at Cambridge University (CU) utilized angle and sharp turning-point indicators to recognize the words for the numbers one to ten. It achieved a 98.5% correct rate.
- Bellegarda *et al.* [6, 7] at IBM used local position, curvature, and global information bearing features to recognize characters from on an 81 character data corpus. It achieved an 18.9% character error rate without grammar in WI mode.
- Starner *et al.* [8] [9] at BBN used angle, delta angle, delta x (x is the horizontal position), delta y (y is the vertical position), pen lifts, and $sgn(x - \max(x))$ features to

recognize words from a 25K word corpus. With a bi-gram grammar, it achieved a 4.2% word error rate in WD mode. In the WI mode, the word error rate was 13.8%¹.

- Schenkel *et al.* [10] at AT&T used x and y coordinates, pen-lifts, speed, direction, and curvature features to recognize words from a 25K word corpus, it achieved an approximately 20% word error rate with a dictionary in WI mode.

Table 1-1: Summary of HMM-based on-line handwriting recognition systems

	Features	WI vs. WD	Character Set Size	Vocabulary Size	Grammar	Character Error (%)	Word Error (%)
CU '86	angle, sharp turning	-	15	10	No	-	1.5 ²
IBM '94	local position, curvature, global information	WI	81	300	No	18.9	-
BBN WD '94	angle, delta angle, delta x, delta y, pen-lifts, $\text{sgn}(x-\max(x))$	WD	89	25,595	bi-gram	1.4	4.2
BBN WI '95	same as above	WI	89	25,595	bi-gram	-	13.8
AT&T '94	x , y , pen-lifts, speed, direction, curvature	WI	52	~25,000	No	11 ²	20 ²

Table 1-1 illustrates the most important features of HMM-based on-line handwriting recognition systems. Bear in mind that the error rates alone do not suffice for comparing systems with each other because these error rates were obtained using very different parameters, such as variable data corpora, grammar usage, *etc.* This comparison of error rates is only valid if the data corpora and other conditions are the same.

1.3 Summary and Research Goals

With the increasing demand for general purpose on-line handwriting recognition technology, research in the area of on-line handwriting recognition has shifted away from the traditional template-matching techniques. More sophisticated techniques have been

¹ The writer-independent result was not reported in [8] [9]. The author of this thesis obtained the writer-independent result for comparison purposes.

² The error rate is commonly referred to as the sum of three separate error components: the insertion error, the deletion error, and the substitution error. Here, the original author reported the substitution error only, so the error percentage should be even larger.

experimentally applied to the harder problems of cursively written, WI, and large vocabulary on-line handwriting recognition.

Much of the recent work in on-line handwriting recognition has been on adapting well-known HMM algorithms in speech recognition to these difficult on-line handwriting problems. However, most of the work has been in feasibility studies. These studies have shown HMMs to be effective in solving these difficult on-line handwriting problems. Little research has been focused on the optimal representation of on-line handwriting, *i.e.*, the kind of features that should be used. In the speech domain, after years of research and experimentation, *Mel-frequency cepstral coefficients* and *delta Mel-frequency cepstral coefficients* have become the *de facto* features of choice. In the handwriting domain, it is not yet clear which features are best.

In this thesis, answers to some of these questions will be explored while focusing on useful features to effectively adapt the modeling and decoding techniques of an HMM-based pattern recognizer. We will also attempt to build a real-time handwriting recognition system. It is only through a real-time system that a potential user can effectively evaluate how usable this on-line technology really is.

In Chapter 2, a clear definition of hidden Markov models will help to explain how they are used for on-line handwriting recognition. For the feature experiment, we will describe the on-line handwriting data corpus in Chapter 3, the baseline on-line handwriting recognition system in Chapter 4, and the details of the feature experiments in Chapter 5. In Chapter 6, the real-time on-line cursive handwriting recognition demonstration system is described. Chapter 7 draws conclusions and suggests directions for future work.

2. Hidden Markov Models for On-line Handwriting Recognition

Hidden Markov Models (HMMs) were initially developed in the 1960's by Baum and Eagon at the Institute for Defense Analyses (IDA) [11-13]. In the 1970's, Baker at Carnegie-Mellon University (CMU) [14], Jelinek at IBM [15], and other applied HMMs to the problem of speech recognition. In 1980, IDA invited a number of research organizations in speech recognition, among them were AT&T and BBN, for a workshop on HMMs. In the mid 1980's, several HMM-based speech recognition systems from AT&T, BBN, and CMU showed superior results [16-18]. The success of these systems dramatically increased interest in applying HMMs to continuous speech recognition and other difficult pattern recognition problems such as handwriting recognition.

There are two types of HMMs classified by their observation densities: discrete-density HMMs and continuous-density HMMs. For simplicity, the discussion here will be limited to discrete-density HMMs. A more detailed explanation of HMMs can be found in [4, 18, 19].

2.1 Model Parameters of Hidden Markov Model¹

Hidden Markov models (HMMs) can be viewed as extensions of discrete-state Markov processes. To fully understand HMMs, a review of the discrete-state Markov process is necessary, as well as an explicit definition of the extension.

2.1.1 Discrete-State Markov Process

A *Markov process* is a stochastic process whose future behavior depends only on its present state, not on the past, *i.e.*, it satisfies the *Markov condition*. A *discrete-state Markov process* can be in one of a set of N distinct discrete states, S_1, S_2, \dots, S_N at any given time. In Figure 2-1, the number N of distinct discrete states is 3. Let Q_n denote the process state at time n . The probability of the process being in state S_i at time n is

¹ The material presented in this section is based on [20], [4], and [18].

denoted by $P(Q_n=S_i)$. Specifically, the *Markov condition* (or the *state-independence assumption*) states:

$$P(Q_n=S_i|Q_{n-1}=S_j, Q_{n-2}=S_a, \dots, Q_0=S_b) = P(Q_n=S_i|Q_{n-1}=S_j), \quad \forall i, j, a, b, \text{ and } n. \quad (2.1)$$

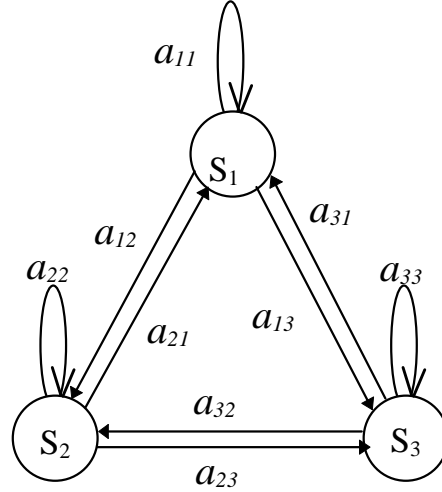


Figure 2-1: Illustration of a Markov chain with 3 states (labeled S_1 , S_2 , and S_3).

Since a discrete-state Markov process satisfies the Markov condition, the *initial state probabilities* and the *state transition probabilities* together characterize the process completely. The initial state probabilities are denoted $\Pi = \{\pi_i\}$, where:

$$\pi_i = P(Q_0 = S_i), \quad 1 \leq i \leq N. \quad (2.2)$$

The state transition probabilities are denoted $A = \{a_{ij}\}$, where:

$$a_{ij} = P(Q_n = S_j | Q_{n-1} = S_i), \quad 1 \leq i, j \leq N, \text{ and} \quad (2.3)$$

$$\sum_j a_{ij} = 1, \quad \forall i. \quad (2.4)$$

In Figure 2-1,

$$A = \{a_{ij}\} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \quad \text{and} \quad (2.5)$$

$$\Pi = \{\pi_k\} = [\pi_1 \quad \pi_2 \quad \pi_3]. \quad (2.6)$$

The duple, $\{A, \Pi\}$, completely parameterizes a discrete-state Markov process. For a more detailed treatment, refer to [21].

2.1.2 Hidden Markov Models: An Extension of Discrete-State Markov Processes

Each state of a discrete-state Markov process can be associated with a deterministic observation, *i.e.*, the symbol O_i is always observed when the process is in the state i . However, for most speech or handwriting recognition applications, the constraint of a deterministic observation for each state in the model is too restrictive. When this constraint is eliminated by allowing the observation within each state to be probabilistic, we obtain a *hidden Markov model*. In this extended model, the observation sequence does not have a corresponding deterministic state sequence. In general, there are many possible state sequences which generate an observation sequence. Hence, the state sequence is *hidden*.

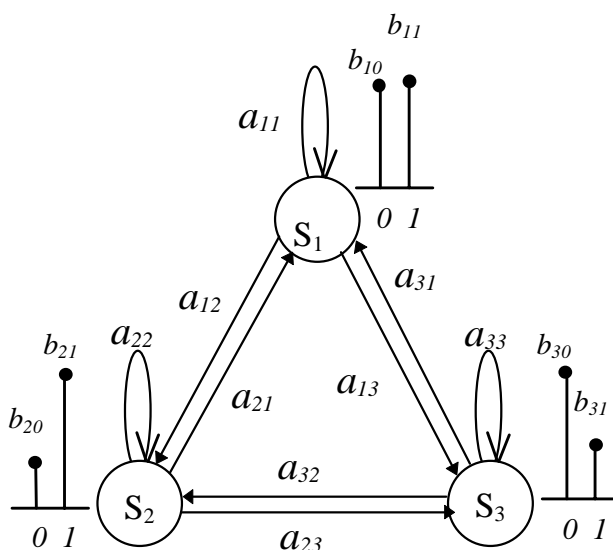


Figure 2-2: Illustration of a hidden Markov model with 3 states.

Now, a formal definition for HMMs will be given. Let M be the number of distinct observation symbols in each state. Let O_n be the observation at time n . An event for which the observation symbol is k is denoted by v_k . The *state observation probabilities* are $B = \{b_{iv_k}\}$, where

$$b_{iv_k} = P(O_n = v_k | Q_n = S_i), \quad 1 \leq i \leq N \text{ and } 1 \leq k \leq M. \quad (2.7)$$

Since an HMM satisfies an additional *output independence assumption*, we have:

$$P(O_n=v_k/O_{n-1}=v_a, O_{n-2}=v_b, \dots, O_0=v_c, Q_n=S_k) = P(O_n=v_k/Q_n=S_k), \quad \forall k, a, b, c, \text{ and } n. \quad (2.8)$$

The triple, $\{A, B, \Pi\}$, is a complete parameter set of an HMM. Let λ denote this triple, *i.e.*, $\lambda = \{A, B, \Pi\}$. In Figure 2-2, the number of distinct discrete states, $N=3$. The number of observation symbols, $M=2$. The state transition probabilities are

$$A = \{a_{ij}\} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}. \quad (2.9)$$

The state observation probabilities are

$$B = \{b_{i v_k}\} = \begin{bmatrix} b_{10} & b_{11} \\ b_{20} & b_{21} \\ b_{30} & b_{31} \end{bmatrix}. \quad (2.10)$$

And the initial state probabilities are

$$\Pi = \{\pi_k\} = [\pi_1 \quad \pi_2 \quad \pi_3]. \quad (2.11)$$

2.2 Three Basic HMM Problems¹

Given the hidden Markov model defined in the previous section, three basic problems of interest need to be solved efficiently; an explanation for why the three problems will be given in the next section:

- **The Evaluation Problem:** Given an observation sequence, $O = O_1 O_2 \dots O_T$, and the complete parameter set of an HMM, $\lambda = \{A, B, \Pi\}$, what is $P(O/\lambda)$, the probability of the observation sequence O given the model parameter set λ ?
- **The Decoding Problem:** Given an observation sequence O and the complete parameter set of an HMM λ , what is the optimal state sequence $Q = Q_1 Q_2 \dots Q_T$ which maximizes $P(Q, O/\lambda)$?
- **The Training Problem:** Given an observation sequence O , what is the optimal model λ which maximizes $P(O/\lambda)$?

¹ The material presented in this section is based on [20], [4], and [18].

2.2.1 A Solution to the Evaluation Problem – The Forward Algorithm

The evaluation problem is to compute $P(O|\lambda)$, the probability of the observation sequence, $O=O_1O_2\dots O_T$, given the model parameter λ .

Since the state sequence, $Q=Q_1Q_2\dots Q_T$, corresponding to the observation sequence O is hidden, $P(O|\lambda)$ has to be computed by summing $P(O, Q|\lambda)$ over all possible state sequences.

$$P(O|\lambda) = \sum_{\text{all } Q} P(O, Q|\lambda) \quad (2.12)$$

$$P(O, Q|\lambda) = P(O|Q, \lambda) P(Q|\lambda). \quad (2.13)$$

According to the state independence assumption (2.1), we can write:

$$P(Q|\lambda) = \pi_{Q_1} a_{Q_2Q_1} a_{Q_3Q_2} \dots a_{Q_TQ_{T-1}}. \quad (2.14)$$

Also, the output independence assumption (2.8) allows us to conclude:

$$P(O|Q, \lambda) = b_{Q_1O_1} b_{Q_2O_2} \dots b_{Q_TO_T}. \quad (2.15)$$

Therefore,

$$P(O|\lambda) = \sum_{\text{all } Q} \pi_{Q_1} \cdot (a_{Q_2Q_1} a_{Q_3Q_2} \dots a_{Q_TQ_{T-1}}) \cdot (b_{Q_1O_1} b_{Q_2O_2} \dots b_{Q_TO_T}). \quad (2.16)$$

The direct calculation of $P(O|\lambda)$ (2.16) involves calculations on the order of $2TN^T$. This computation becomes unfeasible as the number of possible states, N , or the length of the observation sequence T increases. This necessitates a more efficient way of computing $P(O|\lambda)$.

Fortunately, an efficient algorithm exists. First, let us define the *forward variable*:

$$\alpha_t(i) = P(O_1 O_2 \dots O_t, Q_t = S_i | \lambda). \quad (2.17)$$

The variable $\alpha_t(i)$ denotes the joint probability of the partial observation sequence, $O_1O_2\dots O_t$, and the state S_i at time t , given the model λ . It can be calculated recursively:

$$\alpha_t(i) = \begin{cases} \pi_i b_{iO_1} & t = 1, 1 \leq i \leq N \\ \left[\sum_{j=1}^N \alpha_{t-1}(j) a_{ji} \right] b_{iO_t}, & 2 \leq t \leq T, 1 \leq i \leq N. \end{cases} \quad (2.18)$$

From the definition of the forward variable (2.17), it is clear that the probability of the entire sequence can be expressed as:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i). \quad (2.19)$$

Equations (2.17) through (2.19) illustrate how to compute $P(O|\lambda)$ by first recursively evaluating the forward variables, $\alpha_t(i)$, from $t=1$ to $t=T$ (2.18), and then summing all the forward variables at time T , the $\alpha_T(i)$'s (2.19). The above steps are often referred to as the *forward algorithm*. The number of calculations involved is on the order of TN^2 instead of $2TN^T$ (2.16). Hence, the forward algorithm can be used to solve the evaluation problem much more efficiently.

2.2.2 A Solution to the Decoding Problem – The Viterbi Algorithm

The decoding problem involves finding an optimal state sequence given the observation sequence, $O=O_1O_2\dots O_T$, and the model parameter λ . Assume that the optimality criterion is to maximize $P(Q, O|\lambda)$, the joint probability of the state sequence, $Q=Q_1Q_2\dots Q_T$, and the observation sequence O given the model λ . The optimal state sequence is denoted by Q^* .

The well-known *Viterbi algorithm*, based on dynamic programming, solves exactly this optimization problem. In it, $\delta_t(i)$ denotes the maximum probability of the optimal partial state sequence, $Q_1Q_2\dots Q_{t-1}$, with the state S_i at time t and observing the partial observation sequence, $O_1O_2\dots O_t$, given the model λ .

$$\delta_t(i) = \max_{Q_1Q_2\dots Q_{t-1}} P(Q_1Q_2\dots Q_{t-1}Q_t = S_i, O_1O_2\dots O_t|\lambda). \quad (2.20)$$

Much like the forward variable $\alpha_t(i)$, $\delta_t(i)$ can be calculated recursively:

$$\delta_t(i) = \begin{cases} \pi_i b_{iO_1} & t=1, 1 \leq i \leq N \\ \max_{1 \leq j \leq N} \delta_{t-1}(j) a_{ji} b_{iO_t}, & 2 \leq t \leq T, 1 \leq i \leq N. \end{cases} \quad (2.21)$$

From the definition of $\delta_t(i)$ (2.20), it is clear that

$$P(Q^*, O|\lambda) = \max_{1 \leq i \leq N} \delta_T(i). \quad (2.22)$$

Using (2.21) and (2.22), one can compute the joint probability of the optimal state sequence and the observation sequence given the model, $P(Q^*, O|\lambda)$. Note that the memory usage is very efficient, *i.e.*, at any time t , only N forward variables, $\delta_t(i)$ need to

be stored. By keeping track of the argument i in both equations as $P(Q, O/\lambda)$ is being maximized, one can recover the optimal state sequence completely.

Also note that $P(Q^*, O/\lambda)$ can be viewed as the biggest component of $P(O/\lambda)$ in (2.12). When $P(Q^*, O/\lambda)$ is a good approximation of $P(O/\lambda)$, one can use the Viterbi algorithm instead of the forward algorithm for the evaluation problem. This will conserve computation. Since the computational complexity of the Viterbi algorithm is even less than that of the forward algorithm. For speech recognition applications, this approximation is sometimes used during recognition or training.

2.2.3 A Solution to the Training Problem – The Baum-Welch Algorithm

The training problem computes the optimal model parameter, λ , given an observation sequence, $O=O_1O_2\dots O_T$. Here, the optimality criterion is to maximize $P(O/\lambda)$, the probability of the observation sequence given the model λ . The training problem is by far the most difficult of the three basic problems. In fact, no known analytical solution exists for this optimization problem. However, an iterative procedure known as the *Baum-Welch algorithm* or *forward-backward algorithm* guarantees a locally optimal solution to the training problem. The Baum-Welch algorithm is a special case of the EM (Expectation-Maximization) algorithm [22]. In this section, we describe the Baum-Welch Algorithm.

First, let us define the *backward variable*:

$$\beta_t(i)=P(O_{t+1} O_{t+2} \dots O_T|Q_t=S_i, \lambda). \quad (2.23)$$

The variable $\beta_t(i)$ denotes the probability of the partial observation sequence, $O_{t+1} O_{t+2} \dots O_T$, given the state S_i at time t and the model λ . The backward variable is similar to the forward variable (2.17). It can also be calculated recursively:

$$\beta_t(i) = \begin{cases} 1 & t = T, 1 \leq i \leq N \\ \sum_{j=1}^N \beta_{t+1}(j) a_{ij} b_{jO_{t+1}}, & 2 \leq t \leq T, 1 \leq i \leq N. \end{cases} \quad (2.24)$$

From the definition of the backward variable (2.23) and the definition of the initial state probabilities (2.2), it is clear that

$$P(O|\lambda) = \sum_{i=1}^N \beta_1(i) \pi_i. \quad (2.25)$$

Second, let us define $\xi_t(i, j)$, the joint probability of the state S_i at time t and the state S_j at time $t+1$, given the observation sequence O and the model λ .

$$\xi_t(i, j) = P(Q_t = S_i, Q_{t+1} = S_j | O, \lambda). \quad (2.26)$$

$\xi_t(i, j)$ can be completely expressed in terms of the forward variable, the backward variable, and the model λ .

$$\begin{aligned} \xi_t(i, j) &= \frac{P(Q_t = S_i, Q_{t+1} = S_j, O|\lambda)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_{jO_{t+1}} \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_{jO_{t+1}} \beta_{t+1}(j)}. \end{aligned} \quad (2.27)$$

Note that the denominator of (2.27) needs to be calculated only once. This quantity, which is equivalent to $P(O|\lambda)$, is often referred to as the *alpha terminal*. It indicates how well the model λ matches the observation sequence O .

With the current model as $\lambda=(A, B, \Pi)$, we can iteratively reestimate the model, $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\Pi})$, where

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}, \quad (2.28)$$

$$\bar{b}_{iv_k} = \frac{\sum_{t=1}^T \sum_{j=1}^N \xi_t(i, j)}{\sum_{t=1}^T \sum_{j=1}^N \xi_t(i, j)}, \text{ and} \quad (2.29)$$

$$\bar{\pi}_i = \sum_{j=1}^N \xi_1(i, j). \quad (2.30)$$

\bar{a}_{ij} can be seen as the ratio of the expected number of transitions from state S_i to S_j to the expected number of transitions from state S_i to any state. Similarly, \bar{b}_{iv_k} can be seen as the ratio of the expected number of times in state i while observing the symbol v_k to the

expected number of times in state i . $\bar{\pi}_i$ can be seen as the expected number of times in state S_i at time $t=1$.

The above iterative procedure for updating the model λ is the essence of the Baum-Welch algorithm. Baum and others have proven that $P(O|\bar{\lambda}) \geq P(O|\lambda)$ for every iteration of the algorithm [22, 23]. Hence, $P(O|\bar{\lambda}) \approx P(O|\lambda)$ is used as the stopping criterion for the algorithm. The likelihood function, $P(O|\lambda)$ will eventually converge to a local maximum.

2.3 Using HMMs for On-line Handwriting Recognition

Now, armed with these tools, one can solve all the basic problems associated with an HMM. Why should one use HMMs to model on-line cursive handwriting? How does one use HMMs for on-line cursive handwriting recognition?

These questions will be answered by showing how one can model letters, words, and sentences with HMMs, and how one can perform the recognition of isolated words and sentences based on the solutions to the three basic HMM problems.

2.3.1 Modeling Letters, Words, and Sentences with HMMs

In this section, the modeling of letters, words, and sentences is described respectively.

2.3.1.1 Modeling Letters

The most natural unit of handwriting is a letter. A letter is represented by a 7-state left-to-right HMM. The HMM model is illustrated in Figure 2-3.

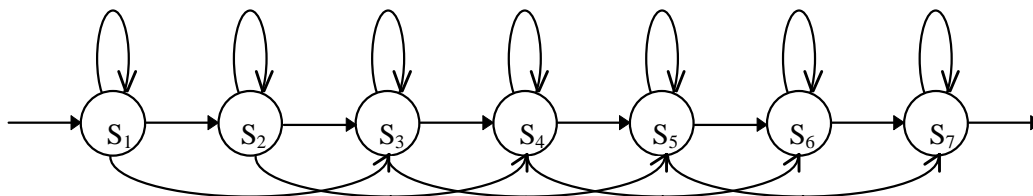


Figure 2-3: A 7-state HMM for a letter.

The left-to-right type of HMM, a special class of HMMs, have an additional property that the state index is non-decreasing as the time increases, *i.e.*

$$a_{ij} = P(Q_n = S_j | Q_{n-1} = S_i) = 0, \quad i > j. \quad (2.31)$$

Since the left-to-right HMM can effectively model the time-dependent property in a signal, the on-line handwriting signal can be modeled by the left-to-right model. The number of states, 7, was chosen experimentally to maximize the recognition accuracy [5].

Every state of the HMM has a self transition loop. Also, the first five states can make transitions which skip the immediate next states. These skipping transitions give the HMM flexibility to model the variability in time. Because of these skipping transitions, the minimum length of an observation sequence for a letter is four, *i.e.*, going through the state S_1 , S_3 , S_5 , and S_7 once each. For this on-line cursive handwriting recognition task, 89 letter symbols were used. Consequently, we used 89 different 7-state HMMs to model them. However, one 7-state HMM per letter symbol is not enough. Contextual effects of cursive handwriting introduce some variability. For example, the letter “*i*” written after an “*m*” can be very different from that after a “*v*” (see Figure 2-4). HMMs modeling trigraphs instead of HMMs modeling letters can be used to model these contextual effects. Each trigraph represents a combination of three letters. “ $p[e]n$ ” denotes a trigraph of letter “*e*” with left-context letter “*p*” and right-context letter “*n*”. Theoretically, up to 704969 (89^3) trigraphs would be used. However, in English only a small subset of them are frequently presented (about 6,500 in the training data portion of the BBN handwriting data corpus).



Figure 2-4: Illustration of contextual effects on a cursively written “*i*”. The left “*i*” is written after an “*m*”; the right “*i*” is written after a “*v*”.

2.3.1.2 Modeling Words

A word is made of a sequence of letters. Knowing how to model a letter with HMMs, one can model words simply by concatenating a number of 7-state HMMs, each of which models a letter. For example, an HMM for the letter “*p*”, an HMM for the letter “*e*”, and an HMM for the letter “*n*” form the HMM model for the word, “*pen*” (see Figure 2-5).

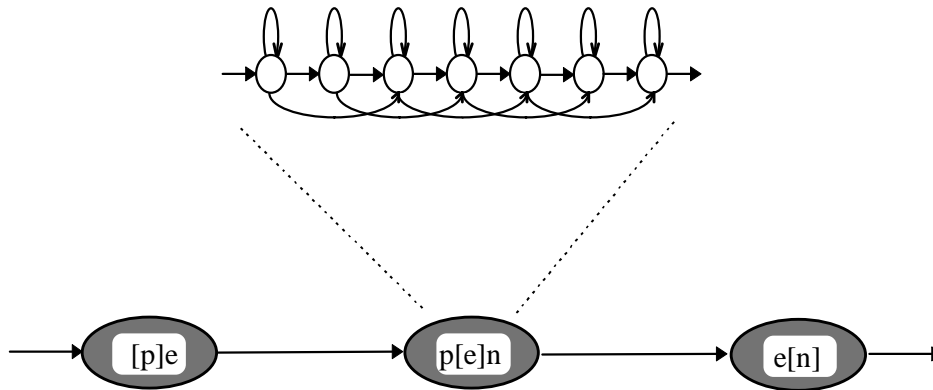


Figure 2-5: Illustration of an HMM modeling of the word "pen", consists of three 7-state HMMs for the trigraph "[p]e", "p[e]n", and "e[n]" .

As a rule in cursive writing, for words with the letters "i", "j", "x", or "t", the writer adds the dots or crosses at the end of writing the word. Let call these dots or crosses the *dot and cross strokes*. To model these words, letter-HMMs modeling three special characters, the "i" or "j" dot, the "t" cross, and the "x" cross, can be concatenated to the end of the HMM modeling these words. However, since these dots and crosses can be written in an arbitrary order, each of these words would have multiple word-HMMs representing each of them. The number of word-HMMs representing the same word can grow quite large as the number of "i", "j", "x", or "t" letters increase. To simplify, these special letters are represented by a single letter, the "backspace" character. Therefore, for each letter, "i", "j", "x", or "t" in any word, we simply concatenate one more 7-state HMM for the "backspace" letter. For example, the HMM model of the word 'it' consists of four individual 7-state HMMs, each of which represents the letter "i", "t", "backspace", and "backspace", respectively.

2.3.1.3 Modeling Sentences

Although the number of all possible words is limited in this recognition task (25,595 words in total), the number of sentences that can be composed with these words is very large. To model each sentence explicitly is simply computationally impossible. Fortunately, a probabilistic sentence network can be constructed to represent all of the possible sentences.

Let us assume that the words in a sentence satisfy the *Markov condition*:

$$P(W_n/W_{n-1}, W_{n-2}, \dots, W_0) = P(W_n/W_{n-1}), \quad \forall n. \quad (2.32)$$

The Markov condition of word in a sentence means that the current word is only dependent on the previous word, and not any other previous words. $P(W_n/W_{n-1})$ is called the *bigram probability*. The bigram probabilities and the *initial word probabilities*, $P(W_0)$, together specify a bigram grammar for sentences. The probability of any sentence composed of a set of words, $W_0W_1\dots W_{n-1}W_n$, can be approximated with this bigram grammar:

$$P(W_0W_1\dots W_{n-1}W_n) \approx P(W_0)P(W_1/W_0)P(W_2/W_1)\dots P(W_{n-1}/W_{n-2})P(W_n/W_{n-1}). \quad (2.33)$$

For example, Figure 2-6 shows a simple bigram grammar made up of words such as "It", "I", "is", "am", "orange", and "young". This bigram grammar encodes knowledge about what sentences made out of words "It", "I", "is", "am", "orange", and "young" are more likely and which are not. For instance, "It is orange", "It is young", and "I am young", these three sentences are all quite plausible, while the sentence, "I am orange", is not.

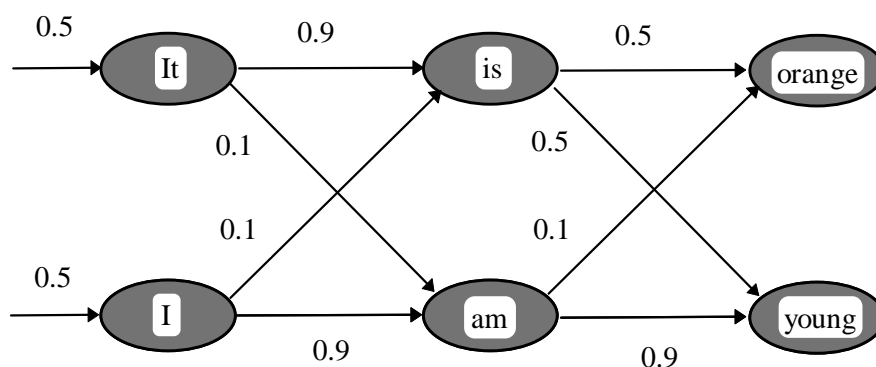


Figure 2-6: Illustration of a simple bigram grammar for sentences composed of the words "It", "I", "is", "am", "orange", and "young".

The bigram grammar can be estimated from a sentence data corpus. Here, the data corpus consists of approximately two million sentences taken from the Wall Street Journal from 1987 to 1989. The sentences were used to compute the bigram probabilities and initial word probabilities.

To model all possible sentences made of the 25,595 words with HMMs, we constructed a bigram grammar with all of these words using the Wall Street Journal sentence data corpus. Then, we replaced each node of the bigram grammar with the HMM model for the corresponding word. This new composite HMM represents all possible sentences made of the 25,595 words.

2.3.2 Recognition of Handwritten Isolated Letters Using HMMs

Now, let us assume that an observation sequence, $O=O_1O_2\dots O_T$, is to be obtained from someone writing an isolated letter. Furthermore, let us assume that the HMM model parameters, $\lambda_i=\{A_i, B_i, \Pi_i\}$, for each of the 89 letters are known. The problem of recognizing isolated handwriting letters is equivalent to deciding from which one of 89 letters the observation sequence O is observed.

Since we are able to solve the evaluation problem, it is possible to perform isolated letter recognition using HMMs. First, we can compute $P(O/\lambda_i)$, which is the probability of the observation sequence O given the HMM model parameters for each of 89 letters using the forward algorithm. Then the letter corresponding to the maximum probability, $P(O/\lambda_i)$, is chosen as the optimal answer. According to Bayesian classification theory, picking this letter minimizes the probability of error, therefore:

$$l_{opt} = \arg \max_{l \in \{88 \text{ letters}\}} P(O/\lambda_l). \quad (2.34)$$

2.3.3 Recognition of Cursively Handwritten Sentences Using HMMs

The problem of recognizing cursively handwritten sentences is equivalent to deciding from which sentence an observation sequence, $O=O_1O_2\dots O_T$, is observed. Here, we not only assume that the HMM model parameters of each letter, $\lambda_i=\{A_i, B_i, \Pi_i\}$, are known, but that the parameters of the bigram grammar are also known.

Since the decoding problem is now solvable, it is possible to perform sentence recognition using HMMs. First, we can compute the optimal state sequence, $Q^*=Q_1^*Q_2^*\dots Q_T^*$ which corresponds to the observation sequence using the Viterbi algorithm. Since the optimal state sequence is associated with a deterministic sequence of letters and words, this sequence of words is the desired result for the sentence. One might suggest using the forward algorithm for isolated letter recognition to solve the problem of sentence recognition. In fact, one could. Imagine having a unique HMM for each sentence i . It would then be necessary to compute the probability $P(O/\lambda_i)$ for each sentence. Since the number of possible sentences grows exponentially with the number of words, this method of utilizing the forward algorithm is computationally impractical.

Therefore, it is necessary to use the Viterbi algorithm in order to solve the problem of sentence recognition.

2.4 Summary

By extending the discrete-state Markov process to define HMMs, the HMMs are able to model both the variance in time and signal space presented in cursively written letters, words, and sentences. Solutions to three basic problems of HMMs enabled us to perform various cursive handwriting recognition tasks: the solution to the training problem for training the HMM model parameters, the solution to the evaluation problem for handwriting recognition of isolated letters, and the solution to the decoding problem for handwriting recognition of sentences. The solutions to these problems are fundamental in explaining the feature experiments for HMM-based on-line handwriting recognition systems. Before one can fully appreciate the details of the feature experiments, an in-depth description of the data corpus as well as the baseline system is necessary.

3. BBN On-line Cursive Handwriting Data Corpus

3.1 Overview

The feature experiments were run on the BBN on-line cursive handwriting data corpus. The corpus contains cursive handwritten sentences from the ARPA Wall Street Journal task [24]. It contains 25,595 words made up of 89 unique symbols: 52 lower and upper case letters, 10 digits, 24 punctuation marks, a special symbol, space, and *backspace*. Refer to Appendix Appendix A: for a detailed listing of the symbols. Individual digits and punctuation marks are considered as words. Refer to Appendix Appendix B: for a sample list of words from the data corpus. The *backspace* symbol models "t" and "x" crosses, and "i" and "j" dots. On average, a sentence has about 25 words. Figure 3-1 illustrates a typical sentence from this corpus.

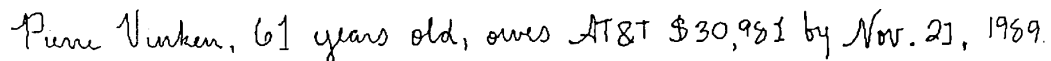


Figure 3-1: Sample handwriting from experimental corpus. (Transcription: Pierre Vinken, 61 years old, owes AT&T \$30,981 by Nov. 21, 1989.)

3.2 Building BBN On-line Cursive Handwriting Data Corpus

3.2.1 Data Collection

The handwriting was sampled by a GRiD™ Pen Top computer (Model 2260). The GRiD™ computer runs on an Intel 80386SL CPU with Microsoft Windows for Pen Computing 1.0 as its operating system. It has a side-lit, 9.5 inch, liquid crystal display (LCD). Unlike most LCDs, a person can write on the LCD with a wireless pen; the LCD not only displays what one has just written, but also records the pen location at 400 dots per inch (dpi) resolution at the same time. The display also has a mechanism to detect whether the pen is touching the display by sensing a micro-switch tip inside the wireless pen. If the pen is touching the display, the display would sense the *on* state of the micro-switch, thus detecting a *PenDown*. Otherwise a *PenUp* would be detected. Once the display detects a *PenDown*, it starts to sample the handwriting at 100Hz until a *PenUp* is

detected. Through a rolling median filter, the computer calculates and records the current pen x and y position.

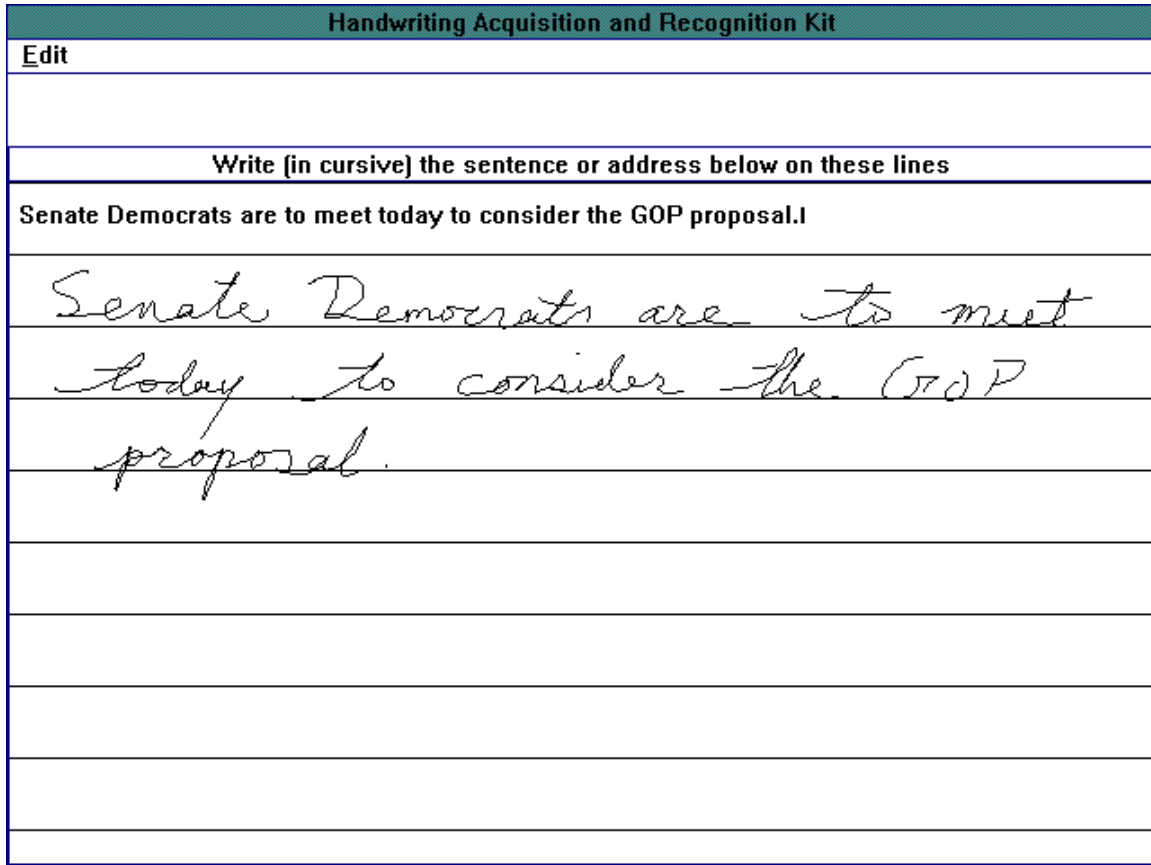


Figure 3-2: Screen caption of the GRiD™ LCD during a data collection session

The data were collected from students and young professionals from the Cambridge, Massachusetts area. The writers were financially compensated for their time. Sitting down at a table or desk for about 90 minutes per data collection session, the writers were told to copy the text displayed on the GRiD™ LCD by writing in cursive on the lines below the text of the same screen.

Figure 3-2 shows a screen caption of the GRiD™ LCD during a data collection session. The writers were also told that it is preferred that they write capitals in cursive but that it was not necessary. Additionally, all lower case letters had to be connected, *i.e.*, the pen could not be lifted to dot *i*'s or *j*'s or cross *x*'s or *t*'s as a word was being written, only after the last letter was finished. Furthermore, no touch-ups by the writers were allowed, *i.e.*, letters could not be made to look more distinct after they were written.

However, subjects were allowed to "erase" or "undo" the last pen strokes made to correct mistakes.

3.2.2 Data Preparation

After the handwriting data were collected, they had to be visually verified. Using an in-house software package, only three types of touch-ups were made: a) adding missed *i*-dots (mostly not picked up by the computer); b) deleting stray marks (including extra *i*-dots and periods); c) correcting stroke order (period and commas should come after *i*-dots and *t*-crosses).

The handwriting on the GRiD™ LCD was written on multiple lines. To model handwriting on multiple lines would be unnecessarily difficult. Fortunately, the *unwrapping filter* could join handwriting on multiple lines into text on a single line. The writer still saw the writing on multiple lines, but the “image” of the handwriting for the computer was converted to be on a single line by the unwrapping filter. Since the line heights of the multiple lines were constant, the unwrapping filter could decide on which line the sampled data was originally written by its *y* position. From this line number information, the filter could simply shift both its *x* and *y* positions by some constant amount to transform a sample point. After applying the unwrapping filter to the entire sample, point by point, the resulting data points appeared as if they were written on a signal line.

The resulting data from the unwrapping filter was processed by a *sampling distance filter*. The original handwriting data was sampled at 400dpi, but the writer could only view the same data at about 80dpi (GRiD™ LCD displaying resolution: 640x480-pixel on a 9.5 inch screen). Hence, the mismatched sample and display resolution can potentially cause problems. For example, the sampled data could resemble a miniature zero, “0”, while the intended writing was a period, “.”. The sampling distance filter eliminates this type of problem. While preserving the endpoints of each stroke, the filter eliminated any data points within 10 sampling units of the preceding data point. This can also be viewed as a filter which removes some spatial noise.

3.3 BBN On-line Cursive Handwriting Data Corpus and UNIPEN

In the literature of on-line handwriting recognition, performance results have been reported on different data corpora. This makes the comparison of recognition performance very difficult. In contrast, speech recognition research has benefited from using common data corpora for many years. UNIPEN, a project sponsored by the US National Institute of Standards and Technology (NIST) and the Linguistic Data Consortium (LDC), is addressing this issue by constructing a common corpus for on-line handwriting recognition [25].

Today, UNIPEN has gathered on-line handwriting data from over thirty organizations. NIST has distributed the data for research. It is also planning to have the first evaluation using this data in 1997. Unfortunately, the UNIPEN data were unavailable for the feature experiments because the official evaluation data were not released at the time. However, the BBN data corpus is a significant part of the UNIPEN data. Therefore, future evaluation results on the BBN portion of the UNIPEN data can be meaningfully compared with the results obtained in this thesis.

3.4 Training Data Set and Testing Data Set

For the feature experiment, only a subset of the BBN data corpus was used. This subset includes handwriting data from six different writers. Their initials are *aim*, *dsf*, *rgb*, *shs*, *slb*, and *wcd*. Table 3-1 summarizes the amount of data from each writer. Overall, this subset contains 3,637 sentences or 98,420 words of cursively handwritten data.

Table 3-1: Summary of handwriting data of the six writers.

subject	No. of sentences	No. of words
<i>aim</i>	634	17867
<i>dsf</i>	609	16674
<i>rgb</i>	653	17767
<i>shs</i>	634	16976
<i>slb</i>	618	15323
<i>wcd</i>	489	13813

total	3637	98420
-------	------	-------

The data from each writer is divided into two parts. One part of 70 sentences is for testing, and the rest is for training. To obtain WI recognition performance on the testing data of a writer, the training data from other five writers is used for training the model. Table 3-2 summarizes the size of training data and testing data for each writer. In summary, the testing data size is 420 sentences. Note that the actual training data only contains 3,217 non-duplicated sentences ($3217 = 3637 - 420$).

Table 3-2: Summary of training data set and testing data set.

Subject	No. of training sentences	No. of training words	No. of testing sentences	No. of testing words
<i>aim</i>	2653	71844	70	1623
<i>dsf</i>	2678	73159	70	1745
<i>rgb</i>	2634	72131	70	1810
<i>shs</i>	2653	72794	70	1682
<i>slb</i>	2669	74419	70	1654
<i>wcd</i>	2798	76093	70	1818

4. The Baseline BYBLOS On-line Cursive Handwriting Recognition System

4.1 Overview

The BYBLOS on-line cursive handwriting recognition system is based on the BBN BYBLOS Continuous Speech Recognition (BYBLOS CSR) system [26]. The BYBLOS CSR system is an HMM-based recognition system. Since the on-line handwriting recognition task is very similar to the task of continuous speech recognition, the two systems are also similar. Each system consists of three major components: the front-end, the trainer, and the decoder. The two systems only differ in their front-ends.

In 1994, Starner *et al.* first adapted the BYBLOS CSR system to the task of on-line handwriting recognition [8] [9]. In [8] and [9], the performance of the writer-dependent (WD) system was published. However, the performance of the writer-independent (WI) system has never been published. For the purposes of these experiments, the WI system will be used as the baseline system. In the following section, we will first discuss the functionality of the three modules for the BYBLOS on-line handwriting recognition system. The performance of the WI system will then be stated in the results section.

4.2 Modules

4.2.1 Front-end

The goal of the front-end module is to generate an observation sequence for the HMM from the input handwriting data. In this case, the data is assumed to be from the BBN on-line cursive handwriting data corpus. This is accomplished in three steps: preprocessing the original data, computing the feature vectors, and calculating the observation symbols.

4.2.1.1 Preprocessing

Two sub-modules are involved with the preprocessing of handwriting data. One is the *invisible stroke construction sub-module*, and the other is the *padding filter*.

In Section 3.2.1, it was said that the GRiD™ LCD samples the handwriting between each *PenDown* and *PenUp*. However, no data samples were generated between *PenUp* and *PenDown*. To simulate a continuous-time signal for the HMM, the invisible stroke construction sub-module connected a straight line between the *PenUp* sample and the *PenDown* sample and then it sampled the line ten times. Since the straight line were never explicitly written by the writer, we refer to them as *invisible strokes*. Figure 4-1 gives some examples of invisible strokes, where the dash parts are the invisible strokes. Note that these invisible strokes are not only results of spaces between words, but can also be the result of dotting *i*'s and *j*'s and crossing *t*'s and *x*'s.



Figure 4-1: Illustration of invisible strokes. The highlighted dash lines are the invisible strokes.

A *stroke* consists of the writing from *PenDown* to *PenUp* or from *PenUp* to *PenDown*. Writing from *PenDown* to *PenUp* will be referred to as *visible strokes*. Since each stroke can potentially represent a letter which is modeled by a 7-state HMM, each stroke must have an observation sequence made up of at least four observation symbols, *i.e.*, four data points. The *padding filter* fulfills this minimum stroke length criterion. While preserving the endpoints of each stroke, the filter re-samples the stroke path at 10 equal-time locations. This re-sampling process is only done on strokes consisting of less than 10 data points.

4.2.1.2 Computing Feature Vectors

An analysis program computes a six-element feature vector for each data point. The six features used in the baseline system are the writing angle, the delta writing angle, the delta *x* position, the delta *y* position, the *PenUp/PenDown* bit, and the $sgn(x-max(x))$

bit. The first four elements have real number values, while the last two elements have binary values.

Assigning the *PenUp/PenDown* bit for each data point is quite simple. The *PenUp/PenDown* bit denotes whether a data point is actually written by a writer or is artificially generated (e.g., *invisible strokes*). Therefore, we assign *PenUp* to a data point which is inside an *invisible stroke*, and we assign *PenDown* to all other data points.

The delta x position and the delta y position are computed locally. For any data point, the delta x position is equal to the difference between the x position of the data points two samples after and the x position of the data points two samples before. However, there are a couple of exceptions. Since “two samples before” is not well defined for the first and the second data points, their own x positions are used as the “two samples before” instead. Also since “two samples after” is not well defined for the last and the second to last data point, their x positions are used as the “two samples after” instead.

$$\text{delta_}x_t = \begin{cases} x_{t+2} - x_t & t = 0, 1 \\ x_{t+2} - x_{t-2} & 2 \leq t \leq N - 2 \\ x_t - x_{t-2} & t = N - 1, N. \end{cases} \quad (4.1)$$

The delta y positions are calculated similarly to the calculation of delta x positions.

The writing angle is computed from the delta x position and the delta y position. It is approximated by the *arctangent* of the ratio between the delta x position and the delta y position. The delta writing angle of the first data point is set to 0.0. For all other data points, the delta writing angle is equal to the difference between writing angles of the data point and the previous data point. Figure 4-2 illustrates the writing angle and the delta writing angle features.

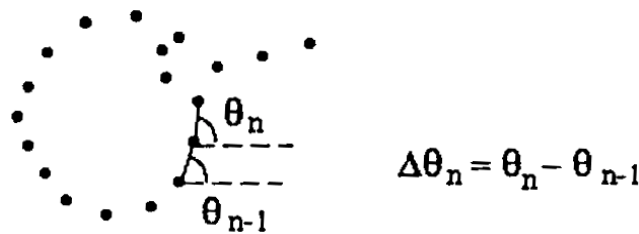


Figure 4-2: The writing angle and the delta writing angle features.

The $sgn(x-max(x))$ bit indicates whether the x position of the current data point is greater or less than the x positions of all of the preceding data points. A value of 0 denotes when it is less, and a value of 1 denotes when it is greater. For example, as illustrated in Figure 4-3, the lighter portions are encoded with 0 , and the darker portions are encoded with 1 . To compute the $sgn(x-max(x))$ bit, the maximum x position of all the preceding x positions is update, and called $max(x)$. Then the x position of the current data point is compared with $max(x)$. If it is greater, it is assigned 1 . Otherwise, it is assigned 0 .



Figure 4-3: Calculation of the $sgn(x-max(x))$ bit for the word “the” only. The highlighted portions are encoded with 1 and the gray portions are encoded with 0 , for the $sgn(x-max(x))$ bit.

The six features for each data point can be sorted into three groups. The first group, which includes the information presented in the delta x position, the delta y position, and the *PenUp/PenDown* bit, can be used to reconstruct the original handwriting image. This reconstruction can also be done with the second group, which consists of the writing angle, the delta writing angle, and the *PenUp/PenDown* bit. Hence, these two groups of three features can separately characterize the handwriting image completely. Naturally, one would ask the question, “Are the two separate descriptions redundant?” Yes, they are. However, the two descriptions used together outperformed either of the two used alone. We can think of the two descriptions as two different views of the same handwriting image, and as such they provide the HMM with different statistical information. The third group consists of the $sgn(x-max(x))$ bit, which simply provides additional information for the HMM.

4.2.1.3 Calculating Observation Symbols

Since a discrete-density HMM is used, observation symbols for each data point must be supplied to the HMM instead of the feature vectors. Essentially, the 6-dimensional feature vector needs to be converted into a set of M discrete observation

symbols. For the baseline system, the number of observation symbols, M , is equal to 256. So, how does one convert a multi-dimensional feature vector into M observation symbols?

This is essentially a data compression step. It is done through vector quantization (VQ) [27], a technique to convert a multi-dimensional vector into a discrete symbol. A *VQ codebook* is first computed by *k-means clustering* [28] on a quarter of the training data (about 805 sentences). A VQ codebook consists of M *prototype vectors*. The M prototype vectors divide the 6-dimensional feature space into M clusters, where each prototype vector characterizes the *centroid* of the cluster. To obtain the discrete observation symbol, a feature vector is compared with each of the prototype vectors according to some *distance measure*, and the feature vector is assigned the observation symbol i , which corresponds to the index i of the prototype vector of the shortest distance. The distance measure used here is the Euclidean distance. Note that the VQ codebook is only calculated once, and it is used for the computation of all of the discrete observation symbols.

4.2.2 Trainer

The goal of the trainer module is to train the HMM parameter, $\lambda = \{A, B, \Pi\}$, for each letter and trigraph. The trainer utilizes the Baum-Welch algorithm extensively. Refer to section 2.2.3 for details on the Baum-Welch algorithm.

The HMM models for letters are trained first, so they can be used to initialize the models for trigraphs. To train the HMM models for letters, their own model parameters need to be initialized. The transition probabilities, a_{ij} , are assigned such that the expected time of going through all 7 states of the HMM is 30 frames. (In the BBN data corpus, the “length” of each letter is 30 frames on average.) The observation probabilities, $b_{i v_k}$, are set to reflect a uniform distribution. Note that the initial model for each of the 89 letters are exactly the same; they do not need to be initialized with actual data.

With the initial model for each letter, five iterations of the Baum-Welch algorithm are run on the 3,217 training sentences. For each sentence, one can create a sentence model using the letter models according to its transcription (refer to Section 2.3.1 for

detail on how to construct the sentence model). Finally, all of the letter models inside this sentence model are trained against the actual observation sequence of the sentence using the Baum-Welch algorithm.

After five iterations of training the letter models, the trigram models are initialized with the corresponding letter models. Now, the trigram models are ready to be trained. Instead of creating a sentence model using the letter models, a sentence model is constructed with the trigram models. Then the trigram models inside each sentence model are trained against the actual observation sequence of the sentence using the Baum-Welch algorithm. Six iterations of the Baum-Welch algorithm are run for training of the trigram models.

Since the training data is limited, some trigrams would appear in the training data only a few times. Thus, the observation probabilities, b_{iv_k} , for these trigrams would not be reliable. To ensure that all of the trigram models are dependable, we can smooth all of the observation probabilities of each trigram model using the following smoothing function:

$$b_{iv_k}^{trigram\ smoothed} = \lambda^{trigram} * b_{iv_k}^{trigram} + (1 - \lambda^{trigram}) * b_{iv_k}^{letter}, \quad (4.2)$$

where $\lambda^{trigram}$ is positively correlated with the number of trigram occurrences in the training data, $N^{trigram}$. If $N^{trigram}$ is very large, then the smoothed trigram observation probability, $b_{iv_k}^{trigram\ smoothed}$, would be very close to the trigram observation probability, $b_{iv_k}^{trigram}$. On the other hand, if $N^{trigram}$ is very small, then the smoothed trigram observation probability, $b_{iv_k}^{trigram\ smoothed}$, would be very close to the letter observation probability, $b_{iv_k}^{letter}$. However, it is tricky to choose the optimal $\lambda^{trigram}$. It is usually done heuristically.

4.2.3 Decoder

The baseline decoder uses the *forward-backward search* (FBS) algorithm [29] [30] [31]. The FBS algorithm is mathematically related to the Baum-Welch algorithm. It involves two search passes, the first pass is a forward *fast-match search*, and the second pass is a detailed backward *Viterbi's beam search* (VBS).

The VBS algorithm is an approximate Viterbi algorithm. Refer to Section 2.2.2 for details on the Viterbi algorithm. Recall that the Viterbi algorithm stores N of the variables $\delta_t(i)$ at any given time t . The number N is the number of possible states at time t . Each variable $\delta_t(i)$ represents the maximum probability of the optimum partial state sequence, $Q_1Q_2\dots Q_{t-1}$, with the state S_i at time t given the model λ . We can treat the N variables $\delta_t(i)$ as the scores of each N candidate state sequences. However, for this large-vocabulary, cursive handwriting recognition task, keeping N to be the number of all possible states at time t is simply too computationally expensive. Therefore, a direct application of the Viterbi algorithm is computationally unfeasible. Fortunately, the Viterbi algorithm can be approximated with the VBS algorithm by limiting the number N with a *pruning beamwidth* or *threshold* on the scores. For example, the pruning beamwidth can be set to 100 , so only the top 100 $\delta_t(i)$ are kept at any given time. Similarly, the threshold to the scores can be set such that only forward variables within 50% of the best score are kept, and the rest are disregarded. With appropriate pruning beamwidth or threshold, the VBS algorithm can approximate the Viterbi algorithm. As the pruning beamwidth increases and the threshold lessens, the VBS algorithm approaches the Viterbi algorithm.

However, picking the “appropriate” pruning beamwidth or threshold for the VBS algorithm is usually very difficult. The pruning beamwidth or threshold of a VBS algorithm is usually based on the variable $\delta_t(i)$, but unfortunately, a high value of $\delta_t(i)$ does not guarantee a high overall probability for the entire observation sequence because the future observation is not taken into account. Additionally, the algorithm can waste resources by keeping this high $\delta_t(i)$ as a candidate. Similarly, a low forward score does not mean a low overall score, and the algorithm can impair performance by dismissing the low value of $\delta_t(i)$ as a candidate.

The FBS algorithm avoids the above problem. It modifies the VBS algorithm by first performing a forward fast match, thus enabling the backward VBS algorithm to use $\tilde{P}(O|\lambda)$, an approximation of the probability of the *entire* observation sequence given the

model, $P(O|\lambda)$, as a pruning threshold. The key insight of the FBS algorithm comes from the following equation:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_t(i) \beta_t(i), \quad 1 \leq t \leq T. \quad (4.3)$$

The equation can be easily derived from the definitions of forward and backward variables (2.17) and (2.23). From the equation, one can realize that the probability of the *entire* observation sequence given the model $P(O|\lambda)$ can be calculated by their forward and backward variables at any given time t . During the forward fast match pass, a list of $\tilde{\alpha}_t(i)$, an approximation of the forward variables, $\alpha_t(i)$, are stored. To conserve memory, only those $\tilde{\alpha}_t(i)$ which occur at the ends of words are stored. During the backward VBS pass, $\tilde{P}(O|\lambda)$ are computed along with the backward variables $\beta_t(i)$ by:

$$\begin{aligned} \tilde{P}(O|\lambda) &\approx P(O|\lambda) \\ &= \sum_{i=1}^N \alpha_t(i) \beta_t(i) \\ &\approx \max_{i \in \{1, \dots, N\}} \alpha_t(i) \beta_t(i) \\ &\approx \tilde{\alpha}_t(i) \beta_t(i). \end{aligned} \quad (4.4)$$

To use $\tilde{P}(O|\lambda)$ for the pruning threshold, let us first define $\tilde{P}^{\max}(O|\lambda)$ to be the maximum of all $\tilde{P}(O|\lambda)$ at a given time t . For the backward VBS pass, only the backward variables which have $\tilde{P}(O|\lambda)$ within some percentage of $\tilde{P}^{\max}(O|\lambda)$ are kept. The percentage can define the pruning threshold for the FBS algorithm. Varying the threshold also varies the complexity of the search.

Experimentally, the FBS algorithm has shown an increase in search speed by a factor of 40 over the VBS algorithm while maintaining the same recognition performance [29]. Because of the FBS algorithm's speed saving and its more intuitive pruning threshold, we decided to use the FBS algorithm instead of the VBS algorithm for the decoder of the baseline system.

4.3 Results

The results of the baseline system were obtained after using the training data and testing data. During training, the training data set with transcriptions is first passed to the front-end module to generate observation symbols. These observation symbols are passed to the trainer module to generate the HMM model parameters. Figure 4-4 shows the block diagram of the training process.

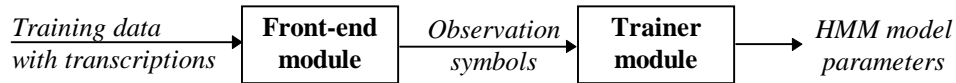


Figure 4-4: Block diagram of the training process.

For testing, the data of the testing data set is also first passed to the front-end module to generate observation symbols. With the HMM model parameters from the training process along with the observation symbols, the decoder module decodes the most likely sentence. Figure 4-5 shows the block diagram of the testing process.

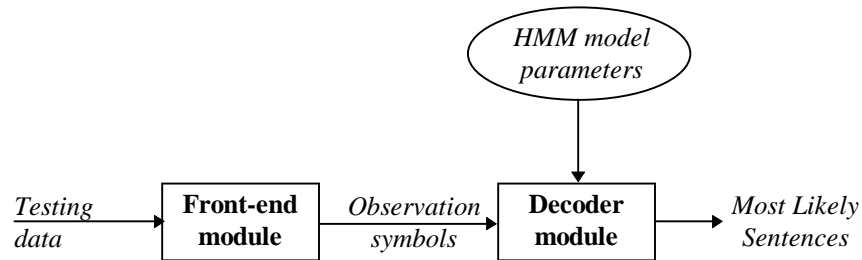


Figure 4-5: Block diagram of testing process.

Table 4-1: Baseline BBN on-line cursive handwriting recognition system.

Subject	Correct (%)	Substitution (%)	Deletion (%)	Insertion (%)	Total (%)
<i>aim</i>	85.8	10.2	3.9	1.2	15.3
<i>dsf</i>	80.1	16.7	3.3	5.7	25.6
<i>rgb</i>	90.9	6.6	2.5	1.5	10.6
<i>shs</i>	94.7	3.7	1.6	1.0	6.2
<i>slb</i>	84.2	12.3	3.6	1.5	17.3
<i>wcd</i>	92.6	4.5	3.0	0.6	8.0
Ave.	88.1	8.9	3.0	1.9	13.8

The results of the baseline BBN on-line cursive handwriting recognition system is shown in Table 4-1. The testing data consists of 420 sentences in total, 70 sentences per writer for six writers. The total writer-independent word error rate is 13.8%.

The performance of the baseline system is quite good, especially for the few constraints under which the system is evaluated. However, the baseline system is still far from the being usable in any realistic large vocabulary handwriting recognition application. In the next chapter, experiments which attempted to improve the performance by adding new features to the feature vector will be discussed.

5. Features Experiments

5.1 Introduction

In the baseline system, a six-dimensional feature vector is calculated for each sample point of the handwriting signal. The features are delta x position, delta y position, writing angle, delta writing angle, *PenUp/PenDown*, and $sgn(x-max(x))$. Refer to Section 4.2.1.2 for a detailed description of the features. With these features, the baseline system obtained a word error rate of 13.8% under writer-independent mode on the BBN data corpus.

Since handwriting recognition systems have reported their error rates under different conditions in the research literature, it is hard to compare their performance directly. However, the baseline system would more than likely obtain one of the lowest error rates if the same large vocabulary, writer-independent, cursive on-line handwriting recognition task were performed by all the system.

Nevertheless, the error rate of the baseline system is still unacceptable for some handwriting recognition applications. We believe that significant portion of the error rate can be attributed to the inadequate information represented by the feature vector. The performance of the baseline system can be improved significantly by augmenting the six baseline features with new features, which would provide the HMM with information that was not represented by the original features.

There are several sources of inadequately represented information. Since the baseline feature vector does not contain explicit information about the y position, the baseline system had difficulties distinguishing an apostrophe, “ ’ ”, from a comma, “ , ” because the two punctuation marks are very similar in shape and the only way to distinguish the two is by their y position information. In Section 5.2.1, a new vertical height feature tries to incorporate the y position information into the feature vector.

Spaces between letters or words could help the system to determine the boundaries for letters and words. In the baseline system, due to the binary nature of the

$sgn(x-max(x))$ feature, the spacing is inadequately characterized. In Section 5.2.2, a new space feature attempts to represent the space information better.

HMM has an inherent *output independence assumption* (2.8), *i.e.*, the current output is only dependent on the current state. But the on-line handwriting signal does not satisfy this assumption of output independence. For example, the l-shaped stroke of "t" and the "t" cross stroke are very dependent on each other, but they can be separated by many HMM states. In Section 5.2.3, the hat stroke features were designed to overcome this type of problem.

In Section 5.2.4, the substroke features are used to incorporate more global information bearing features into the feature vector since the original features only represented information locally.

These four proposed features were investigated to see how they each would affect the recognition performance. The features were studied incrementally using the BBN data corpus, and the potential benefits of each feature were studied as well.

5.2 The Four Feature Experiments

5.2.1 The Vertical Height

The x position denotes the horizontal position. The y position denotes the vertical position. Figure 5-1 shows the definition of the x and y directions. The six features of the baseline system did not include the x position nor the y position. Can we possibly use them for our new system?

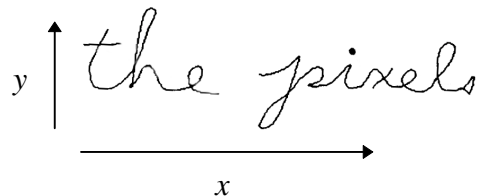


Figure 5-1: Definition of the x and y directions.

The x position of a particular sample measures the horizontal distance between the first sample of the sentence and the sample itself. Suppose that the handwriting has been normalized in the x direction and that the x position is measured from the first sample of

the character instead, then the x position information can indeed be useful. However, computing the distance by this method would require an error-prone step of pre-segmentation at the character level. In addition, normalizing the samples by the writing size is also very difficult. Therefore, it is impractical to adopt the x position as a new feature for the system.

The y position of a particular sample measures the vertical distance between the sample and a baseline. If the writing is not slanted and written on a straight baseline, and with some normalization in the y direction, the y position can provide useful information for classifying the different characters. For example, most handwriting of English can be divided into three zones: the upper zone, the middle zone, and the lower zone. Figure 5-2 illustrates the concept of the three zones. Guerfali and Plamondon had suggested the usefulness of the zones in [32]. For example, letters “a”, “c”, and “e” are completely inside the middle zone. On the other hand, letters “b”, “d”, and “h” are inside the middle and upper zones, and “g”, “p”, and “q” are inside the middle and lower zones.

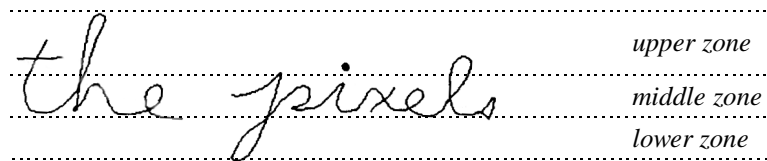


Figure 5-2: Illustration of three zones, lower, middle, and upper zones, of English handwriting.

If the zones can be reliably estimated for each sample, the zone information can be directly used for on-line handwriting recognition. To reliably estimate the zones, preprocessing steps such as correcting the baseline and normalizing the character slant are necessary. However, these steps are difficult and would be worthwhile as thesis projects of their own. Here, though, a very simple algorithm was used to convert the y position to the height feature.

First, the mean and standard variance of the y position were calculated for each sentence. The height feature of a sample was represented by the distance between the y position of the sample and the mean in standard deviations. Note that the height feature could be either positive or negative. Since most samples of a sentence fell into the middle zone, the samples with height feature value more than 1 was more likely to be in the upper zone, the samples with value less than -1 was more likely to be in the lower

zone, and the remaining samples were more likely to be in the middle zone. The probability of a particular sampling being in one of the three zones was related to the height feature.

The newly calculated height feature was added to the six baseline features to form a seven dimensional feature vector. By training and decoding with this new feature vector, the new system achieved a word error rate of 13.6% comparing to the baseline performance of 13.8%. The reduction in error rate¹ is 1.4%. Table 5-1 illustrates the detailed results.

Table 5-1: Results of the feature experiment with vertical height.

Subject	Correct (%)	Substitution (%)	Deletion (%)	Insertion (%)	Total (%)
<i>aim</i>	86.7	7.6	5.7	0.8	14.1
<i>dsf</i>	83.4	13.8	2.9	3.8	20.5
<i>rgb</i>	87.5	8.9	3.6	1.5	14.0
<i>shs</i>	93.5	4.5	2.0	1.0	7.4
<i>slb</i>	85.5	11.1	3.4	2.1	16.6
<i>wcd</i>	91.6	5.0	3.4	0.6	9.0
Ave.	88.1	8.5	3.5	1.6	13.6

From the performance of the new system, it is clear that this simple height feature did not give any significant improvement over the baseline system. However, it is not enough to conclude from this experiment that other y position-transformed features are not useful for on-line handwriting recognition. Quite possibly, with better baseline normalization and slant correction algorithms, the usage of y position-transformed features would improve the performance of on-line handwriting recognition systems.

5.2.2 Space Feature

How were the spaces between words modeled in the baseline system? Since the acceptable spaces between words for handwriting could vary greatly, an *OptionalSpace* model and a *Space* model were used for modeling inter-word spaces. The *Space* HMM would model the mandatory and minimum-size space between two words. The *OptionalSpace* HMM would model the portion of space longer than the minimum-size

¹ Reduction in error rate = (new error rate - original error rate) / original error rate.

space. Every model for words was extended by an extra *Space* model. When constructing a model for a sentence, the *OptionalSpace* model is inserted between two HMMs modeling words. A skipping transition, connecting the two words directly without connecting to the *OptionalSpace* model, is also inserted. Figure 5-3 illustrates the sentence model with *OptionalSpace* for the sentence, “*Lassie was rewarded*”. This setup accommodates variable length inter-word spaces very well. However, it is not without problems.

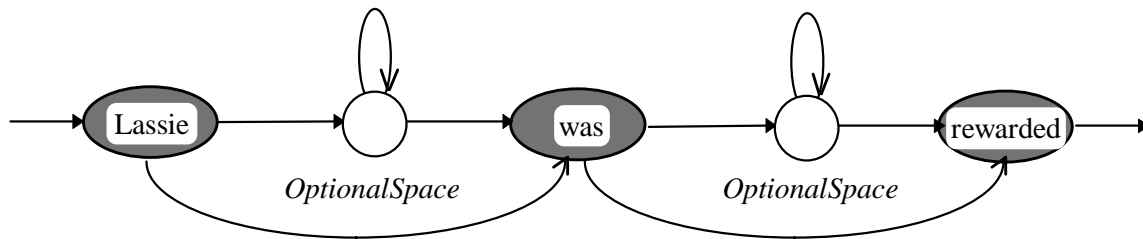


Figure 5-3: HMMs with *OptionalSpace* modeling the sentence, “*Lassie was rewarded*”. The dark ellipses represent HMMs modeling a word and the white circles represent HMMs modeling an *OptionalSpace*. Skipping transitions connect the two neighboring words directly.

The root of the problem lies in the requirement for mandatory and minimum-size spaces between any two words because a space may not always exist between certain two-word pairs. For example, there does not usually exist any space between the word “*over*” and the word period “*.*”. Actually, there usually exist no space between any word composed of ciphers and any word composed of punctuation marks. Therefore, when the HMM is forced to train *Space* models with some data which does not correspond to any actual space, the *Space* models are poorly constructed.

Two major changes were made to overcome this problem of poorly constructed *Space* models. First, the *Space* models were removed from the end of each word model. Thus, all the spaces were modeled by the *OptionalSpace* model. Second, a new space feature was used to characterize the spaces.

Only invisible strokes should represent gaps between words. Therefore, the space feature is only calculated for invisible strokes. The space feature of a visible stroke is simply assigned with 0 to represent a gap size of 0. For invisible strokes, the feature was calculated in two steps. First, a variable *gap* was calculated. Second, the variable *gap*

was scaled for the space feature. The variable *gap* represents the horizontal spaces between two words. Figure 5-4 illustrates the calculation of the variable *gap*.

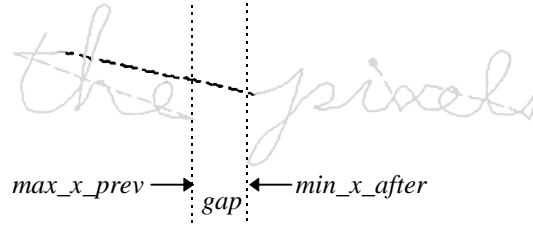


Figure 5-4: Calculation of the variable *gap* for the highlighted dash invisible stroke.

The variable *gap* is computed by the following algorithm for invisible strokes only:

1. $max_x_prev \leftarrow max_x$ of the visible stroke before the current invisible stroke.
2. $min_x_after \leftarrow$ minimum x position of all the visible strokes after the current invisible stroke.
3. $gap = min_x_after - max_x_prev$.

The space feature is a scaled version of the variable *gap*. The sampling resolution of GRiD™ LCD is 400dpi. A *gap* value of 100 represents a quarter inch wide space. Scaling any *gap* values of more than 100 on a logarithmic scale is equivalent to narrowing any inter-word spaces of more than a quarter inch to around a quarter inch. For *gap* less than 100, it is not scaled.

$$space = \begin{cases} gap & gap \leq 100 \\ 100 + \log_{10}(gap - 99), & gap > 100. \end{cases} \quad (5.5)$$

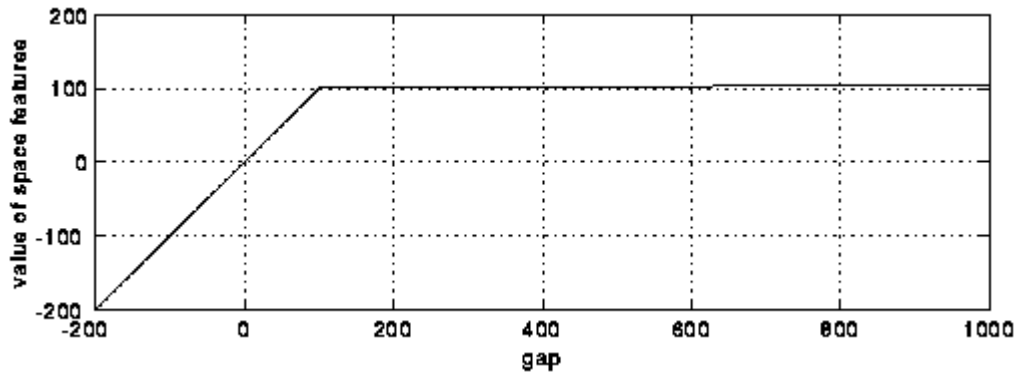


Figure 5-5: Scaling function of the variable *gap* for the space feature.

The newly calculated space feature was added to the other seven features, six baseline features and the height feature. Now, an eight dimensional feature vector represented the original handwriting image. By training and decoding with this new feature vector, the new system achieved a word error rate of 11.3% comparing to the

previous system performance of 13.6%. The reduction in error rate is 16.9%. Table 5-2 illustrates the detailed results.

Table 5-2: Results of the feature experiment with space.

Subject	Correct (%)	Substitution (%)	Deletion (%)	Insertion (%)	Total (%)
<i>aim</i>	89.4	7.2	3.4	1.0	11.6
<i>dsf</i>	85.8	11.8	2.4	2.7	16.9
<i>rgb</i>	89.1	8.7	2.3	1.2	12.2
<i>shs</i>	95.1	3.9	1.1	0.5	5.4
<i>slb</i>	87.1	10.6	2.4	1.2	14.1
<i>wcd</i>	92.8	5.6	1.6	0.3	7.5
Ave.	89.9	8.0	2.2	1.2	11.3

The new system shows a big improvement over the previous system. It is clear that the new method of representing inter-word spaces is superior over the original method because the new *OptionalSpace* model takes into account the inter-word space more accurately. It is not surprising to see such a large reduction in the error rate since the *OptionalSpace* model is one of the most frequently used word models.

5.2.3 Hat Stroke Features

Since the dots and crosses for the letter “*i*”, “*j*”, “*l*” and “*x*” are always written at the end of a cursive word containing these letters, these letters were actually modeled with two separate letter-HMM models within the baseline system. For example, the letter “*i*” is modeled with an HMM modeling a *partial* letter “*i*” and an additional HMM modeling the *backspace* character. Within an HMM model for a word, these two letter-HMM models can be separated by several other letter-HMM models. With the features in the baseline system, the feature vectors corresponding to the partial letter “*i*” is not distinguishable from the feature vectors which correspond to the letter “*i*”. Therefore, within the baseline system, the word “*late*” and the word “*tale*” can not be distinguished by the underlying HMM models.

The hat stroke feature attempts to solve this problem by attaching the feature vector of the dot and cross stroke with the feature vector of the associated letter. The hat stroke features consists of six separate features: *hat_stroke*, *dist_to_hat_x*, *dist_to_hat_y*, *closest_hat_length*, *closest_hat_angle*, and *closest_hat_error*. They are computed in

three steps. First, a decision is made about whether a visible stroke should correspond to a dot and cross stroke. Second, three features, *hat_angle*, *hat_length*, and *hat_error*, are computed for each dot and cross stroke. Third, the samples close to the dot and cross strokes are attached with features about the dot and cross stroke. They are computed by the following algorithm:

```

1. for (each visible stroke)
    if ((the space feature of the preceding invisible stroke is negative) and
        ((the entire stroke is less than 20 samples long) or
         (a quarter of the samples in the stroke has  $x$  less than current  $max_x$ )))
        hat_stroke = 1.0;
    else hat_stroke = 0.0;
2. for (each stroke such that hat_stroke is 1.0)
    hat_length  $\leftarrow$  total length of the stroke;
    Estimate the best line fit of the hat stroke samples;
    hat_angle  $\leftarrow$  orientation of the line;
    hat_error  $\leftarrow$  residue of the linear fit;
3. for (each sample)
    closest_hat_stroke  $\leftarrow$  the closest hat_stroke from the sample;
    if ( $x\_distance(sample, closest\_hat\_stroke) < MAX\_TO\_HAT\_DIST$ )
        closest_hat_length  $\leftarrow$  hat_length of closest_hat_stroke;
        closest_hat_angle  $\leftarrow$  hat_angle of closest_hat_stroke;
        closest_hat_error  $\leftarrow$  hat_error of closest_hat_stroke;
        dist_to_hat_x  $\leftarrow$   $x\_distance(sample, closest\_hat\_stroke)$ ;
        dist_to_hat_y  $\leftarrow$   $y\_distance(sample, closest\_hat\_stroke)$ ;
    else
        closest_hat_length = -1.0;           closest_hat_angle =  $\pi$ ;
        dist_to_hat_x =  $MAX\_TO\_HAT\_DIST$ ;   dist_to_hat_y = 0.0;
        closest_hat_error = -1.0;

```

The purpose of the first step is to determine whether a visible stroke is a hat stroke. For this, there are two necessary conditions to satisfy:

- 1) the space feature of the preceding invisible stroke must be negative;
- 2) the entire stroke must be less than 20 samples long or a quarter of the samples must be less than the current *max_x*.

In the second step, for each hat stroke, the total length of the stroke is calculated for its *hat_length*. A straight line is fitted to the hat stroke. Then, the orientation of the line is calculated for its *hat_angle*, and the estimation error is calculated for its *hat_error*.

In the third step, for each sample point, if a hat stroke is within the *MAX_TO_HAT_DIST* distance, then the *hat_length*, the *hat_angle*, and the *hat_error* features are assigned to the sample point. The minimum distance in *x* and *y* between the sample and the hat stroke are assigned to the *dist_to_hat_x* and *dist_to_hat_y* features respectively. If no hat stroke is within *MAX_TO_HAT_DIST* distance, the features are assigned with some constant values.

In summary, the six new features were calculated for the hat stroke feature. The previous calculated eight dimensional feature vector were augmented to include the new features. By training and decoding with this new fourteen dimensional feature vector, the new system achieved a word error rate of 10.7% comparing to the previous system performance of 11.3%. The reduction in error rate was 5.3%. Table 5-3 lists the detailed results.

Table 5-3: Results of the feature experiment with hat stroke.

Subject	Correct (%)	Substitution (%)	Deletion (%)	Insertion (%)	Total (%)
<i>aim</i>	89.7	7.0	3.3	0.6	10.8
<i>dsf</i>	86.1	10.9	3.0	1.8	15.8
<i>rgb</i>	88.7	8.7	2.5	1.3	12.6
<i>shs</i>	95.4	3.5	1.1	0.3	4.9
<i>slb</i>	87.2	10.0	2.8	0.8	13.7
<i>wcd</i>	93.9	4.4	1.7	0.2	6.3
Ave.	90.2	7.4	2.4	0.8	10.7

From the performance results of the new system, one can see that the new features improved the performance somewhat, which means the new features were useful. The suspected reason for only slight reduction in error was that not very many confusable word pairs, e.g. “late” and “tale”, exist in the 25K vocabulary. If this experiment had been done on a baseline system which did not have a closed vocabulary, these new features would have improved the baseline system more substantially than in this experiment.

5.2.4 Substroke Features

All six baseline features for a data sample were calculated from local data only. During the calculation of delta *x* and delta *y* position, the two samples before and two

samples after were used. Thus, the original feature vector of a particular sample contains at most information of five sample points, *i.e.*, within two samples away from the original sample. Substroke features attempt to incorporate information about samples within a bigger neighborhood of the sample.

To compute the substroke features, each stroke is first divided into substrokes. A straight line is fitted to each substroke. Each substroke is constrained such that the maximum error of the estimation stays below a constant, *ERROR_THRESHOLD*. Here, the variable *ERROR_THRESHOLD* is set to 10.0. The number of the samples within each substroke is assigned to the feature *substroke_pixels*. The length of the substroke is assigned to the feature *substroke_length*. The error of estimation between the line and the substroke data is assigned to the feature *substroke_error*. The orientation of the line fit to the substroke is assigned to the feature *substroke_angle*.

In summary, four new features, *substroke_pixels*, *substroke_length*, *substroke_error*, and *substroke_angle*, were calculated for the substroke features. These four new features were added to the previously calculated fourteen dimensional feature vector. By training and decoding with this new eighteen dimensional feature vector, the new system achieved a word error rate of 9.1%, an improvement over the previous system performance of 10.7%. The total reduction in error rate at this point was 15.0%. Table 5-4 lists the detailed results.

Table 5-4: Results of the feature experiment with substrokes.

Subject	Correct (%)	Substitution (%)	Deletion (%)	Insertion (%)	Total (%)
<i>aim</i>	89.6	7.1	3.3	0.5	10.9
<i>dsf</i>	89.2	8.9	1.9	1.5	12.3
<i>rgb</i>	91.5	6.7	1.8	1.0	9.6
<i>shs</i>	95.7	3.4	0.9	0.5	4.8
<i>slb</i>	89.5	7.8	2.7	0.4	10.9
<i>wcd</i>	93.6	4.6	1.9	0.1	6.5
Ave.	91.5	6.4	2.0	0.7	9.1

From the performance of the new system, one can see that the new features improved the performance markedly. Since it was not our goal to find the “optimal” way

to represent the substroke feature and global information, it is quite possible that other representations of substroke features can improve the recognition performance further.

5.3 Summary of Results

In this chapter, four separate feature experiments were performed. With all four sets of new global-information bearing features, the system obtained a word error rate of 9.1%, a 34% reduction in error from the performance of the baseline system of 13.8%. Among them, the space feature and the substroke features were most effective. Each of them reduced the error rate approximately 15%. Table 5-5 summaries the error rates from all four feature experiments.

Table 5-5: Summary of total word error rate of all four feature experiments.

Experiments	baseline Total (%)	height Total (%)	space Total (%)	hat stroke Total (%)	substroke Total (%)
<i>aim</i>	15.3	14.1	11.6	10.8	10.9
<i>dsf</i>	25.6	20.5	16.9	15.8	12.3
<i>rgb</i>	10.6	14.0	12.2	12.6	9.6
<i>shs</i>	6.2	7.4	5.4	4.9	4.8
<i>slb</i>	17.3	16.6	14.1	13.7	10.9
<i>wcd</i>	8.0	9.0	7.5	6.3	6.5
Ave.	13.8	13.6	11.3	10.7	9.1
Reduction in error	-	1.4	16.9	5.3	15.0
Tot. red. in error	-	1.4	18.1	22.5	34.1

Figure 5-6 illustrates the results of the feature experiments.

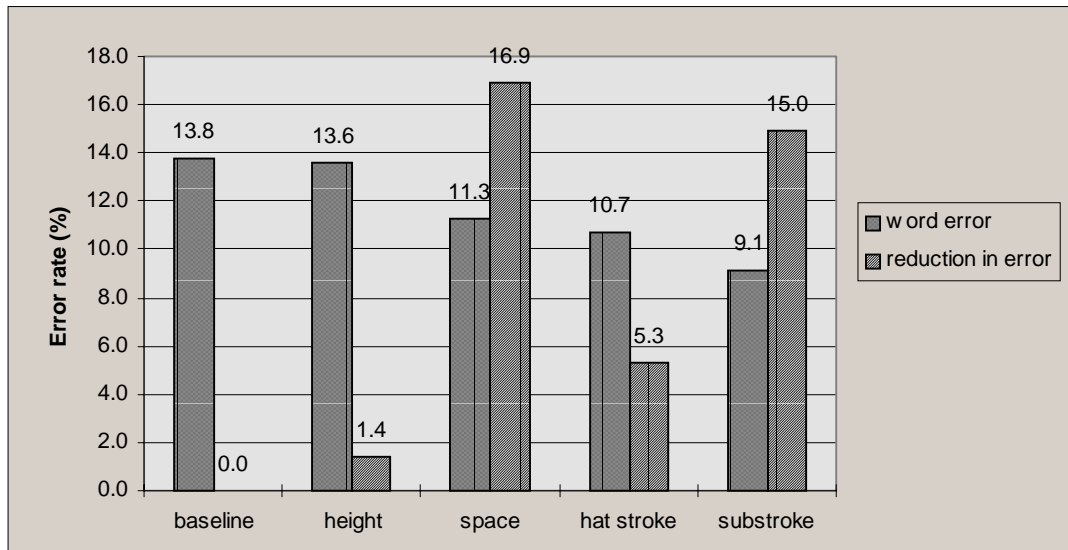


Figure 5-6: Total word error rates of four feature experiments.

6. Real-time On-line Cursive Handwriting Recognition Demonstration

6.1 Introduction

For most on-line handwriting recognition systems, the ability to perform in real time, *i.e.*, displaying recognition results immediately, is essential. For example, a schedule book with handwriting recognition technology requiring users to wait for several seconds after completing an entry would not be very effective. With this in mind, a system was constructed to show how the baseline system could be modified to run in real time.

Two factors affect an on-line cursive handwriting recognition system's ability to run in real time. One is the amount of computation power of the underlying hardware, and the other is the efficiency of the underlying algorithm in the software. In essence, the algorithm has to be capable of running in real time on the underlying hardware. In this BYBLOS on-line system, all algorithms have to run in real time on an SGI Indy workstation and GRiD™ PC.

Since the input tablet digitizer runs on a PC and both the front-end feature extraction and HMM recognizer run on a UNIX workstation due to computation and memory constraints, a real-time data link between the PC and the UNIX workstation was required. The design and implementation of this data link and its associated control structure posed interesting and difficult technical challenges.

6.2 Design & Implementation

The least error rate and modularity were a few of the design principles that guided the overall design of the real-time cursive handwriting recognition system. Smaller error rates generally improve the usability of a system. Modularity simplifies the processing of modifying a system. Since the communication cost between modules was not expensive computationally, four stand-alone modules were designed. These are the real-time data sampling module, the real-time front-end, the real-time decoder, and the results module.

The detailed functionality of the four modules is discussed in the following sections. Figure 6-1 illustrates how the four components were connected to construct the real-time recognition system.

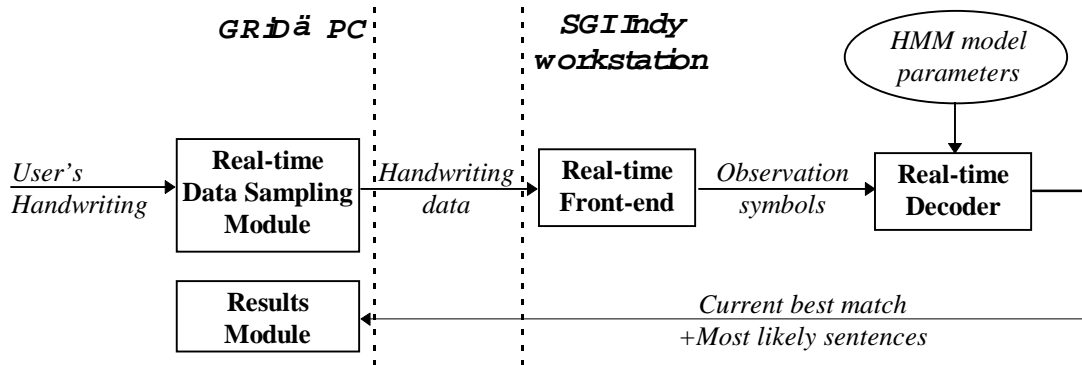


Figure 6-1: Block diagram of the real-time cursive handwriting recognition system.

To obtain the lowest error rate, the observation symbols have to arrive at the real-time decoder as early as possible. To achieve this, the real-time front-end must be as fast as possible. Since the SGI Indy workstation is much faster than the GRiD™ PC, the real-time front-end and the real-time decoder were implemented on the workstation. However, the real-time data sampling modules have to run on the PC because the handwriting can only be sampled in through the GRiD™ LCD, and the results module also have to be on the PC because the results must be displayed on the same LCD where the user inputs the handwriting.

Another major consideration for the real-time system is *causality*, *i.e.*, for any processing at time t , it can only use the inputs from before time t , not any from after time t . In Section 6.2.2, the causality of algorithms will be discussed in detail.

The communication between the modules uses TCP/IP as its network protocol. On the PC, network-related functionality was implemented using the standard Window Winsock interface library. On the SGI Indy workstation, the standard UNIX socket library was used.

6.2.1 Real-time Data Sampling Module

The real-time data sampling module on the GRiD™ PC samples the user's handwriting and sends the x and y positions and *PenUp/PenDown* information to the front-end module as the user writes. The sampling part of the module is very similar in

functionality to the data collection program described in Section 3.2.1. The data collection program was modified to convert to the real-time data sampling module.

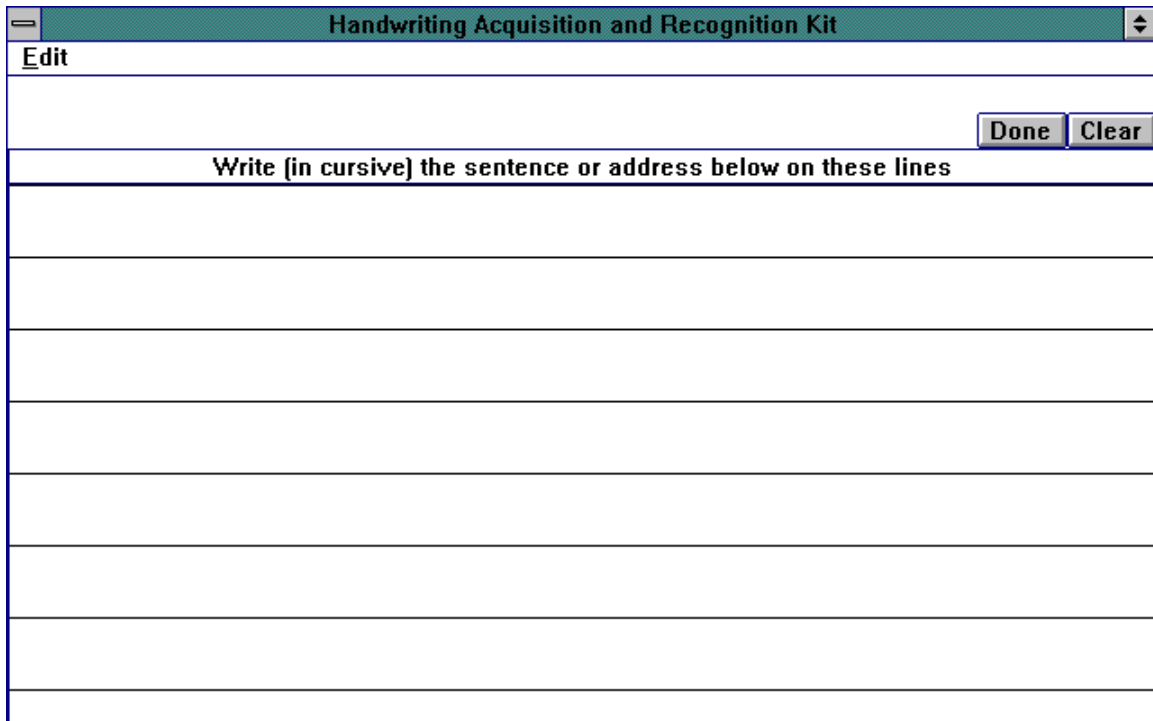


Figure 6-2: Screen capture of the real-time data sampling module.

Several changes had to be implemented. To increase the overall speed of the system, the sampled x and y positions and *PenUp/PenDown* information was sent to the real-time front-end immediately as the user writes on the GRiD™ LCD. The text on the top portion of the screen, for the writers to copy during data collection, was no longer needed, so it was removed. Two new buttons were added to the interface to enable the writer to write multiple sentences. One button, labeled “*end*,” signals the system that the current sentence is done. The real-time data sampling module sends an “*end_sentence*” signal to the real-time front-end to indicate the current sentence is ending and a new sentence is beginning. The other button is labeled “*clear*,” clears the user’s handwriting from the program window, and an “*end_sentence*” signal is sent out because clearing the handwriting also means the current sentence is ending and a new sentence is starting. Figure 6-2 shows the screen capture of this module.

6.2.2 Real-time Front-end

The real-time front-end runs on an SGI Indy workstation. It receives the sampled x and y positions and *PenUp/PenDown* information from the real-time data sampling module, and performs all the processing needed to convert the sampled data to discrete observation symbols for the real-time decoder.

Three sub-modules are involved with the processing of the sampled data: the real-time preprocessing sub-module, the feature-vectors computation sub-module, and the observation-symbols calculation sub-module. Figure 6-3 shows the block diagram of the real-time front-end.

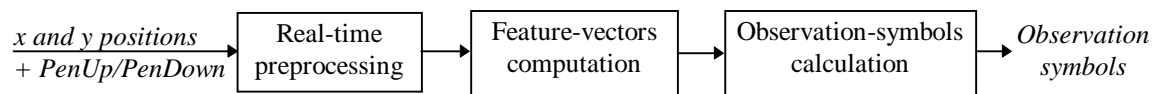


Figure 6-3: Block diagram of the real-time front-end.

The real-time preprocessing sub-module combined the data preparation step during the data collection described in Section 3.2.2 and the preprocessing step in the front-end described in Section 4.2.1.1. In brief, the sampled data was first passed through the *unwrapping filter* to convert handwriting on multiple lines to handwriting on a single line. Next, the *sampling distance filter* was applied to eliminate the problem of mismatched sampling and displaying resolution. Then, the *padding filter* was used to fulfill the minimum stroke length criterion. Both the unwrapping filter and the sampling distance filter were causal, but the padding filter was not. However, the padding filter had a maximum delay of 10 sample points which occurs at the beginning of a stroke. The original padding filter could be adopted to process in real time by delaying the computation on the current sample points by up to 10 samples. After applying the real-time versions of the three filters, the handwriting was ready for the feature-vectors calculation sub-module. Figure 6-4 showed the block diagram of the real-time preprocessing sub-module.

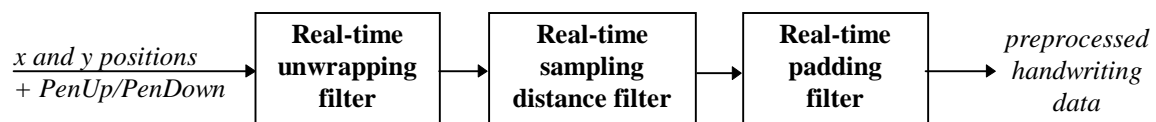


Figure 6-4: Block diagram of the real-time preprocessing sub-module.

From the preprocessed handwriting data, the feature-vectors computation sub-module computed six features: the writing angle, the delta writing angle, the delta x position, the delta y position, the *PenUp/PenDown* bit, and the $\text{sgn}(x-\max(x))$ bit. Since the new features from the four feature experiments all require non-causal processing, they were not adopted for the real-time system. The computation of these features was very similar to the partial description of the front-end in Section 4.2.1.2. The calculation was modified to compute from causal information only. The calculation of the *PenUp/PenDown* bit was causal originally. By delaying the feature vector computation of the current data point by two samples to accommodate the computation of differences for the other five features, it could also be calculated in real time.

From the feature vectors, the observation-symbols calculation sub-module computed the discrete observation symbols. This was done similarly to the partial description of the front-end in Section 4.2.1.3. With the VQ codebook already computed by k-mean clustering during the training process, the feature vector was compared with each prototype vector of the VQ codebook by a distance measure. The discrete observation symbol corresponds to the prototype vector index of the shortest distance. Since the process was inherently causal, no modification was necessary.

Over all three processing steps, the real-time front-end processing step could have a maximum delay of 12 sample points. In other words, the preprocessing step guarantees that at any given time t , all the samples corresponding to before the time $t-12$ would have already been converted into discrete observation symbols. Since an average letter was made of 30 sample points, a 12-sample delay did not hamper the performance of the overall system.

6.2.3 Real-time Decoder

Since the forward-backward search (FBS) algorithm described in Section 4.2.3 is very efficient in time, by changing its *pruning threshold*, the FBS algorithm can perform cursive handwriting decoding in real time.

As the real-time decoder receives discrete observation symbols from the real-time front-end, the real-time decoder performs the forward fast match and saves the

approximate forward scores. After the user signals the end of sentences by pressing the “end” button on the GRiD™ PC, or after the real-time decoder receives the “*end_sentence*” signal from the real-time front-end, the real-time decoder then performs the backward Viterbi’s beam search (VBS). The pruning threshold for the forward fast match is tuned such that the real-time decoder “keeps up” with one’s writing speed. Similarly, the pruning threshold for the backward VBS is tuned such that the most likely sentence of the VBS can be found within a couple of seconds after the writer finishes the sentence.

6.2.4 Results Module

The results module “listens” to the real-time decoder and displays whatever the real-time decoder sends to it on the GRiD™ PC. The best match is displayed as the user inputs handwriting on the GRiD™ LCD and as the real-time decoder performs the forward fast match search. The best match changes quickly as more handwriting is input on the LCD. In addition, the most likely sentence is displayed as the backward VBS finishes the backward decoding after the user signals the end of sentence.

6.3 Summary & Results

At the start of the real-time cursive handwriting recognition system, all four of the modules first have to confirm that the socket communication connections function properly and then each of the modules need to reset itself to be ready for a new sentence. This can be done by the data sampling module which sends out an “*end_sentence*” signal.

As the user inputs handwriting on the GRiD™ LCD, the data sampling module sends the sampled data to the real-time front-end. Next, the real-time front-end computes the observation symbols for each sampled data points and sends the observation symbols to the real-time decoder. Then, the real-time decoder performs the forward fast match on the observation sequence as the real-time decoder receives them and sends the current best match to the results module on the PC. Finally, the results module displays what it receives from the decoder. This whole process continues until the “*end*” or the “*clear*” button is pressed by the user to indicate the end of the current sentence. Figure 6-5 shows

a screen capture of the real-time data sampling module and the results module during a real-time handwriting recognition demonstration.

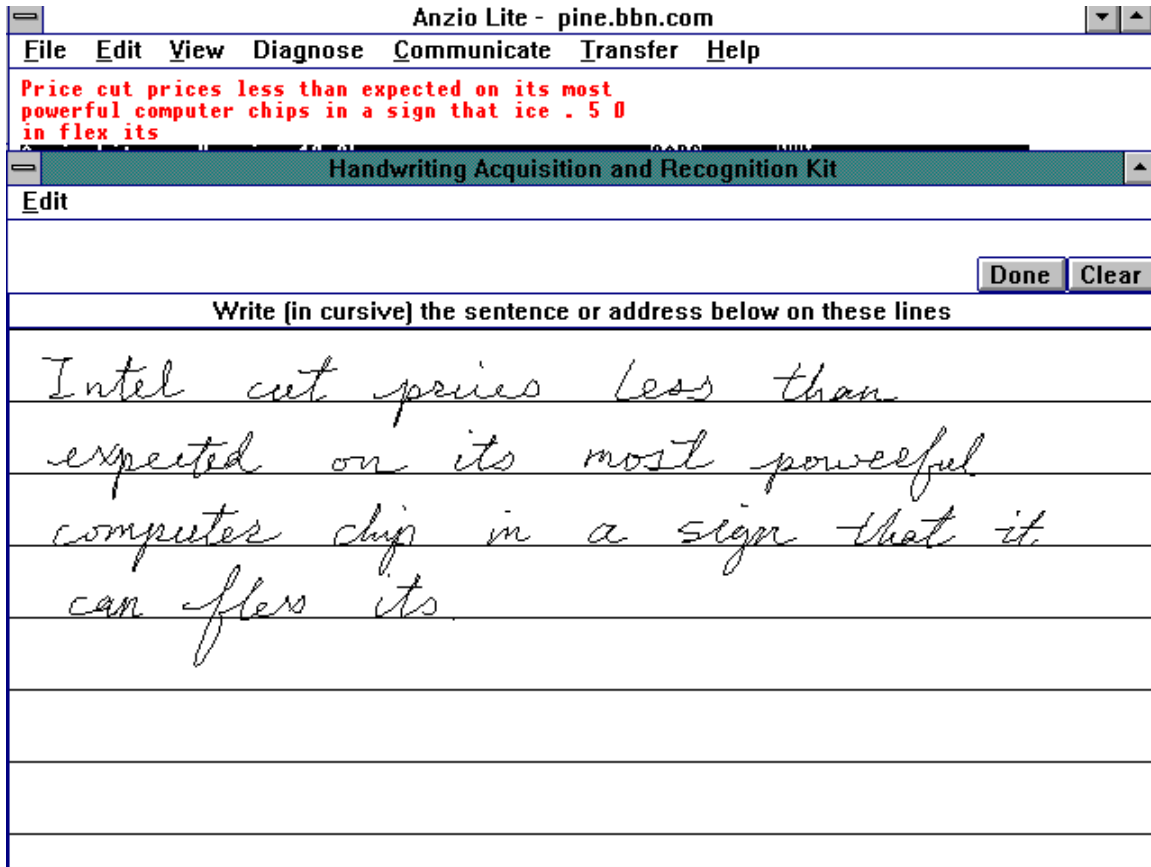


Figure 6-5: Screen capture of the real-time data sampling module and the results module during a real-time handwriting recognition demonstration. The writer wrote “Intel cut prices less than expected on its most powerful computer chip in a sign that it can flex its.” On the real-time data sampling module. The current best match is “Price cut prices less than expected on its most powerful computer chips in a sign that ice . 5 0 in flex its” on the results module.

Once the “end” or the “clear” button is pressed, the “end_sentence” signal is sent to the real-time front-end from the data sampling module. The real-time front-end computes observation symbols for all the leftover samples, and sends them to the decoder. At the same time, the real-time front-end resets itself to prepare for a new sentence. The real-time decoder finishes the forward fast match as it receives the last portion of the observation sequence, and immediately starts the backward Viterbi’s beam search (VBS). The real-time decoder then sends the result of the backward VBS, the most likely sentence, to the results module. After the completion of the backward VBS,

the real-time decoder also resets itself to prepare for a new sentence. At last, the results module displays the most likely sentence.

7. Summary & Future Directions

7.1 Summary

In this thesis I have presented a series of feature experiments aimed at showing that the performance of the baseline system can be improved dramatically by augmenting the six baseline features with new features, which would provide the HMM with information about the handwriting which was not represented by the original features. A new vertical height feature was used to characterize vertical height. A new space feature was used to represent inter-word space. The hat stroke features were used to overcome HMM's output independence assumption. The substroke features were implemented to improve the characterization of global information. By training and testing on the BBN on-line cursive handwriting data corpus, with the new features the system obtained a word error rate of 9.1%, a 34% reduction in error from the baseline error rate of 13.8%. The space feature and the substroke features each reduced the word error rate approximately 15%. The new features improved the HMM's modeling of handwriting, thus, also improved the recognition performance of the overall system significantly.

The details of the HMM-based on-line cursive handwriting recognition system were presented: from modeling of letters, words, and sentences with HMMs to the training and decoding algorithms for HMMs; from the detailed description of the front-end to the algorithms of the trainer and the decoder. Also, by modifying the front-end to perform causal processing and constructing communication channels between a GRiD™ PC and an SGI Indy workstation, a real-time, large vocabulary, write-independent, on-line cursive handwriting recognition system was demonstrated.

7.2 Suggestions for Future Work

The results of the feature experiments showed that additional features, representing more information such as inter-word space and substroke features, can potentially improve the performance of the cursive on-line handwriting recognition system. The implemented space feature and substroke features were by no means

optimal. Further investigations can possibly lead to better ways of representing the intended information.

In connection with substroke features, improvements can potentially be made. First, instead of dividing strokes into substrokes by fitting straight lines to handwriting data, arcs with constant curvature can be used. Also, after constructing the substrokes, substroke pairs crossing each other are then represented by two set of values: one representing itself, the other representing its crossing “partner.”

From the results of the vertical height feature experiment, we were not able to draw enough conclusive evidence to support that the y position related features would be useful. However, since implementation of complicated normalization algorithms, *e.g.* slant removal and baseline normalization, was avoided, we were not able to conclude the ineffectiveness of y position related features. To further evaluate the ineffectiveness of the y position related features, normalization algorithms would have to be implemented first.

Although the handwriting recognition system can recognize words from a large vocabulary, it is still impossible for it to recognize out-of-vocabulary words such as new names and new abbreviations. However, it would not be that difficult to convert the current system to be one with an open-vocabulary by adding an word-HMM model for out-of-vocabulary words. Much like the method of representing all the sentences with a finite set of words (see Section 2.3.1.3), all the words can be represented by a finite set of characters by a character network where the transition probabilities are character-level bigram probabilities.

Since the space feature and the substroke features both have improved the recognition performance significantly, it is a natural extension to incorporate the two new sets of features into the real-time demonstration system to improve its recognition performance. Only the real-time front-end need to be modified. With some small, additional delay within the real-time front-end, both the space feature and the substroke features can be calculated with causal information only.

Feature, improve in hardware , common corpora

A large vocabulary, writer-independent, cursive on-line handwriting recognition system with a word error rate under 10% was conceptualized and implemented together with hidden Markov models and global information-bearing features. A similar system was created to perform recognition in real time. Further research will undoubtedly yield even greater gains in recognition performance than what has been achieved in this thesis. Improvements in computer hardware and usage of common data corpora, such as UNIPEN, in handwriting recognition will accelerate technological advances in this area. The realization of high performance, real-time, open-vocabulary, writer-independent, mixed style handwriting recognition systems will consequently be close to hand. These further improvements will pave the way to a revolution in human-machine interaction.

Appendix A: List of 89 Letters of BBN On-line Cursive Handwriting Data Corpus and the BYBLOS Symbols

<i>Letter</i>	<i>BYBLOS Symbol</i>	<i>Letter</i>	<i>BYBLOS Symbol</i>	<i>Letter</i>	<i>BYBLOS Symbol</i>
&	*a*	A	A	a	a
'	*ap*	B	B	b	b
@	*at*	C	C	c	c
`	*bq*	D	D	d	d
,	*c*	E	E	e	e
:	*co*	F	F	f	f
\$	*d*	G	G	g	g
-	*da*	H	H	h	h
.	*do*	I	I	i	i
=	*e*	J	J	j	j
[*lb*	K	K	k	k
{	*lc*	L	L	l	l
(*lp*	M	M	m	m
#	*p*	N	N	n	n
%	*pe*	O	O	o	o
+	*pl*	P	P	p	p
"	*q*	Q	Q	q	q
?	*qu*	R	R	r	r
]	*rb*	S	S	s	s
}	*rc*	T	T	t	t
)	*rp*	U	U	u	u
;	*se*	V	V	v	v
/	*sl*	W	W	w	w
*	*st*	X	X	x	x
<i>Space</i>	_	Y	Y	y	y
<i>Backspace</i>	<	Z	Z	z	z
0	*zz*	5	*zi*	<i>OptionalSpace</i>	-
1	*zo*	6	*zs*	--	--
2	*zt*	7	*ze*	--	--
3	*zh*	8	*zg*	--	--
4	*zf*	9	*zn*	--	--

Appendix B: A Sampling of the Vocabulary of the BBN On-line Cursive Handwriting Data Corpus

Since the entire list consists of 25,595 words, *only* words starting with the capital letter A is list below.

A	A_
A's	A-*ap*-s-_
AB	A-B-_
ABA	A-B-A-_
ABA's	A-B-A-*ap*-s-_
ABB	A-B-B-_
ABBIE	A-B-B-I-E-_
ABC	A-B-C-_
ABC's	A-B-C-*ap*-s-_
ABD	A-B-D-_
ABORTION	A-B-O-R-T-I-O-N-_
ABUSE	A-B-U-S-E-_
AC	A-C-_
ACCEPTANCES	A-C-C-E-P-T-A-N-C-E-S-_
ACCOUNT	A-C-C-O-U-N-T-_
ACCOUNTING	A-C-C-O-U-N-T-I-N-G-_
ACQUISITION	A-C-Q-U-I-S-I-T-I-O-N-_
ACTUAL	A-C-T-U-A-L-_
ADB	A-D-B-_
ADRs	A-D-R-s-_
AEP	A-E-P-_
AFL	A-F-L-_
AFRICAN	A-F-R-I-C-A-N-_
AG	A-G-_
AG's	A-G-*ap*-s-_
AGIP	A-G-I-P-_
AGREES	A-G-R-E-E-S-_
AH	A-H-_
AIDS	A-I-D-S-_
AIRLINE	A-I-R-L-I-N-E-_
AIRLINES	A-I-R-L-I-N-E-S-_
AK	A-K-_
ALCOHOL	A-L-C-O-H-O-L-_
ALII	A-L-I-I-_
ALLIANCES	A-L-L-I-A-N-C-E-S-_
ALLWASTE	A-L-L-W-A-S-T-E-_
ALLY	A-L-L-Y-_
ALPA	A-L-P-A-_
AMERICAN	A-M-E-R-I-C-A-N-_
AMEX	A-M-E-X-_
AMR	A-M-R-_
AMT	A-M-T-_
AN	A-N-_
ANC	A-N-C-_
ANC's	A-N-C-*ap*-s-_

AND	A-N-D- <u> </u>
ANGRY	A-N-G-R-Y- <u> </u>
ANNOUNCED	A-N-N-O-U-N-C-E-D- <u> </u>
ANNUITIES	A-N-N-U-I-T-I-E-S- <u> </u>
APARTHEID	A-P-A-R-T-H-E-I-D- <u> </u>
APPEARS	A-P-P-E-A-R-S- <u> </u>
ARBITRAGE	A-R-B-I-T-R-A-G-E- <u> </u>
ART	A-R-T- <u> </u>
AS	A-S- <u> </u>
ASA	A-S-A- <u> </u>
ASCAP	A-S-C-A-P- <u> </u>
ASEAN	A-S-E-A-N- <u> </u>
ASLACTON	A-S-L-A-C-T-O-N- <u> </u>
ASSET	A-S-S-E-T- <u> </u>
ASSETS	A-S-S-E-T-S- <u> </u>
ASSOCIATES	A-S-S-O-C-I-A-T-E-S- <u> </u>
ASSOCIATION	A-S-S-O-C-I-A-T-I-O-N- <u> </u>
AST	A-S-T- <u> </u>
AT	A-T- <u> </u>
ATS	A-T-S- <u> </u>
ATT	A-T-T- <u> </u>
ATT's	A-T-T-*ap*-s- <u> </u>
AUCTION	A-U-C-T-I-O-N- <u> </u>
AUDITS	A-U-D-I-T-S- <u> </u>
AUSTRALIAN	A-U-S-T-R-A-L-I-A-N- <u> </u>
AUTO	A-U-T-O- <u> </u>
AZT	A-Z-T- <u> </u>
Aaa	A-a-a- <u> </u>
Aalseth	A-a-l-s-e-t-h-<- <u> </u>
Aaron	A-a-r-o-n- <u> </u>
Abalkin	A-b-a-l-k-i-n-<- <u> </u>
Abbey	A-b-b-e-y- <u> </u>
Abbie	A-b-b-i-e-<- <u> </u>
Abbie's	A-b-b-i-e-*ap*-s-<- <u> </u>
Abbot	A-b-b-o-t-<- <u> </u>
Abby	A-b-b-y- <u> </u>
Aberdeen	A-b-e-r-d-e-e-n- <u> </u>
Abitibi	A-b-i-t-i-b-i-<<<<- <u> </u>
Abortion	A-b-o-r-t-i-o-n-<<<- <u> </u>
About	A-b-o-u-t-<- <u> </u>
Above	A-b-o-v-e- <u> </u>
Abraham	A-b-r-a-h-a-m- <u> </u>
Abrams	A-b-r-a-m-s- <u> </u>
Abramson	A-b-r-a-m-s-o-n- <u> </u>
Abrupt	A-b-r-u-p-t-<- <u> </u>
Absolutely	A-b-s-o-l-u-t-e-l-y-<- <u> </u>
Absorbed	A-b-s-o-r-b-e-d- <u> </u>
Abyss	A-b-y-s-s- <u> </u>
Academic	A-c-a-d-e-m-i-c-<- <u> </u>
Academically	A-c-a-d-e-m-i-c-a-l-l-y-<- <u> </u>
Academy	A-c-a-d-e-m-y- <u> </u>
Acadia	A-c-a-d-i-a-<- <u> </u>
Acapulco	A-c-a-p-u-l-c-o- <u> </u>
Acceptance	A-c-c-e-p-t-a-n-c-e-<- <u> </u>
Accepted	A-c-c-e-p-t-e-d-<- <u> </u>

Accessories	A-c-c-e-s-s-o-r-i-e-s-<_
Acclaim	A-c-c-l-a-i-m-<_
Accomplishing	A-c-c-o-m-p-l-i-s-h-i-n-g-<<_
Accor	A-c-c-o-r_
Accor's	A-c-c-o-r-*ap*-s_
Accord	A-c-c-o-r-d_
According	A-c-c-o-r-d-i-n-g-<_
Accordingly	A-c-c-o-r-d-i-n-g-l-y-<_
Account	A-c-c-o-u-n-t-<_
Accounting	A-c-c-o-u-n-t-i-n-g-<<_
Accounts	A-c-c-o-u-n-t-s-<_
Accrued	A-c-c-r-u-e-d_
Accumulation	A-c-c-u-m-u-l-a-t-i-o-n-<<_
Achenbaum	A-c-h-e-n-b-a-u-m_
Achenbaum's	A-c-h-e-n-b-a-u-m-*ap*-s_
Achievement	A-c-h-i-e-v-e-m-e-n-t-<<_
Ackerman	A-c-k-e-r-m-a-n_
Acquired	A-c-q-u-i-r-e-d-<_
Acquisition	A-c-q-u-i-s-i-t-i-o-n-<<<<_
Act	A-c-t-<_
Acting	A-c-t-i-n-g-<<_
Action	A-c-t-i-o-n-<<_
Active	A-c-t-i-v-e-<<_
Activity	A-c-t-i-v-i-t-y-<<<<_
Actually	A-c-t-u-a-l-l-y-<_
Ad	A-d_
Adam	A-d-a-m_
Adams	A-d-a-m-s_
Adams's	A-d-a-m-s-*ap*-s_
Add	A-d-d_
Added	A-d-d-e-d_
Addison	A-d-d-i-s-o-n-<_
Additionally	A-d-d-i-t-i-o-n-a-l-l-y-<<<<_
Addressing	A-d-d-r-e-s-s-i-n-g-<_
Adds	A-d-d-s_
Adia	A-d-i-a-<_
Adjusted	A-d-j-u-s-t-e-d-<<_
Adjusters	A-d-j-u-s-t-e-r-s-<<_
Adler	A-d-l-e-r_
Adley	A-d-l-e-y_
Administration	A-d-m-i-n-i-s-t-r-a-t-i-o-n-<<<<<<_
Administration's	A-d-m-i-n-i-s-t-r-a-t-i-o-n-*ap*-s-<<<<<<_
Administrator	A-d-m-i-n-i-s-t-r-a-t-o-r-<<<<<_
Administrators	A-d-m-i-n-i-s-t-r-a-t-o-r-s-<<<<<_
Admittedly	A-d-m-i-t-t-e-d-l-y-<<<<_
Adolph	A-d-o-l-p-h_
Adopting	A-d-o-p-t-i-n-g-<<_
Adults	A-d-u-l-t-s-<_
Advance	A-d-v-a-n-c-e_
Advanced	A-d-v-a-n-c-e-d_
Advancing	A-d-v-a-n-c-i-n-g-<_
Advertisers	A-d-v-e-r-t-i-s-e-r-s-<<<_
Advertising	A-d-v-e-r-t-i-s-i-n-g-<<<<_
Advest	A-d-v-e-s-t-<_
Advice	A-d-v-i-c-e-<_

Adviser	A-d-v-i-s-e-r-<-_
Advisers	A-d-v-i-s-e-r-s-<-_
Advisor	A-d-v-i-s-o-r-<-_
Advisors	A-d-v-i-s-o-r-s-<-_
Advisory	A-d-v-i-s-o-r-y-<-_
Advocates	A-d-v-o-c-a-t-e-s-<-_
Aerobet	A-e-r-o-j-e-t-<<-_
Aeronautical	A-e-r-o-n-a-u-t-i-c-a-l-<<-_
Aeronautics	A-e-r-o-n-a-u-t-i-c-s-<<-_
Aerospace	A-e-r-o-s-p-a-c-e_
Aetna	A-e-t-n-a-<-_
Aetna's	A-e-t-n-a-*ap*-s-<-_
Af	A-f_
Affair	A-f-f-a-i-r-<-_
Affairs	A-f-f-a-i-r-s-<-_
Afghan	A-f-g-h-a-n_
Afghanistan	A-f-g-h-a-n-i-s-t-a-n-<<-_
Afghanistan's	A-f-g-h-a-n-i-s-t-a-n-*ap*-s-<<-_
Afif	A-f-i-f-<-_
Aflatoxin	A-f-l-a-t-o-x-i-n-<<<-_
Afnasjev	A-f-n-a-s-j-e-v-<-_
Africa	A-f-r-i-c-a-<-_
Africa's	A-f-r-i-c-a-*ap*-s-<-_
African	A-f-r-i-c-a-n-<-_
After	A-f-t-e-r-<-_
Afterward	A-f-t-e-r-w-a-r-d-<-_
Aga	A-g-a_
Again	A-g-a-i-n-<-_
Against	A-g-a-i-n-s-t-<<-_
Age	A-g-e_
Agencies	A-g-e-n-c-i-e-s-<-_
Agency	A-g-e-n-c-y_
Agent	A-g-e-n-t-<-_
Agents	A-g-e-n-t-s-<-_
Agnelli	A-g-n-e-l-l-i-<-_
Agnellis	A-g-n-e-l-l-i-s-<-_
Agnellis'	A-g-n-e-l-l-i-s-*ap*-<-_
Agnew	A-g-n-e-w_
Ago	A-g-o_
Agreement	A-g-r-e-e-m-e-n-t-<-_
Agricultural	A-g-r-i-c-u-l-t-u-r-a-l-<<-_
Agriculture	A-g-r-i-c-u-l-t-u-r-e-<<-_
Agro	A-g-r-o_
Ahead	A-h-e-a-d_
Ahmanson	A-h-m-a-n-s-o-n_
Ahoy	A-h-o-y_
Aichi	A-i-c-h-i-<<-_
Aichi's	A-i-c-h-i-*ap*-s-<<-_
Aiken	A-i-k-e-n-<-_
Ailes	A-i-l-e-s-<-_
Aimed	A-i-m-e-d-<-_
Air	A-i-r-<-_
Air's	A-i-r-*ap*-s-<-_
Aircraft	A-i-r-c-r-a-f-t-<<-_
Airline	A-i-r-l-i-n-e-<<-_

Airlines	A-i-r-l-i-n-e-s-<-<-_
Airlines'	A-i-r-l-i-n-e-s-*ap*-<-<-_
Airplanes	A-i-r-p-l-a-n-e-s-<-_
Airport	A-i-r-p-o-r-t-<-<-_
Airways	A-i-r-w-a-y-s-<-_
Ait	A-i-t-<-<-_
Aitken	A-i-t-k-e-n-<-<-_
Aiwa	A-i-w-a-<-_
Ajinomoto	A-j-i-n-o-m-o-t-o-<-<-<-_
Akerfeldt	A-k-e-r-f-e-l-d-t-<-_
Akio	A-k-i-o-<-_
Akron	A-k-r-o-n_
Aktiebolaget	A-k-t-i-e-b-o-l-a-g-e-t-<-<-<-_
Akzo	A-k-z-o_
Akzo's	A-k-z-o-*ap*-s-_
Al	A-l_
Al's	A-l-*ap*-s-_
Ala	A-l-a_
Alabama	A-l-a-b-a-m-a_
Alagoas	A-l-a-g-o-a-s-_
Alamos	A-l-a-m-o-s-_
Alan	A-l-a-n_
Alan's	A-l-a-n-*ap*-s-_
Alar	A-l-a-r_
Alaska	A-l-a-s-k-a_
Alaska's	A-l-a-s-k-a-*ap*-s-_
Alaskan	A-l-a-s-k-a-n_
Albanese	A-l-b-a-n-e-s-e_
Albanians	A-l-b-a-n-i-a-n-s-<-_
Albany	A-l-b-a-n-y_
Alberg	A-l-b-e-r-g_
Albert	A-l-b-e-r-t-<-_
Albert's	A-l-b-e-r-t-*ap*-s-<-_
Alberta	A-l-b-e-r-t-a-<-_
Albion	A-l-b-i-o-n-<-_
Albuquerque	A-l-b-u-q-u-e-r-q-u-e_
Alden	A-l-d-e-n_
Alderson	A-l-d-e-r-s-o-n_
Alert	A-l-e-r-t-<-_
Alex	A-l-e-x-<-_
Alexander	A-l-e-x-a-n-d-e-r-<-_
Alfred	A-l-f-r-e-d_
Algeria	A-l-g-e-r-i-a-<-_
Algerian	A-l-g-e-r-i-a-n-<-_
Algiers	A-l-g-i-e-r-s-<-_
Algom	A-l-g-o-m_
Ali	A-l-i-<-_
Alice	A-l-i-c-e-<-_
Alisarda	A-l-i-s-a-r-d-a-<-_
Alisky	A-l-i-s-k-y-<-_
Alito	A-l-i-t-o-<-<-_
All	A-l-l_
Allan	A-l-l-a-n_
Allegany	A-l-l-e-g-a-n-y_
Alleghany	A-l-l-e-g-h-a-n-y_

Allegheny	A-l-l-e-g-h-e-n-y- __
Allegran	A-l-l-e-g-r-a-n- __
Allen	A-l-l-e-n- __
Allen's	A-l-l-e-n-*ap*-s- __
Allendale	A-l-l-e-n-d-a-l-e- __
Allergan	A-l-l-e-r-g-a-n- __
Alley	A-l-l-e-y- __
Alliance	A-l-l-i-a-n-c-e-<- __
Allianz	A-l-l-i-a-n-z-<- __
Allianz's	A-l-l-i-a-n-z-*ap*-s-<- __
Allied	A-l-l-i-e-d-<- __
Allies	A-l-l-i-e-s-<- __
Alligood	A-l-l-i-g-o-o-d-<- __
Alltel	A-l-l-t-e-l-<- __
Almost	A-l-m-o-s-t-<- __
Aloha	A-l-o-h-a- __
Along	A-l-o-n-g- __
Alongside	A-l-o-n-g-s-i-d-e-<- __
Alphonsus	A-l-p-h-o-n-s-u-s- __
Alpine	A-l-p-i-n-e-<- __
Alps	A-l-p-s- __
Already	A-l-r-e-a-d-y- __
Also	A-l-s-o- __
Alson	A-l-s-o-n- __
Alstyne	A-l-s-t-y-n-e-<- __
Altair	A-l-t-a-i-r-<<- __
Alternative	A-l-t-e-r-n-a-t-i-v-e-<<<- __
Alternatively	A-l-t-e-r-n-a-t-i-v-e-l-y-<<<- __
Althea	A-l-t-h-e-a-<- __
Although	A-l-t-h-o-u-g-h-<- __
Altman	A-l-t-m-a-n-<- __
Alto	A-l-t-o-<- __
Altogether	A-l-t-o-g-e-t-h-e-r-<<- __
Alton	A-l-t-o-n-<- __
Aluminum	A-l-u-m-i-n-u-m-<- __
Aluminum's	A-l-u-m-i-n-u-m-*ap*-s-<- __
Alurralde	A-l-u-r-r-a-l-d-e- __
Alvin	A-l-v-i-n-<- __
Always	A-l-w-a-y-s- __
Alyce	A-l-y-c-e- __
Alysia	A-l-y-s-i-a-<- __
Alzheimer's	A-l-z-h-e-i-m-e-r-*ap*-s-<- __
Am	A-m- __
Amadou	A-m-a-d-o-u- __
Amalgamated	A-m-a-l-g-a-m-a-t-e-d-<- __
Amarillo	A-m-a-r-i-l-l-o-<- __
Amateur	A-m-a-t-e-u-r-<- __
Amaury	A-m-a-u-r-y- __
Amazing	A-m-a-z-i-n-g-<- __
Amazon	A-m-a-z-o-n- __
Amazonia	A-m-a-z-o-n-i-a-<- __
Amazonian	A-m-a-z-o-n-i-a-n-<- __
Amdahl	A-m-d-a-h-l- __
Amendment	A-m-e-n-d-m-e-n-t-<- __
Amer	A-m-e-r- __

America	A-m-e-r-i-c-a-<_
America's	A-m-e-r-i-c-a-*ap*-s-<_
American	A-m-e-r-i-c-a-n-<_
American's	A-m-e-r-i-c-a-n-*ap*-s-<_
Americana	A-m-e-r-i-c-a-n-a-<_
Americans	A-m-e-r-i-c-a-n-s-<_
Amerinvest	A-m-e-r-i-n-v-e-s-t-<<_
Ames	A-m-e-s-_
Amex	A-m-e-x-<_
Amfac	A-m-f-a-c-_
Amicable	A-m-i-c-a-b-l-e-<_
Amid	A-m-i-d-<_
Amin	A-m-i-n-<_
Amityvilles	A-m-i-t-y-v-i-l-l-e-s-<<<_
Ammonium	A-m-m-o-n-i-u-m-<_
Amoco	A-m-o-c-o-_
Amoco's	A-m-o-c-o-*ap*-s-_
Among	A-m-o-n-g-_
Amor	A-m-o-r-_
Amparano	A-m-p-a-r-a-n-o-_
Amschel	A-m-s-c-h-e-l-_
Amsterdam	A-m-s-t-e-r-d-a-m-<_
Amtran	A-m-t-r-a-n-<_
Amvest	A-m-v-e-s-t-<_
Amy	A-m-y-_
An	A-n-_
Ana	A-n-a-_
Anac	A-n-a-c-_
Anadarko	A-n-a-d-a-r-k-o-_
Analog	A-n-a-l-o-g-_
Analyses	A-n-a-l-y-s-e-s-_
Analysis	A-n-a-l-y-s-i-s-<_
Analyst	A-n-a-l-y-s-t-<_
Analyst's	A-n-a-l-y-s-t-*ap*-s-<_
Analysts	A-n-a-l-y-s-t-s-<_
Analytical	A-n-a-l-y-t-i-c-a-l-<<_
Anchor	A-n-c-h-o-r-_
Ancient	A-n-c-i-e-n-t-<<_
And	A-n-d-_
Andean	A-n-d-e-a-n-_
Andersen	A-n-d-e-r-s-e-n-_
Anderson	A-n-d-e-r-s-o-n-_
Anderson's	A-n-d-e-r-s-o-n-*ap*-s-_
Andersson	A-n-d-e-r-s-s-o-n-_
Andes	A-n-d-e-s-_
Andover	A-n-d-o-v-e-r-_
Andrea	A-n-d-r-e-a-_
Andrew	A-n-d-r-e-w-_
Andrews	A-n-d-r-e-w-s-_
Andy	A-n-d-y-_
Angeles	A-n-g-e-l-e-s-_
Angeles's	A-n-g-e-l-e-s-*ap*-s-_
Angell	A-n-g-e-l-l-_
Angelo	A-n-g-e-l-o-_
Angels	A-n-g-e-l-s-_

Angier	A-n-g-i-e-r-<_
Anglia	A-n-g-l-i-a-<_
Anglian	A-n-g-l-i-a-n-<_
Anglo	A-n-g-l-o_
Angola	A-n-g-o-l-a_
Anheuser	A-n-h-e-u-s-e-r_
Animals	A-n-i-m-a-l-s-<_
Anita	A-n-i-t-a-<<_
Ankara	A-n-k-a-r-a_
Anku	A-n-k-u_
Ann	A-n-n_
Annalee	A-n-n-a-l-e-e_
Annapolis	A-n-n-a-p-o-l-i-s-<_
Anne	A-n-n-e_
Annenberg	A-n-n-e-n-b-e-r-g_
Anniston	A-n-n-i-s-t-o-n-<<_
Annual	A-n-n-u-a-l_
Annualized	A-n-n-u-a-l-i-z-e-d-<_
Annuities	A-n-n-u-i-t-i-e-s-<<<_
Annuity	A-n-n-u-i-t-y-<<_
Another	A-n-o-t-h-e-r-<_
Ansco	A-n-s-c-o_
Answers	A-n-s-w-e-r-s_
Antar	A-n-t-a-r-<_
Antar's	A-n-t-a-r-*ap*-s-<_
Antarctica	A-n-t-a-r-c-t-i-c-a-<<<_
Anthong	A-n-t-h-o-n-g-<_
Anthony	A-n-t-h-o-n-y-<_
Anthropology	A-n-t-h-r-o-p-o-l-o-g-y-<_
Anti	A-n-t-i-<<_
Anticipated	A-n-t-i-c-i-p-a-t-e-d-<<<<_
Antilles	A-n-t-i-l-l-e-s-<<_
Antinori's	A-n-t-i-n-o-r-i-*ap*-s-<<<_
Antitrust	A-n-t-i-t-r-u-s-t-<<<<_
Antoine	A-n-t-o-i-n-e-<<_
Antolini	A-n-t-o-l-i-n-i-<<<_
Anton	A-n-t-o-n-<_
Antonio	A-n-t-o-n-i-o-<<_
Anxiety	A-n-x-i-e-t-y-<<<_
Anxious	A-n-x-i-o-u-s-<<_
Any	A-n-y_
Anyone	A-n-y-o-n-e_
Anything	A-n-y-t-h-i-n-g-<<_
Anything's	A-n-y-t-h-i-n-g-*ap*-s-<<<_
Anyway	A-n-y-w-a-y_
Aoun	A-o-u-n_
Aoyama	A-o-y-a-m-a_
Apache	A-p-a-c-h-e_
Apart	A-p-a-r-t-<_
Apicella	A-p-i-c-e-l-l-a-<_
Appalachian	A-p-p-a-l-a-c-h-i-a-n-<_
Appalled	A-p-p-a-l-l-e-d_
Apparently	A-p-p-a-r-e-n-t-l-y-<_
Appeals	A-p-p-e-a-l-s_
Appelbaum	A-p-p-e-l-b-a-u-m_

Appell	A-p-p-e-l-l- __
Appellate	A-p-p-e-l-l-a-t-e-<- __
Applause	A-p-p-l-a-u-s-e- __
Apple	A-p-p-l-e- __
Applegate	A-p-p-l-e-g-a-t-e-<- __
Appleyard	A-p-p-l-e-y-a-r-d- __
Applications	A-p-p-l-i-c-a-t-i-o-n-s-<-<-<- __
Applied	A-p-p-l-i-e-d-<- __
Appropriations	A-p-p-r-o-p-r-i-a-t-i-o-n-s-<-<-<- __
April	A-p-r-i-l-<- __
Apt	A-p-t-<- __
Aptitude	A-p-t-i-t-u-d-e-<-<-<- __
Aquino	A-q-u-i-n-o-<- __
Aquitaine	A-q-u-i-t-a-i-n-e-<-<-<- __
Arab	A-r-a-b- __
Arabia	A-r-a-b-i-a-<- __
Arabian	A-r-a-b-i-a-n-<- __
Arabic	A-r-a-b-i-c-<- __
Arabs	A-r-a-b-s- __
Arafat	A-r-a-f-a-t-<- __
Araskog	A-r-a-s-k-o-g- __
Arbitrage	A-r-b-i-t-r-a-g-e-<-<- __
Arbitraging	A-r-b-i-t-r-a-g-i-n-g-<-<-<- __
Arbor	A-r-b-o-r- __
Arcata	A-r-c-a-t-a-<- __
Arch	A-r-c-h- __
Archibald	A-r-c-h-i-b-a-l-d-<- __
Archipelago	A-r-c-h-i-p-e-l-a-g-o-<- __
Archuleta	A-r-c-h-u-l-e-t-a-<- __
Arco	A-r-c-o- __
Arctic	A-r-c-t-i-c-<-<- __
Arden	A-r-d-e-n- __
Ardent	A-r-d-e-n-t-<- __
Ardent's	A-r-d-e-n-t-*ap*-s-<- __
Are	A-r-e- __
Area	A-r-e-a- __
Area's	A-r-e-a-*ap*-s- __
Areas	A-r-e-a-s- __
Aren't	A-r-e-n-*ap*-t-<- __
Argentina	A-r-g-e-n-t-i-n-a-<-<- __
Argentine	A-r-g-e-n-t-i-n-e-<-<- __
Argentinian	A-r-g-e-n-t-i-n-i-a-n-<-<-<- __
ArgoSystems	A-r-g-o-S-y-s-t-e-m-s-<- __
Argonne	A-r-g-o-n-n-e- __
Argus	A-r-g-u-s- __
Ariail	A-r-i-a-i-l-<-<- __
Arias	A-r-i-a-s-<- __
Arias's	A-r-i-a-s-*ap*-s-<- __
Arighi	A-r-i-g-h-i-<-<- __
Ariz	A-r-i-z-<- __
Arizona	A-r-i-z-o-n-a-<- __
Ark	A-r-k- __
Arkansas	A-r-k-a-n-s-a-s- __
Arkla	A-r-k-l-a- __
Arkla's	A-r-k-l-a-*ap*-s- __

Arkoma	A-r-k-o-m-a- <u> </u>
Arlington	A-r-l-i-n-g-t-o-n-<<-<- <u> </u>
Arm	A-r-m- <u> </u>
Armed	A-r-m-e-d- <u> </u>
Armenia	A-r-m-e-n-i-a-<- <u> </u>
Armenian	A-r-m-e-n-i-a-n-<- <u> </u>
Armstrong	A-r-m-s-t-r-o-n-g-<- <u> </u>
Armstrong's	A-r-m-s-t-r-o-n-g-*ap*-s-<- <u> </u>
Army	A-r-m-y- <u> </u>
Army's	A-r-m-y-*ap*-s- <u> </u>
Arnold	A-r-n-o-l-d- <u> </u>
Aronson	A-r-o-n-s-o-n- <u> </u>
Around	A-r-o-u-n-d- <u> </u>
Arraignments	A-r-r-a-i-g-n-m-e-n-t-s-<<-<- <u> </u>
Arrest	A-r-r-e-s-t-<- <u> </u>
Arrested	A-r-r-e-s-t-e-d-<- <u> </u>
Arrow	A-r-r-o-w- <u> </u>
Arroyo	A-r-r-o-y-o- <u> </u>
Arseneault	A-r-s-e-n-e-a-u-l-t-<- <u> </u>
Art	A-r-t-<- <u> </u>
Arthur	A-r-t-h-u-r-<- <u> </u>
Article	A-r-t-i-c-l-e-<<-<- <u> </u>
Articles	A-r-t-i-c-l-e-s-<<-<- <u> </u>
Artist	A-r-t-i-s-t-<<-<-<- <u> </u>
Artist's	A-r-t-i-s-t-*ap*-s-<<-<-<- <u> </u>
Artra	A-r-t-r-a-<- <u> </u>
Arts	A-r-t-s-<- <u> </u>
Arvind	A-r-v-i-n-d-<-<- <u> </u>
As	A-s- <u> </u>
Asada	A-s-a-d-a- <u> </u>
Asahi	A-s-a-h-i-<-<- <u> </u>
Ascii	A-s-c-i-i-<<-<-<- <u> </u>
Asea	A-s-e-a- <u> </u>
Ash	A-s-h- <u> </u>
Asher	A-s-h-e-r- <u> </u>
Ashurst	A-s-h-u-r-s-t-<-<- <u> </u>
Asia	A-s-i-a-<-<- <u> </u>
Asia's	A-s-i-a-*ap*-s-<-<- <u> </u>
Asian	A-s-i-a-n-<-<- <u> </u>
Asians	A-s-i-a-n-s-<-<-<- <u> </u>
Aside	A-s-i-d-e-<-<- <u> </u>
Asil	A-s-i-l-<-<- <u> </u>
Aska	A-s-k-a- <u> </u>
Asked	A-s-k-e-d- <u> </u>
Aslacton	A-s-l-a-c-t-o-n-<-<-<- <u> </u>
Assembly	A-s-s-e-m-b-l-y- <u> </u>
Assemblyman	A-s-s-e-m-b-l-y-m-a-n- <u> </u>
Assessment	A-s-s-e-s-s-m-e-n-t-<-<-<- <u> </u>
Asset	A-s-s-e-t-<-<- <u> </u>
Assets	A-s-s-e-t-s-<-<-<- <u> </u>
Assistant	A-s-s-i-s-t-a-n-t-<<-<-<-<- <u> </u>
Assoc	A-s-s-o-c- <u> </u>
Associated	A-s-s-o-c-i-a-t-e-d-<<-<-<- <u> </u>
Associates	A-s-s-o-c-i-a-t-e-s-<<-<-<-<- <u> </u>
Association	A-s-s-o-c-i-a-t-i-o-n-<<-<-<-<-<- <u> </u>

Association's	A-s-s-o-c-i-a-t-i-o-n-*ap*-s-<<<<_
Assume	A-s-s-u-m-e_
Assuming	A-s-s-u-m-i-n-g-<_
Assurance	A-s-s-u-r-a-n-c-e_
Astec	A-s-t-e-c-<_
Astoria	A-s-t-o-r-i-a-<<<_
At	A-t-<_
Ateliers	A-t-e-l-i-e-r-s-<<<_
Athletics	A-t-h-l-e-t-i-c-s-<<<<_
Athletics'	A-t-h-l-e-t-i-c-s-*ap*-s-<<<<_
Atkins	A-t-k-i-n-s-<<<_
Atlanta	A-t-l-a-n-t-a-<<<_
Atlanta's	A-t-l-a-n-t-a-*ap*-s-<<<_
Atlantic	A-t-l-a-n-t-i-c-<<<<_
Atsushi	A-t-s-u-s-h-i-<<<_
Attendants	A-t-t-e-n-d-a-n-t-s-<<<<_
Attention	A-t-t-e-n-t-i-o-n-<<<<<_
Attic	A-t-t-i-c-<<<<_
Attitudes	A-t-t-i-t-u-d-e-s-<<<<<_
Attorney	A-t-t-o-r-n-e-y-<<<_
Attorney's	A-t-t-o-r-n-e-y-*ap*-s-<<<<_
Attorneys	A-t-t-o-r-n-e-y-s-<<<<_
Attwood	A-t-t-w-o-o-d-<<<<_
Attwood's	A-t-t-w-o-o-d-*ap*-s-<<<<_
Auctions	A-u-c-t-i-o-n-s-<<<<_
Audit	A-u-d-i-t-<<<<_
Auditors	A-u-d-i-t-o-r-s-<<<<_
Audrey	A-u-d-r-e-y_
Audubon	A-u-d-u-b-o-n_
Aug	A-u-g_
August	A-u-g-u-s-t-<_
August's	A-u-g-u-s-t-*ap*-s-<<_
Augusta	A-u-g-u-s-t-a-<_
Augustines	A-u-g-u-s-t-i-n-e-s-<<<<_
Aurora	A-u-r-o-r-a_
Austin	A-u-s-t-i-n-<<<<_
Australia	A-u-s-t-r-a-l-i-a-<<<<_
Australia's	A-u-s-t-r-a-l-i-a-*ap*-s-<<<<_
Australian	A-u-s-t-r-a-l-i-a-n-<<<<_
Australians	A-u-s-t-r-a-l-i-a-n-s-<<<<_
Austria	A-u-s-t-r-i-a-<<<<_
Austrian	A-u-s-t-r-i-a-n-<<<<_
Author	A-u-t-h-o-r-<_
Authorities	A-u-t-h-o-r-i-t-i-e-s-<<<<<<_
Authority	A-u-t-h-o-r-i-t-y-<<<<<_
Auto	A-u-t-o-<_
Automated	A-u-t-o-m-a-t-e-d-<<<<_
Automatic	A-u-t-o-m-a-t-i-c-<<<<<_
Automobile	A-u-t-o-m-o-b-i-l-e-<<<<<_
Automotive	A-u-t-o-m-o-t-i-v-e-<<<<<<_
Autry	A-u-t-r-y-<_
Ave	A-v-e_
Avedisian	A-v-e-d-i-s-i-a-n-<<<<_
Avena	A-v-e-n-a_
Avenue	A-v-e-n-u-e_

Avenue's	A-v-e-n-u-e-*ap*-s- _
Average	A-v-e-r-a-g-e- _
Average's	A-v-e-r-a-g-e-*ap*-s- _
Avery	A-v-e-r-y- _
Avery's	A-v-e-r-y-*ap*-s- _
Avi	A-v-i-<- _
Aviation	A-v-i-a-t-i-o-n-<-<-<- _
Aviv	A-v-i-v-<- _
Aviva	A-v-i-v-a-<- _
Aviva's	A-v-i-v-a-*ap*-s-<- _
Avon	A-v-o-n- _
Avondale	A-v-o-n-d-a-l-e- _
Avrett	A-v-r-e-t-t-<-<- _
Aw	A-w- _
Ayer	A-y-e-r- _
Ayers	A-y-e-r-s- _
Azara's	A-z-a-r-a-*ap*-s- _
Azem	A-z-e-m- _
Azerbaijan	A-z-e-r-b-a-i-j-a-n-<-<-<- _
Aziza	A-z-i-z-a-<- _
Azoff	A-z-o-f-f- _
Aztec	A-z-t-e-c-<- _
Azucena	A-z-u-c-e-n-a- _

Appendix C: A Sampling of the Decoding Results for the Substroke Feature Experiment

“REF” denotes the reference transcription, and “HYP” denotes the corresponding hypothesized decoding result. The underline emphasizes the decoding errors.

aimr114

REF: Small wonder, since he's asking San Francisco taxpayers to sink up to \$100 million into the new stadium.

HYP: Small wonder, since he's asking San Francisco's taxpayers to sink up to \$10_ million into the new structure.

aimr118

REF: Something like one-third of the nation's 60 largest cities are thinking about new stadiums, ranging from Cleveland to San Antonio and St. Petersburg.

HYP: Something like one-third of the nation's 60 largest cities are thinking about new stadiums, ranging from Cleveland to San Antonio and Set. Petersburg.

aimr123

REF: Voters generally agree when they are given a chance to decide if they want to sink their own tax dollars into a new mega-stadium.

HYP: Voters generally agree when they are given a chance to decide if they want to sink their own tax dollars into a new mega-strain.

aimr127

REF: But voters decided that if the stadium was such a good idea someone would build it himself, and rejected it 59% to 41%.

HYP: But voters decided that if the stadium was such a good idea someone would issued it himself, and rejected it 39% to 42%.

dsfr107

REF: First Federal named Mr. Ottoni president and chief executive to succeed Mr. Clarkson and elected him as a director.

HYP: First Federal named Mr. Ottoni president and chief executive to succeed Mr. Clark seem and elected him as a directors.

dsfr120

REF: "An ultimate target of 40,000 units annually has been set for a still-to-be-named model," Toyota said.

HYP: "In ultimate target of 40,00_ units annually has been set for a still-to_be-naked model," Toyota said.

dsfr124

REF: Toyota also announced a number of major overseas purchases.

HYP: Toyota also announced a number of major overseas purchases.

dsfr130

REF: The trust has an option to convert its shares to a 71.5% equity stake in the center by the year 2000.

HYP: The trust has an option to convert its shares to a 47.5% equity stake in the center by the year 200_.

rgbr104

REF: "When scientific progress moves into uncharted ground, there has to be a role for society to make judgments about its applications," says Myron Genel, associate dean of the Yale Medical School.

HYP: "When scientific progress over into uncharted ground, there has to be a role for society to make judgments about its application," says Myron fuel, associate dean of the Yale medical school.

rgbr109

REF: Despite the flap over transplants, federal funding of research involving fetal tissues will continue on a number of fronts.

HYP: Despite the flap over transports, federal funding of research involving fetal tissues will continue on a number of fronts.

rgbr115

REF: Yesterday's share turnover was well below the year's daily average of 133.8 million.

HYP: Yesterday's share turnover was well below the year " a daily average of 13_.8 million.

rgbr120

REF: The bank stocks got a boost when Connecticut Bank & Trust and Bank of New England said they no longer oppose pending legislation that would permit banks from other regions to merge with Connecticut and Massachusetts banks.

HYP: The bank stocks got a court when Connecticut Bank & Trust and Bank of New England said ray no longer oppose pending legislature that would permit banks from other yen to merge with Connecticut and Massachusetts banks.

shsr108

REF: The borrowing to raise these funds would be paid off as assets of sick thrifts are sold.

HYP: The borrowing to raise these funds would be paid off as assets of sick thrifts are sold.

shsr112

REF: The RTC will have to sell or merge hundreds of insolvent thrifts over the next three years.

HYP: The RTC will have to sell or merge hundreds of insolvent thrifts over the next three years.

shsr118

REF: But the worst possibility would be raising no working capital, he said.

HYP: But the worst possibility would be raising no working capital, he said.

shsr124

REF: Not a gripping question, unless you're the pastry chef of this city's Chez Panisse restaurant and you've just lost your priceless personal dessert notebook.

HYP: But a gripping question, unless you in the pastry chef of this city's Chez Panisse restaurant and going just last year priceless personal dessert notebook.

slbr104

REF: This year's results included a gain of \$70.2 million on the disposal of seafood operations.

HYP: This year's results included a gain of \$70.2 million on one disposal of one good operations.

slbr110

REF: Its cereal division realized higher operating profit on volume increases, but also spent more on promotion.

HYP: Its cereal division realized higher operating profit on volume increases, but also spent menu on promotion.

slbr115

REF: The companies are followed by at least three analysts, and had a minimum five-cent change in actual earnings per share.

HYP: The companies are followed by at least three analysts, and had a minimum five-cent change in actual earnings per gram.

slbr137

REF: First Chicago Corp. said it completed its \$55.1 million cash-and-stock acquisition of closely held Ravenswood Financial Corp., another Chicago bank holding company.

HYP: First Chicago Corp. said it completed its \$5.6 million cash-and-stock acquisition of closely held Ravenswood Financial Corp., another Chicago bank holding company.

wcdr104

REF: Conversely, strong consumer spending in the U.S. two years ago helped propel the local economy at more than twice its current rate.

HYP: Conversely, strong consumer spending in the U.S. two years ago helped propel the local economy at more than twice its current rate.

wcdr117

REF: While Wall Street is retreating from computer-driven program trading, big institutional investors are likely to continue these strategies at full blast, further roiling the stock market, trading executives say.

HYP: While Wall Street is retreating from computer-driven program trading, big institutional investors are likely to continue these strategies at full blast, further roiling the stock market, trading executives say?

wcdr121

REF: Trading executives privately say that huge stock-index funds, which dwarf Wall Street firms in terms of the size of their program trades, will continue to launch big programs through the stock market.

HYP: Trading executives privately say that huge stock-index funds, which dwarf Wall Street firms in terms of the size of their program trades, will continue to launch big programs through the stock market.

wcdr125

REF: Consequently, abrupt swings in the stock market are not likely to disappear anytime soon.

HYP: Consequently, abrupt swings in the stock market are not likely to disappear anytime soon.

References

- [1] C. C. Tappert, "Adaptive on-line handwriting recognition," presented at The 7th International Conference on Pattern Recognition, 1984.
- [2] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The State of the Art in On-Line Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 787-808, 1990.
- [3] J. Makhoul and R. Schwartz, "State of the art in continuous speech recognition," *Proceedings of National Academy of Science*, vol. 92, pp. 9956-9963, 1995.
- [4] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1993.
- [5] R. Nag, K. H. Wong, and F. Fallside, "Script Recognition using Hidden Markov Models," presented at Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Tokyo, Japan, 1986.
- [6] J. R. Bellegarda, D. Nahamoo, K. S. Nathan, and E. J. Bellegarda, "Supervised Hidden Markov Modeling for On-Line Handwriting Recognition," presented at International Conference on Acoustics, Speech, and Signal Processing, 1994.
- [7] K. S. Nathan, J. R. Bellegarda, D. Nahamoo, and E. J. Bellegarda, "On-line Handwriting Recognition using Continuous Parameter Hidden Markov Models," presented at International Conference on Acoustics, Speech, and Signal Processing, 1993.
- [8] T. Starner, J. Makhoul, R. Schwartz, and G. Chou, "On-Line Cursive Handwriting Recognition Using Speech Recognition Methods," presented at International Conference on Acoustics, Speech, and Signal Processing, 1994.
- [9] J. Makhoul, T. Starner, R. Schwartz, and G. Chou, "On-Line Cursive Handwriting Recognition Using Hidden Markov Models and Statistical Grammars," presented at Human Language Technology Workshop, Plainsboro, NJ., 1994.
- [10] M. Schenkel, I. Guyon, and D. Henderson, "On-line Cursive Script Recognition using Time Delay Neural Networks and Hidden Markov Models," presented at International Conference on Acoustics, Speech, and Signal Processing, 1994.

- [11] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Ann. Math. Stat.*, vol. 37, pp. 1554-1563, 1966.
- [12] L. E. Baum and J. A. Egon, "An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology," *Bull. Amer. Meteorol. Soc.*, vol. 74, pp. 360-3363, 1967.
- [13] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Stat.*, vol. 41, pp. 164-171, 1970.
- [14] J. K. Baker, "The dragon system-An overview," *IEEE Trans. Acoustic Speech Signal Processing*, vol. ASSP-23, pp. 24-29, 1975.
- [15] F. Jelinek, "Continuous speech recognition by statistical methods," *Proceedings of IEEE*, vol. 64, pp. 5332-536, 1976.
- [16] L. R. Rabiner, J. G. Wilpon, and F. K. Soong, "High Performance Connected Digit Recognition Using Hidden Markov Models," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, 1988.
- [17] Y. L. Chow, M. O. Dunham, O. A. Kimball, M. A. Krasner, G. F. Kubala, J. Makhoul, S. Roucos, and R. M. Schwartz, "BYBLOS: The BBN Continuous Speech Recognition System," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, 1987.
- [18] K. F. Lee, "Large-vocabulary speaker-independent continuous speech recognition: The SPHINX system," in *Department of Computer Science: Carnegie-Mellon University*, 1988.
- [19] X. D. Huang, Y. Ariki, and M. A. Jack, *Hidden Markov Models for Speech Recognition*. Edinburgh: Redwood Press Limited, 1990.
- [20] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *IEEE Proceedings*, 1988.
- [21] A. W. Drake, "Discrete-state Markov processes," in *Fundamentals of Applied Probability Theory*. New York, NY: McGraw-Hill, 1967.

- [22] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the *EM* Algorithm," *Journal of the Royal Statistical Society*, vol. 39, pp. 1-21, 1977.
- [23] L. E. Baum and G. R. Sell, "Growth functions for transformations on manifolds," *Pac. J. Math.*, vol. 27, pp. 221-227, 1968.
- [24] D. Paul, "The Design for the Wall Street Journal-based CSR Corpus," presented at Proceedings of the DARPA Speech and Natural language Workshop, 1992.
- [25] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "UNIPEN project of on-line data exchange and benchmarks," presented at International Conference on Pattern Recognition, ICPR'94, Jerusalem, Israel, 1994.
- [26] F. Kubala, A. Anastasakos, J. Makhoul, L. Nguyen, R. Schwartz, and G. Zavaliagkos, "Comparative Experiments on Large Vocabulary Speech Recognition," presented at International Conference on Acoustics, Speech, and Signal Processing, 1994.
- [27] J. Makhoul, S. Roucos, and H. Gish, "Vector Quantization in Speech Coding," *Proceedings of the IEEE*, vol. 73, pp. 1551-1588, 1985.
- [28] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York, N. Y.: John Wiley & Sons, 1973.
- [29] S. Austin, R. Schwartz, and P. Placeway, "The Forward-Backward Search Algorithm," presented at IEEE International Conference on Acoustics, Speech, and Signal Processing, Toronto, Ontario, Canada, 1991.
- [30] L. Nguyen, R. Schwartz, Y. Zhao, and G. Zavaliagkos, "Is N-best Dead?," presented at ARPA Human Language Technology Workshop, Plainsboro, NJ, 1994.
- [31] R. Schwartz, L. Nguyen, and J. Makhoul, "Multiple-Pass Search Strategies," in *Advanced Topics in Automatic Speech and Speaker Recognition*, C.-H. Lee and F. K. Soong, Eds. Boston, MA: Kluwer, 1996, pp. 429-456.
- [32] W. Guerfali and M. V. Plamondon, "Normalizing and restoring on-line handwriting," *Pattern Recognition*, vol. 26, pp. 419-431, 1993.