Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1996-06

# Diphone-based speech recognition using neural networks

Cantrell, Mark E.

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/32063

# NAVAL POSTGRADUATE SCHOOL
## MONTEREY, CALIFORNIA

# THESIS

DIPHONE-BASED SPEECH RECOGNITION USING
NEURAL NETWORKS

by

Mark E. Cantrell

June, 1996

Coadvisors:
Dan C. Boger
Robert B. McGhee

**Approved for public release; distribution is unlimited**

| REPORT DOCUMENTATION PAGE | | Form Approved OMB No. 0704-0188 |
|---|---|---|

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

| 1.AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE JUNE 1996 | 3.REPORT TYPE AND DATES COVERED Master's Thesis | |
|---|---|---|---|
| 4.TITLE AND SUBTITLE DIPHONE-BASED SPEECH RECOGNITION USING NEURAL NETWORKS | | | 5.FUNDING NUMBERS |
| 6. AUTHOR(S)   Cantrell, Mark E. | | | |
| 7.PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000 | | | 8.PERFORMING ORGANIZATION REPORT NUMBER |
| 9.SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | | 10.SPONSORING/MONITORING AGENCY REPORT NUMBER |

11.SUPPLEMENTARY NOTES
 The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 12a.  DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. | 12b.DISTRIBUTION CODE |
|---|---|

13.

Speaker-independent automatic speech recognition (ASR) is a problem of long-standing interest to the Department of Defense. Unfortunately, existing systems are still too limited in capability for many military purposes. Most large-vocabulary systems use phonemes (individual speech sounds, including vowels and consonants) as recognition units. This research explores the use of diphones (pairings of phonemes) as recognition units. Diphones are acoustically easier to recognize because coarticulation effects between the diphone's phonemes become recognition features, rather than confounding variables as in phoneme recognition. Also, diphones carry more information than phonemes, giving the lexical analyzer two chances to detect every phoneme in the word. Research results confirm these theoretical advantages. In testing with 4490 speech samples from 163 speakers, 70.2% of 157 test diphones were correctly identified by one trained neural network. In the same tests, the correct diphone was one of the top three outputs 89.0% of the time. During word recognition tests, the correct word was detected 85% of the time in continuous speech. Of those detections, the correct diphone was ranked first 41.6% of the time and among the top six 74% of the time. In addition, new methods of pitch-based frequency normalization and network feedback- based time alignment are introduced. Both of these techniques improved recognition accuracy on male and female speech samples from all eight dialect regions in the U.S. In one test set, frequency normalization reduced errors by 34%. Similarly, feedback-based time alignment reduced another network's test set errors from 32.8% to 11.0%.

| 14.SUBJECT TERMS Automatic speech recognition, diphone, neural network, speaker independent, continuous speech | | | 15.NUMBER OF PAGES 357 |
|---|---|---|---|
| | | | 16.PRICE CODE |
| 17.SECURITY CLASSIFICATION OF REPORT Unclassified | 18.SECURITY CLASSIFICATION OF THIS PAGE Unclassified | 19.SECURITY CLASSIFICATION OF ABSTRACT Unclassified | 20.LIMITATION OF ABSTRACT UL |

i

# DIPHONE-BASED SPEECH RECOGNITION USING NEURAL NETWORKS

Mark E. Cantrell
Major, United States Marine Corps
B.S., Oregon State University, 1982

Submitted in partial fulfillment
of the requirements for the degrees of

## MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY

and

## MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

## NAVAL POSTGRADUATE SCHOOL
## June 1996

Author: _____

Mark E. Cantrell

Approved by:_____

Dan C. Boger, Coadvisor

_____

Robert B. McGhee, Coadvisor

_____

Dan C. Boger, Chairman Command Control &
Communications Academic Group

_____

Ted Lewis, Chairman Department of Computer Science

# ABSTRACT

Speaker-independent automatic speech recognition (ASR) is a problem of long-standing interest to the Department of Defense. Unfortunately, existing systems are still too limited in capability for many military purposes. Most large-vocabulary systems use phonemes (individual speech sounds, including vowels and consonants) as recognition units. This research explores the use of diphones (pairings of phonemes) as recognition units. Diphones are acoustically easier to recognize because coarticulation effects between the diphone's phonemes become recognition features, rather than confounding variables as in phoneme recognition. Also, diphones carry more information than phonemes, giving the lexical analyzer two chances to detect every phoneme in the word. Research results confirm these theoretical advantages. In testing with 4490 speech samples from 163 speakers, 70.2% of 157 test diphones were correctly identified by one trained neural network. In the same tests, the correct diphone was one of the top three outputs 89.0% of the time. During word recognition tests, the correct word was detected 85% of the time in continuous speech. Of those detections, the correct diphone was ranked first 41.6% of the time and among the top six 74% of the time. In addition, new methods of pitch-based frequency normalization and network feedback- based time alignment are introduced. Both of these techniques improved recognition accuracy on male and female speech samples from all eight dialect regions in the U.S. In one test set, frequency normalization reduced errors by 34%. Similarly, feedback-based time alignment reduced another network's test set errors from 32.8% to 11.0%.

# TABLE OF CONTENTS

x

# ACKNOWLEDGMENT

The author is grateful to Professors Dan Boger and Robert McGhee for their advice,

encouragement, and patience during this research. Also, this work would not have been possible

without the generous assistance with computing resources provided by James Schmit and other

computing staff members at the Naval Postgraduate School. Finally, the author thanks Patricia,

Brittany, and Nicholas Cantrell for their help with the illustrations, as well as Natalie and

Douglas for their tolerance.

# EXECUTIVE SUMMARY

The Department of Defense (DoD) has long recognized the potential value of automatic speech recognition. In fact, the DoD, often acting through the Defense Advanced Research Projects Agency (DARPA), has funded or facilitated much of the most significant speech recognition research over the past few decades. As command, control, communications, computer, and intelligence (C4I) systems continue to swell in importance, speech recognition is likely to assume an increasingly important role. Unfortunately, existing systems are still too limited in capability for many military purposes.

Most large-vocabulary systems use phonemes (individual speech sounds, including vowels and consonants) as recognition units. This architecture may place an upper limit on the performance of these systems. This research explores the use of diphones (pairings of phonemes) as recognition units. Diphones are acoustically easier to recognize because coarticulation effects between the diphone's phonemes become recognition features, rather than confounding variables as in phoneme recognition. Also, diphones carry more information than phonemes, giving higher levels in the recognition system two chances to detect every phoneme in the word. Research results confirm these theoretical advantages.

In testing with 4490 speech samples from 163 speakers, 70.2% of 157 test diphones were correctly identified by one trained neural network. In the same tests, the correct diphone was one of the top three outputs 89.0% of the time. During word recognition tests, the correct word was detected 85% of the time in continuous speech. Of those detections, the correct diphone was ranked first 41.6% of the time and among the top six 74% of the time. In addition, new methods of pitch-based frequency normalization and network feedback- based time alignment are

introduced. Both of these techniques improved recognition accuracy on male and female speech samples from all eight dialect regions in the U.S. In one test set, frequency normalization reduced errors by 34%. Similarly, feedback-based time alignment reduced another network's test set errors from 32.8% to 11.0%.

These results imply that diphones may indeed be suitable as recognition units for large vocabulary, speaker independent, continuous speech recognition systems.

# I. INTRODUCTION

## A. SPEECH RECOGNITION

Science fiction writers seldom portray the keyboard as the computer input device of the future. Instead, futurists have long predicted that humans will one day interact with their computers through speech. Presumably, that interaction will be two-way (and the human will be in charge). If so, the computer will be performing both *speech synthesis* and *automatic speech recognition (ASR)*. The latter process is the subject of the research described in this thesis. Specifically, this research involves evaluation of the diphone (defined in Chapter II) as a recognition unit for ASR systems.

For the purposes of this research, automatic speech recognition is the process of translating the human speech signal from its acoustic form into more something meaningful to a computer, such as ASCII text. Current ASR systems can be classified with respect to the following characteristics:

- *Speaker independence* - speaker independent systems are designed to recognize speech for a broad class of speakers (e.g., adult American males), while speaker dependent systems must be trained by the intended user.

- Vocabulary size - actual numbers, when given, depend on the author. But a small vocabulary system would probably recognize from a few to a few hundred words while large vocabulary systems should recognize tens of thousands of words. A medium vocabulary would fall somewhere in between.

- *Discrete* versus *continuous speech* - in normal (continuous) human speech the words tend to flow together with little, if any, silence between. Discrete-utterance recognition systems require that the speaker pause between words. Naturally, continuous speech recognition systems would be easier to use. But they are harder to build.

Ideally, a speech recognition system would be speaker independent, have a large vocabulary, and be capable of recognizing continuous speech. Unfortunately, for reasons to be discussed later, current systems essentially force us to choose just one of those three characteristics.

## B. THE IMPORTANCE OF SPEECH RECOGNITION

It is not difficult to imagine applications for ASR. Some of the more obvious include:

- Voice dictation and transcription - doctors and dentists could record their observations while examining their patients. Fewer details would be forgotten and the results would be legible.

- Remote control - everything from alarm clocks to robots could be controlled without hunting for a hand-held transmitter.

- Machine operation in hostile environments - where gloves would otherwise interfere.

- Assistance for the handicapped - quadriplegics could operate wheelchairs and the deaf attend lectures.

- Friendlier user interfaces - databases and computer expert systems would be more accessible to less technically-oriented users.

- Public information systems - Voice-operated telephone switchboards could replace touch-tone menu systems and computers could "man" information booths where (or when) humans are too costly.

- Cockpit systems - pilots could operate avionics and weapons without removing their hands from the throttle and stick.

- Military intelligence systems - computers could use word spotting algorithms to sift through monitored communications, flagging only the most useful for further study by human translators.

There are, of course, many other potential uses for speech recognition technology. Many applications will probably only come to light when more capable systems are finally developed.

## C. RELEVANCE TO THE DEPARTMENT OF DEFENSE

The Department of Defense (DoD) has long recognized the potential value of automatic speech recognition. In fact, the DoD, often acting through the Defense Advanced Research Projects Agency (DARPA), has funded or facilitated much of the most significant speech recognition research over the past few decades. As command, control, communications, computer, and intelligence (C4I) systems continue to swell in importance, speech recognition is likely to assume an increasingly important role.

## D. SCOPE OF RESEARCH

Automatic speech recognition, as discussed in more depth later, is a more challenging goal than one might expect. Therefore, as is so often the case with complex problems, ASR is better approached as a series of smaller and more manageable subproblems. Specifically, existing systems typically operate at acoustic, lexical, syntactic, and (to a lesser degree) semantic and pragmatic levels . *Acoustic analysis* consists of translating the acoustic waveform into a series of speech symbols, such as phonemes (individual speech sounds, discussed in more detail later). *Lexical analysis* consists of translating the speech symbols into word hypotheses. *Syntactic analysis* uses the rules of sentence structure to choose from among the words hypothesized by the lexical analyzer. The *semantic* level uses knowledge about the domain of discourse to refine the sentence hypotheses provided by lower levels. Finally, *pragmatic analysis* involves using world knowledge to achieve what can best be called "understanding" of the utterance. The last two levels, semantic and pragmatic, clearly require what can best be termed *artificial intelligence*. Thus, except for rudimentary forms (usually domain specific) of semantic analysis, satisfactory implementations of the last two levels are likely to remain distant goals. [Ref. 1]

This research primarily involves ASR at the acoustic and lexical levels, although the other levels are considered and receive some mention. The following specific research questions are addressed:

- Is the diphone a suitable choice as the recognition unit (speech symbol) for ASR systems?

- Can artificial neural networks recognize diphones in the acoustic signal?

- What can be done to improve recognition accuracy?

- Will accuracy in identifying the selected recognition unit translate into high word recognition accuracy?

## E.    ORGANIZATION OF THESIS

Following this introductory chapter, background information is provided on speech recognition and neural networks in Chapter II. The research methodology used, and the rationale behind it, is covered in Chapter III. The results are summarized in Chapter IV and conclusions in Chapter V. Finally, Chapter VI offers recommendations for further research.

## II. BACKGROUND

## A.   INTRODUCTION

One cannot hope to build a working speech recognition system without knowing something about speech.  Nor can one hope to use neural networks effectively without knowing something about how they work.  Therefore, this chapter will provide a brief overview of human speech and the challenges it presents.  Then, the fundamentals of back-propagation neural networks will be introduced.  The research methodology and research results, presented in later chapters, should make more sense in the light of this information.

## B.   SPEECH RECOGNITION

### 1. Production Mechanisms

Figure 2.1 is a simplified block diagram of the human speech production system [after Ref. 1].  *Voiced speech* is produced by forcing air from the lungs through the vocal folds (or cords).  The air causes the vocal folds to vibrate and produce sounds dominated by a fundamental frequency, perceived as the speaker's *pitch*.  The sound of the vocal fold vibrations must then pass through several cavities which serve as resonance chambers, reinforcing some frequencies while suppressing others.  Finally, the air and the sound must pass several speech-related organs, known as *articulators*, which further modify the speech waveform.  Voicing, also known as *phonation*, produces all of the *vowels* and is involved directly in some of the consonants. *Semivowels* (*liquids* such as the "w" in "wet" and *glides* such as the "y" in "yam") are essentially formed by transitioning between vowel-like sounds.

*Consonants*, some voiced and some unvoiced, are also produced by forcing air past the articulators and resonance chambers of the vocal tract.  But the vocal folds remain silent in the case of unvoiced consonants.  For example, unvoiced *fricatives* (such as the "s" in "lease") are produced by forcing air through some constriction in the vocal tract other than the vocal folds (such as a constriction formed between the tongue and teeth).  *Plosives*, also known as *stops*, are produced by building air pressure behind a constriction and then suddenly releasing the pressure to produce sound.  The "t" and "p" in "stop" are both plosives.  The *nasals* (such as the "n" in

5

"noise") are produced through a combination of phonation and opening of the velum so that the sound passes through both oral and nasal cavities. Finally, *affricates* ("ch" in "change" and "j" in "jam") are formed by transitions between stops and fricatives.
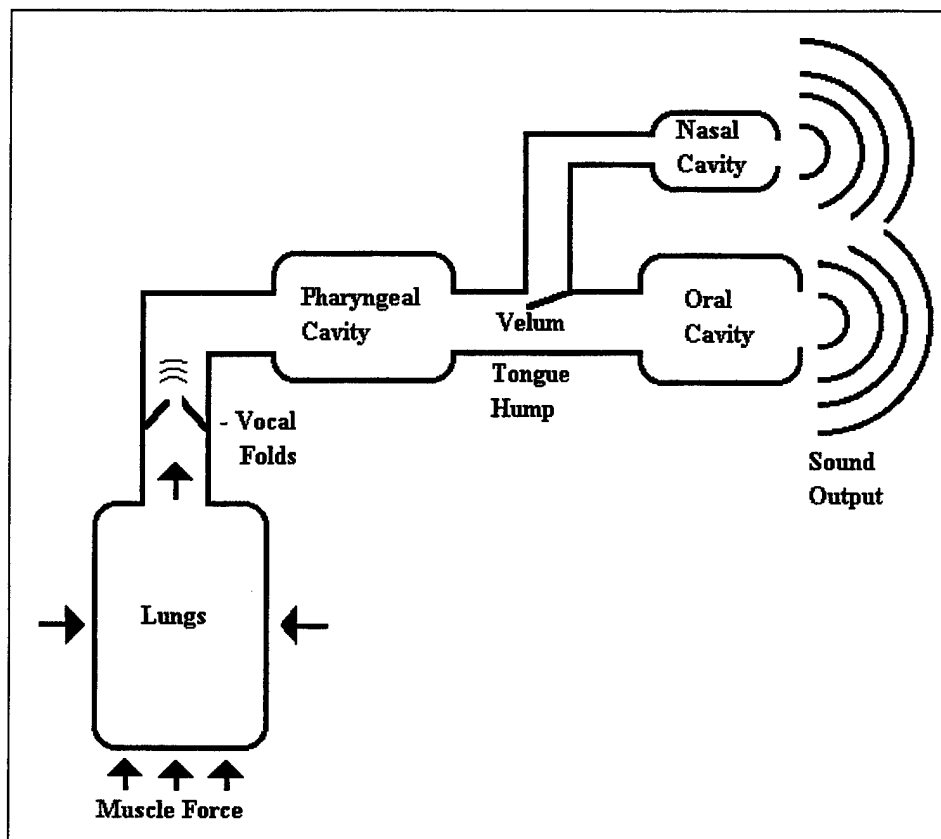


Figure 2.1. A Block Diagram of the Speech Production System [after Ref. 1].

The classification of speech sounds by their method of production is useful because the speech signal contains acoustic clues as to the method of a sound's production. Those clues may, in turn, be used to determine what words were spoken.

Unfortunately, conventional alphabets are unsuitable for unambiguously referring to particular speech sounds. So phoneticians have defined a set of atomic speech units known as *phonemes**, as well as several phonetic alphabets for referring to those phonemes. The *International Phonetic Alphabet* (IPA), developed by a group of European phoneticians in 1888, has enough symbols to describe all of the phonemes in all of the world's languages. But the IPA

---

\* Phoneticians actually distinguish between the phoneme, a theoretical sound unit, and phones, the sounds actually produced by a speaker. Associated with each phoneme is a set of phones, called allophones, which constitute acoustics variations of that phoneme. The term phoneme is often loosely used, in this thesis and elsewhere, to refer to all three speech units.

is most suited to hand transcription, since it is not practical for use with conventional keyboards and printers. In the United States, the *ARPAbet* (developed under the auspices of ARPA) is more widely used. The version of the uppercase ARPAbet used in the TIMIT lexicon (described later) is reproduced in Table 2.1. The symbols listed in Table 2.1 have been used (in bold upper case letters) to refer to specific phonemes in the remainder of this thesis.

| CLASS | SYMBOL | EXAMPLE | CLASS | SYMBOL | EXAMPLE |
|---|---|---|---|---|---|
| Stops: | B | Bee | Glides: | R | Ray |
| | D | Day | | Y | Yacht |
| | G | Gay | Africates: | JH | Joke |
| | P | Pea | | CH | CHoke |
| | T | Tea | Vowels: | IY | bEEt |
| | K | Key | | IH | bIt |
| Fricatives: | S | Sea | | EH | bEt |
| | SH | She | | EY | bAIt |
| | Z | Zone | | AE | bAt |
| | ZH | aZure | | AA | bOttom |
| | F | Fin | | AW | abOUt |
| | TH | THin | | AY | bIte |
| | V | Van | | AH | bUt |
| | DH | THen | | AO | bOUGHt |
| | M | Mom | | OY | bOY |
| | N | Noon | | OW | bOAt |
| | NG | siNG | | UH | bOOk |
| | EM | bottOM | | UW | bOOt |
| | EN | buttON | | ER | bIRd |
| | ENG | washINGton | | AX | abOUt |
| Semivowel: | EL | bottLE | | IX | debIt |
| Liquids: | L | Lay | | AXR | bUtter |
| | W | Way | | | |

Table 2.1. The Timit Lexicon's Phonetic Alphabet [after Ref. 2].

Before leaving the topic of speech production, it is useful to look at speech production from the communications system perspective (as suggested by Parsons [Ref. 3]). Specifically,

7

the lungs and vocal folds provide excitation (the carrier) and the speech articulators provide modulation. As with communications systems, the information content is in the modulation produced by the articulators, not the excitation produced by the vocal folds.

## 2. Characteristics of the Speech Signal

The vocal fold vibrations mentioned earlier do not produce anything like a clean sinusoidal waveform. Instead, the fundamental frequency is accompanied by many harmonics. Articulator effects and time-varying changes in the size and shape of resonance chambers further complicate the signal. Figures 2.2 and 2.3, showing just a few seconds of speech, illustrate how enormously complex the speech waveform is. The time-varying nature of the signal's amplitude is obvious from the speech "envelope" in Figure 2.2. But Figure 2.3 shows that the frequency varies widely with time as well.
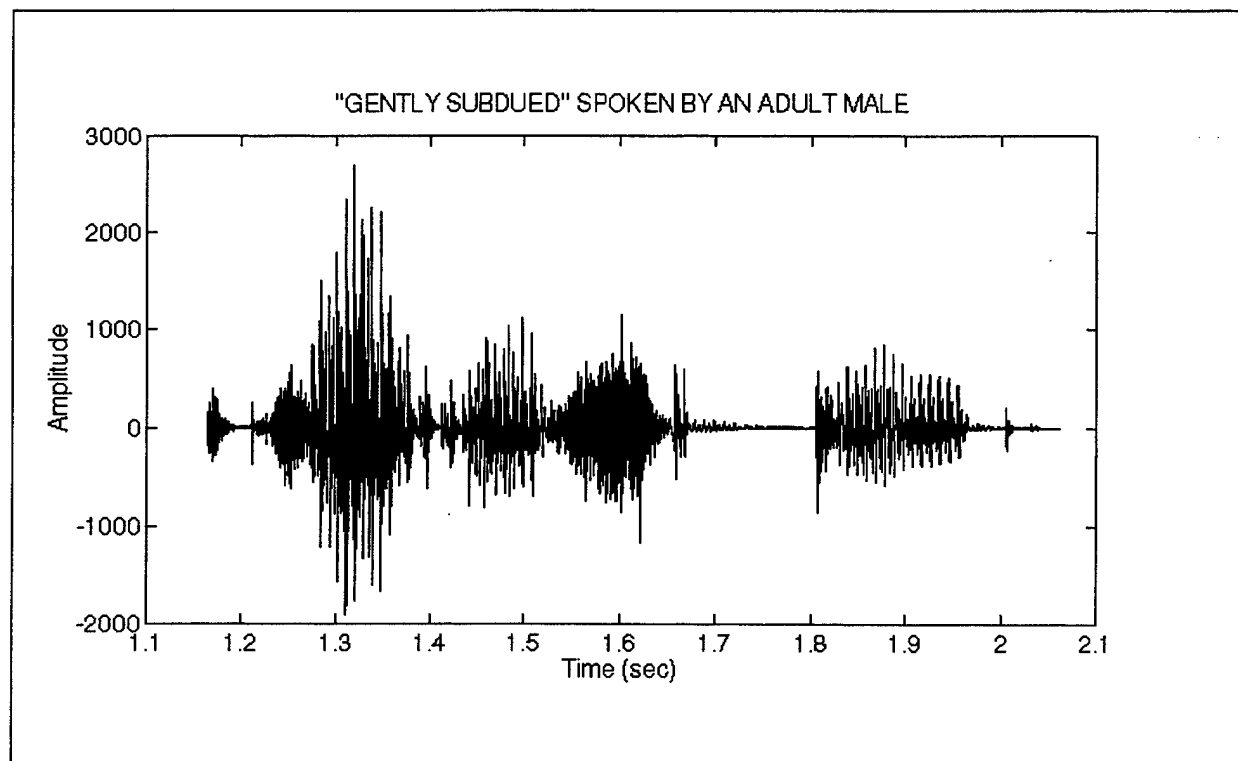


Figure 2.2. An Example of Digitized Speech

8
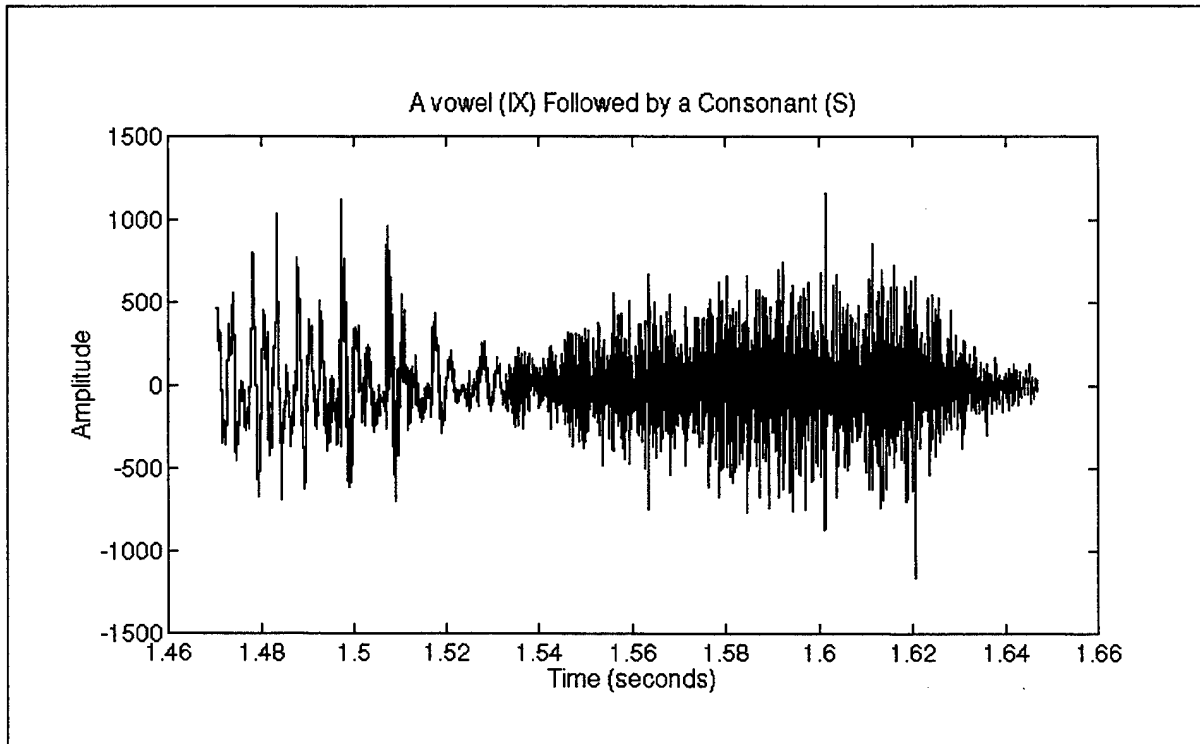
Figure 2.3. Frequency Variations with Time (the **IX S** sound, as in "aeglS")

Unfortunately, the time-domain view of speech (shown in Figures 2.2 and 2.3) is of limited use in speech recognition. For example, it may appear at first glance that four words were recorded in Figure 2.2. But the waveform includes only two words. And the word break is at 1.53 seconds, rather than at 1.75 seconds, as one might expect. A stop consonant (**B**) resulted in a longer silence (1.69 to 1.81 seconds) than is found at the actual junction between words.

Since the time domain is so troublesome, it is natural to look to the frequency domain. The invention of the spectrograph at Bell Laboratories in 1941 first made this practical [Ref. 3]. Today, Pulse Code Modulation (another Bell Labs invention) and Fast Fourier Transforms (FFTs) allow digital signal processors and computers to, in real time, determine the frequency components present in a speech waveform. Figure 2.4, produced using MATLAB's FFT function, shows the spectral characteristics of 20 milliseconds of voiced human speech (the vowel **IY**, as in bEEt, uttered by an adult male). Note the high frequency *combing* produced by the vocal folds, and the three prominent peaks in the spectral envelope. The largest peak, and the one with the lowest frequency, is at 300 Hz in Figure 2.4. This first peak is known as the *first formant* (or F1). Its position, when present, is determined by the degree of oral or pharyngeal

9

constriction of the vocal tract. The *second formant* (F2) is at about 2200 Hz in Figure 2.4. When it is present, the second formant's position is determined by the degree of constriction produced by the front or back of the tongue. The position of the *third formant* (at 3000 Hz in the same figure, a partial *blend* with F2), when present, is largely determined by vocal tract length. The frequencies of all three formants are inversely proportional to vocal tract length and any lip-rounding during articulation. [Ref. 1]
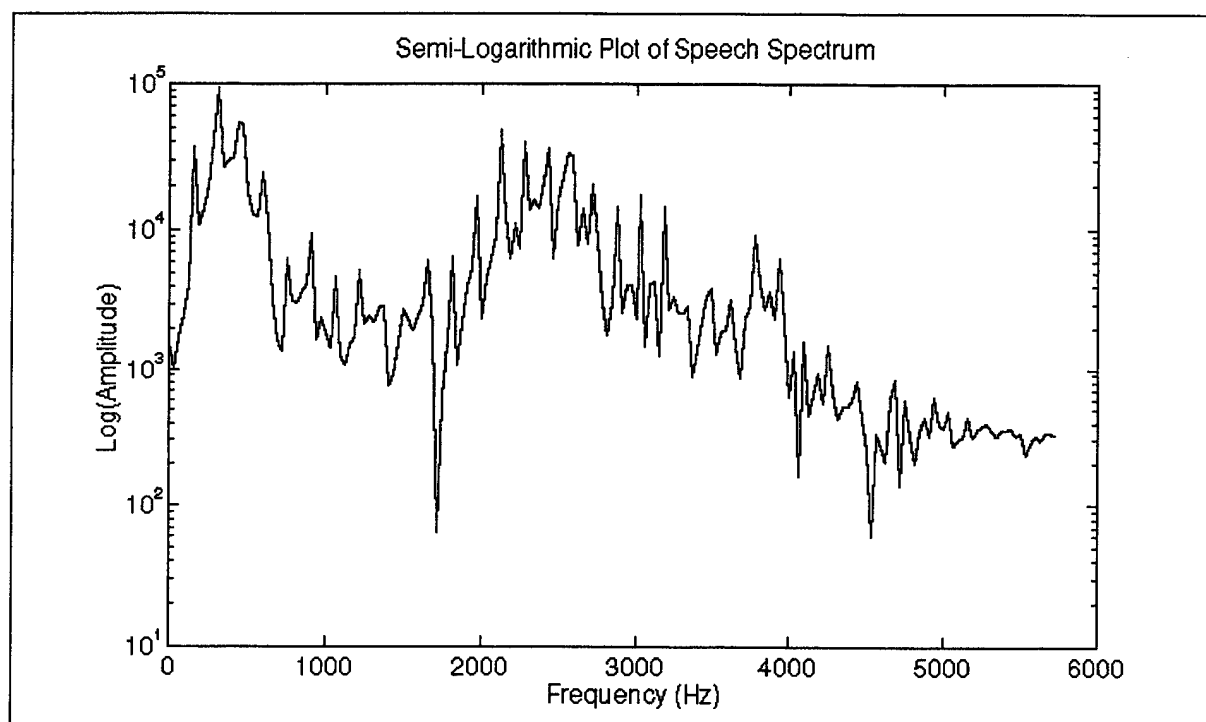


Figure 2.4. Frequency Components of a Vowel (**IY** as in bEEt) Spoken by a Male

Theoretically, the speech signal contains an infinite number of formants, just as it contains an infinite number of harmonics. But the higher formants become increasingly weak and the first three formants (particularly the first two) carry the bulk of speech information. Similarly, speech can include frequency components throughout the range of human hearing (about 15-20,000 Hz [Ref. 4]). But most of the information is at the lower frequencies. This is evidenced by our ability to recognize speech over telephone systems that have been band-limited to between 300 and 3000 Hz.

Because articulator movements affect formant frequencies, spectral information provides important speech recognition clues, at least for vowels. In fact, the frequencies and intensities of the first three formants can be used to identify vowels. Figure 2.5 (after Ref. 1) shows the average formant locations for vowels in American English.
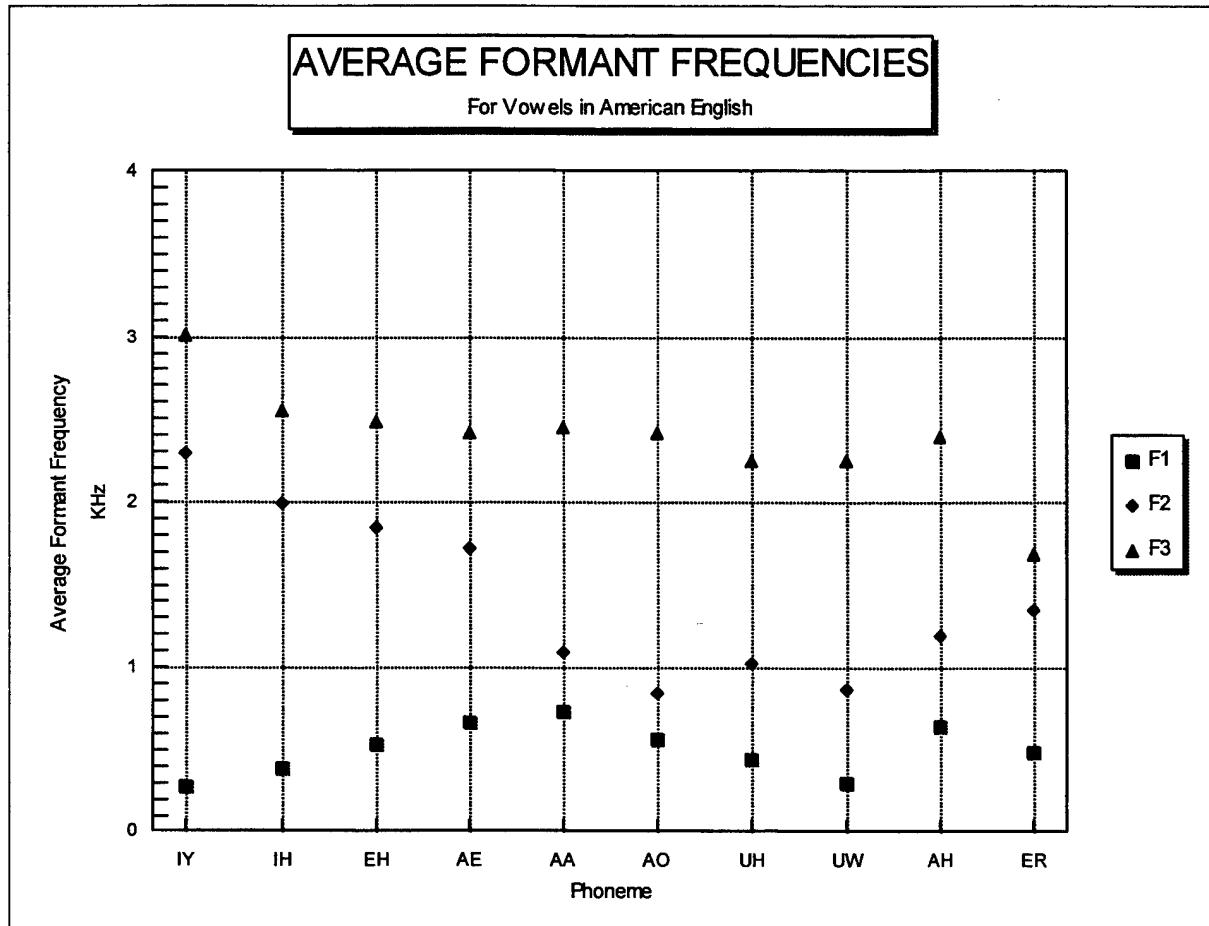


Figure 2.5. Average Formant Frequencies for Vowels in American English [after Ref. 1].

Unfortunately, the unvoiced consonants are more difficult to identify from their spectral characteristics. Figure 2.6, a spectrogram of the S consonant, illustrates this fact. The spectral envelope lacks the formant peaks characteristic of voiced sounds. Instead, the spectrum is dominated by higher frequencies (a noise-like hissing in the case of an S). In fact, the best clues as to the identity of a consonant are often found in the formant transitions of neighboring vowels [Ref. 3,5]. To a lesser extent, the speech envelope in the time domain is also of some use in consonant identification. For example, the *stop gap* in Figure 2.2 is a clue that a plosive follows. These facts will become important in later discussions of the recognition unit and feature vectors.

11
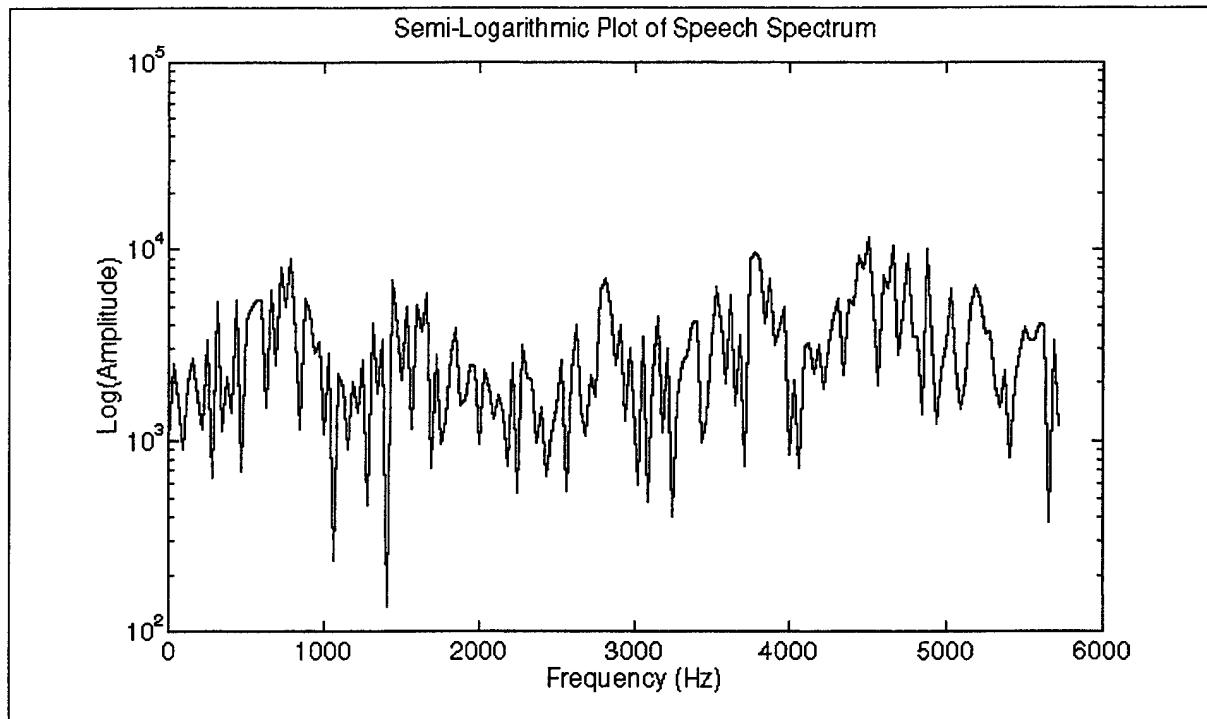
Figure 2.6. Frequency Spectrum of a Consonant (S) Spoken by a Male

As further evidence of the value of frequency domain analysis, it should be noted that at least one researcher, Victor Zue, learned to read spectrograms with greater than 90 percent accuracy [Ref. 3]. Furthermore, he was able to teach others to do the same.

## 3. Human hearing

When setting out to build an automatic speech recognition system, it is natural to ask how humans perform the same task. In particular, how do humans perceive pitch or frequency? In an effort to answer this question, two major theories were developed in the 19th century. The place theory, advanced by Helmholz, suggested that frequencies are encoded by the place of activation of hair cell receptors in the *cochlea* (the spiral-shaped tube in Figure 2.7). Rutherford's frequency theory, on the other hand, suggested that frequency is encoded by the frequency of discharge of neurons in the auditory system. The issue was partially settled by Georg von Bekesy, who was awarded the Nobel Prize in 1962 for his work. Sound produces vibrations of the *tympanic membrane* (eardrum), which are transferred by tiny bones (the *ossicles*) to the *oval window* at one end of the cochlea. In a famous series of experiments, von Bekesy showed that vibrations of the

12
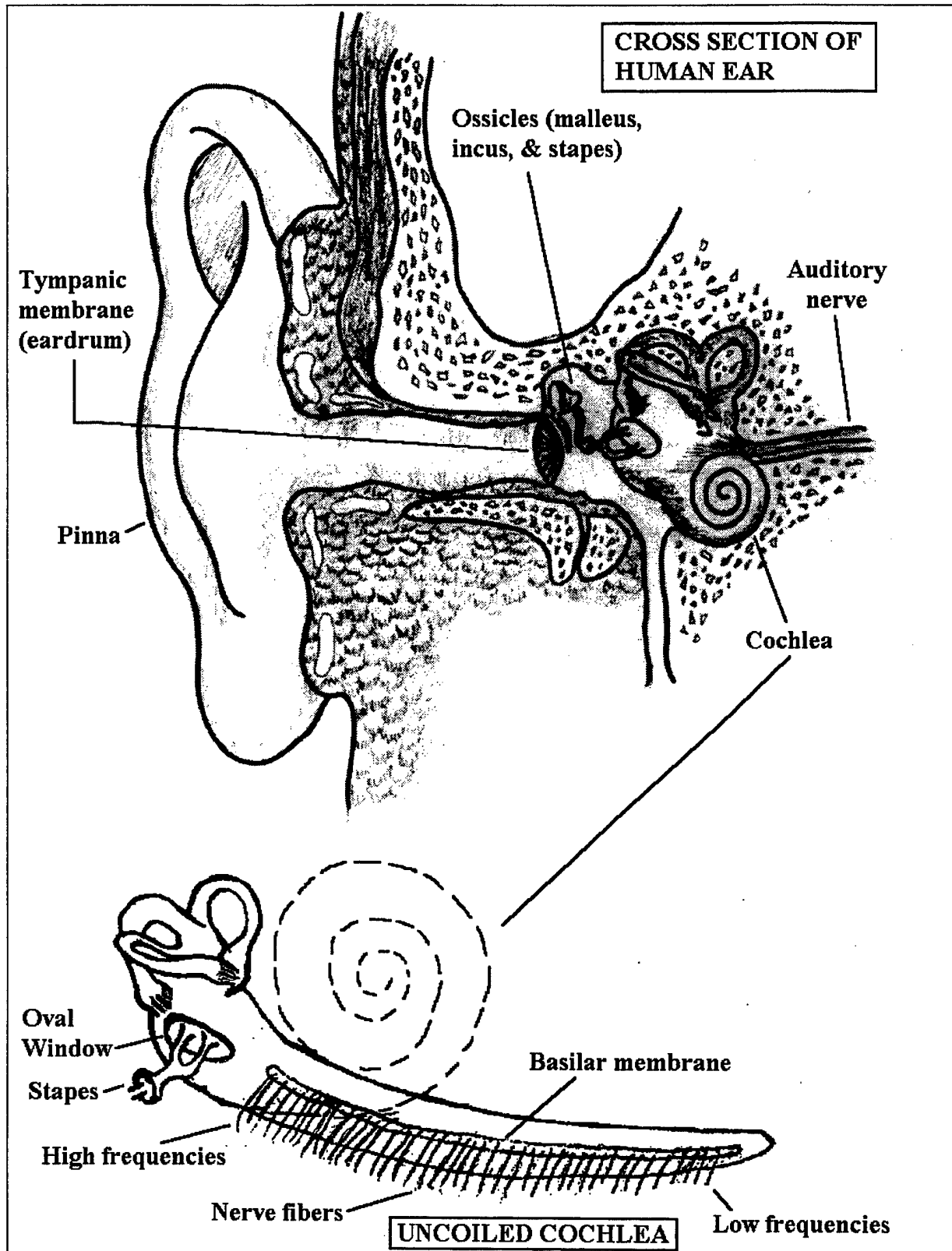
Figure 2.7. Cross-Section of the Human Ear and Cochlea [after Ref. 21].

13

oval window produce standing waves in the fluid filled cochlea. These standing waves activate tiny hair cell receptors in the basilar membrane that runs the length of the cochlea. The sound's frequency determines the point of maximum displacement of the basilar membrane. Higher frequencies, for example, cause standing waves with maximum amplitude very close to the oval window. Low frequencies activate nerves far from the oval window. The place theory is given further credence by experiments showing that individual neurons are incapable of firing more than about 1000 times per second. Yet humans are capable of resolving frequencies up to about 20,000 Hz. So, although some research indicates that the frequency theory is valid below about 1000 Hz, the place theory seems to best explain perception of higher frequencies. [Ref. 3,4]

Although we now know quite a bit about the function of the ear itself, much less is known about the higher levels in the auditory system. Unfortunately, the inner ear and auditory processing centers of the brain are surrounded by the skull, brain tissues, and other critical structures that make studies on living patients very difficult. However, some of the research that has been done implies that the auditory system includes *feature detectors* similar to those discovered in the (more accessible) visual cortex of the brain. The visual cortex includes simple feature detectors, such as edge detectors (neurons) that respond to edges with a particular orientation in a specific portion of the visual field. The outputs of regions of these neurons feed into complex neurons. The outputs of complex neurons feed into hypercomplex and higher-order hypercomplex neurons. At each stage of processing, the neurons are responsible for recognizing more complex patterns over wider regions of the retina. Researchers at Harvard even located a neuron in a monkey's brain that responded only to the shape of a hand. [Ref. 4]

If we assume that the auditory cortex operates like the visual cortex, it should contain feature detectors. Some feature detectors might respond to falling pitch, some to rising pitch, and others to temporal combinations of both. A number of studies support these assumptions. It is also interesting to note that animals with damaged auditory cortexes retain the ability to recognize pure tones but lose the ability to recognize more complex sounds (such as melodies). [Ref. 6]

It is encouraging to learn of the ear's ability to encode speech frequencies at such a low level in the auditory system. This ability implies that the frequency domain may play a key role

in human speech decoding. What we know of the auditory cortex further supports this theory. In any case, the frequency domain has been used extensively in speech research.

Before leaving the topic of human hearing, it would be worthwhile to briefly discuss its selectivity. As mentioned earlier, humans are capable of detecting tones from about 20 to 20,000 Hz. But the ear is most sensitive to some frequencies than others. Research has shown that the ear is more sensitive to frequencies between about 500 and 5000 Hz [Ref. 3]. Perhaps not coincidentally, this is also the range where most of the information content of speech resides.

In addition to nonuniform sensitivity, humans exhibit nonuniform pitch perception. For example, Middle C on the piano is 261.6 Hz while the C one octave higher is at twice that frequency, 523.2 Hz. Yet the second C does not sound like twice the pitch [Ref. 4]. More will be said of this in Chapter III. For now it is probably enough to say that humans can detect very slight changes in frequency below about 1000 Hz. But absolute sensitivity to a change in frequency falls off approximately logarithmically at higher frequencies. Again, the increased sensitivity in the speech band is interesting.

### 4. Why Speech Recognition is Difficult

The information presented so far might lead one to believe that speech recognition is a relatively straightforward process. It would seem that the FFT and an expanded version of Figure 2.5 could turn speech recognition into a fairly simple pattern recognition problem. In fact, this illusion of simplicity has been deceiving researchers, and those who fund research, since at least the 1930s. According to the brief history presented by Parsons [Ref. 3], the earliest serious attempts to build speech recognition hardware were part of an effort to replace human switchboard operators in the rapidly expanding phone system. When this effort failed to produce a system that could recognize digits spoken by a variety of people, the rotary dialing and switching system was developed. In the 1950s, many incorrectly predicted that the digital computer would soon make automatic speech recognition possible. Since then, many advances have been made. But we are still a long way from duplicating the speech recognition capabilities of humans.

The following subsections detail some of the problems plaguing automatic speech recognition systems.

### a. Coarticulation

*Coarticulation* is one of the most difficult challenges facing speech systems. There are approximately 45 phonemes in American English (actual numbers vary slightly in the literature). Unfortunately, the acoustic characteristics of a given phoneme can vary widely depending on its phonetic context. For example, the second **D** and the **Y** in the sentence, "Did you find her?" can sound more like a **JH** in continuous speech [after Ref. 1]. Coarticulation occurs because the articulators must move rapidly between the target positions for each phoneme. In fact, they often never reach the target position for one phoneme before they start transitioning to the next. Thus, speech recognition systems must cope with a very large number of actual speech sounds. This is far more challenging than identifying 45 or so phonemes in isolation.

### b. Segmentation

Another problem in speech systems is in locating the boundaries between words and between phonemes. As mentioned earlier, there is frequently no pause whatsoever between words. And coarticulation makes for extremely fuzzy boundaries between phonemes. So a speech recognition system must recognize patterns of varying lengths in a rapidly changing acoustic stream. This problem partially accounts for the failure of some phoneme recognizors to perform nearly as well on words. It is one thing to identify a hand segmented slice of speech. It is quite another to spot the same phoneme in a waveform when the system lacks a priori knowledge of where to start and stop looking.

### c. Speaker Variations

Obviously, accents, dialects, and speech pathologies will present problems for speech recognition systems. But more fundamental differences between speakers also wreak havoc with recognition systems. As mentioned earlier, formant frequencies tend to be inversely proportional to vocal tract length. Thus, adult males, with vocal tracts an average of about 30 percent longer than women, exhibit formant frequencies that are lower than females for the same phoneme [Ref. 7]. These individual variations cause overlap between formant positions for vowels of different classes. Humans apparently adjust rapidly to the formant frequencies of a particular

speaker [Ref. 7] and some researchers have attempted to build recognition systems that do the same [Ref. 8].

In addition to vocal tract-related differences, recognition systems must cope with variations in speech rate, volume, inflection, and pitch. Even separate utterances by the same speaker will vary considerably from time to time. For example, a pilot handling an in-flight emergency might speak more rapidly and at a higher pitch than normal.

### d. Confusability and Ambiguity

The easiest way to introduce confusability and ambiguity is with a few examples [after Ref. 7]:

- Homonyms - "red" versus "read" or "two" versus "to"

- Word segmentation - "I scream for ice cream." versus "ice cream for ice cream" or "gray tape" versus "great ape" or "light housekeeper" versus "lighthouse keeper"

- Syntactic ambiguity - "The boy jumped over the stream with the fish." (Did the boy have a fish or did the stream?)

- Semantic ambiguity - "The council decided to subsidize the port." (Is this an alcoholic or salt water port?) or "He saw that gas can burn." (Is "can" a verb or a noun? [Ref. 1])

A certain amount of world knowledge is needed to make sense out of these examples in a given context. Thus, artificial intelligence techniques must be used to attack such problems. However, artificial intelligence techniques will be more likely to succeed if lower levels (acoustic and lexical analysis) of the recognition system perform well.

### e. Noise

Ultimately, speech recognition systems may be judged by their ability to handle noise. Humans do a superb job of understanding speech under exceptionally noisy conditions. Even interfering speech, the worst possible form of noise, is easily handled by humans. This is evidenced by the well-known *cocktail party effect*, our ability to isolate and listen to one

speaker's voice in a crowd. [Ref. 3] But current speech recognition systems are hard-pressed to perform well even on clean speech.

## 5. Existing Commercial Systems

Existing commercial systems typically use *hidden Markov models* (HMMs) at the heart of their recognition engines. The HMM dates back to the 1950s, when statisticians were looking for ways to characterize random processes for which they had only incomplete observations. They developed the HMM to model these problems as *doubly stochastic processes*. In an HMM, the observed data is considered to be the result of passing the "true" (hidden) process through a "censor," resulting in a second observed process. Both processes are characterized using only the one that can be observed. The HMM was introduced to the speech recognition field in the 1970's through the independent work of Baker at Carnegie-Mellon and Jelinek and his colleagues at IBM. [Ref. 1]

Hidden Markov Model speech recognition systems operate by using a training process to build a statistical model for each word (or other utterance) in the vocabulary. Each model is imagined to be a *finite state machine* capable of generating observation strings. During recognition, given an actual observation string, the system determines the likelihood that each word's HMM produced that particular observation string. The word with the highest likelihood is declared to be the word recognized. [Ref. 1]

Dragon Systems and IBM currently offer two of the most well-known commercial speech recognition systems. The following specifications were taken from Internet Home Pages describing their respective products (as of May 1996):

### a. Dragon Systems DragonDictate 2.0 for Windows (Power Edition)

- 60,000 word primary (in RAM) dictionary

- 120,000 word backup (on disk) dictionary

- Up to 64.000 word user dictionary

- Domain-specific dictionaries available

18

- Discrete-speech - QuickTalk feature allows user to shorten the pause between words. Dragon Systems calls this semi-connected speech. Also, some commands and numbers may be spoken in a truly continuous manner.

- Speaker Adaptable - Dragon Systems claims reasonably accurate recognition "out of the box." Training or "enrollment" is optional. The system automatically adapts to the user during use (over time).

- 16 MB RAM required, 20 MB recommended (8 MB dedicated to DragonDictate).

- 42 MB hard disk for system (minimum) plus 14 MB per user (for speaker-dependent data).

- DSP card not required but recommended for computationally-intensive applications (uses ordinary sound card otherwise).

### b. IBM VoiceType Dictation for Windows

- 22,000 word general-purpose dictionary

- Up to 2.000 word user dictionary

- Domain-specific dictionaries available (including 20,000 Radiology words, 16,000 Emergency Medicine words, 30,000 Journalism words, or 25,000 Legal words)

- Discrete-speech - IBM states that a pause between words of as little as 1/10 of a second is sufficient. They also claim that users can achieve dictation rates of at least 70 to 100 words per minute

- Speaker Adaptable - A brief "enrollment" period is required.

- 12 MB RAM required, 16 MB recommended (8 MB dedicated to Dragon Dictate).

- 33 MB hard disk for system (minimum) plus 14 MB per user.

- IBM claims that accuracy percentages in the high 90s are not uncommon. (No accuracy figures were offered in the Dragon Systems home page.)

- IBM DSP card required.

Not surprisingly, neither home page offered details on the recognition algorithms used. But the product descriptions used terms implying that HMMs are used for language and word models in both systems. To put some of the product specifications in perspective, consider the following facts:

- The most commonly used words make up about 75% of typical text samples. A vocabulary of 5000 words is sufficient for about 95% of typical text. The remaining 5% of words will depend on the user and the domain of discourse. [Ref. 7]

- The word search space increases exponentially with vocabulary size. So both recognition time and error frequency are likely to increase with vocabulary size.

- At least one study indicates that users find discrete word recognition systems quite acceptable for some office tasks [Ref. 7]. This is not surprising, in that the alternative to careful enunciation may be "hunt and peck" typing for many. However, the general public is much less cooperative with someone else's automatic recognition system than an office worker would be with his own system. This fact has plagued telephone industry researchers since the earliest days of speech recognition research.

- Both systems place relatively hefty demands on the personal computer.

Much more could be said about the state of the art, and speech recognition in general. But further details will be deferred until the discussion of related issues in subsequent chapters. For example, previous research on formant normalization will be included in Chapter III's discussion of that topic.

## C. BACK-PROPAGATION NEURAL NETWORKS

Having covered some of the fundamentals of speech recognition, it is time to describe the back-propagation neural network, the primary pattern recognition tool used in this research.

Artificial neural networks are an attempt to emulate the massively parallel processing approach used by the human brain. Unfortunately, we still know far too little about biological neural networks themselves. So existing artificial neural networks (ANNs) are still quite primitive. Nevertheless, they have proven to be extremely useful in a wide variety of fields. And neural network research is continuing on many fronts.

No attempt will be made to provide a comprehensive introduction to neural networks. The subject has inspired hundreds of books and thousands of research papers over the last 50 years. Instead, only back-propagation neural networks will be described. This architecture has been said to account for about 80% to 90% percent of practical ANN applications to date [Ref. 9].

### 1. MATLAB Neural Network Toolbox

The bulk of this research was conducted using the *MATLAB Neural Network Toolbox,* a product of The Mathworks, Incorporated. MATLAB itself offers a rich collection of signal processing, numeric computation, and visualization tools. The Neural Network Toolbox includes a variety of functions specific to neural network applications. MATLAB also includes a powerful script language for developing custom algorithms and extending the toolbox.

In addition to the Neural Network Toolbox, the MATLAB code in Appendix A makes use of several functions from the *Signal Processing and Communications Toolbox.* The latter toolbox was developed at the Naval Postgraduate School to support coursework in speech and signal processing [Ref. 10].

In the sections and chapters that follow, equations are often expressed in terms of MATLAB function calls. This should reduce ambiguity and simplify cross referencing of equations with code in Appendix A. Details on MATLAB script language syntax and functions are found in user's guides for the Neural Network Toolbox and MATLAB itself [Ref. 9]. The Neural Network Toolbox and its User's Guide also include clear and concise tutorials on neural networks, as well as demonstration scripts to illustrate concepts and facilitate experimentation.

## 2. The Biological Neuron

A typical biological neuron is pictured in Figure 2.8. A number of branch-like projections called *dendrites* radiate from the cell body (*soma*). Neurons receive signals from other neurons through connections, called *synapses*, at the dendrites or the cell body itself. Some synapses are *excitory* and tend to raise the activation level of the neuron. Other synapses are *inhibitory* and lower the activation level. The incoming signals arrive, typically as asynchronous pulse trains, and contribute to what can best be called a decaying weighted average. When the neuron's activation level reaches some threshold, it fires, sending a signal to other neurons via its *axon* and *axon branches*. A neuron can directly receive signals (via synapses) from up to 10,000 or so other neurons. Likewise, it can send signals to 10,000 or so other neurons (including connections to its own dendrites or soma). [Ref. 11]
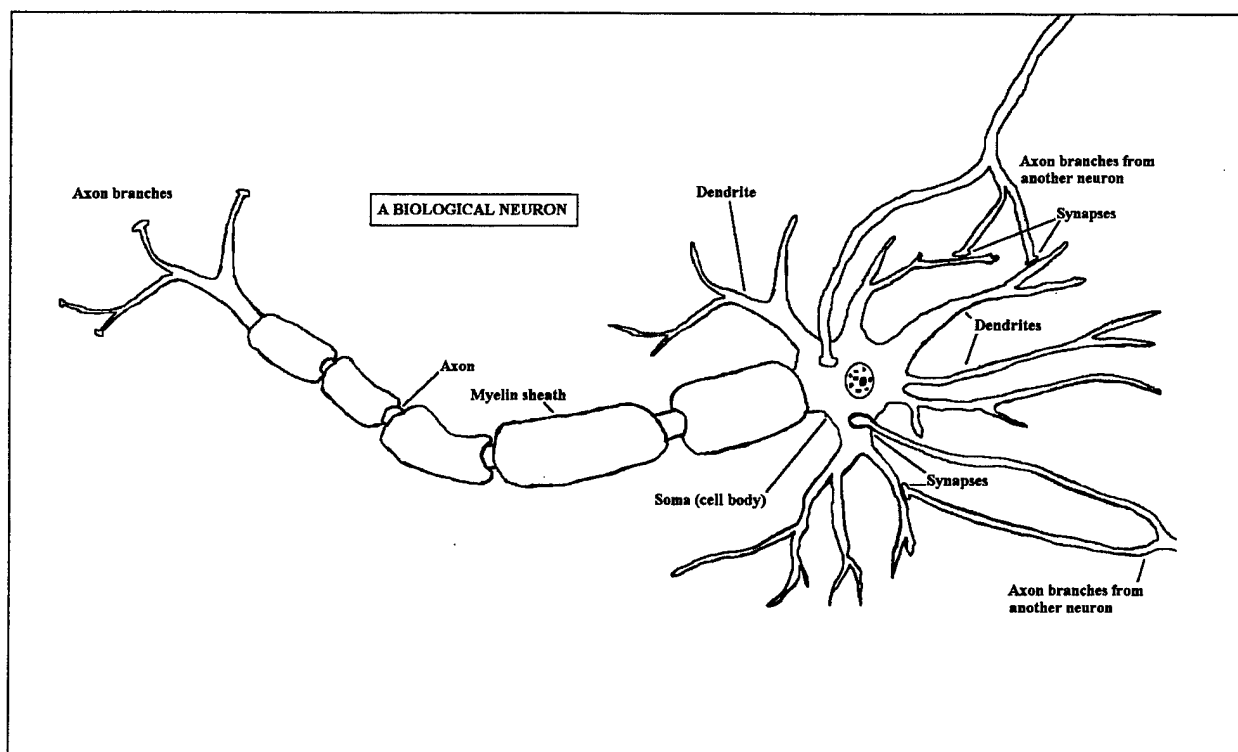


Figure 2.8. The Structure of a Typical Biological Neuron [after Ref. 21].

The human brain contains on the order of 10 billion total neurons (approximately the number of stars estimated to make up the Milky Way galaxy). These neurons form a network with an estimated 60 trillion interconnections between neurons. (By contrast, an artificial neural

22

network is considered large if it contains a few thousand neurons and a few million connections.) Many of these connections form during the first two years of life, at a rate of up to one million synapses per second. The mechanisms behind this process, and its exact relationship to learning, are largely unknown. [Ref. 11]

The neurons in the cerebral cortex (the portion apparently responsible for our higher mental processes) are arranged into a wrinkled sheet two to three millimeters thick and about 2,200 square centimeters in area (about the area of two computer keyboards). Based on reaction time studies and the speed at which biological neurons switch (a few milliseconds), it has been estimated that human perceptual decisions cannot involve more than 100 or so serial steps [Ref. 11].

### 3. The Artificial Neuron

Back-propagation neural networks are constructed of what are commonly called McCulloch and Pitts (MP) neurons, named in honor of two pioneering researchers at the University of Chicago in the 1940s. The MP neuron, pictured in Figure 2.9, is a greatly simplified abstraction of the biological neuron. The inputs are multiplied by their respective weights and then added. The weighted sum is then passed through a *transfer function* (also called *activation function*) to determine if the neuron will "fire." In practice, a *bias input* is usually added, for reasons to be explained later. The bias input is always one. But the weight associated with the bias is adjusted during training, as are the other weights.
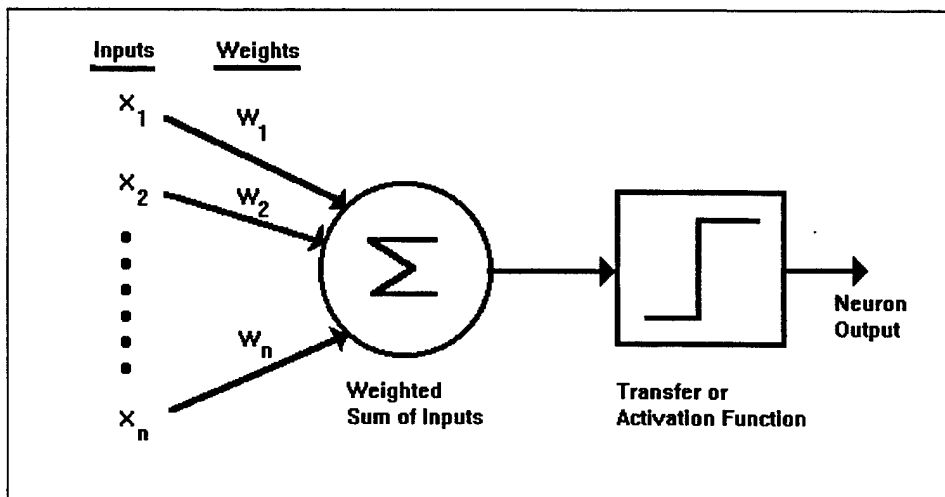


Figure 2.9. A McCulloch and Pitts Neuron [after Ref. 11].

Neural network training and interconnection strategies are introduced later. First, it is useful to see how a single MP neuron can be applied to simple problems.

## 4. Artificial Neural Networks as Classifiers

Figure 2.10 illustrates a simple classification problem. Each of the two inputs can be one or zero. There are four possible pairs of inputs, commonly known as *input vectors*. An input vector consisting of two ones should be classified into output category one (plus symbols in the figure). All other input vectors should result in an output of zero (circles in the figure). Clearly, this is a logical AND operation. So we can certainly solve it using conventional hardware or programming techniques. But it is useful for illustrating how neurons operate.
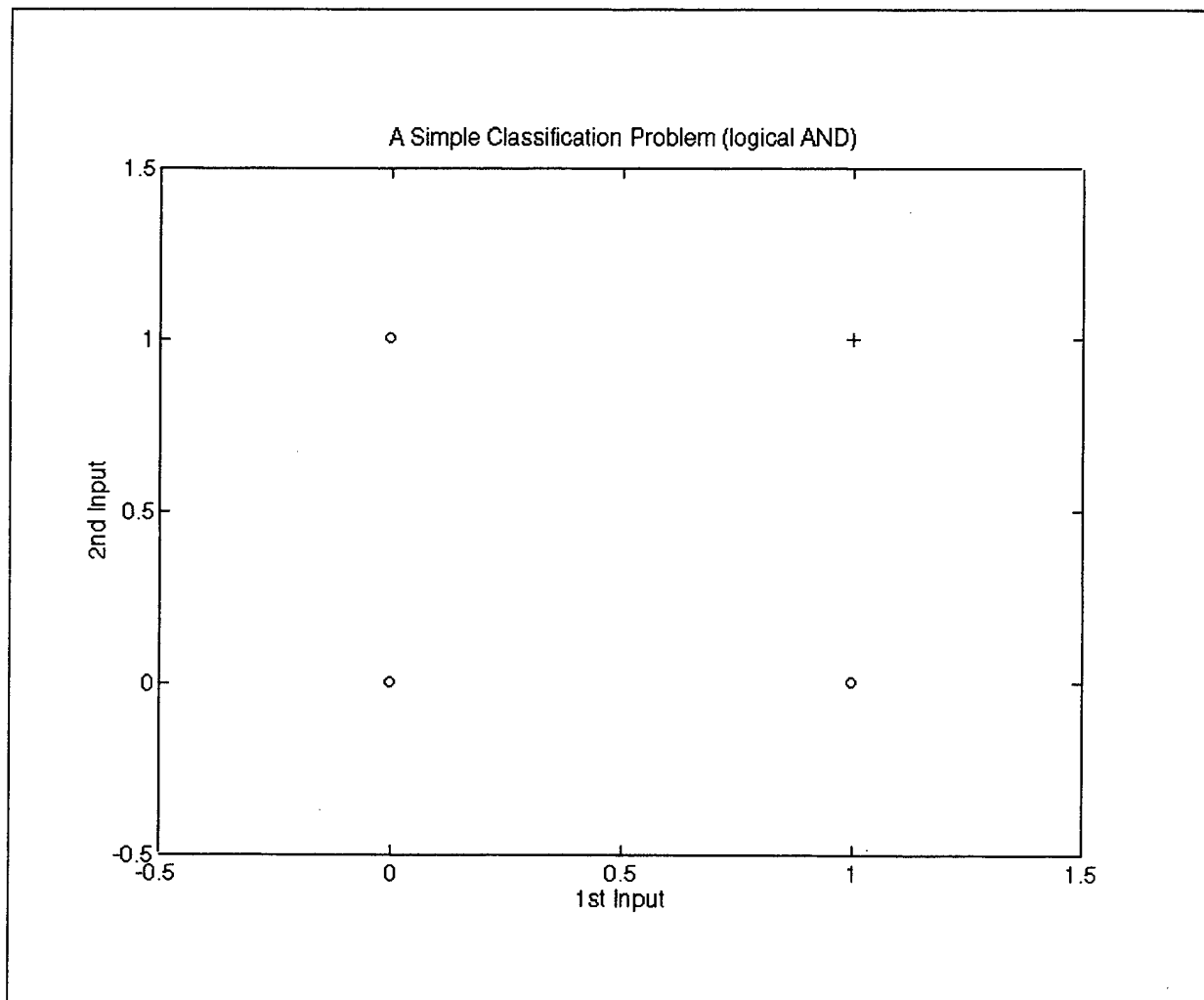


Figure 2.10. A simple classification problem (logical AND). The two inputs are plotted along the axes. Correct output classes are indicated with "+" (true) and "o" (false) symbols.

It is immediately obvious from Figure 2.10 that the input vectors can be properly separated into the correct output classes by drawing a diagonal line through the input space. Recall that the general form, for the equation of a line is:

$$Ax + By + C = 0 \qquad (2.1)$$

where A, B, and C are constants. This can easily be translated into neural network form by renaming some variables:

$$x_1W_1 + x_2W_2 + B = 0 \qquad (2.2)$$

where $x_1$ and $x_2$ are inputs, $W_1$ and $W_2$ are weights, and B is a bias. Thus, a MP neuron with two inputs can be used to represent a line in a two-dimensional input space. If we assume a threshold transfer function, an input vector on one side of the line will make the left side of Equation 2.2 negative and output will be zero. Input vectors falling on the other side of the line will make the left side of the equation positive and the output one. Inputs falling exactly on the line will result in a sum of zero. In that unlikely case, the output would normally also be zero. Without a bias, all classification lines would necessarily pass through the origin. Thus, a *bias* is usually included to allow classification lines anywhere in the input space.

At this point, it is natural to wonder how the weights in Equation 2.2 are determined. In practice, they are usually initialized to random numbers that have been scaled based on the range of input values. The weights are then adjusted until the resulting line leads to the correct classifications. This process will be described in more detail later. For now, it is sufficient to observe the results. Figure 2.11 shows the initial (random) classification line of a single neuron. Figure 2.12 shows the final line defined by the neuron's weights after training.

If a third input is added, the neuron's summation formula becomes:

$$x_1W_1 + x_2W_2 + x_3W_3 + B = 0 \qquad (2.3)$$

This, of course, is analogous to the general form of the equation for a plane. Thus, our input space becomes three-dimensional and the input classes are separated using planes. With

four or more inputs, the input space becomes n-dimensional *hyperspace* and the input classes are separated using *hyperplanes*.
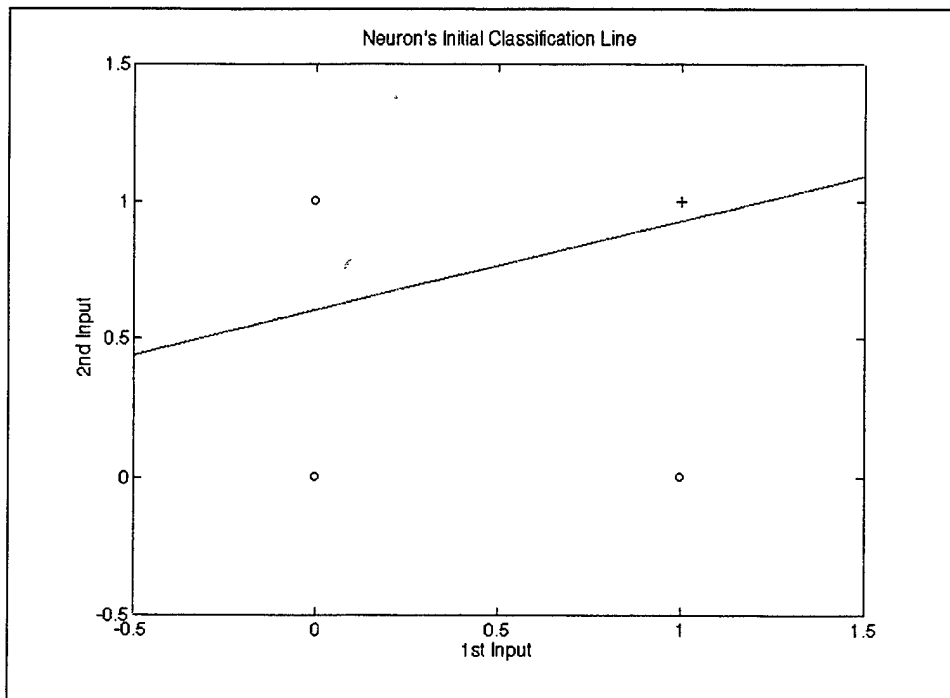


Figure 2.11. Classification line defined by the AND neuron's initial (random) weights.
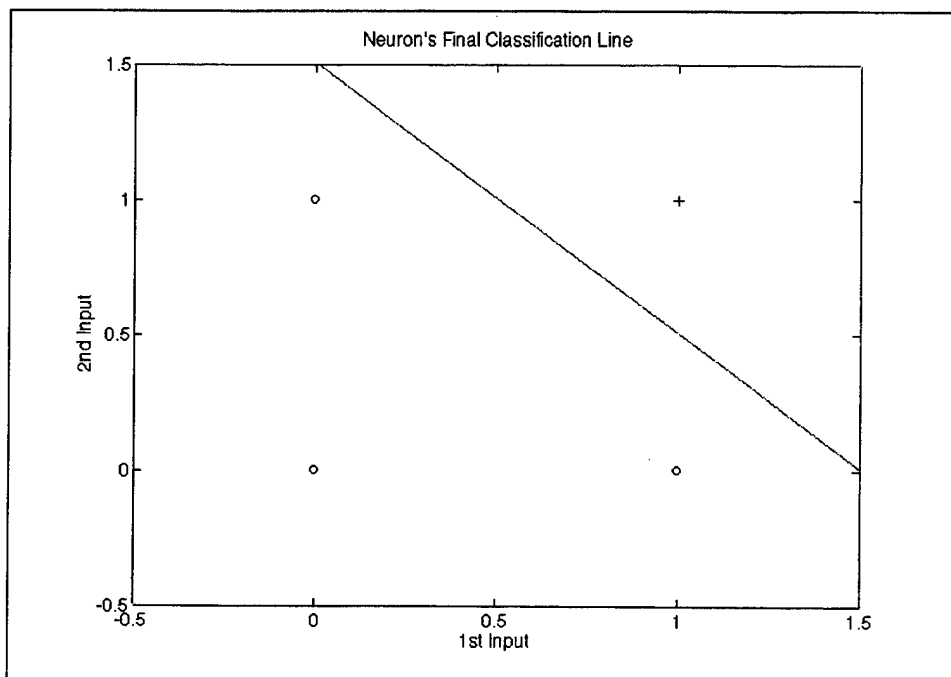


Fig 2.12. Classification line defined by the AND neuron's final (trained) weights.

26

## 5. Linear Separability and Multi-layer Networks

The McCulloch and Pitts neuron, after a brief period of excitement in the 1940s, received little attention until the 1960s. Then, Rosenblatt's *perceptron convergence theorem* led to a flurry of research. But research, and funding, largely ended in the early 1970s. A 1969 book by Minsky and Papert is usually credited (or blamed) for the dramatic loss of interest in the perceptron (usually defined to be a simple MP neuron with a threshold transfer function). The most important point made by Minsky and Papert is that a simple perceptron cannot learn to correctly classify input vectors unless the input vectors from different output classes are linearly separable. The most famous example of a non-linearly separable problem is the exclusive-OR function. Figure 2.13 depicts the possible input vectors and their correct classifications. Note that no single line can be used to properly divide the input vectors into the correct output classes. Some researchers suspected that multi-layer networks would be able to solve this problem. But the perceptron training method would not allow weight corrections to propagate through multiple layers. And Minsky expressed doubt that multi-layer perceptrons would be useful, even if they could be built. [Ref. 12]
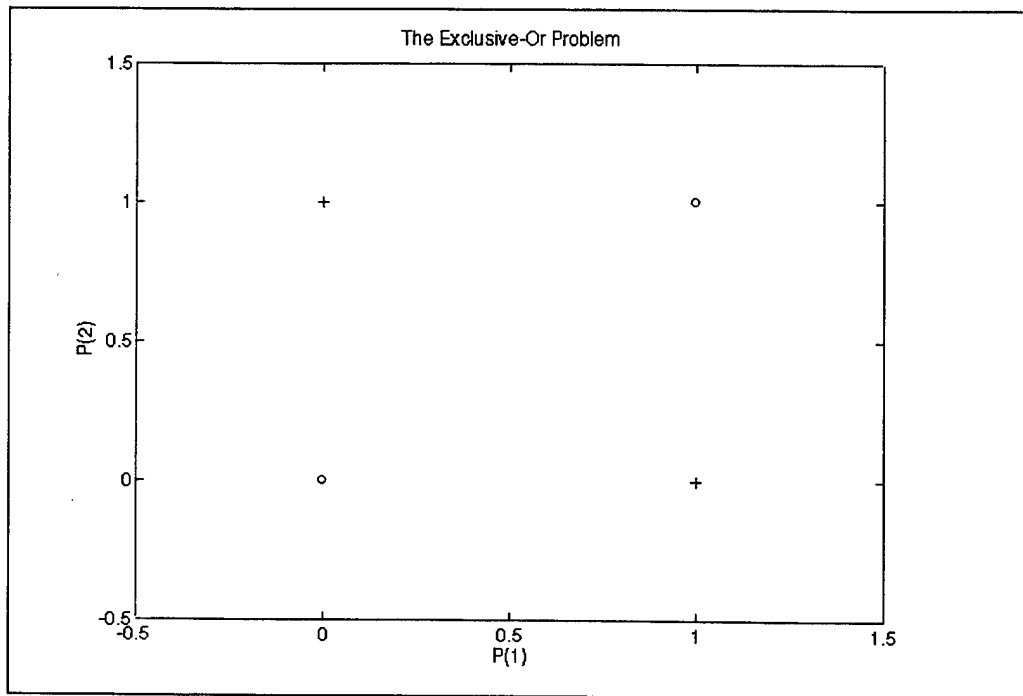


Figure 2.13. A non-linearly separable problem (the exclusive-OR problem).

27

In the mid 1980s, several researchers[*] more or less simultaneously proposed solutions to the problem of training multi-layer neural networks. Their back-propagation learning rule allows weight corrections to propagate from the output layer back through one or more *hidden layers*. This process will be described in more detail in the next section. First, it is necessary to understand how MP neurons are interconnected to form neural networks. Figure 2.14 shows a neural network capable of solving the exclusive-or problem. The network consists of two inputs, two neurons in a single hidden *layer*, and a single output neuron. Note that all neurons in the first hidden layer are connected to all of the inputs. Also, every output neuron (only one in this case) receives inputs from every neuron in the preceding hidden layer. This is referred to as a *fully connected feed-forward network*. Some network architectures are not fully connected and some involve connections between neurons of the same layer (or from a neuron to itself). But these architectures are less common (at least in artificial neural networks) and they will not be discussed further here.
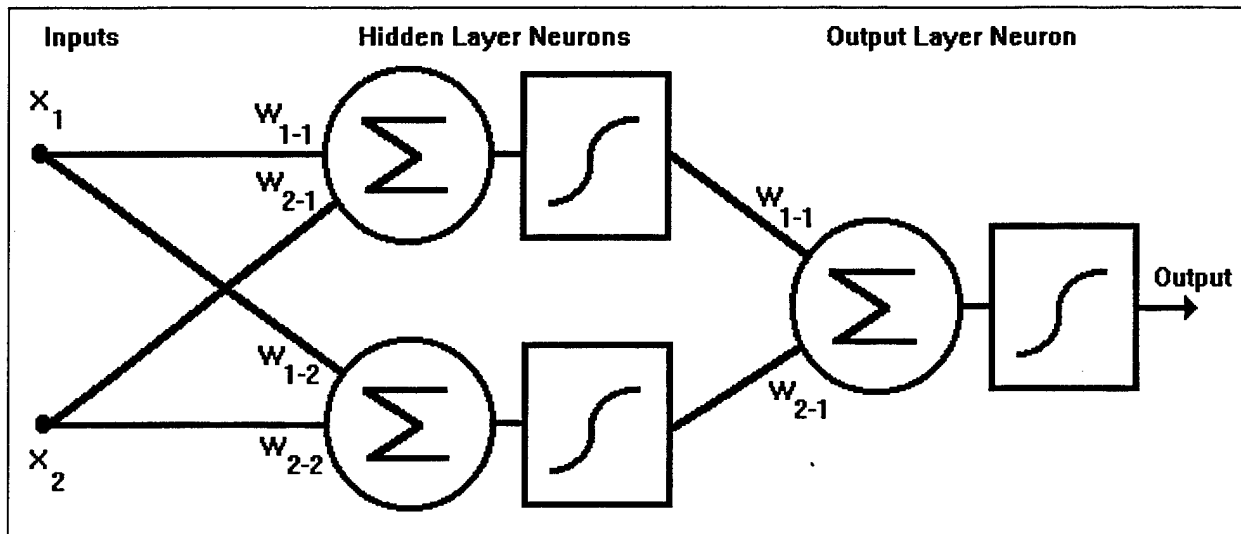


Figure 2.14. A multi-layer neural network capable of solving the exclusive-OR problem.

Figure 2.15 shows the two lines defined by the two hidden layer neurons of the network in Figure 2.14 (after learning to solve the exclusive-OR problem). Figure 2.16 shows the outputs of the hidden layer, mapped into the output neuron's input space. Note that this new input space, though still two-dimensional, is not the same as the original input space. This time, one axis is

---

[*] Rumelhart, Hinton, & Williams, 1986; Parker, 1985; LeCun, 1985; and Werbos, whose doctoral dissertation of 1974 solved the problem but was not noticed until much later.

the first hidden node's output and the other hidden node's output is plotted on the second axis. The output classes have again been indicated using plus symbols or circles. Note that the vectors are now linearly separable. The line in Figure 2.16 was plotted using the actual weights of a trained network (as have all previous classification lines). This simple example illustrates how multi-layer neural networks solve non-linearly separable problems. They use hidden neurons, often called feature detectors, to map the input vectors into a new input space, usually of different dimensionality. Given enough hidden neurons, any problem becomes linearly separable. This issue will be addressed in more detail in the next chapter.
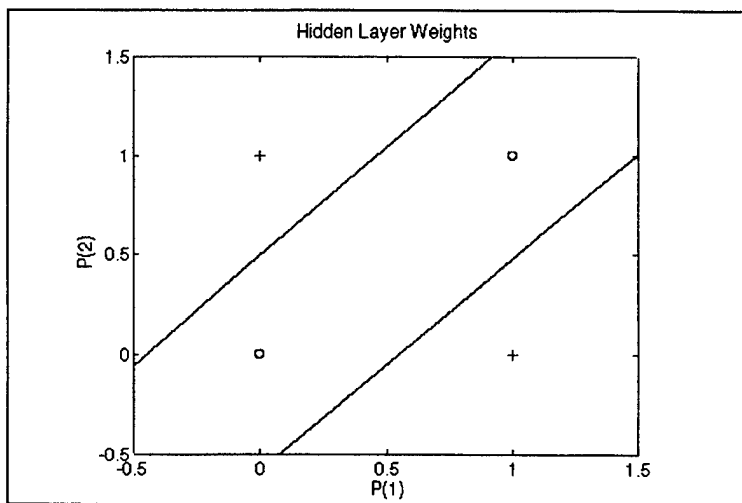


Figure 2.15. Classification lines defined by hidden layer weights for exclusive-OR problem.
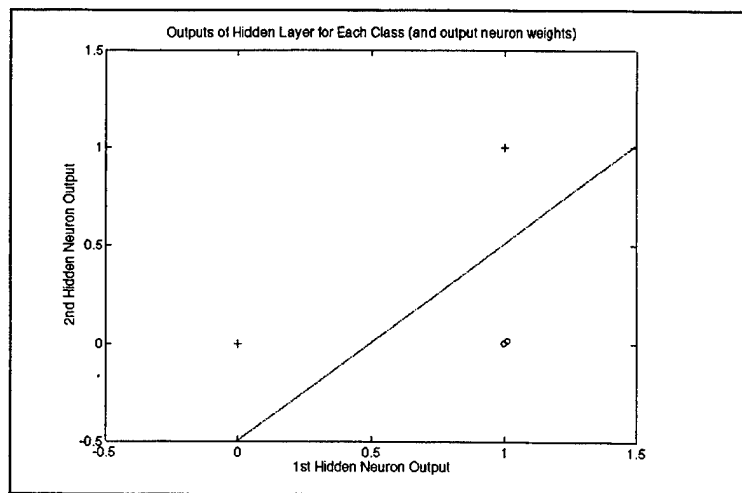


Figure 2.16. Classification line of output neuron plotted in the output space of hidden layer. Note that the hidden layer has mapped the exclusive-OR problem into a new two-dimensional space, where it is now linearly separable.

29

## 6. Training Methods

Multi-layer neural networks are typically simulated in conventional computers using matrices. In MATLAB's Neural Network Toolbox, a weight matrix for a given layer consists of an m by n matrix where m represents the number of neurons (layer outputs) and n is the number of inputs. Thus, the weight given to input i by neuron j is found in matrix position (j,i).

The outputs for a layer are computed by arranging the inputs into a column vector. The input vector is multiplied by the weight matrix to produce an output vector. The transfer function is then applied to each element of the output vector to produce the final output vector. In a multi-layer network, this output vector would become the input vector for the next layer. This is the reason for the term "feed-forward network." In matrix notation, the process is quite simple:

$$\text{output} = F(W*p + b) \tag{2.4}$$

where F(x) is the transfer function applied to each element of matrix x.

For example, the hidden layer outputs for the exclusive-OR network described earlier are found by:

$$\text{output vector} = F( \begin{bmatrix} -7.7068 & 7.2724 \\ -8.6552 & 7.8783 \end{bmatrix} \times \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} + \begin{bmatrix} 4.2454 \\ -3.8833 \end{bmatrix} ) \tag{2.5}$$

## 7. The Back-Propagation Learning Rule

Neurons with a threshold transfer function are unsuitable for multi-layer networks because it is impossible to determine which hidden layer neuron weights were the most responsible for incorrect (or correct) outputs. Thus, it is impossible to apply weight changes to only the incorrect weights because those weights cannot be identified. This is known as the *credit assignment problem* [Ref. 13]. Back-propagation solves this problem by using a differentiable transfer function, such as the *log-sigmoid* function:

$$\text{logsig(n)} = 1/(1+e^{-n}) \tag{2.6}$$

With a differentiable transfer function, the network can properly assign credit for wrong answers, and reduce more heavily the input weights that contributed most to that incorrect output. Note in the following discussion that the back-propagation learning rule is really a form of gradient descent. The elements of the *delta vectors* (Equations 7 and 9) serve as error surface partial derivatives. Training proceeds as follows [Ref. 14]:

- An input vector from the training set is presented to the first hidden layer's inputs. The results feed forward through each layer as described earlier.

- The network's actual outputs are compared to the desired outputs. The output layer's weight corrections are based on the difference, an *error vector*. For output neuron *j*, a *delta* value must first be computed using the error vector (in parentheses), the weighted sum of the neuron's inputs, and the transfer function's derivative:

$$\delta_j = (t \arg et_j - output_j) f'(\Sigma W_i I_i) \tag{2.7}$$

where f' is the derivative of the transfer function.

- The output neuron's delta value is then used to determine the change to apply to each input weight $W_{ji}$ as follows:

$$\Delta W_{ji} = \eta \delta_j I_i \tag{2.8}$$

where $\eta$ is the *learning rate*, typically between .25 and .75, and $I_i$ is the input associated with the weight being adjusted. In practice, the learning rate is often adjusted dynamically as training proceeds. This research was conducted using the adaptive learning rate algorithms included in the MATLAB Neural Network Toolbox.

- The hidden layer weight adjustments are slightly more complicated. To compute the delta value for a hidden neuron, the delta values of the output neurons are multiplied by the weights of their connections to that hidden neuron. The hidden neuron's delta is thus the weighted sum of the output layer deltas. For hidden neuron *k*, the delta value is:

$$\delta_k = (\Sigma \delta_i W_{ki}) f'(\Sigma W_i I_i) \tag{2.9}$$

31

where $W_{kj}$ is the weight between this hidden neuron and output neuron j and $f'(\Sigma W_i I_i)$ is again the transfer function's derivative for the sum of the hidden neuron's weighted inputs.

- Once the hidden neuron's delta value has been computed, its weights are adjusted as before (using Equation 2.8).

- If the network has more than one hidden layer, the error corrections continue to back-propagate in this manner until all layers have been adjusted.

- Once the weights of all layers have been adjusted, the next training input vector is presented to the network and the entire is process repeated.

- A momentum term $\alpha$ is often added to Equation 2.8 to helps the network skip over small local minima [Ref. 13]:

$$\Delta W_{ij}(n) = \alpha \Delta W_{ji}(n-1) + (1-\alpha)\eta \delta_j(n) I_i(n) \qquad (2.10)$$

In this form of the learning rule, each weight is adjusted during epoch n by an amount equal to the weighted average of the previous weight change (epoch n - 1) and the weight change dictated by Equation 2.8. Thus, the network follows a running average of the gradient rather than just the local gradient. If the momentum constant is zero, this degenerates to pure gradient descent (Equation 2.8). If the constant is one, the gradient is ignored and the last weight adjustment is repeated (not very useful). A constant of between 0.90 and 0.95 (usually the latter) was used in this research.

The training paradigm just described is known as *supervised learning*. The training algorithm must know each training vector's correct output class in order to perform the weight adjustments. *Unsupervised learning* algorithms also exist. But they are less common (at least in artificial neural networks) and they will not be discussed here.

## 8. Neural Networks Versus Conventional Programming Techniques

The previous examples are problems that could easily be solved with the kind of conditional statements found in every programming language. But consider a program to process

32

loan applications. One might initially think that it would be possible to simply apply thresholds to each of the inputs and approve the applications where all or a certain number of inputs exceed the threshold. For example, a recent bankruptcy could be used to justify an immediate rejection. But what if the applicant now has a six-figure income? Humans deal easily with situational thresholds and fuzzy logic. But the thought processes used by humans don't easily reduce to nested if/then statements. For this reason, the average loan officer would probably find it difficult to write an unambiguous set of instructions describing how to evaluate loan applications.

Artificial neural networks deal easily with problems like this. In fact, some financial institutions currently screen loan applications using neural networks. The networks are trained by repeatedly presenting training data (such as actual loan applications) and comparing the network's output with the "correct" output (such as an indication of whether the applicant defaulted on the loan). If the network's output is incorrect, the weights that contributed to the incorrect output are reduced. Training then resumes until the error rate becomes acceptable.

## 9. Artificial Versus Biological Neural Networks

It is somewhat traditional, and wise, to include a disclaimer in descriptions of artificial neural networks. Specifically, we must remember that ANNs bear little real resemblance to the biological neural networks that inspired them. This is particularly true of back-propagation neural networks (BPNNs). Some of the key differences include:

- BPNNs operate in a synchronous feed-forward manner. All inputs are sampled at the same time. The first layer's weights are then all calculated and synchronously applied to the next layer's inputs. Biological networks, on the other hand, operate asynchronously. A biological neural network's neurons integrate their inputs over time, firing asynchronously. The resulting activity patterns bear little resemblance to the orderly lock-step operation of ANNs.

- The neurons in a BPNN layer are typically fully connected with preceding layers but do not receive inputs from the same or succeeding layers. Again, some ANNs do employ feedback and competition within layers. But the majority of practical ANN applications have used the feed-forward architecture. Biological neural networks, on

33

the other hand, often employ connections that imply feedback and competition within "layers." Biological networks also imply much more varied, and far less symmetric, connection strategies. In particular, clusters of richly interconnected biological neurons often form subnetworks, a very different strategy from the monolithic fully-interconnected ANN.

- The supervised learning paradigm described earlier is biologically implausible. Self-organizing ANNs have been built. But far too little is known about the mechanics of human learning to judge how close those ANNs come to the mechanisms of their biological counterparts.

This disclaimer is certainly not intended to slight the importance of ANNs. They have proven quite useful in spite of their simplicity. And it may never be necessary to exactly mimic the operation of the brain. After all, manmade machines seldom exactly duplicate the operation of the biological structures that inspire them. Otherwise, our airplanes would have feathers and our automobiles would travel on hooves.

## 10. Neural Networks and Speech Recognition

The commercial speech recognition systems described earlier are quite impressive compared to the systems of just a few years ago. But no existing system can claim to have human-like continuous speech recognition abilities. And the HMM approach has disadvantages, some theoretical and some practical, that may place limits on its long-term usefulness. One such theoretical disadvantage is the necessary assumption that the features extracted within a phonetic segment are uncorrelated with each other. This is obviously not true [Ref. 15]. A more practical disadvantage is that large-vocabulary HMMs are very computationally intensive, as evidenced by the hardware requirements described earlier. It simply may not be possible to implement real-time speaker-independent large-vocabulary continuous speech recognition on conventional (serial) computers using HMMs alone.

With the HMM perhaps reaching its "Peter principle" limits, many researchers have turned to the artificial neural network. Once trained, ANNs have the potential to greatly speed up recognition tasks because they can be implemented in highly parallel hardware. They have also

34

proven to be well suited to complex pattern recognition tasks, such as computer vision and speech recognition.

In fact, speech recognition is prominent in nearly every author's list of suitable neural network applications.

In the last ten years, hundreds of papers have been written describing neural network-based speech research. No attempt is made here to provide a comprehensive summary of previous research. But the following papers are illustrative of the scope of recent research:

- Hirai and Waibel developed a speaker-dependent time-delay neural network (TDNN*) for Japanese phoneme spotting, combined with a higher level network for word recognition. [Ref. 16]

- Wendell and Abdelhamied used modular TDNNs to develop a phoneme recognizor. In a modular neural network the recognition problem is partitioned into subproblems and subnetworks are trained to solve particular subproblems. For example, one subnetwork might be trained to identify stop consonants and another vowels. [Ref. 17]

- Matsuura, Miyazawa, and Skinner built a small vocabulary phoneme-based neural network word recognizor. [Ref. 18]

- Iso and Watanabe investigated the use of demi-syllables (described later) in a large-vocabulary speaker-dependent Japanese language recognition system. [Ref. 19]

- Witbrock and Haffner constructed a recognition system which adapts to individual speakers to improve recognition accuracy. A Speaker Voice Code, representing a speaker's individual speech characteristics, was presented to a TDNN along with more conventional acoustic inputs. The system's accuracy on French digits was in excess of 99 percent. [Ref. 8]

- Morgan and Bourlard, and others, have researched the use of hybrid ANN/HMM architectures. [Ref. 15]

---

\*      A TDNN neuron receives both current inputs and delayed prior inputs.

- Dalsgaard, Anderson, and Jorgensen applied self-organizing neural networks to feature identification in continuous speech. [Ref. 20]

- Sivakumar and Robertson showed that neural network consonant recognition could be improved by using formant transitions in neighboring vowels. [Ref. 5]

The state of neural network-based speech recognition can perhaps best be summarized by saying that the field is young but promising.

## D. SUMMARY

This chapter has summarized some of the fundamentals of speech recognition and artificial neural networks. These facts will be referred to, and expanded on, in Chapter III. In particular, the rationale behind the research methodology is explained in terms of these fundamentals.

# III. METHODOLOGY AND RATIONALE

## A. INTRODUCTION

Chapter III describes the specifics of how this research was conducted. This chapter also offers some of the reasoning behind those specifics. In particular, the following sections describe the rationale behind the choice of recognition unit, neural network feature vector, frequency normalization method, time alignment method, neural network topology, and word recognition algorithm.

## B. CHOICE OF RECOGNITION UNIT

Probably the most fundamental decision in speech recognition system design is the choice of recognition unit. Systems can be, and have been, designed to detect words, syllables, demi-syllables, and phonemes. Another recognition unit, the diphone, has been used effectively in speech synthesis. But it has received far less attention in speech recognition. The diphone is discussed in depth later in this section. First, the alternative recognition units are described.

### 1. Words

Small-vocabulary word recognizors have been built. However, it is probably impractical to build a large-vocabulary, speaker independent system using the word as a recognition unit. The most obvious problem is in the sheer size of the task. The system would be required to recognize on the order of 100,000 words. In order to achieve speaker independence, the system would probably have to train on hundreds of examples of each word, from a wide variety of speakers. Just assembling the training and test data would be a daunting task.

Another problem with words is in their length. Longer speech segments magnify the impact of speaker variabilities such as speech rate. Also, an enormous amount of storage would be required to hold acoustic templates for 100,000 words. The situation is analogous to assigning a unique symbol to every word in the English language (rather than spelling each word with combinations of 26 letters). If humans were to attempt this, we would no doubt have to devote a great deal of our mental storage capacity to storing images of those symbols.

A final problem is in segmentation. As mentioned earlier, words acoustically merge in continuous speech. Word recognition systems normally get around this problem by requiring discrete utterances. Continuous-speech word recognizors would have to spot words of widely varying length in a continuous speech stream.

### 2. Syllable

A syllable consists of a vowel (in the center) and, usually, one or more starting and/or ending consonants. Syllables are attractive because they are relatively easy to segment. They are identifiable in the speech envelope by the large amount of energy (and voicing) associated with the vowel. They also account for coarticulation between phonemes in various phonetic contexts (see the phoneme paragraph for more on this issue). Unfortunately, ambiguity can arise in dealing with strings of consonants. Specifically, the recognizor must decide whether to group the consonants with the preceding vowel or the succeeding one. Another problem is, again, in the size of the search space. Though fewer in number than words, there are still an estimated 10,000 syllables in the English language. [Ref. 7]

### 3. Triphone

The triphone is a combination of three phonemes. This is attractive because the triphone would also account for coarticulation. In fact, every phoneme could be modeled in every possible phonetic context. But the numbers are, again, quite large. If we use the 45 phonemes of Table 2-2, plus a dummy phoneme to represent silence, there are theoretically $46^3$ or 97336 possible triphones. Undoubtedly, far fewer combinations actually appear in English. But the number is still rather large, probably of about the same magnitude as the number of syllables.

### 4. Demisyllable

A demisyllable consists of half of a syllable. If each syllable in the English language is divided in half at the vowel, the 10,000 English syllables can be reconstructed from the resulting 2000 or so unique demisyllables. Thus, the number of required recognition units can be reduced significantly by employing demisyllables instead of syllables. As mentioned earlier, some research has been conducted using the demisyllable as the recognition unit [Ref. 19]. Though

38

very promising, the demisyllable requires slightly more complicated heuristics than some of the alternatives. In particular, the segmentation algorithm must deal with consonant clusters, as described earlier for syllables.

## 5. Phoneme

The phoneme is an attractive alternative if for no other reason than the small numbers, less than 50 in English. Also, the phoneme is tempting because of the large body of knowledge and theory behind it. Phonemes are appealing as well, from a practical viewpoint, because a large number of tools, including phonetically transcribed speech databases and phonetic dictionaries, have been built around the phoneme. Thus, the phoneme is the most commonly used sub-word recognition unit. Unfortunately, the phoneme is much more a theoretical unit of speech than a practical one. As mentioned in the section on coarticulation, a given phoneme's acoustic properties depend greatly on its phonetic context. Thus, a recognition system must force some 100 to 200 allophones [Ref. 7] into 50 or so phoneme classes. Another challenge is in segmentation. In general, there is no sharp dividing line between phonemes in flowing speech. This causes problems both in training and during recognition (this issue is discussed in more detail in the next section).

## C.   THE DIPHONE

The final sub-word speech unit to be discussed is the diphone, the recognition unit selected for use in this thesis research. A diphone consists of two phonemes. Theoretically, there are $46^2$ or 2116 possible combinations of the diphones in Table 2-1 (a dummy phoneme is again used to represent silence). But not all of these actually occur in English. The 6229 entry TIMIT lexicon used in this research contains only 1195 diphones. And the 100,000 word Carnegie Mellon phonetic dictionary contains only 1331 distinct diphones[*]. Furthermore, these diphones are by no means evenly distributed. As indicated in Figure 3.1, some diphones are far more common than others. In fact, the ninety most common diphones, just 7.5 percent of the total, account for over 50 percent of the diphone occurrences in the TIMIT lexicon. Similarly, the 490 most common

---

[*]    These numbers include all diphones formed by pairing phonemes with a period of silence, as at the beginning or end of an isolated word. The numbers were obtained using DIPCOUNT.ADA, a utility contained in Appendix B.

diphones, 41 percent of the total diphones, account for over 90 percent of the diphone occurrences. That means that fairly large vocabularies can be built from far fewer than 1220 or 1300 diphones.

### Cumulative Distribution of (1195) Diphones in Timit Lexicon
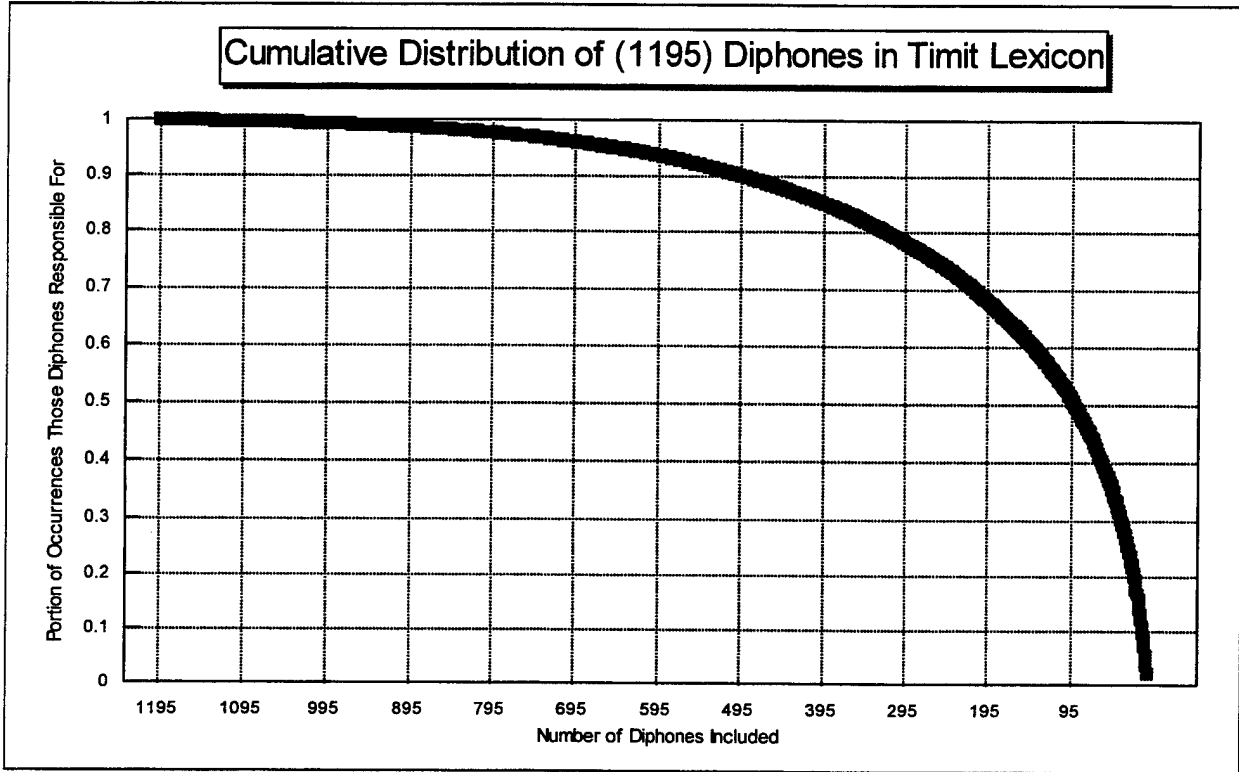


Figure 3.1. Cumulative distribution of diphones in the TIMIT lexicon. Although 1195 diphones exist in the lexicon, some are far more common than others. Thus, fairly large vocabularies can be constructed with far fewer diphones.

The remaining paragraphs of this section describe additional advantages of diphone-based recognition.

## 1. Coarticulation

The junction between two phonemes constituting a diphone is acoustically influenced by both phonemes. To a phoneme recognizor these coarticulation effects are a curse. But to a diphone recognizor this transition region is the key to recognition. As mentioned earlier, vowels are relatively easy to recognize due to their characteristic formant frequencies. But consonants are less recognizable in isolation. This issue was specifically investigated by Sivakumar and Robertson [Ref. 5], who showed that formant transitions in neighboring vowels could significantly increase accuracy in consonant recognition (especially stop consonants). Such an

40

advantage is particularly important in view of the fact that consonants actually carry most of the information in speech*. As further evidence of their potential, diphones have already proven useful in speech synthesis [Ref. 7]. If diphones lead to more natural sounding synthesized speech, it is reasonable to assume that diphone models would more closely match actual speech during recognition.

## 2. Redundancy and Probability of Detection

Each diphone carries twice as much information as a phoneme. To understand why this is useful, consider a hypothetical diphone recognizor and a hypothetical phoneme recognizor, both with 85 percent recognition accuracy. For the sake of comparison, assume that both recognizors are uniformly accurate for all recognition units and that the error probabilities between adjacent speech units are independent**. For the word "robot," consisting of five phonemes (**R OW B AA T**), the probability of a phoneme recognizor spotting the word (assuming no missing phonemes can be tolerated) would be:

$$p(\text{word spotted} \mid \text{phoneme recognizor}) = .85^5 = .44 \qquad (3.1)$$

The same five phoneme word would be described with six diphones (**HX-R  R-OW  OW-B  B-AA  AA-T  T-HX\*\*\***). But each diphone carries two phonemes. So two adjacent diphones would have to be missed to cause the word to be missed. In this case, the probability of word detection is given by:

$$p(\text{word} \mid \text{diphones}) = .85^6 + 6(.85^5 .15^1) + 10(.85^4 .15^2) + 4(.85^3 .15^3) = .90 \qquad (3.2)$$

The formula in Equation 3.2 was reached by counting the 21 success modes out of the $2^6$ or 64 possible outcomes. Only one outcome consisted of six correct diphones (contributing $.85^5$

---

\* This can easily be verified by removing the vowels from a typical sentence. The result will be much more understandable than the same sentence with all consonants removed.

\*\* Both assumptions are, of course, incorrect. Some diphones and phonemes will enjoy lower error rates and "burst errors" are very likely to occur in continuous speech.

\*\*\* The symbol "HX" is used throughout this thesis as a dummy phoneme consisting of some period of silence.

to the overall probability). Six outcomes included five correct and one incorrect diphone (adding $6[.85^5 .15^1]$ to the overall probability). Similarly, there were 10 outcomes with 4 diphones correct and 2 incorrect; and four outcomes with three diphones correct and three incorrect.

Thus, in this example and given identical sub-word recognition accuracies, the diphone recognizor is theoretically capable of more than twice the word recognition accuracy of a phoneme recognizor.

### 3. Search Space Reduction

The diphone offers one final advantage as a recognition unit. This last advantage becomes evident when considering the lexical search that might result from detecting a given speech unit in the speech stream. Again, consider a hypothetical phoneme recognizor and a hypothetical diphone recognizor. If the phoneme recognizor detects what appears to be an **R** in the speech stream, the lexical analyzer must consider all 341 words in the TIMIT lexicon that begin with that phoneme. And enough spurious phonemes will probably be detected after that to keep most of those words in the search tree.

Contrast this situation with a diphone recognizor that has detected the **R-OW** diphone in the input stream. Only 20 words in the TIMIT lexicon begin with that diphone. And not just any **B** in the input stream will be accepted in the lexical analyzer's consideration of the word "robot." The **B** must come in the form of an **OW-B** or **B-AA** diphone. Thus, diphones should reduce the lexical analyzer's burden because the search tree can be pruned much more quickly.

### D.  FEATURE VECTOR

A critical decision facing the neural network designer is the choice of input, or feature, vector. One cannot expect a neural network, or any other tool, to perform pattern recognition well unless relevant inputs are provided. For example, a neural network provided with the height and weight of applicants could not be expected to accurately predict loan defaults.

In addition to carefully choosing the ANN's inputs, it is often desirable to pre-process inputs to make the neural network's recognition task as straightforward as possible. In speech

recognition, the most obvious example of pre-processing is transformation from the time domain to the frequency domain.

The following subsections describe the inputs selected for this research, the preprocessing steps used, and the rationale behind their selection.

### 1. Differenced Cepstral Coefficients

Formant peaks can be difficult to identify in conventional spectra, as is evident from Figure 3.2. *Cepstral smoothing* is a method frequently used to make the formant peaks more easily identifiable. Cepstral smoothing is explained in Figure 3.3 [after Ref. 1].
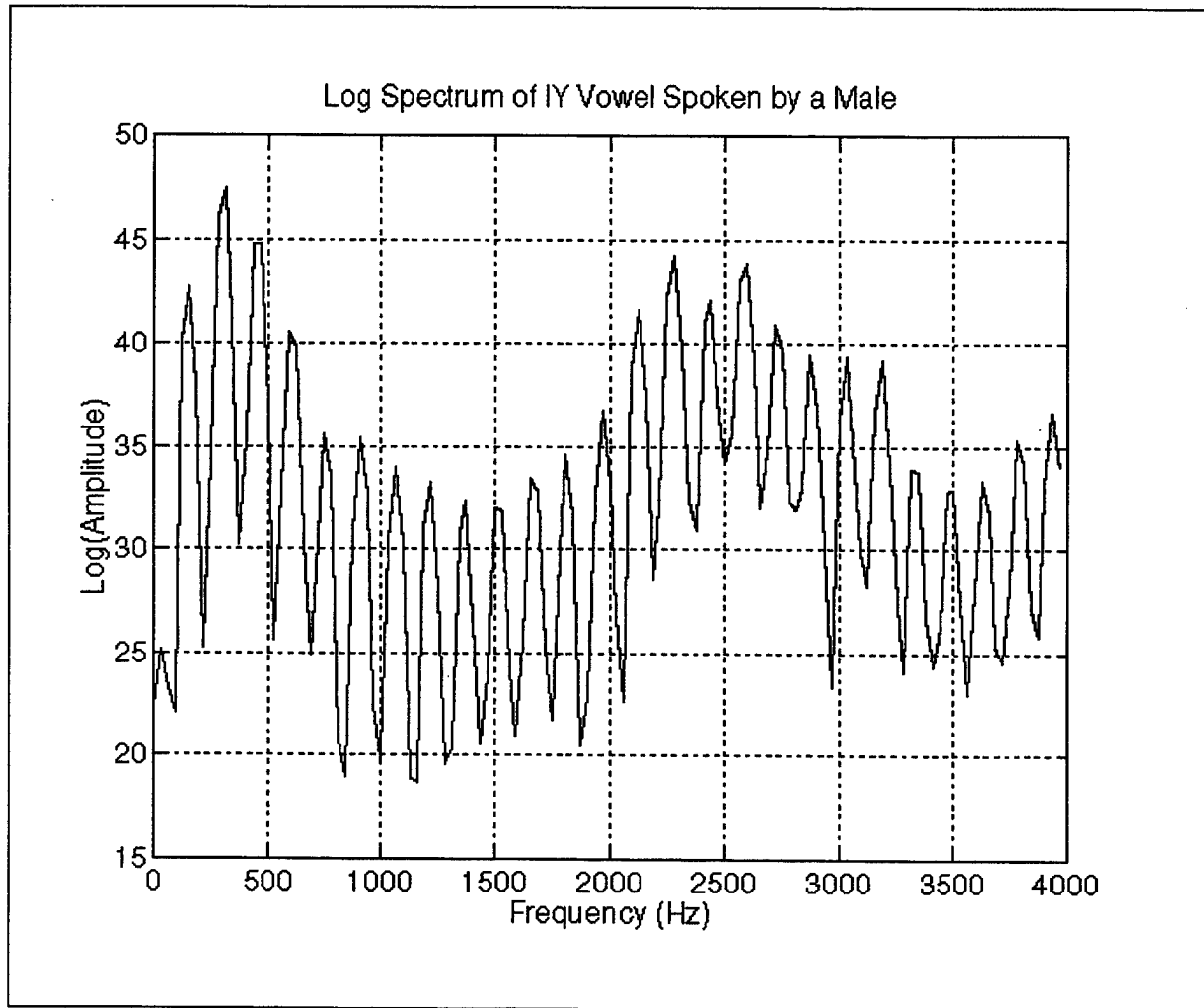


Figure 3.2 Log spectrum of the IY vowel spoken by an adult male. Note that F3 (about 3200 Hz) has partially blended with F2 (about 2600 Hz). This, combined with pitch combing, makes it difficult to exactly determine the positions of the formant peaks.

FFT(voice)　　　　　=　　　　FFT(pitch)　　　X　　　FFT(articulators)

log(FFT(voice))　　　=　　　log(FFT(pitch))　　　+　　　log(FFT(articulators))

IFFT(log(FFT(voice))　　=　　　Pitch Effects　　　+　　　Articulator Effects
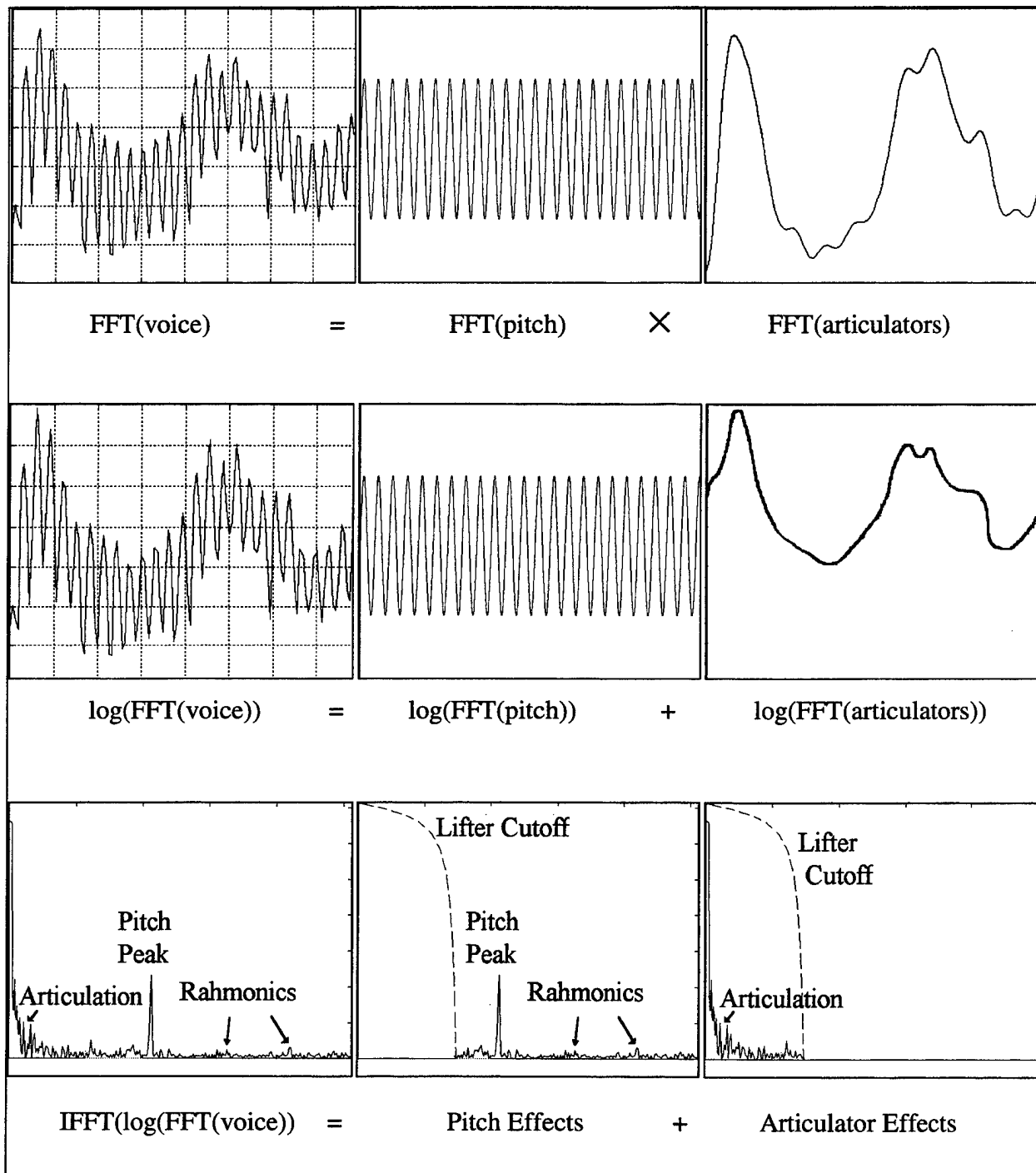
Figure 3-3. An intuitive explanation of cepstral smoothing of a voice signal [after Ref. 1]. The top row shows that a voice signal in the conventional frequency domain is a nonlinear combination (product) of vocal fold excitation effects (pitch "combing") and articulator effects (the "spectral envelope"). The two components are difficults to separate. But the logarithm of this product is the sum of the logarithms of the components (middle row). Thus, an inverse Fourier transform can be used to separate the pitch and articulator effects in the "quefrency domain." The bottom row shows a cepstral plot of a voice signal. In the cepstrum, the articulator effects are concentrated at the low quefrency end and pitch effects dominate at high quefrencies. A low-time lifter can be used to remove the pitch peak and its "rahmonics." A final Fourier transform then produces the smoothed spectrum at the right end of the middle row.

44

In MATLAB, the *cepstrum*[*] can be computed with the following function calls:

$$\text{cepstrum} = \text{ifft(log(abs(fft(signal .* hamming(512)))));} \qquad (3.3)$$

where "signal" is a PCM speech vector[**], "ifft" is the MATLAB inverse fourier transform function, and "fft" is the fast fourier transform.

Figure 3.4 is a cepstral version of Figure 3.2. The low *quefrency* components are due to articulator effects. The speaker's pitch is evident in the series of high quefrency spikes. A *low time lifter* is used to remove the pitch effects (the dotted line in Figure 3.4 shows the coefficients of the lifter used) and a final FFT produces the smoothed spectrum of Figure 3.5. In MATLAB's script language, these final steps could be accomplished by:

$$\text{smoothed\_spectrum} = \text{fft( cepstrum .* lifter );} \qquad (3.4)$$

where "lifter" is a vector of low time lifter coefficients.

A 512 point FFT is used to compute the cepstral coefficients used in the actual feature surface. The samples are Hamming windowed and each window position overlaps the previous window position by approximately one third (170 samples or about 10.6 milliseconds at the 16 KHz sample rate). Figure 3.6 illustrates the windowing strategy. Note that the term "sample window" is used throughout this thesis to refer to the 512 point FFT sample window. The term "recognition window" is used to refer to the longer window from which the overall feature vector is produced. A number of overlapped sample windows are required to constitute a single recognition window.

---

[*]    The cepstrum domain is not the frequency domain. So unique terms have been invented to deal with it. The most commonly used terms include *quefrency* (modification of the word frequency), *lifter* (from filter), *rahmonics* (from harmonics), and *cepstrum* itself (from spectrum). [Ref. 1]

[**]    In MATLAB's script language, the ".*" operator means to multiply each element in one vector by the corresponding element in the other vector. The hamming(n) function returns a n-point hamming window vector.
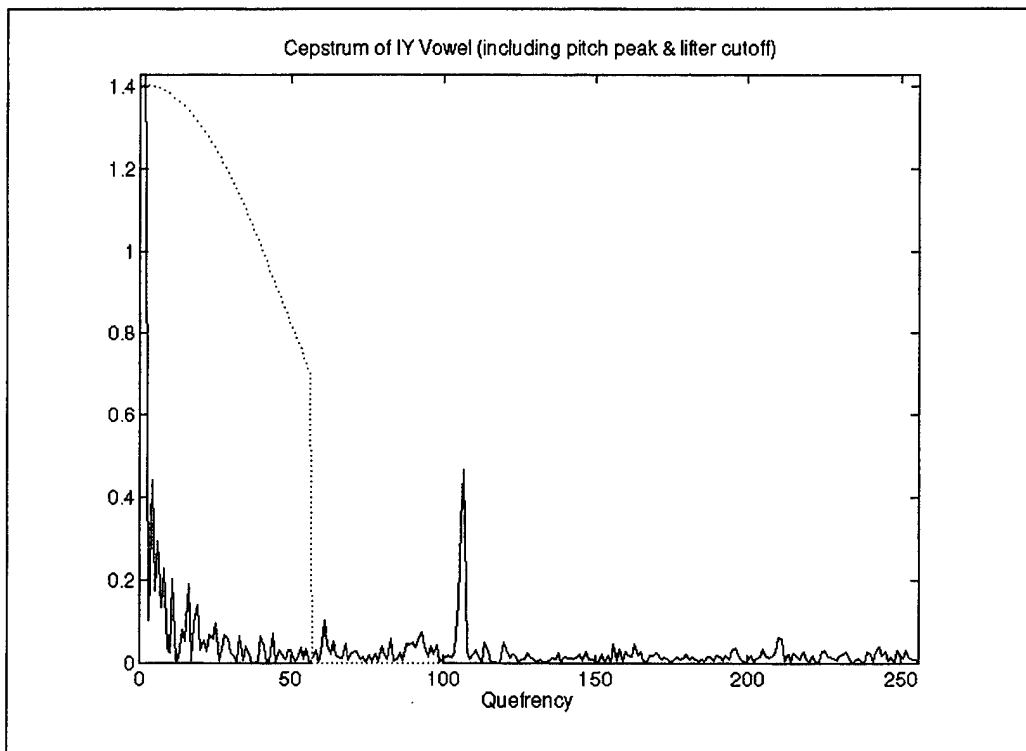
Figure 3.4. A cepstral version of Figure 3.2. The prominent spike near the center of the quefrency scale is a pitch peak. Articulator effects are concentrated at the left end of the quefrency scale. The dotted outline illustrates the location of the low-time lifter cutoff.
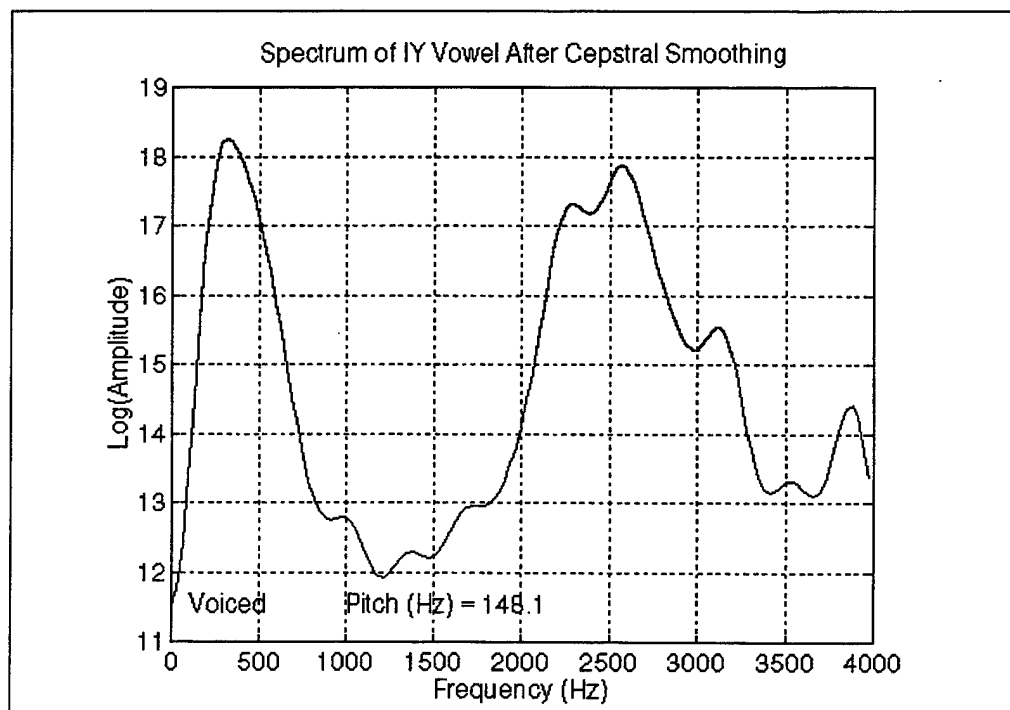


Figure 3.5. The spectrum of Figure 3.2 after cepstral smoothing. Note that the formant peaks are now easier to identify.

46
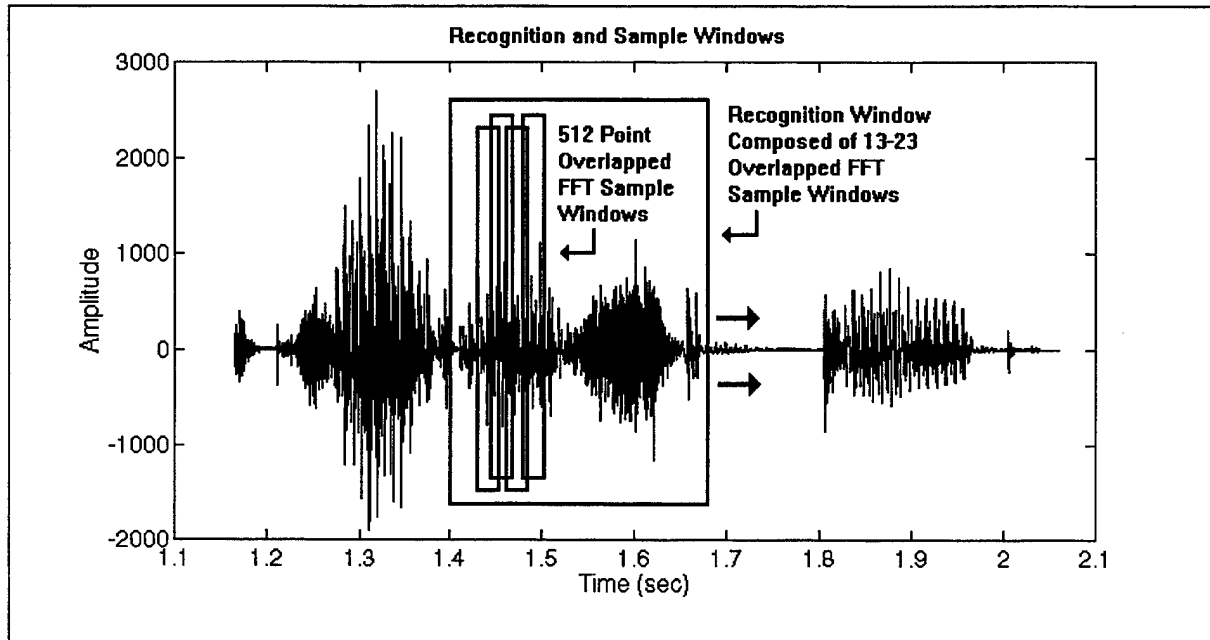
**Figure 3.6.** Windowing strategy used by the diphone recognizor. The recognition window is composed of a sequence of overlapped FFT (512 point) sample windows.

Figure 3.7 is an example of the feature surface used in this research. Each row is produced from a single sample window position. Twenty-nine columns of each row in the feature surface are devoted to cepstral information. The thirtieth column contains information related to the signal's short-time power (explained in more detail later in this section). The 30 columns of the feature surface are ultimately sequenced into a single feature vector for presentation to the neural network. Later paragraphs describe the number of feature surface rows used in a feature vector (its length in the time domain).

A speaker's volume can, of course, effect the magnitude of the spectral peaks. To reduce the impact of this potentially confounding variable, each cepstral coefficient in the feature vector actually represents the difference between that time period's coefficient and the coefficient of the previous time step. This is analogous to the use of capacitive coupling to remove a direct current offset from an oscilloscope's input. The delta or differenced cepstrum, as this technique is called, has been used with some success in previous research [Ref. 1]. It seems especially appropriate for diphone recognition, where the dynamic aspects of speech are relevant, rather than annoying.

47
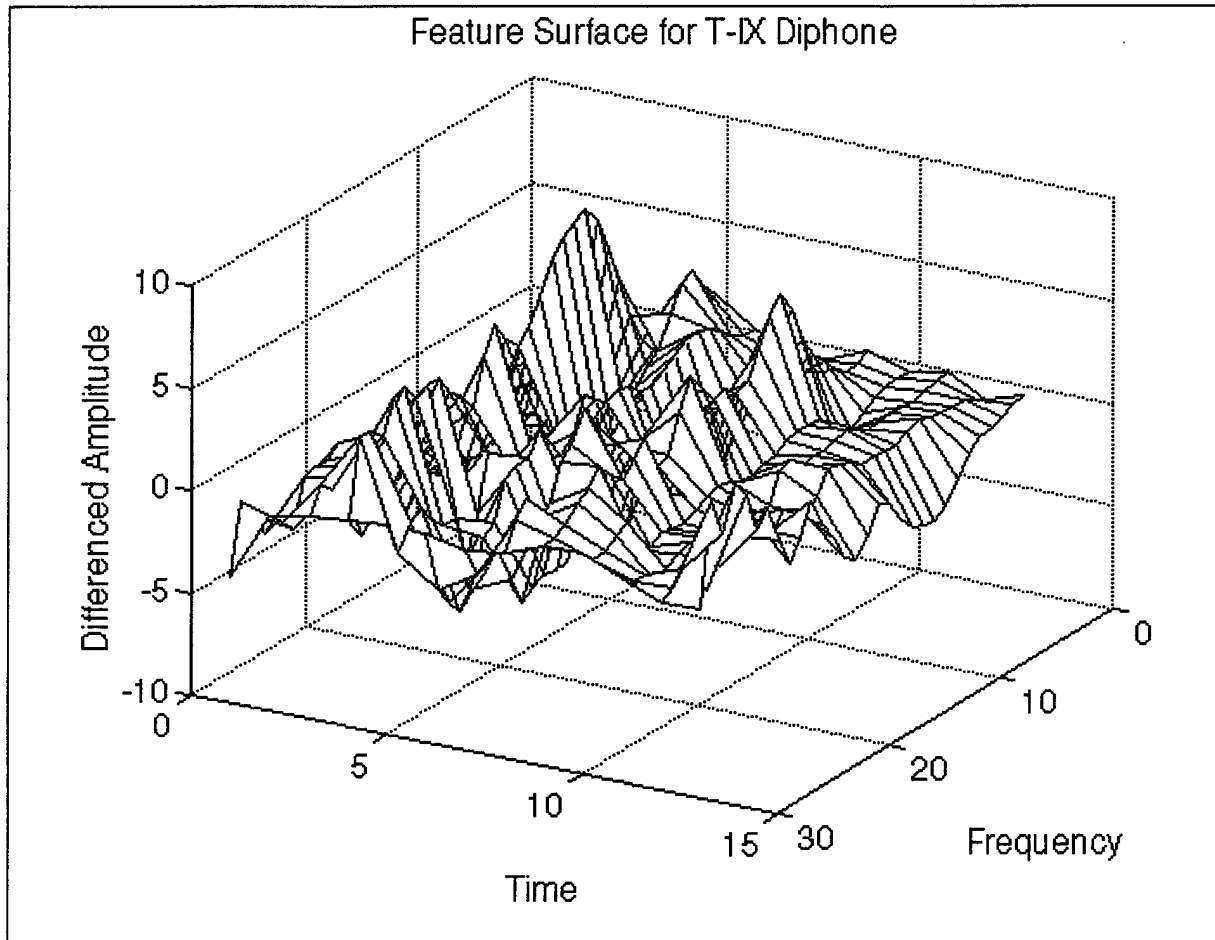
Figure 3.7. An example of the feature surface used for diphone recognition. The final design included 13 rows (about 289 ms) by 30 columns (29 cepstrally-smoothed, differenced frequency values and one differenced RMS amplitude measure). The columns were sequenced into a single (390 element) feature vector for presentation to the neural network.

## 2. Critical Bands

The 512 point FFT returns spectral information in 256 "frequency bins" of 31.25 Hz per bin (at a sampling rate of 16 KHz). Even if it were practical to include 256 inputs for each 32 millisecond slice of speech, it would not be desirable. The resulting feature vector would probably exhibit too much granularity. In particular, F1 frequencies (for the same vowel) differing by just 20 or 30 Hz would trigger entirely different inputs. On the other hand, evenly dividing the 8000 Hz spectrum between the 29 columns of Figure 3.7 would give each input a bandwidth of 276 Hz. The resulting loss in resolution would often cause the F1 frequencies of entirely different vowels to merge into a single input.

48

This dilemma is often resolved through the use of critical band filters. Psychoacoustic research has shown that human perception of a given frequency is influenced by a critical band of neighboring frequencies [Ref. 7]. At lower frequencies, the critical bands are narrower. As a result, the human auditory system exhibits finer resolution at lower frequencies.

In an effort to mimic critical bands, the FFT's output bins were unevenly distributed between the 29 columns of each feature surface row. Figure 3.8 shows the bandwidth of each of the 29 differenced cepstral inputs of the feature surface columns. The exact frequency assignments can be found in Appendix A (the FEATURES.M script file). The bandwidth for each center frequency was computed using the following approximation [Ref. 22]:

$$BW_i \ = \ 25 \ + \ 75 \ [ \ 1 \ + \ 1.4( \ F_c/1000 \ )^2 \ ]^{\ 0.69} \tag{3.5}$$

where $F_c$ is the center frequency.
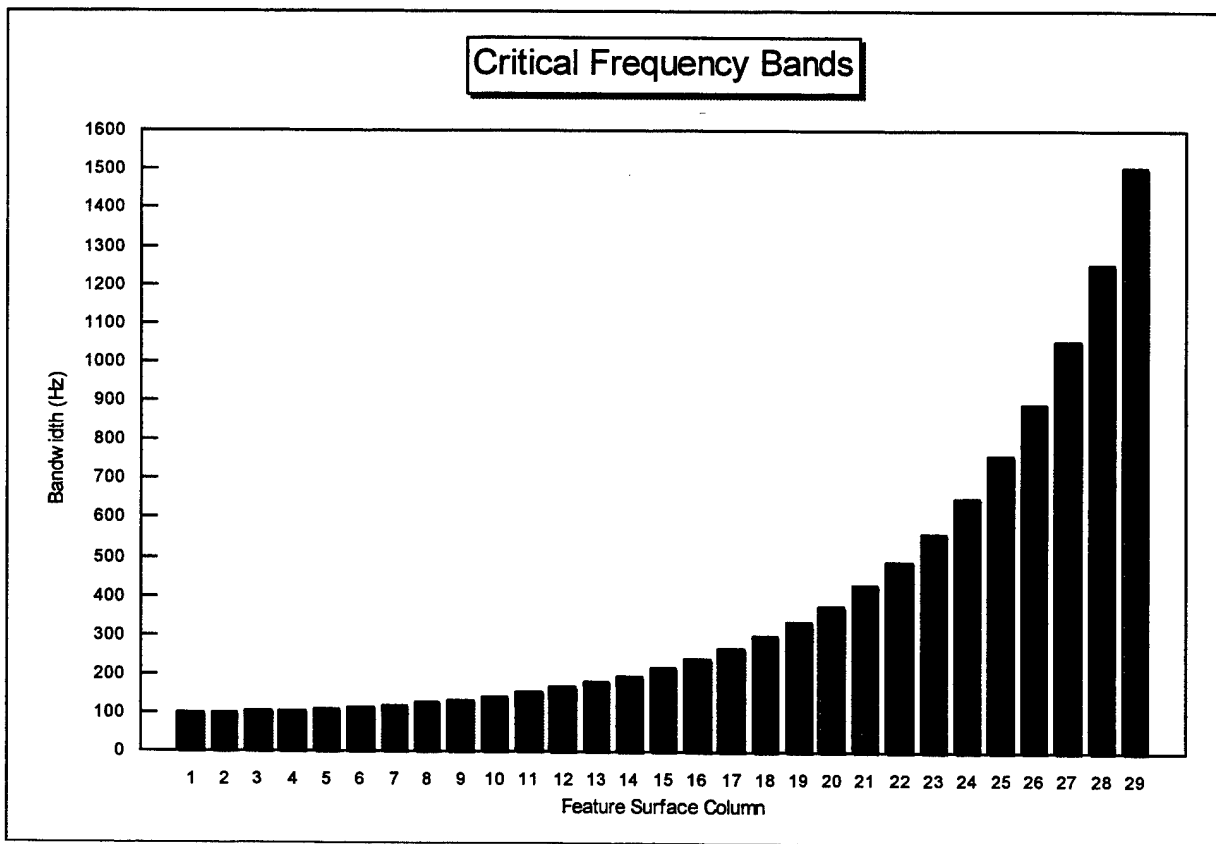


Figure 3.8. Critical frequency bands used in computation of feature surface. At low frequencies, fewer FFT frequency coefficients are averaged to produce the feature surface column values. Thus, spectral resolution is finer at low frequencies (as it is for human hearing).

49

Each center frequency was computed using the recursive formula:

$$FC_j = START_i + 0.75 (BW_i) + (BW_i / 2) \tag{3.6}$$

where $START_i$ is the previous band's start frequency.

This results in an overlap between bands ranging from about 25 percent at the low frequency end to about 29 percent at the high end. The first band's center frequency was defined to be 50 Hz with a bandwidth of 100 Hz. The actual value used for a given feature surface column is the mean of the contents of the FFT frequency bins falling in that column's bandwidth.

### 3. Short-Time Power Differences

Although the frequency domain is more useful in many cases, it does not seem wise to completely ignore the time domain. As one example of its value, recall that the speech envelope offers important clues to the presence of stop consonants. Thus, the thirtieth column of each feature surface row originally contained a measure of the differenced short-term power [Ref. 1] for that column's sample window position. For the ($n$-long) sample window at position $k$:

$$\Delta P(k) = ((\Sigma s^2(t))/n) - \Delta P(k-1) \tag{3.7}$$

where $s^2(t)$ is the square of the amplitude of the voice signal at time t.

As explained in Chapter IV, a better choice would have been the differenced RMS amplitude:

$$\Delta A_{RMS}(k) = \sqrt{\Sigma s^2(t)/n} - \Delta A_{RMS}(k-1) \tag{3.8}$$

### 4. Recognition Window Length

So far, no mention has been made of the length of the recognition window. This is actually a difficult question. Figure 3.9 is a histogram of maximum durations of diphones in the TIMIT SX and SI sentences (described later). Since the majority of diphones last no more than 500 milliseconds, the initial version of the feature surface contained 23 rows, with each containing spectral and short-time power information from a single 512 point (32 millisecond) sample

window. Because the windows were overlapped by one third, the recognition window was approximately 502 milliseconds wide (8036 samples).
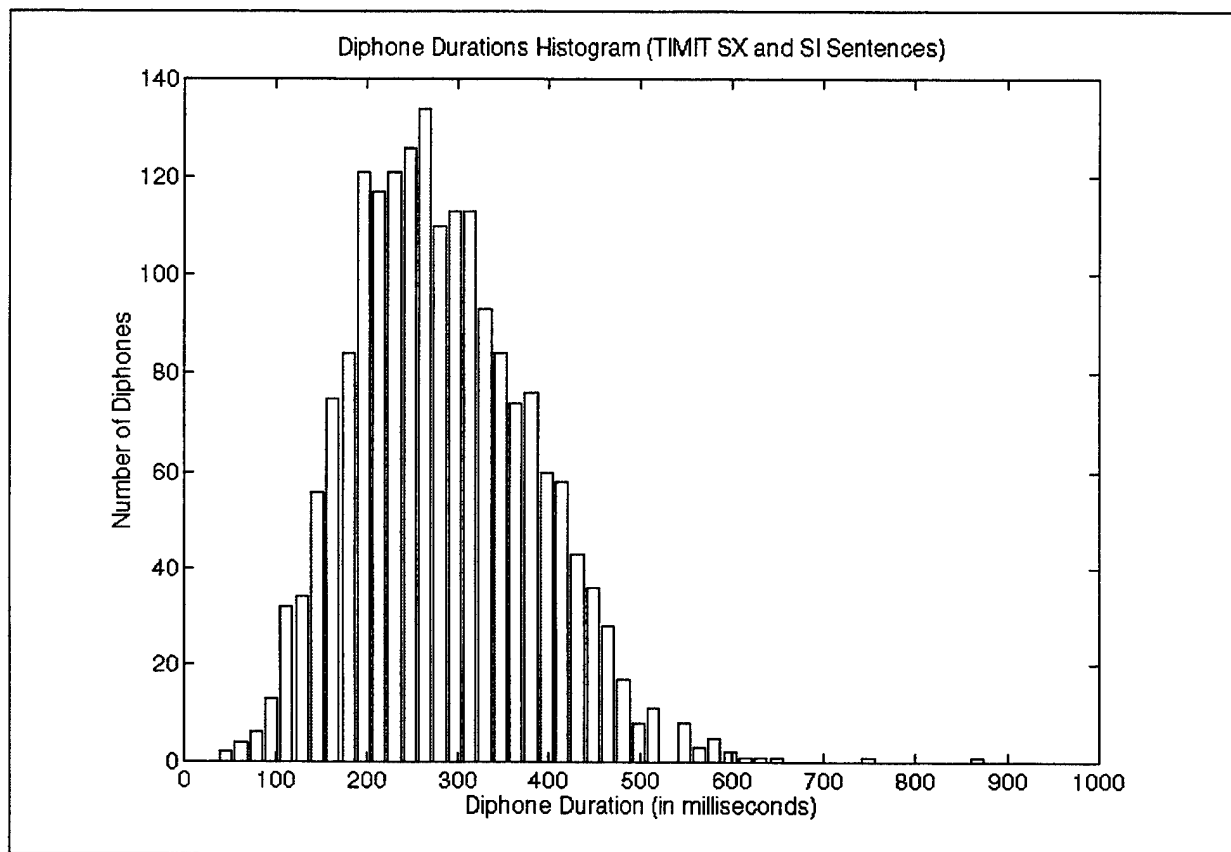


Figure 3.9. Histogram of durations of diphones in the TIMIT SX and SI sentences.

It is clear from Figure 3.9 that the vast majority of diphones are too short in duration to fill such a large recognition window. It was hoped that the neural network would, during training, reduce the weights of the outlying rows for short diphones. As explained in Chapter IV, it later became clear that the recognition window size should be reduced to approximately 289 milliseconds (4616 samples). This is actually not surprising, in that Kamm and Singhal found an input span of about 125 milliseconds worked best for phoneme recognition [Ref. 23], and a diphone consists of two phonemes.

## 5. Precomputation of Feature Vectors

Before leaving the topic of feature vector design, it should be mentioned that all of the feature vectors used for network training and testing were precomputed and saved in MATLAB data files. Each feature vector requires approximately 800,000 to one million floating point

51

operations to compute. And the same feature vectors were used during each of thousands of passes through the data (by hundreds of networks). It would have been very inefficient to recompute the feature vectors from the raw speech waveforms during each pass through the data (up to 20,190 feature vectors per pass).

## E. PITCH-BASED FREQUENCY NORMALIZATION

As mentioned earlier, it is desirable to remove from the feature vector any features that are not relevant. The speaker's pitch or fundamental frequency is such a feature. Pitch is a much better indicator of the gender (or age) of the speaker than of what is being spoken.

As mentioned in Chapter II, formant frequencies and pitch tend to be inversely proportional to vocal tract length. Several researchers; most notably Gersten, Wakita, and Neuburg; have attempted to normalize formant frequencies of individual speakers to improve recognition accuracy [Ref. 3]. Wakita's effort is especially interesting. He used measured formant data to estimate the speaker's vocal tract length. He then normalized the formant frequencies based on the vocal tract length estimate. Normalization reduced a vowel recognizor's error rate from 21 to 15.6 percent. Neuburg also attempted various scalings of the frequency scale based on the measured formant frequencies.

Unfortunately, the methods described in [Ref. 3] depend on identification of the speaker's formant frequencies and then normalization of those frequencies. These techniques add to the complexity of the preprocessing, they are difficult to implement in real time, and errors in formant identification could be expected to reduce accuracy.

Because both pitch and formant frequencies are inversely proportional to vocal tract length, it seems reasonable to use pitch itself as the basis for formant normalization. Therefore, an early part of this research was the measurement of pitch and formant frequencies for a variety of speakers. The results of this investigation are described in Chapter IV. For now, it is sufficient to say that pitch does appear to be a useful basis for normalization. Accordingly, the feature vectors used during this research were frequency-normalized using the procedure described in Chapter IV.

## F. TIME ALIGNMENT USING NEURAL NETWORK FEEDBACK

Because of the dynamic nature of diphones, correct time-alignment with the recognition window is doubly important. One cannot expect a neural network to easily see the similarities in time-shifted versions of a rapidly changing waveform. Fortunately, the TIMIT speech data used in this research includes time-aligned phonetic transcriptions that can be used to (approximately) center the diphone's phoneme junction in the recognition window. But the phonetic transcriptions are based on a semi-automated process of labeling phoneme boundaries that are themselves usually quite indistinct [Ref. 24]. Figure 3.10 illustrates that variability exists, even in diphones with sharply defined features, such as stop consonants. So it seems likely that recognition accuracy could be improved by more accurate time-alignment.
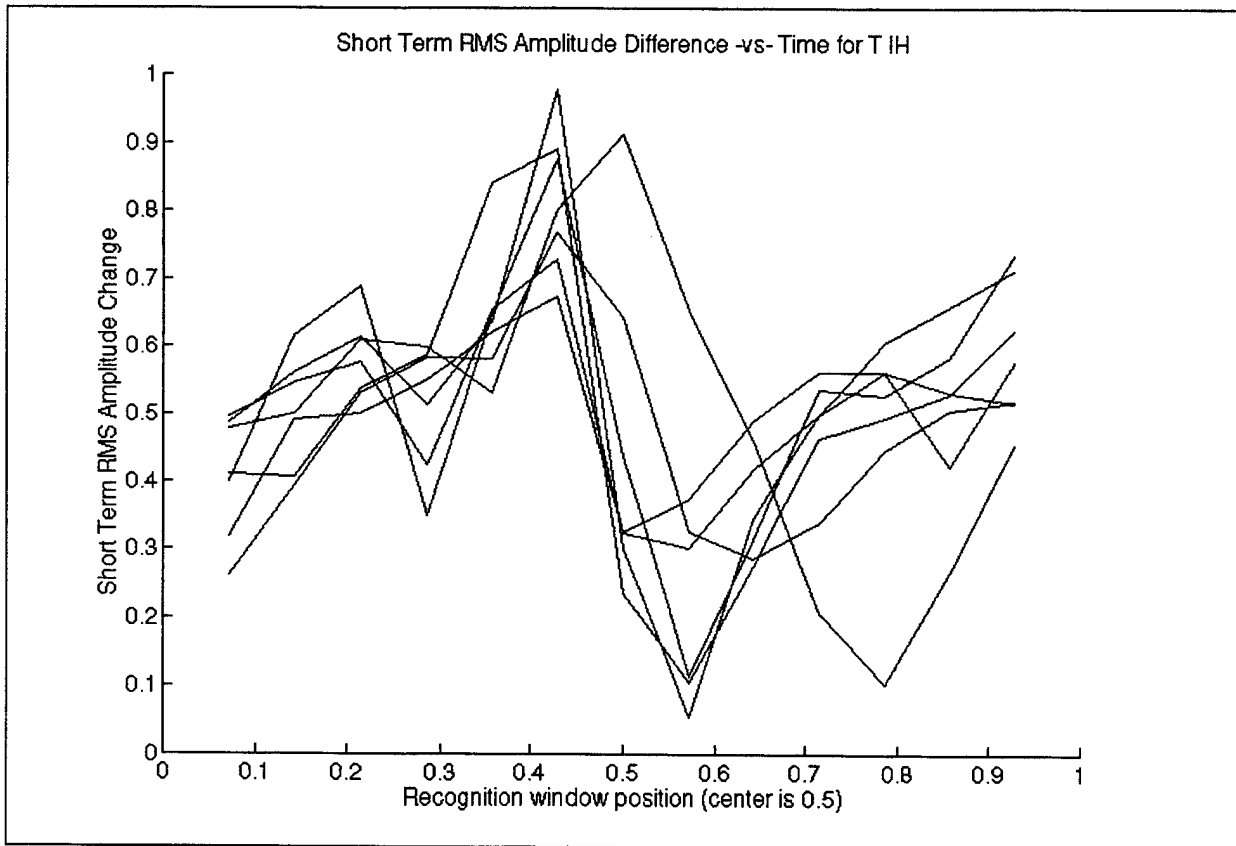


Figure 3.10. RMS amplitude columns of several feature surfaces for the same diphone (different speakers). Notice the variability in time alignment (based on TIMIT phonetic transcriptions).

Unfortunately, it is not easy to determine just what recognition window position is correct. One could conceivably compare the speech envelope with some diphone-specific template and use the window position that minimized the sum-squared error. But it would be necessary to

53

de-emphasize the tails of the window when aligning short diphones. Also, the process of developing templates would be challenging and error prone. Finally, the usefulness of such an approach might be hampered by the fact that not all important features are evident in the time domain. For example, voice onset time (period of time between a stop's release and the resumption of voicing) is a key feature for distinguishing voiced plosives from unvoiced plosives. Figure 3.11 shows a plot of one spectral band of the same diphones shown in Figure 3.10. Note that this band shows even greater variability than the RMS amplitude plot.
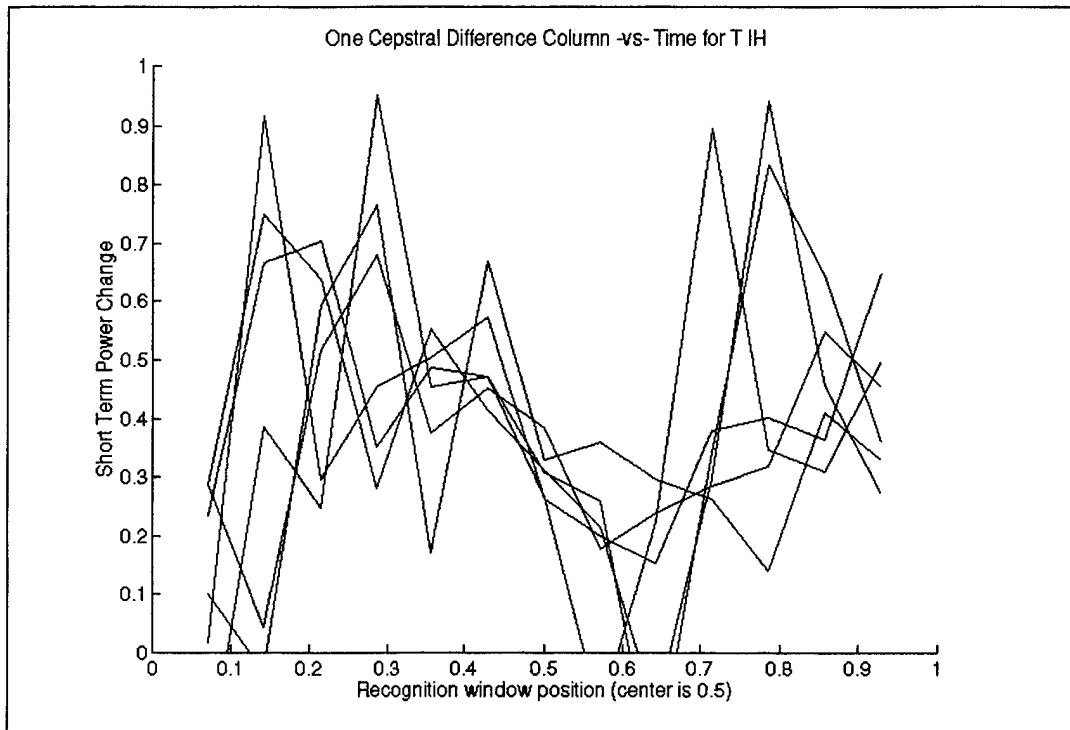


Figure 3.11. One feature surface cepstral difference column for same speech samples as Figure 3.10.

The time-alignment method used in this research makes use of feedback from the neural network itself. Specifically, the neural network is trained using transcription-based alignments until test set errors appear to have reached a minimum. Then, each feature vector is shifted through the recognition window until the correct output is maximized. The window position that maximizes the correct output is used to perform the weight adjustments. With this approach in mind, the precomputed feature surface files contained 27 rows. This allowed the initial 23 row recognition window to be shifted left or right by up to two time steps (about 43 milliseconds).

A second feature surface was also computed for each diphone. The second feature surface was computed by shifting the recognition window to a start position about 10 milliseconds earlier

54

in the speech sample. This meant that the two feature surfaces were interleaved in time (in case 20 millisecond time steps were too coarse). As a result, the effective temporal resolution was about 10 milliseconds. Because the time-alignment method evolved considerably during the conduct of this research, further details are deferred until Chapter IV.

## G. NEURAL NETWORK TOPOLOGY

Neural network design is still more an art than a science. There are few generally accepted theories to guide the designer of networks that must be used on real-world problems. Instead, researchers must often resort to rules of thumb and experimentation. In particular, it can be difficult to determine the number of hidden nodes required to solve a non-trivial problem. This section describes the architecture selected for this research, as well as the justification for that selection.

### 1. Single Hidden Layer Versus Two Hidden Layers

One key architectural question is the number of hidden layers to use in the neural network. Hornik, Stinchcombe, and White proved in 1989 that a neural network with a single hidden layer and sigmoidal activation function[*] can approximate any mapping to any degree of accuracy given enough nodes in the hidden layer [Ref. 25]. Though their existence proof offers no advice on the number of hidden nodes required, it is encouraging to know that a single hidden layer can theoretically handle any mapping. Nevertheless, other researchers have shown that two hidden layers will perform better (fewer nodes and better accuracy) on problems where the output classes cannot be separated into convex[**] open or closed regions [Ref. 26]. Some texts have even referenced Hornik's proof on one page, while stating (incorrectly) on another that a single hidden layer is only sufficient if the decision boundaries are convex.

While the theoretical arguments seem to favor two hidden layers on complex mappings, practitioners often argue for a single hidden layer [Ref. 27]. Because it is impractical to determine whether diphone classes can be bounded by convex regions in an input space that has

---

[*]   In 1991 Hornik removed the requirement that the transfer function be sigmoidal. It is only necessary that is be sufficiently smooth.

[**]   A convex region is a region in which any two points in the region can be connected by a line that does not leave the region [Ref. 12].

several hundred dimensions, a simple experiment was conducted in an effort to shed light on the convexity issue. Figure 3.12 shows a problem with three output classes. Note that none of the three classes can be bounded by a convex region without including members of another class. This problem was solved with a back-propagation neural network consisting of two inputs, 10 hidden neurons, and three output neurons. Figure 3.13 shows the classification lines defined by the trained hidden layer's weights. The *sum-squared error** (SSE) after 755 *epochs* (an epoch is one complete pass through the training data) was less than $10^{-6}$. While this example does not prove that a single hidden layer is sufficient for every problem, it does disprove the notion that networks with a single hidden layer are incapable of solving all problems with non-convex decision boundaries.
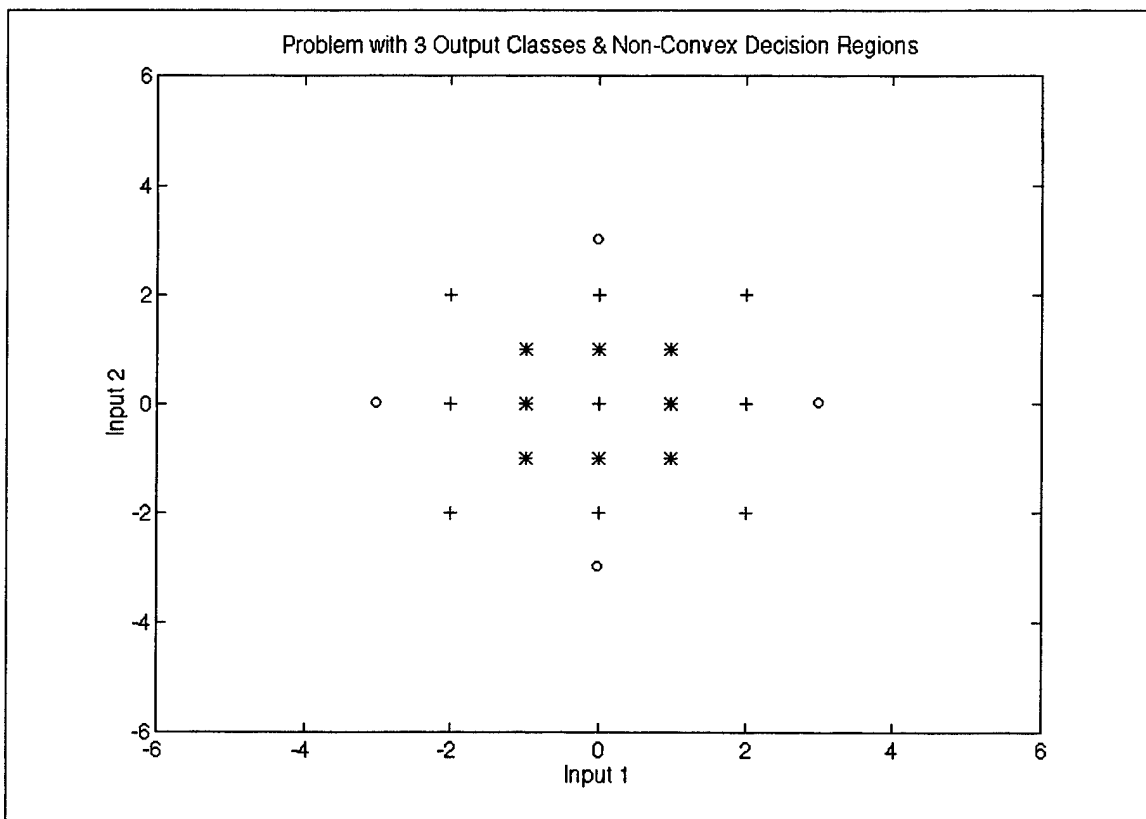


Figure 3.12. A classification problem with three output classes that cannot be separated using convex decision regions. A neural network with a single hidden layer proved capable of handling this problem.

---

*      A sum-squared error of zero would require that the correct neuron's output be a one and all other outputs zero. In practice, it is usually only necessary that the correct output be the strongest for correct classification. In this example, all input vectors were correctly classified long before epoch 755. The SSE error goal of $10^{-6}$ was arbitrarily selected.
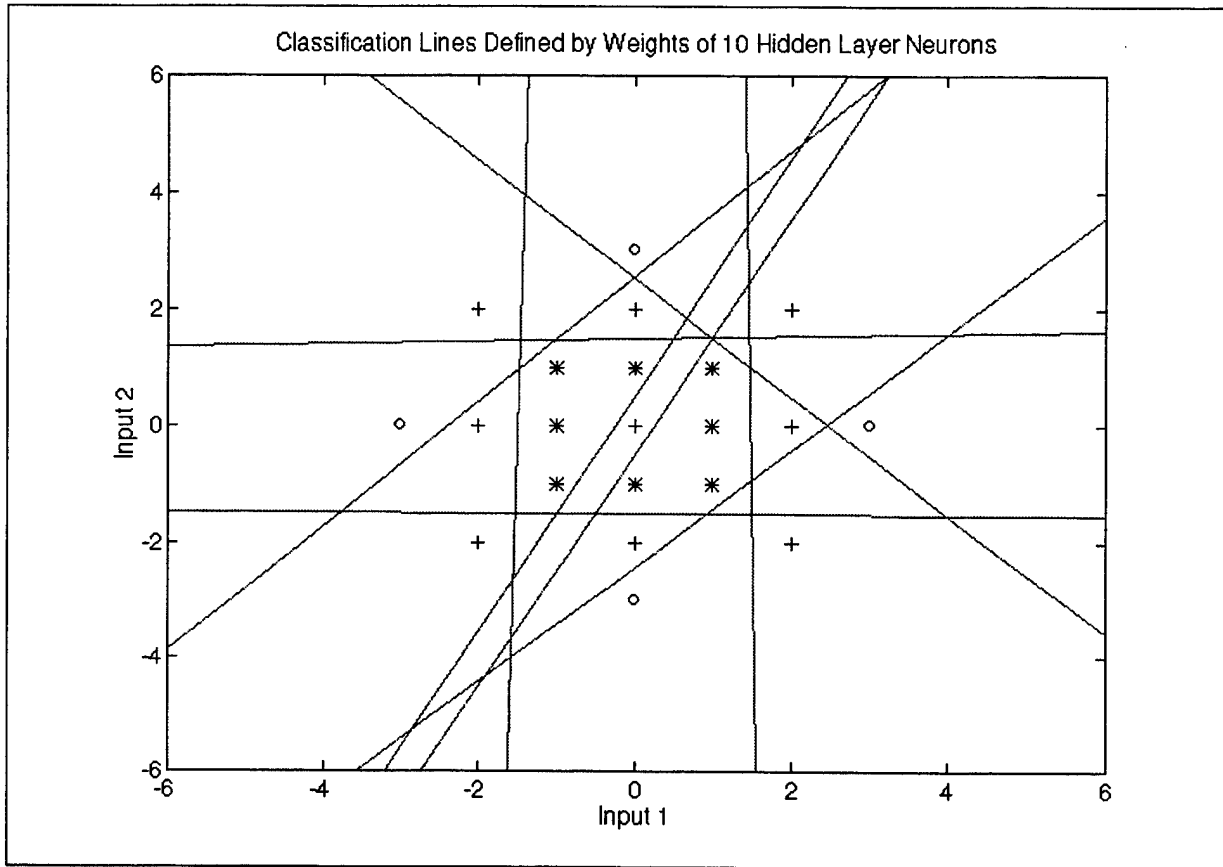
Figure 3.13. Classification lines defined by 10 sets of hidden layer weights. The output layer's input space has 10 dimensions so it cannot be plotted.

For comparison purposes, several networks with two hidden layers were trained on the same problem. After some experimentation, it was determined that at least five neurons in the first hidden layer and five neurons in the second hidden layer were required to solve the same problem. While this is the same number of hidden nodes as the network with just one hidden layer, the new network required 5436 epochs to reach the same error goal. Thus, networks with two hidden layers are apparently not always superior, even with non-convex decision boundaries. It should also be noted that this second network was constructed only after it was known that 10 neurons were sufficient for a single hidden layer. Armed with this knowledge, and the assumption that two hidden layers would require a total of nearly that many neurons, it was a fairly simple matter to come up with a working design.

Although two hidden layers may be better, at least theoretically, for some problems, a network with just one hidden layer has an important practical advantage. Specifically, it is only

necessary to determine the size of one layer. The size of the feature vector and the number of output neurons can be derived from the nature of the problem. But experimentation is the only sure way to determine the optimal number of hidden neurons. In the example problem just described, the network with a single hidden layer was first trained with 32 hidden nodes. The hidden layer was then reduced in size until the network failed to converge (at 9 or fewer neurons). The network with two hidden layers was actually harder to size and test because two hidden layers had to be sized.

Because the networks used in this research required up to five weeks of training time and trillions of floating point operations, it was important to minimize the number of network topologies to be tested. Primarily for this reason, the neural networks used in the remainder of this research consisted of just one hidden layer.

## H.  TRANSLATING DIPHONES TO WORDS

Whether using the diphone or the phoneme as a recognition unit, high accuracy at the sub-word level is no guarantee of similar accuracy at the word level. So a prototype lexical analyzer was constructed, using Common LISP, to determine if the assumed advantages of diphones were real. The following approach is used:

- Two interleaved recognition windows are stepped through the PCM speech file[*], each moving 21 milliseconds (342 samples) per time step. The resulting temporal resolution is about 10 milliseconds (170 samples).

- For each recognition window position, the three output neurons with the strongest outputs are identified. A text line is then appended to the output file, listing the identities and output values of these neurons, as well as the current window position (to serve as a "time stamp").

- A Common LISP program reads in the neural network's output as a LISP list. In cases where a given diphone appears in the one of the top three positions for more than one

---

[*]     Precomputed feature vectors were used during training only. TIMIT PCM speech files were used during word recognition tests.

consecutive time step, only the entry with the highest output value (the output peak) is retained.

- The remaining output peaks are then processed sequentially and in chronological order. Each diphone peak is assumed to potentially be the start of a word.

- For each word in the phonetic dictionary that begins with the current diphone, the lexical analyzer scans forward in the diphone stream, looking for diphones containing the remaining phonemes of that word. The search is bounded by maximum and minimum expected durations for the phoneme. In other words, any diphones containing the required phoneme must be tagged with times that make sense in view of the phoneme's expected duration. Otherwise, they are ignored.

- A word is written to the lexical analyzer's output if all of its phonemes appear in the diphone stream (and with reasonable time stamps). A confidence value (a simple average of the diphone recognizor outputs for the diphones in the word) and time stamp are also written to the output.

- Thresholds are used to prevent weak diphone recognizor outputs from triggering this search process. Similarly, a threshold is used to prevent the output of words with low overall confidence values.

- A second phonetic dictionary is maintained, sorted by the second phoneme in each word. This allows the lexical analyzer to spot words in the diphone stream even if the first two diphones are lost (taking the word's first phoneme with them).

Further details of this algorithm can be found in Chapter IV and Appendix C (primarily in RECOGNIZ.CL).

## I. TRAINING AND TEST DATA

Early speech recognition researchers were forced to build their own speech databases. Thus, a significant amount of time and money had to be invested in gathering training and test data before any useful research could be conducted. In an effort to solve this problem, DARPA

59

launched a large speech database development program in 1984 as part of its Strategic Computing speech program [Ref. 24]. One of the results of this effort was the TIMIT speech database, developed jointly by researchers at Texas Instruments (TI), the Massachusetts Institute of Technology (MIT), and SRI International.

The TIMIT database is currently distributed on CD-ROM through the National Institute of Standards and Technology (NIST). The database consists of a total of 6,300 sentences from 630 speakers, totaling over 5 hours of speech. The sentences are already partitioned into recommended training and test sets, with 27 percent of the speakers assigned to the test set. Test set speakers do not also appear in the training set. Both the test set and the training set are partitioned into eight dialect regions (New England, Northern, North Midland, South Midland, Southern, New York City, Western, and "Army Brat" [moved around]). Each dialect region is further partitioned by individual speaker, with ten sentences per speaker. Of the total, 70 percent of the speakers are male and 30 percent female. [Ref. 2]

To support the variety of research being conducted, three types of sentences were included in the TIMIT database. The dialect sentences (prefixed by SA in the database) were designed to expose dialectical variations between speakers. Only two such sentences are included in the database and each speaker recorded both. The phonetically diverse (SI) sentences were randomly selected from a variety of sources to generate diversity in sentence types and phonetic contexts. Each speaker read three SI sentences and no two speakers read the same sentence. Finally, the phonetically-compact (SX) sentences were hand-designed at MIT to be compact and provide good phonetic coverage. Each speaker read five SX sentences. [Ref. 2]

Because of their compactness and comprehensive phonetic coverage, the TIMIT SX sentences were selected for use in this research. Only sentences from the TIMIT training set were used for training (weight adjustments) and only test set sentences were used for testing.

Before the neural network feature vectors could be computed, it was necessary to identify the specific diphones and sentences to be used. A number of automated tools were developed to assist with this task. The ADA source code for these utilities can be found in Appendix B. Four speech data sets were ultimately used for neural network training. The first two sets, built around 19 and 44 diphones respectively, were used primarily for software testing and experimentation with various training algorithms. These two data sets will not be discussed in depth. The other

two speech data sets included samples of the 157 most common diphones in the SX sentences. These two data sets are described in more detail later in this section. Details on the mechanics of building the speech data sets can be found in the source code comments of Appendix B. For now, the following facts should suffice:

- A utility (DIPNUMS.ADA) was used to sift through the TIMIT phonetic transcriptions and determine how many and which diphones were available.

- It was assumed (based on intuition) that at least thirty occurrences of each diphone would be needed to train the neural networks to a reasonable degree of generalization. It was also assumed that 100 occurrences would be better.

- Only 157 diphones were common enough to occur at least 100 times in the TIMIT SX training set. These 157 diphones were therefore used to produce the two speech data sets described in the following subsections. The diphones themselves are listed in Appendix D.

## 1. The Medium-Sized Data Set

It would not have been wise to use an overly large data set for early neural network training attempts. A great deal of trial and error was needed to determine the number of hidden nodes required. It was also frequently necessary to refine the training methodology and determine the effect of a number of parameters described in Chapter IV. Since a common UNIX workstation requires approximately eight hours for a single pass through the large data set (described later), it was necessary to do much of the experimentation with a more modest data set. The medium training set consisted of 30 examples of each of the 157 diphones mentioned earlier (4710 files). Because some diphones are more common in the TIMIT training set than in the test set, most but not all of the 157 diphones occur 10 times in the medium test set (1568 files). The TIMIT file listings were "shuffled" before selection of all data sets, in an effort to achieve a more uniform gender and dialectical distribution (the selection procedure is described in Appendix B). Table 3.1 lists the demographics of the speakers included in the medium training and test sets.

61

| CATEGORY | TRAINING SET | TEST SET | TOTAL |
|---|---|---|---|
| MALE SPEAKERS / SAMPLES | 325 / 3338 | 110 / 1068 | 435 / 4406 |
| FEMALE SPEAKERS / SAMPLES | 136 / 1372 | 53 / 500 | 189 / 1872 |
| DIALECT REGION 1 SAMPLES | 398 | 108 | 506 |
| DIALECT REGION 2 SAMPLES | 820 | 219 | 1,039 |
| DIALECT REGION 3 SAMPLES | 784 | 247 | 1,031 |
| DIALECT REGION 4 SAMPLES | 663 | 262 | 925 |
| DIALECT REGION 5 SAMPLES | 682 | 265 | 947 |
| DIALECT REGION 6 SAMPLES | 310 | 106 | 416 |
| DIALECT REGION 7 SAMPLES | 792 | 228 | 1,020 |
| DIALECT REGION 8 SAMPLES | 261 | 133 | 394 |
| TOTAL DIPHONE SAMPLES | 4,710 | 1,568 | 6,278 |

Table 3.1. Description of Medium Data Set

## 2. The Large Data Set

The large training set consisted of 100 examples of each of the 157 diphones in Appendix D (15,700 files). Again, some diphones are more common in the TIMIT training set than in the test set. So most but not all of the 157 diphones occur 30 times in the large test set (4490 files). Table 3-2 lists the demographics of the speakers included in the large training and test sets.

| CATEGORY | TRAINING SET | TEST SET | TOTAL |
|---|---|---|---|
| MALE SPEAKERS / SAMPLES | 326 / 11106 | 110 / 3945 | 436 / 15051 |
| FEMALE SPEAKERS / SAMPLES | 136 / 4594 | 53 / 545 | 189 / 5139 |
| DIALECT REGION 1 SAMPLES | 1,315 | 294 | 1,609 |
| DIALECT REGION 2 SAMPLES | 2,702 | 686 | 3,388 |
| DIALECT REGION 3 SAMPLES | 2,597 | 691 | 3,288 |
| DIALECT REGION 4 SAMPLES | 2,270 | 727 | 2,997 |
| DIALECT REGION 5 SAMPLES | 2,293 | 808 | 3,101 |
| DIALECT REGION 6 SAMPLES | 1,112 | 291 | 1,403 |
| DIALECT REGION 7 SAMPLES | 2,629 | 660 | 3,289 |
| DIALECT REGION 8 SAMPLES | 782 | 333 | 1,115 |
| TOTAL DIPHONE SAMPLES | 15,700 | 4,490 | 20,190 |

Table 3.2. Description of Large Data Set

# J. TRAINING METHODOLOGY

The MATLAB Neural Network Toolbox and MATLAB script files were used to simulate and train the neural networks used in this research. Appendix A contains the script files specifically written for this purpose. The main training script (TRNBPX.M) operates as follows:

- At startup, the script file reads a file containing a number of training parameters, such as the number of hidden nodes and the recognition window width. It also loads a list of diphones defining the output classes. Finally, it opens two text files listing the files containing the precomputed diphone feature vectors.

- After one complete pass through the training set, the new weights are tested using the entire test set. Statistics for both training and test sets are then written to a text file. If the test results indicate that the test set errors have reached a new minimum, the weights are saved to a file. If, on the other hand, the new weights actually increase training set SSE by a predetermined amount (set by the "er" parameter), the weight changes are discarded and the epoch is repeated with a lower learning rate.

- The current state, including the latest weights, is written to a file to allow resumption of training even if the job terminates unexpectedly (with up to several weeks of CPU time invested in the network's weights, it would not be wise to tempt fate).

- The entire process (an *epoch*) is repeated until the error rate becomes satisfactory or the job is terminated.

It is, of course, quite expensive, in terms of CPU time, to run through the entire test set during every epoch. But test set errors can exhibit sharp drops and spikes from one epoch to the next. And training set errors are a poor short term indicator of test set errors. In fact, a drop in training set errors is often accompanied by at least a temporary rise in test set errors. Also, a neural network's ability to generalize (correctly categorize never-before-seen data) will suffer if the network is overtrained. The usual intuitive explanation is that the network learns too many details about the test set, details that do not contribute to correct classification of unfamiliar data.

63

Thus, a pass through the test set after every epoch is the best way to ensure that the best weights will be found.

## K. SUMMARY

Chapter III has outlined the plan under which this research was conducted. The next chapter describes the results of carrying out that plan.

# IV. RESULTS

## A. INTRODUCTION

Not surprisingly, some details of the methodology evolved as results accumulated and were analyzed. However, this research largely supports the assumptions and expectations of Chapter III. The following sections describe the key results. In particular, frequency normalization results, diphone and word recognition statistics, and the impact of feedback-based time alignment are described. Emphasis is also given to reporting the apparent impact of various neural network training alternatives.

## B. FREQUENCY NORMALIZATION

As mentioned in Chapters II and III, both pitch and formant frequencies vary inversely with the speaker's vocal tract length. Pitch can be detected fairly easily in a voiced speech signal*. So a study was conducted to determine if pitch would be a useful basis for formant frequency normalization. If so, speaker independent recognition accuracy could be expected to improve.

The IY vowel was selected as the subject of this study. As indicated in Figure 2.5, this vowel is characterized by good separation between its formant frequencies. After cepstral smoothing, the formant frequencies were visually identified and recorded along with the speaker's pitch. A total of 209 data points were taken from 70 speakers, 57 samples from 21 females and 152 samples from 49 males. All samples were taken from SX sentences and Dialect Region 2. The following subparagraphs describe the data and the normalization method based on that data.

### 1. Distribution of Formant Frequencies

Figures 4.1, 4.2, and 4.3 show scatter plots for F1, F2, and F3 respectively. It appears from the plots that pitch frequency and formant frequency are related. Table 4.1 lists key statistics for formant frequencies of the three data sets. The raw data is listed in Appendix H.

---

\* Cepstral pitch detection was used in this research, backed up by the Average Magnitude Difference Function [Ref. 1]. The code in Appendix A can be consulted for specifics on the technique used.
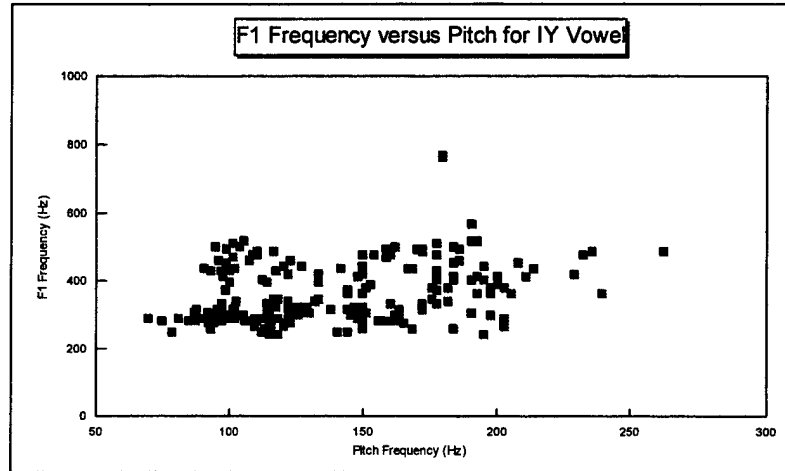
Figure 4.1. Plot of F1 frequency versus pitch for 209 samples of the **IY** vowel (70 speakers).
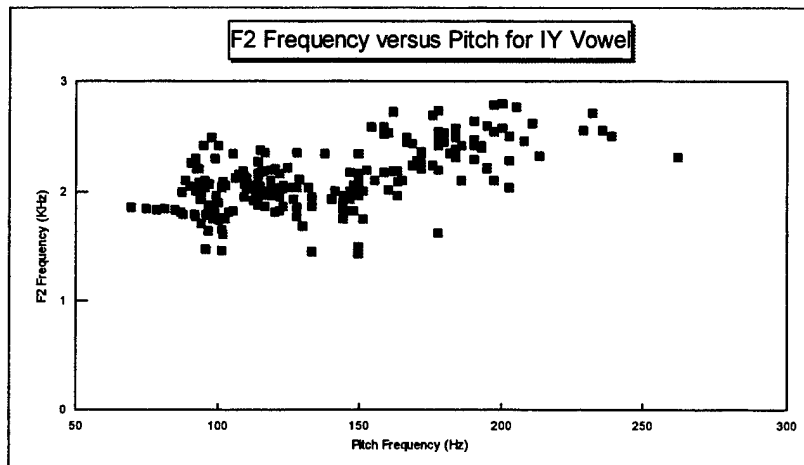


Figure 4.2. Plot of F2 frequency versus pitch for the same speakers as in Figure 4.1.
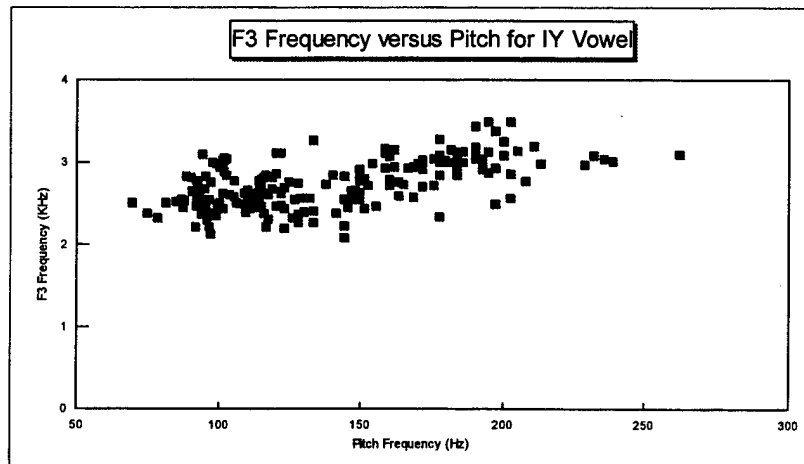


Figure 4.3. Plot of F3 frequency versus pitch for the same speakers as in Figure 4.1.

66

| FORMANT | SAMPLE MEAN (Hz) | SAMPLE VARIANCE | PITCH/FORMANT CORRELATION COEFFICIENT (r) | REGRESSION COEFFICIENT OF DETERMINATION ($r^2$) | REGRESS. X COEFFIC. |
|---|---|---|---|---|---|
| F1 | 362 | 7,904 | 0.35 | 0.13 | 0.08 |
| F2 | 2,125 | 80,262 | 0.62 | 0.38 | 4.48 |
| F3 | 2,729 | 81,669 | 0.58 | 0.34 | 4.25 |

Table 4.1. Statistics from 209 Measurements of **IY** Formant Frequencies

The correlation coefficients, and the plots, indicate a moderate correlation between pitch and formant frequencies, particularly F2 and F3. The correlation between pitch and F1 is weaker. Furthermore, the weak regression coefficients of determination ($r^2$) indicate that the relationships are nonlinear. This is consistent with previous research on the subject [Ref. 3]. Finally, the slopes of the regression lines are steeper for F2 and F3 than for F1. This rules out a simple shift of the frequency axis proportional to pitch changes. Such a shift would overcorrect F1 or undercorrect F2 and F3.

## 2. Normalization Method Used

Because the neural network must recognize formant peaks in the overall spectrum, it was necessary to use a normalization method that did not depend on detection of the formants themselves. Instead, each frequency component of the FFT output was shifted by an amount proportional to both the frequency being shifted and the difference between that speakers pitch and a "standard" pitch (100 Hz). Thus, the expected "error" in a given frequency component was modeled by:

$$\Delta f_{formant} \approx k \, f_{measured} \, \Delta f_{pitch} \qquad (4.1)$$

where k is a constant of proportionality. If the speakers measured pitch is 100 Hz, the right side of the equation would equal zero and no correction would be made. For any other pitch, the measured frequency would be corrected by adding the left side of Equation 4.1 to the measured frequency. The constant of proportionality was estimated by solving for k as follows:

$$\Delta f_{formant} \, / \, \Delta f_{pitch} \approx k \, f_{measured} \qquad (4.2)$$

$$(\Delta f_{formant} \, / \, \Delta f_{pitch}) \, / \, f_{measured} \approx k \qquad (4.3)$$

The left side of Equation 4.2 can be approximated by the slopes of the regression lines in Table 4.1. A value for $f_{measured}$ is obtained for each formant by plugging a pitch of 200 Hz into the equation for that formant's regression line. The three resulting equations form the divisors and the regression slopes are the dividends in Equations 4.4 through 4.6. For the first formant:

$$\text{F1:} \quad 0.805 / (0.805 \times 200 + 251.4) = 0.00195 \qquad (4.4)$$

where 251.4 is the regression line's intercept (Table 4.1). For F2:

$$\text{F2:} \quad 4.48 / (4.48 \times 200 + 1508) = 0.00186 \qquad (4.5)$$

where 1508 is the intercept. Finally, for F3:

$$\text{F3:} \quad 4.25 / (4.25 \times 200 + 2143) = 0.00142 \qquad (4.6)$$

where 2143 is the intercept.

Because the F2 regression line best fit the data, the value from Equation 4.5 was given more weight and 0.00185 was the proportionality constant actually used to perform normalization. Equation 4.1 yields an estimate of the difference between measured frequency and normalized frequency. Thus, the frequency components from the FFT can be normalized by:

$$f_{normalized} = f_{measured} + 0.00185 \, f_{measured} (100 - f_{pitch}) \qquad (4.7)$$

The right addend in Equation 4.7 is simply Equation 4.1 with the proportionality constant from Equations 4.4 through 4.6. The parenthesized quantity is the difference between the speaker's pitch and the "standard" pitch of 100 Hz ($\Delta f_{pitch}$ in earlier equations). In words, the measured frequency is corrected by an amount proportional to the measured frequency and the difference between the speaker's pitch and a standard pitch (again, 100 Hz was used).

### 3. Impact of Normalization

Figures 4.4 and 4.5 show the distribution of F2 frequencies for the same 209 F2 data points, before and after normalization using Equation 4.7. While considerable variance remains, the distribution after normalization is clearly tighter. As shown by Table 4.2, normalization also reduces the variance of F1 and F3 frequencies for the same data.

68
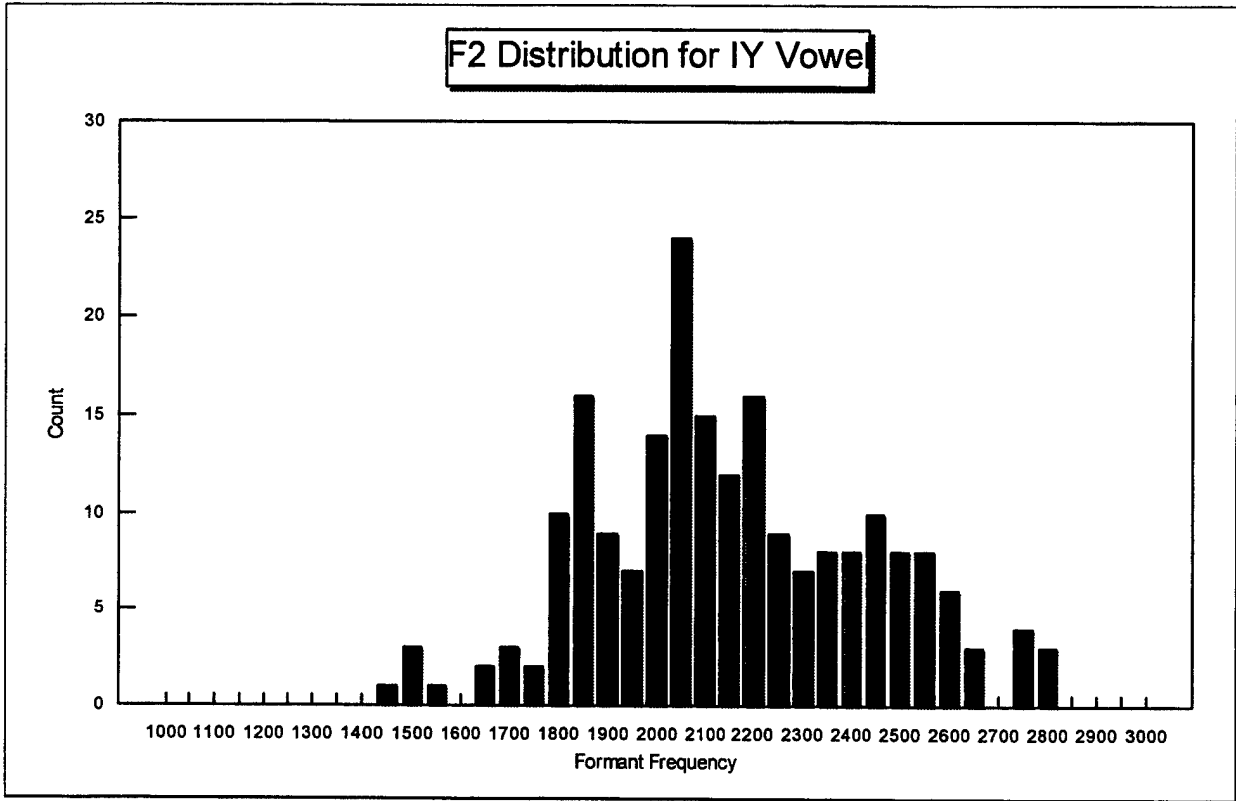
Figure 4.4. Distribution of F2 frequencies (**IY** vowel) before normalization.
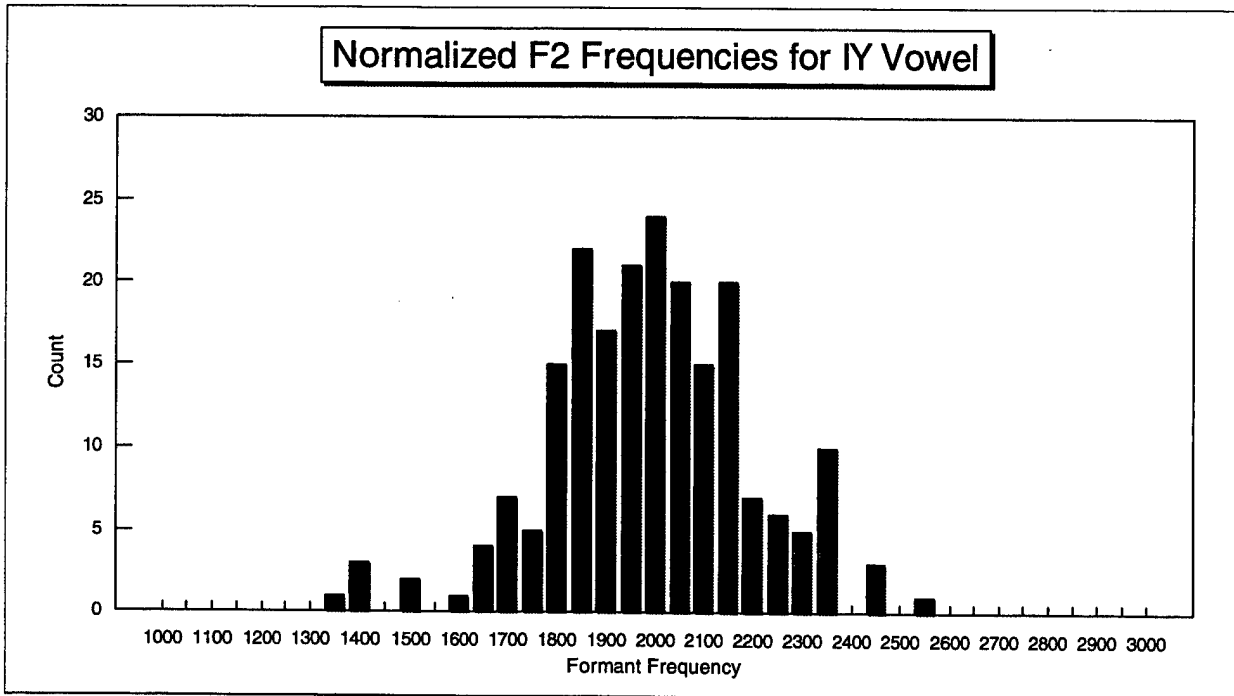


Figure 4.5. Distribution of F2 frequencies after pitch-based normalization.

69

| FORMANT | VARIANCE BEFORE NORMALIZATION | VARIANCE AFTER NORMALIZATION |
|---------|-------------------------------|------------------------------|
| F1      | 7,904.5                       | 5,820                        |
| F2      | 80,261.7                      | 42,111.8                     |
| F3      | 81,669.1                      | 49,325.2                     |

Table 4.2. Variance of Formant Frequencies for **IY** Vowel Before and After Frequency Normalization

No attempt was made to collect data on vowels other than **IY**. It was assumed that all vowels would exhibit similar relationships between pitch and formant frequencies. Instead, feature vectors were produced both with and without frequency normalization. Identical neural networks were then trained on the same speech samples from the same speakers, the only difference being the use or absence of normalization. In every case the neural network trained with normalized frequencies outperformed the network trained without normalization. Figure 4.6 shows the learning curves from the largest such comparison. The solid line shows the test set error rate for the network trained with normalized data. The dashed line shows the test set errors for an identical network trained without normalization. In this case, the network without normalization suffered over 34% more errors (using the "top three" error metric, explained in the next section).
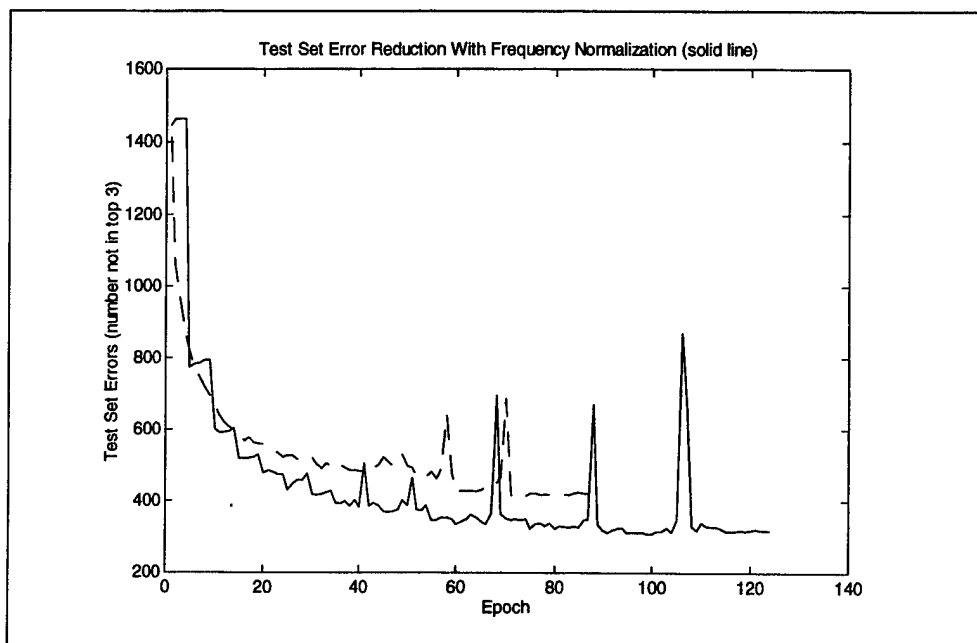


Figure 4.6. Error plots for identical networks with and without frequency normalization. Network with normalization (solid line) converged to lower error rate.

The relationship modeled in Equations 4.1 and 4.7 may have very little to do with the true relationship between pitch and formant frequency. But, judging from the data gathered so far, the frequency normalization method described in this section is at least useful.

## C.  DIPHONE RECOGNITION RESULTS

### 1. Error Metrics

Two metrics were used to evaluate the accuracy of all neural networks trained in conjunction with this research:

#### a. Percent Correct

This metric represents the percentage of test set diphones which triggered the correct neural network output. In other words, if the correct output neuron had the strongest output, the diphone was correctly recognized. Test set diphones (and speakers) were never used to train (adjust weights of) the networks. So these figures should give a much more accurate idea of a network's generalization ability than the training set figures. Therefore, all comments and accuracy figures refer to the test set unless specified otherwise.

#### b. Percent in Top Three

This metric is the percentage of diphones which activated the correct output neuron strongly enough to place it among the top three neuron outputs. This figure is useful for two reasons:

- The lexical analyzer (described in Chapter III) is only shown the network's top three outputs for each recognition window position. Thus, the diphone must be in the top three to be useful at the next stage of recognition. Of course, the lexical analyzer could be redesigned to scan the top four or five neural network outputs. But that would increase its own search space considerably. Thus, the top three figure is a tradeoff between the diphone recognizor and the lexical analyzer. If the diphone recognizor top three percentage is too low some of the burden must be shifted to the lexical analyzer.

71

- The top three metric is commonly used in speech recognition research. So its use here will facilitate comparisons between diphone and phoneme recognition results.

Although sum-squared error (SSE) was recorded for every epoch of each network, it is not often quoted in this thesis. In many cases, sum-squared error drops were accompanied by increases in test set errors.

## 2. Results

The first entry in Table 4.3 describes the most accurate neural network of those trained on the large and medium-sized data sets (157 diphones). In short, this network correctly recognized 70.2% of the test diphones. For the same network, 89.0% of the correct diphones were among the top three outputs.

Table 4.3 also lists key statistics from other selected neural networks trained with the medium-sized and large data sets described in Chapter III. The networks included in this table were chosen because they seemed to illustrate the impact of various training alternatives. For example, the apparent impact of frequency normalization can be seen by comparing the third and fourth networks in the table. The third network was trained without frequency normalization. The fourth network was trained with normalization. The following comments also apply to the table:

- The numbers in the first column of the table are included only to simplify references in the text to a particular network. They have nothing to do with a ranking of the results or the sequence in which the networks were trained.

- The training set accuracy figures are for the same epoch as the test set figures. It proved possible to achieve training set accuracy of 100% (or very nearly 100%) on all of the data sets. But doing so requires overtraining, at the expense of test set accuracy.

- Training was stopped when test set accuracy no longer appeared to be improving. It is sometimes difficult to judge when progress has truly ended. So it is theoretically possible that some networks were stopped too soon.

72

| ID # | Data Set | Test Correct | Test Top 3 | Train Correct | Train Top 3 | Frequency Normalized | Time Alignment | RMS Amplitude | Window Width | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | L100 | 70.2% | 89.0% | 98.2% | 98.4% | YES | YES | YES | 13 | |
| 2 | L100 | 69.4% | 87.9% | 97.8% | 98.0% | YES | YES | NO | 13 | |
| 3 | M30 | 54.5% | 73.5% | 87.2% | 87.3% | NO | NO | NO | 13 | Compare w/4 |
| 4 | M30 | 56.8% | 80.3% | 94.3% | 94.4% | YES | NO | NO | 13 | Compare w/3 |
| 5 | M30 | 52.0% | 72.0% | 84.7% | 85.1% | NO | NO | NO | 13 | Compare w/8 |
| 6 | M30 | 59.4% | 78.2% | 90.6% | 90.7% | YES | NO | NO | 13 | Seed = 0 |
| 7 | M30 | 55.3% | 77.8% | 93.1% | 93.1% | YES | NO | NO | 13 | Seed = 5 |
| 8 | M30 | 58.0% | 79.0% | 91.1% | 91.1% | YES | NO | NO | 13 | |
| 9 | M30 | 55.8% | 77.2% | 89.9% | 89.9% | YES | NO | NO | 13 | Long initial set |
| 10 | M30 | N/A | N/A | 43.3% | 44.6% | YES | NO | NO | 13 | Inputs batched |
| 11 | M30 | 59.6% | 80.1% | 98.9% | 99.0% | YES | YES | NO | 13 | Shuffled often |
| 12 | M30 | 61.4% | 81.8% | 99.8% | 99.8% | YES | YES | NO | 13 | Shuffled once |
| 13 | M30 | 58.3% | 79.1% | 92.7% | 92.8% | YES | NO | NO | 11 | |
| 14 | M30 | 58.0% | 79.0% | 91.1% | 91.1% | YES | NO | NO | 13 | |
| 15 | M30 | 56.1% | 77.2% | 94.1% | 94.2% | YES | NO | NO | 15 | Too wide |
| 16 | M30 | 56.8% | 80.3% | 94.3% | 94.4% | YES | NO | NO | 13 | No initial set |
| 17 | M30 | 60.7% | 81.6% | 91.4% | 91.5% | YES | NO | NO | 13 | Compare w/16 |
| 18 | M30 | 57.1% | 82.5% | 99.8% | 99.9% | YES | YES | NO | 13 | Early feedback |
| 19 | M30 | 62.5% | 83.5% | 99.1% | 99.2% | YES | YES | YES | 13 | RMS vice STP |

Table 4.3. Key Statistics from Selected Neural Network Diphone Recognizors

## 3. Caveat on Neural Network Comparisons

In the discussions that follow, every effort has been made to compare results between networks that are identical in every respect except the parameter at issue. For example, the third and fourth networks in the table were initialized with the same random weights and they were trained on the same speech samples from the same speakers. They had the same number of hidden neurons and both networks were trained without feedback-based time alignment. The only significant difference was in the use of frequency normalization for the third network. However, it is wise to remember that two identical networks can achieve very different results simply by starting at different points on the error surface. Networks 6 and 7 in Table 4.3 illustrate this point. The only difference between these two networks is in the random seed used to initialize MATLAB's pseudo-random number generator prior to initialization of the weights. Network 6's random seed was set to zero and Network 7's was set to five. Judging from the

learning curves, Network 7 fell into an inferior local minimum and never achieved the accuracy of Network 6.

To eliminate this potentially confounding variable, and to make any particular network's course of training repeatable, the networks trained in this research all started with random seeds of zero unless specified otherwise. But this is still no guarantee that a particular comparison will reliably indicate the impact of a given parameter. Even networks with the same initial weights can fall into different local minima if something causes them to take different paths down the error surface. In one case, a network that had been training for several weeks was interrupted by the failure of the host's hard drive. Training was resumed on another host using a backup copy of some earlier weights. But the second host never achieved the accuracy level of the first. The second network took a different path down the error surface because the training algorithm uses an adaptive learning rate and a momentum term. When training was interrupted the learning rate dropped back to the default and the momentum term was zeroed out for the first epoch. This caused the second network to deviate from the path taken by the first. Unfortunately, the second path was inferior. It would be possible to reproduce the results of the first run, given some patches to the training code and a few more weeks of training time. But some other network might just as easily benefit from an unplanned interruption.

Although the preceding discussion suggests that comparisons between neural network runs are risky, trends are difficult to ignore. None of the conclusions described in the following subsections are based on a single comparison. For example, Networks 5 and 8 show essentially the same frequency normalization advantage as do Networks 3 and 4. And early test runs on smaller data sets support the same conclusions. Similarly, the advantages of feedback based time alignment were apparent in dozens of training runs. Again, the comparisons discussed in the text are based on careful matching of training parameters, including factors such as the momentum term and error ratio used by MATLAB's training algorithms.

The impact of frequency normalization has already been discussed. The following subsections describe the apparent impact of a number of other key training alternatives.

## 4. Number of Output Classes

Although all of the entries in Table 4.3 describe neural networks trained on 157 diphones, a number of smaller networks were trained to recognize smaller diphone sets. Table 4.4 shows the highest accuracy rates achieved for the three diphone set sizes. The *Small IH* data set training set consisted of 30 examples of each of 19 diphones, all starting with the vowel IH. The *Small IH-T* data set consisted of 30 examples of each of 44 diphones, all starting with the IH vowel or the T consonant (thus, it is a superset of the Small IH data set). The medium data set was described in Chapter III. In hindsight, some of the training techniques used on the smaller networks were inferior. Yet the smaller networks achieved noticeably higher accuracy levels. This issue is revisited in Chapter VI and the next paragraph's discussion of input batching.

| DATA SET | NUMBER DIPHONES | TRAIN / TEST SAMPLES | TRAINING CORRECT | TRAINING TOP 3 | TEST CORRECT | TEST TOP 3 |
|---|---|---|---|---|---|---|
| SMALL (IH) | 19 | 570 / 181 | 100% | 100% | 84.0% | 100% |
| SMALL(IH-T) | 44 | 1320 / 397 | 100% | 100% | 71.5% | 95.7% |
| MEDIUM | 157 | 4710 / 1568 | 99.1% | 99.2% | 62.5% | 83.5% |

Table 4.4. Comparison of Accuracy Levels for Neural Networks with Varying Numbers of Output Classes

## 5. Size of Training Set

As expected, networks trained on the large data set performed better on their test sets than networks trained with the medium data set. Network 1 in Table 4.3, for example, was trained using 100 samples of each diphone (15,700 input vectors). Network 19, identical in all other respects, was trained using just 30 samples of each diphone (4,710 input vectors). The larger training set clearly improved Network 1's generalization ability.

## 6. Batching of Weight Changes

Weight changes from each test set input vector are often accumulated until the end of an epoch and then all applied at once. This is commonly known as *input* or *weight change batching*. This method is the default in MATLAB's Neural Network Toolbox. It was also used successfully in training a number of networks to recognize the two smaller diphone sets. But dozens of networks failed to converge during batched input training on the medium data set.

Training was attempted with hidden layer sizes ranging from 64 to 512 neurons. Figure 4.7 shows the SSE plot (*learning curve*) for one such network with 256 hidden neurons. Note the abrupt leveling off of the curve at about the tenth epoch. This is typical of the neural networks that became stuck in local minima. For comparison purposes, the dashed line in the same figure shows the SSE plot for a later network which converged to a much lower minimum. The principal difference in the second network is the absence of weight change batching. In the second network, and all of the networks successfully trained on the medium-sized and large data sets, weight changes were made immediately after each input vector.
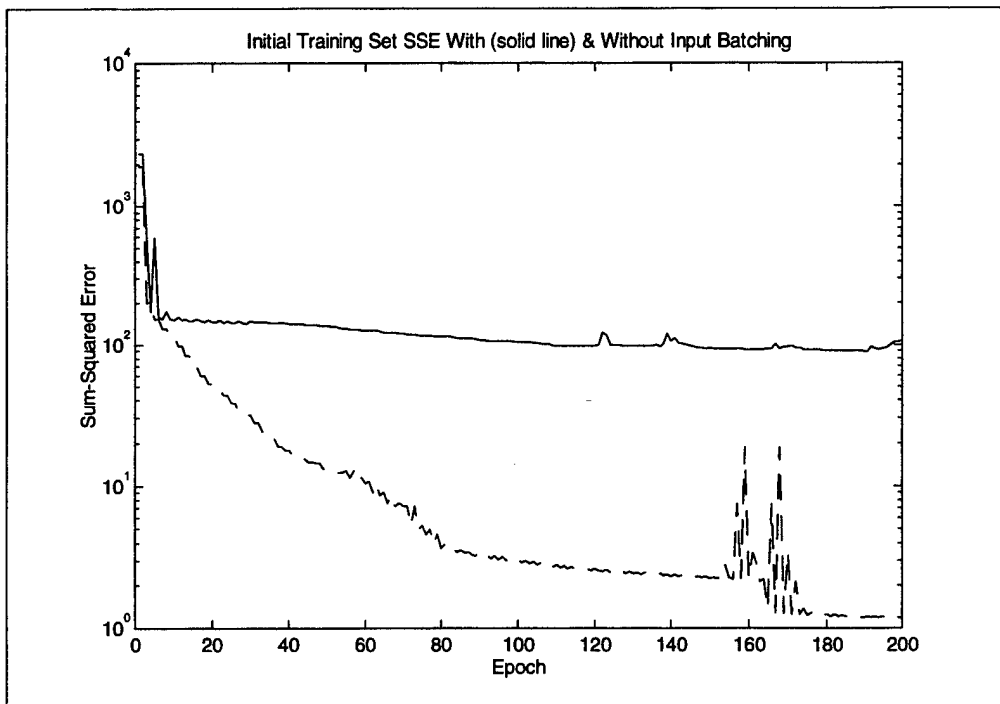
Figure 4.7. Error plots for identical networks, with and without batching of weight changes. The network with batched weight changes (solid line) stopped learning at approximately epoch 10.

Weight change batching is discussed in [Ref. 13]. The crux of that discussion is that batching is useful in some situations. But it depends on the nature of the problem. In other situations, input batching can render a network more prone to becoming stuck in local minima. Network 10 in Table 4.3 also illustrates this point. This network correctly identified only 68 of the 157 diphones in its initial training set (just one example of each diphone). Apparently, weight change batching is of questionable value in diphone recognition systems.

## 7. Hidden layer sizes

The majority of the networks trained on the large and medium-sized data sets included 256 neurons in the hidden layer. This produced the lowest error rate of the networks tested without input batching. Figure 4.8 shows test set performance for the hidden layer sizes which performed well enough initially to justify training on the full training set. It may be that a smaller number of hidden neurons would suffice. But there would probably be no advantage to pruning the hidden layer other than reducing the CPU's floating point burden. Research by Sietsma and Dow [Ref. 28] suggests that, contrary to neural network "folklore," neural networks with smaller hidden layers do not generalize better than those with oversized hidden layers. Networks often generalize poorly because of overtraining. But if training is stopped when test set errors reach a minimum, and training is conducted with "noisy" data, there is no accuracy penalty for oversized hidden layers. These conclusions are consistent with informal observations on the issue during this research.
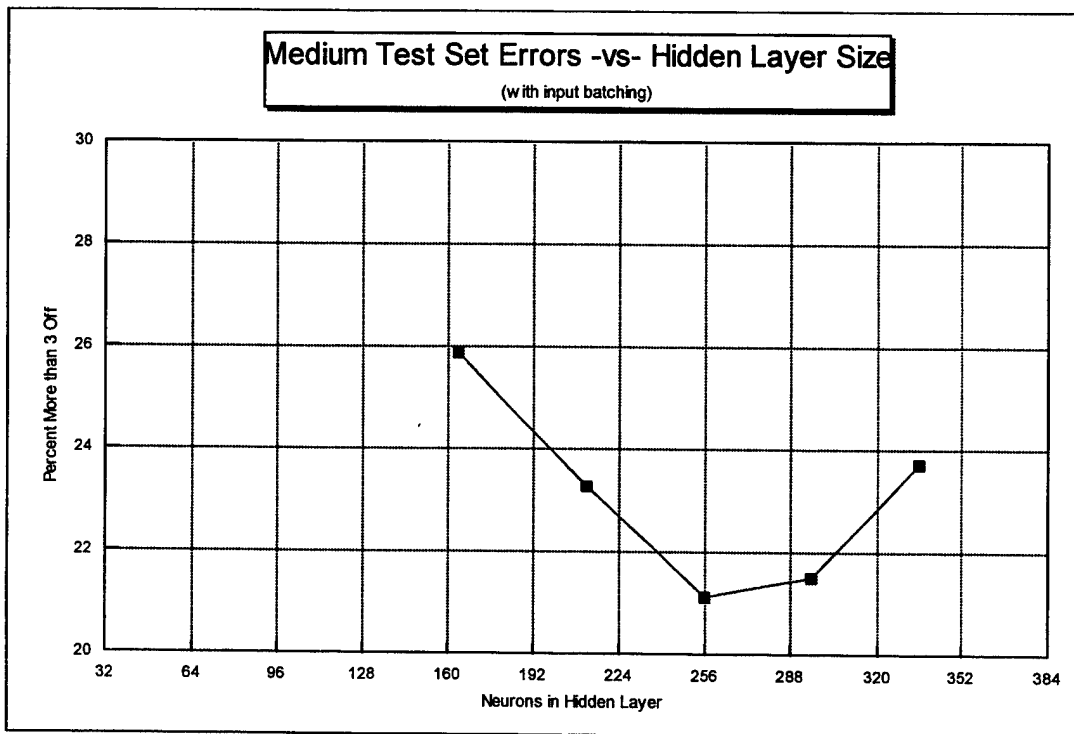


Figure 4.8. Medium test set errors versus hidden layer size (with batched weight changes). Most networks successfully trained on the medium and large data sets employed 256 hidden neurons (the error minimum in the above plot). However, more data points would reveal that the above relationship is not smooth (peaks and valleys exist between the points plotted).

77

## 8. Randomized Sequencing

When inputs are batched, input vectors may be presented in any sequence during a training epoch. Since all weight changes are deferred until the end of the epoch, their ordering is irrelevant. That is not the case when input batching is disabled. Without input batching, weight changes for a given input vector are determined after applying weight changes for the preceding input vectors. So changes in the input sequence can lead to entirely new paths down the error surface. Therefore, randomized sequencing of inputs is often recommended [Ref. 13]. When training on just one example of each of 157 diphones, randomized sequencing did indeed appear to speed up network convergence. However, when training with the medium data set (30 examples of each of 157 diphones), there was no apparent advantage to "shuffling" the input vectors before each epoch. For example, Network 11 in Table 4.3 was trained by shuffling the input vectors before every epoch. The input vectors for Network 12, on the other hand, were shuffled just once, prior to the first epoch. The input sequence was identical for the remaining epochs. The results for these two networks (and similar comparisons) imply there is no significant benefit to frequent re-sequencing of the inputs. Therefore, the added algorithmic complexity and run time appear to be unjustified.

## 9. Feature Vector Length

As discussed in Chapter III, the recognition window was initially 23 (overlapped) sample windows or 502 milliseconds wide. This design worked well enough during initial testing with 19 diphones (dominated by vowels). But the error rate increased considerably during testing with a data set of 44 diphones (and a larger proportion of stop consonants and consonant pairs). The error statistics implied that the shorter diphones were suffering higher error rates than the longer ones. So tests were run with these 44 diphones and recognition window widths of 7, 9, 11, 13, 15, 17, 19, and 23 sample windows. Error rates were consistently lowest when the input vector was 13 sample windows wide (289 milliseconds). Also, comparisons were later run with the medium data set and input vector widths of 11, 13, and 15 sample windows. The results can be seen in table 4.3 (Networks 13, 14, and 15 respectively). Although the difference between 11 and 13 sample windows is less apparent with the larger data set, both are clearly better than the

78

15 sample window version. Therefore, a recognition window width of about 289 milliseconds appears to be a reasonable compromise, at least for the 157 diphones used in this research.

## 10. Training Phases

Some early networks were trained on the entire training set from the first epoch. This method had several disadvantages:

- A trial and error approach had to be used to determine the number of neurons required in the hidden layer. Much CPU time was wasted in training networks for hundreds of epochs on thousands of files, only to discover that the networks needed more hidden neurons.

- Some networks failed to converge, even though later trials demonstrated that the networks had enough hidden nodes.

Both of these problems were addressed by training later networks with a single (arbitrarily selected) example of each diphone initially. After the networks performed well on the *initial set*, training was resumed with the complete training set. With only 157 files in the initial set, epochs accumulated quickly. If the network failed to learn the initial set, more hidden nodes could be added and another attempt made with little time lost. This technique also seemed to ultimately result in more accurate networks. Networks 16 and 17 in Table 4.3 are examples. However, the difference was not dramatic and it was much less apparent for some comparisons than others. Also, training for too long on the initial set actually reduced final network performance. Network 9 in table 4.3, for example, was identical with Network 8 in every respect but one, the length of training on the initial set.

It should be noted that these observations are consistent with the recommendation of many researchers that initial training be conducted without "noise." For example, an optical character recognition system should be trained on sharply defined letters before attempting to train on smudged ones. Although no speaker's diphones can be considered "noise free," the initial set in the preceding discussion takes on a similar role. The individual variations of other speakers can then be viewed as a form of noise.

## 11. Feedback-Based Time Alignment

Figure 4.9 shows the output of the **N-D** output neuron as a neural network's recognition window is stepped through a speech sample containing the **N-D** diphone. Note that the neural network's output peaked sharply when the recognition window was properly aligned and fell off rapidly both before and after that window position. Figure 4.10 shows the same output sequence from a different perspective. The solid line in Figure 4.10 is the same output plotted in Figure 4.9. The dashed lines show the outputs of competing neurons during the same time period. Again, the penalty for recognition window misalignment is obvious. Because of the dynamic nature of diphone phoneme transitions, a shift of just one window position (about 20 milliseconds, 10 with interleaving as described in Chapter III) can completely remove the correct output from the top three outputs.
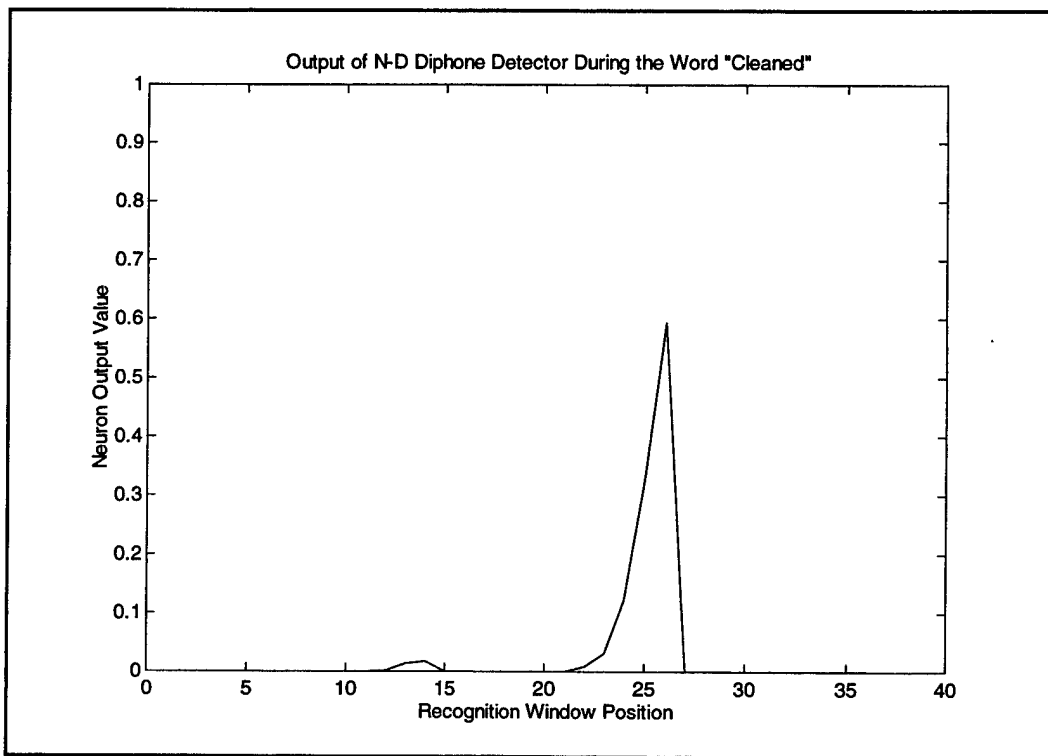


Figure 4.9. Output of the **N-D** neuron as the recognition window is stepped through a speech sample containing the **N-D** diphone. Note that the neural network is sensitive to correct alignment (window positions are about 10 milliseconds apart).

This again raises an issue discussed in Chapter III, recognition window time alignment. During training, the TIMIT phonetic transcriptions provide a rough guide. However, as discussed in Chapter III, these transcriptions are only approximate and they may be based on

80

time-domain features that may be less important to the neural networks than other (frequency domain) features. Furthermore, during recognition the networks must recognize diphones with no a priori knowledge of where in the speech stream to look.
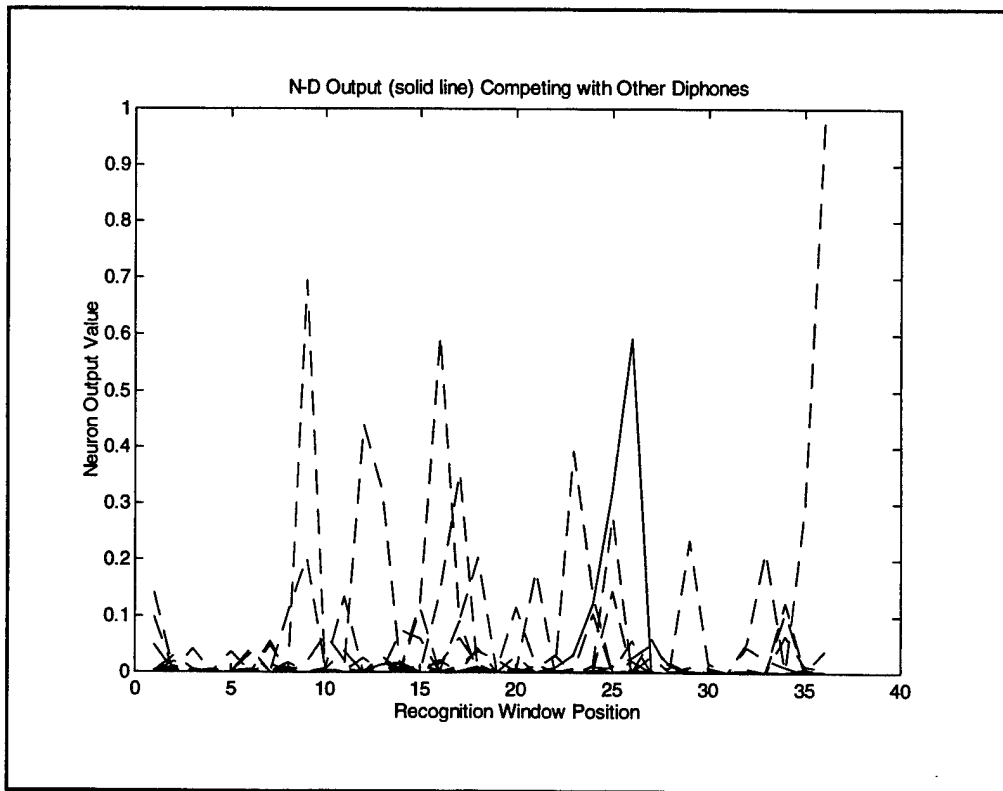


Figure 4.10. The same output as in Figure 4.9, but joined by plots of the other 156 output neurons during the same time period.

The recognition time alignment approach was described in Chapter III. The top three network outputs are recorded for every window position during recognition. The lexical analyzer then sorts out which output peaks to believe. The training problem is actually more challenging. The TIMIT transcriptions are adequate during initial training. But small variations in time alignment apparently blur diphone features as the networks become more discriminating. Thus, recognition window alignment should be adjusted as carefully as possible prior to all weight changes. The following subsections describe three approaches to time alignment and the results obtained from each.

### a. Timit Transcription-Based Alignment

Network 17 in Table 4.3 was trained using only the TIMIT phonetic transcriptions to time align the recognition window. In dozens of comparisons, this method limited the neural networks to accuracy levels lower than those obtained with both of the other methods.

### b. Feedback-Based Alignment Started Early

Several dozen networks were first trained on an initial set (one example of each diphone) and then with feedback-based time alignment on the entire training set. Network 18 in Table 4.3 was trained in this manner. These networks converged to lower error levels than those trained with transcription-based alignment. However, they also exhibited more variability in recognition window offsets (from the transcription-based window position) than expected. Figure 4.11 is a histogram of test set recognition window offsets for a small network (44 diphones, 1320 training files, and 397 test files) trained with this method.
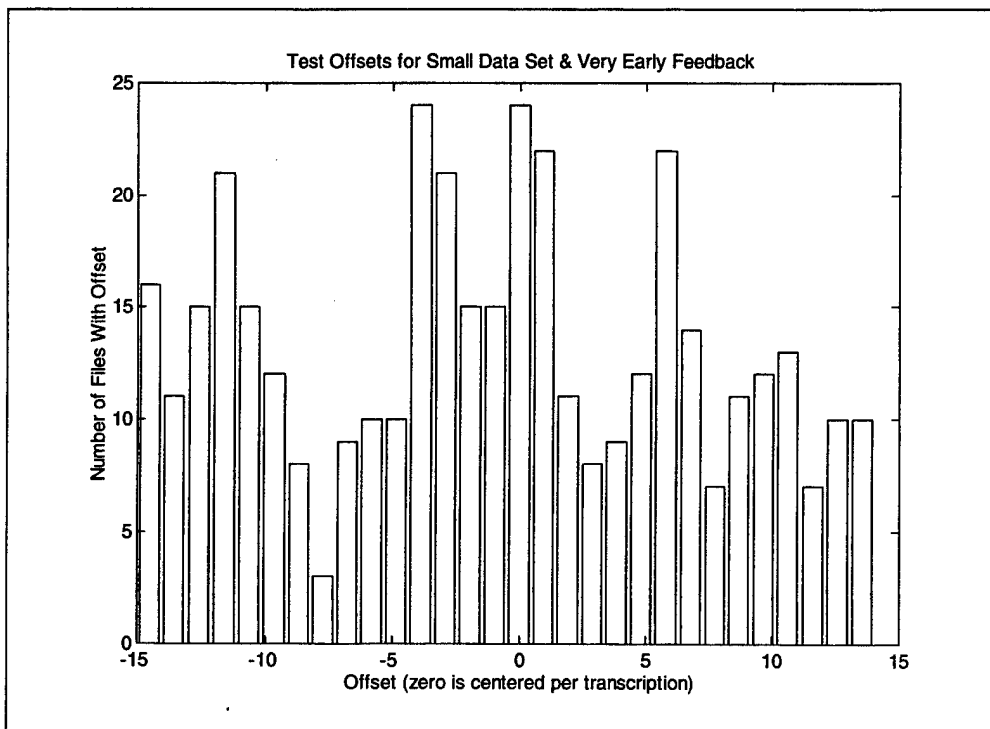


Figure 4.11. Offsets between recognition window center and transcription-based diphone position for a network trained with early feedback-based time alignment.

After training a number of networks using this method, one network proved slow to converge on the initial set. Thus, this network trained substantially longer (several hundred

82

epochs) on one example of each diphone before feedback was used to time-align the entire training set. The resulting weights performed better on the test set and also exhibited less variability in feedback-based recognition window offsets. Apparently, after lengthier training on the initial set the network was better able to sense the "correct" recognition window position.

Figure 4.12 is a histogram of test set recognition window offsets for a network immediately after lengthy training on the medium data set's initial set (one example of each of 157 diphones). Figure 4.13 is a histogram of the same network's test set window offsets after training on the entire training set. These distributions are clearly tighter than in Figure 4.11. Because lengthier initial set training improved error performance, it is reasonable to expect even better performance from networks trained on the entire training set before relying on network feedback for recognition window alignment. The results of this approach are described in the next subsection.
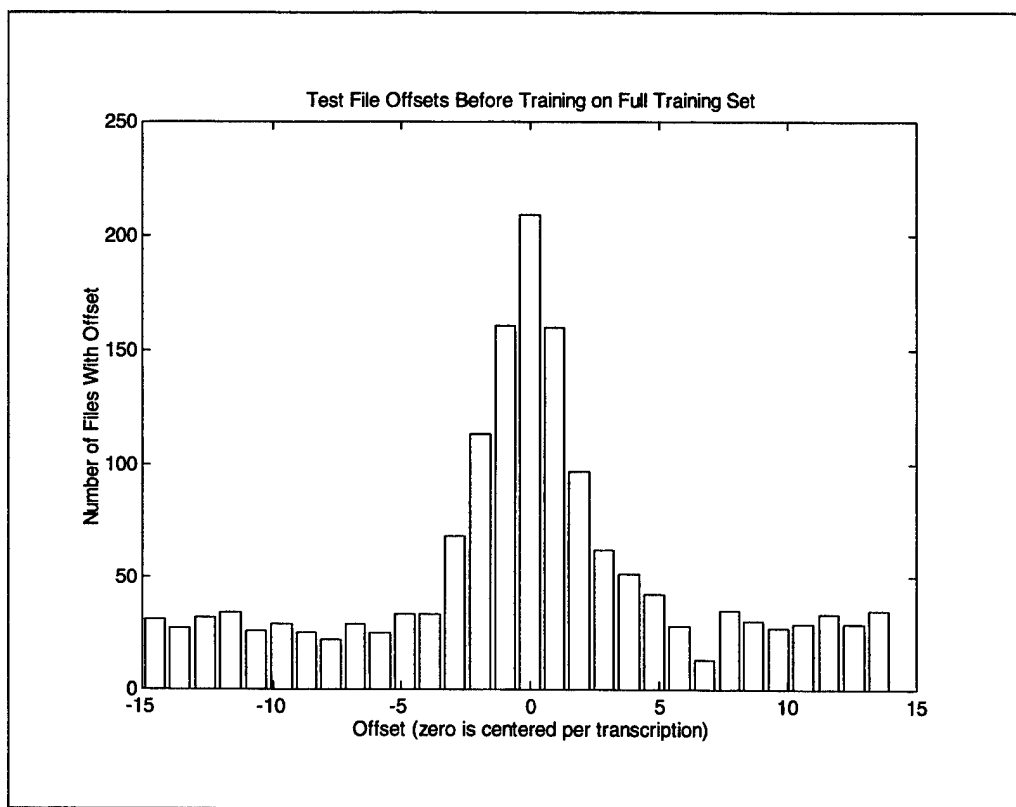


Figure 4.12. Medium test set recognition window offsets for a network immediately after lengthy training on the medium data set's initial set (one example of each of 157 diphones)

83
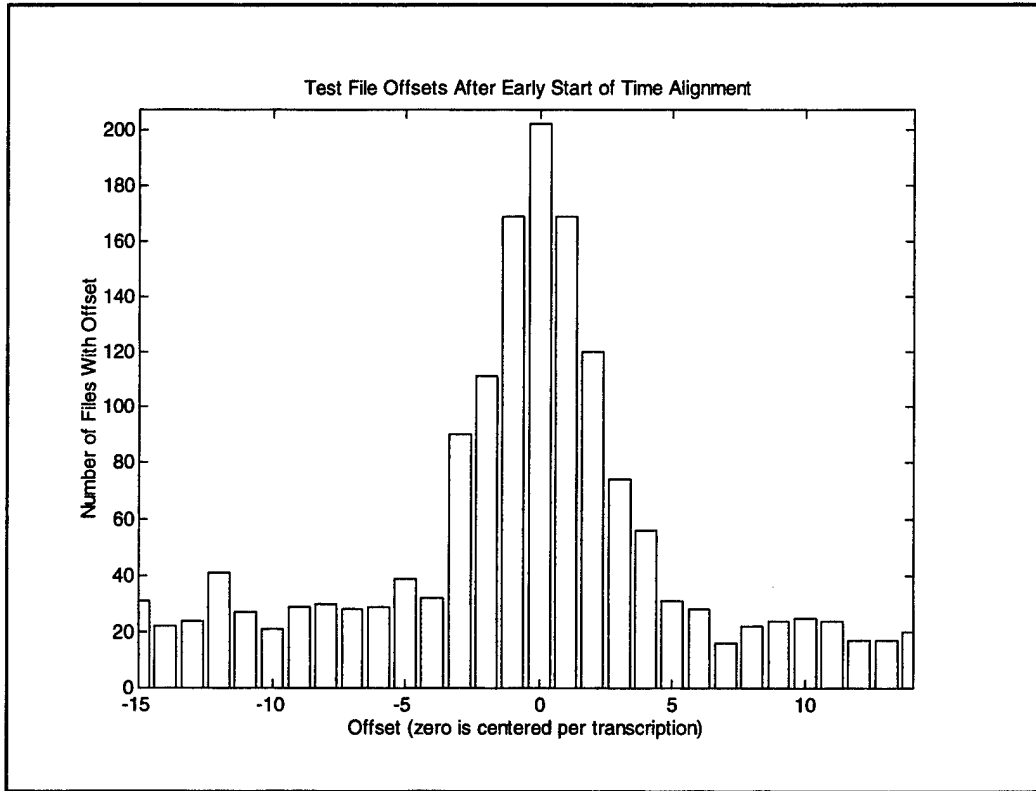
**Figure 4.13.** Test set recognition window offsets (for the same network as in Figure 4.12) after training with feedback-based alignment on the full training set (and lengthier initial set training than in Figure 4.11). Note that the neural network usually preferred recognition window positions clustered around the position indicated in the phonetic transcription.

### c. Feedback-Based Alignment After Training With Transcription

Networks 11, 12, and 19 in Table 4.3 were trained on the medium data set's initial set, followed by transcription-based time alignment on the entire training set. When test set errors appeared to have reached a minimum, training was completed with feedback-based alignment and the entire training set. As a precaution, recognition window offsets were also limited to plus or minus two window positions from the transcription-based position. This method resulted in the lowest error rate of all three methods used. Figures 4.14 and 4.15 also show that these networks exhibit less variability in window offsets, both before and after the use of feedback-based alignment.
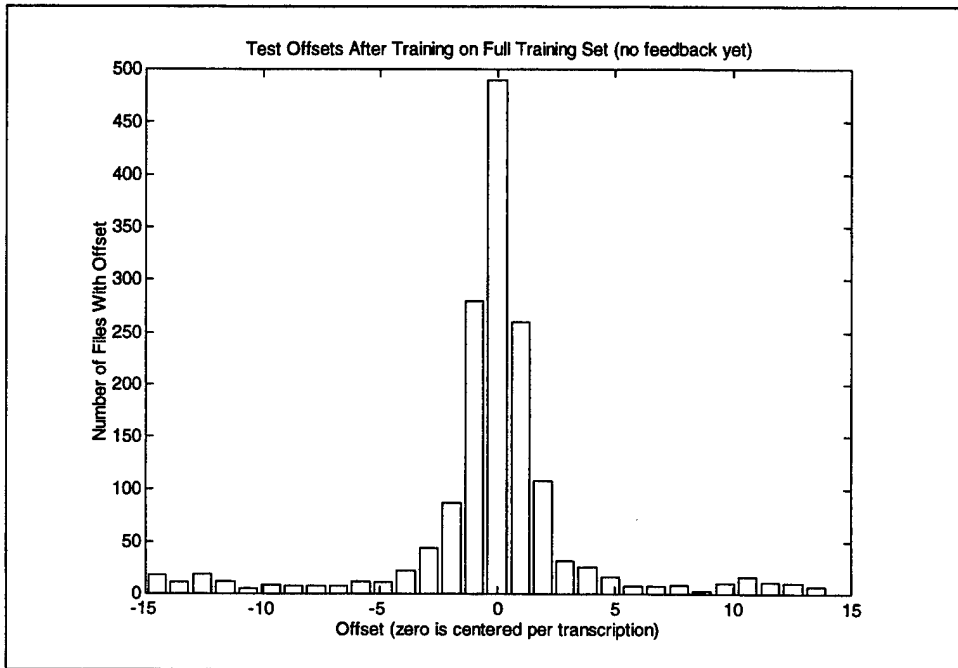
Figure 4.14. Test set recognition window offsets after lengthy training on the full training set (but before training with feedback-based alignment). Note that the additional training has apparently improved the network's ability to determine when the diphone is properly aligned with the recognition window.
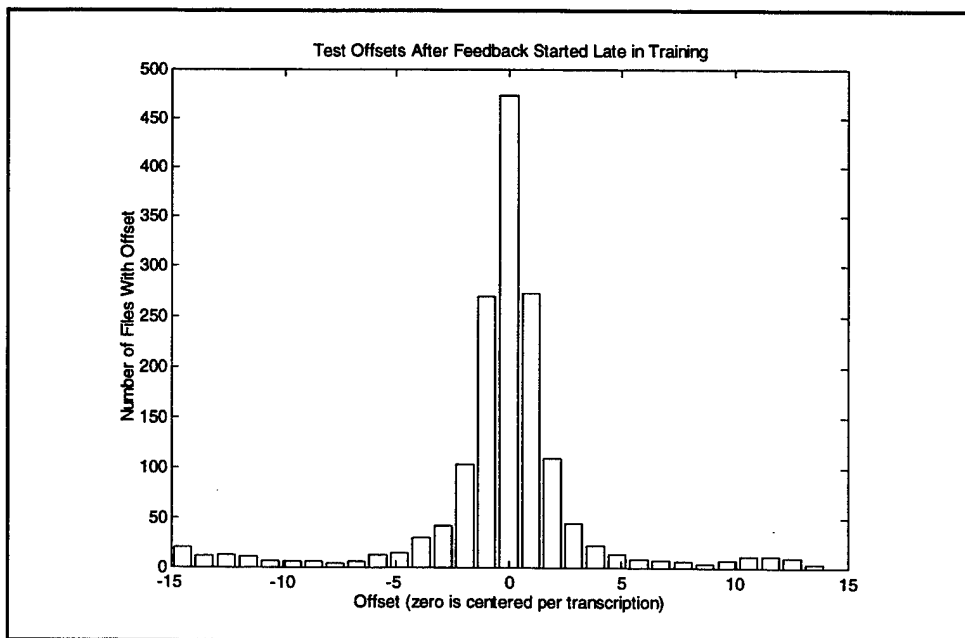


Figure 4.15. Test set offsets after training with feedback-based alignment. Note that the network still favors positions near the transcription position, even after being given the freedom to deviate.

Figure 4.16 is a plot of test set errors for Network 1. Note the sharp drop in errors when feedback-based alignment began (epoch 196). Feedback-based offsets are recomputed every five epochs. So one would expect the quality of the network's feedback to improve as training progresses. Although it is difficult to determine all of the mechanisms involved, the error rate does continue to improve for some time after feedback-based time alignment commences.
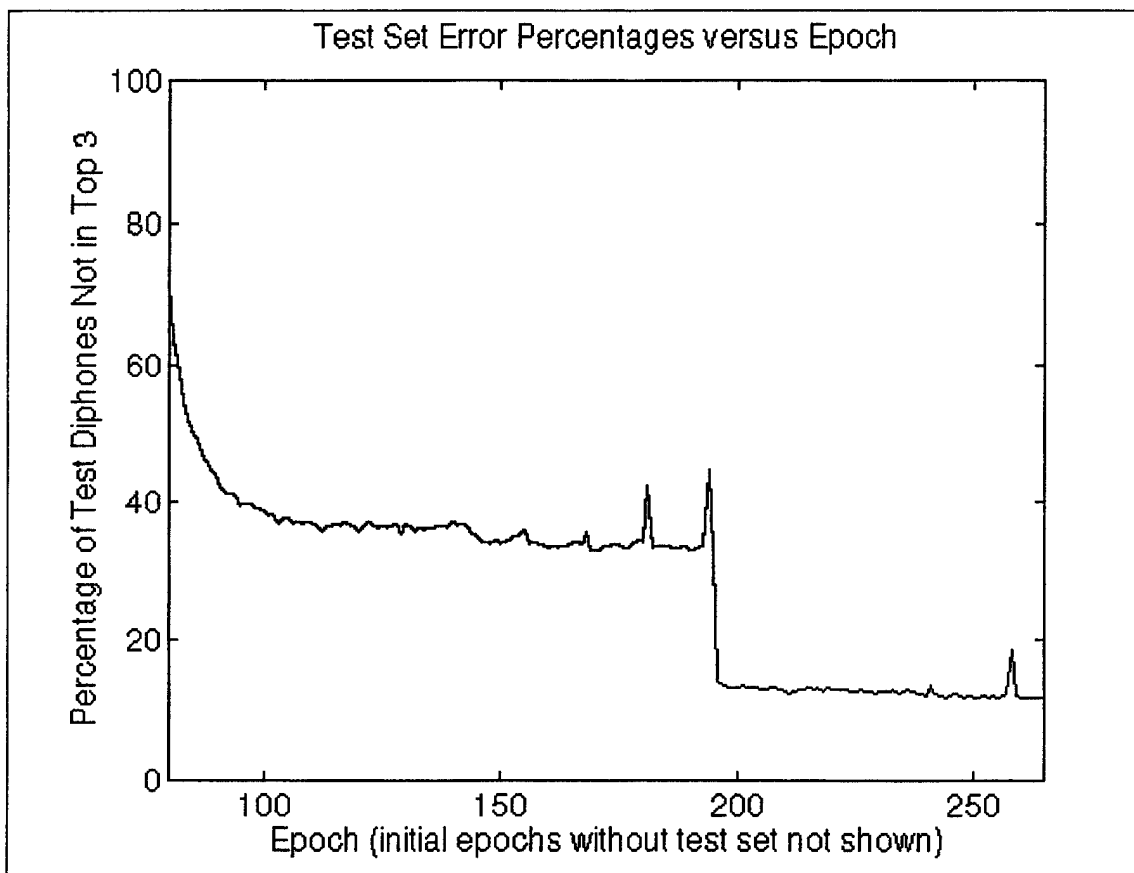


Figure 4.16. Test set error percentage for Network 1 in Table 4.3. Note the sharp drop in errors when feedback-based alignment began at epoch 196.

## 12. Short-Time Power Versus RMS Amplitude

As discussed in Chapter III, the input feature surface was initially designed with one column devoted to short time power. Figure 4.17 shows plots of short time power for 5 examples of the **T-IY** diphone. Note that one of the plots is an outlier. Apparently, that speaker was louder than most of the other speakers. Unfortunately, the squaring operation used to arrive at short-time power accentuated the differences between the outlier's waveform and the other

signals. Figure 4.18 shows the same signals converted to RMS amplitude. Note that the outlier is much less prominent in the second figure.
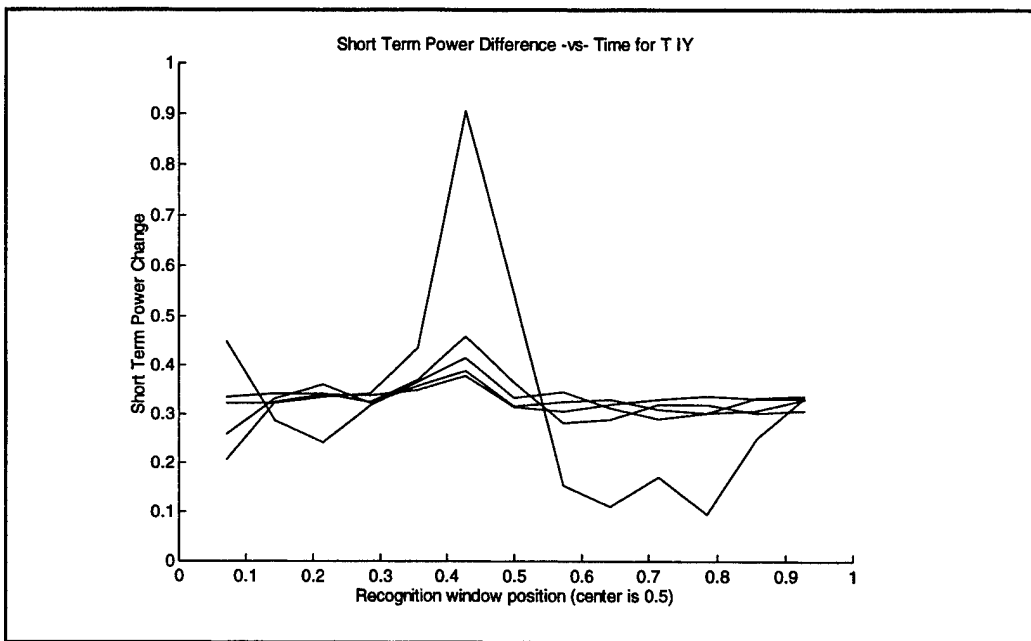


Figure 4.17. Short time power for 5 examples of the **T-IY** diphone. Note that one of the plots is an outlier. The outlier's differences are accentuated by the squaring operation.
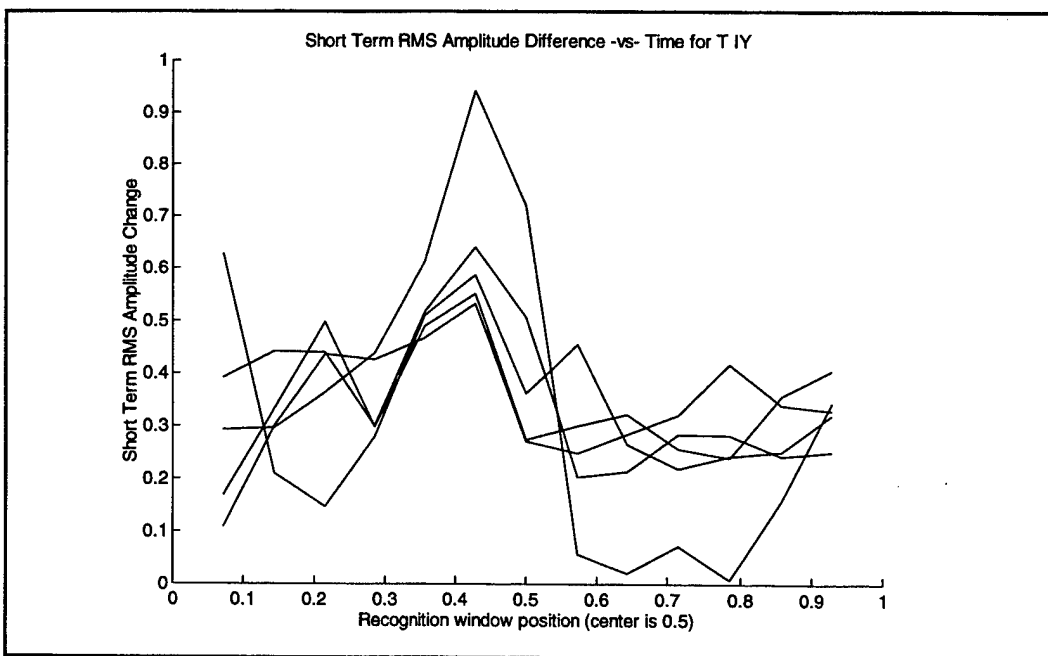


Figure 4.18. RMS amplitude for the same 5 diphone samples as in Figure 4.17. Note that the outlier's differences are now less extreme.

Judging from several comparisons, including Networks 1 and 19 in Table 4.3, RMS amplitude is indeed a better feature to use in diphone recognition than short-time power. This is consistent with observations by Hamming [Ref. 29] that $L_1$ measures are often more useful in perceptual matters than $L_2$ measures.

## D. WORD RECOGNITION RESULTS

Figure 4.19 shows a plot of all 157 outputs from Network 1 (Table 4.3) when presented with a speech sample containing the word "cleaned." Some of the output spikes are, of course, from diphone recognizors properly triggered by this word. The other spikes are noise from false neuron activations. Figure 4.20 shows a contour plot of only the relevant diphone detector outputs. Note that it is possible to visually construct the correct word from the peaks in this figure.
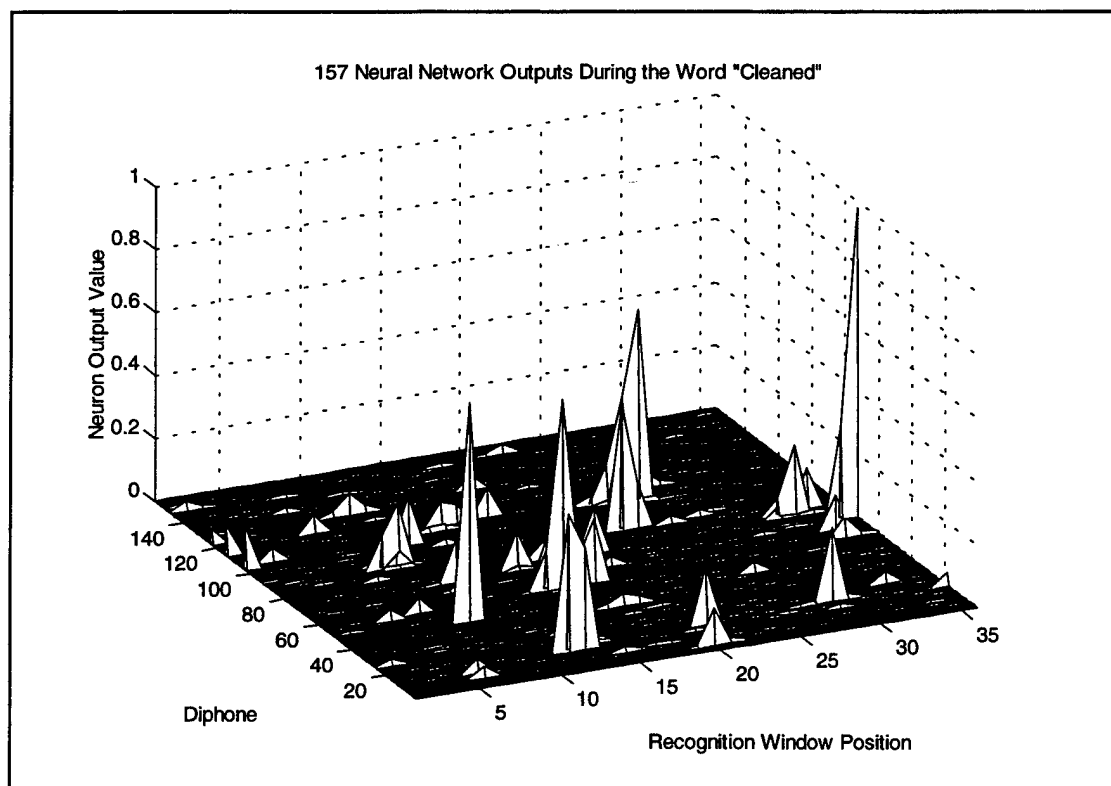


Figure 4.19. A plot of all 157 outputs from Network 1 (Table 4.3) when presented with a speech sample containing the word "cleaned."

88
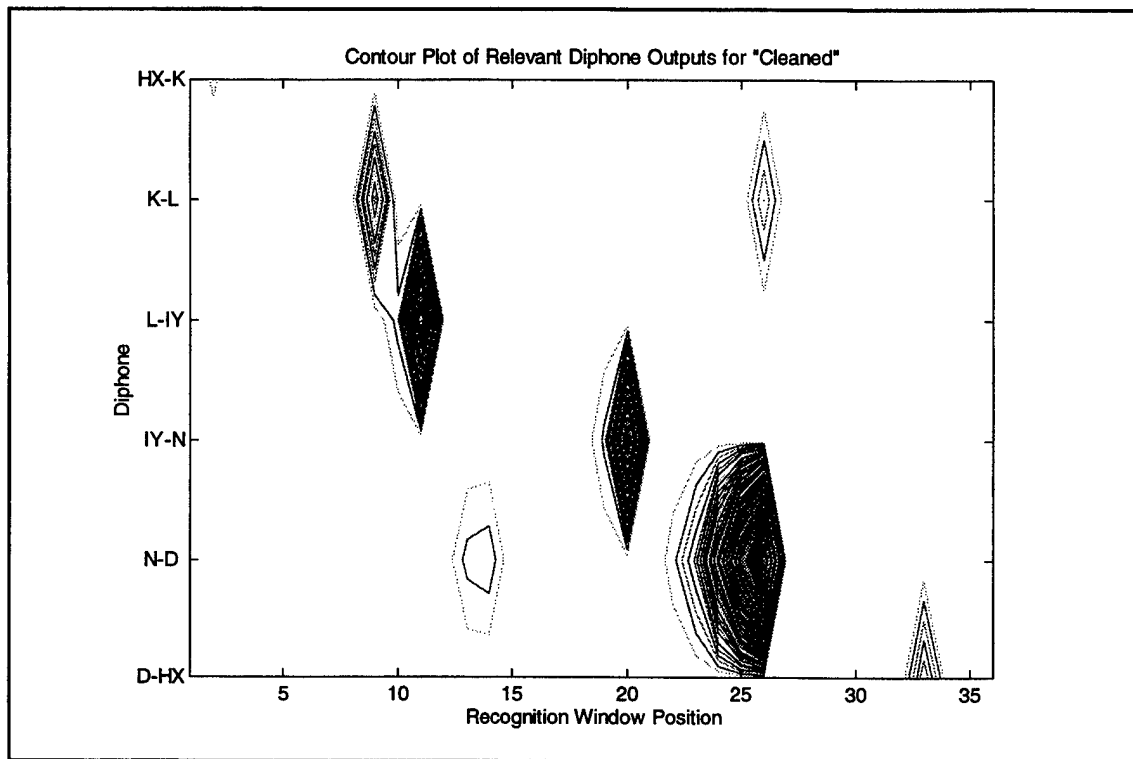
Figure 4.20. A contour plot of only the relevant diphone detector outputs from Figure 4.19. A diagonal from the upper left to lower right corners roughly connects the output peaks for the word "cleaned."

Figure 4.21 shows the corresponding text output from the diphone detector (RECOGNIZ.M in Appendix A). As discussed in Chapter III, only the three strongest outputs are passed to the LISP lexical analyzer.

Finally, Figure 4.22 shows a portion of the output from the LISP lexical analyzer. Notice that the correct word was detected in the diphone stream. The syntactic and semantic analyzers would be expected, in a continuous speech recognition system, to select from the candidate words nominated by the lexical analyzer.

Using the 157 diphones selected for the medium and large data sets, 3070 of the 110,935 entries in the Carnegie Mellon lexicon (version 0.3 dated 9-7-94) can be constructed. The same diphones allow a vocabulary of 340 out of the 6224 entries in the TIMIT lexicon. Of these, 1977 samples of 120 words could be found in the SX test set sentences. Unfortunately, these 120 words did not include uniform coverage of the 157 diphones in the Medium and Large data sets. For example, the word "the" alone accounted for 409 of the 1171 in-vocabulary speech samples. Thus, the word recognition statistics presented in Table 4.5 do not necessarily indicate accuracy levels achievable on more uniform samples.

```
(word_is cleaned)
((171 (EH S   ) 0.317149) (171 (IH NG  ) 0.195618) (171 (R AA   ) 0.116300))
...
((4788 (F R   ) 0.695278) (4788 (K R   ) 0.202686) (4788 (K L   ) 0.054996))
((4959 (R IY  ) 0.067056) (4959 (L IY  ) 0.014768) (4959 (K R   ) 0.007128))
((5130 (L IY  ) 0.153800) (5130 (IX Z  ) 0.134072) (5130 (IX N  ) 0.037814))
((5301 (AX N  ) 0.438422) (5301 (N IY  ) 0.025831) (5301 (DH IY ) 0.003564))
((5472 (AX N  ) 0.324274) (5472 (D IH  ) 0.029013) (5472 (L EH  ) 0.018131))
((5643 (N IH  ) 0.073388) (5643 (AXR Z ) 0.018055) (5643 (N D   ) 0.016444))
((5814 (IH NG ) 0.126056) (5814 (IY IX ) 0.114169) (5814 (N IH  ) 0.059837))
((5985 (IH NG ) 0.598576) (5985 (IX M  ) 0.141212) (5985 (IH M  ) 0.022087))
((6156 (IX M  ) 0.354304) (6156 (IH NG ) 0.092821) (6156 (N IX  ) 0.091743))
((6327 (IH NG ) 0.203180) (6327 (S AX  ) 0.043799) (6327 (EY N  ) 0.038053))
((6498 (EY N  ) 0.021970) (6498 (IY N  ) 0.013394) (6498 (IX T  ) 0.006094))
((6669 (IY N  ) 0.146805) (6669 (AE M  ) 0.114745) (6669 (IX N  ) 0.029980))
((6840 (AX V  ) 0.180183) (6840 (AE M  ) 0.013460) (6840 (R IH  ) 0.011701))
((7011 (W IH  ) 0.031189) (7011 (N D   ) 0.009318) (7011 (M P   ) 0.002605))
((7182 (K R   ) 0.391690) (7182 (N D   ) 0.031431) (7182 (R EY  ) 0.007329))
((7353 (K R   ) 0.137409) (7353 (N D   ) 0.122368) (7353 (N DH  ) 0.104988))
((7524 (N D   ) 0.323681) (7524 (M P   ) 0.277733) (7524 (N T   ) 0.145333))
((7695 (N D   ) 0.594521) (7695 (M EY  ) 0.056732) (7695 (N S   ) 0.026563))
((7866 (R AE  ) 0.060286) (7866 (N S   ) 0.049713) (7866 (HX D  ) 0.029375))
((8037 (K S   ) 0.017934) (8037 (N S   ) 0.012144) (8037 (AXR Z ) 0.007918))
((8208 (AXR Z ) 0.233987) (8208 (AX N  ) 0.009879) (8208 (K W   ) 0.008760))
((8379 (AXR Z ) 0.016868) (8379 (IX T  ) 0.009465) (8379 (D IX  ) 0.003793))
((8550 (IY IX ) 0.001285) (8550 (R AA  ) 0.000765) (8550 (W IY  ) 0.000573))
((8721 (JH IX ) 0.051646) (8721 (IY S  ) 0.044583) (8721 (T HX  ) 0.006520))
((8892 (JH IX ) 0.215402) (8892 (D HX  ) 0.034956) (8892 (IY S  ) 0.024411))
((9063 (K AE  ) 0.123909) (9063 (IX N  ) 0.106962) (9063 (IH Z  ) 0.065737))
((9234 (IX Z  ) 0.290279) (9234 (IH Z  ) 0.011543) (9234 (Z AX  ) 0.006819))
((9405 (IX Z  ) 0.978204) (9405 (AX V  ) 0.038427) (9405 (IH Z  ) 0.005301))
((9576 (SH IX ) 0.236237) (9576 (IX V  ) 0.058200) (9576 (AX V  ) 0.020296))
((9747 (S T   ) 0.109034) (9747 (K T   ) 0.056291) (9747 (SH IX ) 0.034269))
((9918 (AX L  ) 0.075804) (9918 (IY N  ) 0.008476) (9918 (Z IX  ) 0.006940))
((10089 (D HX  ) 0.118372) (10089 (T EH  ) 0.112912) (10089 (S HX  ) 0.030426))
...
((10602 (Z HX  ) 0.330007) (10602 (S AH  ) 0.132011) (10602 (D R   ) 0.089018))
(word_end cleaned)
```

Figure 4.21. Output from all 157 diphone detectors for the speech sample plotted in Figures 4.19 and 4.20. Only the three strongest outputs for each time step are included in the output (and passed to the LISP lexical analyzer), with one output line per time step. The diphone outputs used by the lexical analyzer to build the word "cleaned" are in bold type. Each diphone output consists of the sample number where the diphone was detected, the diphone name, and the numerical output of the neuron (used as a measure of confidence by the lexical analyzer). The "word_is" and "word_end" labels at the beginning and end of the stream are used to capture error statistics, not for recognition.

90

```
-----------------------------[ WORD RECOGNIZOR OUTPUT ]--------------------
(word_is cleaned)
((171 7182) ROCK 0.253995)

...

((4617 9063) WICK 0.057184998)
((4617 9063) QUICK 0.057184998)
((4788 7182) RAY 0.3513035)
((4788 10602) PHRASE 0.34420466)
((4788 7524) RIM 0.32823732)
((4788 5985) CLING 0.326786)
((4788 8892) RIDGE 0.30746034)
((4788 10089) READ~V_PRES 0.29356867)
((4788 10602) RACE 0.278206)
((4788 9747) WRIST 0.27200434)
((4788 10089) RATE 0.27183965)
((4788 7695) CLEAN 0.26777232)
((4788 10431) REAL 0.25929865)
((4788 9747) RAKED 0.252966)
((4788 10431) RAIL 0.23938966)
((4788 10089) CLEANED 0.23042224)          <- Correct word ranked 14th among words with same start time
((4788 7695) SCREEN 0.21856575)
((4788 10089) AFRAID 0.20774475)
((4788 10602) CLEANS 0.171402)
((4788 10602) LAMB~S 0.16658266)
((4788 10602) CLAMS 0.16658266)
((4788 7524) RIM 0.16404)
((4788 7524) LAMP 0.149158)
((4788 7524) CLAMP 0.149158)
((4788 8208) CLEANERS 0.14739701)

...

((4959 6669) THREE 0.07462034)
((4959 6669) TREE 0.07462034)
((5130 10602) EASE 0.24190348)
((5130 7695) CLEANED 0.2262815)            <- Same instance of word, but built starting at 2nd diphone
((5130 7182) KNICK- 0.16763067)
((5130 10602) CLEANS 0.160153)
((5130 8208) CLEANERS 0.136148)
((5130 10089) LEAD 0.136086)

...

-----------------------------[ DEBUGGING OUTPUT ]--------------------
(CLEANED ((4788 (K L) 0.054996)           <- Correct word built starting from K-L diphone
          (5130 (L IY) 0.1538)
          (7695 (N D) 0.594521)
          (10089 (D HX) 0.118372))
         0.23042224)
(CLEANED ((-1 (K L) 0.01)                 <- Dummy diphone inserted by lexical analyzer
          (5130 (L IY) 0.1538)            <- This word instance constructed by starting search from L-IY
          (6669 (IY N) 0.146805)
          (7695 (N D) 0.594521))
         0.2262815)
```

Figure 4.22. Output stream from the lexical analyzer for the speech sample of Figures 4.19 through 4.21. Each word candidate is listed with its start sample number and overall confidence value (average of its diphone confidence values. A more detailed output format (for debugging) is listed at the bottom. Note the large number of rhyming words and word fragments detected. Note also that the analyzer was able to detect the correct word even without the first diphone.

| DESCRIPTION | OF THOSE DETECTED | OVERALL |
|---|---|---|
| Percentage of time correct word was in lexical analyzer's output stream | - | 85.0% |
| Correct word ranked first of word candidates with same start time | 41.6% | 35.4% |
| Correct word ranked among top 3 word candidates with same start time | 60.0% | 51.0% |
| Correct word ranked among top 6 word candidates with same start time | 74.0% | 62.9% |

Table 4.5. Summary of Word Recognition Statistics Using Network 1 from Table 4.3

The neural network and lexical analyzer were deliberately presented with speech segments that started before and ended after the word being tested. Thus, the neural network and lexical analyzer achieved the results of Table 4.5 on what amounts to continuous speech. Also, the word recognition tests were conducted before (by about two percent) Network 1 reached the error minimum listed in Table 4.3. So slightly higher scores might be achievable, even with the limitations of the word test set.

As mentioned in the table's description column, the accuracy rates in Table 4.5 are for all word candidates with starting sample numbers that match the correct word's starting sample number. In many cases, the correct word was both globally and locally the strongest candidate nominated by the lexical analyzer. In other cases, the correct word had a high ranking locally (among word candidates with the same start time) but other words ranked higher globally. In the latter case, it would be up to the syntactic and semantic analyzers to properly segment the sentence and select the appropriate word candidates. On the average, the 85 percent of the correct words that were detected were ranked number 5.44 among words with the same starting time.

Appendix G contains a more detailed list of the word recognition results summarized in Table 4.5. It is interesting to look at both the successes and failures of the lexical analyzer. Although no detailed analysis has yet been made of the errors, some samples have been listed in Table 4.6, along with comments as appropriate.

Further details on the lexical analysis algorithm are available in the source code comments in Appendix C. Finally, Chapter VI includes recommendations on how the word recognition rates of Table 4.5 might be improved.

| CORRECT WORD | LOCAL RANK | GLOBAL RANK | LOCAL WINNER | GLOBAL WINNER | REMARKS |
|---|---|---|---|---|---|
| fish | 1 | 1 | fish | fish | first 4 are local and global winners |
| of | 1 | 1 | of | of | |
| by | 1 | 1 | buy | buy | no attempt to resolve homonyms |
| may | 1 | 1 | may | may | |
| frantically | 1 | 20 | frantically | leaf | local best but not global |
| in | 1 | 7 | in | the | "the" is next word in sentence |
| of | 1 | 384 | of | am | |
| run | 1 | 230 | run | may | |
| leap | 1 | 7 | leap | knee | |
| candy | 1 | 6 | candy | tea | |
| quick | 1 | 16 | quick | wick | "wick" portion of word stronger |
| made | 9 | 28 | may | may | "may" part stronger |
| my | 3 | 3 | mind | mind | "my" part real, rest is fiction |
| money | 4 | 18 | my | knee | "knee" part stronger |
| are | 7 | 482 | art | in | next word starts with a stop |
| intelligence | 7 | 274 | intent | ten | more confident in first part |
| in | 8 | 20 | dim | dim | nasals are hard to tell apart |
| become | N/A | N/A | BE | BE | missed last part of word |
| hard | N/A | N/A | ARE | ARE | missed first part of word |

Table 4.6. Sample Word Recognition Results

## E.  ANALYSIS OF REMAINING ERRORS

As with word recognition errors, no detailed analysis has yet been conducted of remaining diphone recognition errors. However, Figure 4.23 shows that these errors are at least fairly uniformly distributed among diphones. Again, Chapter VI offers recommendations on how the remaining errors might be reduced.

## F.  SUMMARY

The results reported in this chapter are encouraging. The next chapter suggests what these results imply with respect to the basic research questions of this thesis.
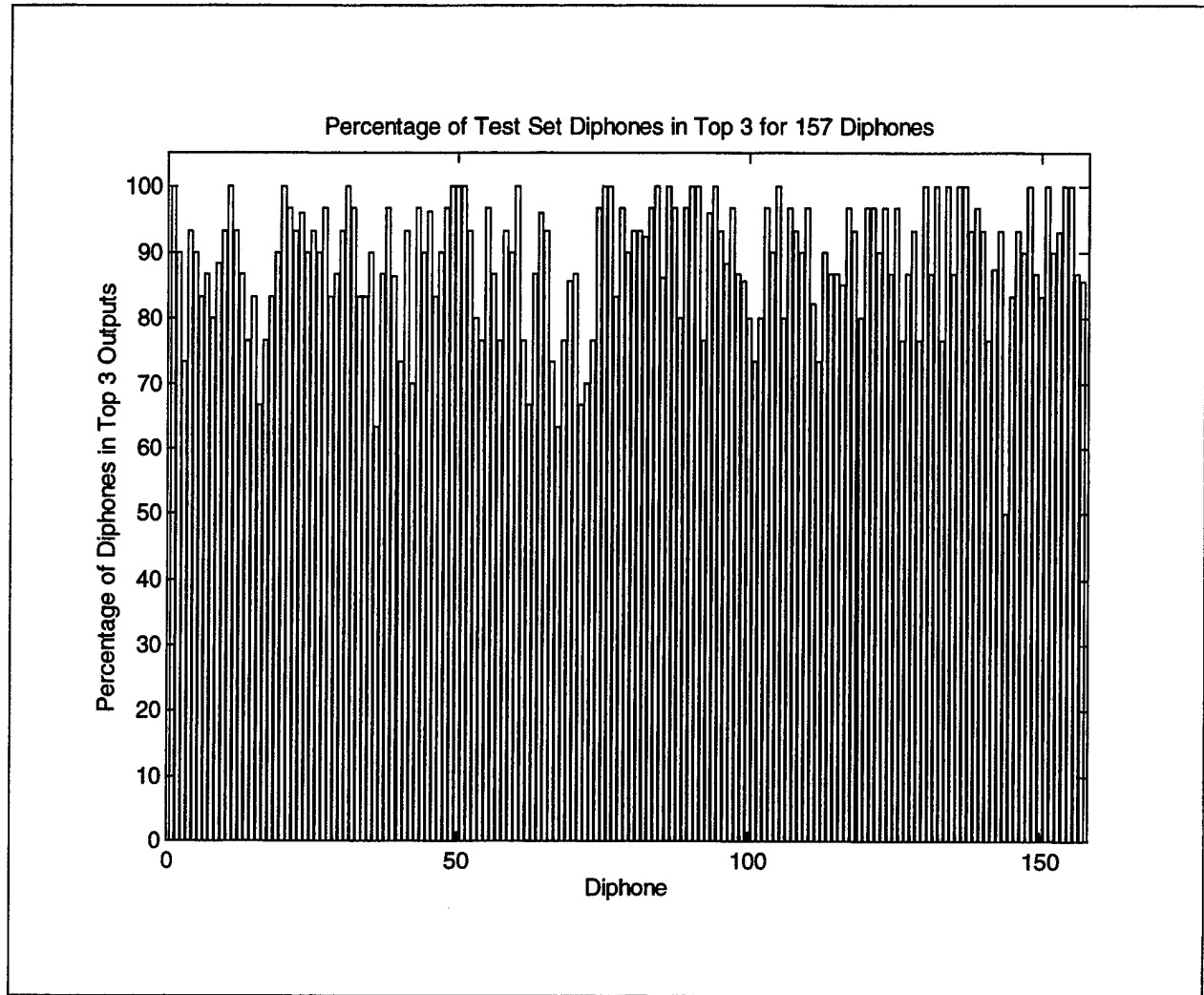
Figure 4.23. Accuracy rates (top 3 metric) for 157 diphones of Network 1 in Table 4.3.

# V. CONCLUSIONS

## A. INTRODUCTION

This chapter briefly summarizes major conclusions that may be drawn from the results of Chapter IV. In particular, the research questions listed in Chapter I are addressed.

## B. THE DIPHONE AS A RECOGNITION UNIT

The results reported in Chapter IV imply that the diphone is indeed a suitable recognition unit for speaker-independent continuous-speech recognition systems. As further evidence of their potential, Table 5.1 compares the performance of two diphone recognizors (from Tables 4.3 and 4.4) with two recent neural network-based phoneme recognition systems.

| NETWORK | RECOGNITION UNIT | NUMBER OF OUTPUT CLASSES | TRAINING SET CORRECT | TRAINING TOP 3 | TEST SET CORRECT | TEST SET TOP 3 |
|---------|------------------|--------------------------|----------------------|----------------|------------------|----------------|
| [Ref. 23] | PHONEME | 46 | - | - | 27.2% | 55.6% |
| [Ref. 30] | PHONEME | 47 | 58.5% | 68%* | 52.3% | 74%* |
| Table 4.4 | DIPHONE | 44 | 100% | 100% | 71.5% | 95.7% |
| Table 4.3 | DIPHONE | 157 | 98.2% | 98.4% | 70.2% | 89.0% |
| * Estimated from Figure 2 in the reference | | | | | | |

Table 5.1. A Comparison of Recognition Rates for Several Phoneme and Diphone Recognizors.

The statistics in Table 5.1 suggest that the diphone is inherently easier to recognize from its acoustics properties than is the phoneme. Note particularly the contrast between the first three networks in the table. With nearly the same number of output classes, the diphone recognizor's accuracy rates are substantially higher in every category.

## C. FREQUENCY NORMALIZATION

Although more study is needed, this research suggests that pitch can serve as a basis for improving speaker independence through frequency normalization. Table 5.2 again

shows the apparent impact of frequency normalization on several networks (from Table 4.3).

| ID # | Data Set* | Test Correct | Test Top 3 | Train Correct | Train Top 3 | Frequency Normalized | Time Alignment | RMS Amplitude | Window Width | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | M30 | 54.5% | 73.5% | 87.2% | 87.3% | NO | NO | NO | 13 | Compare w/4 |
| 4 | M30 | 56.8% | 80.3% | 94.3% | 94.4% | YES | NO | NO | 13 | Compare w/3 |
| 5 | M30 | 52.0% | 72.0% | 84.7% | 85.1% | NO | NO | NO | 13 | Compare w/8 |
| 8 | M30 | 58.0% | 79.0% | 91.1% | 91.1% | YES | NO | NO | 13 | Compare w/5 |

\*     The medium data set (30 examples of each diphone in training set) is indicated by M30. The large data set (100 examples of each diphone in training set) is labeled L100.

Table 5.2. Comparison of Neural Networks With and Without Frequency Normalization

## D.  TIME ALIGNMENT

Proper alignment of diphone feature vectors with the recognition window has proven to be absolutely critical. Because of the dynamic nature of diphones, an alignment error of just one recognition window position can drop the correct output from first place to the noise level. However, this research shows that neural network feedback can be used to improve alignment prior to weight adjustments. Specifically, the best results were obtained by training in three phases:

- Train on a single sample of each diphone until the average sum-squared error (total SSE divided by the number of input vectors) is on the order of $10^{-3}$.

- Train with the full training set and phonetic transcription-based alignment until test set errors reach a minimum.

- Resume training and use neural network feedback to position the training vectors in the recognition window. To avoid unreasonable window positions, the maximum offset from the transcription position should be constrained (plus or minus about 40 milliseconds seemed adequate).

During recognition, the sharp temporal selectivity of the trained network can actually be considered an advantage. This issue is discussed further in the next chapter.

96

## E.  TRANSLATION OF DIPHONES TO WORDS

This research implies that continuous-speech word recognition can be achieved using diphones as the recognition unit. Discrete utterances were not assumed in the algorithm described in Chapters III and IV. Yet that algorithm usually spotted the correct word in the neural network's output stream. Unfortunately, it also spotted words that were not present in the speech sample. This does not mean that the basic approach is unworkable. But it does mean that the syntactical and semantic analyzers will need to share some of the word recognition burden. Specifically, they will need to use syntactic and semantic knowledge to eliminate non-sensical words from the lexical analyzer's output stream. This may not be as difficult or unreasonable as it seems. After all, humans routinely use syntactic and semantic knowledge to compensate for poor acoustic signals. One factor that would help is the temporal information provided by the lexical analyzer. Once a word is recognized with some confidence, all word candidates overlapping that word in time can be eliminated from further consideration (unless, of course the system is forced to later reject that word and backtrack). The process can also be facilitated by the use of word segmentation algorithms such as those currently used by discrete speech recognition systems. This issue is discussed further in the next chapter.

## F.  INPUT VECTOR DESIGN

No attempt was made to experiment with input vector alternatives other than length and frequency normalization. Since the input vector design used in this research was based largely on intuition, some aspects of that design may not be optimal. However, the results imply that it has some merit. In particular, the design probably benefited from the inclusion of RMS amplitude (rather than short-time power), frequency normalization of differenced cepstral coefficients, the use of critical bands, and a length of about 289 milliseconds.

## G. NEURAL NETWORK TRAINING METHODS

It would be impossible to conduct research of this nature without drawing conclusions about which neural network training techniques work and which do not. The following comments summarize the key lessons learned:

- Use a single hidden layer unless there are very compelling reasons to use more.

- As a rough rule of thumb, the hidden layer can be started with about twice as many neurons as there are output classes. Neurons can then be pruned or added as needed. Of course, this suggestion is very specific to diphone recognition using the feature vector described in Chapter III.

- In any neural network classifications system, the feature vector should be carefully designed to eliminate frivolous and accentuate relevant features. This may require extensive preprocessing.

- Training should be conducted without noise first [Ref. 9].

- The larger the number of training exemplars, the better. The sequence of presentation should be randomized (although a single "shuffling" may suffice if the training set is large).

- Training exemplars should be evenly divided between output classes.

- The random seed should be set explicitly prior to weight initialization. This makes results repeatable and it allows for deliberate restarts from different points on the error surface.

- Weight changes should not be batched (at least not when training with diphones).

- Momentum and an adaptive learning rate should be used [Ref. 9].

- Use a separate test set (on every epoch) and stop training when test set errors reach a minimum.

## H.  SUMMARY

While the conclusions offered in this chapter are encouraging, many questions remain.  The next chapter will suggest avenues of research that may address some of those questions.

# VI. RECOMMENDATIONS FOR FURTHER RESEARCH

## A. INTRODUCTION

Previous chapters included remarks on a few of the areas where further diphone-related research is needed. For example, further study is needed to determine if the frequency normalization method introduced in Chapter IV can be improved on (perhaps by gathering data on other vowels). Also, both diphone and word recognition errors that remain should be studied in depth. If any trends are identified, they may inspire improvements to the neural network's feature vector design.

Two additional research areas seem promising enough to justify particular emphasis in this chapter. They are summarized in the next two sections.

## B. MODULAR NEURAL NETWORKS

As illustrated in Table 4.4, extremely high recognition rates can be achieved by neural networks trained to recognize up to about 50 diphones. Yet large vocabularies require over 1000 diphones. The accuracy levels achievable on 157 diphones are encouraging. However, *modular neural networks* may scale up better than the monolithic neural networks used in this research. Modular networks are discussed in depth in [Ref. 13]. The basic idea is to break the problem into smaller subproblems, each assigned to its own subnetwork. A top-level network then selects a winner from the outputs of the subnetworks.

Average duration might be a useful basis for partitioning the diphones. The recognition window length of each subnetwork could then be more closely matched to the lengths of the diphones to be recognized. Diphones could also be partitioned based on the first phoneme. This approach was used successfully in the two diphone recognizors of Table 4.4. In either case, the overall recognition rate should be quite good if the top-level network also performs well.

## C. LEXICAL ANALYZER IMPROVEMENTS

Unfortunately, time did not permit extensive experimentation with the lexical analyzer. In particular, improvements in two areas might increase word recognition accuracy. The first area is in the method used to compute the word *confidence level*. This is currently just the average of

the confidence levels (neural network output values) of the diphones used to build the word. This approach has several disadvantages:

- The lexical analyzer tends to favor pieces of words, if those pieces are themselves words and the pieces have higher average confidence values than the correct word. Both the complete word and its pieces will be in the output stream, but the syntactic and semantic analyzers will have to work harder to find the correct word if it is ranked lower.

- Short words can have their confidence values unduly influenced by one or two diphones. A single strong diphone can rocket the word to the top of the list. Similarly, a single weak diphone can bury the word at the bottom of the candidate list, or even cause it to be missed altogether.

- Words with only half of their diphones actually detected (every other diphone is sufficient for word detection) may score higher than words with every diphone detected (but a lower average confidence value). This is counter-intuitive.

Expert system developers face similar evidence weighting problems. Unfortunately, the uncertainty management methods typically applied to expert systems do not (at first glance) seem to fit this problem well.

The second area of the lexical analyzer that deserves scrutiny is related to the first. Other than rejecting diphones that appear in the stream unreasonably early or late, temporal information is not incorporated into the word confidence value calculations. It seems likely that the times between two given diphones would be normally distributed. Diphones which appear earlier or later than the expected time should, it seems, be penalized. Although it would add a small amount of complexity to the algorithm, doing so might greatly improve the lexical analyzer's discrimination.

## D.  SUMMARY AND CONCLUSIONS

Many questions remain unanswered. And the methods described in this thesis may require some adaptation before they can be applied to practical systems. But this research implies that

the diphone is indeed a promising recognition unit. Also, pitch-based frequency normalization and feedback-based time alignment appear to be potentially useful for speech recognition in general, and diphone recognition in particular. Hopefully, these techniques will bring the field a small step closer to the day when research can focus on the most serious challenges of speech recognition, semantic and pragmatic analysis.

# APPENDIX A. MATLAB CODE

## A. CONTENTS

### 1. TRNBPX.M

This is a script for training a neural network to recognize diphones. It requires a text file listing precomputed feature vectors (files) produced by MAKVECTS.M.

### 2. RECOGNIZ.M

This script file produces a text file listing diphones detected in designated speech (.WAV) files. Output from this script is used by the lexical analyzer for word recognition.

### 3. FEATURES.M

This function returns a feature surface for a designated portion of a speech (.WAV) file. Used by MAKVECTS.M.

### 4. FEATUREL.M

Same as FEATURES.M except the feature surfaces are for a word instead of a single diphone. Used by RECOGNIZ.M.

### 5. TESTNET.M

This script produces a detailed report of test set errors for a given set of network weights.

### 6. MAKVECTS.M

This script computes feature surfaces for a list of speech files. The speech file list must include a diphone name and sample numbers for each file. See Appendix B for the utilities used to produce that list.

### 7. INPUTS.M

This script obtains training parameters from the user before training the network. Invoked by TRNBPX.M.

### 8. P_AMDF.M

This function returns the speaker's apparent pitch for a sample of speech. Uses the Average Magnitude Difference Function (AMDF).

### 9. P_CEPS.M

This function returns the speaker's apparent pitch for a sample of speech. Uses cepstral pitch detection.

### 10. VECTORIZ.M

This function is used by TRNBPX.M to convert a feature surface into a feature vector.

### 11. DIPNUMOF.M

This function is used by TRNBPX.M to return an index number for a given diphone name.

### 12. DIFF_MAT.M

This function is used by MAKVECTS to compute the differenced version of a feature surface.

### 13. LD16BIT.M

This function (from the NPS SPC Toolbox) loads a PCM speech file.

### 14. LESSTHAN.M

This is a less than function for use in the binary search performed by DIPNUMOF.M.

### 15. STRCAT.M

This is a simple string concatenation function.

## B.    TRNBPX.M

```
% TRNBPX - Trains a neural network to recognize diphones
%
% This code contains many patches and a great deal of scaffolding for the
% purpose of trying out various training alternatives.  Like most prototypes,
% it should be thrown out and replaced with something cleaner and more
% efficient.
%
% User defineable parameters are loaded from inputs.mat, if it exists.
% Otherwise, INPUTS.M is invoked to obtain settings from the user.
% Remove inputs.mat before invoking TRNBPX.M if you wish to change
% settings from the previous run.
%
% The following text files must reside in the current directory:
%
% (1) A file listing the diphones to be recognized (one diphone
% per line, a single space separating the phonemes of the diphone,
% and all caps)
%
% (2) A file listing the training MAT files that were created
% by MAKVECTS.M
%
% (3) [optionally] a file listing the test MAT files that were created
% by MAKVECTS.M
%
% Written by:  Major Mark Cantrell, USMC
%
% Modified 11 Feb 96 (new time alignment method)
%          12 Feb 96 (trimmed fat to speed up code)
%          26 Feb 96 (re-anchored ref set in middle of recog window)
%           2 Mar 96 (Test files now use window center if time align off)
%           7 Mar 96 (Fixed bug in binary search from 12 Feb changes)
%           8 Mar 96 (Once again unleashed ref set)
%          17 Apr 96 * User inputs moved to INPUTS.M
%                    * Static file offsets optionally loaded from mat file
%                    * Randomness inserted in training file sequencing
%          20 Apr 96 (Allowed test set time alignment based on max_win_offset)
%          18 May 96 (Patched to use RMS Amplitude vice Short Time Power)
%          13 Jun 96 (fixed inconsistencies in use of w1 versus new_w1 and
%                          b1 versus new_b1 [same for w2 & b2])
%
%%%%%%%% Distribution and Copyright %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% This prologue must be included in all copies of this software.
%
% This software is released to the Public Domain
%
% Note:
%
%       Software released to the Public Domain is not subject
%       to copyright protection.
%
%       The MathWorks, Inc. holds the copyright on the MATLAB
%       Neural Network Toolbox functions invoked by this script file.
```

107

```
% Restrictions on use or distribution:  NONE
%
%%%%%%%% Disclaimer %%%%%%%%%%%%%%%%-
%
% This software and its documentation are provided "AS IS" and
% without any expressed or implied warranties whatsoever.
% No warranties as to performance, merchantability, or fitness
% for a particular purpose exist.
%
% Because of the diversity of conditions and hardware under
% which this software may be used, no warranty of fitness for
% a particular purpose is offered.  The user is advised to
% test the software thoroughly before relying on it.  The user
% must assume the entire risk and liability of using this
% software.
%
% In no event shall any person or organization of people be
% held responsible for any direct, indirect, consequential
% or inconsequential damages or lost profits.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOAD USER DEFINABLE PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if exist('inputs.mat')
  load('inputs');
else
  inputs;
end

revision = '12.0 (L1)';

pack

nice_start = 6;    % this is hour when training slowdown starts
nice_stop = 18;    % this is hour when training slowdown stops


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRAINING PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ~exist('seed')      % start rand num gen at known point each time to
  seed = 0;            % ensure that any performance changes aren't result
end                    % of different start pos

rand('seed',seed);

lr = 0.01;     % Learning rate, 0.01.
im = 1.05;     % Learning rate increase, default = 1.05.
dm = 0.7;      % Learning rate decrease, default = 0.7.

f1='logsig';   % hidden layer's transfer function
f2='logsig';   % output layer's transfer function
```

```
[hidden_size input_size] = size(w1);
win_width=input_size/30;

mid_pos = round((27-win_width)/2) + 1;
win_slack = 27-win_width+1;
win_scan_start = 14 - (round(win_width/2) + max_win_offset - 1);
win_scan_stop = win_scan_start + (2 * max_win_offset);
if win_scan_start < 1
  win_scan_start = 1;
end
if win_scan_stop > win_slack
  win_scan_stop = win_slack;
end

report_filename = 'nnreport.txt';


ste_start = 29*win_width+1;    % part of STE patch

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    LOAD THE LIST OF DIPHONES TO BE RECOGNIZED
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num_dips = 0;
dip_names = zeros(575,7);

if lfname==0
  return
else
  lfpname=strcat(lpname,lfname);
  list_file = fopen(lfpname);
  line_read = '  ';
  while (~feof(list_file)) & (length(line_read)>2)   % for every diphones listed
    line_read = upper(fgetl(list_file));
    if length(line_read) > 2
      num_dips = num_dips + 1;
      dip_names(num_dips,:) = blanks(length(dip_names(num_dips,:)));
      dip_names(num_dips,1:length(line_read)) = line_read;
    end
  end
  fclose(list_file);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% open training file list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if mfname==0
  return
end
mfpname=strcat(mpname,mfname);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% open test file list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if sep_test_data == 1

  if tfname==0
   return
  end
  tfpname=strcat(tpname,tfname);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   INITIALIZE THE NETWORK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

flops(0);      % reset flop counter
gigaflops = 0;

if (reset_weights == 1)      % randomize weights & reset epoch counter

  flops(0);     % reset counters
  gigaflops = 0;

  epoch_count = 1;

  best_correct = 0;          % used to determine when a set of wts
  best_wayoff = inf;         % is best so far
  best_SSE = inf;

  SSE_record = zeros(1,1);
  t_SSE_record = zeros(1,1);

  disp(strcat('Initializing the neural network for output layer size of ',num2str(num_dips)));

  input_limits = ones(win_width*30,2);   % input vector is win_width*30 inputs
  input_limits(1:win_width*29,1) = -10 * input_limits(1:win_width*29,1);       % freq mins
  input_limits(1:win_width*29,2) = 10 * input_limits(1:win_width*29,2);        % freq maxs
  input_limits(win_width*29+1:win_width*30,1) = -30 * input_limits(win_width*29+1:win_width*30,1);   %
freq mins
  input_limits(win_width*29+1:win_width*30,2) = 30 * input_limits(win_width*29+1:win_width*30,2);     %
freq maxs

  [w1,b1,w2,b2] = initff(input_limits,hidden_size,'logsig',num_dips,'logsig');

else        % resuming with same weights so use same epoch count & record

  epoch_count = epoch_count + 1;

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   final setup for training the network
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

df1 = feval(f1,'delta');
df2 = feval(f2,'delta');
```

```matlab
dw1 = w1*0;
db1 = b1*0;
dw2 = w2*0;
db2 = b2*0;

MC = 0;

last_SSE = inf;

new_w1 = w1;
new_b1 = b1;
new_w2 = w2;
new_b2 = b2;

                        % write header to report file
report_file = fopen(report_filename,'at+');
fprintf(report_file,'%s\n','-----------------------------------------------');
fprintf(report_file,'%s%s%s\n\n','NNET TRAINING RECORD Ver. ',revision,'
(Epochs,SSEs,#Correct,#MoreThan3off)');
fprintf(report_file,'%s\n',remarks);
fprintf(report_file,'%s\n',remarks2);
fprintf(report_file,'win_width = %d, max_offset = %d, seed = %d\n',win_width,max_win_offset,seed);
fprintf(report_file,'%s/%s: Hidden = %d, Num_Dips = %d , mc = %f , er =
%f\n\n',f1,f2,hidden_size,num_dips,mc,er);
tv = fix(clock);
time_stamp = strcat('Start Time = ',num2str(tv(1)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(2)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(3)));
time_stamp = strcat(time_stamp,' ');
time_stamp = strcat(time_stamp,num2str(tv(4)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(5)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(6)));
fprintf(report_file,'%s\n',time_stamp);
fclose(report_file);

if (time_align == 1) & (static_offsets == 1)
  load(offname);
else
  train_offsets = zeros(num_dips,1);     % for recording recognition window offsets
end

test_offsets = zeros(1,1);

if single_instance_only == 1
  ref_dips = zeros(1,1);
end

pass_num = 1;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    START OF EPOCH LOOP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for epoch_count = epoch_count:me   % epoch loop (one pass through training list)

                      % for 2nd & subsequent passes through training files,
                      % generate a random integer for each training file to
                      % allow shuffling of presentation sequence (see comments
                      % below)
if pass_num > 1
  rand_array = round(10*rand(1,length(train_offsets)));
  max_draws = 10;
else
  max_draws = 0;    % no shuffling on 1st pass through files
end

tv = fix(clock);

disp(strcat(num2str(epoch_count),' ---------------------[TRAINING]--------------------------'));
disp(tv);

correct = 0;
wrong_by_alot = 0;
SSE = 0;
dip_record = zeros(num_dips,3);       % for each dip in a pass: # seen, # correct, # way off
num_found = 0;

mat_list_file = fopen(mfpname);

for dip_set = 0:max_draws  % make max_draws+1 passes thru files, only training on
                      % a file when rand_array(file_num) == dip_set.
                      % This results in shuffling sequence of training
                      % file presentations each epoch.  On 1st epoch, all
                      % files are presented in file list sequence and in
                      % a single pass through the file list.

line_read = ' ';
dip_name = blanks(7);
file_num = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% training file loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while (~feof(mat_list_file)) & (length(line_read)>0)   % for every MAT file

line_read = (fgetl(mat_list_file));

file_num = file_num + 1;

        % if line not blank & this file's number is up (or 1st pass thru files)
```

```matlab
if (pass_num > 1) & (length(line_read) > 2)
  if (rand_array(file_num) == dip_set)
    file_enable = 1;
  else
    file_enable = 0;
  end
else
  file_enable = 1;
  if length(train_offsets) < file_num
    train_offsets(file_num,1) = mid_pos;
  end
end

if file_enable == 1

  if (length(line_read) > 2)
    load nice.txt -ascii;
    niceness=nice;

                              %sleep for niceness seconds if during work hours
    tv = fix(clock);
    if (niceness > 0) & (tv(4) > nice_start) & (tv(4) < nice_stop)
        pause(niceness);
    end

    filename = line_read;        % used to load file

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% pick out diphone name from filename
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    dot_loc=findstr(filename,'.');
    if dot_loc>0
        dip_str=filename(1:dot_loc-1);
        j = length(dip_str);
        while (dip_str(j) >= '0') & (dip_str(j) <= '9')
          j = j - 1;
        end
        i = j;
        while (dip_str(i) ~= '/') & (dip_str(i) ~= '\') & (i > 1)
          i = i - 1;
        end
        if i > 1
          i = i + 1;
        end
        dip_name = blanks(7);
        dip_name(1:j-i+1) = upper(dip_str(i:j));
        for i = 1:length(dip_name)         % get rid of hyphen
          if dip_name(i) == '-'
            dip_name(i) = ' ';
            break;
          end
        end
    else
        error(strcat('Bad MAT filename: ',filename));
    end
```

113

```
    elseif length(line_read) < 3
      break;                % bail out of eof loop if line blank
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% determine target output vector
% note that dipnames file MUST be alpha sorted & pound ('#') signs replaced by 'X' !!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          % use bin search to find dip's column number
    dip_num = dipnumof(dip_name,dip_names(1:num_dips,:));

    if strcmp(dip_names(dip_num,:),dip_name) ~= 1
      error(strcat('Is dip list sorted & are #s replaced by Xs? Unable to find ',dip_name));
    end
                      % target vector is all zeroes except a 1 at correct dip col
    t = zeros(num_dips,1);
    t(dip_num) = 1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if training on full set or this is first example of this diphone, load MAT file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if pass_num > 1
      if ref_dips(file_num,1) > 0
        ref_dip = 1;
      else
        ref_dip = 0;
      end
    else
      if dip_record(dip_num,1) < 1
        ref_dip = 1;
        ref_dips(file_num,1) = 1;
      else
        ref_dip = 0;
        ref_dips(file_num,1) = 0;
      end
    end

    if (single_instance_only == 0) | (ref_dip == 1)

                      % load mat file
      load(filename,'-mat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if this is first pass or (or every fifth pass) thru the files...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if (pass_num == 1) | (round(pass_num/5) == (pass_num/5))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ... and not time aligning
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if (time_align == 0)
```

```matlab
train_offsets(file_num,1) = mid_pos;     % center recognition window per timit transcription
win_pos = mid_pos;
best_pos = mid_pos;
win_pos = mid_pos;


                                        % sequence each column from matrix into vector
p = vectoriz(feature_surf_a,win_width,win_pos);

ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ...elseif applicable, use window offset from static offsets file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif static_offsets == 1

win_pos = train_offsets(file_num,1);     % center recognition window per offsets file
best_pos = win_pos;


                                        % sequence each column from matrix into vector
if win_pos < (win_slack+1)
  p = vectoriz(feature_surf_a,win_width,win_pos);
  ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
  p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));
else
  win_pos = win_pos - win_slack;
  p = vectoriz(feature_surf_b,win_width,win_pos);
  ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
  p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ...else use NNET feedback to time align vector.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

else

  best_out = -inf;
  best_pos = 0;
  best_rank = num_dips;


                    % look for best window pos in vector a
for win_pos = win_scan_start:win_scan_stop


                    % sequence each column from matrix into vector
  p = vectoriz(feature_surf_a,win_width,win_pos);
  ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
  p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

  if batch == 1
    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);
  else
    a1 = feval(f1,new_w1*p,new_b1);
    a2 = feval(f2,new_w2*a1,new_b2);
  end
```

115

```
a2_copy = a2;
[max_val max_pos] = max(a2_copy);

a2_copy(max_pos) = 0;
test_wrong_by = 0;
sum_winners = 0;
while max_pos ~= dip_num
  sum_winners = sum_winners + a2_copy(max_pos);
  [max_val max_pos] = max(a2_copy);
  a2_copy(max_pos) = 0;
  test_wrong_by = test_wrong_by +1;
end

if (test_wrong_by < best_rank)
  best_rank = test_wrong_by;
  best_out = a2(dip_num) - sum_winners;
  best_pos = win_pos;
  best_p = p;
elseif (test_wrong_by == best_rank)&((a2(dip_num) - sum_winners) > best_out)
  best_out = a2(dip_num) - sum_winners;
  best_pos = win_pos;
  best_p = p;
end

end

                    % look for best window pos in vector b
for win_pos = win_scan_start:win_scan_stop

                                        % sequence each column from matrix into vector
p = vectoriz(feature_surf_b,win_width,win_pos);
ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

if batch == 1
  a1 = feval(f1,w1*p,b1);
  a2 = feval(f2,w2*a1,b2);
else
  a1 = feval(f1,new_w1*p,new_b1);
  a2 = feval(f2,new_w2*a1,new_b2);
end

a2_copy = a2;
[max_val max_pos] = max(a2_copy);

a2_copy(max_pos) = 0;
test_wrong_by = 0;
sum_winners = 0;
while max_pos ~= dip_num
  sum_winners = sum_winners + a2_copy(max_pos);
  [max_val max_pos] = max(a2_copy);
  a2_copy(max_pos) = 0;
  test_wrong_by = test_wrong_by +1;
end
```

```
        if (test_wrong_by < best_rank)
          best_rank = test_wrong_by;
          best_out = a2(dip_num) - sum_winners;
          best_pos = win_pos + win_slack;
          best_p = p;
        elseif (test_wrong_by == best_rank)&((a2(dip_num) - sum_winners) > best_out)
          best_out = a2(dip_num) - sum_winners;
          best_pos = win_pos + win_slack;
          best_p = p;
        end

      end

      train_offsets(file_num,1) = best_pos;        % remember this window pos for next epoch
      p = best_p;

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ...else load correct window offset from array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

else

    win_pos = train_offsets(file_num);
    best_pos = win_pos;
    if win_pos < (win_slack+1)
                                              % sequence each column from matrix into vector
      p = vectoriz(feature_surf_a,win_width,win_pos);
      ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
      p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

    else
      win_pos = win_pos - win_slack;
                                              % sequence each column from matrix into vector
      p = vectoriz(feature_surf_b,win_width,win_pos);
      ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
      p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine outputs of layers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if batch == 1
    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);
else
    a1 = feval(f1,new_w1*p,new_b1);
    a2 = feval(f2,new_w2*a1,new_b2);
end
e = t - a2;
str = ' ';
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% record whether NNET output was correct or way off
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[max_val max_pos] = max(a2);
if (dip_record(dip_num,1) < 1)
    num_found = num_found + 1;
end
dip_record(dip_num,1) = dip_record(dip_num,1) + 1;
if max_pos ~= dip_num
    a2_copy = a2;
    a2_copy(max_pos) = 0;
    wrong_by = 0;
    while max_pos ~= dip_num
      [max_val max_pos] = max(a2_copy);
      a2_copy(max_pos) = 0;
      wrong_by = wrong_by +1;
    end
    str = strcat(strcat(strcat(str,' (wrong by '),num2str(wrong_by)),')');
    if wrong_by > 2
      wrong_by_alot = wrong_by_alot+1;
      dip_record(dip_num,3) = dip_record(dip_num,3) + 1;
    end
else
    correct = correct+1;
    dip_record(dip_num,2) = dip_record(dip_num,2) + 1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BACKPROPAGATION - Call approp delta function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d2 = feval(df2,a2,e);
if batch == 1
    d1 = feval(df1,a1,d2,w2);
else
    d1 = feval(df1,a1,d2,new_w2);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LEARNING - Use backprop to find weight changes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[dw1,db1] = learnbpm(p,d1,lr,MC,dw1,db1);
[dw2,db2] = learnbpm(a1,d2,lr,MC,dw2,db2);

new_w1 = new_w1 + dw1;       % adjust temp wts & bias's
new_b1 = new_b1 + db1;
new_w2 = new_w2 + dw2;
new_b2 = new_b2 + db2;


                         %  Determine effect of new weights on SSE
if batch == 1
    a1 = feval(f1,new_w1*p,new_b1);
    a2 = feval(f2,new_w2*a1,new_b2);
```

118

```
      e = t - a2;
   end

   SSE = SSE + sumsqr(e);

   end      % end of "if single instance..."

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % break out of file loop if one of each dip found & only one needed
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   if (num_found == num_dips) & (single_instance_only == 1)
     if (pass_num > 1)
        break;
     end
   end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % end of training set "while not(eof)" loop
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  end

  frewind(mat_list_file);

end  % end of file shuffling loop
fclose(mat_list_file);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%  TEST FILE CODE  %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MC = mc;

if (single_instance_only == 0) & (sep_test_data == 1)

  disp(strcat(num2str(correct),strcat(' correct and ',strcat(num2str(wrong_by_alot),' way off'))));

  disp(strcat(num2str(epoch_count),' ----------------------[TESTING]----------------------------'));
  disp(fix(clock));

  test_file = fopen(tfpname);

  line_read = ' ';
  dip_name = blanks(7);
  test_file_num = 0;
  test_SSE = 0;
  test_correct = 0;
  test_wrong_by_alot = 0;
  test_dip_record = zeros(num_dips,3);      % for each dip in last pass: # seen, # correct, # way off
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% test file loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while (~feof(test_file)) & (length(line_read)>0)   % for every test file
  line_read = (fgetl(test_file));
  if length(line_read) > 2

     load nice.txt -ascii;
     niceness=nice;

                              %sleep for niceness seconds if during work hours
     tv = fix(clock);
     if (niceness > 0) & (tv(4) > nice_start) & (tv(4) < nice_stop)
        pause(niceness);
     end

     filename = line_read;           % used to load file
     test_file_num = test_file_num + 1;

     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     % pick out diphone namefrom filename
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     dot_loc=findstr(filename,'.');
     if dot_loc>0
       dip_str=filename(1:dot_loc-1);
       j = length(dip_str);
       while (dip_str(j) >= '0') & (dip_str(j) <= '9')
         j = j - 1;
       end
       i = j;
       while (dip_str(i) ~= '/') & (dip_str(i) ~= '\') & (i > 1)
         i = i - 1;
       end
       if i > 1
         i = i + 1;
       end
       dip_name = blanks(7);
       dip_name(1:j-i+1) = upper(dip_str(i:j));
       for i = 1:length(dip_name)          % get rid of hyphen
         if dip_name(i) == '-'
           dip_name(i) = ' ';
           break;
         end
       end
     else
       error(strcat('Bad TEST filename: ',filename));
     end

else
   break;              % bail out of eof loop if line blank
end
```

120

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% determine target output vector
% note that dipnames file MUST be alpha sorted & pound ('#') signs replaced by 'X' !!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % use bin search to find dip's column number
dip_num = dipnumof(dip_name,dip_names(1:num_dips,:));

if strcmp(dip_names(dip_num,:),dip_name) ~= 1
    error(strcat('Is dip list sorted & are #s replaced by Xs? Unable to find ',dip_name));
end
                        % target vector is all zeroes except a 1 at correct dip col
t = zeros(num_dips,1);
t(dip_num) = 1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% load test MAT file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load(filename, '-mat');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% use NNET feedback to time align input vectors
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


best_out = -inf;
best_pos = 0;
best_rank = num_dips;

                    % look for best window pos in vector a
for win_pos = win_scan_start:win_scan_stop

                                        % sequence each column from matrix into vector
    p = vectoriz(feature_surf_a,win_width,win_pos);
    ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
    p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);

    a2_copy = a2;
    [max_val max_pos] = max(a2_copy);

    a2_copy(max_pos) = 0;
    test_wrong_by = 0;
    sum_winners = 0;
    while max_pos ~= dip_num
        sum_winners = sum_winners + a2_copy(max_pos);
        [max_val max_pos] = max(a2_copy);
        a2_copy(max_pos) = 0;
        test_wrong_by = test_wrong_by +1;
    end

    if (test_wrong_by < best_rank)
```

```matlab
      best_rank = test_wrong_by;
      best_out = a2(dip_num) - sum_winners;
      best_pos = win_pos;
      best_p = p;
    elseif (test_wrong_by == best_rank)&((a2(dip_num) - sum_winners) > best_out)
      best_out = a2(dip_num) - sum_winners;
      best_pos = win_pos;
      best_p = p;
    end

  end

                    % look for best window pos in vector b
  for win_pos = win_scan_start:win_scan_stop

                                        % sequence each column from matrix into vector
    p = vectoriz(feature_surf_b,win_width,win_pos);
    ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
    p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);

    a2_copy = a2;
    [max_val max_pos] = max(a2_copy);

    a2_copy(max_pos) = 0;
    test_wrong_by = 0;
    sum_winners = 0;
    while max_pos ~= dip_num
      sum_winners = sum_winners + a2_copy(max_pos);
      [max_val max_pos] = max(a2_copy);
      a2_copy(max_pos) = 0;
      test_wrong_by = test_wrong_by +1;
    end

    if (test_wrong_by < best_rank)
      best_rank = test_wrong_by;
      best_out = a2(dip_num) - sum_winners;
      best_pos = win_pos + win_slack;
      best_p = p;
    elseif (test_wrong_by == best_rank)&((a2(dip_num) - sum_winners) > best_out)
      best_out = a2(dip_num) - sum_winners;
      best_pos = win_pos + win_slack;
      best_p = p;
    end

  end

  test_offsets(test_file_num,1) = best_pos;     % save window ofset for next epoch
  p = best_p;


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Determine outputs of layers
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
a1 = feval(f1,w1*p,b1);
a2 = feval(f2,w2*a1,b2);
e = t - a2;
test_SSE = test_SSE + sumsqr(e);

str = ' ';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% record whether NNET output was correct or way off
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[max_val max_pos] = max(a2);
test_dip_record(dip_num,1) = test_dip_record(dip_num,1) + 1;
if max_pos ~= dip_num
    a2_copy = a2;
    a2_copy(max_pos) = 0;
    test_wrong_by = 0;
    while max_pos ~= dip_num
      [max_val max_pos] = max(a2_copy);
      a2_copy(max_pos) = 0;
      test_wrong_by = test_wrong_by +1;
    end
    str = strcat(strcat(strcat(str,' (wrong by '),num2str(test_wrong_by)),')');
    if test_wrong_by > 2
      test_wrong_by_alot = test_wrong_by_alot+1;
      test_dip_record(dip_num,3) = test_dip_record(dip_num,3) + 1;
    end
  else
    test_correct = test_correct+1;
    test_dip_record(dip_num,2) = test_dip_record(dip_num,2) + 1;
  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% end of while not eof test_file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fclose(test_file);

end       % end of "if sep_test_data..."

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% store SSE(s) in array(s)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (single_instance_only==1)
  SSE_record(epoch_count) = SSE/num_dips;
else
  SSE_record(epoch_count) = SSE/file_num;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% display epoch progress
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

123

```matlab
disp(strcat('* SSE went from ',num2str(last_SSE)));
disp(strcat('*          to ',num2str(SSE)));
disp(strcat('*    change was ',num2str(SSE-last_SSE)));
disp(strcat(num2str(correct),strcat(' trng vects OK & ',strcat(num2str(wrong_by_alot),' way off'))));
if (sep_test_data == 1)
  t_SSE_record(epoch_count) = test_SSE/test_file_num;
  disp(strcat(num2str(test_correct),strcat(' test vectors OK & ',strcat(num2str(test_wrong_by_alot),' way
off'))));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% save weights if this is best result so far
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (single_instance_only==0)

 if sep_test_data == 0
   test_correct = correct;
   test_wrong_by_alot = wrong_by_alot;
 end

 if (test_wrong_by_alot < best_wayoff)I((test_wrong_by_alot == best_wayoff) & (test_correct >
best_correct))
   efname = 'bestwts';
   eval(['save ' efname ]);   % save best wts
   best_wayoff = test_wrong_by_alot;
   best_correct = test_correct;
 end

 if test_SSE < best_SSE
   efname = 'bestsse';
   eval(['save ' efname ]);   % save best sse wts
   best_SSE = test_SSE;
 end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% momentum & adaptive learning rate phases
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if SSE > last_SSE * er

 lr = lr * dm;
 MC = 0;

 new_w1 = w1;
 new_b1 = b1;
 new_w2 = w2;
 new_b2 =-b2;

 disp('* Error increased.  Rejecting weight changes & decreasing learning rate.');
 str = '****';

else
```

```
  if SSE < last_SSE
      lr = lr * im;
  end

  w1 = new_w1;
  b1 = new_b1;
  w2 = new_w2;
  b2 = new_b2;
  last_SSE = SSE;

  str = '-';

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% record epoch stats in report file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

report_file = fopen(report_filename,'at+');
fprintf(report_file,'%d, TRAIN,%f,%d,%d,',epoch_count,SSE,correct,wrong_by_alot);
if sep_test_data== 1
  fprintf(report_file,' TEST,%f,%d,%d, %s\n',test_SSE,test_correct,test_wrong_by_alot,str);
else
  fprintf(report_file,' %s\n',str);
end
fclose(report_file);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% break out of for loop if error goal reached
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (single_instance_only ==0) & (SSE < (eg * file_num))
  break;
elseif SSE < (eg * num_dips)
  break;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% increment pass counter & prevent flop counter overflow
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pass_num = pass_num + 1;

flop_count = flops;   % prevent flop counter overflow
if flop_count >= 1e9
  gigaflops = gigaflops + (flop_count/(1e9));
  flops(0);
end

save lastwts;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bottom of epoch loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

125

```matlab
disp('----------------------------------------------');

save finalwts;     % save final state (including final wts & bias')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Display final results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (single_instance_only ==0) & (SSE < (eg * file_num))
  str = 'Error goal reached!  Final SSE:';
elseif (single_instance_only ==1) & (SSE < (eg * num_dips))
  str = 'Error goal reached!  Final SSE:';
else
  str = 'Error goal not reached!  Final SSE:';
end

disp(str);
disp(SSE);
if gigaflops > 0
  gigaflops = gigaflops + (flop_count/(1e9));
  flops(0);
  disp('Giga-Flops:');
  disp(gigaflops);
else
  disp('Flops:');
  disp(flops);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write trailer to report file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

report_file = fopen(report_filename,'at+');
tv = fix(clock);
time_stamp = strcat('Stop Time = ',num2str(tv(1)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(2)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(3)));
time_stamp = strcat(time_stamp,' ');
time_stamp = strcat(time_stamp,num2str(tv(4)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(5)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(6)));
fprintf(report_file,'\n%s\n',time_stamp);
if gigaflops > 0
  fprintf(report_file,'%f giga-flops\n',gigaflops);
else
  fprintf(report_file,'%d flops\n',flops);
end
fprintf(report_file,'%s  %f\n',str,SSE);
fprintf(report_file,'\n%s\n\n','END OF NNET TRAINING RECORD');
fclose(report_file);
```

## C.   RECOGNIZ.M

```
% RECOGNIZ - Use the currently loaded weights to produce a text file
% showing diphone detector outputs for wav files listed in another
% text file.
%
%   Sample WAV file list format:
%   /timit/test/dr1/mcje0/sx33.wav ocean 1234 3499
%   ...
%   /timit/test/dr8/mtrw0/sx343.wav rabbit 21234 23499
%
%   Sample Output:
%   (word_is ocean)
%   ((1008 (AE P) 0.8) (1008 (R OW) 0.6) (1008 (IY T) 0.4))
%   ((1010 (OW K) 0.9) (1010 (OW B) 0.6) (1010 (UH JH) 0.4))
%   ...
%   (word_end ocean)
%
% Written by:  Major Mark Cantrell, USMC
% Date: 20 Dec 1995
%
%       2 Jun 1996  -  Switched to 2 interleaved feature surfaces to improve
%                       resolution
%       7 Jun 1996  -  Switched from short-time power to RMS amplitude in
%                       column 30 of feature surfaces
%

revision = '3.0';


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  VARIABLES & PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

speech_data = [];      % vector with entire speech file's contents
sampsize = [];         % size of above

fname = [];     % will be string with filename
pname = [];     % will be string with pathname
fpname = [];    % will be string with filename + pathname
varname = [];   % will be string with filename less extension

win_step = 512-170;  % This is how much (samples) each recognition window moves
                     % each time step. resolution is half this with interleaving

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get wav/word file list filename from user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


[fname,pname] = uigetfile('*.*','WAV/Word File List File');
if fname==0
  return
end
fpname=strcat(pname,fname);
```

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% read in list of diphone names
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num_dips = 0;
dip_names = zeros(575,7);

dip_file = fopen('diplist.txt');
line_read = '  ';
while (~feof(dip_file)) & (length(line_read)>2)   % for every diphone listed
  line_read = upper(fgetl(dip_file));
  if length(line_read) > 2
    num_dips = num_dips + 1;
    dip_names(num_dips,:) = blanks(length(dip_names(num_dips,:)));
    dip_names(num_dips,1:length(line_read)) = line_read;
  end
end
fclose(dip_file);

dip_names = dip_names(1:num_dips,:);


list_file = fopen(fpname);

result_file = fopen('recogniz.out','wt');

line_read = fgetl(list_file);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main list file loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while (~feof(list_file)) & (length(line_read)>0)   % for every filename listed in list file

  disp('----------------------------------------------------------------');
  disp(line_read);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%

  dot_loc=findstr(line_read,'.');  % pick out pathname from line read
  fpname_end=dot_loc+3;
  fpname = line_read(1:fpname_end);

  disp (strcat('Pathname is ',pname));

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

              % pick out word name
  wordname_start = fpname_end + 2;
  i = wordname_start;
  while line_read(i) ~= ' '
    i=i+1;
  end
  wordname_end = i - 1;
  wordname = line_read(wordname_start:wordname_end);
```

128

```
disp(strcat('Word is ',wordname));

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % get location of this diphone's phoneme junction

word_limits = sscanf(line_read(wordname_end+1:length(line_read)),'%d %d');
word_start = word_limits(1);
word_end = word_limits(2);

disp(strcat('Word start is: ',num2str(word_start)));
disp(strcat('Word end is: ',num2str(word_end)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(strcat('Loading ',fpname));

speech_data = ld16bit(fpname);
speech_data=speech_data';
speech_data=speech_data(513:length(speech_data));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compute 2 feature surfaces (interleaved in time)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[feature_surf,median_pitch,formants_a] = feature1(speech_data,word_start-170,word_end-170);
feature_surf1 = diff_mat(feature_surf);

ste=feature_surf1(:,30);  % convert STP to RMS amplitude
feature_surf1(:,30) = sign(ste).*sqrt(abs(ste));

[feature_surf,median_pitch,formants_a] = feature1(speech_data,word_start,word_end);
feature_surf2 = diff_mat(feature_surf);

ste=feature_surf2(:,30);  % convert STP to RMS amplitude
feature_surf2(:,30) = sign(ste).*sqrt(abs(ste));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% determine network outputs using current weights
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[feature_rows fc] = size(feature_surf1);

win_slack = feature_rows - win_width + 1;

result_matrix = zeros(2*win_slack,num_dips);

for win_pos = 1:win_slack

    for i = 1:30     % sequence each column from matrix 1 into vector
        p(i*win_width-win_width+1:i*win_width,1) = feature_surf1(win_pos:win_pos+win_width-1,i);
    end
    a1 = feval(f1,w1*p,b1);
    a2 = feval(f2,w2*a1,b2);
    result_matrix(2*win_pos-1,:) = a2';

    for i = 1:30     % sequence each column from matrix 2 into vector
```

```
            p(i*win_width-win_width+1:i*win_width,1) = feature_surf2(win_pos:win_pos+win_width-1,i);
        end
        a1 = feval(f1,w1*p,b1);
        a2 = feval(f2,w2*a1,b2);
        result_matrix(2*win_pos,:) = a2';

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % write this word's outputs to the file
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fprintf(result_file,'(word_is %s)\n',wordname);

    for win_pos = 1:2*win_slack

        outputs_copy = result_matrix(win_pos,:);

        [max_val max_pos] = max(outputs_copy);
        out1 = max_val;
        dip1 = dip_names(max_pos,:);
        outputs_copy(max_pos) = 0;

        [max_val max_pos] = max(outputs_copy);
        out2 = max_val;
        dip2 = dip_names(max_pos,:);
        outputs_copy(max_pos) = 0;

        [max_val max_pos] = max(outputs_copy);
        out3 = max_val;
        dip3 = dip_names(max_pos,:);
        outputs_copy(max_pos) = 0;

        fprintf(result_file,'((%d (%s) %f) (%d (%s) %f) (%d (%s) %f))\n',
            in_pos*win_step/2,dip1,out1,win_pos*win_step/2,dip2,out2,win_pos*win_step/2,dip3,out3);

    end

    fprintf(result_file,'(word_end %s)\n',wordname);

    line_read = fgetl(list_file);

end     % end of while not EOF(file list file) loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fclose(result_file);
fclose(list_file);
```

## D.  FEATURES.M

```
function [out_matrix,out_pitch,form_matrix] = features(speech_data,dip_boundry)
%FEATURES      Return the matrix form of a feature vector for a given
%              speech slice.  The recognition window will be centered
%              on the sample number specified in the second argument.
%              Matrix will include an extra two time periods on either
%              side of the 24 needed for producing the final vector.
%              The second value returned is median pitch for sample.
%
%              Note: This version currently returns a measure of short-time
%              power in column 30 of the feature surface.  As explained in
%              the thesis, RMS amplitude would have been better.  TRNBPX has
%              been patched to use the latter.  But it would be more efficient
%              in future if RMS amplitude used in original feature surfaces.
%
%USAGE EXAMPLE:     [feature_matrix pitch] = features(data(100:12000),5000);
%
% Revision 2.0;  Written by Major M. E. Cantrell, USMC
%
%      7 June 1996  -  Patched to use RMS amplitude vice STP

fs = 16000;              % sample rate
ps = 1/fs;              % sample period
n = 512;                % window size for FFT
win_overlap = 170;      % how much sliding window overlaps last window
L=round(n/10);          % initial value for the "low time lifter" used with cepstrum

speech_slice=zeros(1,9746);             % will hold the slice of main file in
                                        % in recognition window
slice_size=9746;

                                        % center window
speech_start = dip_boundry - 4446;      % start of 14th window pos is @4447
speech_end = dip_boundry + 5299;
slice_start = 1;
slice_end = 9746;
speech_length = length(speech_data);

        % following vectors hold start & stop freqs for freq bins.
        % these will be adapted for speaker's pitch & then used to
        % determine which formant surface columns to average for each
        % freq bin in output matrix.
bin_starts = [
0
74.4
149.4
225.3
302.7
382.2
464.5
550.4
640.6
735.9
837.3
945.8
1062.6
```

```
1189.1
1326.8
1477.6
1643.6
1827.2
2031.5
2260.0
2516.9
2807.4
3137.7
3515.5
3950.0
4453.1
5039.5
5727.5
6541.0
];

bin_ends = [
100
175.6
252.3
330.8
411.7
495.6
583.4
675.8
773.8
878.2
990.3
1111.2
1242.5
1385.8
1543.1
1716.6
1909.2
2124.1
2365.0
2636.8
2945.1
3296.7
3700.3
4166.5
4708.4
5342.9
6091.3
6981.0
8047.8
];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % pad voice data with zeros if needed
if speech_start < 1
  slice_start = 2 - speech_start;
  speech_start = 1;
end
```

```
if speech_end > speech_length
   slice_end = 4447 + (speech_length - dip_boundry);
   speech_end = speech_length;
end

speech_slice(slice_start:slice_end) = speech_data(speech_start:speech_end);

%plot(speech_slice);
%drawnow;z=input('Press enter to continue...');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         % build matrices that will hold cepstral & short time energy data

form_cols = n/2;           % at fs=16K & n=512 this covers 0-8K Hz
form_rows = 28;

form_matrix = zeros(form_rows,form_cols);

out_matrix = zeros(form_rows,30);

ste_array = zeros(form_rows,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hamming_vector=zeros(1,n);     % compute window & "lifter" function for size=n
hamming_vector=hamming(n)';

                %compute "lifter's" coefficients for cepstrum/formants
lifter1=zeros(1,n);
last=n;
for i = 0:L
    lifter1(i+1) = 1+cos(pi*i/(2*L));
    lifter1(last-i) = lifter1(i+1);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main "program" loop starts here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

win_start = 1;        % start sliding window at first sample
win_end = n;
pitch_array = zeros(form_rows,1);      % for computing median of pitch values
pitch_count = 0;
form_x = 1;

win_size = win_end - win_start + 1;

while (form_x <= form_rows)          % main processing loop

 ste = (speech_slice(1,win_start:win_end) .* speech_slice(1,win_start:win_end))/100000;
 ste_array(form_x,1) = mean(ste(1,:));

 if max(abs(speech_slice(1,win_start:win_end))) > 0

                    % compute FFT
```

133

```
abs_fft=abs(fft(hamming_vector.*speech_slice(1,win_start:win_end)));

                    % now compute "cepstrum";

RCEP=ifft(log(abs_fft));
rcep=abs(RCEP(1:win_size/2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % determine pitch

[peak_val,peak_loc] = max(rcep(L:length(rcep)));
pitch_prominence = peak_val/(10*median(rcep(L:length(rcep))));
cep_pitch=fs/(L+peak_loc);
if (pitch_prominence >= 1.5)    % 1.5 is probably a safer number (but slow!)
  pitch=cep_pitch;
  pitch_count = pitch_count + 1;
  pitch_array(pitch_count)=pitch;
elseif ((pitch_prominence >= 0.6) & (pitch_count < 1))
              % if two pitch estimates are within 10 % of each other, use cep_pitch
  a_pitch=p_amdf(speech_slice(1,win_start:win_end),fs,n-(1+((1/50)/(1/fs))));
  if ((((abs(cep_pitch-a_pitch))/cep_pitch) < 0.10)) | ((cep_pitch >= 100) & (cep_pitch <= 200) &
(pitch_count < 1))
      pitch=cep_pitch;
      pitch_count = pitch_count + 1;
      pitch_array(pitch_count)=pitch;
  end
else
  pitch=0;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % smooth formants

RCEPL1 = zeros(1,length(RCEP));
                              %"lifter" out pitch & rhamonics
RCEPL1(1:length(RCEP)) = RCEP(1:length(RCEP)) .* lifter1(1:length(RCEP));

FORMANTS = fft(RCEPL1);        %now take FFT to see formants
form_matrix(form_x,:) = abs(FORMANTS(1:n/2));

%disp('Pitch_Prom:');
%disp(pitch_prominence);
%disp('Pitch:');
%disp(pitch);
%plot(abs(FORMANTS(1:n/2))),title('FORMANTS');
%drawnow;z=input('Press enter to continue...');
%plot(rcep(L:length(rcep))),title('CEPSTRUM');
%drawnow;z=input('Press enter to continue...');

  end        % end of "if non-zero amplitude"

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % slide window to next (overlapped) position

form_x = form_x + 1;
```

```
  win_start = win_end + 1 - win_overlap;
  win_end = win_start + n - 1;

end  % while

%mesh(form_matrix);
%drawnow;z=input('Press enter to continue...');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main "program" loop ends here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % now "normalize" data for speaker's pitch

if pitch_count > 0
  pitch = median(pitch_array(1:pitch_count));
  if (pitch < 50) I (pitch > 400)
    pitch = 100;
    out_pitch = 0;
  else
    out_pitch = pitch;
  end
else
  pitch = 100;
  out_pitch = 0;
end

bin_starts = bin_starts / (1.185 - (0.00185 * pitch));
bin_ends = bin_ends / (1.185 - (0.00185 * pitch));

for i = 1:29
  bin_start = round(n*bin_starts(i)/fs);
  if bin_start < 1
    bin_start = 1;
  end
  if bin_start > n/2
    bin_start = n/2;
  end
  bin_end = round(n*bin_ends(i)/fs);
  if bin_end < 1
    bin_end = 1;
  end
  if bin_end > n/2
    bin_end = n/2;
  end
  for j = 1:form_rows
    out_matrix(j,i) = mean(form_matrix(j,bin_start:bin_end));
  end
end

out_matrix(:,30) = ste_array(:,1);

%mesh(out_matrix);
%drawnow;z=input('Press enter to continue...');
```

135

## E.    FEATUREL.M

```
function [out_matrix,out_pitch,form_matrix] = featurel(speech_data,start,end)
%FEATUREL      Return the matrix form of a feature vector for a given
%              speech slice.
%              The second value returned is median pitch for sample.
%
%              Note: This version currently returns a measure of short-time
%              power in column 30 of the feature surface.  As explained in
%              the thesis, RMS amplitude would have been better.  RECOGNIZ has
%              been patched to use the latter.  But it would be more efficient
%              in future if RMS amplitude used in original feature surfaces.
%
%USAGE EXAMPLE:      [feature_matrix pitch] = featurel(data(100:12000));
%
% Revision 2.0;  Written by Major M. E. Cantrell, USMC
%
%      7 June 1996  -  Patched to use RMS amplitude vice STP


fs = 16000;           % sample rate
ps = 1/fs;            % sample period
n = 512;              % window size for FFT
win_overlap = 170;    % how much sliding window overlaps last window
L=round(n/10);        % initial value for the "low time lifter" used with cepstrum

                            % center window
speech_start = start - 4446;       % start of 14th window pos is @4447
speech_end = end + 5299;
speech_slice=zeros(1,speech_end-speech_start+1);

slice_start = 1;
slice_end = length(speech_slice);
slice_size = slice_end;
speech_length = length(speech_data);


                      % pad voice data with zeros if needed
if speech_start < 1
  slice_start = 2 - speech_start;
  speech_start = 1;
end

if speech_end > speech_length
  speech_end = speech_length;
  slice_end = slice_start + (speech_end - speech_start);
%  slice_end = 4447 + (speech_length - end);
%  speech_end = speech_length;
end

speech_slice(slice_start:slice_end) = speech_data(speech_start:speech_end);

%plot(speech_slice);
%drawnow;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % following vectors hold start & stop freqs for freq bins.
        % these will be adapted for speaker's pitch & then used to
        % determine which formant surface columns to average for each
        % freq bin in output matrix.
bin_starts = [
0
74.4
149.4
225.3
302.7
382.2
464.5
550.4
640.6
735.9
837.3
945.8
1062.6
1189.1
1326.8
1477.6
1643.6
1827.2
2031.5
2260.0
2516.9
2807.4
3137.7
3515.5
3950.0
4453.1
5039.5
5727.5
6541.0
];

bin_ends = [
100
175.6
252.3
330.8
411.7
495.6
583.4
675.8
773.8
878.2
990.3
1111.2
1242.5
1385.8
1543.1
1716.6
1909.2
2124.1
```

```
2365.0
2636.8
2945.1
3296.7
3700.3
4166.5
4708.4
5342.9
6091.3
6981.0
8047.8
];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % build matrices that will hold cepstral & short time energy data

form_cols = n/2;            % at fs=16K & n=512 this covers 0-8K Hz
form_rows = round(slice_size/(n-win_overlap)) + 1;

form_matrix = zeros(form_rows,form_cols);

out_matrix = zeros(form_rows,30);

ste_array = zeros(form_rows,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hamming_vector=zeros(1,n);     % compute window & "lifter" function for size=n
hamming_vector=hamming(n)';

            %compute "lifter's" coefficients for cepstrum/formants
lifter1=zeros(1,n);
last=n;
for i = 0:L
    lifter1(i+1) = 1+cos(pi*i/(2*L));
    lifter1(last-i) = lifter1(i+1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main "program" loop starts here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

win_start = 1;        % start sliding window at first sample
win_end = n;
pitch_array = zeros(form_rows,1);     % for computing median of pitch values
pitch_count = 0;
form_x = 1;

win_size = win_end - win_start + 1;

while (win_end <= slice_end)          % main processing loop

ste = (speech_slice(1,win_start:win_end) .* speech_slice(1,win_start:win_end))/100000;
ste_array(form_x,1) = mean(ste(1,:));

if max(abs(speech_slice(1,win_start:win_end))) > 0
```

```
                    % compute FFT

  abs_fft=abs(fft(hamming_vector.*speech_slice(1,win_start:win_end)));

                    % now compute "cepstrum";

  RCEP=ifft(log(abs_fft));
  rcep=abs(RCEP(1:win_size/2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % determine pitch

  [peak_val,peak_loc] = max(rcep(L:length(rcep)));
  pitch_prominence = peak_val/(10*median(rcep(L:length(rcep))));
  cep_pitch=fs/(L+peak_loc);
  if (pitch_prominence >= 1.5)
    pitch=cep_pitch;
    pitch_count = pitch_count + 1;
    pitch_array(pitch_count)=pitch;
  elseif (pitch_prominence >= 0.6)
                    % if two pitch estimates are within 10 % of each other, use cep_pitch
    a_pitch=p_amdf(speech_slice(1,win_start:win_end),fs,n-(1+((1/50)/(1/fs))));
    if ((((abs(cep_pitch-a_pitch))/cep_pitch) < 0.10)) | ((cep_pitch >= 100) & (cep_pitch <= 200) &
(pitch_count < 1))
        pitch=cep_pitch;
        pitch_count = pitch_count + 1;
        pitch_array(pitch_count)=pitch;
    end
  else
    pitch=0;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % smooth formants using real cepstrum

  RCEPL1 = zeros(1,length(RCEP));
                              %"lifter" out pitch & rhamonics
  RCEPL1(1:length(RCEP)) = RCEP(1:length(RCEP)) .* lifter1(1:length(RCEP));

  FORMANTS = fft(RCEPL1);      %now take FFT to see formants
  form_matrix(form_x,:) = abs(FORMANTS(1:n/2));

%disp('Pitch_Prom:');
%disp(pitch_prominence);
%disp('Pitch:');
%disp(pitch);
%plot(abs(FORMANTS(1:n/2))),title('FORMANTS');
%pause;
%plot(rcep(L:length(rcep))),title('CEPSTRUM');
%pause;

  end        % end of "if non-zero amplitude"

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % slide window to next (overlapped) position
```

```
  form_x = form_x + 1;

  win_start = win_end + 1 - win_overlap;
  win_end = win_start + n - 1;

end  % while

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main "program" loop ends here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % now "normalize" data for speaker's pitch

if pitch_count > 0
  pitch = median(pitch_array(1:pitch_count));
  if (pitch < 50) | (pitch > 400)
    pitch = 100;
    out_pitch = 0;
  else
    out_pitch = pitch;
  end
else
  pitch = 100;
  out_pitch = 0;
end

bin_starts = bin_starts / (1.185 - (0.00185 * pitch));
bin_ends = bin_ends / (1.185 - (0.00185 * pitch));

for i = 1:29
  bin_start = round(n*bin_starts(i)/fs);
  if bin_start < 1
    bin_start = 1;
  end
  if bin_start > n/2
    bin_start = n/2;
  end
  bin_end = round(n*bin_ends(i)/fs);
  if bin_end < 1
    bin_end = 1;
  end
  if bin_end > n/2
    bin_end = n/2;
  end
  for j = 1:form_rows
    out_matrix(j,i) = mean(form_matrix(j,bin_start:bin_end));
  end
end

out_matrix(:,30) = ste_array(:,1);

%plot(ste_array(:,1));
%drawnow;
```

## F. TESTNET.M

```
% TESTNET - Gathers statistics for neural network weights trained by TRNBPX.
%         These weights must already be loaded when this script file is
%         started.
%
% You must provide two text files:
%
% (1) A text file listing the diphones to be recognized (one diphone
% per line, a single space separating the phonemes of the diphone,
% and all caps)
%
% (2) A text file listing the test MAT files created by MAKVECTS.M
%
% Written by:  Major Mark Cantrell, USMC Feb 1996
%
% Last Updated  8 Feb 96   (new time alignment method)
%          20 Apr 96   (bug fixes)
%           7 Jun 96   (patched to use RMS Amplitude vice short-time power)

revision = '5.2';

if ~exist('disp_files')
  disp_files = 0;
end


max_win_offset = input('Maximum offset from center for recognition window? ');
mid_pos = round((27-win_width)/2) + 1;
win_slack = 27-win_width+1;
win_scan_start = 14 - (round(win_width/2) + max_win_offset - 1);
win_scan_stop = win_scan_start + (2 * max_win_offset);
if win_scan_start < 1
  win_scan_start = 1;
end
if win_scan_stop > win_slack
  win_scan_stop = win_slack;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % write header to report file
report_filename = 'nntest.txt';
report_file = fopen(report_filename,'at+');
fprintf(report_file,'%s\n','-----------------------------------------------');
fprintf(report_file,'%s%s%s\n\n','NNET TEST RECORD Ver. ',revision','
(Epochs,SSEs,#Correct,#MoreThan3off)');
fprintf(report_file,'%s\n',remarks);
fprintf(report_file,'%s\n win_width = %d, max_offset = %d, seed = %d\n',win_width,max_win_offset);
fprintf(report_file,'%s/%s: Hidden = %d, Num_Dips = %d , mc = %f , er =
%f\n\n',f1,f2,hidden_size,num_dips,mc,er);
tv = fix(clock);
time_stamp = strcat('Start Time = ',num2str(tv(1)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(2)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(3)));
time_stamp = strcat(time_stamp,' ');
```

141

```matlab
time_stamp = strcat(time_stamp,num2str(tv(4)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(5)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(6)));
fprintf(report_file,'%s\n',time_stamp);
fclose(report_file);

test_record = zeros(num_dips,5);
test_offsets = zeros(1,1);
pass_num = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% TEST CODE START
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(strcat(num2str(epoch_count),' ---------------------[TESTING]---------------------------'));
disp(fix(clock));

test_file = fopen(tfpname);

line_read = ' ';
dip_name = blanks(7);
test_file_num = 0;
test_SSE = 0;
test_correct = 0;
test_wrong_by_alot = 0;
test_dip_record = zeros(num_dips,3);      % for each dip in last pass: # seen, # correct, # way off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% test file loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

report_file = fopen(report_filename,'at+');
fprintf(report_file,'\nDIP    OUT    ER RECOG   OUT    OFFSET \n');

while (~feof(test_file)) & (length(line_read)>0)   % for every test file
    line_read = (fgetl(test_file));
    if length(line_read) > 2

        filename = line_read;           % used to load file
        test_file_num = test_file_num + 1;
        if disp_files == 1
          disp(strcat(strcat(num2str(test_file_num),' = test file '),line_read));
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % pick out diphone namefrom filename
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        dot_loc=findstr(filename,'.');
        if dot_loc>0
          dip_str=filename(1:dot_loc-1);
          j = length(dip_str);
          while (dip_str(j) >= '0') & (dip_str(j) <= '9')
            j = j - 1;
          end
```

```
      i = j;
      while (dip_str(i) ~= '/') & (dip_str(i) ~= '\') & (i > 1)
        i = i - 1;
      end
      if i > 1
        i = i + 1;
      end
      dip_name = blanks(7);
      dip_name(1:j-i+1) = upper(dip_str(i:j));
      for i = 1:length(dip_name)          % get rid of hyphen
        if dip_name(i) == '-'
          dip_name(i) = ' ';
        end
      end
    else
      error(strcat('Bad TEST filename: ',filename));
    end

  else
    break;              % bail out of eof loop if line blank
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% determine target output vector
% note that dipnames file MUST be alpha sorted & pound ('#') signs replaced by 'X' !!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % use bin search to find dip's column number
dip_num = dipnumof(dip_name,dip_names(1:num_dips,:));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% load test MAT file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load(filename, '-mat');
p = zeros(win_width*30,1);

best_out = -inf;
best_pos = 0;
best_rank = num_dips;

                    % look for best window pos in vector a
for win_pos = win_scan_start:win_scan_stop

                    % sequence each column from matrix into vector
  p = vectoriz(feature_surf_a,win_width,win_pos);
  ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
  p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

  a1 = feval(f1,w1*p,b1);
  a2 = feval(f2,w2*a1,b2);

  a2_copy = a2;
  [max_val max_pos] = max(a2_copy);

  a2_copy(max_pos) = 0;
  test_wrong_by = 0;
```

```
sum_winners = 0;
while max_pos ~= dip_num
  sum_winners = sum_winners + a2_copy(max_pos);
  [max_val max_pos] = max(a2_copy);
  a2_copy(max_pos) = 0;
  test_wrong_by = test_wrong_by +1;
end

if (test_wrong_by < best_rank)
  best_rank = test_wrong_by;
  best_out = a2(dip_num) - sum_winners;
  best_pos = win_pos;
  best_p = p;
elseif (test_wrong_by == best_rank)&((a2(dip_num) - sum_winners) > best_out)
  best_out = a2(dip_num) - sum_winners;
  best_pos = win_pos;
  best_p = p;
end

end


                    %sleep for niceness seconds if during work hours
tv = fix(clock);
if (niceness > 0) & (tv(4) > 6) & (tv(4) < 15)
      pause(niceness);
end

                    % look for best window position in vector b
for win_pos = win_scan_start:win_scan_stop

                    % sequence each column from matrix into vector
  p = vectoriz(feature_surf_b,win_width,win_pos);
  ste=p(ste_start:input_size);   % this is patch to convert STPwr to RMS Amplitude
  p(ste_start:input_size)=sign(ste).*sqrt(abs(ste));

  a1 = feval(f1,w1*p,b1);
  a2 = feval(f2,w2*a1,b2);

  a2_copy = a2;
  [max_val max_pos] = max(a2_copy);

  a2_copy(max_pos) = 0;
  test_wrong_by = 0;
  sum_winners = 0;
  while max_pos ~= dip_num
    sum_winners = sum_winners + a2_copy(max_pos);
    [max_val max_pos] = max(a2_copy);
    a2_copy(max_pos) = 0;
    test_wrong_by = test_wrong_by +1;
  end

  if (test_wrong_by < best_rank)
    best_rank = test_wrong_by;
    best_out = a2(dip_num) - sum_winners;
    best_pos = win_pos + win_slack;
    best_p = p;
```

144

```matlab
      elseif (test_wrong_by == best_rank)&((a2(dip_num) - sum_winners) > best_out)
        best_out = a2(dip_num) - sum_winners;
        best_pos = win_pos + win_slack;
        best_p = p;
      end

   end
                        %sleep for niceness seconds if during work hours
   tv = fix(clock);
   if (niceness > 0) & (tv(4) > 6) & (tv(4) < 15)
        pause(niceness);
   end

   test_offsets(test_file_num,1) = best_pos;    % save window ofset for next epoch
   p = best_p;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PRESENTATION PHASE - Determine outputs of layers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   a1 = feval(f1,w1*p,b1);
   a2 = feval(f2,w2*a1,b2);
   e = t - a2;
   test_SSE = test_SSE + sumsqr(e);

                        %sleep for niceness seconds if during work hours
   tv = fix(clock);
   if (niceness > 0) & (tv(4) > 6) & (tv(4) < 15)
      pause(niceness);
   end


   if disp_files == 1           % display progress
      str =strcat(strcat(num2str(epoch_count),' = epoch.  SSE = '),num2str(sumsqr(e)));
   else
      str = ' ';
   end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % record whether NNET output was correct or way off
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   [max_val max_pos] = max(a2);
   test_record(test_file_num,:) = zeros(1,5);
   test_record(test_file_num,1) = dip_num;
   test_record(test_file_num,2) = a2(dip_num);
   test_record(test_file_num,4) = max_pos;
   test_record(test_file_num,5) = max_val;

   test_dip_record(dip_num,1) = test_dip_record(dip_num,1) + 1;
   if max_pos ~= dip_num
      a2_copy = a2;
      a2_copy(max_pos) = 0;
      test_wrong_by = 0;
```

145

```
        while max_pos ~= dip_num
          [max_val max_pos] = max(a2_copy);
          a2_copy(max_pos) = 0;
          test_wrong_by = test_wrong_by +1;
        end
        test_record(test_file_num,3) = test_wrong_by;
        str = strcat(strcat(strcat(str,' (wrong by '),num2str(test_wrong_by)),')');
        if test_wrong_by > 2
          test_wrong_by_alot = test_wrong_by_alot+1;
          test_dip_record(dip_num,3) = test_dip_record(dip_num,3) + 1;
        end
      else
        test_record(test_file_num,3) = 0;
        test_correct = test_correct+1;
        test_dip_record(dip_num,2) = test_dip_record(dip_num,2) + 1;
      end
      if disp_files == 1
        str = strcat(str,strcat(' @',num2str(best_pos)));
        disp(str);
        disp(test_record(test_file_num,:));
      end

      v = test_record(test_file_num,:);
      o = test_offsets(test_file_num,1);

      fprintf(report_file,'%s, %f, %d, %s, %f,%d\n',
              dip_names(v(1),:),v(2),v(3),dip_names(v(4),:),v(5),o);

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% end of while not eof test_file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    fclose(test_file);

end        % end of "if sep_test_data..."

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% record epoch stats in report file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  report_file = fopen(report_filename,'at+');
  fprintf(report_file,'\n%d, TRAIN,%f,%d,%d,',epoch_count,SSE,correct,wrong_by_alot);
  if sep_test_data
    fprintf(report_file,'  TEST,%f,%d,%d, %s\n',test_SSE,test_correct,test_wrong_by_alot,str);
  else
    fprintf(report_file,' %s\n',str);
  end
  fclose(report_file);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write trailer to report file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
report_file = fopen(report_filename,'at+');
tv = fix(clock);
```

146

```
time_stamp = strcat('Stop Time = ',num2str(tv(1)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(2)));
time_stamp = strcat(time_stamp,'/');
time_stamp = strcat(time_stamp,num2str(tv(3)));
time_stamp = strcat(time_stamp,' ');
time_stamp = strcat(time_stamp,num2str(tv(4)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(5)));
time_stamp = strcat(time_stamp,':');
time_stamp = strcat(time_stamp,num2str(tv(6)));
fprintf(report_file,'\n%s\n',time_stamp);
if gigaflops > 0
  fprintf(report_file,'%f giga-flops\n',gigaflops);
else
  fprintf(report_file,'%d flops\n',flops);
end
fprintf(report_file,'%s  %f\n',str,SSE);
fprintf(report_file,'\n%s\n\n','END OF NNET TEST RECORD');
fclose(report_file);

save testrec

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

## G. MAKVECTS.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MAKVECTS.M  -  MatLab script file for creating precomputed feature
%              vector files for use by TRNBPX.
%
% Written by Major M.E. Cantrell, USMC
%
% Last Modified:
%
%      7 June 1996
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
clc;

ch = input('Use frequency normalization? ','s');
if upper(ch(1)) == 'N'
  freq_norm = 0;
  out_ext = '.BAD';
else
  freq_norm = 1;
  out_ext = '.vec';
end


ch = input('Two versions? ','s');
if upper(ch(1)) == 'N'
  two_ver = 0;
else
  two_ver = 1;
end

niceness_setting = input('Niceness setting (will sleep that many seconds per file)? ');

save nice.txt niceness_setting -ascii
niceness = niceness_setting;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

speech_data = [];     % vector with entire speech file's contents
sampsize = [];        % size of above

fname = [];     % will be string with filename
pname = [];     % will be string with pathname
fpname = [];    % will be string with filename + pathname
varname = [];   % will be string with filename less extension

last_pname = [];

vect_count = 0;
no_pitch_count = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
                    % get file list filename from user

[lfname,lpname] = uigetfile('*.*','Filename Containing List of Voice Files');
if lfname==0
  return
end
lfpname=strcat(lpname,lfname);
ldot_loc=findstr(lfname,'.');
if length(ldot_loc)>0
  lvarname=lfname(1:ldot_loc-1);
  lfname_ext=upper(lfname(ldot_loc+1:length(lfname)));
else
  lvarname=lfname;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

list_file = fopen(lfpname);
badmat_file = fopen('badmats.dir','wt');
report_file = fopen('makevrpt.txt','wt');
fprintf(report_file,'FOLLOWING DIPHONES WERE UNVOICED:\n\n');

tic;         % start timer

line_read = fgetl(list_file);
while (~feof(list_file)) & (length(line_read)>0)   % for every filename listed in list file

                    %sleep for niceness seconds if during work hours
  tv = fix(clock);
  if (niceness > 0) & (tv(4) > 6) & (tv(4) < 18)
    pause(niceness);
  end

  disp('---------------------------------------------------------------');

  disp(fix(clock));

  disp(line_read);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  dot_loc=findstr(line_read,'.');  % pick out pathname from line read
  fpname_end=dot_loc+3;
  fpname = line_read(1:fpname_end);
  i = dot_loc-1;
  while (i > 0) &
(((fpname(i)>='A')&(fpname(i)<='Z'))|((fpname(i)>='a')&(fpname(i)<='z'))|((fpname(i)>='0')&(fpname(i)<='9
')))
    i=i-1;
  end
  pname=fpname(1:i);
  varname=fpname(i+1:dot_loc-1);

  disp (strcat('Pathname is ',pname));
```

149

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                % pick out diphone name
dipname_start = fpname_end + 2;
i = dipname_start;
while line_read(i) ~= ' '
  i=i+1;
end
dipname_end = i - 1;
dipname = upper(line_read(dipname_start:dipname_end));

disp(strcat('Diphone is ',dipname));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                % get location of this diphone's phoneme junction

dip_limits = sscanf(line_read(dipname_end+1:length(line_read)),'%d %d %d');
dip_junction = dip_limits(2);

disp(strcat('Diphone junction is: ',num2str(dip_junction)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(strcat('Loading ',fpname));

speech_data = ld16bit(fpname);
speech_data=speech_data';
speech_data=speech_data(513:length(speech_data));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                % create feature "vectors"
disp('Computing feature vectors...');

if freq_norm == 1
  [feature_surf,median_pitch,formants_a] = features(speech_data,dip_junction);
  feature_surf_a = diff_mat(feature_surf);
  if two_ver == 1
    [feature_surf,median_pitch_2,formants_b] = features(speech_data,dip_junction-170);
    feature_surf_b = diff_mat(feature_surf);
  end
else
  [feature_surf,median_pitch,formants_a] = featureb(speech_data,dip_junction);
  feature_surf_a = diff_mat(feature_surf);
  if two_ver == 1
    [feature_surf,median_pitch_2,formants_b] = featureb(speech_data,dip_junction-170);
    feature_surf_b = diff_mat(feature_surf);
  end
end


if median_pitch == 0
  median_pitch = median_pitch_2;
end
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```matlab
                % save feature "vectors" to a file with same path
filenum = 0;
file_exists = 1;
while file_exists
  out_fpname = strcat(pname,dipname);
  out_fpname = strcat(out_fpname,num2str(filenum));
  out_fpname = lower(strcat(out_fpname,out_ext));
  file_exists = exist(out_fpname);
  filenum = filenum + 1;
end

disp(strcat('Median pitch is ',num2str(median_pitch)));

if median_pitch <= 0
  no_pitch_count = no_pitch_count + 1;
  fprintf(report_file,'%s\n',line_read);
  fprintf(badmat_file,'%s\n',out_fpname);
end

vect_count = vect_count + 1;
disp(strcat('Vector count now ',num2str(vect_count)));
disp(vect_count);

disp(strcat('Saving features to ',out_fpname));

if two_ver == 1
  eval(['save ' out_fpname ' feature_surf_a feature_surf_b median_pitch']);
else
  eval(['save ' out_fpname ' feature_surf_a median_pitch']);
end

clear speech_data;        % clean up workspace before next phoneme

line_read = fgetl(list_file);     % get next filename from list file

end     % while not eof(list_file)

fclose(list_file);
fclose(report_file);
fclose(badmat_file);

disp('Total elapsed time:');

toc;          % display total elapsed time in seconds

pack;
```

151

## H. INPUTS.M

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUTS - Get training parameters from user for use by trnbpx.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

revision = '1.0';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GET USER DEFINABLE PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

remarks2 = ' ';

me = input('Maximum total epochs? ');

mc = input('Momentum Constant (0.90 to .95 is standard)? ');

er = input('Allowable Error Ratio (1.04 to 1.05 is standard)? ');

eg = input('Error goal per output (0.02 is standard)? ');

ch = input('Reset weights to random values? ','s');
if upper(ch(1)) == 'Y'
  hidden_size = input('How many neurons in hidden layer? ');
  win_width = input('Width of recognition window (13 standard)? ');
  reset_weights = 1;
  remarks2 = 'New Weights, ';
else
  reset_weights = 0;
  remarks2 = 'Same Weights, ';
end

ch = input('Train on first instance of each diphone only? ','s');
if upper(ch(1)) == 'Y'
  single_instance_only = 1;
  remarks2 = strcat(remarks2,'First Instance Only, ');
else
  single_instance_only = 0;
  remarks2 = strcat(remarks2,'Entire File Set, ');
end

ch = input('Batch weight changes? ','s');
if upper(ch(1)) == 'Y'
  batch = 1;
else
  batch = 0;
  remarks2 = strcat(remarks2,'NO BATCHING, ');
end

if (single_instance_only == 0)
  ch = input('Time-Align Feature Vectors? ','s');
  if upper(ch(1)) == 'Y'
    time_align = 1;
    remarks2 = strcat(remarks2,'Time-Aligning, ');
    ch = input('Use static file offsets? ','s');
```

```matlab
  if upper(ch(1)) == 'Y'
    static_offsets = 1;
    if exist('trnoffs.mat')
      offname = 'trnoffs';
    else
      ls
      offname = input('What is name of matlab file with offsets (only)? ','s');
    end
  else
    static_offsets = 0;
    offname = 'trnoffs';
  end
  max_win_offset = input('Maximum offset from center for recognition window? ');
else
  time_align = 0;
  remarks2 = strcat(remarks2,'Not Time-Aligning, ');
  static_offsets = 0;
  offname = 'trnoffs';
end
ch = input('Separate test data? ','s');
if upper(ch(1)) == 'Y'
  sep_test_data = 1;
  remarks2 = strcat(remarks2,'Separate Test Set,');
  if time_align == 0
    max_win_offset = input('Maximum offset from center for test files? ');
  end
else
  sep_test_data = 0;
  remarks2 = strcat(remarks2,'No Separate Test Set,');
end
else
  sep_test_data = 0;
  time_align = 0;
  static_offsets = 0;
  max_win_offset = 1;
end

remarks = input('Remarks for record? ','s');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
niceness_setting = input('Niceness setting (will sleep 9 times that many seconds per file)? ');

save nice.txt niceness_setting -ascii
niceness = niceness_setting;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% determine operating system in use
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if findstr('\',pwd) == []
  op_sys = 'unix';
else
  op_sys = 'dos ';
end
```

153

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get diphone list filename from user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lpname = pwd;
if exist('diplist.txt')
  lfname = 'diplist.txt';
else
  ls
  lfname = input('What is name of diphone list file? ','s');
end

if op_sys == 'unix'
  lpname = strcat(lpname,'/');
else
  lpname = strcat(lpname,'\');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get training file list filename from user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mpname = pwd;
ls
mfname = input('What is name of training file list? ','s');
if op_sys == 'unix'
  mpname = strcat(mpname,'/');
else
  mpname = strcat(mpname,'\');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get test file list filename from user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (sep_test_data == 1)
  tpname = pwd;
  ls
  tfname = input('What is name of test file list? ','s');
  if op_sys == 'unix'
    tpname = strcat(tpname,'/');
  else
    tpname = strcat(tpname,'\');
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% save inputs to a mat file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

save inputs
```

154

# I.    P_AMDF.M

```
function [pitch] = p_amdf(speech_data,fs,n)
%P_AMDF Voice Pitch using Average Magnitude Difference Function
%
%       This function computes the pitch of a voice signal (first argument).
%       The sample frequency is required as the second argument and the third
%       must be the AMDF window size.  Because of the nature of the algorithm, the
%       speech vector must be longer than the window size by 1+((1/50)/(1/fs)) or
%       321 at 16 KHz.  This version is specifically intended for voice signals.
%       So it will only detect pitches in the range 50 to 500 Hz.  Note that
%       accuracy suffers near the end of a voiced segment of speech.  Also, this
%       func can be slow.  Try a smaller N to speed things up.  You'll probably
%       get the same answer as you would with a larger N (this N is not related
%       to the N used with FFT; it only determines how much of speech segment
%       to use for pitch determination).
%
%       This code is based on [Ainsworth p96].  See thesis for complete reference
%       information.
%
%
%USAGE EXAMPLE:        pitch = p_amdf(data(1:n+321),fs,n);
%


    if ~(nargin == 3)
        error('Invalid number of arguments to P_AMDF.M');
    end

    max_lag = 1+(1/50)/(1/fs);  % valid pitch range 50-500 Hz
    min_lag = 1+(1/500)/(1/fs);

    last=length(speech_data);

    if n+max_lag <= last       % can't do it if too near end file
        lag_array = zeros(1,max_lag);      % only need this for plot of lags
        best_lag_sum=inf;   % initialize big so anything looks better
        for lag = min_lag:1:max_lag % chg of 1 in step size = about 15 Hz
                                    % (at pitch=500). step 5 = 68 Hz (too big).


            lag_sum=sum(abs(speech_data(1:n)-speech_data(lag+1:n+lag)));

            if lag_sum <= best_lag_sum      % remember best; if tie use lower Hz
                best_lag_sum=lag_sum;
                best_lag=lag;
            end
            lag_array(lag) = lag_sum;       % only needed for lag plot
        end
        pitch=1/((best_lag-1)*(1/fs));      % compute pitch from best lag
    else
        error('Sample too small.  Need at least N + 1 + ((1/50)/(1/fs)) samples.');
    end
```

## J.    P_CEPS.M

```
function [pitch , confidence] = p_ceps(ceps_data,fs,n,L)
%P_CEPS Voice Pitch using Cepstral Peak
%
%      This function computes the pitch of a voice signal using cepstral
%      coefficients (first argument).  The sample frequency is required as
%      the second argument, the third must be the window size, and the fourth
%      is the cutoff between the pitch peak and the formant peak.  If requested,
%      a number indicating the prominence of the pitch peak is also returned
%      (one tenth the ratio of the peak's height to mean amplitude of the
%      surrounding plot).  If this value is less than one, a pitch value of zero
%      is returned (unless the confidence number is also requested).  The second
%      returned value may be used as an indicator of the degree of voicing, as
%      well as an indication of the reliability of the pitch figure.  Note that
%      the window size N and Fs must be the same as those used to produce the
%      cepstrum.  Because this function is intended for speech processing, a
%      zero will be returned unless the computed pitch is between 50 and 500 Hz.
%
%
%USAGE EXAMPLE:      cep_vector=cepstrum(data(1:n),fs,hamming(n));
%                    [pitch confidence]=p_ceps(cep_vector,fs,n,round(n/10));
%

if ~(nargin == 4)
  error('Invalid number of arguments in P_CEPS');
end
if length(fs)==0
  error('Invalid sample frequency in P_CEPS');
end
if length(ceps_data) == n/2
  rceps=abs(ceps_data);
elseif length(ceps_data) == n
  rceps=(abs(ceps_data(1:n/2)));
else
  error('Window size and cepstrum vector size must agree');
end

confid = max(rceps(L:length(rceps)))/(10*mean(rceps(L:length(rceps))));

                                  % determine pitch (if strongly voiced or user asked for confid)
if (confid >= 1)|(nargout==2)    % Try using cepstral peak to determine pitch
    [peak_val,peak_loc] = max(rcep(L:length(rcep)));
    cep_pitch=fs/(L+peak_loc);
    if (cep_pitch<500)&(cep_pitch>50)  % only return pitch if in reasonable range
      pitch=cep_pitch;
    else
      pitch=0;
    end
else
    pitch=0;
end

if nargout==2
  confidence=confid;
end
```

156

# K.   VECTORIZ.M

```
function [p] = vectoriz(feature_surf,win_width,win_pos)
%VECTORIZ      sequence each column from feature matrix into a vector
%
% USAGE:       input_vect = vectoriz(in_mat,win_width,win_pos)
%

%for i = 1:30, sequence each column from matrix into vector
%  this function does not use for loop because MATLAB for loops are SLOW

p=zeros(win_width*30,1);

p(1*win_width-win_width+1:1*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,1);
p(2*win_width-win_width+1:2*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,2);
p(3*win_width-win_width+1:3*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,3);
p(4*win_width-win_width+1:4*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,4);
p(5*win_width-win_width+1:5*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,5);
p(6*win_width-win_width+1:6*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,6);
p(7*win_width-win_width+1:7*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,7);
p(8*win_width-win_width+1:8*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,8);
p(9*win_width-win_width+1:9*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,9);
p(10*win_width-win_width+1:10*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,10);

p(11*win_width-win_width+1:11*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,11);
p(12*win_width-win_width+1:12*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,12);
p(13*win_width-win_width+1:13*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,13);
p(14*win_width-win_width+1:14*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,14);
p(15*win_width-win_width+1:15*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,15);
p(16*win_width-win_width+1:16*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,16);
p(17*win_width-win_width+1:17*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,17);
p(18*win_width-win_width+1:18*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,18);
p(19*win_width-win_width+1:19*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,19);
p(20*win_width-win_width+1:20*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,20);

p(21*win_width-win_width+1:21*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,21);
p(22*win_width-win_width+1:22*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,22);
p(23*win_width-win_width+1:23*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,23);
p(24*win_width-win_width+1:24*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,24);
p(25*win_width-win_width+1:25*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,25);
p(26*win_width-win_width+1:26*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,26);
p(27*win_width-win_width+1:27*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,27);
p(28*win_width-win_width+1:28*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,28);
p(29*win_width-win_width+1:29*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,29);
p(30*win_width-win_width+1:30*win_width,1) = feature_surf(win_pos:win_pos+win_width-1,30);
```

## L. DIPNUMOF.M

```
function[p] = dipnumof(dip_name,dip_names)
%DIPNUMOF    Return the index position of a diphone name in an array
%           of diphone names. Uses binary search. Requires LESSTHAN.M.
%
%    Usage: dip_num = dipnumof(dipname,dipnames);

   num_dips=length(dip_names);
   i = 1;
   k = num_dips;
   found = 0;

   while (i <= k)&(found ~= 1)
    j = round((i+k)/2);
    if strcmp(dip_names(j,:),dip_name)==1
       found=1;
    elseif lessthan(dip_name,dip_names(j,:))==1
       k = j-1;
    else
       i = j+1;
    end
   end

   p = j;
```

## M. DIFF_MAT.M

```
function [out_matrix] = diff_mat(in_mat)
%DIF_MAT      Return a difference matrix for specified input matrix.
%
%    Row i of the output matrix is the difference between rows i-1 and i
%    of the input matrix. The resulting surface represents the
%    rate of change of the function (assuming row number increases with
%    time). Note that the resulting matrix will have one less row than
%    input matrix.
%
%
%
%USAGE EXAMPLE:     delta_surface = diff_mat(data(1:maxrow,1:maxcol));
%

num_rows_in = size(in_mat,1);
num_cols_in = size(in_mat,2);

result_mat = zeros(num_rows_in-1,num_cols_in);

result_mat = (in_mat(2:num_rows_in,:)-in_mat(1:num_rows_in-1,:));

out_matrix = result_mat;
```

## N.  LD16BIT.M

```
function [y] = ld16bit(name,number,offset)
%LD16BIT  Load signed 16-bit binary data file.
%       [Y] = LD16BIT('NAME') load the entire file 'NAME' as 16-bit
%       signed data samples.
%
%       [Y] = LD16BIT('NAME',NUMBER) loads the first NUMBER
%       samples starting from the beginning of the file.
%
%       [Y] = LD16BIT('NAME',NUMBER,OFFSET) loads NUMBER samples
%       beginning at OFFSET samples from the beginning of the
%       file. If NUMBER=0, loads starting from OFFSET to the EOF.
%
%       See also LD16BIT, LD32BIT, SV8BIT, SV16BIT, SV32BIT

%       LT Dennis W. Brown 7-17-93, DWB 1-23-94
%       Naval Postgraduate School, Monterey, CA
%       May be freely distributed.
%       Not for use in commercial products.

% default output
y = [];
% name must be a string
if ~isstr(name),
        error('ld16bit: Filename must be a string argument!');
end;
% open the file
fid = fopen(name,'rb');
if fid == (-1)
        error(['ld16bit: Could not open file ' name]);
end
% get file length
fseek(fid,0,'eof');
flength = ftell(fid);
fseek(fid,0,'bof');
if nargin == 1
        % read the entire file
        y = fread(fid,flength/2,'int16');
elseif nargin == 2
    % read some of the samples beginning at the start of file
        % error check
        if number > flength/2,
                fclose(fid);
                error('ld16bit: Requested number of samples greater than filelength!');
        end;
        % read samples
        fseek(fid,0,'bof');
        y = fread(fid,number,'int16');
elseif nargin == 3,
    % read some of the samples beginning somewhere in the file
        % error check (offset sample is included in number)
        if number+offset-1 > flength/2,
                fclose(fid);
                error('ld16bit: Requested number+offset of samples greater than filelength!');
        end;
```

159

```
        if number == 0,
                % read to end-of-file
                fseek(fid,2*(offset-1),'bof');
                y = fread(fid,flength/2-offset+1,'int16');
        else
                % read specified number of samples
                fseek(fid,2*(offset-1),'bof');
                y = fread(fid,number,'int16');
        end;
    end;
    fclose(fid);
```

## O.   LESSTHAN.M

```
function[r] = lessthan(str1,str2)
%LESSTHAN    Return 1 if diphone name in str1 is alphabetically
%            less than diphone name in str2.
%            STRINGS MUST BE 7 CHARACTERS LONG!
%
%   Usage:  if lessthan(dip_names(i,1:7),dip_names(j,1:7)) == 1; break; end
%

colvals = [10000 1000 100 10 1 0.1 0.01];  % used to convert MATLAB's "<"
                                           % result (a vector) to a alpha
                                           % comparison.  Crude, but it works
                                           % (for 7 char strings anyway).

r = sum(colvals.*(str1 < str2)) > sum(colvals.*(str1 > str2));
```

## P.   STRCAT.M

```
function [out_str] = strcat(left,right)
%STRCAT       Append two strings.
%
% USAGE:       answer_str = strcat('Answer is ',int2str(result));
%

    left_len=length(left);
    right_len=length(right);
    new_last=left_len+right_len;
    out_str = blanks(new_last);
    out_str(1:left_len)=left(1:left_len);
    out_str(left_len+1:new_last)=right(1:right_len);
```

# APPENDIX B. ADA CODE

## A. CONTENTS

### 1. DIPCOUNT.ADA

This utility counts the number of occurrences of each diphone in a phonetic dictionary.

### 2. DIPDURS.ADA

This utility gathers information on the maximum durations of diphones in a list of phonetic transcription files.

### 3. DIPNUMS.ADA

This utility counts the number of occurrences of diphones in the phonetic transcription files. This tells how many exemplars are available for training and testing.

### 4. MAKFLIST.ADA

This utility produces a list containing the requested number of speech (.WAV) files containing the designated diphones. The output is used by MAKVECTS.M.

### 5. VOCAB.ADA

This utility determines the vocabulary that can be built from a given list of diphones (for a particular phonetic dictionary).

### 6. MAKWLIST.ADA

This program produces a list of speech (.WAV) files, and the words they contain, for use by RECOGNIZ.M (in recognizing diphones prior to word tests).

### 7. PHNDURS.ADA

This program gathers statistics on phoneme durations from phonetic transcription files.

## B. USING THE UTILITIES

The utilities in the appendix were used as follows:

1. A directory listing of all TIMIT SX phonetic transcription (.PHN) files was redirected to a text file (eg. by typing "dir /b /s *.phn > phnfiles.txt" at the DOS command prompt or "find" at the UNIX prompt). The list of transcription files was used by DIPNUMS.ADA to create a list of all diphones found in the PHN files. The output file was in the following format (the numbers refer to start, phoneme junction, and ending sample number):                          e:\timit\train\dr1\falk0\sx132.wav b-r 0 2200 4300

2. The list produced by the preceding step was sorted by an arbitrary column (one of the sample number fields) in order to scramble the speech files. Otherwise, speakers from the first few dialect regions would have dominated the training and test files selected in the next step (because they are early in the directory listing).

3. The shuffled file list from the preceding step was used by MAKFLIST.ADA to produce a final list of the speech files to be used. This utility also required a file (produced by DIPNUMS.ADA) listing the diphones to be included in the data set .

4. The file produced by MAKFLIST.ADA was used by a MATLAB script file (MAKVECTS.M) to locate and compute feature vectors from the actual TIMIT speech files.

5. Both DIPNUMS.ADA and MAKFLIST.ADA require an integer prescribing how many occurrences of each diphone to include.

6. Lists of the resulting feature vector files (again, produced by redirecting directory listings) were used by TRNBPX.M and TESTNET to locate the feature vector files.

7. After the network was trained, VOCAB.ADA was used to produce a list of words in the vocabulary achievable DIPLIST.TXT.

8. The vocabulary list produced by VOCAB.ADA was used by MAKWLIST.ADA to build a test set (list of .WAV files and their word locations) for use by RECOGNIZ.M.

162

## C. DIPCOUNT.ADA

```
--****************************************************************
--****************************************************************
--** PROGRAM NAME: DipCount  (DipCount.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a phonetic dictionary & counting
--** diphones.
--**
--** DATE MODIFIED: 3 July, 1995  (see below for version number)
--**
--****************************************************************
--****************************************************************

with Arg,
     Text_IO;

use Text_IO;

procedure DipCount is

  Max_Str_Len : constant := 255;
  Max_Phn_Len : constant := 5;
  Col_Widths : constant := 6;

  Usage_Error : exception;

  Type STR is record
    Txt : String(1..Max_Str_Len);
    Len : Integer range 0..Max_Str_Len := 0;
  end record;

  Temp,Tail,Atom,Atom_Str,Tail_Str,Phon_Str : STR;

  Type Phoneme is (AA,AE,AH,AO,AW,AY,B,CH,D,DH,EH,ER,EY, -- 1st 3 lines from CMU lex
                   F,G,HH,IH,IY,JH,K,L,M,N,NG,OW,OY,P,R,
                   S,SH,T,TH,UH,UW,V,W,Y,Z,ZH,
                   EM,EN,ENG,EL,AX,IX,AXR,HX,         -- this line from TIMIT
                   XXX);                  -- use XXX for unknown codes

  Diphone_Matrix : array(Phoneme,Phoneme) of natural :=
                                                  (others => (others => 0));

  This_Phoneme,Last_Phoneme : Phoneme;

  Debugging : boolean := false;

  Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
  Name_Last : Natural;

  Input_Line : string(1..Max_Str_Len);
  Input_Last : natural;
```

```
Version : constant String := "DipCount Version #1.0";

Input_DataFile,Output_DataFile : Text_IO.File_Type;

--*************************************************************************
--*  GET_ATOM - Gets next atom from a str
--*************************************************************************

Procedure Get_Atom(In_Str : Str;Atom_Str,Tail_Str : out Str) is
 i,a_start,a_end,a_len,t_len : integer := 1;
Begin
 If In_Str.Len > 0 then
   While (In_Str.Txt(i) = ' ') and (i < In_Str.Len) loop
       i := i + 1;
   end loop;
   If In_Str.Txt(i) /= ' ' then
       a_start := i;
       While (In_Str.Txt(i) /= ' ') and (i < In_Str.Len) loop
        i := i + 1;
       end loop;
       if In_Str.Txt(i) = ' ' then
        a_end := i - 1;
       else
        a_end := i;
       end if;
       a_len := a_end - a_start + 1;
       t_len := In_Str.Len - a_end;
       Tail_Str.Len := t_len;
       Atom_Str.Len := a_len;
       for i in 1..a_len loop
         Atom_Str.Txt(i) := In_Str.Txt(a_start + i - 1);
       end loop;
       for i in 1..t_len loop
         Tail_Str.Txt(i) := In_Str.Txt(a_end + i);
       end loop;
   else
       Atom_Str.Len := 0;
       Tail_Str.Len := 0;
   end if;
 else
   Atom_Str.Len := 0;
   Tail_Str.Len := 0;
 end if;
end Get_Atom;


--*************************************************************************
--* GET_PHONEME - Returns Phoneme corresponding to an atom (atom must be free
--* of leading & trailing blanks & other garbage (accent digits on end OK)
--*************************************************************************

Function Get_Phoneme (In_Atom : Str) return Phoneme is
 i,Phn_End : Integer;
 Phn_String : String (1..Max_Phn_Len);
 Out_Phoneme : Phoneme;
Begin
 Phn_End := In_Atom.Len;
```

```
if In_Atom.Len > 0 then
  While ((In_Atom.Txt(Phn_End) in '0'..'9')) and (Phn_End > 1) loop
      Phn_End := Phn_End - 1;
  end loop;
end if;
If (Phn_End > 0) and (Phn_End <= Max_Phn_Len) then
  Phn_String(1..Phn_End) := In_Atom.Txt(1..Phn_End);
  begin  -- block to catch constraint error if not a valid phoneme
      Out_Phoneme := Phoneme'Value(Phn_String(1..Phn_End));
      return Out_Phoneme;
  exception
      when constraint_error =>
        return XXX;
  end; -- block
else
  return XXX;
end if;
end Get_Phoneme;


--***********************************************************************
--* PUT_PHON - A Put routine that translates from my enum
--* representations to standard symbols
--***********************************************************************


Procedure Put_Phon(O_File : File_Type; Phon : Phoneme) is
begin
 If Phon = HX then
   Put(O_File,"H#");
 elsif Phon = XXX then
   Put(O_File,"???");
 else
   Put(O_File,Phoneme'image(Phon));
 end if;
end Put_Phon;


--***********************************************************************
--* DISPLAY_MATRIX - Prints final diphone matrix to output file (in 2 pieces)
--***********************************************************************


Procedure Display_Matrix is
 row,col,mid_col : Phoneme;
 Count : natural := 0;
begin

 Put_Line(Output_Datafile,"***********************************************************************************");
 Put_Line(Output_Datafile,"               DIPHONE COUNTS FOR FILE " & Input_Filename);
 Put_Line(Output_Datafile,"***********************************************************************************");


 mid_col := Phoneme'val(integer((Phoneme'pos(Phoneme'last))/2));

 new_line(Output_Datafile);


                                         -- left half of matrix
 for col in Phoneme'first..Mid_Col loop  -- column labels
   set_col(Output_Datafile,col_widths*(1 + Phoneme'pos(col)));
```

165

```
      Put_Phon(Output_Datafile,Col);
    end loop;
    new_line(Output_Datafile,2);
    for row in Diphone_Matrix'range loop
      Put_Phon(Output_Datafile,Row);        -- row label
      for col in Phoneme'first..Mid_Col loop
          set_col(Output_Datafile,col_widths*(1 + Phoneme'pos(col)));
          put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));

          if Diphone_Matrix(row,col) > 0 then  -- count nonzero entries
            count := count + 1;
          end if;

    end loop;
    new_line(Output_Datafile);
    end loop;
    new_line(Output_Datafile,2);


                                    -- right half of matrix
    for col in Phoneme'succ(Mid_Col)..Phoneme'last loop   -- col labels
      set_col(Output_Datafile,col_widths*(Phoneme'pos(col) - Phoneme'pos(Mid_Col)));
      Put_Phon(Output_Datafile,col);
    end loop;
    new_line(Output_Datafile,2);
    for row in Diphone_Matrix'range loop
      Put_Phon(Output_Datafile,Row);        -- row label
      for col in Phoneme'succ(Mid_Col)..Phoneme'last loop
          set_col(Output_Datafile,col_widths*(Phoneme'pos(col) - Phoneme'pos(Mid_Col)));
          put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));

          if Diphone_Matrix(row,col) > 0 then  -- count nonzero entries
            count := count + 1;
          end if;

    end loop;
    new_line(Output_Datafile);
    end loop;

    New_Line(Output_Datafile,3);
    Put_Line(Output_Datafile,"****************************************************************************************");
    Put_Line(Output_Datafile,"            Total Diphones with nonzero entries = " & integer'image(count));

    Put_Line(Output_Datafile,"****************************************************************************************");

    New_Line(Output_Datafile,3);

                                    -- Now list nonzero entries
    Put_Line(Output_Datafile,"NONZERO ENTRIES ARE:");
    New_Line(Output_Datafile);

    for row in Diphone_Matrix'range loop
      for col in Phoneme'first..Phoneme'last loop
          if Diphone_Matrix(row,col) > 0 then
            Put_Phon(Output_Datafile,Row);
            set_col(Output_Datafile,col_widths);
            Put_Phon(Output_Datafile,Col);
```

```ada
            set_col(Output_Datafile,(3*col_widths)-(integer'image(Diphone_Matrix(row,col))'length));
            put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));
            new_line(Output_Datafile);
          end if;
      end loop;
    end loop;

end Display_Matrix;

--*************************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--*************************************************************************

Procedure Get_File(Prompt_String : String;
                    F_Name : out String;
                    F_Mode : File_Mode;
                    F_Handle : in out File_Type) is
  Str_Last : Integer;
  Str_Text : String(1..Max_Str_Len) := (others => ' ');
begin


  loop    -- loop until valid filename entered or user aborts (^C)
    Begin   -- block statements to allow another try if bad filename entered

      Put(Prompt_String);
      Get_Line(Str_Text,Str_Last);


      if F_Mode = In_File then
        Text_IO.Open(File => F_Handle,
              Mode => F_Mode,
              Name => Str_Text(1..Str_Last));
      else
        Text_IO.Create(File => F_Handle,
              Mode => F_Mode,
              Name => Str_Text(1..Str_Last));
      end if;

      F_Name := Str_Text;

      exit;

    exception
      When Name_Error => Put(Ascii.Bel);
                    Put_Line("***  Unable to open " & Str_Text(1..Str_Last));
                    New_Line;
      When others => raise;
    end;
  end loop;

end Get_File;
```

```
--************************************************************************
--* DIPCOUNT Main Program
--************************************************************************

Begin    -- DipCount

  if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                               Arg.Data(3) = "/D" or
                               Arg.Data(3) = "-d" or
                               Arg.Data(3) = "-D" or
                               Arg.Data(3) = "d"  or
                               Arg.Data(3) = "D") then
    Debugging := True;
  elsif Arg.Count > 2 then  -- note that program name is 1st entry in array
    raise Usage_Error;
  end if;

  New_Line(25);           -- blank lines & program "banner"

  Put_Line(Version);
  New_Line(2);

  if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
    begin
      Text_IO.Open(File => Input_DataFile,
               Mode => In_File,
               Name => Arg.Data(2));
      Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
    exception            -- handle bad filenames from command line
    When Name_Error  |
      Status_error |
      Mode_error   |
      Use_error    |
      Device_error =>  New_Line(3);
                       Put_Line("*** Unable to open " & Arg.Data(2));
                       New_Line;
                       raise Usage_Error;
    end;
  else
    Get_File("Enter name of input file: ",
           Input_Filename,
           In_File,
           Input_Datafile);
  end if;

                              -- get output filename from user
  Get_File("Enter name of output file: ",
         Output_Filename,
         Out_File,
         Output_Datafile);


  If Debugging then
    New_Line;
    Put_Line("---------------------[ PROCESSING FILE ]----------------------------");
  end if;
```

```
While NOT End_Of_File(Input_DataFile) loop

  Get_Line(Input_DataFile,Input_Line,Input_Last);

                                            -- if not a comment line
  if Input_Last > 0 and then Input_Line(1) in 'A'..'z' then

    if debugging then
        Put_Line(Input_Line(1..Input_Last));
    end if;

    for i in 1..Input_Last loop        -- remove slashes. don't need them.
        if Input_Line(i) = '/' then
          Input_Line(i) := ' ';
        end if;
    end loop;

    Temp.Txt(1..Input_Last) := Input_Line(1..Input_Last);
    Temp.Len := Input_Last;
    Get_Atom(Temp,Atom,Tail); -- get & discard the word
    if Debugging then
        Put_Line("Word = " & Atom.Txt(1..Atom.Len));
    end if;


    Last_Phoneme := HX;  -- Consider silence & first phoneme to be a diphone

    loop            -- for each phoneme on the line (in the word)
        Temp := Tail;
        Get_Atom(Temp,Atom,Tail);
        if Atom.Len > 0 then
          This_Phoneme := Get_Phoneme(Atom);

          if Debugging then
            Put_Line(" Diphone = " &
                  Phoneme'Image(Last_Phoneme) & "/" &
                  Phoneme'Image(This_Phoneme));
          end if;

                  -- increment counter for diphone formed by this & prior phoneme
          Diphone_Matrix(Last_Phoneme,This_Phoneme) :=
            Diphone_Matrix(Last_Phoneme,This_Phoneme) + 1;

          Last_Phoneme := This_Phoneme;
        end if;
        exit when Tail.Len <= 0;
    end loop;


    if Debugging then
        Put_Line(" Diphone = " &
          Phoneme'Image(Last_Phoneme) & "/" &
          Phoneme'Image(HX));
    end if;
```

```
                    -- Consider last phoneme & silence to be a diphone
        Diphone_Matrix(Last_Phoneme,HX) :=
              Diphone_Matrix(Last_Phoneme,HX) + 1;

    end if;

  end loop;

  Display_Matrix;

  If Debugging then
    Put_Line("------------------------[ DONE ]-----------------------------");
    New_Line;
  end if;

  Text_IO.Close(Input_Datafile);
  Text_IO.Close(Output_Datafile);


exception

  When Usage_Error =>
    Put(Ascii.Bel);
    New_Line(3);
    Put_Line("Usage:  DipCount infilename [-d]");
    New_Line;
    Put_Line("        -d = enable debugging");
    New_Line;
    return;


end DipCount;
```

# D.   DIPDURS.ADA

```
--**********************************************************************
--**********************************************************************
--** PROGRAM NAME: DipDurs  (DipDurs.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a list of phonetic transcription
--** files & recording max diphone durations listed
--**
--** DATE MODIFIED: 17 Sep, 1995  (see below for version number)
--**
--**********************************************************************
--**********************************************************************

with Arg,
    Text_IO;

use Text_IO;

procedure DipDurs is

  Max_Str_Len : constant := 255;
  Max_Phn_Len : constant := 5;
  Col_Widths : constant := 6;
  Fs : constant := 16000.0;
  FList_Filename : constant string := "DIPFLIST.TXT";

  Usage_Error : exception;

  Type STR is record
    Txt : String(1..Max_Str_Len);
    Len : Integer range 0..Max_Str_Len := 0;
  end record;

  Temp,Tail,Atom,Atom_Str,Tail_Str,Phon_Str : STR;

  Type Phoneme is (AA,AE,AH,AO,AW,AY,B,CH,D,DH,EH,ER,EY, -- 1st 3 lines from CMU lex
                F,G,HH,IH,IY,JH,K,L,M,N,NG,OW,OY,P,R,
                S,SH,T,TH,UH,UW,V,W,Y,Z,ZH,
                EM,EN,ENG,EL,AX,IX,AXR,HX,        -- this line from TIMIT
                XXX);                             -- use XXX for unknown codes

  Diphone_Matrix : array(Phoneme,Phoneme) of natural :=
                                            (others => (others => 0));

  Max_Dip_X,Max_Dip_Y : Phoneme := HX;
  Max_Duration : Integer := 0;
  Max_Fname : Str;

  Debugging : boolean := false;

  Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
  Name_Last : Natural;
```

171

```
Input_Line : string(1..Max_Str_Len);
Input_Last : natural;

Version : constant String := "DipDurs Version #1.0";

Input_DataFile,Output_DataFile,FList_File : Text_IO.File_Type;

--*************************************************************************
--* UPPER_CASE - Returns upper case version of input string
--*************************************************************************
Function  Upper_Case(In_String : string) return string is
  Result_String : String(In_String'first..In_String'last);
Begin
  for index in In_String'first..In_String'last loop
    If In_String(index) in 'a'..'z' then
  Result_String(index) := character'val(character'pos(In_String(index))
                  - character'pos('a')
                  + character'pos('A'));
    else
  Result_String(index) := In_String(index);
    end if;

  end loop;
  return Result_String;
end Upper_Case;


--*************************************************************************
--*  GET_ATOM - Gets next atom from a str
--*************************************************************************

Procedure Get_Atom(In_Str : Str;Atom_Str,Tail_Str : out Str) is
  i,a_start,a_end,a_len,t_len : integer := 1;
Begin
  If In_Str.Len > 0 then
    While (In_Str.Txt(i) = ' ') and (i < In_Str.Len) loop
        i := i + 1;
    end loop;
    If In_Str.Txt(i) /= ' ' then
        a_start := i;
        While (In_Str.Txt(i) /= ' ') and (i < In_Str.Len) loop
          i := i + 1;
        end loop;
        if In_Str.Txt(i) = ' ' then
          a_end := i - 1;
        else
          a_end := i;
        end if;
        a_len := a_end - a_start + 1;
        t_len := In_Str.Len - a_end;
        Tail_Str.Len := t_len;
        Atom_Str.Len := a_len;
        for i in 1..a_len loop
          Atom_Str.Txt(i) := In_Str.Txt(a_start + i - 1);
        end loop;
        for i in 1..t_len loop
```

172

```
                    Tail_Str.Txt(i) := In_Str.Txt(a_end + i);
                end loop;
            else
                Atom_Str.Len := 0;
                Tail_Str.Len := 0;
            end if;
        else
            Atom_Str.Len := 0;
            Tail_Str.Len := 0;
        end if;
    end Get_Atom;


--*********************************************************************
--* GET_PHONEME - Returns Phoneme corresponding to an atom (atom must be free
--* of leading & trailing blanks & other garbage (accent digits on end OK)
--*********************************************************************

    Function Get_Phoneme (In_Atom : Str) return Phoneme is
        i,Phn_End : Integer;
        Phn_String : String (1..Max_Phn_Len);
        Out_Phoneme : Phoneme;
    Begin
        Phn_End := In_Atom.Len;
        if In_Atom.Len > 0 then
            While ((In_Atom.Txt(Phn_End) in '0'..'9')) and (Phn_End > 1) loop
                Phn_End := Phn_End - 1;
            end loop;
        end if;
        If (Phn_End > 0) and (Phn_End <= Max_Phn_Len) then
            Phn_String(1..Phn_End) := Upper_Case(In_Atom.Txt(1..Phn_End));

            begin  -- block to catch constraint error if not a valid phoneme
                Out_Phoneme := Phoneme'Value(Phn_String(1..Phn_End));
                return Out_Phoneme;
            exception
                when constraint_error =>
                    if Phn_End = 2 and then Phn_String(1..2) = "H#" then
                        return HX;
                    else
                        return XXX;
                    end if;
            end;  -- block

        else
            return XXX;
        end if;
    end Get_Phoneme;


--*********************************************************************
--* ECHO_DATA - Used by Process_Phn_File to echo the data read from file
--*********************************************************************

    Procedure Echo_Data (Start_N, End_N : Long_Integer;
                         Phon_Str : Str;
                         Phoneme_Read : Phoneme) is
```

173

```
begin

    if Debugging then
      Put_Line(" Start   = " & long_integer'image(Start_N));
      Put_Line(" End     = " & long_integer'image(End_N));
      Put_Line(" Phoneme  '" & Phon_Str.Txt(1..Phon_Str.Len) &
               "' = " & Phoneme'image(Phoneme_Read));
    end if;

  end Echo_Data;


  --*********************************************************************
  --* PUT_PHON - A Put routine that translates from my enum
  --* representations to standard symbols.  Used by Display_Matrix.
  --*********************************************************************

  Procedure Put_Phon(O_File : File_Type; Phon : Phoneme) is
  begin
    If Phon = HX then
      Put(O_File,"H#");
    elsif Phon = XXX then
      Put(O_File,"???");
    else
      Put(O_File,Phoneme'image(Phon));
    end if;
  end Put_Phon;


  --*********************************************************************
  --* DISPLAY_MATRIX - Prints final diphone matrix to output file (in 2 pieces)
  --*********************************************************************

  Procedure Display_Matrix is
    row,col,mid_col : Phoneme;
    Count : natural := 0;
  begin

Put_Line(Output_Datafile,"**********************************************************************************************");
    Put_Line(Output_Datafile,"                              MAX DIPHONE DURATIONS FOR FILES LISTED IN " &
Input_Filename);

Put_Line(Output_Datafile,"**********************************************************************************************");

    mid_col := Phoneme'val(integer((Phoneme'pos(Phoneme'last))/2));

    new_line(Output_Datafile);

                                              -- left half of matrix
    for col in Phoneme'first..Mid_Col loop  -- column labels
      set_col(Output_Datafile,col_widths*(1 + Phoneme'pos(col)));
      Put_Phon(Output_Datafile,Col);
    end loop;
    new_line(Output_Datafile,2);
    for row in Diphone_Matrix'range loop
      Put_Phon(Output_Datafile,Row);        -- row label
      for col in Phoneme'first..Mid_Col loop
          set_col(Output_Datafile,col_widths*(1 + Phoneme'pos(col)));
```

174

```
                 put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));

              if Diphone_Matrix(row,col) > 0 then  -- count nonzero entries
                count := count + 1;
              end if;

          end loop;
          new_line(Output_Datafile);
        end loop;
        new_line(Output_Datafile,2);

                                          -- right half of matrix
        for col in Phoneme'succ(Mid_Col)..Phoneme'last loop   -- col labels
          set_col(Output_Datafile,col_widths*(Phoneme'pos(col) - Phoneme'pos(Mid_Col)));
          Put_Phon(Output_Datafile,col);
        end loop;
        new_line(Output_Datafile,2);
        for row in Diphone_Matrix'range loop
          Put_Phon(Output_Datafile,Row);        -- row label
          for col in Phoneme'succ(Mid_Col)..Phoneme'last loop
              set_col(Output_Datafile,col_widths*(Phoneme'pos(col) - Phoneme'pos(Mid_Col)));
              put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));

              if Diphone_Matrix(row,col) > 0 then  -- count nonzero entries
                count := count + 1;
              end if;

          end loop;
          new_line(Output_Datafile);
        end loop;

        New_Line(Output_Datafile,3);

    Put_Line(Output_Datafile,"*****************************************************************************************************");
        New_Line(Output_Datafile);
        Put_Line(Output_Datafile,"              Total Diphones with nonzero entries = " & integer'image(count));

        New_Line(Output_Datafile);
        Put_Line(Output_Datafile," Maximum diphone duration data:");
        Put(Output_Datafile,"   Diphone = ");
        Put_Phon(Output_Datafile,Max_Dip_X);
        Put(Output_Datafile,"/");
        Put_Phon(Output_Datafile,Max_Dip_Y);
        New_Line(Output_Datafile);
        Put_Line(Output_Datafile,"   Duration (samples) = " & integer'image(Max_Duration));
        Put(Output_Datafile,"   Duration (milliseconds) = ");
        Put(Output_Datafile,integer'image(integer(1000.0*(float(Max_Duration)/Fs))));
        New_Line(Output_Datafile);
        Put_Line(Output_Datafile,"   Found in file: " & Max_Fname.Txt(1..Max_FName.Len));
        New_Line(Output_Datafile);

    Put_Line(Output_Datafile,"*****************************************************************************************************");

        New_Line(Output_Datafile,3);

                                -- Now list nonzero entries
```

175

```
Put_Line(Output_Datafile,"NONZERO ENTRIES ARE:");
New_Line(Output_Datafile);

for row in Diphone_Matrix'range loop
  for col in Phoneme'first..Phoneme'last loop
      if Diphone_Matrix(row,col) > 0 then
        Put_Phon(Output_Datafile,Row);
        set_col(Output_Datafile,col_widths);
        Put_Phon(Output_Datafile,Col);
        set_col(Output_Datafile,(3*col_widths)-(integer'image(Diphone_Matrix(row,col))'length));
        put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));
        new_line(Output_Datafile);
      end if;
  end loop;
end loop;

end Display_Matrix;


--*************************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--*************************************************************************

Procedure Get_File(Prompt_String : String;
                    F_Name : out String;
                    F_Mode : File_Mode;
                    F_Handle : in out File_Type) is
  Str_Last : Integer;
  Str_Text : String(1..Max_Str_Len) := (others => ' ');
begin


  loop    -- loop until valid filename entered or user aborts (^C)
    Begin   -- block statements to allow another try if bad filename entered

        Put(Prompt_String);
        Get_Line(Str_Text,Str_Last);

        if F_Mode = In_File then
          Text_IO.Open(File => F_Handle,
                  Mode => F_Mode,
                  Name => Str_Text(1..Str_Last));
        else
          Text_IO.Create(File => F_Handle,
                  Mode => F_Mode,
                  Name => Str_Text(1..Str_Last));
        end if;

        F_Name := Str_Text;

        exit;

    exception
        When Name_Error => Put(Ascii.Bel);
                           Put_Line("*** Unable to open " & Str_Text(1..Str_Last));
                           New_Line;
        When others => raise;
```

```
    end;
  end loop;

end Get_File;


--*************************************************************************
--*  RECORD_DURATION - Used by Process_Phn_File to record a diphone's duration
--*  in Diphone_Matrix.  Durations of HX (silence) & XXX (unknown) not counted
--*************************************************************************

Procedure Record_Duration (Start_N, Mid_N, End_N : Long_Integer;
                           First_Phon,Second_Phon : Phoneme;
                           Phon_Filename : Str) is
  Dip_Duration : Integer;
  m_Sec : Integer;
  Local_FName : Str;
  Valid_Dip : Boolean := false;  -- only valid dips count for max duration

begin

  if (First_Phon = HX) or (First_Phon = XXX) then
    Dip_Duration := integer(End_N - Mid_N);
  elsif (Second_Phon = HX) or (Second_Phon = XXX) then
    Dip_Duration := integer(Mid_N - Start_N);
  else
    Dip_Duration := integer(End_N - Start_N);
    Valid_Dip := True;
  end if;

  if Dip_Duration > Diphone_Matrix(First_Phon,Second_Phon) then
    Diphone_Matrix(First_Phon,Second_Phon) := Dip_Duration;
  end if;

  if Debugging then
    Put_Line("Diphone = " &
    Phoneme'Image(First_Phon) & "/" &
    Phoneme'Image(Second_Phon) &
    " duration = " &
    integer'image(Dip_Duration));
    New_Line;
  end if;

                              -- if new global max, record it
  If Valid_Dip and (Dip_Duration > Max_Duration) then
    Max_Duration := Dip_Duration;
    Max_Dip_X := First_Phon;
    Max_Dip_Y := Second_Phon;
    Max_Fname := Phon_Filename;
  end if;

end Record_Duration;


--*************************************************************************
--*  PROCESS_PHN_FILE - Opens designated file & records diphone durations
--*************************************************************************
Procedure Process_Phn_File (Phn_Filename : string) is
```

177

```
Phn_File : Text_IO.File_Type;
Last_PhnAtom,Phn_FName : Str;
This_Start,This_End,Last_Start,Last_End,Closure_Start,Closure_End : long_integer;
Closure_Pending : Boolean := False;
Phn_Line : string(1..Max_Str_Len);
Phn_Last : natural;
This_Phoneme,Last_Phoneme,Closure_Phoneme : Phoneme;
Closure_String : String(1..3);

begin
                        -- set up filename str for Record_Duration calls
Phn_FName.Txt(1..Phn_Filename'length) := Phn_Filename;
Phn_FName.Len := Phn_Filename'length;

Text_IO.Open(File => Phn_File,
             Mode => In_File,
             Name => Phn_Filename);

get_line(Phn_File,Phn_Line,Phn_Last);

if Phn_Last > 5 then

        if debugging then
          Put_Line(Phn_Line(1..Phn_Last));
        end if;

        Temp.Txt(1..Phn_Last) := Upper_Case(Phn_Line(1..Phn_Last));
        Temp.Len := Phn_Last;

        Get_Atom(Temp,Atom,Tail); -- get the start N
        Last_Start := Long_Integer'value(Atom.Txt(1..Atom.Len));

        Temp := Tail;
        Get_Atom(Temp,Atom,Tail); -- get the ending N
        Last_End := Long_Integer'value(Atom.Txt(1..Atom.Len));

        Temp := Tail;
        Get_Atom(Temp,Atom,Tail); -- get the phoneme string
        Last_Phoneme := Get_Phoneme(Atom);
        Last_PhnAtom := Atom;

        Echo_Data (Last_Start,Last_End,Atom,Last_Phoneme);

                        -- in case file starts with closure
        If Atom.Len = 3 and then
                ((Atom.Txt(1..Atom.Len) = "BCL") or
                 (Atom.Txt(1..Atom.Len) = "DCL") or
                 (Atom.Txt(1..Atom.Len) = "GCL") or
                 (Atom.Txt(1..Atom.Len) = "PCL") or
                 (Atom.Txt(1..Atom.Len) = "TCL") or
                 (Atom.Txt(1..Atom.Len) = "KCL") or
                 (Atom.Txt(1..Atom.Len) = "DCL") or
                 (Atom.Txt(1..Atom.Len) = "TCL")) then

                Closure_Pending := True; `
```

178

```
            Closure_Start := Last_Start;
            Closure_End := Last_End;
            Closure_String := Atom.Txt(1..Atom.Len);

            Last_Phoneme := HX;    -- ignore the "last" values

            If Debugging then
              Put_Line(" (Stop closure will be combined with stop's duration)");
            end if;
end if;

While NOT End_Of_File(Phn_File) loop


 get_line(Phn_File,Phn_Line,Phn_Last);

 if Phn_Last > 5 then

  if debugging then
        Put_Line(" New line    = " & Phn_Line(1..Phn_Last));
  end if;

  Temp.Txt(1..Phn_Last) := Upper_Case(Phn_Line(1..Phn_Last));
  Temp.Len := Phn_Last;

  Get_Atom(Temp,Atom,Tail); -- get the start N
  This_Start := Long_Integer'value(Atom.Txt(1..Atom.Len));

  Temp := Tail;
  Get_Atom(Temp,Atom,Tail); -- get the ending N
  This_End := Long_Integer'value(Atom.Txt(1..Atom.Len));

  Temp := Tail;
  Get_Atom(Temp,Atom,Tail); -- get the phoneme string
  This_Phoneme := Get_Phoneme(Atom);

  Echo_Data (This_Start,This_End,Atom,This_Phoneme);

  If Closure_Pending then
                    -- if this is stop to match pending closure
        If ((Closure_String = "BCL") and (This_Phoneme = B)) or
           ((Closure_String = "DCL") and (This_Phoneme = D)) or
           ((Closure_String = "GCL") and (This_Phoneme = G)) or
           ((Closure_String = "PCL") and (This_Phoneme = P)) or
           ((Closure_String = "TCL") and (This_Phoneme = T)) or
           ((Closure_String = "KCL") and (This_Phoneme = K)) or
           ((Closure_String = "DCL") and (This_Phoneme = JH))or
           ((Closure_String = "TCL") and (This_Phoneme = CH)) then

           -- record duration from last phoneme, thru closure to this
           -- phoneme's end.  next diphone starts with CLOSURE's start

           Record_Duration(Last_Start,Last_End,This_End,
                        Last_Phoneme,This_Phoneme,Phn_Fname);

           Last_Start := Closure_Start;
```

179

```
            Last_End := This_End;
            Last_Phoneme := This_Phoneme;
            Last_PhnAtom := Atom;

        else         -- else closure was not followed by matching stop

            -- record duration from last phoneme, thru closure's end
            -- as one diphone & from closure's start thru this
            -- phoneme's end as another.  next diphone starts with this
            -- phoneme's start

            Temp.Txt(1) := Closure_String(1);
            Temp.Len := 1;
            Closure_Phoneme := Get_Phoneme(Temp);

            Record_Duration(Last_Start,Last_End,Closure_End,
                            Last_Phoneme,Closure_Phoneme,Phn_Fname);

            Record_Duration(Closure_Start,Closure_End,This_End,
                            Closure_Phoneme,This_Phoneme,Phn_Fname);

            Last_Start := This_Start;
            Last_End := This_End;
            Last_Phoneme := This_Phoneme;
            Last_PhnAtom := Atom;

        end if;

        Closure_Pending := false;

                        -- see if THIS is a closure
    elsif Atom.Len = 3 and then
        ((Atom.Txt(1..Atom.Len) = "BCL") or
         (Atom.Txt(1..Atom.Len) = "DCL") or
         (Atom.Txt(1..Atom.Len) = "GCL") or
         (Atom.Txt(1..Atom.Len) = "PCL") or
         (Atom.Txt(1..Atom.Len) = "TCL") or
         (Atom.Txt(1..Atom.Len) = "KCL") or
         (Atom.Txt(1..Atom.Len) = "DCL") or
         (Atom.Txt(1..Atom.Len) = "TCL"))  then

        If Debugging then
          Put_Line(" (Stop closure will be combined with stop's duration)");
        end if;

        Closure_Pending := True;
        Closure_Start := This_Start;
        Closure_End := This_End;
        Closure_String := Atom.Txt(1..Atom.Len);

    else     -- handle simple case where no closures involved

        Record_Duration(Last_Start,Last_End,
                        This_End,Last_Phoneme,This_Phoneme,Phn_Fname);

        Last_Start := This_Start;
```

180

```
                Last_End := This_End;
                Last_Phoneme := This_Phoneme;
                Last_PhnAtom := Atom;

            end if;

        Echo_Data (Last_Start,Last_End,Last_PhnAtom,Last_Phoneme);

        end if;

    end loop;

                            -- handle case where last phoneme was a closure
    if Closure_Pending then

                    -- closure was not followed by matching stop

                -- record duration from last phoneme, thru closure's end
                -- as one diphone

        Temp.Txt(1) := Closure_String(1);
        Temp.Len := 1;
        Closure_Phoneme := Get_Phoneme(Temp);

        Record_Duration(Last_Start,Last_End,Closure_End,
                        Last_Phoneme,Closure_Phoneme,Phn_Fname);

    end if;

  end if;

  Close(Phn_File);

exception

  When Name_Error | Use_Error | Device_Error =>
            New_Line;
            Put_Line("***  Unable to open PHN file " & Phn_Filename);
            New_Line;


end Process_Phn_File;

--********************************************************************
--* DipDurs Main Program
--********************************************************************

Begin    -- DipDurs

  if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                               Arg.Data(3) = "/D" or
                               Arg.Data(3) = "-d" or
                               Arg.Data(3) = "-D" or
                               Arg.Data(3) = "d"  or
                               Arg.Data(3) = "D") then
        Debugging := True;
```

181

```
elsif Arg.Count > 2 then  -- note that program name is 1st entry in array
  raise Usage_Error;
end if;

New_Line(25);          -- blank lines & program "banner"

Put_Line(Version);
New_Line(2);

if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
  begin
    Text_IO.Open(File => Input_DataFile,
            Mode => In_File,
            Name => Arg.Data(2));
    Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
  exception              -- handle bad filenames from command line
When Name_Error  |
    Status_error |
    Mode_error   |
    Use_error    |
    Device_error =>  New_Line(3);
                        Put_Line("*** Unable to open " & Arg.Data(2));
                        New_Line;
                        raise Usage_Error;
  end;
else
  Get_File("Enter name of input file: ",
        Input_Filename,
        In_File,
        Input_Datafile);
end if;

                        -- get output filename from user
Get_File("Enter name of output file: ",
        Output_Filename,
        Out_File,
        Output_Datafile);

If Debugging then
  New_Line;
  Put_Line("--------------------[ PROCESSING FILE ]----------------------------");
end if;

While NOT End_Of_File(Input_DataFile) loop

  Get_Line(Input_DataFile,Input_Line,Input_Last);

  Input_Line := Upper_Case(Input_Line);

  if Input_Last > 0 then

    If Debugging then
        Put_Line("#### " & Input_Line(1..Input_Last) & " ####");
    end if;

    Process_Phn_File(Input_Line(1..Input_Last));
```

182

```
      If Debugging then
          Put_Line("############################################");
      end if;

    end if;

  end loop;

  Display_Matrix;

  If Debugging then
    Put_Line("------------------------[ DONE ]------------------------------");
    New_Line;
  end if;

  Text_IO.Close(Input_Datafile);
  Text_IO.Close(Output_Datafile);

exception

  When Usage_Error =>
    Put(Ascii.Bel);
    New_Line(3);
    Put_Line("Usage:  DipDurs infilename [-d]");
    New_Line;
    Put_Line("       -d = enable debugging");
    New_Line;
    return;

end DipDurs;
```

# E.    DIPNUMS.ADA

```
--************************************************************************
--************************************************************************
--** PROGRAM NAME: DipNums  (DipNums.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a list of phonetic transcription
--** files & recording how many times each diphone was found
--**
--** DATE MODIFIED: 30 Sep, 1995  (see below for version number)
--**
--************************************************************************
--************************************************************************

with Arg,
     Text_IO;

use Text_IO;

procedure DipNums is

  Max_Str_Len : constant := 255;
  Max_Phn_Len : constant := 5;
  Col_Widths : constant := 6;
  Fs : constant := 16000.0;
  FList_Filename : constant string := "DIPFLIST.TXT";
  DipList_Filename : constant string := "DIPLIST.TXT";

  Usage_Error : exception;

  Type STR is record
    Txt : String(1..Max_Str_Len);
    Len : Integer range 0..Max_Str_Len := 0;
  end record;

  Temp,Tail,Atom,Atom_Str,Tail_Str,Phon_Str : STR;


  Type Phoneme is (AA,AE,AH,AO,AW,AY,B,CH,D,DH,EH,ER,EY, -- 1st 3 lines from CMU lex
                   F,G,HH,IH,IY,JH,K,L,M,N,NG,OW,OY,P,R,
                   S,SH,T,TH,UH,UW,V,W,Y,Z,ZH,
                   EM,EN,ENG,EL,AX,IX,AXR,HX,      -- this line from TIMIT
                   XXX);                    -- use XXX for unknown codes

  Diphone_Matrix : array(Phoneme,Phoneme) of natural :=
                                             (others => (others => 0));


  Dip_Num_Needed : integer := 0;

  Debugging : boolean := false;
  Recording : Boolean := false;
```

```
Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
Name_Last : Natural;

Input_Line : string(1..Max_Str_Len);
Input_Last : natural;

Version : constant String := "DipNums Version #1.0";

Input_DataFile,Output_DataFile,FList_File,DipList_OutFile : Text_IO.File_Type;

--*************************************************************************
--* UPPER_CASE - Returns upper case version of input string
--*************************************************************************
Function  Upper_Case(In_String : string) return string is
  Result_String : String(In_String'first..In_String'last);
Begin
 for index in In_String'first..In_String'last loop
   If In_String(index) in 'a'..'z' then
 Result_String(index) := character'val(character'pos(In_String(index))
                    - character'pos('a')
                    + character'pos('A'));
   else
 Result_String(index) := In_String(index);
   end if;

 end loop;
 return Result_String;
end Upper_Case;


--*************************************************************************
--*  GET_ATOM - Gets next atom from a str
--*************************************************************************

Procedure Get_Atom(In_Str : Str;Atom_Str,Tail_Str : out Str) is
 i,a_start,a_end,a_len,t_len : integer := 1;
Begin
 If In_Str.Len > 0 then
   While (In_Str.Txt(i) = ' ') and (i < In_Str.Len) loop
       i := i + 1;
   end loop;
   If In_Str.Txt(i) /= ' ' then
       a_start := i;
       While (In_Str.Txt(i) /= ' ') and (i < In_Str.Len) loop
         i := i + 1;
       end loop;
       if In_Str.Txt(i) = ' ' then
         a_end := i - 1;
       else
         a_end := i;
       end if;
       a_len := a_end - a_start + 1;
       t_len := In_Str.Len - a_end;
       Tail_Str.Len := t_len;
       Atom_Str.Len := a_len;
       for i in 1..a_len loop
         Atom_Str.Txt(i) := In_Str.Txt(a_start + i - 1);
```

```
            end loop;
            for i in 1..t_len loop
              Tail_Str.Txt(i) := In_Str.Txt(a_end + i);
            end loop;
        else
            Atom_Str.Len := 0;
            Tail_Str.Len := 0;
        end if;
      else
        Atom_Str.Len := 0;
        Tail_Str.Len := 0;
      end if;
    end Get_Atom;


    --*************************************************************************
    --* GET_PHONEME - Returns Phoneme corresponding to an atom (atom must be free
    --* of leading & trailing blanks & other garbage (accent digits on end OK)
    --*************************************************************************


    Function Get_Phoneme (In_Atom : Str) return Phoneme is
      i,Phn_End : Integer;
      Phn_String : String (1..Max_Phn_Len);
      Out_Phoneme : Phoneme;
    Begin
      Phn_End := In_Atom.Len;
      if In_Atom.Len > 0 then
        While ((In_Atom.Txt(Phn_End) in '0'..'9')) and (Phn_End > 1) loop
            Phn_End := Phn_End - 1;
        end loop;
      end if;
      If (Phn_End > 0) and (Phn_End <= Max_Phn_Len) then
        Phn_String(1..Phn_End) := Upper_Case(In_Atom.Txt(1..Phn_End));

        begin  -- block to catch constraint error if not a valid phoneme
            Out_Phoneme := Phoneme'Value(Phn_String(1..Phn_End));
            return Out_Phoneme;
        exception
            when constraint_error =>
              if Phn_End = 2 and then Phn_String(1..2) = "H#" then
                return HX;
              else
                return XXX;
              end if;
        end;  -- block

      else
        return XXX;
      end if;
    end Get_Phoneme;


    --*************************************************************************
    --* ECHO_DATA - Used by Process_Phn_File to echo the data read from file
    --*************************************************************************


    Procedure Echo_Data (Start_N, End_N : Long_Integer;
                         Phon_Str : Str;
```

186

Phoneme_Read : Phoneme) is

begin

  if Debugging then
    Put_Line(" Start   = " & long_integer'image(Start_N));
    Put_Line(" End     = " & long_integer'image(End_N));
    Put_Line(" Phoneme  '" & Phon_Str.Txt(1..Phon_Str.Len) &
              "' = " & Phoneme'image(Phoneme_Read));
  end if;

end Echo_Data;


--***********************************************************************
--* PUT_PHON - A Put routine that translates from my enum
--* representations to standard symbols.  Used by Display_Matrix.
--***********************************************************************

Procedure Put_Phon(O_File : File_Type; Phon : Phoneme) is
begin
  If Phon = HX then
    Put(O_File,"H#");
  elsif Phon = XXX then
    Put(O_File,"???");
  else
    Put(O_File,Phoneme'image(Phon));
  end if;
end Put_Phon;


--***********************************************************************
--* DISPLAY_MATRIX - Prints final diphone matrix to output file (in 2 pieces)
--***********************************************************************

Procedure Display_Matrix is
  row,col,mid_col : Phoneme;
  Count : natural := 0;
  maxed_count : natural := 0;
begin


Put_Line(Output_Datafile,"***********************************************************************************");
      Put_Line(Output_Datafile,"                    MAX DIPHONE DURATIONS FOR FILES LISTED IN " &
Input_Filename);

Put_Line(Output_Datafile,"***********************************************************************************");


  mid_col := Phoneme'val(integer((Phoneme'pos(Phoneme'last))/2));

  new_line(Output_Datafile);

                                        -- left half of matrix
  for col in Phoneme'first..Mid_Col loop  -- column labels
    set_col(Output_Datafile,col_widths*(1 + Phoneme'pos(col)));
    Put_Phon(Output_Datafile,Col);
  end loop;

187

```
new_line(Output_Datafile,2);
for row in Diphone_Matrix'range loop
  Put_Phon(Output_Datafile,Row);        -- row label
  for col in Phoneme'first..Mid_Col loop
      set_col(Output_Datafile,col_widths*(1 + Phoneme'pos(col)));
      put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));

      if Diphone_Matrix(row,col) > 0 then  -- count nonzero entries
        if (Row /= XXX) and (Col /= XXX) and
          (Diphone_Matrix(row,col) >= Dip_Num_Needed) then
          maxed_count := maxed_count + 1;
          Put_Phon(DipList_Outfile,row);
          Put(DipList_Outfile,' ');
          Put_Phon(DipList_Outfile,col);
          New_Line(DipList_Outfile);
        end if;
        count := count + 1;
      end if;

  end loop;
  new_line(Output_Datafile);
end loop;
new_line(Output_Datafile,2);

                              -- right half of matrix
for col in Phoneme'succ(Mid_Col)..Phoneme'last loop   -- col labels
  set_col(Output_Datafile,col_widths*(Phoneme'pos(col) - Phoneme'pos(Mid_Col)));
  Put_Phon(Output_Datafile,col);
end loop;
new_line(Output_Datafile,2);
for row in Diphone_Matrix'range loop
  Put_Phon(Output_Datafile,Row);        -- row label
  for col in Phoneme'succ(Mid_Col)..Phoneme'last loop
      set_col(Output_Datafile,col_widths*(Phoneme'pos(col) - Phoneme'pos(Mid_Col)));
      put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));

      if Diphone_Matrix(row,col) > 0 then  -- count nonzero entries
        if (Row /= XXX) and (Col /= XXX) and
          (Diphone_Matrix(row,col) >= Dip_Num_Needed) then
          maxed_count := maxed_count + 1;
          Put_Phon(DipList_Outfile,row);
          Put(DipList_Outfile,' ');
          Put_Phon(DipList_Outfile,col);
          New_Line(DipList_Outfile);
        end if;
        count := count + 1;
      end if;

  end loop;
  new_line(Output_Datafile);
end loop;

New_Line(Output_Datafile,3);

Put_Line(Output_Datafile,"*********************************************************************************************");
  New_Line(Output_Datafile);
```

188

```
      Put_Line(Output_Datafile,"                    Total Recognized Diphones with nonzero entries = " &
integer'image(count));
      New_Line(Output_Datafile);
      Put_Line(Output_Datafile,"            Total Recognized Diphones with requested number of files = " &
integer'image(maxed_count));
      New_Line(Output_Datafile);

Put_Line(Output_Datafile,"******************************************************************************************");

   New_Line(Output_Datafile,3);


                                    -- Now list nonzero entries
      Put_Line(Output_Datafile,"NONZERO ENTRIES ARE:");
      New_Line(Output_Datafile);

   for row in Diphone_Matrix'range loop
      for col in Phoneme'first..Phoneme'last loop
         if Diphone_Matrix(row,col) > 0 then
           Put_Phon(Output_Datafile,Row);
           set_col(Output_Datafile,col_widths);
           Put_Phon(Output_Datafile,Col);
           set_col(Output_Datafile,(3*col_widths)-(integer'image(Diphone_Matrix(row,col))'length));
           put(Output_Datafile,integer'image(Diphone_Matrix(row,col)));
           new_line(Output_Datafile);
         end if;
      end loop;
   end loop;


end Display_Matrix;



--*****************************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--*****************************************************************************

Procedure Get_File(Prompt_String : String;
                   F_Name : out String;
                   F_Mode : File_Mode;
                   F_Handle : in out File_Type) is
 Str_Last : Integer;
 Str_Text : String(1..Max_Str_Len) := (others => ' ');
begin


  loop   -- loop until valid filename entered or user aborts (^C)
    Begin   -- block statements to allow another try if bad filename entered

       Put(Prompt_String);
       Get_Line(Str_Text,Str_Last);


       if F_Mode = In_File then
         Text_IO.Open(File => F_Handle,
```

189

```
                Mode => F_Mode,
                Name => Str_Text(1..Str_Last));
        else
          Text_IO.Create(File => F_Handle,
                Mode => F_Mode,
                Name => Str_Text(1..Str_Last));
        end if;

        F_Name := Str_Text;

        exit;

    exception
        When Name_Error => Put(Ascii.Bel);
                        Put_Line("*** Unable to open " & Str_Text(1..Str_Last));
                        New_Line;
        When others => raise;
    end;
  end loop;

end Get_File;


--************************************************************************
--* Record_Diphone - Used by Process_Phn_File to record a diphone's duration
--* in Diphone_Matrix.  Durations of HX (silence) & XXX (unknown) not counted
--************************************************************************

Procedure Record_Diphone (Start_N, Mid_N, End_N : Long_Integer;
                        First_Phon,Second_Phon : Phoneme;
                        Phon_Filename : Str) is
  Local_FName : Str;
  Valid_Dip : Boolean := true;  -- only valid dips count

begin

  if (First_Phon = XXX) or (Second_Phon = XXX) then
    Valid_Dip := False;
  end if;

  if recording and Valid_Dip then
    Local_Fname := Phon_Filename;
    If (Local_Fname.Len > 4) and then
        (Local_FName.Txt((Local_Fname.Len-3)..Local_Fname.Len)) = ".PHN" then
        Local_FName.Txt((Local_Fname.Len-3)..Local_Fname.Len) := ".WAV";
        Put(FList_File,Local_FName.Txt(1..Local_Fname.Len) & " ");
        Put(FList_File,Phoneme'Image(First_Phon) & "-" & Phoneme'Image(Second_Phon) & " ");
        Put(FList_File,long_integer'image(Start_N) & " ");
        Put(FList_File,long_integer'image(Mid_N) & " ");
        Put_Line(FList_File,long_integer'image(End_N));
    end if;
  end if;


  Diphone_Matrix(First_Phon,Second_Phon) := Diphone_Matrix(First_Phon,Second_Phon) + 1;
```

```ada
 if Debugging then
   Put_Line("Diphone = " &
   Phoneme'Image(First_Phon) & "-" &
   Phoneme'Image(Second_Phon));
   New_Line;
 end if;

end Record_Diphone;
```

--*************************************************************************
--* PROCESS_PHN_FILE - Opens designated file & records diphone durations
--*************************************************************************
```ada
Procedure Process_Phn_File (Phn_Filename : string) is

 Phn_File : Text_IO.File_Type;
 Last_PhnAtom,Phn_FName : Str;
 This_Start,This_End,Last_Start,Last_End,Closure_Start,Closure_End : long_integer;
 Closure_Pending : Boolean := False;
 Phn_Line : string(1..Max_Str_Len);
 Phn_Last : natural;
 This_Phoneme,Last_Phoneme,Closure_Phoneme : Phoneme;
 Closure_String : String(1..3);

begin
                       -- set up filename str for Record_Diphone calls
 Phn_FName.Txt(1..Phn_Filename'length) := Phn_Filename;
 Phn_FName.Len := Phn_Filename'length;

 Text_IO.Open(File => Phn_File,
              Mode => In_File,
              Name => Phn_Filename);

 get_line(Phn_File,Phn_Line,Phn_Last);

 if Phn_Last > 5 then

        if debugging then
          Put_Line(Phn_Line(1..Phn_Last));
        end if;

        Temp.Txt(1..Phn_Last) := Upper_Case(Phn_Line(1..Phn_Last));
        Temp.Len := Phn_Last;

        Get_Atom(Temp,Atom,Tail); -- get the start N
        Last_Start := Long_Integer'value(Atom.Txt(1..Atom.Len));

        Temp := Tail;
        Get_Atom(Temp,Atom,Tail); -- get the ending N
        Last_End := Long_Integer'value(Atom.Txt(1..Atom.Len));

        Temp := Tail;
        Get_Atom(Temp,Atom,Tail); -- get the phoneme string
        Last_Phoneme := Get_Phoneme(Atom);
        Last_PhnAtom := Atom;
```

191

```
Echo_Data (Last_Start,Last_End,Atom,Last_Phoneme);

                   -- in case file starts with closure
If Atom.Len = 3 and then
        ((Atom.Txt(1..Atom.Len) = "BCL") or
         (Atom.Txt(1..Atom.Len) = "DCL") or
         (Atom.Txt(1..Atom.Len) = "GCL") or
         (Atom.Txt(1..Atom.Len) = "PCL") or
         (Atom.Txt(1..Atom.Len) = "TCL") or
         (Atom.Txt(1..Atom.Len) = "KCL") or
         (Atom.Txt(1..Atom.Len) = "DCL") or
         (Atom.Txt(1..Atom.Len) = "TCL"))  then

        Closure_Pending := True;
        Closure_Start := Last_Start;
        Closure_End := Last_End;
        Closure_String := Atom.Txt(1..Atom.Len);

        Last_Phoneme := HX;     -- ignore the "last" values

        If Debugging then
          Put_Line(" (Stop closure will be combined with stop's duration)");
        end if;
end if;

While NOT End_Of_File(Phn_File) loop

 get_line(Phn_File,Phn_Line,Phn_Last);

 if Phn_Last > 5 then

  if debugging then
       Put_Line(" New line    = " & Phn_Line(1..Phn_Last));
  end if;

  Temp.Txt(1..Phn_Last) := Upper_Case(Phn_Line(1..Phn_Last));
  Temp.Len := Phn_Last;

  Get_Atom(Temp,Atom,Tail); -- get the start N
  This_Start := Long_Integer'value(Atom.Txt(1..Atom.Len));

  Temp := Tail;
  Get_Atom(Temp,Atom,Tail); -- get the ending N
  This_End := Long_Integer'value(Atom.Txt(1..Atom.Len));

  Temp := Tail;
  Get_Atom(Temp,Atom,Tail); -- get the phoneme string
  This_Phoneme := Get_Phoneme(Atom);

  Echo_Data (This_Start,This_End,Atom,This_Phoneme);


  If Closure_Pending then
                      -- if this is stop to match pending closure
        If ((Closure_String = "BCL") and (This_Phoneme = B)) or
```

192

```
            ((Closure_String = "DCL") and (This_Phoneme = D)) or
            ((Closure_String = "GCL") and (This_Phoneme = G)) or
            ((Closure_String = "PCL") and (This_Phoneme = P)) or
            ((Closure_String = "TCL") and (This_Phoneme = T)) or
            ((Closure_String = "KCL") and (This_Phoneme = K)) or
            ((Closure_String = "DCL") and (This_Phoneme = JH))or
            ((Closure_String = "TCL") and (This_Phoneme = CH)) then

         -- record duration from last phoneme, thru closure to this
         -- phoneme's end.  next diphone starts with CLOSURE's start

         Record_Diphone(Last_Start,Last_End,This_End,
                        Last_Phoneme,This_Phoneme,Phn_Fname);

         Last_Start := Closure_Start;
         Last_End := This_End;
         Last_Phoneme := This_Phoneme;
         Last_PhnAtom := Atom;

      else         -- else closure was not followed by matching stop

         -- record duration from last phoneme, thru closure's end
         -- as one diphone & from closure's start thru this
         -- phoneme's end as another.  next diphone starts with this
         -- phoneme's start

         Temp.Txt(1) := Closure_String(1);
         Temp.Len := 1;
         Closure_Phoneme := Get_Phoneme(Temp);

         Record_Diphone(Last_Start,Last_End,Closure_End,
                        Last_Phoneme,Closure_Phoneme,Phn_Fname);

         Record_Diphone(Closure_Start,Closure_End,This_End,
                        Closure_Phoneme,This_Phoneme,Phn_Fname);

         Last_Start := This_Start;
         Last_End := This_End;
         Last_Phoneme := This_Phoneme;
         Last_PhnAtom := Atom;

      end if;

      Closure_Pending := false;

                      -- see if THIS is a closure
elsif Atom.Len = 3 and then
      ((Atom.Txt(1..Atom.Len) = "BCL") or
       (Atom.Txt(1..Atom.Len) = "DCL") or
       (Atom.Txt(1..Atom.Len) = "GCL") or
       (Atom.Txt(1..Atom.Len) = "PCL") or
       (Atom.Txt(1..Atom.Len) = "TCL") or
       (Atom.Txt(1..Atom.Len) = "KCL") or
       (Atom.Txt(1..Atom.Len) = "DCL") or
       (Atom.Txt(1..Atom.Len) = "TCL")) then
```

```
          If Debugging then
            Put_Line(" (Stop closure will be combined with stop's duration)");
          end if;

          Closure_Pending := True;
          Closure_Start := This_Start;
          Closure_End := This_End;
          Closure_String := Atom.Txt(1..Atom.Len);

     else    -- handle simple case where no closures involved

          Record_Diphone(Last_Start,Last_End,
                      This_End,Last_Phoneme,This_Phoneme,Phn_Fname);

          Last_Start := This_Start;
          Last_End := This_End;
          Last_Phoneme := This_Phoneme;
          Last_PhnAtom := Atom;

     end if;

     Echo_Data (Last_Start,Last_End,Last_PhnAtom,Last_Phoneme);

    end if;

  end loop;

                  -- handle case where last phoneme was a closure
  if Closure_Pending then

                  -- closure was not followed by matching stop

          -- record duration from last phoneme, thru closure's end
          -- as one diphone

          Temp.Txt(1) := Closure_String(1);
          Temp.Len := 1;
          Closure_Phoneme := Get_Phoneme(Temp);

          Record_Diphone(Last_Start,Last_End,Closure_End,
                      Last_Phoneme,Closure_Phoneme,Phn_Fname);

     end if;

  end if;

  Close(Phn_File);

exception

  When Name_Error | Use_Error | Device_Error =>
             New_Line;
             Put_Line("***  Unable to open PHN file " & Phn_Filename);
             New_Line;
```

194

```
end Process_Phn_File;

--*************************************************************************
--*  DipNums Main Program
--*************************************************************************


Begin    -- DipNums

if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                             Arg.Data(3) = "/D" or
                             Arg.Data(3) = "-d" or
                             Arg.Data(3) = "-D" or
                             Arg.Data(3) = "d"  or
                             Arg.Data(3) = "D") then
  Debugging := True;
elsif Arg.Count > 2 then  -- note that program name is 1st entry in array
  raise Usage_Error;
end if;


New_Line(25);            -- blank lines & program "banner"

Put_Line(Version);
New_Line(2);

if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
  begin
    Text_IO.Open(File => Input_DataFile,
             Mode => In_File,
             Name => Arg.Data(2));
    Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
  exception             -- handle bad filenames from command line
  When Name_Error  |
     Status_error |
     Mode_error   |
     Use_error    |
     Device_error =>  New_Line(3);
                      Put_Line("*** Unable to open " & Arg.Data(2));
                      New_Line;
                      raise Usage_Error;
  end;
else
  Get_File("Enter name of input file: ",
        Input_Filename,
        In_File,
        Input_Datafile);
end if;

                             -- get output filename from user
Get_File("Enter name of output file: ",
        Output_Filename,
        Out_File,
        Output_Datafile);


        -- get number of files to record per diphone
loop    -- loop until valid integer entered or user aborts (^C)
```

195

```
Begin   -- block statements to allow another try if bad filename entered

        Put("How many instances of each diphone required (0 if none)? ");
        Get_Line(Input_Line,Input_Last);

        Dip_Num_Needed := integer'value(input_line(1..input_last));

        If Dip_Num_Needed > 0 then

          Text_IO.Create(File => FList_File,
                         Mode => Out_File,
                         Name => FList_Filename);

          Text_IO.Create(File => DipList_OutFile,
                         Mode => Out_File,
                         Name => DipList_Filename);

          Recording := True;

        else

          Recording := false;

        end if;

        exit;

    exception
        When constraint_Error => Put(Ascii.Bel);
                                 Put_Line("*** You must enter an integer (0 to cancel recording) ***");
                                 New_Line;
        When others => raise;
    end;
end loop;

If Debugging then
  New_Line;
  Put_Line("---------------------[ PROCESSING FILE ]----------------------------");
end if;

While NOT End_Of_File(Input_DataFile) loop

 Get_Line(Input_DataFile,Input_Line,Input_Last);

 Input_Line := Upper_Case(Input_Line);

 if Input_Last > 0 then

  If Debugging then
      Put_Line("#### " & Input_Line(1..Input_Last) & " ####");
  end if;

  Process_Phn_File(Input_Line(1..Input_Last));

  If Debugging then
      Put_Line("####################################################");
```

196

```
    end if;

  end if;

end loop;

Display_Matrix;

If Debugging then
  Put_Line("------------------------[ DONE ]-----------------------------");
  New_Line;
end if;

Text_IO.Close(Input_Datafile);
Text_IO.Close(Output_Datafile);

If Recording then
  Text_IO.Close(FList_File);
  Text_IO.Close(DipList_Outfile);
end if;

exception

When Usage_Error =>
  Put(Ascii.Bel);
  New_Line(3);
  Put_Line("Usage:  DipNums infilename [-d]");
  New_Line;
  Put_Line("       -d = enable debugging");
  New_Line;
  return;

end DipNums;
```

# F. MAKFLIST.ADA

```
--*********************************************************************
--*********************************************************************
--** PROGRAM NAME: MakFList  (MakFList.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a list of TIMIT WAV/diphone
--** files & generating a new list with N occurences of each diphone
--** contained in DIPLIST.TXT.
--**
--** DATE MODIFIED: 3 Oct, 1995  (see below for version number)
--**
--*********************************************************************
--*********************************************************************

with Arg,
    Text_IO;

use Text_IO;

procedure MakFList is

  Max_Str_Len : constant := 255;
  Max_Phn_Len : constant := 5;
  Col_Widths : constant := 6;
  Fs : constant := 16000.0;

  DipList_Filename : constant string := "DIPLIST.TXT";

  Usage_Error : exception;

  Type STR is record
    Txt : String(1..Max_Str_Len);
    Len : Integer range 0..Max_Str_Len := 0;
  end record;

  Temp,Tail,Atom,Atom_Str,Tail_Str,Phon_Str : STR;


  Type Phoneme is (AA,AE,AH,AO,AW,AY,B,CH,D,DH,EH,ER,EY, -- 1st 3 lines from CMU lex
                F,G,HH,IH,IY,JH,K,L,M,N,NG,OW,OY,P,R,
                S,SH,T,TH,UH,UW,V,W,Y,Z,ZH,
                EM,EN,ENG,EL,AX,IX,AXR,HX,         -- this line from TIMIT
                XXX);                    -- use XXX for unknown codes

  Diphone_Matrix : array(Phoneme,Phoneme) of natural :=
                                          (others => (others => integer'last));


  Dip_Rec_Limit : integer := 0;

  Debugging : boolean := false;
  Recording : Boolean := false;
```

198

```
Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
Name_Last : Natural;

Input_Line : string(1..Max_Str_Len);
Input_Last : natural;

Version : constant String := "MakFList Version #1.0";

Input_DataFile,Output_DataFile,DipList_InFile : Text_IO.File_Type;

--************************************************************************
--* UPPER_CASE - Returns upper case version of input string
--************************************************************************
Function Upper_Case(In_String : string) return string is
  Result_String : String(In_String'first..In_String'last);
Begin
  for index in In_String'first..In_String'last loop
    If In_String(index) in 'a'..'z' then
Result_String(index) := character'val(character'pos(In_String(index))
                   - character'pos('a')
                   + character'pos('A'));
    else
Result_String(index) := In_String(index);
    end if;

  end loop;
  return Result_String;
end Upper_Case;

--************************************************************************
--* GET_ATOM - Gets next atom from a str
--************************************************************************

Procedure Get_Atom(In_Str : Str;Atom_Str,Tail_Str : out Str) is
  i,a_start,a_end,a_len,t_len : integer := 1;
Begin
  If In_Str.Len > 0 then
    While (In_Str.Txt(i) = ' ') and (i < In_Str.Len) loop
        i := i + 1;
    end loop;
    If In_Str.Txt(i) /= ' ' then
        a_start := i;
        While (In_Str.Txt(i) /= ' ') and (i < In_Str.Len) loop
          i := i + 1;
        end loop;
        if In_Str.Txt(i) = ' ' then
         a_end := i - 1;
        else
         a_end := i;
        end if;
        a_len := a_end - a_start + 1;
        t_len := In_Str.Len - a_end;
        Tail_Str.Len := t_len;
        Atom_Str.Len := a_len;
        for i in 1..a_len loop
```

199

```
            Atom_Str.Txt(i) := In_Str.Txt(a_start + i - 1);
          end loop;
          for i in 1..t_len loop
            Tail_Str.Txt(i) := In_Str.Txt(a_end + i);
          end loop;
      else
            Atom_Str.Len := 0;
            Tail_Str.Len := 0;
      end if;
    else
      Atom_Str.Len := 0;
      Tail_Str.Len := 0;
    end if;
end Get_Atom;


--*****************************************************************
--* GET_PHONEME - Returns Phoneme corresponding to an atom (atom must be free
--* of leading & trailing blanks & other garbage (accent digits on end OK)
--*****************************************************************

Function Get_Phoneme (In_Atom : Str) return Phoneme is
  i,Phn_End : Integer;
  Phn_String : String (1..Max_Phn_Len);
  Out_Phoneme : Phoneme;
Begin
  Phn_End := In_Atom.Len;
  if In_Atom.Len > 0 then
    While ((In_Atom.Txt(Phn_End) in '0'..'9')) and (Phn_End > 1) loop
        Phn_End := Phn_End - 1;
    end loop;
  end if;
  If (Phn_End > 0) and (Phn_End <= Max_Phn_Len) then
    Phn_String(1..Phn_End) := Upper_Case(In_Atom.Txt(1..Phn_End));

    begin -- block to catch constraint error if not a valid phoneme
        Out_Phoneme := Phoneme'Value(Phn_String(1..Phn_End));
        return Out_Phoneme;
    exception
        when constraint_error =>
        if Phn_End = 2 and then Phn_String(1..2) = "H#" then
          return HX;
        else
          return XXX;
        end if;
    end; -- block

  else
    return XXX;
  end if;
end Get_Phoneme;


--*****************************************************************
--* PUT_PHON - A Put routine that translates from my enum
--* representations to standard symbols.  Used by Display_Matrix.
--*****************************************************************
```

```
Procedure Put_Phon(O_File : File_Type; Phon : Phoneme) is
begin
  If Phon = HX then
    Put(O_File,"H#");
  elsif Phon = XXX then
    Put(O_File,"???");
  else
    Put(O_File,Phoneme'image(Phon));
  end if;
end Put_Phon;


--************************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--************************************************************************

Procedure Get_File(Prompt_String : String;
                    F_Name : out String;
                    F_Mode : File_Mode;
                    F_Handle : in out File_Type) is
  Str_Last : Integer;
  Str_Text : String(1..Max_Str_Len) := (others => ' ');
begin


  loop    -- loop until valid filename entered or user aborts (^C)
    Begin   -- block statements to allow another try if bad filename entered

        Put(Prompt_String);
        Get_Line(Str_Text,Str_Last);


        if F_Mode = In_File then
          Text_IO.Open(File => F_Handle,
                Mode => F_Mode,
                Name => Str_Text(1..Str_Last));
        else
          Text_IO.Create(File => F_Handle,
                Mode => F_Mode,
                Name => Str_Text(1..Str_Last));
        end if;

        F_Name := Str_Text;

        exit;

    exception
        When Name_Error => Put(Ascii.Bel);
                           Put_Line("*** Unable to open " & Str_Text(1..Str_Last));
                           New_Line;
        When others => raise;
    end;
  end loop;

end Get_File;
```

```
--*************************************************************************
--* Record_Diphone - Output the line if this diphone still needed
--*************************************************************************

Procedure Record_Diphone (File_Line : String;
                          First_Phon,Second_Phon : Phoneme) is

  Valid_Dip : Boolean := true;  -- only valid dips count

begin

  if (First_Phon = XXX) or (Second_Phon = XXX) then
    Valid_Dip := False;
  end if;

  if recording and Valid_Dip and
    (Diphone_Matrix(First_Phon,Second_Phon) < Dip_Rec_Limit) then

    Put_Line(Output_Datafile,File_Line);
    Diphone_Matrix(First_Phon,Second_Phon) := Diphone_Matrix(First_Phon,Second_Phon) + 1;

  end if;

  if Debugging then
    Put_Line("Diphone = " &
    Phoneme'Image(First_Phon) & "-" &
    Phoneme'Image(Second_Phon));
    New_Line;
  end if;

end Record_Diphone;

--*************************************************************************
--* PROCESS_LINE - Process line from file list & output if that diphone is
--*  still needed
--*************************************************************************
Procedure Process_Line(File_Line : string) is

  Hyphen_Pos : natural := 0;
  First_Phn_Str,Last_Phn_Str,Line_Str,Tail,Temp : Str;
  Row,Col : Phoneme;

begin

  Line_Str.Txt(1..File_Line'length) := File_Line;
  Line_Str.Len := File_Line'length;

  Get_Atom(Line_Str,Temp,Tail);      -- fetch and discard filename

  Temp := Tail;

  Get_Atom(Temp,First_Phn_Str,Tail); -- fetch diphone string

  Last_Phn_Str := First_Phn_Str;     -- store diphone in both str's & then
                                     -- remove one phoneme from each
```

```
For i in 1..First_Phn_Str.Len loop -- determine where first phoneme ends
   if (First_Phn_Str.Txt(i) = '-') or (First_Phn_Str.Txt(i) = '/') then
      Hyphen_Pos := i;
   end if;
end loop;


First_Phn_Str.Len := Hyphen_Pos - 1; -- "remove" 2nd phoneme from tail of 1st


                                    -- remove 1st phoneme from front of 2nd
For i in Hyphen_Pos+1..Last_Phn_Str.Len loop
   Last_Phn_Str.Txt(i-Hyphen_Pos) := Last_Phn_Str.Txt(i);
end loop;
Last_Phn_Str.Len := Last_Phn_Str.Len - Hyphen_Pos;


Row := Get_Phoneme(First_Phn_Str);
Col := Get_Phoneme(Last_Phn_Str);



Record_Diphone(File_Line,Row,Col);

end Process_Line;


--***********************************************************************
--* INIT_MATRIX - Set matrix cells to zero if that diphone is listed in
--* diplist.txt.  That allows file lines with those diphones to be output.
--***********************************************************************
Procedure Init_Matrix is

Input_Line : String(1..Max_Str_Len);
Input_Last : natural;
First_Phn_Str,Last_Phn_Str,Line_Str,Tail,Temp : Str;
Row,Col : Phoneme;

Begin

Text_IO.Open(File => DipList_InFile,
                     Mode => In_File,
                     Name => DipList_Filename);

While NOT End_Of_File(DipList_InFile) loop

Get_Line(DipList_InFile,Input_Line,Input_last);

Line_Str.Txt(1..Input_Last) := Input_Line(1..Input_Last);
Line_Str.Len := Input_Last;

Get_Atom(Line_Str,First_Phn_Str,Tail); -- fetch first phoneme str

Get_Atom(Tail,Last_Phn_Str,Temp);      -- fetch last phoneme str


Row := Get_Phoneme(First_Phn_Str);
Col := Get_Phoneme(Last_Phn_Str);

Diphone_Matrix(row,col) := 0;
```

```
    end loop;

  end Init_Matrix;

--*************************************************************************
--* MakFList Main Program
--*************************************************************************

Begin   -- MakFList

  if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                               Arg.Data(3) = "/D" or
                               Arg.Data(3) = "-d" or
                               Arg.Data(3) = "-D" or
                               Arg.Data(3) = "d"  or
                               Arg.Data(3) = "D") then
    Debugging := True;
  elsif Arg.Count > 2 then  -- note that program name is 1st entry in array
    raise Usage_Error;
  end if;

  New_Line(25);          -- blank lines & program "banner"

  Put_Line(Version);
  New_Line(2);

  if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
    begin
      Text_IO.Open(File => Input_DataFile,
               Mode => In_File,
               Name => Arg.Data(2));
      Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
    exception             -- handle bad filenames from command line
    When Name_Error  |
         Status_error |
         Mode_error   |
         Use_error    |
         Device_error =>  New_Line(3);
                          Put_Line("*** Unable to open " & Arg.Data(2));
                          New_Line;
                          raise Usage_Error;
    end;
  else
    Get_File("Enter name of input file: ",
         Input_Filename,
         In_File,
         Input_Datafile);
  end if;

                          -- get output filename from user
  Get_File("Enter name of output file: ",
       Output_Filename,
       Out_File,
       Output_Datafile);
```

```
                    -- get number of files to record per diphone
loop    -- loop until valid integer entered or user aborts (^C)
   Begin   -- block statements to allow another try if bad filename entered

      Put("How many filenames required for each diphone? ");
      Get_Line(Input_Line,Input_Last);

      Dip_Rec_Limit := integer'value(input_line(1..input_last));

      If Dip_Rec_Limit > 0 then

        Init_Matrix;
        Recording := True;

       else

        raise constraint_error;

       end if;

       exit;

    exception
        When constraint_Error => Put(Ascii.Bel);
                                 Put_Line("*** You must enter a nonzero integer ***");
                                 New_Line;
        When others => raise;
   end;
end loop;


If Debugging then
  New_Line;
  Put_Line("--------------------[ PROCESSING FILE ]----------------------------");
end if;

While NOT End_Of_File(Input_DataFile) loop

 Get_Line(Input_DataFile,Input_Line,Input_Last);

 Input_Line := Upper_Case(Input_Line);

 if Input_Last > 0 then

  If Debugging then
      Put_Line("#### " & Input_Line(1..Input_Last) & " ####");
  end if;

  Process_Line(Input_Line(1..Input_Last));

  If Debugging then
      Put_Line("#########################################");
  end if;

 end if;
```

```
   end loop;

   If Debugging then
      Put_Line("------------------------[ DONE ]-----------------------------");
      New_Line;
   end if;

   Text_IO.Close(Input_Datafile);
   Text_IO.Close(Output_Datafile);
   Text_IO.Close(DipList_Infile);

exception

   When Usage_Error =>
      Put(Ascii.Bel);
      New_Line(3);
      Put_Line("Usage:  MakFList infilename [-d]");
      New_Line;
      Put_Line("        -d = enable debugging");
      New_Line;
      return;

end MakFList;
```

# G.  VOCAB.ADA

```
--**********************************************************************
--**********************************************************************
--** PROGRAM NAME: Vocab  (vocab.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a phonetic dictionary & creating a
--** text file list of words in that dictionary that are composed of
--** diphones in DIPLIST.TXT.
--**
--** DATE MODIFIED: 8 Mar, 1996  (see below for version number)
--**
--**********************************************************************
--**********************************************************************

with Arg,
     Text_IO;

use Text_IO;

procedure Vocab is

  Version : constant String := "Vocab Version #3.0";

  DipList_FileName : constant String := "diplist.txt";

  Max_Str_Len : constant := 255;
  Max_Phn_Len : constant := 5;
  Col_Widths : constant := 6;

  Usage_Error : exception;

  Type STR is record
    Txt : String(1..Max_Str_Len);
    Len : Integer range 0..Max_Str_Len := 0;
  end record;

  Temp,Tail,Atom,Word,Atom_Str,Tail_Str,Phon_Str : STR;


  Type Phoneme is (AA,AE,AH,AO,AW,AY,B,CH,D,DH,EH,ER,EY, -- 1st 3 lines from CMU lex
              F,G,HH,IH,IY,JH,K,L,M,N,NG,OW,OY,P,R,
              S,SH,T,TH,UH,UW,V,W,Y,Z,ZH,
              EM,EN,ENG,EL,AX,IX,AXR,HX,        -- this line from TIMIT
              XXX);                      -- use XXX for unknown codes

  Diphone_Matrix : array(Phoneme,Phoneme) of natural :=
                                          (others => (others => 0));


  This_Phoneme,Last_Phoneme : Phoneme;

  Debugging : boolean := false;
```

```
Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
Name_Last : Natural;

Input_Line : string(1..Max_Str_Len);
Input_Last : natural;


Input_DataFile,Output_DataFile,DipList_InFile : Text_IO.File_Type;

In_Vocab : boolean;

Splice_Count : natural;
Splicing : Boolean;
Splice_Array : array(1..Max_Str_Len,1..2) of Phoneme;

Mid_Splice_Enable : Boolean := false;  -- toggled on to allow mid-word splicing

--***********************************************************************
--* GET_ATOM - Gets next atom from a str
--***********************************************************************

Procedure Get_Atom(In_Str : Str;Atom_Str,Tail_Str : out Str) is
 i,a_start,a_end,a_len,t_len : integer := 1;
Begin
 If In_Str.Len > 0 then
   While (In_Str.Txt(i) = ' ') and (i < In_Str.Len) loop
       i := i + 1;
   end loop;
   If In_Str.Txt(i) /= ' ' then
       a_start := i;
       While (In_Str.Txt(i) /= ' ') and (i < In_Str.Len) loop
        i := i + 1;
       end loop;
       if In_Str.Txt(i) = ' ' then
        a_end := i - 1;
       else
        a_end := i;
       end if;
       a_len := a_end - a_start + 1;
       t_len := In_Str.Len - a_end;
       Tail_Str.Len := t_len;
       Atom_Str.Len := a_len;
       for i in 1..a_len loop
        Atom_Str.Txt(i) := In_Str.Txt(a_start + i - 1);
       end loop;
       for i in 1..t_len loop
        Tail_Str.Txt(i) := In_Str.Txt(a_end + i);
       end loop;
   else
       Atom_Str.Len := 0;
       Tail_Str.Len := 0;
   end if;
 else
  Atom_Str.Len := 0;
  Tail_Str.Len := 0;
```

```
  end if;
end Get_Atom;


--*************************************************************************
--* GET_PHONEME - Returns Phoneme corresponding to an atom (atom must be free
--* of leading & trailing blanks & other garbage (accent digits on end OK)
--*************************************************************************

Function Get_Phoneme (In_Atom : Str) return Phoneme is
 i,Phn_End : Integer;
 Phn_String : String (1..Max_Phn_Len);
 Out_Phoneme : Phoneme;
Begin
 Phn_End := In_Atom.Len;
 if In_Atom.Len > 0 then
   While ((In_Atom.Txt(Phn_End) in '0'..'9')) and (Phn_End > 1) loop
        Phn_End := Phn_End - 1;
   end loop;
 end if;
 If (Phn_End > 0) and (Phn_End <= Max_Phn_Len) then
   Phn_String(1..Phn_End) := In_Atom.Txt(1..Phn_End);
   begin  -- block to catch constraint error if not a valid phoneme
        Out_Phoneme := Phoneme'Value(Phn_String(1..Phn_End));
        return Out_Phoneme;
   exception
        when constraint_error =>
          return XXX;
   end;  -- block
 else
   return XXX;
 end if;
end Get_Phoneme;


--*************************************************************************
--* PUT_PHON - A Put routine that translates from my enum
--* representations to standard symbols
--*************************************************************************

Procedure Put_Phon(O_File : File_Type; Phon : Phoneme) is
begin
 If Phon = HX then
   Put(O_File,"H#");
 elsif Phon = XXX then
   Put(O_File,"???");
 else
   Put(O_File,Phoneme'image(Phon));
 end if;
end Put_Phon;



--*************************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--*************************************************************************

Procedure Get_File(Prompt_String : String;
                    F_Name : out String;
```

209

```
                      F_Mode : File_Mode;
                      F_Handle : in out File_Type) is
     Str_Last : Integer;
     Str_Text : String(1..Max_Str_Len) := (others => ' ');
   begin


     loop    -- loop until valid filename entered or user aborts (^C)
       Begin   -- block statements to allow another try if bad filename entered

           Put(Prompt_String);
           Get_Line(Str_Text,Str_Last);


           if F_Mode = In_File then
             Text_IO.Open(File => F_Handle,
                     Mode => F_Mode,
                     Name => Str_Text(1..Str_Last));
           else
             Text_IO.Create(File => F_Handle,
                     Mode => F_Mode,
                     Name => Str_Text(1..Str_Last));
           end if;

           F_Name := Str_Text;

           exit;

       exception
           When Name_Error => Put(Ascii.Bel);
                           Put_Line("*** Unable to open " & Str_Text(1..Str_Last));
                           New_Line;
           When others => raise;
       end;
     end loop;

   end Get_File;

   --***************************************************************************
   --* INIT_MATRIX - Set matrix cells to one if that diphone is listed in
   --* diplist.txt.
   --***************************************************************************
   Procedure Init_Matrix is

    Input_Line : String(1..Max_Str_Len);
    Input_Last : natural;
    First_Phn_Str,Last_Phn_Str,Line_Str,Tail,Temp : Str;
    Row,Col : Phoneme;

   Begin

    Text_IO.Open(File => DipList_InFile,
                     Mode => In_File,
                     Name => DipList_Filename);

    While NOT End_Of_File(DipList_InFile) loop
```

210

```ada
    Get_Line(DipList_InFile,Input_Line,Input_last);

    Line_Str.Txt(1..Input_Last) := Input_Line(1..Input_Last);
    Line_Str.Len := Input_Last;

    Get_Atom(Line_Str,First_Phn_Str,Tail);  -- fetch first phoneme str

    Get_Atom(Tail,Last_Phn_Str,Temp);       -- fetch last phoneme str


    Row := Get_Phoneme(First_Phn_Str);
    Col := Get_Phoneme(Last_Phn_Str);

    Diphone_Matrix(row,col) := 1;

  end loop;

  Text_IO.Close(DipList_InFile);

end Init_Matrix;

--*************************************************************************
--* VOCAB Main Program
--*************************************************************************

Begin    -- Vocab

 if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                              Arg.Data(3) = "/D") then
   Debugging := True;
 elsif (Arg.Count > 2) and then (Arg.Data(3) = "/m" or   -- process command line
                              Arg.Data(3) = "/M") then
   Mid_Splice_Enable := True;
 elsif Arg.Count > 2 then  -- program name is 1st entry in array
   raise Usage_Error;
 end if;

 New_Line(2);         -- blank lines & program "banner"

 Put_Line(Version);
 New_Line(2);

 if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
   begin
     Text_IO.Open(File => Input_DataFile,
             Mode => In_File,
             Name => Arg.Data(2));
     Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
   exception            -- handle bad filenames from command line
 When Name_Error  I
     Status_error I
     Mode_error   I
     Use_error    I
     Device_error =>  New_Line(3);
                      Put_Line("*** Unable to open " & Arg.Data(2));
```

211

```
                        New_Line;
                        raise Usage_Error;
    end;
else
    Get_File("Enter name of input file: ",
            Input_Filename,
            In_File,
            Input_Datafile);
end if;

                            -- get output filename from user
Get_File("Enter name of output file: ",
        Output_Filename,
        Out_File,
        Output_Datafile);

If Debugging then
    New_Line;
    Put_Line("--------------------[ PROCESSING FILE ]--------------------------");
end if;

Init_Matrix;

While NOT End_Of_File(Input_DataFile) loop

    Get_Line(Input_DataFile,Input_Line,Input_Last);

                                -- if not a comment line
    if Input_Last > 0 and then Input_Line(1) in 'A'..'z' then

        if debugging then
            Put_Line(Input_Line(1..Input_Last));
        end if;

        for i in 1..Input_Last loop      -- remove slashes. don't need them.
            if Input_Line(i) = '/' then
                Input_Line(i) := ' ';
            end if;
        end loop;

        Temp.Txt(1..Input_Last) := Input_Line(1..Input_Last);
        Temp.Len := Input_Last;
        Get_Atom(Temp,Atom,Tail); -- get the word
        Word := Atom;            -- save it
        if Debugging then
            Put_Line("Word = " & Word.Txt(1..Word.Len));
        end if;

        Last_Phoneme := HX;  -- Consider silence & first phoneme to be a diphone

        In_Vocab := true;

        Splicing := false;   -- This flag will go true if must splice around a dip
        Splice_Count := 0;   -- count splices for each word

        loop           -- for each phoneme on the line (in the word)
```

212

```
             Temp := Tail;
             Get_Atom(Temp,Atom,Tail);
             if Atom.Len > 0 then
               This_Phoneme := Get_Phoneme(Atom);

               if Debugging then
                 Put_Line(" Diphone = " &
                       Phoneme'Image(Last_Phoneme) & "/" &
                       Phoneme'Image(This_Phoneme));
               end if;

                       -- include word if composed of diphones in diplist.txt
               if (Diphone_Matrix(Last_Phoneme,This_Phoneme) = 0) and
                 ((Mid_Splice_Enable = true) or (Last_Phoneme = HX)) and
                 (Splicing = false) then    -- splice around if just one missing
                 Splicing := true;
                 Splice_Count := Splice_Count + 1;
                 Splice_Array(Splice_Count,1) := Last_Phoneme;
                 Splice_Array(Splice_Count,2) := This_Phoneme;
               elsif Diphone_Matrix(Last_Phoneme,This_Phoneme) = 0 then
                 In_Vocab := false;        -- can't splice if two diphones missing
               else
                 Splicing := false;        -- not splicing if diphone found
               end if;

               Last_Phoneme := This_Phoneme;
             end if;
             exit when Tail.Len <= 0;
         end loop;

         if Debugging then
             Put_Line(" Diphone = " &
                 Phoneme'Image(Last_Phoneme) & "/" &
                 Phoneme'Image(HX));
         end if;

                         -- Consider last phoneme & silence to be a diphone
  --     if (Diphone_Matrix(Last_Phoneme,HX) = 0) and (Splicing = false) then
  --       Splice_Count := Splice_Count + 1;
  --       Splice_Array(Splice_Count,1) := Last_Phoneme;
  --       Splice_Array(Splice_Count,2) := HX;
  --     elsif Diphone_Matrix(Last_Phoneme,HX) = 0 then
  --       In_Vocab := false;
  --     end if;

                         -- output word if it was composed of diphones in diplist.txt
         if In_Vocab then
             Text_IO.Put(Output_Datafile,Word.Txt(1..Word.Len));
             Text_IO.Put(Output_Datafile,',');
             Text_IO.Put(Output_Datafile,integer'image(splice_count));
             if (Splice_Count > 0) then
               for i in 1..splice_count loop
                 Text_IO.Put(Output_Datafile,',');
                 Put_Phon(Output_Datafile,Splice_Array(i,1));
                 Text_IO.Put(Output_Datafile,'/');
                 Put_Phon(Output_Datafile,Splice_Array(i,2));
```

213

```ada
          end loop;
        end if;
        Text_IO.New_Line(Output_Datafile);
      end if;

    end if;

  end loop;

  If Debugging then
    Put_Line("------------------------[ DONE ]----------------------------");
    New_Line;
  end if;

  Text_IO.Close(Input_Datafile);
  Text_IO.Close(Output_Datafile);

exception

  When Usage_Error =>
    Put(Ascii.Bel);
    New_Line(3);
    Put_Line("Usage:  Vocab infilename [-d]");
    New_Line;
    Put_Line("      /d = enable debugging");
    Put_Line("      /m = enable mid-word splices");
    New_Line;
    return;

end Vocab;
```

# H. MAKWLIST.ADA

```
--***********************************************************************
--***********************************************************************
--** PROGRAM NAME: MakWList  (MakWList.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a list of TIMIT *.WRD
--** files & generating a new list with occurences of each word
--** contained in a vacabulary file
--**
--** DATE MODIFIED: 21 Jan, 1996  (see below for version number)
--**
--***********************************************************************
--***********************************************************************


with Arg,
    Cody_Math,
    Text_IO;

use Text_IO;

procedure MakWList is

  package Long_Int_IO is new Text_IO.Integer_IO(Long_Integer); use Long_Int_IO;


  Max_Str_Len : constant := 255;
  Max_Word_Len : constant := 20;
  Max_Num_Words : constant := 1000;

  Vocab_Filename : constant string := "VOCAB.TXT";

  Usage_Error : exception;

  SubType Word_Str is String(1..Max_Word_Len);
  Vocab_Array : array(1..Max_Num_Words) of Word_Str :=
                            (others => (others => ' '));

  Debugging : boolean := false;

  Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
  Name_Last : Natural;

  Input_Line : string(1..Max_Str_Len);
  Input_Last : natural;

  Version : constant String := "MakWList Version #1.0";

  Input_DataFile,Output_DataFile : Text_IO.File_Type;

  Num_Words : natural := 0;
```

```
--*****************************************************************
--* UPPER_CASE - Returns upper case version of input string
--*****************************************************************
Function Upper_Case(In_String : string) return string is
 Result_String : String(In_String'first..In_String'last);
Begin
 for index in In_String'first..In_String'last loop
   If In_String(index) in 'a'..'z' then
Result_String(index) := character'val(character'pos(In_String(index))
                    - character'pos('a')
                    + character'pos('A'));
   else
Result_String(index) := In_String(index);
   end if;

 end loop;
 return Result_String;
end Upper_Case;


--*****************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--*****************************************************************

Procedure Get_File(Prompt_String : String;
                   F_Name : out String;
                   F_Mode : File_Mode;
                   F_Handle : in out File_Type) is
 Str_Last : Integer;
 Str_Text : String(1..Max_Str_Len) := (others => ' ');
begin


 loop    -- loop until valid filename entered or user aborts (^C)
   Begin   -- block statements to allow another try if bad filename entered

       Put(Prompt_String);
       Get_Line(Str_Text,Str_Last);


       if F_Mode = In_File then
         Text_IO.Open(File => F_Handle,
               Mode => F_Mode,
               Name => Str_Text(1..Str_Last));
       else
         Text_IO.Create(File => F_Handle,
               Mode => F_Mode,
               Name => Str_Text(1..Str_Last));
       end if;

       F_Name := Str_Text;

       exit;

   exception
       When Name_Error => Put(Ascii.Bel);
                       Put_Line("*** Unable to open " & Str_Text(1..Str_Last));
```

216

```
                        New_Line;
        When others => raise;
    end;
  end loop;

end Get_File;

--*********************************************************************
--* IN_VOCAB - Determine if a word is in the vocabulary
--*********************************************************************

Function In_Vocab(Key : String) return boolean is

 mid : natural;
 s : natural := 1;
 e : natural := Num_Words;
 found : boolean := false;

begin

 If debugging then
   Text_IO.Put("Checking for: <");
   Text_IO.Put(Key);
   Text_IO.Put_Line(">");
 end if;

 while (found = false) and (S <= E) loop
   Mid := integer(Cody_Math.round(float(S+E)/2.0));

   If debugging then
       Text_IO.Put("Comparing to: <");
       Text_IO.Put(Vocab_Array(Mid));
       Text_IO.Put_Line(">");
   end if;

   If Vocab_Array(mid) = Key then
       found := true;
   elsif Key <= Vocab_Array(mid) then
       e := Mid - 1;
   else
       s := Mid + 1;
   end if;
 end loop;

 If found then
   return true;
 else
   return false;
 end if;

end In_Vocab;

--*********************************************************************
--*  PROCESS_LINE - Process line from wrd file list & output if that word
--*  is in vocabulary
--*********************************************************************
```

217

```
Procedure Process_Line(File_Line,Wav_Filename : string) is

  Word_Read : String(1..Max_Word_Len) := (others => ' ');
  Start_Sample,End_Sample : Long_Integer;
  Index_1,Index_2,Word_Index : natural;

begin
  Long_Int_Io.Get(File_Line,Start_Sample,Index_1);
  Long_Int_Io.Get(File_Line(Index_1+1..File_Line'length),End_Sample,Index_2);

  Word_Index := 0;
  For i in Index_2+1..File_Line'length loop
   if File_Line(i) /= ' ' then
        Word_Index := Word_Index + 1;
        Word_Read(Word_Index) := File_Line(i);
   end if;
  end loop;

  If debugging then
   Text_IO.Put("File_Line is: <");
   Text_IO.Put(File_Line);
   Text_IO.Put_Line(">");
  end if;

  if In_Vocab(Upper_Case(Word_Read)) then
   Text_Io.Put(Output_Datafile,Wav_Filename);
   Text_Io.Put(Output_Datafile,' ');
   Text_Io.Put(Output_Datafile,Word_Read(1..Word_Index));
   Text_Io.Put(Output_Datafile,' ');
   Long_Int_Io.Put(Output_Datafile,Start_Sample,0);
   Text_Io.Put(Output_Datafile,' ');
   Long_Int_Io.Put(Output_Datafile,End_Sample,0);
   Text_IO.New_Line(Output_Datafile);
  end if;

end Process_Line;

--******************************************************************************
--*  PROCESS_WRD_FILE - Process lines from a word file: pass each line
--*  of each wrd file to process_line for vocab check & possible output
--******************************************************************************
Procedure Process_Wrd_File(Wrd_Filename : String) is

  Dot_Loc : natural := 0;
  Wrd_Infile : Text_IO.File_Type;
  Word_Line : String(1..Max_Str_Len);
  Line_Last : natural;
  WAV_Filename : String(1..Wrd_Filename'length) := Wrd_Filename;

begin

  Text_IO.Open(File => Wrd_InFile,
          Mode => In_File,
          Name => Wrd_Filename);

  loop           -- determine WAV filename
```

218

```
    dot_loc := dot_loc + 1;
    exit when (Dot_Loc > Wrd_Filename'length) or else (Wrd_Filename(Dot_Loc) = '.') ;
  end loop;
  WAV_Filename(Dot_Loc+1..Dot_Loc+3) := "WAV";

                       -- process all lines in WRD file
  while NOT End_of_File(Wrd_InFile) loop
    Text_Io.Get_Line(Wrd_InFile,Word_Line,Line_Last);
    Process_Line(Word_Line(1..Line_Last),WAV_Filename);
  end loop;

  Text_IO.Close(Wrd_Infile);

end Process_Wrd_File;


--*********************************************************************
--* PROCESS_FILELIST - Process lines from a file list file & pass each line
--*  of each wrd file to process_wrd_file
--*********************************************************************
Procedure Process_Filelist(FileList_Infile : Text_IO.File_Type) is
  FileName_Line : String(1..Max_Str_Len);
  Filename_Last : Natural;
begin

                       -- process all lines in file list file
  while NOT End_of_File(FileList_InFile) loop
    Text_Io.Get_Line(FileList_InFile,Filename_Line,Filename_Last);

    If debugging then
        Text_IO.Put("File Name is: <");
        Text_IO.Put(Filename_Line(1..Filename_Last));
        Text_IO.Put_Line(">");
    end if;


    Process_WRD_File(Filename_Line(1..Filename_Last));
  end loop;

end Process_FileList;


--*********************************************************************
--* Read_Vocab - Read words from vocab file into vocab array
--*********************************************************************

Procedure Read_Vocab is

  Vocab_Infile : Text_IO.File_Type;
  Word_Line : String(1..Max_Str_Len);
  Word_Last,Word_End : Natural;
  Word_Read : String(1..Max_Word_Len);

begin

  Text_IO.Open(File => Vocab_InFile,
          Mode => In_File,
          Name => Vocab_Filename);
```

219

```
                    -- process all lines in WRD file
    while NOT End_of_File(Vocab_InFile) loop
      Text_Io.Get_Line(Vocab_InFile,Word_Line,Word_Last);
      Word_Read := (others => ' ');
      Word_End := 0;
      loop            -- determine word read
          word_end := word_end + 1;
          exit when (Word_End > Max_Word_Len) or else
                  (Word_End > Word_Last) or else
                  (Word_Line(Word_End) = ',');
          Word_Read(Word_End) := Word_Line(Word_End);
      end loop;
      Num_Words := Num_Words + 1;
      Vocab_Array(Num_Words) := Upper_Case(Word_Read);
    end loop;

    Text_IO.Close(Vocab_InFile);

    If debugging then
      Text_IO.Put_Line("Vocab is:");
      For i in 1..Num_Words loop
          Text_IO.Put_Line(Vocab_Array(i));
      end loop;
    end if;

  end Read_Vocab;

  --*************************************************************************
  --*  MakWList Main Program
  --*************************************************************************

Begin    -- MakWList

  if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                              Arg.Data(3) = "/D" or
                              Arg.Data(3) = "-d" or
                              Arg.Data(3) = "-D" or
                              Arg.Data(3) = "d"  or
                              Arg.Data(3) = "D") then
    Debugging := True;
  elsif Arg.Count > 2 then  -- note that program name is 1st entry in array
    raise Usage_Error;
  end if;

  New_Line(2);          -- blank lines & program "banner"

  Put_Line(Version);
  New_Line(2);

  if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
    begin
      Text_IO.Open(File => Input_DataFile,
              Mode => In_File,
              Name => Arg.Data(2));
      Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
    exception            -- handle bad filenames from command line
```

```
When Name_Error   |
   Status_error |
   Mode_error   |
   Use_error    |
   Device_error =>   New_Line(3);
                           Put_Line("*** Unable to open " & Arg.Data(2));
                           New_Line;
                           raise Usage_Error;
   end;
else
   Get_File("Enter name of input file: ",
            Input_Filename,
            In_File,
            Input_Datafile);
end if;

                              -- get output filename from user
Get_File("Enter name of output file: ",
         Output_Filename,
         Out_File,
         Output_Datafile);

If Debugging then
   New_Line;
   Put_Line("----------------------[ READING VOCAB.TXT ]----------------------------");
end if;

Read_Vocab;

If Debugging then
   Put_Line("--------------------[ DONE READING VOCAB.TXT ]-----------------------------");
   New_Line(2);
   Put_Line("----------------------[ PROCESSING FILE LIST ]----------------------------");
end if;

Process_Filelist(Input_Datafile);

If Debugging then
   Put_Line("-----------------------------[ DONE ]------------------------------");
   New_Line;
end if;

Text_IO.Close(Input_Datafile);
Text_IO.Close(Output_Datafile);

exception

When Usage_Error =>
   Put(Ascii.Bel);
   New_Line(3);
   Put_Line("Usage:  MakWList infilename [-d]");
   New_Line;
   Put_Line("       -d = enable debugging");
   New_Line;
   return;
end MakWList;
```

221

# I.    PHNDURS.ADA

```
--*********************************************************************
--*********************************************************************
--** PROGRAM NAME: PhnDurs  (PhnDurs.ada)
--**
--** AUTHOR: Maj Mark Cantrell, USMC (cantrell@nps.navy.mil)
--**
--** DESCRIPTION: Program for reading a list of phonetic transcription
--** files & recording max diphone durations listed
--**
--** DATE MODIFIED: 17 Sep, 1995  (see below for version number)
--**
--*********************************************************************
--*********************************************************************

with Arg,
     Text_IO;

use Text_IO;

procedure PhnDurs is

  Max_Str_Len : constant := 255;
  Max_Phn_Len : constant := 5;
  Col_Widths : constant := 6;
  Fs : constant := 16000.0;
  FList_Filename : constant string := "DIPFLIST.TXT";

  Usage_Error : exception;

  Type STR is record
    Txt : String(1..Max_Str_Len);
    Len : Integer range 0..Max_Str_Len := 0;
  end record;


  Type Phn_Rec is record
    Min : integer := integer'last;
    Max : integer := 0;
    Sum : float := 0.0;
    Count : Integer := 0;
  end record;

  Temp,Tail,Atom,Atom_Str,Tail_Str,Phon_Str : STR;


  Type Phoneme is (AA,AE,AH,AO,AW,AY,B,CH,D,DH,EH,ER,EY, -- 1st 3 lines from CMU lex
              F,G,HH,IH,IY,JH,K,L,M,N,NG,OW,OY,P,R,
              S,SH,T,TH,UH,UW,V,W,Y,Z,ZH,
              EM,EN,ENG,EL,AX,IX,AXR,HX,        -- this line from TIMIT
              XXX);                             -- use XXX for unknown codes

  Phoneme_Array : array(Phoneme) of Phn_Rec;
```

```
--  Max_Dip_X,Max_Dip_Y : Phoneme := HX;
  Max_Duration : Integer := 0;
  Max_Fname : Str;

  Debugging : boolean := false;

  Input_Filename,Output_FileName : string(1..Max_Str_Len) := (others => ' ');
  Name_Last : Natural;

  Input_Line : string(1..Max_Str_Len);
  Input_Last : natural;

  Version : constant String := "PhnDurs Version #1.0";

  Input_DataFile,Output_DataFile,FList_File : Text_IO.File_Type;


--***************************************************************************
--* UPPER_CASE - Returns upper case version of input string
--***************************************************************************
Function  Upper_Case(In_String : string) return string is
  Result_String : String(In_String'first..In_String'last);
Begin
  for index in In_String'first..In_String'last loop
    If In_String(index) in 'a'..'z' then
  Result_String(index) := character'val(character'pos(In_String(index))
                          - character'pos('a')
                          + character'pos('A'));
    else
  Result_String(index) := In_String(index);
    end if;

  end loop;
  return Result_String;
end Upper_Case;


--***************************************************************************
--* GET_ATOM - Gets next atom from a str
--***************************************************************************

Procedure Get_Atom(In_Str : Str;Atom_Str,Tail_Str : out Str) is
  i,a_start,a_end,a_len,t_len : integer := 1;
Begin
  If In_Str.Len > 0 then
    While (In_Str.Txt(i) = ' ') and (i < In_Str.Len) loop
        i := i + 1;
    end loop;
    If In_Str.Txt(i) /= ' ' then
        a_start := i;
        While (In_Str.Txt(i) /= ' ') and (i < In_Str.Len) loop
          i := i + 1;
        end loop;
        if In_Str.Txt(i) = ' ' then
          a_end := i - 1;
        else
          a_end := i;
```

223

```
        end if;
        a_len := a_end - a_start + 1;
        t_len := In_Str.Len - a_end;
        Tail_Str.Len := t_len;
        Atom_Str.Len := a_len;
        for i in 1..a_len loop
          Atom_Str.Txt(i) := In_Str.Txt(a_start + i - 1);
        end loop;
        for i in 1..t_len loop
          Tail_Str.Txt(i) := In_Str.Txt(a_end + i);
        end loop;
    else
        Atom_Str.Len := 0;
        Tail_Str.Len := 0;
    end if;
  else
    Atom_Str.Len := 0;
    Tail_Str.Len := 0;
  end if;
end Get_Atom;


--***********************************************************************
--* GET_PHONEME - Returns Phoneme corresponding to an atom (atom must be free
--* of leading & trailing blanks & other garbage (accent digits on end OK)
--***********************************************************************

Function Get_Phoneme (In_Atom : Str) return Phoneme is
  i,Phn_End : Integer;
  Phn_String : String (1..Max_Phn_Len);
  Out_Phoneme : Phoneme;
Begin
  Phn_End := In_Atom.Len;
  if In_Atom.Len > 0 then
    While ((In_Atom.Txt(Phn_End) in '0'..'9')) and (Phn_End > 1) loop
        Phn_End := Phn_End - 1;
    end loop;
  end if;
  If (Phn_End > 0) and (Phn_End <= Max_Phn_Len) then
    Phn_String(1..Phn_End) := Upper_Case(In_Atom.Txt(1..Phn_End));

    begin  -- block to catch constraint error if not a valid phoneme
        Out_Phoneme := Phoneme'Value(Phn_String(1..Phn_End));
        return Out_Phoneme;
    exception
        when constraint_error =>
          if Phn_End = 2 and then Phn_String(1..2) = "H#" then
            return HX;
          else
            return XXX;
          end if;
    end;  -- block

  else
    return XXX;
  end if;
end Get_Phoneme;
```

224

```
--*************************************************************************
--* ECHO_DATA - Used by Process_Phn_File to echo the data read from file
--*************************************************************************

    Procedure Echo_Data (Start_N, End_N : Long_Integer;
                         Phon_Str : Str;
                         Phoneme_Read : Phoneme) is

    begin

      if Debugging then
        Put_Line(" Start  = " & long_integer'image(Start_N));
        Put_Line(" End    = " & long_integer'image(End_N));
        Put_Line(" Phoneme  '" & Phon_Str.Txt(1..Phon_Str.Len) &
                 "' = " & Phoneme'image(Phoneme_Read));
      end if;

    end Echo_Data;

--*************************************************************************
--* PUT_PHON - A Put routine that translates from my enum
--* representations to standard symbols.  Used by Display_Array.
--*************************************************************************

    Procedure Put_Phon(O_File : File_Type; Phon : Phoneme) is
    begin
      If Phon = HX then
        Put(O_File,"H#");
      elsif Phon = XXX then
        Put(O_File,"???");
      else
        Put(O_File,Phoneme'image(Phon));
      end if;
    end Put_Phon;

--*************************************************************************
--* Display_Array - Prints final results to output file
--*************************************************************************

    Procedure Display_Array is
      row : Phoneme;
      avg : float;
      count : integer;
    begin

      Put_Line(Output_Datafile,"*************************************************************************************");
      Put_Line(Output_Datafile,"                    MAX PHONEME DURATIONS FOR FILES LISTED IN " &
    Input_Filename);
      Put_Line(Output_Datafile,"*************************************************************************************");

      new_line(Output_Datafile);

      for row in Phoneme_Array'range loop

        count := Phoneme_Array(row).count;
        If count > 0 then
```

225

```
            avg := Phoneme_Array(row).sum/float(Phoneme_Array(row).count);

            Put(Output_Datafile,"(");

            Put_Phon(Output_Datafile,Row);   -- phn name

            Put(Output_Datafile," (");
            Put(Output_Datafile,integer'image(Phoneme_Array(row).Min));
            Put(Output_Datafile," ");
            Put(Output_Datafile,integer'image(Phoneme_Array(row).Max));
            Put(Output_Datafile," ");
            Put(Output_Datafile,integer'image(integer(avg)));
            Put(Output_Datafile,")");

            Put(Output_Datafile,")");

            new_line(Output_Datafile);

      else

            Put(Output_Datafile,"(");
            Put_Phon(Output_Datafile,Row);
            Put(Output_Datafile," (nil nil nil))");
            New_Line(Output_Datafile);

      end if;
   end loop;

end Display_Array;

--*************************************************************************
--* GET_FILE - Prompts user for filename, opens file, & returns name & handle
--*************************************************************************

Procedure Get_File(Prompt_String : String;
                   F_Name : out String;
                   F_Mode : File_Mode;
                   F_Handle : in out File_Type) is
  Str_Last : Integer;
  Str_Text : String(1..Max_Str_Len) := (others => ' ');
begin


  loop    -- loop until valid filename entered or user aborts (^C)
    Begin   -- block statements to allow another try if bad filename entered

         Put(Prompt_String);
         Get_Line(Str_Text,Str_Last);


         if F_Mode = In_File then
           Text_IO.Open(File => F_Handle,
                 Mode => F_Mode,
                 Name => Str_Text(1..Str_Last));
         else
           Text_IO.Create(File => F_Handle,
```

226

```
                    Mode => F_Mode,
                    Name => Str_Text(1..Str_Last));
        end if;

        F_Name := Str_Text;

        exit;

  exception
     When Name_Error => Put(Ascii.Bel);
                        Put_Line("***  Unable to open " & Str_Text(1..Str_Last));
                        New_Line;
     When others => raise;
  end;
 end loop;

end Get_File;


--***********************************************************************
--*  RECORD_DURATION - Used by Process_Phn_File to record a phoneme's duration
--*  in Phoneme_Array.
--***********************************************************************

Procedure Record_Duration (Start_N,End_N : Long_Integer;
                           Phn : Phoneme) is
 Phn_Duration : Integer;
 m_Sec : Integer;
 Local_FName : Str;

begin

 if End_N - Start_N < long_integer(integer'last) then
   Phn_Duration := integer(End_N - Start_N);
 else
   Phn_Duration := integer'last;
 end if;

 if Phn_Duration > Phoneme_Array(Phn).Max then
   Phoneme_Array(Phn).Max := Phn_Duration;
 end if;

  if Phn_Duration < Phoneme_Array(Phn).Min then
   Phoneme_Array(Phn).Min := Phn_Duration;
 end if;

 Phoneme_Array(Phn).Sum := Phoneme_Array(Phn).Sum + float(Phn_Duration);
 Phoneme_Array(Phn).Count := Phoneme_Array(Phn).Count + 1;

end Record_Duration;


--***********************************************************************
--*  PROCESS_PHN_FILE - Opens designated file & records diphone durations
--***********************************************************************
Procedure Process_Phn_File (Phn_Filename : string) is

 Phn_File : Text_IO.File_Type;
```

```
          Temp,Atom,Tail,This_PhnAtom,Phn_FName : Str;
          This_Start,This_End : long_integer;
          Phn_Line : string(1..Max_Str_Len);
          Phn_Last : natural;
          This_Phoneme : Phoneme;

     begin
                              -- set up filename str for Record_Duration calls
          Phn_FName.Txt(1..Phn_Filename'length) := Phn_Filename;
          Phn_FName.Len := Phn_Filename'length;

          Text_IO.Open(File => Phn_File,
                       Mode => In_File,
                       Name => Phn_Filename);

          get_line(Phn_File,Phn_Line,Phn_Last);

          if Phn_Last > 5 then

             While NOT End_Of_File(Phn_File) loop

                   if debugging then
                     Put_Line(Phn_Line(1..Phn_Last));
                   end if;

                   Temp.Txt(1..Phn_Last) := Upper_Case(Phn_Line(1..Phn_Last));
                   Temp.Len := Phn_Last;

                   Get_Atom(Temp,Atom,Tail); -- get the start N
                   This_Start := Long_Integer'value(Atom.Txt(1..Atom.Len));

                   Temp := Tail;
                   Get_Atom(Temp,Atom,Tail); -- get the ending N
                   This_End := Long_Integer'value(Atom.Txt(1..Atom.Len));

                   Temp := Tail;
                   Get_Atom(Temp,Atom,Tail); -- get the phoneme string
                   This_Phoneme := Get_Phoneme(Atom);
                   This_PhnAtom := Atom;

                   Echo_Data (This_Start,This_End,Atom,This_Phoneme);

                   get_line(Phn_File,Phn_Line,Phn_Last);

                   Record_Duration(This_Start,This_End,This_Phoneme);

             end loop;

          end if;

          Close(Phn_File);

     exception

       When Name_Error | Use_Error | Device_Error =>
                   New_Line;
```

```
                    Put_Line("***  Unable to open PHN file " & Phn_Filename);
                    New_Line;

end Process_Phn_File;


--***********************************************************************
--*  PhnDurs Main Program
--***********************************************************************


Begin   -- PhnDurs

   if (Arg.Count > 2) and then (Arg.Data(3) = "/d" or   -- process command line
                                Arg.Data(3) = "/D" or
                                Arg.Data(3) = "-d" or
                                Arg.Data(3) = "-D" or
                                Arg.Data(3) = "d"  or
                                Arg.Data(3) = "D") then
     Debugging := True;
   elsif Arg.Count > 2 then  -- note that program name is 1st entry in array
     raise Usage_Error;
   end if;

   New_Line(25);           -- blank lines & program "banner"

   Put_Line(Version);
   New_Line(2);

   if (Arg.Count > 1) then   -- if input filename on cmd line, open it. else ask
     begin
       Text_IO.Open(File => Input_DataFile,
               Mode => In_File,
               Name => Arg.Data(2));
       Input_Filename(1..Arg.Data(2)'length) := Arg.Data(2);
      exception           -- handle bad filenames from command line
     When Name_Error  |
        Status_error |
        Mode_error   |
        Use_error    |
        Device_error =>   New_Line(3);
                          Put_Line("***  Unable to open " & Arg.Data(2));
                          New_Line;
                          raise Usage_Error;
     end;
   else
     Get_File("Enter name of input file: ",
            Input_Filename,
            In_File,
            Input_Datafile);   .
   end if;

                           -- get output filename from user
   Get_File("Enter name of output file: ",
          Output_Filename,
          Out_File,
          Output_Datafile);
```

```
If Debugging then
  New_Line;
  Put_Line("---------------------[ PROCESSING FILE ]----------------------------");
end if;

While NOT End_Of_File(Input_DataFile) loop

  Get_Line(Input_DataFile,Input_Line,Input_Last);

  Input_Line := Upper_Case(Input_Line);

  if Input_Last > 0 then

    If Debugging then
        Put_Line("#### " & Input_Line(1..Input_Last) & " ####");
    end if;

    Process_Phn_File(Input_Line(1..Input_Last));

    If Debugging then
        Put_Line("##########################################");
    end if;

  end if;

end loop;

Display_Array;

If Debugging then
  Put_Line("------------------------[ DONE ]-----------------------------");
  New_Line;
end if;

Text_IO.Close(Input_Datafile);
Text_IO.Close(Output_Datafile);

exception

When Usage_Error =>
  Put(Ascii.Bel);
  New_Line(3);
  Put_Line("Usage:  PhnDurs infilename [-d]");
  New_Line;
  Put_Line("      -d = enable debugging");
  New_Line;
  return;

end PhnDurs;
```

# APPENDIX C. LISP CODE

## A.   CONTENTS

### 1. RECOGNIZ.CL

This is the main lexical analyzer code.  It reads a list of diphone detections (produced by RECOGNIZ.M) and outputs a list of word candidates.  Note that this code is not limited to the diphone list of Appendix D or the vocabulary of Appendix E.  The entire TIMIT lexicon is scanned by the lexical analyzer (and a larger lexicon may easily be substituted).  If some of a word's diphones are not among the neural network's output classes, the lexical analyzer will simply fail to ever find those diphones in its input stream.  Appendix E contains additional comments on this issue.

### 2. PEAKS.CL

This file contains functions for removing all but the output peaks from the input file used by RECOGNIZ.CL.

### 3. DICT.CL

This file contains phonetic dictionary utilities.

### 4. LEXICON.TXT

This is the TIMIT lexicon converted to a LISP list.  The complete file is too large to include in this appendix.  So an extract is provided as a sample of the format used.

### 5. LEXICON2.TXT

This is the TIMIT lexicon converted to a LISP list.  This version is sorted by the second phoneme in each word.  The complete file is too large to include in this appendix.  So an extract is provided as a sample of the format used.

### 6. PHNINDEX TXT

231

This is an index containing each phoneme (45 entries) and the file position of the first word in LEXICON.TXT that starts with that phoneme.

### 7. PHNINDX2 TXT

This is an index containing each phoneme (45 entries) and the file position of the first word in LEXICON2.TXT that has that phoneme as its second phoneme.

### 8. PHNDURS TXT

This is a list of phonemes and their minimum, average, and maximum expected durations. It was produced by PHNDURS.ADA and is used by RECOGNIZ.CL to bound its search for phonemes in a word.

### B.   RECOGNIZ.CL

```
;******************************************************************************
;******************************************************************************
;
;
; RECOGNIZ.CL - This program converts a diphone peak list into word
; candidates.
;
;
; Written by Maj Mark E. Cantrell, USMC
;
; Version 1.3; 7 June 1996
;
;******************************************************************************
;******************************************************************************
;


;******************************************************************************
; GLOBALS
;******************************************************************************
;

(defvar *diphone_peak_list* nil)      ; peaks list produced by dip2peak
(defvar *phoneme_index* nil)          ; lexicon index by 1st phoneme
(defvar *phoneme_index2* nil)         ; lexicon index by 2nd phoneme
(defvar *phn_durs* nil)               ; list of max & min phn durations
(defvar *dict_file* nil)              ; lexicon sorted by 1st phn
(defvar *dict_file2* nil)             ; lexicon sorted by 2nd phn
(defvar *word_candidates* nil)        ; words from peaks_to_words
```

```
(defvar *last_phns* nil)          ; phns located in last build_word call
(defvar *last_peaks* nil)         ; peaks located in last build_word call
(defvar *failed_phn* nil)         ; phn that last build_word call failed on

(defconstant *dummy_confidence* 0.0)  ; confidence assigned to dummy diphones
(defconstant *min_conf* 0.0005)       ; word confidence must be > this to list
(defconstant *min_peak* 0.0005)    ; start peaks with lower confidence are ignored
(defconstant *win_rate* 170)          ; this number of samples between (interleaved)
                                      ; window positions
;***********************************************************************
; LOAD SUPPORT CODE
;***********************************************************************

(load "peaks")   ; functions for converting NNET outputs to diphone peaks
(load "dict")    ; phonetic dictionary functions


;***********************************************************************
;
;***********************************************************************
;
;
; KEY FUNCTIONS FOR TURNING DIPHONE PEAKS INTO WORD CANDIDATES
;
;***********************************************************************
;
;***********************************************************************
;


;***********************************************************************
;
; PEAKS_TO _WORDS - Try each dip in peak list (recursively) to see if it
; starts a word
;***********************************************************************
;

(defun peaks_to_words (peak_list)
   (cond ((null peak_list) nil)
         (t (if (>= (third (first peak_list)) *min_peak*)
              (findwords (first peak_list) (rest peak_list)))
            (peaks_to_words (rest peak_list)))))


;***********************************************************************
;
; FINDWORDS - Find all possible words in peak list starting with given
; diphone peak
;***********************************************************************
;

(defun findwords (start_peak rest_peak_list)
   (cond ((equal (first_phn start_peak) 'HX)

             ; if detected silence at start of word, chain from start_peak
             (setf *word_candidates*
```

```
                (append *word_candidates*
                        (build_words_I start_peak rest_peak_list))))

        ; else look for words starting w/2nd phn in start_peak (no word gap)
        (t (setf *word_candidates*
                (append *word_candidates*
                        (build_words_I start_peak rest_peak_list)))

        ; and look for words starting with 1st phn in start_peak
        (setf *word_candidates*
                (append *word_candidates*
                        (build_words_II start_peak rest_peak_list)))

        ; and words with 2nd phn = 1st phn in start_peak (a "splice")
        (setf *word_candidates*
                (append *word_candidates*
                        (build_words_III start_peak rest_peak_list))))))
```

```
;***********************************************************************
; BUILD_WORDS_I - Look for all words in peak list starting from a
; silence-first dip
;***********************************************************************

(defun build_words_I (start_peak rest_peaks)
  (let ((file_pos (index_pos (second_phn start_peak) *phoneme_index*))
        (word_entry nil)
        (nw nil)
        (rl nil))

        ; position lexicon file pointer at 1st word starting with 1st phn
   (cond (file_pos

        (file-position *dict_file* file_pos)
        (setf word_entry (read *dict_file* nil nil nil))

        (loop  ; attempt to confirm each word starting with 1st phn
         (when (not (equal (second word_entry) (second_phn start_peak)))
              (return rl))          ; exit when no more words in dictionary
                                    ; start with this phn

;           (if (not_already_found word_entry *word_candidates* start_peak)
                (setf nw (build_word (first word_entry)        ; word candidate
                                 (list (second word_entry)) ; phns found so far
                                 (rest (rest word_entry))   ; phns yet to be found
                                 (list start_peak)        ; peaks used so far
                                 rest_peaks))             ; rest of peak list
```

234

```
;            (setf nw nil))

            (cond ((and (not (null nw)) (>= (confidence_in nw) *min_conf*))
                   (setf rl (append rl (list (print nw))))
                   (setf *last_peaks* (second nw))
                   (setf *last_phns* (rest word_entry))
                   (setf *failed_phn* nil))
                  ((not (null nw))
                   (setf *last_peaks* (second nw))
                   (setf *last_phns* (rest word_entry))
                   (setf *failed_phn* nil)))

            (setf word_entry (read *dict_file* nil nil nil)))))))
```

```
;**********************************************************************
;
; BUILD_WORDS_II - Look for all words in peak list starting with a given
; phoneme
;**********************************************************************
;

(defun build_words_II (start_peak rest_peaks)
  (let* ((word_entry nil)
         (phn1 (first_phn start_peak))
         (phn2 (second_phn start_peak))
         (file_pos (index_pos phn1 *phoneme_index*))
         (nw nil)
         (rl nil))

        ; position lexicon file pointer at 1st word starting with 1st phn
    (cond (file_pos

           (file-position *dict_file* file_pos)
           (setf word_entry (read *dict_file* nil nil nil))

           (loop  ; scan forward in dictionary till 2nd phn of word = 2nd phn of dip

             (when (or (equal (third word_entry) phn2)
                       (not (equal (second word_entry) phn1)))
               (return nil))
             (setf word_entry (read *dict_file* nil nil nil)))

           (loop  ; attempt to confirm each word starting with 1st phn
             (when (not (equal (third word_entry) phn2))
               (return rl))             ; exit when no more words in dictionary
                                        ; start with these 2 phns

        ; The next if clause is commented out because it turned out to be very
```

235

```
                    ; expensive to determine of the same instance of the same word was already
in
                    ; the output list.  Seems better to just process the word & let higher levels
                    ; of analysis sort out the duplications
;           (if (not_already_found word_entry *word_candidates* start_peak)
                    (setf nw (build_word (first word_entry)        ; word candidate
                                    (list phn1 phn2)            ; phns found so far
                                    (rest (rest (rest word_entry))) ; phns yet to be found
                                    (list start_peak)           ; peaks used so far
                                    rest_peaks))                ; rest of peak list
;               (setf nw nil))

                    (cond ((and (not (null nw)) (>= (confidence_in nw) *min_conf*))
                            (setf rl (append rl (list (print nw))))
                            (setf *last_peaks* (second nw))
                            (setf *last_phns* (rest word_entry))
                            (setf *failed_phn* nil))
                        ((not (null nw))
                            (setf *last_peaks* (second nw))
                            (setf *last_phns* (rest word_entry))
                            (setf *failed_phn* nil)))

                    (setf word_entry (read *dict_file* nil nil nil))))

            (t nil))))


;***********************************************************************
;  BUILD_WORDS_III - Look for all words in peak list with a given 2nd
;  phoneme
;***********************************************************************

(defun build_words_III (start_peak rest_peaks)
  (let* ((word_entry nil)
        (phn1 (first_phn start_peak))
        (phn2 (second_phn start_peak))
        (file_pos (index_pos phn1 *phoneme_index2*))
        (nw nil)
        (rl nil))

        ; position lexicon file pointer at 1st word with given 2nd phn
    (cond (file_pos

            (file-position *dict_file2* file_pos)
            (setf word_entry (read *dict_file2* nil nil nil))

            (loop  ; scan forward in dictionary till 3rd phn of word = 3rd phn of dip
```

236

```
              (when (or (equal (fourth word_entry) phn2)
                        (not (equal (third word_entry) phn1)))
                    (return))
              (setf word_entry (read *dict_file2* nil nil nil)))

          (loop ; attempt to confirm each word starting with 1st phn
            (when (or (not (equal (fourth word_entry) phn2))
                      (not (equal (third word_entry) phn1)))
                  (return rl))          ; exit when no more words in dictionary
                                        ; have these 2 phns in 2nd & 3rd pos

;            (if (not_already_found word_entry *word_candidates* start_peak)
                (setf nw (build_word (first word_entry)     ; word candidate
                                     (list (second word_entry) phn1 phn2)   ; phns found so
far
                                     (nthcdr 3 word_entry)   ; phns yet to be found
                                                             ; peaks used so far
                                     (list (dummy_peak (second word_entry) phn1)
start_peak)
                                     rest_peaks))            ; rest of peak list

;              (setf nw nil))

            (cond ((and (not (null nw)) (>= (confidence_in nw) *min_conf*))
                   (setf rl (append rl (list (print nw))))
                   (setf *last_peaks* (second nw))
                   (setf *last_phns* (rest word_entry))
                   (setf *failed_phn* nil))
                  ((not (null nw))
                   (setf *last_peaks* (second nw))
                   (setf *last_phns* (rest word_entry))
                   (setf *failed_phn* nil)))
            (setf word_entry (read *dict_file2* nil nil nil))))
          (t nil))))


;**********************************************************************
; BUILD_WORD - Take advantage of phns & peaks previously found (and needed
; by this word) before looking for rest of phonemes required
;**********************************************************************

(defun build_word (test_word phns_found phns_needed peaks_used rest_peaks)
  (let ((phns_found2 phns_found)
        (phns_needed2 phns_needed)
        (peaks_used2 peaks_used)
        (last_phns2 *last_phns*)
        (last_peaks2 *last_peaks*)
```

237

```
        (recycled_phns nil)
        (matches 0))

(cond    ; if this word started at same peak as last
  ((equal (first peaks_used) (first *last_peaks*))
   (dolist (phn phns_found) ; see if phns so far match phns from last word
       (when (not (equal phn (first last_phns2))) (return))
       (setf last_phns2 (rest last_phns2))
       (setf matches (+ matches 1)))
           ; if phns match so far, look for more usable phns in last word
   (cond ((equal matches (length phns_found))
          (dolist (phn last_phns2) ; recycle phns already proven in last word search
            (when (not (equal phn (first phns_needed2))) (return))
            (setf phns_needed2 (rest phns_needed2))
            (setf phns_found2 (append phns_found2 (list phn)))
            (setf recycled_phns (append recycled_phns (list phn))))

          (dolist (peak peaks_used) ; skip over peaks already listed
            (if (equal (first last_peaks2) peak)
                (setf last_peaks2 (rest last_peaks2))))

          (setf peaks_used2   ; move approp peaks from last list to this one
                 (recycle_peaks last_peaks2
                                recycled_phns
                                phns_needed2
                                peaks_used2))

                         ; don't use phn not found in peaks used
          (cond ((and (not (equal (first_phn (first (last peaks_used2)))
                                  (first (last phns_found2))))
                      (not (equal (second_phn (first (last peaks_used2)))
                                  (first (last phns_found2)))))
                 (setf phns_needed2 (append (last phns_found2)
                                            phns_needed2))
                 (setf phns_found2 (butlast phns_found2))))

          ; fail this word if it depends on same phn that failed last word
          (if (and (equal recycled_phns last_phns2)
                   (equal (first phns_needed2) *failed_phn*)
                   (not (null phns_needed2)))
              nil
              (build_word2 test_word   ; else look for rest of word
                           phns_found2
                           phns_needed2
                           peaks_used2
                           (if (> (length peaks_used2)
```

238

```
                              (length peaks_used))
                              (all_after rest_peaks (first (last peaks_used2)))
                              rest_peaks))))

                   ; no matches with last word so start from scratch
                        (t   (build_word2   test_word   phns_found   phns_needed   peaks_used
rest_peaks))))

        (t ; this peak is a new starting point in peaks so don't bother trying to recycle
            (build_word2 test_word phns_found phns_needed peaks_used rest_peaks)))))


;*************************************************************************
;
;  BUILD_WORD2 - Attempt (recursively) to find phonemes of given word in
;  peak list called by build_words_I, build_words_II, and build_words_III
;  through build_word.
;*************************************************************************
;

(defun build_word2 (test_word phns_found phns_needed peaks_used rest_peaks)
  (let* ((link (link_dip peaks_used phns_needed rest_peaks))
         (peak_used (first link))
         (rest_peaks2 (second link))
         (need nil))
    (setf *last_peaks* peaks_used) ; save results so far to avoid search for
    (setf *last_phns* phns_found)  ; same phns again on similar word
    (setf *failed_phn* (first (last phns_needed)))
    (cond ((null phns_needed)      ; success if no more phonemes to find
           (list test_word peaks_used (confidence_val peaks_used)))

          ((null rest_peaks) nil)  ; failure if no more peaks to try

          (peak_used          ; if there is a nearby peak with next phn

           (better_of (build_word2 test_word  ; try to chain from this link
                           (update_found phns_found peak_used)
                           (setf need (update_needed phns_needed peak_used))
                           (append peaks_used (list peak_used))
                           rest_peaks2)

                      (build_word2 test_word  ; if one link doen't work, try another
                           phns_found
                           phns_needed
                           peaks_used
                           rest_peaks2)))

           ; else if link to this phn not found but last peak was real, try splice
;          ((real_peak (car (last peaks_used)))
```

239

```lisp
;          (build_word2 test_word
;                       (append phns_found (list (first phns_needed)))
;                       (rest phns_needed)
;                       (append peaks_used
;                               (list (dummy_peak (car (last phns_found))
;                                                 (first phns_needed))))
;                       (skip_early rest_peaks
;                               (first (first (last peaks_used)))
;                               (list (first (last phns_found))
;                                       (first phns_needed)
;                                       (second phns_needed)))))

          (t nil))))      ; else word not found


;**********************************************************************
;**********************************************************************
;
;
;  SUPPORT  FUNCTIONS  FOR  TURNING  DIPHONE  PEAKS  INTO  WORD
CANDIDATES
;
;**********************************************************************
;**********************************************************************
;

;**********************************************************************
;
; LINK_DIP - Return a diphone that supplies needed phoneme (in 1st or 2nd
; pos)
;**********************************************************************
;

(defun link_dip (peaks_list phns_needed rest_peaks)
  (let* ((last_peak (first (last peaks_list)))
         (rest_peaks2 rest_peaks)
         (min_span_start (first last_peak))
         (max_span_start min_span_start)
         (last_phn (second_phn last_peak))
         (phn_needed (first phns_needed))
         (second_phn_needed (second phns_needed))
         (max_last_phn_dur (max_dur last_phn))
         (max_phn_needed_dur (max_dur phn_needed))
         (min_last_phn_dur (min_dur last_phn))
         (min_phn_needed_dur (min_dur phn_needed))
         (max_both_durs (+ max_last_phn_dur max_phn_needed_dur))
         (min_both_durs (+ min_last_phn_dur min_phn_needed_dur))
         (only_need_one (equal (length phns_needed) 1)))

    (cond
      ((null rest_peaks) nil)
```

240

```lisp
      (t (cond ((< min_span_start 0)      ; if last peak was dummy peak
                (setf min_span_start      ; estimate missing start pos
                      (- (first (first rest_peaks))
                         *win_rate*))
                (setf max_span_start (first (first rest_peaks)))))

        (dolist (peak rest_peaks)
          (let ((min_span (- (first peak) max_span_start))
                (max_span (- (first peak) min_span_start))
                (first_phn_peak (first_phn peak))
                (second_phn_peak (second_phn peak)))

            (setf rest_peaks2 (rest rest_peaks2))
            (cond ((> min_span (+ max_last_phn_dur max_phn_needed_dur))
                   (return nil))            ; if gone too far, return nil

                  ((and (equal first_phn_peak last_phn) ;peak's phn1=last phn2
                        (equal second_phn_peak phn_needed) ; peak's phn2=phn needed
                        (<= min_span max_last_phn_dur)    ; if timing looks right
                        (>= max_span min_last_phn_dur))
                   (return (list peak rest_peaks2)))

                  ((and (equal first_phn_peak phn_needed) ; peak's phn1=phn needed
                        (<= min_span max_both_durs)    ; if timing looks right
                        (>= max_span min_both_durs)
                        ; and this is last phn needed or both phns match
                        (or only_need_one
                            (equal second_phn_peak second_phn_needed)))
                   (return (list peak rest_peaks2)))
                  (t nil))))))))
```

;**********************************************************************
; NOT_ALREADY_FOUND - Return nil if current peak starts a word that was
; already found; else T
;**********************************************************************

```lisp
(defun not_already_found (word_entry word_list start_peak)
 (let ((word (first word_entry))
       (word_phn1 (second word_entry))
       (word_phn2 (third word_entry))
       (peak_phn1 (first_phn start_peak))
       (last_word_start (first (first (second (first (last word_list))))))
       (peak_start (first start_peak)))

   (cond ((null word_list) t)
```

241

```
        ((and (equal peak_phn1 word_phn2)  ; use duration of both phns
            (< last_word_start
                (- peak_start
                    (+ (max_dur word_phn1)
                        (max_dur word_phn2))))
            (> last_word_start
                0))
        t)   ; we have looked far enough back, so not found (T)
        ((and (equal peak_phn1 word_phn1)  ; only one phn duration counts
            (< last_word_start
                (- peak_start
                    (max_dur word_phn1)))
            (> last_word_start
                0))
        t)   ; we have looked far enough back, so not found (T)

        ((and (equal (first (first (last word_list))) word)) ; same word

        nil)  ; this start_peak is part of a word already found

        (t (not_already_found word_entry (butlast word_list) start_peak)))))

;*********************************************************************
;
; DUMMY_PEAK - Return a dummy peak containing desired phonemes (& low
; confidence).  Negative start sample number is used to flag peak as
; a dummy peak.
;*********************************************************************
;

(defun dummy_peak (phn1 phn2)
  (list -1 (list phn1 phn2) *dummy_confidence*))


;*********************************************************************
;
; REAL_PEAK - Return T if given peak is not a dummy peak (else nil)
;*********************************************************************
;

(defun real_peak (peak)  ; returns true if peak is real vice dummy
  (>= (first peak) 0))   ; dummy peaks have negative sample number


;*********************************************************************
;
; RECYCLE_PEAKS - If last word search located phonemes useful for this
; word, add appropriate peaks from last search to this word's peak list
;*********************************************************************
;

(defun recycle_peaks (last_peaks phns_recycled phns_needed peaks_used)
  (let ((last_peaks2 last_peaks)
        (recycled_phns phns_recycled)
```

```lisp
        (peaks_used2 peaks_used))

    (loop      ; add useful peaks previously found
      (cond ((null last_peaks2) (return))
            ((null recycled_phns) (return))
            ((and (equal (first_phn (first last_peaks2)) ;pk phn 1 =
                         (first recycled_phns))       ;next needed phn

              (or (and (null (second recycled_phns)) ; no more to recycle
                       (or (null (first phns_needed)) ; & no more needed
                           (equal (first phns_needed) ; or 1st phn needed =
                                  (second_phn (first last_peaks2))))) ; peak's 2nd
```
phn
```lisp
                  (equal (second_phn (first last_peaks2)) ;pk phn 2
                         (second recycled_phns))))

              (setf peaks_used2 (append peaks_used2
                                        (list (first last_peaks2))))

              (setf last_peaks2 (rest last_peaks2))
              (setf recycled_phns (rest (rest recycled_phns))))

            ((and (equal (first_phn (first last_peaks2))  ;pk phn1=
                         (second_phn (first (last peaks_used2)))) ;last phn in peak
```
list
```lisp
                  (equal (second_phn (first last_peaks2)) ;pk phn 2=
                         (first recycled_phns)))  ; next needed phn

              (setf peaks_used2 (append peaks_used2
                                        (list (first last_peaks2))))

              (setf last_peaks2 (rest last_peaks2))
              (setf recycled_phns (rest recycled_phns)))

            (t (return))))

    peaks_used2))

;*********************************************************************
;
; ALL_AFTER - Return portion of peak list after given peak
;*********************************************************************
;

(defun all_after (rest_peaks last_peak_used)
  (cond ((null rest_peaks) nil)
        ((null last_peak_used) rest_peaks)
        ((equal (first rest_peaks) last_peak_used)
```

```
                     (rest rest_peaks))
                 (t  (all_after (rest rest_peaks) last_peak_used))))


;********************************************************************
; SKIP_EARLY - Trim peaks from peak list that are too close to start pos
; to contain needed phonemes
;********************************************************************

(defun skip_early (rest_peaks start_pos phns)

  (let ((dur_sum 0)
        (loc_list rest_peaks)
        (limit nil))

    (dolist (phn phns)
      (setf dur_sum (+ dur_sum (min_dur phn))))

    (setf limit (+ start_pos dur_sum))

    (loop

      (cond ((null loc_list)
             (return nil))
            ((< (first (first loc_list)) limit)
             (setf loc_list (rest loc_list)))
            (t (return loc_list))))))


;********************************************************************
; UPDATE_FOUND - Add 1 or 2 phns to found list (depending on link type)
;********************************************************************

(defun update_found (phns_found peak_used)     ; if peak phn1 seen before
  (if (equal (first_phn peak_used) (first (last phns_found)))
    (append phns_found (rest (second peak_used)))  ; only add 1 phn
    (append phns_found (second peak_used))))        ; else add both phns


;********************************************************************
; UPDATE_NEEDED - Remove 1 or 2 phns from phn needed list (depending on
; link type)
;********************************************************************

(defun update_needed (phns_needed peak_used)    ; if peak phn1=1st needed
  (if (equal (first_phn peak_used) (first phns_needed))
    (rest (rest phns_needed))   ; remove both phns
    (rest phns_needed)))         ; else only remove 1 phn
```

244

```
;*************************************************************************
;
; MIN_DUR - Return minimum expected duration of given phoneme (in samples)
;*************************************************************************
;

(defun min_dur (phn)
  (let ((dur_list (index_pos phn *phn_durs*)))

    (if (null (first dur_list))
        0
        (first dur_list))))


;*************************************************************************
;
; MAX_DUR - Return maximum expected duration of given phoneme (in samples)
;*************************************************************************
;

(defun max_dur (phn)
  (let ((dur_list (index_pos phn *phn_durs*)))
    (if (null (second dur_list))
        0
        (second dur_list))))


;*************************************************************************
;
; BETTER_OF - Return word with higher confidence level based on peaks used
;*************************************************************************
;

(defun better_of (word1 word2)
  (if (> (confidence_in word1) (confidence_in word2))
      word1
      word2))


;*************************************************************************
;
; CONFIDENCE_VAL - Return confidence level based on peaks used to build
; word
;*************************************************************************
;

(defun confidence_val (peaks)
  (let ((conf_sum 0)
        (count 0))
    (dolist (peak peaks)
      (setf count (+ 1 count))
      (setf conf_sum (+ conf_sum (third peak))))
    (if (> count 0)
        (/ conf_sum count)
        0)))
```

```
;***********************************************************************
; CONFIDENCE_IN - Return confidence level stored in word's list
;***********************************************************************

(defun confidence_in (word)
  (if (null word)
      0
      (third word)))


;***********************************************************************
; FIRST_PHN - Return first phoneme in given peak's phn list
;***********************************************************************

(defun first_phn (peak)
  (if (null peak)
      nil
      (first (second peak))))


;***********************************************************************
; SECOND_PHN - Return first phoneme in given peak's phn list
;***********************************************************************

(defun second_phn (peak)
  (if (null peak)
      nil
      (second (second peak))))


;***********************************************************************
;***********************************************************************
;
;
; SORTING, LISTING, & I/O RELATED CODE
;
;***********************************************************************
;***********************************************************************


;***********************************************************************
; SORT_WORDS - Sort the word candidate list by sample number and then
; by confidence level
;***********************************************************************

(defun sort_words (word_candidates)
  (let ((result nil)
        (wl nil))
    (dolist (word word_candidates)            ; start/stop sample numbers
      (setf wl (list (list (start_samp word)
                           (end_samp word))
```

246

```lisp
                (first word)                ; word
                (confidence_in word)))      ; confidence
        (setf result (append result (list wl))))
      (setf result (sort result #'more_conf))
      (setf result (stable-sort result #'before))))
```

```
;******************************************************************
;  BEFORE - Returns true if wrd1 has lower sample number than wrd2 (for
;  sort)
;******************************************************************
```

```lisp
(defun before (wrd1 wrd2)
  (if (< (first (first wrd1)) (first (first wrd2)))
      t
      nil))
```

```
;******************************************************************
;  MORE_CONF - Boolean function comparing word confidences (for sort)
;******************************************************************
```

```lisp
(defun more_conf (w1 w2)
  (if (> (third w1) (third w2))
      t
      nil))
```

```
;******************************************************************
;  WRITE_WORDS - Write word candidates to designated file handle
;******************************************************************
```

```lisp
(defun write_words (word_list f_handle)
  (dolist (word word_list)
    (format f_handle "~% ~S" word)
    (print word))
  t)
```

```
;******************************************************************
;  LIST_WORDS - Sort and display list of word candidates
;******************************************************************
```

```lisp
(defun list_words ()
  (dolist (word (sort_words *word_candidates*))
    (format t "~% ~S" word))
  t)
```

```
;******************************************************************
;  SAVE_WORDS - Save word list to designated filename
```

```
;*********************************************************************
;

(defun save_words (fname)
  (let ((out_file (open fname :direction :output
                        :if-does-not-exist :create
                        :if-exists :rename)))
    (dolist (word (sort_words *word_candidates*))
      (format out_file "~% ~S" word)
      (print word))
    (close out_file))
  t)


;*********************************************************************
;
; LIST_TO_FILE - Save contents of a list to a file (one item per line)
;*********************************************************************
;

(defun list_to_file (object_name fname)
  (let ((f_handle (open fname :direction :output
                        :if-does-not-exist :create
                        :if-exists :append)))
    (dolist (item object_name)
      (format f_handle "~S~%" item)
      )
    (close f_handle)
    )
  )


;*********************************************************************
;
; START_SAMP - Return starting sample number for a word entry
;*********************************************************************
;

(defun start_samp (word)
  (if (null word) 0
      (if (>= (first (first (second word))) 0)
          (first (first (second word)))
          (first (second (second word)))))))


;*********************************************************************
;
; END_SAMP - Return ending sample number for a word entry
;*********************************************************************
;

(defun end_samp (word)
  (max (first (first (last (second word))))
       (if (butlast (second word))
           (first (first (last (butlast (second word)))))
           0)))
```

248

```
;**********************************************************************
; WORD_ENTRY - Return most confident entry for given word in
; word candidate list
;**********************************************************************
;

(defun word_entry (correct_word word_list)
  (cond ((null word_list) nil)
        ((string-equal (first (first word_list)) correct_word)
         (better_of (first word_list)
                    (word_entry correct_word (rest word_list))))
        (t (word_entry correct_word (rest word_list)))))


;**********************************************************************
; STATS - Return list of statistics reflecting how well word was
; recognized
;**********************************************************************
;

(defun stats (correct_word word_list)
  (let* ((cword (word_entry correct_word word_list))
         (cword_conf (confidence_in cword))
         (cword_lrank 1)
         (cword_rank 1)
         (best_w nil)
         (best_conf 0)
         (lbest_w nil)
         (lbest_conf 0))
    (print correct_word)
    (dolist (w word_list)
      (cond ((> (confidence_in w) best_conf)
             (setf best_conf (confidence_in w))
             (setf best_w (first w))))
      (cond ((> (confidence_in w) cword_conf)
             (setf cword_rank (1+ cword_rank))))
      (cond ((equal (start_samp w)
                    (start_samp cword))
             (if (> (confidence_in w) cword_conf)
                 (setf cword_lrank (1+ cword_lrank)))
             (cond ((> (confidence_in w) lbest_conf)
                    (setf lbest_conf (confidence_in w))
                    (setf lbest_w (first w)))))))
    (list (start_samp cword)
          correct_word
          cword_conf
          cword_lrank
          cword_rank
```

```
        lbest_w
        lbest_conf
        best_w
        best_conf)))


;**********************************************************************

;**********************************************************************
;**********************************************************************
;
;
; TEST CODE
;
;
;**********************************************************************
;**********************************************************************
;


;**********************************************************************
;
;  RECOGNIZE - Open lexicon files & look for words in sample peak list
;**********************************************************************
;

(defun test ()
  (let ((index_file (open "phnindex.txt"))
        (index_file2 (open "phnindx2.txt"))
        (phn_durfile (open "phndurs.txt"))
        (fname "dips"))
    (setf *dict_file* (open "lexicon.txt"))
    (setf *dict_file2* (open "lexicon2.txt"))
    (setf *phoneme_index* (read index_file nil nil nil))
    (setf *phoneme_index2* (read index_file2 nil nil nil))
    (setf *phn_durs* (read phn_durfile nil nil nil))
    (dip2peak fname)
    (setf *word_candidates* nil)
    (peaks_to_words *diphone_peak_list*)
    (close index_file)
    (close index_file2)
    (close phn_durfile)
    (close *dict_file*)
    (close *dict_file2*)))


;**********************************************************************
;
; TEST_WORDS - Open designated diphone recognizer (Matlab script file
; recogniz.m) files & look for words.  Save results to second filename.
;**********************************************************************
;

(defun recognize (nnet_outputsname out_fname)
  (let* ((index_file (open "phnindex.txt"))
         (index_file2 (open "phnindx2.txt"))
```

```lisp
      (phn_durfile (open "phndurs.txt"))
      (debug_fname "recogniz.dbg")
      (report_fname "recogrep.txt")
      (nnet_outputs (open nnet_outputsname))
      (out_file (open out_fname :direction :output
                             :if-does-not-exist :create
                             :if-exists :rename))
      (report_file (open report_fname :direction :output
                                 :if-does-not-exist :create
                                 :if-exists :append))
      (correct_word nil)
      (temp_out nil)
      (obj_read nil))
(setf *dict_file* (open "lexicon.txt"))
(setf *dict_file2* (open "lexicon2.txt"))
(setf *phoneme_index* (read index_file nil nil nil))
(setf *phoneme_index2* (read index_file2 nil nil nil))
(setf *phn_durs* (read phn_durfile nil nil nil))
(loop
  (setf obj_read (read nnet_outputs nil nil nil))
  (when (null obj_read) (return))
  (print obj_read out_file)
  (setf correct_word (second obj_read))
  (format t "~%~S: Reading NNET outputs..." correct_word)
  (setf temp_out (open "temp.dip" :direction :output
                             :if-does-not-exist :create
                             :if-exists :rename-and-delete))
  (loop
      (setf obj_read (read nnet_outputs nil nil nil))
      (when (equal (first obj_read) 'word_end) (return))
      (print obj_read temp_out))
  (close temp_out)
  (format t "~%~S: Locating output peaks..." correct_word)
  (dip2peak "temp.dip")
  (setf *word_candidates* nil)
  (format t "~%~S: Looking for words..." correct_word)
  (peaks_to_words *diphone_peak_list*)
  (format t "~%~S: Saving results to file..." correct_word)
  (write_words (sort_words *word_candidates*) out_file)
  (list_to_file (cons (list correct_word) *word_candidates*) debug_fname)
  (print (stats correct_word *word_candidates*) report_file))
(close nnet_outputs)
(close out_file)
(close index_file)
(close index_file2)
(close phn_durfile)
```

```
(close report_file)
(close *dict_file*)
(close *dict_file2*)))
```

## C.  PEAKS.CL

```
;**********************************************************************
;*  RECOG.CL - This program converts the output file from a diphone
;*  recognizer into a string of phonemes and then words.  The following "type
;*  definitions" will be used in program comments (each is implemented as
;*  a LISP list):
;*
;*   diphone ::= (phoneme phoneme);
;*   diphone_record ::= (sample_num diphone confidence_value)
;*   three_best_diphones ::= (diphone_record diphone_record diphone_record)
;*
;*  The diphone recognizer converts a PCM voice file into an ASCII file
;*  of three_best_diphones (one line each).  Each three best diphone
;*  represents the three output classifiers with the highest activation
;*  level at the time represented by the sample_num.  In principal, each
;*  diphone classifier's confidence_value should peak when the sample
;*  window is perfectly aligned with the diphone.  So this program tracks
;*  each diphone from the time it first appears in a three_best_diphones
;*  line until it is replaced by a new "winner."  Once a diphone is
;*  displaced, its peak diphone_record is appended to the *diphone_peak_list*.
;*  The diphone_peak_list is then (time) sorted for conversion to a list of
;*  phonemes by the peak2phn function.
;**********************************************************************
                              ; globals
(defvar *current_best_diphones* nil)
(defvar *new_best_diphones* '(nil nil nil))

;**********************************************************************
;  FOLLOWING  FUNCTIONS  TURN  DIPHONE  RECOGNIZER  OUTPUT  INTO
SEQUENCE ; OF  DIPHONE PEAKS;
;**********************************************************************
  ; DIP2PEAK - main function for opening ASCII file of three_best_diphones
  ; and converting to a string of diphone peaks
(defun dip2peak (fname)
  (let ((dip_data_file (open fname)))
    (setf *current_best_diphones* nil)
    (setf *new_best_diphones* '(nil nil nil))
```

252

```
(setf *diphone_peak_list* nil)
(loop
  (let ((three_best_diphones (read dip_data_file nil nil nil)))
    (if (null three_best_diphones) (return)
                         (record_diphones three_best_diphones))))
(refresh_best_diphones)
(stable-sort *diphone_peak_list* #'is_earlier)
(close dip_data_file)))

; IS_EARLIER - predicate for use by stable-sort
(defun is_earlier (first_dip second_dip)
  (if (< (first first_dip) (first second_dip))
    first_dip
    nil))

; RECORD_DIPHONES - Save latest three_best_diphones in appropriate slot
; in *new_best_diphones*, shift results to *current_best_diphones*,
; and reinitialize *new_best_diphones* to nils.
(defun record_diphones (three_dip_list)
  (when (not (null three_dip_list))
    (record_dip (first three_dip_list))
    (record_dip (second three_dip_list))
    (record_dip (third three_dip_list))
    (refresh_best_diphones)
    (setq *new_best_diphones* (list nil nil nil))
    t ))

; RECORD_DIP - Save a diphone_record in the appropriate slot in
; in *new_best_diphones*.  If the diphone_record contains same
; diphone as an existing *current_best_diphones* entry, save in
; corresponding slot in *new_best_diphones*.  Else tack it on
; end of *new_best_diphones*.  REFRESH_BEST_DIPHONES will then
; know (by a nil in one of first 3 slots) that one of the diphones
; it was tracking has been replaced.
(defun record_dip (new_diphone_rec)
  (cond
    ((equalp (second new_diphone_rec) (second (first *current_best_diphones*)))
       (update_best_diphone 1 new_diphone_rec))
    ((equalp (second new_diphone_rec) (second (second *current_best_diphones*)))
       (update_best_diphone 2 new_diphone_rec))
    ((equalp (second new_diphone_rec) (second (third *current_best_diphones*)))
       (update_best_diphone 3 new_diphone_rec))
    (t (new_best_diphone new_diphone_rec)) ))

; BEST_OF - Used by UPDATE_BEST_DIPHONE to determine which of 2 diphones
; has a higher confidence_value.
```

253

```lisp
(defun best_of (first_dip second_dip)
  (cond ((null first_dip) second_dip)
        ((null second_dip) first_dip)
        ((> (car (last first_dip)) (car (last second_dip))) first_dip)
        (second_dip) ))


; UPDATE_BEST_DIPHONE - Used by RECORD_DIP to put best of old & new dips
; in the appropriate slot of *new_best_diphones* list.
(defun update_best_diphone (best_dip_number new_best_dip_rec)
  (case best_dip_number
    (1 (setf (first *new_best_diphones*)
             (best_of new_best_dip_rec (first *current_best_diphones*)) ))
    (2 (setf (second *new_best_diphones*)
             (best_of new_best_dip_rec (second *current_best_diphones*)) ))
    (3 (setf (third *new_best_diphones*)
             (best_of new_best_dip_rec (third *current_best_diphones*)) ))
    (otherwise nil) ))



; NEW_BEST_DIPHONE - Used by RECORD_DIP to append a new diphone_record
; on end of *new_best_diphones* for future tracking.  This will leave
; a nil in one of first 3 slots of *new_best_diphones* which will, in
; turn, trigger recording of the peak value for the diphone that dropped
; off the "winners" list.
(defun new_best_diphone (new_best_dip_rec)
  (setf *new_best_diphones*
        (append *new_best_diphones* (list new_best_dip_rec)) ))



; REFRESH_BEST_DIPHONES - Used by RECORD_DIPHONES to move the lastest
; "winning" diphones from *new_best_diphones* to *current_best_diphones*.
; But first, any nils in first 3 slots of *new_best_diphones* (meaning
; a diphone has dropped off the "winners list") trigger recording of
; diphone_record in corresponding slot of *current_best_diphones* to
; *diphone_peak_list* (at any time, *current_best_diphones* has most
; confident [peak] value for 3 diphones being tracked).
(defun refresh_best_diphones ()
  (if (null (first *new_best_diphones*))
      (record_diphone_peak (first *current_best_diphones*)))
  (if (null (second *new_best_diphones*))
      (record_diphone_peak (second *current_best_diphones*)))
  (if (null (third *new_best_diphones*))
      (record_diphone_peak (third *current_best_diphones*)))
  (setf *new_best_diphones* (remove_nils *new_best_diphones*))
  (setf *current_best_diphones* *new_best_diphones*)
  t)
```

```
; RECORD_DIPHONE_PEAK - Append a peak diphone_record to *diphone_peak_list.
(defun record_diphone_peak (latest_diphone_peak)
  (when (not (null latest_diphone_peak))
      (setf *diphone_peak_list*
          (append *diphone_peak_list* (list latest_diphone_peak)))
     t ))


; REMOVE_NILS - Used by REFRESH_BEST DIPHONES
(defun remove_nils (nilly_list)
  (if (not (null (first nilly_list)))
     (append (list (first nilly_list)) (remove_nils (rest nilly_list)))
     (if (not (null (rest nilly_list)))
        (remove_nils (rest nilly_list))     ) ))

(defun testfile (fname)
  (dip2peak fname)
  *diphone_peak_list*)

(defun test_peaks ()
  (format t "~% DIPHONE PEAKS:~%")
  (print (testfile "dips"))
  (read-line)
  (format t "~%~%~% FINAL DIPHONE SEQUENCE:~%")
  (print *diphone_peak_list*))
```

## D.    DICT.CL

```
;************************************************************************
;************************************************************************
;
;
; PHONETIC DICTIONARY-RELATED FUNCTIONS
;
;************************************************************************
;************************************************************************
;

;************************************************************************
;************************************************************************
;

(defun write_file (object_name fname)
  (let ((f_handle (open fname :direction :output)))
    (print object_name f_handle)
    (close f_handle) ))
```

```
;*************************************************************************
; CREATE_PHN_DICT - Reads the standard TIMIT phonetic dictionary and
; converts each entry into a LISP list.  Phonemes are placed
; first to allow sorting of entire dictionary by phoneme vice by word.
;*************************************************************************

(defun create_phn_dict (in_fname out_fname)
  (let ((lexicon_file (open in_fname))
        (output_file  (open out_fname :direction :output))
        (word_read nil)  )
    (format output_file "(")
    (loop
      (setf word_read (read_word lexicon_file))
      (if (null word_read)
        (return)
        (if (> (length word_read) 2)
          (print (append (rest word_read)
              (list (first word_read)))
          output_file ))))
    (format output_file ")")
    (close lexicon_file)
    (close output_file)))


;*************************************************************************
;*************************************************************************

; PUT_WORDS_FIRST - Returns phonetic dictionary to word first format
; after sorting by phonemes.
(defun put_words_first (in_fname out_fname)
  (let ((phn_dict_file (open in_fname))
        (output_file  (open out_fname :direction :output))
        (obj_read nil))
    (setf obj_read (read phn_dict_file nil nil nil))
    (if (not (null obj_read)) (file-position phn_dict_file :start))
    (loop
      (setf obj_read (read phn_dict_file nil nil nil))
      (cond ((null obj_read) (return))
            (t (print (append (last obj_read)
                        (butlast obj_read)) output_file))))
    (close phn_dict_file)·
    (close output_file) ))


;*************************************************************************
; CREATE_PHN_DICT2 - Reads the standard TIMIT phonetic dictionary and
; converts each entry into a LISP list.  Phonemes are placed
; first to allow sorting of entire dictionary by phoneme vice by word.
```

256

```
;*************************************************************************

(defun create_phn_dict2 (in_fname out_fname)
  (let ((lexicon_file (open in_fname))
        (output_file  (open out_fname :direction :output))
        (word_read nil) )
    (format output_file "(")
    (loop
      (setf word_read (read_word lexicon_file))
      (if (null word_read)
        (return)
        (if (> (length word_read) 2)
            (print (append (rest (rest word_read))
                      (list (first word_read) (second word_read)) )
                output_file))))
    (format output_file ")")
    (close lexicon_file)
    (close output_file)))

;*************************************************************************
;*************************************************************************

; PUT_WORDS_FIRST2 - Returns phonetic dictionary to word first format
; after sorting by phonemes.
(defun put_words_first2 (in_fname out_fname)
  (let ((phn_dict_file (open in_fname))
        (output_file  (open out_fname :direction :output))
        (obj_read nil))
    (setf obj_read (read phn_dict_file nil nil nil))
    (if (not (null obj_read)) (file-position phn_dict_file :start))
    (loop
      (setf obj_read (read phn_dict_file nil nil nil))
      (cond ((null obj_read) (return))
            ((> (length obj_read) 2)
             (print (append (last (butlast obj_read))
                      (last obj_read)
                      (butlast (butlast obj_read)))
                 output_file))))
    (close phn_dict_file)
    (close output_file)))

;*************************************************************************
;*************************************************************************

; BUILD_INDEX - Creates a list of phoneme/file-position pairs.
; Each pair records the file position of the first word entry that
```

257

```
; starts with the given phoneme.
(defun build_index (fname out_fname)
  (let ((lexicon_file (open fname))
        (last_phn nil)
        (word_read nil)
        (file_pos 0)
        (phn_index nil) )
    (setf word_read (read lexicon_file nil nil nil))
    (if (not (null word_read)) (file-position lexicon_file :start))
    (loop
      (setf file_pos (file-position lexicon_file))
      (setf word_read (read lexicon_file nil nil nil))
      (cond ((null word_read) (return))
            ((not (equal last_phn (second word_read)))
              (setf phn_index
                    (append phn_index
                            (list (list (second word_read) file_pos))))
              (setf last_phn (second word_read)))
            (t nil)))
    (close lexicon_file)
    (write_file phn_index out_fname) ))

;***********************************************************************
; BUILD_INDEX2 - Creates a list of phoneme/file-position pairs.
; Each pair records the file position of the first word entry that
; has phoneme 2 = the given phoneme.
;***********************************************************************

(defun build_index2 (fname out_fname)
  (let ((lexicon_file (open fname))
        (last_phn nil)
        (word_read nil)
        (file_pos 0)
        (phn_index nil))
    (setf word_read (read lexicon_file nil nil nil))
    (if (not (null word_read)) (file-position lexicon_file :start))
    (loop
      (setf file_pos (file-position lexicon_file))
      (setf word_read (read lexicon_file nil nil nil))
      (cond ((null word_read) (return))
            ((not (equal last_phn (third word_read)))
              (setf phn_index
                    (append phn_index
                            (list (list (third word_read) file_pos))))
              (setf last_phn (third word_read)))
            (t nil)))
```

258

```
    (close lexicon_file)
    (write_file phn_index out_fname)))


;**********************************************************************
;**********************************************************************

(defun sort_dict (in_fname out_fname)
  (let ((tmp_dict (open in_fname))
        (out_dict (open out_fname :direction :output))
        (dict_list nil))
    (setf dict_list (read tmp_dict nil nil nil))
    (sort dict_list #'word<)
    (dolist (wrd dict_list)
      (print wrd out_dict))
    (close tmp_dict)
    (close out_dict)))


;**********************************************************************
;**********************************************************************
(defun word< (wrd1 wrd2)
  (if (or (string< (symbol-name (first wrd1))
                   (symbol-name (first wrd2)))
          (and (equal (first wrd1)
                      (first wrd2))
               (string< (symbol-name (second wrd1))
                        (symbol-name (second wrd2)) ) ) )
      t
      nil ))


;**********************************************************************
;**********************************************************************

; READ_WORD_LINE - Returns the next non-comment line from TIMIT dictionary.
(defun read_word_line (file_handle)
  (let ((line_read nil))
    (loop
      (setf line_read (read-line file_handle nil nil nil))
      (cond  ((null line_read) (return))
             ((and (not (equal (char line_read 0) #\;))
                   (not (null (read-from-string line_read))))
              (return line_read)) ) )))


;**********************************************************************
;**********************************************************************

; TEXT_TO_LIST - Converts a TIMIT dictionary entry from text to a LISP list.
```

```
(defun text_to_list (text_line)
  (if (not (null text_line))
    (let ((text_list (format nil "~S" (list text_line))))
      (setf (char text_list 1) #\Space)
      (setf (char text_list (- (length text_list) 2)) #\Space)
      (read-from-string text_list) ) ))
```

```
;**********************************************************************
;**********************************************************************
```

```
; CLEAN_TEXT - Removes extraneous characters from TIMIT dictionary entries.
(defun clean_text (phn_text)
  (if (not (null phn_text))
    (let ((cleaned_text phn_text)
          (extraneous_chars '(#\V #\1 #\2 #\3)))
      (dotimes (i (length phn_text) cleaned_text)
        (if (member (char phn_text i) extraneous_chars)
          (setf (char cleaned_text i) #\Space) )
        (if (equal (char phn_text i) #\')
          (setf (char cleaned_text i) #\~) ) )
      cleaned_text) ))
```

```
;**********************************************************************
;**********************************************************************
```

```
; READ_WORD - Returns next TIMIT dictionary entry as a LISP list.
(defun read_word (file_handle)
  (text_to_list (clean_text (read_word_line file_handle))))
```

```
;**********************************************************************
;**********************************************************************
```

```
; BIN_SEARCH - Binary search used by INDEX_POS to determine first
; dictionary entry with given starting phoneme
(defun bin_search (phoneme start_n end_n index)
  (let ((mid_n (round (/ (+ start_n end_n) 2))))
    (cond ((>= start_n end_n)
            (if (equal (first (nth start_n index)) phoneme)
              (second (nth start_n index))
              nil))
          ((string< (string phoneme)
                    (string (first (nth mid_n index))))
            (bin_search phoneme start_n (1- mid_n) index))
          ((string> (string phoneme)
                    (string (first (nth mid_n index))))
            (bin_search phoneme (1+ mid_n) end_n index))
```

```
      ( t (second (nth mid_n index))))))

;**********************************************************************
;**********************************************************************

 ; INDEX_POS - Uses BIN_SEARCH to determine first
 ; dictionary entry with given starting phoneme
(defun index_pos (phoneme index)
  (bin_search phoneme 0 (- (length index) 1) index))

;**********************************************************************
;**********************************************************************

(defun make_dicts ()
  (format t "~%Preparing for sort #1...")
  (create_phn_dict "timitdic.txt" "lexicon.txt")
  (format t "~%Sorting lexicon #1...")
  (sort_dict "lexicon.txt" "tempdic.txt")
  (format t "~%Reordering #1...")
  (put_words_first "tempdic.txt" "lexicon.txt")
  (format t "~%Building index #1...")
  (build_index "lexicon.txt" "phnindex.txt")
  (format t "~%Preparing for sort #2...")
  (create_phn_dict2 "timitdic.txt" "lexicon2.txt")
  (format t "~%Sorting #2...")
  (sort_dict "lexicon2.txt" "tempdic.txt")
  (format t "~%Reordering #2...")
  (put_words_first2 "tempdic.txt" "lexicon2.txt")
  (format t "~%Indexing #2...")
  (build_index2 "lexicon2.txt" "phnindx2.txt")
  (format t "~%Done building lexicons")
  t)
```

## E.    LEXICON.TXT

(OBJECT~N AA B JH EH K T)
(OBJECTS AA B JH EH K T S)
(OBSESSION AA B S EH SH IX N)
(OBSESSIONS AA B S EH SH IX N Z)
(OBSOLESCENT AA B S AX L EH S EN T)
(OBSTACLES AA B S T IH K EL Z)
(OBVIOUS AA B V IY AX S)
(OBVIOUSLY AA B V IY AX S L IY)
(ODD AA D)
(ODDS AA D Z)
(ODDS-ON AA D Z AO N)
(AHAH AA HH AA)
...      // BULK OF LEXICON OMITTED TO SAVE SPACE  //
(USE~N Y UW S)
(USE~V Y UW Z)
(USING Y UW Z IX NG)
(USUAL Y UW ZH UW EL)
(USUALLY Y UW ZH UW AX L IY)
(UTILIZING Y UW T AX L AY Z IX NG)
(UTOPIANISM Y UW T OW P IY EN IH Z EM)
(UTOPIANS Y UW T OW P IY EN Z)
(YOU Y UW)


## F.    LEXICON2.TXT

(BOB B AA B)
(BOBBY B AA B IY)
(BOBCAT B AA B K AE T)
(COBBLER~S K AA B L AXR Z)
(JOB JH AA B)
(ROB R AA B)
(ROBIN R AA B IX N)
(WOBBLE W AA B EL)
...      // BULK OF LEXICON OMITTED TO SAVE SPACE  //
(EASY IY Z IY)
(EASYGOING IY Z IY G OW IX NG)
(ASTHMA AE Z M AX)

## G. PHNINDEX.TXT

```
((AA 0)       (AE 1406)     (AH 5923)     (AO 8582)     (AW 10417)   (AX 11006)
 (AY 17254)   (B 18190)     (CH 26124)    (D 27445)     (DH 36700)   (EH 37226)
 (ER 40653)   (EY 41015)    (F 41482)     (G 49147)     (HH 53189)   (IH 59913)
 (IY 69715)   (JH 70173)    (K 72175)     (L 86484)     (M 91274)    (N 99381)
 (OW 102942)  (OY 104478)   (P 104628)    (R 116939)    (S 126411)   (SH 145371)
 (T 147921)   (TH 154674)   (UW 156159)   (V 156176)    (W 158877)   (Y 162935))
```

## H. PHNINDX2.TXT

```
((AA 0)       (AE 8496)     (AH 19748)    (AO 26768)    (AW 32328)    (AX 33915)
 (AXR 41419)  (AY 43122)    (B 48901)     (CH 50132)    (D 50245)     (DH 52101)
 (EH 52196)   (ER 63293)    (EY 67429)    (F 72390)     (G 73210)     (HH 74464)
 (IH 74503)   (IX 90607)    (IY 91610)    (JH 97191)    (K 97602)     (L 102931)
 (M 110634)   (N 113252)    (NG 123018)   (OW 123226)   (OY 127976)   (P 128847)
 (R 132537)   (S 146744)    (SH 148084)   (T 148426)    (TH 153434)   (UH 153811)
 (UW 155688)  (V 158995)    (W 160867)    (Y 163488)    (Z 164536))
```

## I. PHNDURS.TXT

```
(
(AA ( 381  7735  1984))
(AE ( 660  6800  2144))
(AH ( 300  5442  1414))
(AO ( 421  5760  1957))
(AW ( 1003  6640  2563))
(AX ( 74  2778  766))
(AXR ( 305  3752  1236))
(AY ( 616  6790  2485))
(B ( 47  1229  279))
(CH ( 320  3200  1382))
(D ( 81  1840  385))
(DH ( 110  1839  563))
(EH ( 343  4498  1448))
(EL ( 440  3810  1440))
(EM ( 656  3300  1281))
(EN ( 204  2943  1236))
(ENG ( 694  1882  1274))
(ER ( 560  5862  1937))
(EY ( 668  6250  2017))
(F ( 213  4880  1640))
(G ( 124  1661  442))
```

(HH ( 220  3440  1079))
(HX ( 480  32767  2714))
(IH ( 333  6386  1244))
(IX ( 186  3189  819))
(IY ( 372  5540  1529))
(JH ( 238  3073  998))
(K ( 98  2716  829))
(L ( 203  3677  970))
(M ( 130  3676  1032))
(N ( 152  3397  880))
(NG ( 256  3844  1003))
(OW ( 559  5520  2039))
(OY ( 1002  5096  2707))
(P ( 100  1930  707))
(R ( 36  2783  893))
(S ( 374  5709  1798))
(SH ( 385  4213  1895))
(T ( 74  2315  773))
(TH ( 201  3602  1404))
(UH ( 292  3509  1217))
(UW ( 390  5034  1674))
(V ( 192  2464  947))
(W ( 152  3280  979))
(Y ( 183  2632  868))
(Z ( 410  4310  1365))
(ZH ( 355  4218  1362))
)

# APPENDIX D.  LIST OF 157 DIPHONES IN MEDIUM AND LARGE DATA SETS

| | | | | |
|---|---|---|---|---|
| AA K | EH K | IX N | L IH | S IX |
| AA L | EH L | IX NG | L IX | S IY |
| AA R | EH N | IX P | L IY | S K |
| AE K | EH R | IX S | M AH | S P |
| AE M | EH S | IX T | M AY | S T |
| AE N | EH V | IX V | M EY | SH EN |
| AH M | EY N | IX Z | M IX | SH IX |
| AH N | EY SH | IY HX | M P | T AE |
| AH S | EY T | IY IX | N D | T AXR |
| AO L | F AO | IY K | N DH | T EH |
| AO R | F AXR | IY N | N HX | T HX |
| AW N | F IH | IY NG | N IH | T IH |
| AX B | F R | IY P | N IX | T IX |
| AX L | G R | IY S | N IY | T IY |
| AX M | HX D | IY Z | N S | T R |
| AX N | HX DH | JH IX | N T | T S |
| AX P | HX HH | K AA | N Z | V AXR |
| AX S | HX K | K AE | P AXR | W AX |
| AX V | HX W | K AO | P L | W IH |
| AXR HX | IH G | K EL | P R | W IX |
| AXR Z | IH K | K IX | R AA | W IY |
| B AY | IH L | K L | R AE | Z AX |
| B EL | IH M | K R | R AH | Z HX |
| B IH | IH N | K S | R AY | Z IX |
| B IY | IH NG | K T | R EH | Z K |
| B R | IH S | K W | R EY | |
| D HX | IH SH | L AA | R IH | |
| D IH | IH Z | L AE | R IX | |
| D IX | IX D | L AX | R IY | |
| D R | IX G | L AY | R OW | |
| DH AX | IX JH | L D | S AH | |
| DH IX | IX K | L EH | S AX | |
| DH IY | IX M | L EY | S HX | |

265

# APPENDIX E. TIMIT VOCABULARY LIST FOR WORD TESTS

## A. DESCRIPTION

The following table lists the 340 words in the TIMIT lexicon that can be constructed using the 157 diphones of the Medium and Large Data Sets. The lexical analyzer scans the diphone stream from the neural network and identifies the phonemes for a word if they are present. Because each phoneme in a word forms part of two diphones, it is possible to detect all of a word's phonemes even if some diphones are missing from the diphone stream (up to every other diphone can be missing). Thus, it is actually possible to detect far more than the 340 words listed here. However, the additional words would suffer a lower probability of detection. Therefore, this vocabulary list is limited to words with all diphones among the neural network's output classes, except as described in the next paragraph.

The number following each word represents the number of that word's diphones that are not among the 157 diphones of the Medium and Large Data Sets. Currently, VOCAB.ADA will only allow a word's first diphone to be missing from the diphone list. In every case, a word's first diphone is considered to be a period of silence (denoted by the dummy phoneme **HX**) combined with the word's first phoneme. The lexical analyzer must be able to detect a word without this diphone because in continuous speech it cannot count on having a period of silence before every word. Thus, VOCAB.ADA will consider a word to be in-vocabulary, even if this first diphone is not in the current list of recognizable diphones. The diphone listed after each number is the name of any diphones that are not among the 157 diphones of the Medium and Large Data Sets. Again, this list will currently only contain, at most, the word's first diphone.

## B. VOCABULARY

| | | |
|---|---|---|
| a, 1,H#/AX | fishing, 1,H#/F | remains, 1,H#/R |
| ability, 1,H#/AX | fist, 1,H#/F | rent, 1,H#/R |
| abyss, 1,H#/AX | fists, 1,H#/F | rented, 1,H#/R |
| accident, 1,H#/AE | fix, 1,H#/F | rest, 1,H#/R |
| act, 1,H#/AE | fixed, 1,H#/F | restricted, 1,H#/R |
| actor, 1,H#/AE | for, 1,H#/F | rickety, 1,H#/R |
| actors, 1,H#/AE | forest, 1,H#/F | rim, 1,H#/R |

| | | |
|---|---|---|
| acts, 1,H#/AE | four, 1,H#/F | risks, 1,H#/R |
| ain't, 1,H#/EY | frantically, 1,H#/F | rock, 1,H#/R |
| all, 1,H#/AO | frequent, 1,H#/F | rockets, 1,H#/R |
| am, 1,H#/AE | friend, 1,H#/F | run, 1,H#/R |
| an, 1,H#/AE | from, 1,H#/F | running, 1,H#/R |
| and, 1,H#/AE | front, 1,H#/F | runs, 1,H#/R |
| ant, 1,H#/AE | frustration, 1,H#/F | rust, 1,H#/R |
| ants, 1,H#/AE | grains, 1,H#/G | scar, 1,H#/S |
| any, 1,H#/EH | grants, 1,H#/G | scenes, 1,H#/S |
| apply, 1,H#/AX | greasing, 1,H#/G | score, 1,H#/S |
| approximated, 1,H#/AX | greasy, 1,H#/G | screen, 1,H#/S |
| approximation, 1,H#/AX | great, 1,H#/G | sea, 1,H#/S |
| are, 1,H#/AA | greater, 1,H#/G | secret, 1,H#/S |
| ate, 1,H#/EY | green, 1,H#/G | see, 1,H#/S |
| aunt, 1,H#/AE | greenness, 1,H#/G | seeing, 1,H#/S |
| aw, 1,H#/AO | grill, 1,H#/G | seeking, 1,H#/S |
| awe, 1,H#/AO | grin, 1,H#/G | seen, 1,H#/S |
| axis, 1,H#/AE | grinned, 1,H#/G | selected, 1,H#/S |
| be, 1,H#/B | grow, 1,H#/G | selecting, 1,H#/S |
| beans, 1,H#/B | grunt, 1,H#/G | solicits, 1,H#/S |
| been, 1,H#/B | i, 1,H#/AY | some, 1,H#/S |
| beep, 1,H#/B | ill, 1,H#/IH | son, 1,H#/S |
| being, 1,H#/B | in, 1,H#/IH | splinter, 1,H#/S |
| big, 1,H#/B | innocence, 1,H#/IH | sprained, 1,H#/S |
| bill, 1,H#/B | insulate, 1,H#/IH | spray, 1,H#/S |
| biscuit, 1,H#/B | insulator, 1,H#/IH | spring, 1,H#/S |
| brain, 1,H#/B | integrated, 1,H#/IH | squeaked, 1,H#/S |
| brand, 1,H#/B | integration, 1,H#/IH | stacked, 1,H#/S |
| breast, 1,H#/B | intelligence, 1,H#/IH | stand, 1,H#/S |
| breeze, 1,H#/B | intelligent, 1,H#/IH | steep, 1,H#/S |
| brick, 1,H#/B | intent, 1,H#/IH | stick, 1,H#/S |
| bricks, 1,H#/B | is, 1,H#/IH | still, 1,H#/S |
| brim, 1,H#/B | knee, 1,H#/N | straight, 1,H#/S |
| bring, 1,H#/B | knees, 1,H#/N | strain, 1,H#/S |
| build, 1,H#/B | knick-, 1,H#/N | strained, 1,H#/S |

268

| | | |
|---|---|---|
| building, 1,H#/B | lack, 1,H#/L | stranded, 1,H#/S |
| buy, 1,H#/B | lamp, 1,H#/L | stray, 1,H#/S |
| by, 1,H#/B | land, 1,H#/L | stress, 1,H#/S |
| call, 0 | late, 1,H#/L | sum, 1,H#/S |
| called, 0 | later, 1,H#/L | sun, 1,H#/S |
| calling, 0 | latest, 1,H#/L | tackle, 1,H#/T |
| camp, 0 | lay, 1,H#/L | tan, 1,H#/T |
| can, 0 | leap, 1,H#/L | tax, 1,H#/T |
| can't, 0 | least, 1,H#/L | taxi, 1,H#/T |
| canister, 0 | less, 1,H#/L | tea, 1,H#/T |
| canned, 0 | lest, 1,H#/L | tell, 1,H#/T |
| canteen, 0 | lie, 1,H#/L | ten, 1,H#/T |
| canter, 0 | lily, 1,H#/L | tenant, 1,H#/T |
| car, 0 | limit, 1,H#/L | tenant's, 1,H#/T |
| cease, 1,H#/S | liquid, 1,H#/L | tend, 1,H#/T |
| chorused, 0 | list, 1,H#/L | tended, 1,H#/T |
| clamp, 0 | listed, 1,H#/L | test, 1,H#/T |
| clay, 0 | lists, 1,H#/L | test-run, 1,H#/T |
| clean, 0 | lock, 1,H#/L | testicle, 1,H#/T |
| cleaned, 0 | locked, 1,H#/L | tests, 1,H#/T |
| cleaning, 0 | main, 1,H#/M | text, 1,H#/T |
| cleans, 0 | maintenance, 1,H#/M | texts, 1,H#/T |
| clever, 0 | mate, 1,H#/M | the, 0 |
| cling, 0 | mates, 1,H#/M | these, 0 |
| collie, 0 | may, 1,H#/M | tim, 1,H#/T |
| cory, 0 | money, 1,H#/M | tin, 1,H#/T |
| crack, 0 | must, 1,H#/M | track, 1,H#/T |
| cracked, 0 | my, 1,H#/M | tracks, 1,H#/T |
| crest, 0 | o', 1,H#/OW | train, 1,H#/T |
| crisp, 0 | of, 1,H#/AX | training, 1,H#/T |
| criss-, 0 | oh, 1,H#/OW | trait, 1,H#/T |
| cry, 0 | or, 1,H#/AO | traits, 1,H#/T |
| dig, 0 | origin, 1,H#/AO | transience, 1,H#/T |
| dim, 0 | plan, 1,H#/P | transit, 1,H#/T |
| discipline, 0 | plane, 1,H#/P | tray, 1,H#/T |

| disciplined, 0 | planes, 1,H#/P | tree, 1,H#/T |
|---|---|---|
| dish, 0 | planets, 1,H#/P | trees, 1,H#/T |
| dishes, 0 | planned, 1,H#/P | trend, 1,H#/T |
| disk, 0 | planning, 1,H#/P | trick, 1,H#/T |
| display, 0 | plans, 1,H#/P | trim, 1,H#/T |
| distance, 0 | plant, 1,H#/P | trish, 1,H#/T |
| distances, 0 | planted, 1,H#/P | try, 1,H#/T |
| distracted, 0 | plate, 1,H#/P | uneasily, 1,H#/AH |
| distress, 0 | plated, 1,H#/P | unseen, 1,H#/AH |
| districts, 0 | play, 1,H#/P | until, 1,H#/AX |
| drain, 0 | please, 1,H#/P | us, 1,H#/AH |
| dress, 0 | plenty, 1,H#/P | we, 0 |
| dressing, 0 | practical, 1,H#/P | we'll, 0 |
| dressy, 0 | practiced, 1,H#/P | weakens, 0 |
| dry, 0 | practices, 1,H#/P | week, 0 |
| ease, 1,H#/IY | princes, 1,H#/P | weeks, 0 |
| easily, 1,H#/IY | print, 1,H#/P | wick, 0 |
| eight, 1,H#/EY | prints, 1,H#/P | will, 0 |
| electric, 1,H#/AX | queen, 0 | win, 0 |
| electrical, 1,H#/AX | quick, 0 | winter, 0 |
| electricity, 1,H#/AX | quickly, 0 | wish, 0 |
| end, 1,H#/EH | quince, 0 | wrist, 1,H#/R |
| ended, 1,H#/EH | rack, 1,H#/R | x, 1,H#/EH |
| enter, 1,H#/EH | rain, 1,H#/R | |
| entity, 1,H#/EH | rains, 1,H#/R | |
| entries, 1,H#/EH | ran, 1,H#/R | |
| entry, 1,H#/EH | rate, 1,H#/R | |
| ever, 1,H#/EH | rated, 1,H#/R | |
| explain, 1,H#/IH | rates, 1,H#/R | |
| explains, 1,H#/IH | rating, 1,H#/R | |
| exquisite, 1,H#/EH | ray, 1,H#/R | |
| extended, 1,H#/IH | recall, 1,H#/R | |
| extending, 1,H#/IH | recalled, 1,H#/R | |
| extent, 1,H#/IH | rector, 1,H#/R | |
| eye, 1,H#/AY | related, 1,H#/R | |

| | | |
|---|---|---|
| fall, 1,H#/F | relaxed, 1,H#/R | |
| fallen, 1,H#/F | release, 1,H#/R | |
| fill, 1,H#/F | released, 1,H#/R | |
| filling, 1,H#/F | relish, 1,H#/R | |
| finish, 1,H#/F | rely, 1,H#/R | |
| finishing, 1,H#/F | remain, 1,H#/R | |
| fiscal, 1,H#/F | remained, 1,H#/R | |
| fish, 1,H#/F | remaining, 1,H#/R | |

# APPENDIX F. TRAINING RECORD FOR NETWORK 1 IN TABLE 4.3

## A.   DESCRIPTION

The training record in this appendix was produced automatically by TRNBPX.M while training Network 1 in Table 4.3. The record shows the three main training phases discussed in Chapter 4. First, the network was trained on just one training set example for each diphone (through epoch 79). Then, the network was trained without feedback-based time alignment until errors seemed to have reached a minimum (epochs 90 through 224, with the error minimum at epoch 196). Finally, training was completed with feedback-based time alignment (starting at epoch 197). Several additional training restarts are evident in this report. Those usually resulted from problems with the host (like system shutdowns for maintenance). Also, a few more errors were eliminated by lowering the MATLAB momentum and error ratio terms and the resuming training at epoch 268.

The weights from epoch 267 were used to perform the word tests described in Chapter IV. The (slightly better) statistics from epoch 278 were reported in Table 4.3.

The fields in each output line of the report are:

1. Epoch number.

2. TRAIN (indicating the data set that the next three fields refer to)

3. Training set Sum Squared Error (SSE) for the epoch

4. Training set number of diphones correct (out of 15,700 in this case).

5. Training set number of diphones not in top three

6. TEST (indicating the data set that the next three fields refer to)

7. Test set Sum Squared Error (SSE) for the epoch

8. Test set number of diphones correct (out of 15,700 in this case)

9. Test set number of diphones not in top three

10. A "-" to indicate SSE decreased (or at worst increased no more than allowed by the MATLAB error ratio parameter) or a "***" to indicate that the training set SSE actually increased as a result of the weight changes (which will cause TRNBPX.M to decrease the weights, reduce the learning rate, a zero out the momentum term for an epoch).

## B. TRAINING RECORD

```
------------------------------------------------
NNET TRAINING RECORD Ver. 11.0 (Epochs,SSEs,#Correct,#MoreThan3off)

Starting random with sqrt(ste) patch on sx100 ref set
New Weights, First Instance Only, NO BATCHING,
win_width = 13, max_offset = 1, seed = 0
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.950000 , er = 1.050000

Start Time = 1996/5/2  20:11:38
1, TRAIN,1995.848124,1,154, -
2, TRAIN,236.366505,9,135, -
3, TRAIN,157.278046,44,105, -
4, TRAIN,136.608430,60,86, -
5, TRAIN,119.669860,68,77, -
6, TRAIN,105.668917,84,64, -
7, TRAIN,94.050810,94,57, -
8, TRAIN,82.302249,101,50, -
9, TRAIN,74.967135,108,46, -
10, TRAIN,67.197963,114,40, -
11, TRAIN,58.882383,121,34, -
12, TRAIN,52.839936,125,31, -
13, TRAIN,47.260793,127,30, -
14, TRAIN,42.735514,127,27, -
15, TRAIN,38.549172,131,24, -
16, TRAIN,35.522931,134,22, -
17, TRAIN,33.105917,137,20, -
18, TRAIN,30.334545,137,20, -
19, TRAIN,28.447628,137,19, -
20, TRAIN,26.658648,138,18, -
21, TRAIN,25.123675,142,15, -
22, TRAIN,23.417798,143,14, -
23, TRAIN,20.889654,144,12, -
24, TRAIN,18.952710,145,12, -
25, TRAIN,16.897846,145,11, -
26, TRAIN,16.273867,145,11, -
27, TRAIN,15.786864,146,11, -
28, TRAIN,15.446025,146,11, -
29, TRAIN,15.159968,146,11, -
30, TRAIN,14.828331,147,10, -
31, TRAIN,14.415502,147,10, -
32, TRAIN,13.573549,147,10, -
33, TRAIN,13.233618,148,8, -
34, TRAIN,12.648216,150,7, -
35, TRAIN,10.177223,151,6, -
36, TRAIN,9.447141,151,5, -
37, TRAIN,8.180229,153,4, -
38, TRAIN,8.076680,155,2, -
39, TRAIN,6.640486,155,2, -
40, TRAIN,5.088253,155,2, -
41, TRAIN,4.035936,155,2, -
42, TRAIN,3.733584,155,2, -
43, TRAIN,3.540276,155,2, -
44, TRAIN,3.420515,155,2, -
45, TRAIN,3.315062,155,2, -
```

46, TRAIN,3.227354,155,2, -
47, TRAIN,3.149244,155,2, -
48, TRAIN,3.080232,155,2, -
49, TRAIN,3.013316,155,2, -
50, TRAIN,2.953998,155,2, -
51, TRAIN,2.900981,155,2, -
52, TRAIN,2.850587,155,2, -
53, TRAIN,2.802942,155,2, -
54, TRAIN,2.755167,155,2, -
55, TRAIN,2.705984,156,1, -
56, TRAIN,2.606806,156,1, -
57, TRAIN,1.719107,156,1, -
58, TRAIN,1.675180,156,1, -
59, TRAIN,1.617412,156,1, -
60, TRAIN,1.579698,156,1, -
61, TRAIN,1.548818,156,1, -
62, TRAIN,1.519882,156,1, -
63, TRAIN,1.493706,156,1, -
64, TRAIN,1.469554,156,1, -
65, TRAIN,1.447213,156,1, -
66, TRAIN,1.424741,156,1, -
67, TRAIN,1.403695,156,1, -
68, TRAIN,1.384237,156,1, -
69, TRAIN,1.366024,156,1, -
70, TRAIN,1.348750,156,1, -
71, TRAIN,1.331886,156,1, -
72, TRAIN,1.315760,156,1, -
73, TRAIN,1.300565,156,1, -
74, TRAIN,1.285063,156,1, -
75, TRAIN,1.268038,156,1, -
76, TRAIN,1.223247,157,0, -
77, TRAIN,21.170338,148,3, ***
78, TRAIN,118.198458,96,38, ***
79, TRAIN,4.269325,156,0, ***
----------------------------------------------
NNET TRAINING RECORD - Train Ver. 11.0 (Epochs,SSEs,#Correct,#MoreThan3off)

Resuming with patched STE -> sqrt(STE)
New Weights, First Instance Only, NO BATCHING,
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.950000 , er = 1.050000

Start Time = 1996/5/3  18:35:34
80, TRAIN,14780.718082,2364,11806,  TEST,4182.651892,758,3222, -
81, TRAIN,13678.547017,4028,9808,  TEST,4008.506783,1040,2871, -
82, TRAIN,12697.597643,5223,8533,  TEST,3885.237038,1216,2648, -
83, TRAIN,11948.871163,6098,7713,  TEST,3801.350937,1318,2457, -
84, TRAIN,11276.061962,6785,7032,  TEST,3733.762359,1447,2349, -
85, TRAIN,10737.701500,7319,6567,  TEST,3650.957049,1528,2248, -
86, TRAIN,10222.402759,7869,6063,  TEST,3667.035001,1541,2208, -
87, TRAIN,9788.995216,8307,5748,  TEST,3594.897039,1626,2112, -
88, TRAIN,9404.012764,8646,5482,  TEST,3581.229172,1641,2044, -
89, TRAIN,9048.987435,8953,5217,  TEST,3546.618377,1665,1999, -
90, TRAIN,8713.546272,9260,5020,  TEST,3537.740306,1684,1962, -

```
-----------------------------------------------
NNET TRAINING RECORD - Train Ver. 11.0 (Epochs,SSEs,#Correct,#MoreThan3off)

Resuming with patched STE -> sqrt(STE)
New Weights, First Instance Only, NO BATCHING,
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.950000 , er = 1.050000

Start Time = 1996/5/6  17:59:20
91, TRAIN,7975.354025,9881,4671,  TEST,3464.289652,1770,1881, -
92, TRAIN,7728.632578,10063,4513,  TEST,3455.937185,1786,1853, -
93, TRAIN,7537.637486,10269,4371,  TEST,3469.391083,1804,1850, -
94, TRAIN,7375.366315,10403,4227,  TEST,3446.215955,1818,1834, -
95, TRAIN,7195.011888,10606,4111,  TEST,3444.325959,1831,1776, -
96, TRAIN,7053.448063,10746,4012,  TEST,3427.014973,1856,1777, -
97, TRAIN,6881.492430,10881,3903,  TEST,3431.242658,1856,1787, -
98, TRAIN,6731.500129,11033,3818,  TEST,3412.183155,1899,1752, -
99, TRAIN,6572.206341,11169,3683,  TEST,3417.696492,1874,1757, -
100, TRAIN,6428.803687,11251,3611,  TEST,3406.952575,1891,1727, -
101, TRAIN,6280.780659,11386,3511,  TEST,3399.402109,1875,1697, -
102, TRAIN,6148.791768,11505,3395,  TEST,3402.027869,1883,1714, -
103, TRAIN,5994.132957,11641,3298,  TEST,3411.016047,1894,1653, -
104, TRAIN,5899.230026,11736,3219,  TEST,3404.408400,1919,1677, -
105, TRAIN,5726.669752,11864,3151,  TEST,3414.412410,1900,1685, -
106, TRAIN,5612.036223,11941,3072,  TEST,3398.611651,1934,1652, -
107, TRAIN,5489.838603,12028,3007,  TEST,3405.882105,1918,1661, -
108, TRAIN,5349.083391,12147,2934,  TEST,3398.237526,1922,1645, -
109, TRAIN,5241.211347,12215,2863,  TEST,3393.825371,1915,1668, -
110, TRAIN,5138.118627,12307,2810,  TEST,3405.201125,1898,1650, -
111, TRAIN,5038.979382,12366,2745,  TEST,3389.981944,1922,1638, -
112, TRAIN,4918.462776,12443,2715,  TEST,3404.274095,1913,1603, -
113, TRAIN,4824.130837,12497,2675,  TEST,3415.498804,1959,1636, -
114, TRAIN,4768.574050,12536,2642,  TEST,3403.473911,1935,1632, -
115, TRAIN,4637.086515,12629,2573,  TEST,3441.186401,1934,1649, -
116, TRAIN,4589.708875,12682,2552,  TEST,3422.591756,1921,1633, -
117, TRAIN,4493.310687,12751,2539,  TEST,3409.049080,1938,1666, -
118, TRAIN,4476.716017,12763,2487,  TEST,3444.327261,1923,1636, -
119, TRAIN,4429.008116,12785,2487,  TEST,3404.492692,1928,1628, -
120, TRAIN,4355.164895,12824,2458,  TEST,3435.040716,1938,1595, -
121, TRAIN,4282.983005,12881,2441,  TEST,3447.211630,1944,1640, -
122, TRAIN,4308.855568,12879,2429,  TEST,3478.964167,1922,1661, -
123, TRAIN,4099.213604,12955,2388,  TEST,3449.377926,1949,1634, -
124, TRAIN,4021.061916,12996,2377,  TEST,3452.047168,1921,1624, -
125, TRAIN,3952.510356,13044,2355,  TEST,3432.037867,1937,1642, -
126, TRAIN,3959.011646,13027,2351,  TEST,3461.000658,1943,1627, -
127, TRAIN,3911.466353,13078,2311,  TEST,3482.624967,1899,1642, -
128, TRAIN,3925.696154,13086,2302,  TEST,3499.253097,1897,1653, -
129, TRAIN,3755.161957,13136,2294,  TEST,3447.783052,1929,1588, -
130, TRAIN,3642.815471,13170,2276,  TEST,3493.262211,1907,1645, -
131, TRAIN,3759.604539,13140,2237,  TEST,3504.411546,1926,1630, -
132, TRAIN,3647.236436,13219,2220,  TEST,3466.460217,1949,1595, -
133, TRAIN,3631.891870,13218,2178,  TEST,3482.082856,1938,1622, -
134, TRAIN,3693.017063,13238,2144,  TEST,3531.928843,1911,1607, -
135, TRAIN,3619.030788,13283,2142,  TEST,3549.534705,1905,1630, -
136, TRAIN,3633.557710,13271,2113,  TEST,3509.971139,1928,1626, -
137, TRAIN,3612.694414,13292,2118,  TEST,3569.236360,1883,1632, -
138, TRAIN,3746.247789,13263,2093,  TEST,3576.428130,1909,1640, -
```

139, TRAIN,3671.266506,13277,2101,  TEST,3545.369132,1923,1623, -
140, TRAIN,3771.082261,13241,2081,  TEST,3620.893958,1864,1660, -
141, TRAIN,3661.457360,13302,2061,  TEST,3605.696769,1913,1642, -
142, TRAIN,3658.837530,13294,2043,  TEST,3601.342024,1896,1649, -
143, TRAIN,3905.667369,13222,2040,  TEST,3598.161397,1863,1644, ***
144, TRAIN,2981.309782,13503,2004,  TEST,3480.571218,1958,1579, -
145, TRAIN,2599.014086,13582,1993,  TEST,3464.565822,1981,1573, -
146, TRAIN,2472.661190,13611,1958,  TEST,3465.696400,1981,1529, -
147, TRAIN,2401.042949,13644,1928,  TEST,3465.150248,2002,1533, -
148, TRAIN,2355.773346,13661,1934,  TEST,3468.859565,1977,1524, -
149, TRAIN,2310.977415,13690,1905,  TEST,3455.253232,1995,1546, -
150, TRAIN,2260.435881,13711,1887,  TEST,3469.646080,1992,1514, -
151, TRAIN,2232.840560,13719,1881,  TEST,3457.880977,1983,1528, -
152, TRAIN,2265.422478,13743,1846, `TEST,3503.785742,2002,1544, -
153, TRAIN,2319.559276,13756,1841,  TEST,3516.270730,1955,1576, -
154, TRAIN,2419.891110,13766,1818,  TEST,3545.553271,1951,1578, -
155, TRAIN,2719.455478,13717,1808,  TEST,3632.305073,1875,1614, ***
156, TRAIN,2221.428023,13790,1784,  TEST,3465.320664,1982,1524, -
157, TRAIN,2066.500580,13838,1757,  TEST,3454.490694,1994,1527, -
158, TRAIN,2028.797252,13861,1755,  TEST,3474.298621,1984,1514, -
159, TRAIN,1999.473567,13879,1731,  TEST,3458.909890,1975,1508, -
160, TRAIN,1973.425495,13899,1720,  TEST,3461.074925,1983,1495, -
161, TRAIN,1946.236275,13920,1689,  TEST,3455.748942,2000,1508, -
162, TRAIN,1927.740203,13943,1677,  TEST,3458.979386,2008,1492, -
163, TRAIN,1909.084655,13947,1670,  TEST,3466.627587,1991,1499, -
164, TRAIN,1896.128762,13957,1653,  TEST,3475.295782,1998,1500, -
165, TRAIN,1884.979569,13984,1643,  TEST,3468.432737,2008,1516, -
166, TRAIN,1891.313717,13990,1636,  TEST,3487.016498,2013,1527, -
167, TRAIN,1901.362461,14002,1626,  TEST,3493.379447,1987,1515, -
168, TRAIN,2027.588978,13997,1611,  TEST,3576.740083,1937,1592, ***
169, TRAIN,1842.233871,14016,1611,  TEST,3471.469868,2000,1475, -
170, TRAIN,1794.783875,14039,1597,  TEST,3459.058367,2015,1484, -
171, TRAIN,1779.636690,14047,1588,  TEST,3462.125653,2006,1480, -
172, TRAIN,1764.067088,14059,1574,  TEST,3468.776442,2012,1506, -
173, TRAIN,1748.054207,14074,1573,  TEST,3466.698850,2005,1501, -
174, TRAIN,1735.403566,14087,1561,  TEST,3472.019076,2021,1514, -
175, TRAIN,1721.130731,14093,1557,  TEST,3471.063414,2009,1511, -
176, TRAIN,1709.732409,14102,1537,  TEST,3468.940027,2014,1486, -
177, TRAIN,1714.751268,14112,1539,  TEST,3474.116880,2000,1489, -
178, TRAIN,1727.191214,14120,1534,  TEST,3467.444379,1993,1516, -
179, TRAIN,1718.408081,14125,1521,  TEST,3505.280511,1989,1546, -
180, TRAIN,1763.553048,14129,1520,  TEST,3542.559993,1984,1531, -
181, TRAIN,4431.622547,13011,1775,  TEST,3900.772867,1638,1900, ***
182, TRAIN,1724.128388,14132,1517,  TEST,3494.324011,1998,1495, -
183, TRAIN,1645.903958,14149,1510,  TEST,3480.092776,2014,1499, -
184, TRAIN,1627.408195,14153,1503,  TEST,3471.566624,2027,1500, -
185, TRAIN,1616.956389,14160,1505,  TEST,3470.751533,2013,1505, -
186, TRAIN,1610.109825,14167,1500,  TEST,3473.814005,2015,1495, -
187, TRAIN,1605.054405,14174,1493,  TEST,3466.090111,2012,1494, -
188, TRAIN,1593.889004,14182,1489,  TEST,3468.167975,2002,1496, -
189, TRAIN,1583.187422,14185,1487,  TEST,3471.082698,2011,1499, -
190, TRAIN,1576.221393,14189,1488,  TEST,3481.495820,2012,1485, -
191, TRAIN,1575.436975,14188,1479,  TEST,3472.419597,2012,1484, -
192, TRAIN,1565.743332,14194,1476,  TEST,3464.885398,2022,1488, -
193, TRAIN,1563.520329,14196,1467,  TEST,3485.699446,2000,1502, -
194, TRAIN,6501.681305,11382,2523,  TEST,3992.711208,1594,2006, ***

277

195, TRAIN,1567.098827,14199,1468, TEST,3462.907920,2016,1472, -
196, TRAIN,1548.229053,14206,1465, TEST,3465.566050,2008,1472, -
197, TRAIN,1544.322378,14207,1461, TEST,3468.253940,2007,1485, -
198, TRAIN,1539.963259,14214,1459, TEST,3475.844092,2006,1488, -
199, TRAIN,1536.591375,14219,1454, TEST,3466.133498,2039,1480, -
200, TRAIN,1532.129389,14222,1449, TEST,3467.418349,2019,1475, -
201, TRAIN,1530.154930,14222,1446, TEST,3464.487411,2019,1502, -
202, TRAIN,1531.010608,14232,1440, TEST,3470.526331,2018,1498, -
203, TRAIN,1553.950750,14231,1436, TEST,3491.823689,2017,1497, -
204, TRAIN,5621.547718,12048,2156, TEST,4009.930183,1587,1994, ***
205, TRAIN,1532.471058,14239,1438, TEST,3478.284700,2023,1510, -
206, TRAIN,1509.441995,14244,1434, TEST,3468.126097,2014,1511, -
207, TRAIN,1503.407466,14246,1427, TEST,3465.847702,2011,1504, -
208, TRAIN,1497.143728,14246,1424, TEST,3469.082578,2010,1496, -
209, TRAIN,1493.893521,14248,1420, TEST,3471.948863,2013,1489, -
210, TRAIN,1494.085322,14252,1423, TEST,3463.616969,2013,1511, -
211, TRAIN,1486.634056,14252,1417, TEST,3466.970954,2010,1497, -
212, TRAIN,1485.370179,14257,1412, TEST,3470.612800,1997,1486, -
213, TRAIN,1987.182866,14097,1426, TEST,3796.316944,1769,1823, ***
214, TRAIN,1481.538272,14261,1413, TEST,3469.187955,2009,1497, -
215, TRAIN,1475.516999,14267,1408, TEST,3468.394800,2015,1504, -
216, TRAIN,1473.540406,14268,1405, TEST,3469.526205,2011,1496, -
217, TRAIN,1468.949037,14274,1402, TEST,3475.300060,2005,1488, -
218, TRAIN,1466.651845,14275,1400, TEST,3472.505020,2005,1488, -
219, TRAIN,1465.825320,14276,1397, TEST,3480.681690,1990,1473, -
220, TRAIN,1460.092556,14281,1398, TEST,3464.965838,2009,1487, -
221, TRAIN,1456.818287,14280,1396, TEST,3473.743757,1999,1486, -
222, TRAIN,1454.127846,14281,1393, TEST,3476.276312,2013,1479, -
223, TRAIN,1451.080625,14285,1390, TEST,3477.229580,2008,1492, -
224, TRAIN,1448.633083,14285,1391, TEST,3474.922538,2005,1484, -
------------------------------------------------
NNET TRAINING RECORD Ver. 11.0 (L1) (Epochs,SSEs,#Correct,#MoreThan3off)

Starting +/- 2 non-fixed TA on L1 SX100 sep tst set
Same Weights, Entire File Set, NO BATCHING, Time-Aligning, Separate Test Set,
win_width = 13, max_offset = 2, seed = 0
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.950000 , er = 1.050000

Start Time = 1996/5/19  12:44:48
197, TRAIN,1513.541055,14604,829, TEST,2518.772806,3083,603, -
198, TRAIN,1439.394587,14621,820, TEST,2514.301597,3097,596, -
199, TRAIN,1386.347128,14643,810, TEST,2510.776058,3086,600, -
200, TRAIN,1343.211479,14661,813, TEST,2503.034539,3097,593, -
201, TRAIN,1333.411093,14688,787, TEST,2502.422200,3094,603, -
202, TRAIN,1290.893539,14702,778, TEST,2489.706907,3099,600, -
203, TRAIN,1256.046615,14720,764, TEST,2494.426775,3101,599, -
204, TRAIN,1225.487427,14759,744, TEST,2494.226942,3100,596, -
205, TRAIN,1195.314909,14790,724, TEST,2488.542763,3086,585, -
206, TRAIN,1182.140609,14817,702, TEST,2489.232357,3090,586, -
207, TRAIN,1142.391678,14818,696, TEST,2487.928071,3081,597, -
208, TRAIN,1113.966376,14832,682, TEST,2481.320034,3094,589, -
209, TRAIN,1089.975837,14844,680, TEST,2479.224909,3102,583, -
210, TRAIN,1067.478307,14858,665, TEST,2476.414089,3092,566, -
211, TRAIN,1072.047388,14875,663, TEST,2474.922586,3074,561, -
212, TRAIN,1041.320592,14891,630, TEST,2473.923165,3085,568, -
213, TRAIN,1019.214789,14912,609, TEST,2476.397882,3089,578, -

214, TRAIN,997.339690,14947,601,  TEST,2473.896079,3112,580, -
215, TRAIN,971.537714,14958,588,  TEST,2470.376026,3086,590, -
216, TRAIN,968.075922,14987,560,  TEST,2487.191149,3081,578, -
---------------------------------------------
NNET TRAINING RECORD Ver. 11.0 (L1 Resume) (Epochs,SSEs,#Correct,#MoreThan3off)

Resuming with same wts & momentum
Same Weights, Entire File Set, NO BATCHING, Time-Aligning, Separate Test Set,
win_width = 13, max_offset = 2, seed = 0
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.950000 , er = 1.050000

Start Time = 1996/5/23  10:5:2
217, TRAIN,945.143514,14995,553,  TEST,2480.250340,3084,597, -
218, TRAIN,912.372877,15007,544,  TEST,2475.951859,3082,570, -
219, TRAIN,885.840668,15024,546,  TEST,2474.175137,3069,595, -
220, TRAIN,866.105779,15035,547,  TEST,2473.650913,3076,584, -
221, TRAIN,857.757543,15049,535,  TEST,2479.908912,3065,579, -
222, TRAIN,831.890409,15062,529,  TEST,2469.921061,3082,581, -
223, TRAIN,809.709769,15070,520,  TEST,2470.608726,3074,587, -
224, TRAIN,793.529201,15077,514,  TEST,2464.441310,3076,574, -
225, TRAIN,775.868605,15089,503,  TEST,2468.215410,3065,568, -
226, TRAIN,782.257357,15100,506,  TEST,2474.483716,3075,577, -
227, TRAIN,752.943357,15111,495,  TEST,2474.985343,3072,570, -
228, TRAIN,735.999096,15122,491,  TEST,2471.898707,3085,563, -
229, TRAIN,722.462653,15127,480,  TEST,2475.751305,3088,557, -
230, TRAIN,709.343887,15140,480,  TEST,2468.132626,3086,565, -
231, TRAIN,719.088278,15144,478,  TEST,2475.642410,3044,563, -
232, TRAIN,695.355814,15151,478,  TEST,2477.686291,3089,569, -
233, TRAIN,677.722562,15159,470,  TEST,2470.165935,3066,576, -
234, TRAIN,661.994739,15167,465,  TEST,2469.918242,3083,557, -
235, TRAIN,653.034637,15172,456,  TEST,2479.341104,3089,565, -
236, TRAIN,666.865135,15182,448,  TEST,2486.021364,3057,580, -
237, TRAIN,642.709643,15194,430,  TEST,2474.226124,3070,568, -
238, TRAIN,626.330215,15205,419,  TEST,2463.815564,3078,559, -
239, TRAIN,608.809529,15226,397,  TEST,2469.151962,3067,558, -
240, TRAIN,598.502574,15245,393,  TEST,2471.260917,3081,547, -
241, TRAIN,631.495208,15264,357,  TEST,2511.615090,3000,614, ***
242, TRAIN,592.654847,15263,361,  TEST,2477.923453,3078,552, -
243, TRAIN,554.659526,15273,348,  TEST,2477.848812,3065,536, -
244, TRAIN,537.495798,15283,341,  TEST,2466.366135,3084,534, -
245, TRAIN,524.297491,15291,344,  TEST,2456.184719,3089,540, -
246, TRAIN,538.024469,15299,342,  TEST,2474.914041,3064,550, -
247, TRAIN,514.665300,15307,331,  TEST,2460.360019,3091,541, -
248, TRAIN,496.897055,15313,320,  TEST,2462.603719,3105,530, -
249, TRAIN,488.029933,15316,324,  TEST,2464.524234,3095,537, -
250, TRAIN,476.621329,15325,314,  TEST,2449.183599,3103,526, -
251, TRAIN,495.044536,15328,315,  TEST,2468.722743,3064,530, -
252, TRAIN,481.718293,15333,310,  TEST,2490.397923,3044,547, -
253, TRAIN,459.882916,15338,310,  TEST,2474.602249,3075,531, -
254, TRAIN,450.579953,15345,302,  TEST,2453.664513,3111,544, -
255, TRAIN,441.116910,15348,300,  TEST,2461.397616,3108,525, -
256, TRAIN,450.966711,15354,300,  TEST,2454.388121,3089,539, -
257, TRAIN,446.203603,15357,293,  TEST,2482.220462,3063,555, -
258, TRAIN,676.023889,15321,296,  TEST,2717.716462,2730,835, ***
259, TRAIN,431.930594,15366,289,  TEST,2472.641715,3078,540, -
260, TRAIN,408.030417,15371,293,  TEST,2453.043495,3120,527, -

279

261, TRAIN,409.799589,15376,286,  TEST,2454.607747,3123,523, -
262, TRAIN,393.362240,15376,281,  TEST,2446.588135,3136,528, -
263, TRAIN,388.592518,15379,278,  TEST,2440.910335,3133,527, -
264, TRAIN,384.084181,15383,275,  TEST,2440.398535,3142,523, -
265, TRAIN,380.538080,15388,268,  TEST,2447.538966,3133,510, -
266, TRAIN,396.418548,15392,271,  TEST,2467.503174,3100,552, -
267, TRAIN,380.869589,15397,266,  TEST,2450.123434,3104,540, -
268, TRAIN,370.767868,15402,256,  TEST,2446.735704,3114,530, -
269, TRAIN,374.873139,15402,254,  TEST,2453.847634,3095,544, -
270, TRAIN,377.943383,15410,256,  TEST,2477.396752,3074,546, -
271, TRAIN,592.656381,15387,254,  TEST,2794.888396,2707,811, ***
272, TRAIN,372.435764,15415,260,  TEST,2480.007066,3076,532, -
273, TRAIN,344.742953,15418,252,  TEST,2444.746685,3103,527, -
274, TRAIN,339.229930,15421,252,  TEST,2442.835629,3118,533, -
275, TRAIN,331.461420,15425,252,  TEST,2447.018322,3136,525, -
276, TRAIN,334.551198,15427,249,  TEST,2446.632702,3105,526, -
277, TRAIN,325.287235,15431,245,  TEST,2440.712682,3130,516, -
278, TRAIN,318.943251,15430,248,  TEST,2448.608129,3107,527, -
279, TRAIN,315.133087,15431,244,  TEST,2449.359481,3120,525, -
280, TRAIN,313.279966,15433,245,  TEST,2442.676588,3132,524, -
281, TRAIN,314.683572,15436,241,  TEST,2455.770905,3106,537, -
282, TRAIN,313.312689,15438,239,  TEST,2454.417279,3122,523, -
283, TRAIN,305.167799,15441,238,  TEST,2456.547355,3107,535, -
284, TRAIN,303.674744,15441,236,  TEST,2445.312476,3114,520, -
285, TRAIN,302.557163,15444,239,  TEST,2451.775172,3117,522, -
286, TRAIN,1669.232315,14800,412,  TEST,3333.798841,2153,1320, ***
287, TRAIN,306.806272,15444,234,  TEST,2467.002340,3109,522, -
288, TRAIN,295.763487,15444,231,  TEST,2436.862463,3121,528, -
289, TRAIN,293.188221,15447,233,  TEST,2443.595743,3137,521, -
290, TRAIN,292.283630,15448,232,  TEST,2440.009398,3140,523, -
------------------------------------------------
NNET TRAINING RECORD Ver. 11.0 (L1) (Epochs,SSEs,#Correct,#MoreThan3off)

starting again at e265 (bestwts)
Same Weights, Entire File Set, NO BATCHING, Time-Aligning, Separate Test Set,
win_width = 13, max_offset = 2, seed = 0
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.950000 , er = 1.050000

Start Time = 1996/6/7  14:2:13
266, TRAIN,378.000310,15396,264,  TEST,2444.860403,3156,510, -
**267, TRAIN,372.059882,15399,264,  TEST,2446.966214,3151,506, -**
268, TRAIN,369.200690,15401,261,  TEST,2447.490611,3152,507, -
269, TRAIN,367.251847,15402,262,  TEST,2446.306237,3144,510, -
270, TRAIN,365.497574,15402,257,  TEST,2451.426672,3150,510, -
------------------------------------------------
NNET TRAINING RECORD Ver. 11.0 (L1) (Epochs,SSEs,#Correct,#MoreThan3off)

Resuming at bestwts with lower mc & er
Same Weights, Entire File Set, NO BATCHING, Time-Aligning, Separate Test Set,
win_width = 13, max_offset = 2, seed = 0
logsig/logsig: Hidden = 256, Num_Dips = 157 , mc = 0.900000 , er = 1.040000

Start Time = 1996/6/9  0:39:1
268, TRAIN,368.908046,15400,261,  TEST,2447.784763,3151,506, -
269, TRAIN,366.809044,15402,259,  TEST,2447.117671,3149,510, -
270, TRAIN,365.067769,15402,258,  TEST,2449.033369,3153,506, -

271, TRAIN,363.903121,15403,257, TEST,2448.651252,3154,505, -
272, TRAIN,362.567736,15404,255, TEST,2447.202886,3155,509, -
273, TRAIN,360.546158,15405,255, TEST,2444.412520,3157,509, -
274, TRAIN,358.825661,15409,254, TEST,2445.037272,3158,505, -
275, TRAIN,357.092112,15411,254, TEST,2442.857028,3160,501, -
276, TRAIN,355.100194,15413,253, TEST,2440.393299,3159,501, -
277, TRAIN,355.513832,15414,253, TEST,2441.208379,3158,498, -
**278, TRAIN,352.115412,15413,253, TEST,2439.095758,3154,496, -**
279, TRAIN,349.741196,15416,251, TEST,2437.164771,3157,501, -
280, TRAIN,348.319405,15414,251, TEST,2436.337215,3150,498, -
281, TRAIN,346.461842,15416,253, TEST,2440.241266,3157,502, -
282, TRAIN,343.155424,15416,252, TEST,2440.484154,3157,502, -
283, TRAIN,340.700546,15417,251, TEST,2439.166613,3149,506, -
284, TRAIN,338.867598,15419,252, TEST,2436.070799,3154,507, -
285, TRAIN,337.138286,15419,252, TEST,2435.799594,3152,503, -
286, TRAIN,335.532533,15421,252, TEST,2435.036998,3156,503, -
287, TRAIN,336.559977,15422,254, TEST,2439.458499,3159,505, -
288, TRAIN,334.654073,15424,253, TEST,2437.624035,3149,507, -
289, TRAIN,332.514629,15425,254, TEST,2438.671405,3144,509, -

# APPENDIX G.  WORD RECOGNITION RESULTS

## A.   DESCRIPTION

The following report was produced by RECOGNIZ.CL during the word tests described in Chapter IV.  This report is intended for statistical purposes, not syntactic or semantic analysis. Each line in the report is a LISP list with nine elements:

1. The sample number where the correct word appeared strongest in the output stream (zero if the word was not detected).

2. The correct word.

3. The word confidence value for the word candidate described in items one and two (zero if the word was not detected).

4. The ranking of the correct word among all word candidates with the same starting time (one if the word was not detected).

5. The global ranking of the correct word (among all words in the output stream).  This is only marginally useful in continuous speech recognition, as discussed in the Chapter IV (the total number of word candidates if the correct word was not detected).

6. The strongest word candidate with the same start time as the correct word (nil if the correct word was not detected).

7. The word confidence value for the word candidate described in item six.

8. The strongest word candidate among all words in the output stream.

9. The word confidence value for the word candidate described in item eight.

## B.   RESULTS

(3933 are 0.1873845 7 482 ART 0.6646775 IN 0.9814075)
(0 for 0 1 871 nil 0 NEXT 0.7313204)
(2736 quick 0.6811147 1 16 QUICK 0.6811147 WICK 0.960505)
(2052 in 0.46714053 15 44 LINE 0.67273206 AM 0.922388)
(3078 in 0.4691025 1 7 IN 0.4691025 THE 0.929525)
(0 consists 0 1 1644 nil 0 THE 0.761428)
(3591 of 0.821699 1 1 OF 0.821699 OF 0.821699)
(4959 and 0.14384134 10 264 LIE 0.38751552 LIE 0.748632)
(2052 are 0.127771 4 365 US 0.49286598 US 0.971366)
(3420 made 0.5320155 9 28 MAY 0.994712 MAY 0.994712)
(3249 of 0.0054145 1 384 OF 0.0054145 AM 0.8061645)
(3591 are 0.482579 5 20 ROAM 0.50531733 BE 0.946454)
(2394 in 0.5402075 8 20 DIM 0.94087 DIM 0.94087)

(5130 stockings 0.29724526 59 1116 TORN 0.8617256 US 0.99528396)

(3762 run 0.304104 1 230 RUN 0.304104 MAY 0.794837)

(4788 dishes 0.27224898 21 693 DENSE 0.71203 HE 0.925552)

(2736 makes 0.93518746 2 3 AM 0.988569 AM 0.988569)

(4275 and 0.029465998 11 563 AIN~T 0.27141333 IT 0.81334496)

(2052 intelligence 0.40138015 7 274 INTENT 0.6046135 TEN 0.986287)

(2907 is 0.20747401 6 77 EAT 0.4195245 TELL 0.555865)

(0 for 0 1 770 nil 0 ILL 0.835065)

(0 become 0 1 1523 nil 0 BE 0.999408)

(5130 artists 0.3802265 6 113 RIM 0.528966 WE 0.955054)

(5130 fish 0.950968 1 1 FISH 0.950968 FISH 0.950968)

(4617 leap 0.5617575 1 7 LEAP 0.5617575 KNEE 0.870113)

(4446 frantically 0.7132638 1 20 FRANTICALLY 0.7132638 LEAF 0.955064)

(2736 of 0.120827004 5 202 SEEN 0.422501 IS 0.63039)

(12312 cat 0.73065 1 29 AT 0.73065 TELL 0.942454)

(3762 my 0.752832 3 3 MIND 0.814682 MIND 0.814682)

(2907 by 0.999798 1 1 BUY 0.999798 BUY 0.999798)

(1539 of 0.3058325 3 48 AGREE 0.32896432 RAIN 0.495736)

(3420 may 0.942935 1 1 MAY 0.942935 MAY 0.942935)

(2907 money 0.783675 4 18 MY 0.8917825 KNEE 0.996276)

(3933 by 0.998938 1 1 BUY 0.998938 BUY 0.998938)

(0 hard 0 1 1835 nil 0 ARE 0.980915)

(6156 candy 0.833961 1 6 CANDY 0.833961 TEA 0.9668495)

(1881 are 0.4910435 2 66 COLLIE 0.4913785 REAL 0.976672)

(0 for 0 1 553 nil 0 LEAD 0.78059447)

(4275 big 0.762662 3 3 IN 0.8091895 IN 0.8091895)

(4275 ambled 0.35976335 2 51 MILK 0.36078367 PLAN 0.61251634)

(2565 increases 0.137292 24 824 NEAREST 0.5557577 EASE 0.969679)

(1026 is 0.254825 6 218 WRIST 0.613279 ARE 0.980894)

(0 controlled 0 1 1429 nil 0 LOT 0.917583)

(3591 from 0.3059175 13 316 ROB 0.60963464 ARE 0.981709)

(1368 is 0.4983955 14 96 EAR 0.973136 EAR 0.973136)

(0 for 0 1 634 nil 0 CALF 0.7700615)

(4104 buyer 0.98096 3 3 BUY 0.997935 BUY 0.997935)

(3762 money 0.88202864 2 8 US 0.9071145 ACE 0.9591505)

(0 for 0 1 437 nil 0 EASE 0.9729785)

(5814 all 0.250349 11 225 LET 0.43589398 SALT 0.570993)

(1881 are 0.49441198 6 73 LOCK 0.98185 LOCK 0.9870567)

(4275 expected 0.56739616 17 103 STRUCK 0.8219268 LOCK 0.87118053)

(2907 comes 0.311604 12 312 DOLL 0.56631 ELSE 0.9000225)

(4788 for 0.12270367 7 362 RUN 0.493005 NONE 0.933492)

(1710 his 0.5756375 1 2 HIS 0.5756375 THE 0.580016)

(4104 broken 0.57375675 12 37 ROCK 0.8566167 PA 0.871716)

(2565 he 0.9774945 1 1 HE 0.9774945 HE 0.9774945)

(0 from 0 1 431 nil 0 ARM 0.7998555)

(0 every 0 1 752 nil 0 EAT 0.9720475)

(3591 in 0.54388297 5 13 IT 0.61600196 SHE 0.7135)

(2052 of 0.128311 6 367 ODD 0.413048 KNEE 0.904689)

(0 cleans 0 1 1309 nil 0 ODD 0.8228425)

(3762 for 0.4036405 3 36 ALL 0.7619205 ILL 0.94701254)

(0 every 0 1 922 nil 0 TEA 0.7970245)

(1368 related 0.27530575 11 362 ROCKETS 0.53542477 AT 0.92029953)

(0 my 0 1 482 nil 0 DOLL 0.679531)

(3420 ability 0.416951 10 54 BILL 0.6112095 BUY 0.984153)

(3249 is 0.178628 3 83 WICK 0.506303 JAW 0.513643)

(3249 made 0.042727 13 437 ATE 0.124715 TREE 0.6406955)

(3420 of 0.06788 2 370 VAULT 0.091452 TREE 0.545698)

(2394 are 0.219975 17 195 ROCK 0.659722 AM 0.911262)

(3420 in 0.49384397 2 15 KNICK- 0.514528 THE 0.933759)

(2736 stockings 0.76056767 4 7 SAY 0.90993404 NONE 0.9181985)

(2223 run 0.66080195 1 4 RUN 0.66080195 DONE 0.89926696)

(5301 dishes 0.481654 1 52 DISHES 0.481654 DISH 0.9055015)

(2907 makes 0.881495 1 8 MAKES 0.881495 BUY 0.99428)

(0 and 0 1 335 nil 0 TEN 0.7172965)

(4446 is 0.38257 2 71 LESS 0.4374865 HE 0.9108505)

(5130 for 0.738793 5 16 ON 0.90576303 KNEE 0.9634295)

(4959 ended 0.23284724 11 276 DUMB 0.52641404 TEA 0.977053)

(4788 please 0.36718166 21 296 EAT 0.6738665 THESE 0.89815354)

(2736 cleaners 0.504659 16 124 KEY 0.7781905 KNEES 0.9892255)

(3933 for 0.515878 1 11 OR 0.515878 ADD 0.6838985)

(1539 cleaners 0.33306125 34 533 QUEEN 0.61019766 KNEE 0.87358904)

(6156 all 0.121824 15 304 LOCKED 0.33104834 US 0.934741)

(4446 are 0.866431 1 2 ARE 0.866431 ILL 0.9536)

(3249 expected 0.61259025 3 40 EXPECT 0.7148502 PECK 0.930991)

(2736 comes 0.33323964 7 182 CAUGHT 0.495526 RAW 0.9304695)

(0 for 0 1 565 nil 0 AIN~T 0.9800635)

(2907 his 0.577604 1 2 HIS 0.577604 THE 0.663486)

(4446 broken 0.143412 39 1026 I~LL 0.4946285 TELL 0.864295)

(3420 he 0.57342803 1 4 HE 0.57342803 KNEE 0.69167054)

(4446 from 0.5095253 1 10 FROM 0.5095253 RUN 0.66486704)

(6156 every 0.7046505 2 3 EVER 0.92598 EASE 0.965269)

(2565 in 0.2646255 10 58 IF 0.5510375 IF 0.5510375)

(3762 of 0.022181 18 571 LEAST 0.30533427 KNEE 0.48568553)

(4788 cleaned 0.25117275 14 81 RAY 0.39502448 LEAF 0.5082985)

(3933 for 0.5500345 3 8 ALL 0.6511795 ILL 0.781802)

(7353 are 0.812259 1 5 ARE 0.812259 KNEE 0.999797)

(4446 all 0.330042 2 115 CORY 0.53609467 BE 0.859098)

(4959 errors 0.7802375 2 6 RAW 0.8750295 OR 0.910335)

(2907 nearest 0.28977123 6 219 NEAR 0.46660066 ERRORS 0.846212)

(3591 may 0.999945 1 1 MAY 0.999945 MAY 0.999945)

(2907 be 0.966185 1 2 BE 0.966185 TEA 0.980755)

(3078 distance 0.534456 14 45 DIM 0.79010266 IS 0.984571)

(3078 buys 0.8794905 1 1 EYES 0.8794905 EYES 0.8794905)

(0 catastrophic 0 1 2944 nil 0 CAST 0.977149)

(4104 intelligence 0.8807526 1 7 INTELLIGENCE 0.8807526 EDGE 0.9434935)

(0 is 0 1 399 nil 0 KNEE 0.7552695)

(3933 for 0.518121 2 31 REAL 0.51882535 KEY 0.95744205)

(0 become 0 1 1646 nil 0 WE 0.96533096)

(0 artists 0 1 1954 nil 0 TEA 0.992695)

(4446 fish 0.77679 1 2 FISH 0.77679 THEY 0.81535)

(3591 leap 0.881914 1 6 LEAP 0.881914 KNEE 0.976488)

(4275 frantically 0.68338925 7 50 ROCK 0.87625533 KNEE 0.99904)

(2907 of 0.533668 1 27 OF 0.533668 SOME 0.95681155)

(3249 did 0.3929135 10 78 IN 0.861815 IN 0.861815)

(3762 buy 0.663005 1 4 BUY 0.663005 DAY 0.951262)

(3249 any 0.16541149 1 135 KNEE 0.16541149 RUN 0.57370245)

(4104 bright 0.42460397 38 319 BREAK 0.86566097 ARM 0.983112)

(1881 red 0.054182995 32 639 RUN 0.61986834 RUN 0.89469945)

(3249 his 0.035689 25 472 SICK 0.150853 EAT 0.95563495)

(0 for 0 1 524 nil 0 EAT 0.836613)

(1881 is 0.3676555 11 110 LIST 0.827277 SEA 0.965967)

(3591 am 0.55402553 3 41 AN 0.810462 RAIN 0.9514865)

(3762 and 0.134138 17 131 IT 0.727708 KNIT 0.794451)

(513 are 0.0644365 17 136 ART 0.47869247 ACT 0.61233234)

(513 and 0.32305935 19 223 TRAIT 0.48860264 IT 0.9717835)

(1710 are 0.401409 1 19 ARE 0.401409 AIR 0.6997295)

(3762 spend 0.82212704 2 5 SPUN 0.9974505 SPUN 0.9974505)

(1026 in 0.52595 1 43 IN 0.52595 MY 0.99193)

(3249 miami 0.94332004 4 14 MY 0.988132 NUMB 0.9886405)

(3249 is 0.52389604 1 16 IS 0.52389604 EASE 0.976816)

(3078 be 0.936814 1 1 BE 0.936814 BE 0.936814)

(3933 need 0.4806575 7 59 KNEE 0.941606 KNEE 0.941606)

(6498 for 0.06872866 2 283 ROCK 0.073731996 SOME 0.904341)

(1539 and 0.095671676 30 450 DOOR 0.5400325 RAW 0.81501305)

(3762 in 0.5900825 2 11 KNICK- 0.6018045 EASE 0.828404)

(4275 big 0.698352 1 1 BIG 0.698352 BIG 0.698352)

(2394 can 0.03508467 8 379 KNEE 0.481827 BE 0.993945)

(3591 be 0.991809 1 1 BE 0.991809 BE 0.991809)

(3078 an 0.54280347 1 9 AN 0.54280347 BE 0.991342)

(2907 sport 0.5554825 5 15 SPHERE 0.6081026 IS 0.943873)

(4788 are 0.060823 8 533 ARC 0.492294 TELL 0.6391855)

(1881 or 0.002651 31 403 ERA 0.715299 ERA 0.98236954)

(0 proceeding 0 1 2498 nil 0 US 0.97203505)

(3933 grows 0.63795096 8 17 ROLE 0.8424557 ROLE 0.8424557)

(1881 he 0.580403 2 5 HIS 0.6197465 THE 0.774014)

(4617 from 0.49266002 3 11 RIM 0.509305 ARE 0.891187)
(4446 every 0.2982155 5 174 VERY 0.551038 IF 0.8537075)
(2394 in 0.66372347 1 1 IN 0.66372347 IN 0.66372347)
(3249 of 0.527506 1 13 OF 0.527506 ROAST 0.77329755)
(0 cleans 0 1 1536 nil 0 AM 0.80486846)
(4104 for 0.95127654 1 1 OR 0.95127654 OR 0.95127654)
(4275 and 0.5518295 11 44 NUT 0.870758 TIM 0.88111496)
(855 are 0.794573 2 5 ARM 0.879726 RAY 0.982962)
(4104 spend 0.96951437 1 4 SPEND 0.96951437 KNICK- 0.993262)
(2907 in 0.85449004 2 6 KNICK- 0.8615955 MY 0.997667)
(3420 miami 0.96991366 3 9 MY 0.994174 MY 0.994174)
(1197 is 0.1403235 4 178 EAR 0.4725025 RARE 0.618141)
(3078 be 0.553444 1 18 BE 0.553444 IN 0.86775804)
(2736 need 0.8220406 2 6 KNEE 0.8880335 KNEE 0.8880335)
(0 for 0 1 535 nil 0 ARE 0.798531)
(6840 and 0.09191799 10 429 AT 0.1302845 PA 0.702636)
(2565 in 0.553236 1 5 IN 0.553236 IS 0.954143)
(4959 big 0.95791996 1 1 BIG 0.95791996 BIG 0.95791996)
(3249 can 0.85563505 1 2 CAN 0.85563505 BE 0.882378)
(3249 be 0.468438 1 39 BE 0.468438 TEN 0.8422125)
(2394 an 0.6909615 1 8 AN 0.6909615 BE 0.871572)
(0 sport 0 1 1509 nil 0 PA 0.99963796)
(4617 did 0.035262667 7 391 TEA 0.710287 HE 0.927991)
(3933 buy 0.861672 1 2 BUY 0.861672 TEA 0.8809395)
(4104 any 0.44171765 4 108 EAT 0.637507 BITES 0.904883)
(4617 bright 0.3132487 38 309 RATE 0.621942 PAY 0.9675015)
(2907 red 0.48889533 26 114 RAN 0.8559303 ATE 0.984944)
(4104 his 0.013795 9 382 KIN 0.4816775 TEN 0.753637)
(0 for 0 1 696 nil 0 LACK 0.7112865)
(7524 is 0.47378 7 82 ILL 0.4791505 CEASE 0.9783765)
(4788 am 0.932438 2 2 MAD 0.952546 MAD 0.952546)
(4959 and 0.19536799 3 130 DOLL 0.633171 OF 0.665208)
(1710 are 0.2249815 2 58 ODD 0.488511 OF 0.974792)
(0 retracted 0 1 2839 nil 0 ACT 0.999018)
(3762 an 0.038595498 7 274 AM 0.4883115 IN 0.72176754)
(3249 michael 0.767261 4 8 LACK 0.8324325 HELD 0.87600994)
(3249 of 0.136959 1 149 OF 0.136959 ARC 0.766384)
(3249 is 0.16013251 6 257 IN 0.48552552 NEAT 0.952827)
(0 big 0 1 827 nil 0 I~D 0.812281)
(3591 in 0.3982255 7 62 ILL 0.63869596 CAR 0.926167)
(2907 customer 0.63639677 11 60 CAR 0.82696337 ELSE 0.992997)
(4617 spilled 0.8980517 1 7 SPILLED 0.8980517 AT 0.98903954)
(2565 some 0.549559 7 62 SEX 0.717819 LACK 0.8690795)
(3762 please 0.19748099 13 294 WRIST 0.5066415 KEY 0.957343)
(855 dig 0.2947655 24 166 AIN~T 0.6577785 LIE 0.813268)

(2052 my 0.363324 27 328 MARK 0.84687304 ART 0.9514485)

(2907 ride 0.57271534 7 51 RAN 0.78348535 ALL 0.976126)

(3591 in 0.190855 8 186 IT 0.4774005 AN 0.791743)

(3420 all 0.757403 1 3 ALL 0.757403 CALL 0.8220907)

(4275 are 0.134143 12 388 ARC 0.56705046 QUICK 0.6629677)

(4617 expected 0.6278721 10 31 EXPECT 0.7600608 PECK 0.9012055)

(2907 comes 0.48489702 1 84 COMES 0.48489702 OF 0.911512)

(2736 for 0.3745065 24 143 SCAR 0.684433 MAY 0.857799)

(0 his 0 1 609 nil 0 HIM 0.957626)

(4617 broken 0.32881474 50 363 ROCK 0.83747166 ART 0.936368)

(2907 call 0.9886445 1 1 CALL 0.9886445 CALL 0.9886445)

(513 an 0.55015695 4 15 LACK 0.8940275 ALL 0.966589)

(3078 for 0.50967395 1 38 OR 0.50967395 OF 0.882862)

(4617 can 0.878659 4 4 AT 0.913415 AT 0.913415)

(4959 buy 0.998515 1 1 BUY 0.998515 BUY 0.998515)

(2565 mine 0.63868666 6 36 MIGHT 0.95347065 MY 0.991084)

(1881 planned 0.30762476 8 474 PLANT 0.469722 LIE 0.981826)

(3078 is 0.518966 2 20 EASE 0.5199305 ALL 0.999285)

(4446 every 0.657411 1 4 EVERY 0.657411 ERA 0.828153)

(513 related 0.4598612 3 218 SHRIEKED 0.63513535 MY 0.99985)

(4104 my 0.999866 1 1 MY 0.999866 MY 0.999866)

(2907 ability 0.5880596 1 81 ABILITY 0.5880596 PICK 0.9721955)

(4275 is 0.932337 1 1 IS 0.932337 IS 0.932337)

(0 expense 0 1 1730 nil 0 BACKED 0.554895)

(5130 his 0.060806002 16 640 IN 0.18996951 OF 0.742972)

(3078 he 0.6044855 1 9 HE 0.6044855 IT 0.941069)

(3591 street 0.64767903 2 23 STAY 0.69518054 LAST 0.97051346)

(4788 needed 0.33841646 13 263 NEED 0.50640696 LIE 0.904281)

(4104 be 0.604895 2 4 DID 0.64988554 OF 0.74373)

(0 elegant 0 1 1533 nil 0 ILL 0.98510647)

(5985 or 0.296297 1 123 OR 0.296297 SEA 0.951602)

(4446 from 0.6363135 1 10 FROM 0.6363135 NUMB 0.9409675)

(4959 answer 0.66709536 4 16 ANTS 0.78353834 EAT 0.95737696)

(0 did 0 1 426 nil 0 BUY 0.991301)

(4104 buy 0.992611 2 2 BUYING 0.994764 BUYING 0.994764)

(4446 any 0.431944 2 68 BE 0.4329035 KNEE 0.80265796)

(0 bright 0 1 1584 nil 0 RAY 0.9772105)

(0 red 0 1 669 nil 0 RAY 0.99966)

(1710 his 0.583349 1 19 HIS 0.583349 IS 0.956598)

(0 for 0 1 563 nil 0 IN 0.9903475)

(2052 is 0.7320885 7 8 ILL 0.885158 ILL 0.885158)

(3933 am 0.736646 11 18 AN 0.961191 NUMB 0.9784835)

(3762 and 0.15436698 13 245 MAIL 0.508257 ILL 0.96893)

(6156 are 0.2802425 1 220 ARE 0.2802425 TEA 0.797853)

(2565 forces 0.4873885 6 157 OFFER 0.50261 SAYS 0.99343395)

(0 from 0 1 704 nil 0 COME 0.648937)
(1368 lily 0.376172 5 111 LID 0.444748 LEAD 0.81766903)
(5985 are 0.713514 1 6 ARE 0.713514 CAT 0.7549135)
(3591 of 0.289176 1 97 OF 0.289176 ROAM 0.8430285)
(1197 come 0.726033 4 4 CAT 0.9088 CAT 0.9088)
(2565 makes 0.6123265 4 8 AM 0.856255 AM 0.856255)
(3933 forces 0.5858515 13 125 OUGHT 0.9883095 OUGHT 0.9883095)
(4959 from 0.20107101 10 336 IS 0.543993 IS 0.82086253)
(342 lily 0.48904565 3 35 KNEE 0.531826 LEAD 0.8587295)
(4788 are 0.1889905 1 171 ARE 0.1889905 OF 0.676669)
(3762 of 0.099152 16 442 VERY 0.3739163 BUY 0.891735)
(2736 come 0.7432185 1 2 COME 0.7432185 DID 0.8234095)
(2565 makes 0.66373205 5 21 AM 0.965889 RACK 0.98733056)
(6498 and 0.047423664 33 1007 DAY 0.524421 PAIN 0.955674)
(1026 sprained 0.84680504 4 4 SPRAY 0.890635 SPRAY 0.890635)
(4617 steep 0.74295264 2 12 TEA 0.8021105 EATING 0.927511)
(6840 can 0.2297025 9 152 CALL 0.62123 RAW 0.965253)
(2907 be 0.4173385 2 45 TEASED 0.50234354 RUN 0.8956425)
(3078 diseases 0.54496515 5 54 TEASED 0.88941234 EASED 0.9972015)
(5130 are 0.154454 6 354 RUN 0.29224348 TORE 0.574553)
(1539 in 0.5547795 1 7 IN 0.5547795 AM 0.872377)
(0 for 0 1 518 nil 0 TRUCK 0.78511834)
(2394 all 0.672967 3 9 CALLED 0.73144895 TRUCK 0.8675534)
(3249 he 0.86329997 1 1 HE 0.86329997 HE 0.86329997)
(7524 lie 0.324414 13 340 NEED 0.9220735 NEED 0.9220735)
(5130 all 0.98153 1 1 ALL 0.98153 ALL 0.98153)
(0 once 0 1 1690 nil 0 LOTS 0.79945636)
(2907 cory 0.9338161 3 8 CAR 0.99947953 CAR 0.99947953)
(1881 and 0.24124269 18 301 CAN~T 0.73680097 EAT 0.8592965)
(2565 played 0.19291966 19 535 PARK 0.582351 SET 0.655815)
(3933 for 0.2529605 1 106 OR 0.2529605 DOLL 0.962564)
(2907 right 0.50427634 30 116 RAY 0.987169 THE 0.991819)
(2736 may 0.4712895 8 54 KNICK- 0.623938 ERA 0.84747005)
(0 be 0 1 463 nil 0 ELM 0.980568)
(0 for 0 1 563 nil 0 ARM 0.9719835)
(2907 business 0.41285002 5 115 BE 0.7109755 TEA 0.76680195)
(0 retracted 0 1 2477 nil 0 RACK 0.8348255)
(3249 an 0.6742895 1 2 AN 0.6742895 THE 0.9671825)
(3762 michael 0.952039 1 2 MICHAEL 0.952039 HE 0.9617035)
(3078 of 0.507769 1 8 OF 0.507769 THE 0.605966)
(3078 is 0.53643 1 7 EARS 0.53643 OF 0.7659205)
(4275 big 0.885615 3 4 EAR 0.936961 ALL 0.988824)
(171 in 0.356783 1 12 IN 0.356783 SHE 0.7273195)
(0 combine 0 1 2335 nil 0 THE 0.90789354)
(7011 all 0.998606 1 1 ALL 0.998606 ALL 0.998606)

(2394 in 0.176769 4 97 ILL 0.275739 PILL 0.727497)
(4788 by 0.465516 2 20 IN 0.4910425 AN 0.71963596)
(2223 and 0.011641 38 478 HOARSE 0.3187473 EDGES 0.4267713)
(4275 stray 0.511911 16 41 TRAIN 0.77393436 TRAIN 0.77393436)
(4275 for 0.0959965 4 301 OFFER 0.335296 SON 0.5828135)
(4959 one 0.52649736 1 19 DONE 0.52649736 SON 0.771438)
(3420 of 0.365346 1 47 OF 0.365346 TEA 0.7059855)
(4959 buy 0.968668 1 1 BUY 0.968668 BUY 0.968668)
(3762 be 0.887181 1 1 BE 0.887181 BE 0.887181)
(3933 right 0.43724 17 147 RATE 0.77861863 HATE 0.989503)
(9234 hand 0.30422333 9 240 AN 0.600215 ATE 0.99579)
(2052 are 0.06934 9 260 LOCK 0.546589 IS 0.934481)
(0 for 0 1 712 nil 0 IN 0.68795455)
(5130 quick 0.44917202 5 97 WICK 0.54375 ODD 0.8171425)
(2565 in 0.446807 2 17 AN 0.45878 THE 0.664602)
(513 in 0.4423485 2 50 RIM 0.472088 AM 0.985919)
(0 consists 0 1 1037 nil 0 TRISH 0.904453)
(3591 of 0.181971 2 146 PAW 0.185603 SCORE 0.5949255)
(855 and 0.07346133 10 365 DREAM 0.409912 STEAM 0.6959022)
(6156 are 0.464872 1 26 ARE 0.464872 LAY 0.799168)
(0 for 0 1 673 nil 0 TRY 0.6944185)
(4788 all 0.754036 1 4 ALL 0.754036 CRY 0.91946)
(855 he 0.7957715 1 3 HE 0.7957715 WE 0.955418)
(4104 lie 0.994933 1 1 LIE 0.994933 LIE 0.994933)
(4959 all 0.873187 1 2 ALL 0.873187 AL 0.910257)
(3078 once 0.256932 12 352 WALLS 0.34868467 LIQUIDS 0.75837594)
(2736 cory 0.98354274 1 2 CORY 0.98354274 OR 0.989743)
(4617 and 0.060024668 1 255 AND 0.060024668 OR 0.987819)
(2736 played 0.61571497 14 43 PLATE 0.9425706 PLATE 0.9425706)
(1710 for 0.081073 3 390 SHE 0.562178 ALL 0.943817)
(2394 right 0.34028998 47 187 RAIN 0.788333 EAR 0.906577)
(2736 may 0.47042698 16 69 NEAR 0.72723365 IN 0.94518)
(1710 be 0.695474 1 1 BE 0.695474 BE 0.695474)
(0 for 0 1 742 nil 0 EFFORT 0.51267797)
(4617 business 0.53689677 8 29 IT 0.971018 IT 0.971018)
(4788 intelligence 0.75730616 1 13 INTELLIGENCE 0.75730616 TELL 0.98859894)
(3078 is 0.54569 11 38 IF 0.9826335 IF 0.9826335)
(3249 for 0.34002 5 137 KEY 0.49852 IF 0.916954)
(0 become 0 1 797 nil 0 IT 0.985831)
(0 artists 0 1 1746 nil 0 ARE 0.995518)
(5130 fish 0.9854655 1 1 FISH 0.9854655 FISH 0.9854655)
(3933 leap 0.906996 1 2 LEAP 0.906996 SLEEP 0.91348)
(5301 frantically 0.70201045 4 40 RAN 0.90828234 LEAP 0.9646855)
(3078 of 0.8867855 1 2 OF 0.8867855 THE 0.96582496)
(2907 he 0.7667085 1 1 HE 0.7667085 HE 0.7667085)

(2736 from 0.429165 3 28 THE 0.46950498 THE 0.80985796)

(0 every 0 1 1235 nil 0 ME 0.74651897)

(2565 in 0.7665295 1 1 IN 0.7665295 IN 0.7665295)

(3249 of 0.130146 4 357 DUMPED 0.25581574 AT 0.9835425)

(6156 cleans 0.32329035 2 265 CLEANED 0.332871 WE 0.932972)

(3933 for 0.43159348 7 70 ALL 0.653118 I~LL 0.8316715)

(4959 are 0.951604 1 1 ARE 0.951604 ARE 0.951604)

(3762 all 0.47815698 1 34 ALL 0.47815698 BUY 0.982248)

(5301 errors 0.516481 16 63 RAN 0.8071805 DANCE 0.844431)

(2736 nearest 0.72135496 4 16 NEAR 0.7779653 IN 0.953043)

(4788 may 0.979853 1 1 MAY 0.979853 MAY 0.979853)

(2907 be 0.706193 1 7 BE 0.706193 ROB 0.7864735)

(3420 distance 0.43981838 11 150 EAR 0.500403 NEAT 0.9200455)

(2907 buys 0.96787345 1 1 EYES 0.96787345 EYES 0.96787345)

(0 catastrophic 0 1 2498 nil 0 ROCK 0.975049)

(3762 expense 0.3953542 8 235 EXQUISITE 0.525064 IS 0.8629875)

(5301 his 0.31044832 6 253 IS 0.625806 KNEE 0.929264)

(0 he 0 1 650 nil 0 KNIT 0.7341735)

(4104 street 0.536439 7 77 TOWARD 0.64848 SLEIGH 0.81821)

(0 needed 0 1 682 nil 0 ARE 0.919893)

(3762 be 0.8684 1 2 BE 0.8684 COME 0.9096455)

(0 elegant 0 1 1302 nil 0 ILL 0.943716)

(6840 or 0.083723 1 286 OR 0.083723 LINE 0.658156)

(4446 from 0.7215745 6 22 SON 0.97371495 SON 0.97371495)

(2394 answer 0.58966035 7 15 CASH 0.90027654 EASE 0.96793103)

(1026 is 0.508889 1 4 IS 0.508889 WAS 0.6000145)

(0 for 0 1 631 nil 0 SEA 0.953652)

(4104 ended 0.096683 10 481 DIM 0.410102 IN 0.9259525)

(1368 please 0.7606743 5 15 PLAY 0.93818855 KEY 0.97387695)

(1539 cleaners 0.617023 2 24 CLEANED 0.64767736 IF 0.997653)

(4104 for 0.31277534 2 110 RAIN 0.33409402 KEY 0.730406)

(684 cleaners 0.35479873 11 278 THEY 0.540189 SAY 0.875486)

(342 and 0.21525735 24 229 PA 0.5436815 KNIT 0.6457385)

(342 are 0.159551 2 122 LOT 0.3031045 AT 0.570912)

(4446 spend 0.58536196 1 7 SPEND 0.58536196 MY 0.977887)

(684 in 0.3691385 3 22 ARM 0.5072465 MY 0.991722)

(3933 miami 0.8216823 6 11 I~D 0.938338 AM 0.995042)

(2736 is 0.094373 11 279 IF 0.5052105 EAR 0.7667885)

(3249 be 0.243758 2 110 ILL 0.357422 AND 0.72485495)

(4104 need 0.433662 7 100 EAT 0.760976 ALL 0.850363)

(5643 for 0.1402585 9 249 BILL 0.7864275 ILL 0.979928)

(0 and 0 1 989 nil 0 PA 0.8785385)

(3591 in 0.093512 2 119 IT 0.1096295 STAB 0.6522033)

(4446 big 0.9960965 1 1 BIG 0.9960965 BIG 0.9960965)

(342 can 0.34459502 3 94 CAB 0.5090985 BE 0.991152)

(4275 be 0.983737 1 1 BE 0.983737 BE 0.983737)

(1881 an 0.38184598 4 20 AIR 0.404366 BE 0.950224)

(4275 sport 0.741544 2 4 PAW 0.930216 PAW 0.930216)

(855 and 0.23219167 34 564 ACTS 0.64047736 BUY 0.978359)

(0 sprained 0 1 850 nil 0 KNEE 0.94786847)

(4788 steep 0.64166135 1 12 STEEP 0.64166135 TEA 0.989935)

(2736 can 0.60271305 3 42 CALL 0.837999 ART 0.948496)

(3762 be 0.983603 1 1 BE 0.983603 BE 0.983603)

(2907 diseases 0.6263478 1 41 DISEASES 0.6263478 TEA 0.99168146)

(5814 are 0.198397 1 249 ARE 0.198397 IS 0.890758)

(1368 in 0.881668 5 12 IT 0.90067303 SEA 0.997706)

(4788 is 0.387555 1 6 IS 0.387555 KIN 0.5595895)

(0 for 0 1 375 nil 0 SEAM 0.531945)

(1881 ended 0.46478274 2 20 DAMP 0.47975636 ROAM 0.688957)

(4104 please 0.23349565 12 378 LEAST 0.794778 KICK 0.95276654)

(4788 cleaners 0.979002 1 1 CLEANERS 0.979002 CLEANERS 0.979002)

(4275 for 0.7545165 3 5 OUGHT 0.86524546 OUGHT 0.86524546)

(5130 cleaners 0.9586566 1 3 CLEANERS 0.9586566 KNEES 0.9913635)

(0 for 0 1 684 nil 0 SCORE 0.7727067)

(2907 all 0.94479597 2 4 WALL 0.9570955 CALL 0.962341)

(2736 he 0.48316 3 17 HIM 0.858088 HIM 0.858088)

(2736 lie 0.988847 1 1 LIE 0.988847 LIE 0.988847)

(4446 all 0.830073 1 4 ALL 0.830073 WE 0.877755)

(0 once 0 1 1081 nil 0 ARE 0.977118)

(2907 cory 0.6680763 10 57 KEY 0.980706 KEY 0.980706)

(4104 and 0.32410467 1 206 AND 0.32410467 AIN~T 0.9538005)

(2736 played 0.74045366 6 34 PLAY 0.9614675 PLAY 0.9614675)

(0 for 0 1 638 nil 0 PA 0.907265)

(2565 right 0.26723933 35 316 RAN 0.586568 MAY 0.972328)

(2565 may 0.5661645 9 34 MAIN 0.70082766 AN 0.92523646)

(2736 be 0.571 5 8 IN 0.7245845 IN 0.7245845)

(513 for 0.028266998 8 396 RIM 0.35361966 MAP 0.5960065)

(0 business 0 1 718 nil 0 IT~S 0.78192)

(4617 cat 0.543817 2 83 CAN~T 0.6814 TEA 0.9534945)

(3078 my 0.030406 10 637 MIGHTY 0.34338033 TEA 0.91964996)

(4104 by 0.081496 7 352 ILL 0.45987698 ILL 0.860828)

(2736 of 0.007097 2 499 UNTIL 0.039967 AN 0.513049)

(3420 may 0.99853 1 1 MAY 0.99853 MAY 0.99853)

(2223 money 0.40653732 1 63 MONEY 0.40653732 END 0.7150555)

(3933 by 0.018854 15 757 I~VE 0.376524 OR 0.969039)

(0 hard 0 1 1086 nil 0 AND 0.73307645)

(4275 candy 0.83523226 7 9 AN 0.9264285 KNEE 0.9562905)

(6840 are 0.486985 1 42 ARE 0.486985 EASE 0.986933)

(2223 or 0.10115901 10 221 ILL 0.200308 PAY 0.5102995)

(0 proceeding 0 1 1693 nil 0 SEA 0.991257)

(4275 grows 0.83805865 2 2 GROW 0.97479796 GROW 0.97479796)
(0 for 0 1 519 nil 0 CALL 0.69420666)
(5814 all 0.957863 1 2 ALL 0.957863 CALL 0.9763005)
(2565 he 0.95453846 1 1 HE 0.95453846 HE 0.95453846)
(3420 lie 0.9928765 1 2 LIE 0.9928765 ERA 0.9950775)
(6327 all 0.1294955 4 119 BUY 0.555606 LOCK 0.6948565)
(3249 once 0.5076793 5 106 LIKE 0.5930154 ICE 0.9866255)
(2907 cory 0.92021227 3 13 CAR 0.983148 CAR 0.983148)
(1539 and 0.607169 10 10 AN 0.9118355 AN 0.9118355)
(3591 played 0.52258533 6 69 PLANE 0.785145 LAKE 0.981653)
(0 for 0 1 457 nil 0 ALL 0.836136)
(2907 right 0.387939 22 142 RARE 0.8179307 MAY 0.977611)
(3420 may 0.999401 1 1 MAY 0.999401 MAY 0.999401)
(2907 be 0.945499 1 1 BE 0.945499 BE 0.945499)
(0 for 0 1 321 nil 0 MAY 0.960168)
(4275 business 0.3376125 19 302 EAR 0.81403 MAY 0.876378)
(3933 made 0.812132 2 4 MAY 0.982031 MAY 0.982031)
(3591 of 0.518254 1 18 OF 0.518254 MAY 0.991932)
(3420 are 0.371087 4 45 ARM 0.545964 MAY 0.720841)
(2394 in 0.9179735 1 2 IN 0.9179735 THE 0.93517303)
(0 stockings 0 1 2010 nil 0 PA 0.99738646)
(6327 run 0.30176634 16 298 HEN 0.5564113 CRACK 0.9660407)
(1881 dishes 0.7509491 5 11 IS 0.7929945 ARE 0.968663)
(2736 makes 0.74032825 5 27 MAKE 0.980865 FIX 0.998897)
(5301 and 0.086665 6 495 TEA 0.5187015 TEA 0.982317)
(2907 call 0.9748075 1 1 CALL 0.9748075 CALL 0.9748075)
(7011 an 0.36097398 1 24 AN 0.36097398 ALL 0.883982)
(0 for 0 1 839 nil 0 IF 0.99611247)
(5130 can 0.8594225 1 1 AN 0.8594225 AN 0.8594225)
(3933 buy 0.027052 18 624 MY 0.990647 MY 0.990647)
(3933 mine 0.2143765 11 345 MY 0.422058 EASE 0.998759)
(3762 planned 0.6314247 4 37 LAST 0.7199361 CLASS 0.93019533)
(1710 is 0.4100755 3 20 LAWS 0.5038235 OF 0.803567)
(2907 did 0.35356867 14 114 DIM 0.6404503 THE 0.7269925)
(3933 buy 0.500594 2 9 BUYER 0.58126104 BE 0.983645)
(4104 any 0.5482805 5 19 KNEE 0.815567 EAT 0.8756125)
(4446 bright 0.5343033 19 55 RAY 0.9957335 RAY 0.9957335)
(2907 red 0.20499833 27 488 RACK 0.741843 RACK 0.9660325)
(4275 his 0.009072334 4 466 ATTACK 0.13780867 TEA 0.91177046)
(0 for 0 1 633 nil 0 ATE 0.813577)
(2394 is 0.4924755 13 76 IT 0.74527097 SEA 0.995898)
(2394 am 0.170798 13 130 AN 0.6440675 AN 0.6440675)
(4104 and 0.12565534 6 135 LET 0.3225835 IS 0.912251)
(3420 are 0.008394 9 376 ART 0.4833155 RIGHT 0.6433345)
(4104 is 0.621084 1 4 IS 0.621084 HIS 0.79905045)

(0 for 0 1 697 nil 0 KEY 0.9709805)

(6327 ended 0.48233074 1 60 ENDED 0.48233074 WICK 0.91735256)

(4275 please 0.135144 16 440 LEAST 0.64782196 KEY 0.905193)

(4959 cleaners 0.81011134 4 15 LEAF 0.8775694 TEAM 0.88858104)

(4617 for 0.9776345 1 2 OR 0.9776345 OR 0.978098)

(1710 cleaners 0.5556795 6 17 KEY 0.972677 KEY 0.972677)

(0 combine 0 1 1734 nil 0 BUY 0.997758)

(4104 all 0.676671 1 2 ALL 0.676671 KNEE 0.8767795)

(2736 in 0.258644 8 112 TELL 0.4616895 IS 0.726613)

(2907 by 0.99858 1 1 BUY 0.99858 BUY 0.99858)

(2052 and 0.15501633 28 539 DISK 0.55520004 SIT 0.8692895)

(2736 stray 0.757498 10 41 STATE 0.98709327 STATE 0.98709327)

(4788 for 0.333365 5 55 ALL 0.695916 ARE 0.791452)

(5301 one 0.3380413 8 162 KNIT 0.632366 IT 0.9395145)

(1368 of 0.4771635 4 17 LOCK 0.65582347 LOCK 0.65582347)

(2736 buy 0.986681 1 1 BUY 0.986681 BUY 0.986681)

(3762 be 0.999554 1 1 BE 0.999554 BE 0.999554)

(3933 right 0.55758363 18 68 RAN 0.9056115 AN 0.967952)

(5301 hand 0.5447083 11 163 ANT 0.888577 RIGHT 0.97379446)

(3591 forces 0.28473848 33 550 RUN 0.57612866 DRESS 0.93540764)

(4275 from 0.52706397 3 9 WROTE 0.59593964 PINK 0.636899)

(2394 lily 0.8609307 2 4 ALL 0.910868 ARE 0.915687)

(171 are 0.635702 1 1 ARE 0.635702 ARE 0.635702)

(2907 of 0.46953952 2 4 WAS 0.599787 STAR 0.607177)

(342 come 0.44964153 32 77 ANT 0.7045635 CALM 0.9035285)

(2736 makes 0.6417075 13 77 MATE 0.8703527 ATE 0.9665415)

(0 forces 0 1 1175 nil 0 OR 0.956887)

(0 from 0 1 779 nil 0 SIT 0.86364055)

(2394 lily 0.59844637 4 12 ALL 0.889877 LEAD 0.8959745)

(4959 are 0.776697 1 2 ARE 0.776697 CAR 0.8382215)

(3420 of 0.423103 1 15 OF 0.423103 ARE 0.822638)

(684 come 0.687477 3 6 CAT 0.691655 ARE 0.78350854)

(3933 makes 0.600615 1 41 MAKES 0.600615 KEY 0.810822)

(3591 intelligence 0.5893994 4 62 TELL 0.74492365 TELL 0.99582946)

(3249 is 0.2164775 11 259 IF 0.5697995 ELSE 0.68548954)

(4446 for 0.6213245 1 27 OR 0.6213245 KEY 0.94128895)

(0 become 0 1 1189 nil 0 ILL 0.990353)

(0 artists 0 1 2394 nil 0 ARE 0.999325)

(5130 fish 0.5410315 5 28 IN 0.85187304 THIS 0.914978)

(2907 leap 0.683645 4 16 LAUGH 0.902829 LAUGH 0.91535604)

(0 frantically 0 1 3172 nil 0 LAY 0.99783)

(6840 of 0.153429 1 137 OF 0.153429 THE 0.91197395)

(0 combine 0 1 1084 nil 0 PA 0.9498085)

(5130 all 0.676796 1 10 ALL 0.676796 ARC 0.7818275)

(3420 in 0.045256503 9 256 AM 0.383708 AM 0.8640535)

(3078 by 0.582103 3 27 I~M 0.7812395 ELM 0.91207004)

(3762 and 0.0899645 3 227 WAS 0.46108952 IS 0.519884)

(4959 stray 0.4495805 6 159 TRAIN 0.61635566 RAIN 0.9146035)

(5472 for 0.48567548 2 33 LAW 0.61257696 ARE 0.949331)

(0 one 0 1 719 nil 0 OR 0.894329)

(1197 of 0.6281405 1 3 OF 0.6281405 EYES 0.6535665)

(3249 buy 0.043296 11 798 ALARM 0.44281274 ARE 0.999757)

(3420 be 0.006503 15 807 KNEE 0.497184 ILL 0.995594)

(3591 right 0.552795 1 6 RIGHT 0.552795 RAY 0.77944)

(0 hand 0 1 1213 nil 0 EYES 0.739716)

(3933 every 0.878931 1 1 EVERY 0.878931 EVERY 0.878931)

(1881 related 0.2250042 35 790 CLEAN 0.7147794 ICE 0.9258395)

(3591 my 0.950697 3 3 MICHAEL 0.9564165 MICHAEL 0.9564165)

(3078 ability 0.686094 4 42 OF 0.845482 ILL 0.971603)

(4617 is 0.224229 1 148 IS 0.224229 EASE 0.972531)

(2736 call 0.9627123 1 1 CALL 0.9627123 CALL 0.9627123)

(4788 an 0.368128 1 36 AN 0.368128 ON 0.74064803)

(0 for 0 1 599 nil 0 MAY 0.970088)

(6156 can 0.688847 1 21 AN 0.688847 ARE 0.985821)

(3078 buy 0.0122535005 19 717 DARK 0.521023 I~LL 0.905662)

(4275 mine 0.954407 1 2 MINE 0.954407 EASE 0.984116)

(3591 planned 0.52060133 3 26 PLANES 0.59975535 HE 0.971161)

(5643 is 0.4661285 1 36 IS 0.4661285 OR 0.966919)

(6840 all 0.574464 1 47 ALL 0.574464 ARC 0.902798)

(3762 are 0.265799 3 111 ARC 0.595085 KNICK- 0.6949625)

(4104 expected 0.66305506 17 41 SON 0.9906335 SON 0.9906335)

(0 comes 0 1 1040 nil 0 SKILLED 0.7625985)

(4275 for 0.45202053 1 26 OR 0.45202053 IF 0.681062)

(0 his 0 1 332 nil 0 THE 0.838398)

(0 broken 0 1 1539 nil 0 TELL 0.918154)

(4959 cat 0.77959204 3 14 AN 0.905579 WICK 0.94805396)

(3420 my 0.905172 2 5 MIKE 0.9492355 LIKE 0.977201)

(3933 by 0.916555 1 1 BUY 0.916555 BUY 0.916555)

(2736 of 0.1464065 1 134 OF 0.1464065 ERA 0.86905)

(3420 may 0.938518 1 1 MAY 0.938518 MAY 0.938518)

(2736 money 0.94886535 1 4 MONEY 0.94886535 BUY 0.980169)

(4446 by 0.984158 1 1 BUY 0.984158 BUY 0.984158)

(0 hard 0 1 1323 nil 0 ARE 0.982227)

(855 candy 0.627506 1 10 CANDY 0.627506 CAST 0.8054465)

(4959 all 0.935452 1 1 ALL 0.935452 ALL 0.935452)

(3762 are 0.47440952 8 242 LOCK 0.93543696 ILL 0.988291)

(4104 expected 0.5802528 11 98 SECTOR 0.859001 STREETS 0.95590496)

(2736 comes 0.33940268 9 132 DUMB 0.7059555 SOME 0.730124)

(4275 for 0.183483 14 231 FROM 0.58310133 MAY 0.958797)

(0 his 0 1 368 nil 0 THIS 0.849543)

(4275 broken 0.7551587 8 29 ROCK 0.9736133 ROCK 0.9736133)

(4275 and 0.23999232 9 135 TEA 0.4809325 KNIT 0.865391)

(2565 are 0.21095149 2 43 DONE 0.47918102 SON 0.62240255)

(3933 spend 0.606105 4 12 SPENT 0.680036 TEA 0.7287055)

(342 in 0.494933 1 25 IN 0.494933 TEA 0.748973)

(3762 miami 0.60861135 3 8 MY 0.856976 KNEE 0.872054)

(855 is 0.488251 2 18 IF 0.4987405 OFFER 0.924778)

(3078 be 0.538946 1 5 BE 0.538946 ILL 0.757727)

(2394 need 0.44443896 13 39 KNEE 0.6850685 IN 0.7775815)

(2052 for 0.14654833 1 148 DOOR 0.14654833 KNEE 0.929848)

(684 and 0.25001967 4 252 DRAIN 0.32404667 TRAY 0.906953)

(3591 in 0.572753 1 13 IN 0.572753 SON 0.643328)

(4104 big 0.507886 3 37 EAR 0.5222395 TAN 0.97978055)

(5130 can 0.22817801 8 312 KNEE 0.5252335 TEA 0.9895445)

(3591 be 0.285808 2 56 BEING 0.42581734 TEA 0.991212)

(2907 an 0.3107515 1 50 AN 0.3107515 EASE 0.693688)

(4617 sport 0.317252 4 148 PAM 0.375765 LINE 0.672041)

(4446 and 0.011882334 21 1124 SON 0.430753 NONE 0.8747025)

(1539 sprained 0.84626395 1 4 SPRAINED 0.84626395 RAY 0.959778)

(4617 steep 0.35940468 13 117 TEA 0.69522953 THE 0.7606315)

(4446 can 0.59986 4 19 AL 0.86114204 AL 0.96231604)

(3420 be 0.999426 1 1 BE 0.999426 BE 0.999426)

(3420 diseases 0.6209614 1 53 DISEASES 0.6209614 EASE 0.977815)

(2052 are 0.4026315 2 51 SPHERE 0.6072603 IN 0.926581)

(2736 in 0.5474965 1 13 IN 0.5474965 SEA 0.999077)

(4446 expense 0.51346856 19 137 SIN 0.8717145 PINS 0.893044)

(4788 his 0.319396 4 103 IS 0.865545 ME 0.903553)

(0 he 0 1 631 nil 0 TEA 0.65759003)

(3762 street 0.74414533 5 18 TIM 0.963028 LOCK 0.979748)

(4959 needed 0.6532647 2 3 NEED 0.8411895 NEED 0.8411895)

(3762 be 0.885067 1 3 BE 0.885067 TEA 0.965696)

(0 elegant 0 1 1052 nil 0 TELL 0.69094145)

(3249 or 0.005577 9 681 ROCKY 0.07812 AM 0.962452)

(0 from 0 1 929 nil 0 COME 0.795348)

(5472 answer 0.52433866 4 8 ENTER 0.61998665 IN 0.76850295)

(5643 expense 0.53576285 17 129 CENT 0.724042 STREETS 0.9743345)

(5643 his 0.20793499 1 452 HIS 0.20793499 SICK 0.9836275)

(0 he 0 1 301 nil 0 EAT 0.9774895)

(4104 street 0.35360527 42 226 TRUCK 0.6964248 ROCK 0.7983915)

(3420 needed 0.09695434 7 510 DAMP 0.33837032 LIE 0.880581)

(4275 be 0.55266 1 23 BE 0.55266 HATE 0.8016355)

(0 elegant 0 1 1276 nil 0 BIKE 0.8145315)

(0 or 0 1 778 nil 0 RAW 0.55914503)

(3933 from 0.71010053 1 3 FROM 0.71010053 NUMB 0.835498)

(3249 answer 0.6668777 1 16 ANSWER 0.6668777 SEA 0.908997)

(0 customer 0 1 1636 nil 0 US 0.882981)
(4446 spilled 0.728777 3 25 PILLS 0.9781647 PILLS 0.9781647)
(1881 some 0.128445 8 320 ACHES 0.5121035 ILL 0.998098)
(4617 please 0.30018133 12 310 LEAST 0.8270085 LEAST 0.8270085)
(3420 dig 0.133525 12 369 DOWN 0.20146699 STICK 0.72370434)
(0 my 0 1 653 nil 0 TIN 0.8300905)
(2736 ride 0.680566 6 13 RUN 0.8325457 RUN 0.8325457)
(1197 in 0.477701 4 5 EAR 0.49816298 EAR 0.49816298)
(5643 cat 0.7185525 4 34 AIR 0.997844 AIR 0.997844)
(3249 my 0.986694 1 1 MY 0.986694 MY 0.986694)
(3078 by 0.114378 14 534 ARC 0.47910097 AN 0.8361265)
(5130 of 0.017055 3 1113 VAIN 0.232869 KNEE 0.989124)
(4959 may 0.6759225 1 1 MATE 0.6759225 MATE 0.6759225)
(2565 money 0.48166335 29 88 MY 0.9084255 BUY 0.976346)
(4446 by 0.889765 1 2 BUY 0.889765 DOWN 0.9772785)
(0 hard 0 1 2162 nil 0 ARE 0.814149)
(3933 candy 0.6633402 5 14 CAN 0.7901676 DARNED 0.81916165)
(4446 every 0.659197 1 1 EVERY 0.659197 EVERY 0.659197)
(1539 related 0.2635196 32 720 ROCKETS 0.6214252 LET 0.972278)
(4104 my 0.986151 1 1 MY 0.986151 MY 0.986151)
(3762 ability 0.37150103 21 215 BELT 0.72267103 IT 0.905287)
(4104 is 0.998903 1 1 IS 0.998903 IS 0.998903)
(0 of 0 1 554 nil 0 DISH 0.63980263)
(0 and 0 1 567 nil 0 EAT 0.9210725)
(3762 prevented 0.1136724 25 882 PRETTY 0.5491807 KNEE 0.721152)
(0 from 0 1 695 nil 0 AM 0.930971)
(0 contains 0 1 1909 nil 0 ATE 0.966529)
(4788 for 0.06732967 7 318 UP 0.13735 SPREAD 0.49306765)
(684 all 0.093182 34 218 ITSELF 0.304359 PA 0.4298955)
(3078 he 0.9281375 1 1 HE 0.9281375 HE 0.9281375)
(4617 lie 0.439317 3 46 I~D 0.6979505 ERRORS 0.92038)
(2394 all 0.328298 3 72 ILL 0.3343845 DAY 0.7741255)
(0 once 0 1 1470 nil 0 EYES 0.85753703)
(2907 cory 0.86084336 3 6 KEY 0.9065755 THE 0.9458435)
(2907 and 0.1449045 22 385 CALF 0.5343795 TEA 0.918062)
(4617 played 0.7413783 10 42 LAND 0.8472547 AN 0.983235)
(0 for 0 1 944 nil 0 SEND 0.5886887)
(4275 right 0.7067585 2 14 I~LL 0.8422365 CRY 0.954267)
(4446 may 0.993221 1 1 MAY 0.993221 MAY 0.993221)
(2907 be 0.069638 3 390 BEING 0.29579452 AM 0.938026)
(0 for 0 1 507 nil 0 MAY 0.955057)
(2736 business 0.69174397 1 16 BUSINESS 0.69174397 SEAM 0.894282)
(1881 are 0.26907802 5 46 HE 0.499338 RIDGE 0.598608)
(4275 for 0.96028304 1 2 OR 0.96028304 OR 0.984758)
(4617 big 0.81938696 1 3 BIG 0.81938696 TEA 0.89177704)

(1881 ambled 0.6312055 6 24 CAMP 0.896365 MAP 0.9428665)

(3591 increases 0.038939 19 1522 INCREASE 0.6343647 EASE 0.99570656)

(6840 and 0.532991 1 38 AN 0.532991 STRAINED 0.971254)

(1539 sprained 0.79859704 3 3 PLAN 0.80701596 PLAN 0.80701596)

(4446 steep 0.7146327 2 4 TEA 0.87518597 TEA 0.90740097)

(513 can 0.534908 5 28 CAN~T 0.6848127 ART 0.89345896)

(3420 be 0.988761 1 1 BE 0.988761 BE 0.988761)

(3249 diseases 0.48845783 1 113 DISEASES 0.48845783 EASE 0.935228)

(5985 are 0.447175 1 10 ARE 0.447175 OF 0.977253)

(855 in 0.266992 7 254 LESS 0.5209915 SEA 0.986476)

(1881 are 0.47406 5 25 HE 0.65294147 THE 0.86414003)

(4446 for 0.3037985 14 163 ON 0.499585 I~M 0.731047)

(4788 big 0.9806285 1 1 BIG 0.9806285 BIG 0.9806285)

(4617 ambled 0.6953997 4 9 AM 0.937151 AM 0.937151)

(0 increases 0 1 1888 nil 0 TRAY 0.9058485)

(1881 is 0.068122 9 255 WRIST 0.514683 TEA 0.73410904)

(0 controlled 0 1 1398 nil 0 ADD 0.925835)

(171 from 0.38913667 6 27 RUN 0.678268 THE 0.986308)

(2052 is 0.186455 14 199 IT 0.43510848 NEAT 0.78260696)

(3933 for 0.3328665 6 41 OFTEN 0.5137785 EAT 0.560658)

(4275 buyer 0.8374985 1 1 BUYER 0.8374985 BUYER 0.8374985)

(3762 money 0.259215 1 372 MONEY 0.259215 ARC 0.80851996)

(0 for 0 1 609 nil 0 GLEE 0.950649)

(0 combine 0 1 1392 nil 0 BUY 0.741635)

(1539 all 0.209123 14 95 LOCK 0.45454934 BUY 0.924078)

(3078 in 0.4273715 6 29 ILL 0.679192 ERA 0.9314735)

(2907 by 0.997186 1 1 BUY 0.997186 BUY 0.997186)

(2052 and 0.1496175 38 309 ANT 0.5470895 ILLNESS 0.7579887)

(2736 stray 0.93600553 1 1 STRAY 0.93600553 STRAY 0.93600553)

(4617 for 0.2175155 7 279 ALL 0.6265375 ARE 0.990799)

(4959 one 0.28297368 2 174 KNICK- 0.30177101 LID 0.6083893)

(0 of 0 1 640 nil 0 TEA 0.580014)

(4275 buy 0.959834 1 1 BUY 0.959834 BUY 0.959834)

(3420 be 0.452547 11 67 ILL 0.73377645 KNEE 0.9064575)

(4788 right 0.566736 4 40 I~D 0.6067855 AN 0.873578)

(3249 hand 0.083348334 9 783 HE 0.3612895 ALL 0.978445)

(3762 are 0.186133 1 255 ARE 0.186133 QUICK 0.84097964)

(5472 all 0.984129 1 1 ALL 0.984129 ALL 0.984129)

(2223 errors 0.81481236 2 5 EVER 0.8298127 ALL 0.927791)

(2736 nearest 0.47288325 7 42 KNEE 0.6990895 ERRORS 0.9959445)

(3933 may 0.457906 2 29 MAIN 0.476056 ARE 0.811062)

(3078 be 0.160617 6 246 EAT 0.56454754 EAT 0.9635745)

(0 distance 0 1 1416 nil 0 DIM 0.895919)

(2907 buys 0.95487654 4 5 BUY 0.996306 BUY 0.996306)

(0 catastrophic 0 1 2762 nil 0 TEA 0.99057245)

(3762 expense 0.6828584 1 34 EXPENSE 0.6828584 BE 0.997291)

(3591 his 0.010188666 12 579 EASE 0.033178 LIST 0.829572)

(0 he 0 1 643 nil 0 ATE 0.971926)

(0 street 0 1 982 nil 0 LOT 0.979736)

(3078 needed 0.7002863 3 12 KNEE 0.982007 KNEE 0.982007)

(2565 be 0.6845275 1 8 BE 0.6845275 CAN 0.813199)

(0 elegant 0 1 1379 nil 0 ELM 0.9427315)

(5472 or 0.210486 1 82 OR 0.210486 ELM 0.769936)

(4788 from 0.19615467 2 268 RUN 0.4887615 TEA 0.8771955)

(2394 answer 0.9052615 8 9 AN 0.9781835 AN 0.9781835)

(4275 are 0.833263 2 5 KNEE 0.942628 KNEE 0.942628)

(1197 for 0.462328 1 39 OR 0.462328 RIM 0.81242466)

(4617 big 0.6156905 2 7 IT 0.77520704 TELL 0.8289505)

(3591 ambled 0.642177 7 9 AIR 0.868595 AIR 0.868595)

(3591 increases 0.45626396 8 385 LET~S 0.6247363 NEAT 0.99631846)

(4617 for 0.040271334 6 575 REGARD 0.09105801 OF 0.93189)

(6498 all 0.242794 2 268 LAWS 0.5888225 OF 0.973385)

(3078 he 0.7721135 1 1 HE 0.7721135 HE 0.7721135)

(4275 lie 0.975584 1 1 LIE 0.975584 LIE 0.975584)

(4617 all 0.789721 1 2 ALL 0.789721 DOLL 0.7901865)

(0 once 0 1 1216 nil 0 ARE 0.970461)

(2907 cory 0.99151397 2 3 KEY 0.9985715 KEY 0.9985715)

(2223 and 0.233779 12 185 CAN~T 0.574616 OR 0.99344)

(4617 played 0.411271 18 170 PLANE 0.68859464 AIN~T 0.8110955)

(4104 for 0.0700685 5 275 RIDE 0.14295234 PA 0.5229645)

(2907 right 0.38094762 33 173 RON~S 0.80220866 THE 0.988127)

(4104 may 0.997625 1 1 MAY 0.997625 MAY 0.997625)

(4104 be 0.996968 1 1 BE 0.996968 BE 0.996968)

(3933 for 0.021034332 7 630 ROB 0.20541 AM 0.779864)

(0 business 0 1 1064 nil 0 ARE 0.910798)

(0 for 0 1 675 nil 0 OLE 0.7221335)

(5985 all 0.741487 1 2 ALL 0.741487 RAW 0.811157)

(3078 he 0.8284555 1 1 HE 0.8284555 HE 0.8284555)

(5130 lie 0.987753 1 1 LIE 0.987753 LIE 0.987753)

(7524 all 0.4718325 1 18 ALL 0.4718325 OF 0.884686)

(0 once 0 1 1398 nil 0 IN 0.928251)

(513 cory 0.97680765 1 3 CORY 0.97680765 OR 0.998108)

(2736 and 0.530195 6 62 CALF 0.6881885 OR 0.989617)

(3078 played 0.40020248 20 233 PAY 0.93032646 ATE 0.992266)

(4788 for 0.26758868 5 98 RISE 0.30035168 ALL 0.94527)

(2907 right 0.483153 9 95 ROCK 0.75791806 MAY 0.966908)

(2907 may 0.541507 12 91 MAIN 0.6819634 LAY 0.97726905)

(2907 be 0.781289 1 1 BE 0.781289 BE 0.781289)

(4446 for 0.06601467 7 410 OF 0.764737 OF 0.764737)

(0 business 0 1 973 nil 0 OF 0.868821)

(3762 are 0.039052 9 399 ROAD 0.0643245 SAW 0.502217)
(7182 all 0.393565 2 42 KNEE 0.6040865 OF 0.907752)
(0 errors 0 1 966 nil 0 KNEES 0.692177)
(4617 nearest 0.66095966 8 13 EAR 0.9452915 EAR 0.9452915)
(4104 may 0.423365 1 26 MAY 0.423365 MY 0.989461)
(0 be 0 1 648 nil 0 ACE 0.8981545)
(0 distance 0 1 1394 nil 0 IS 0.961375)
(2907 buys 0.6137945 7 7 BUY 0.915691 BUY 0.915691)
(0 catastrophic 0 1 3297 nil 0 REST 0.897919)
(2736 call 0.976327 1 1 CALL 0.976327 CALL 0.976327)
(513 an 0.313829 31 130 ALL 0.99931645 ALL 0.99931645)
(0 for 0 1 599 nil 0 MEN 0.724507)
(4959 can 0.942652 1 1 AN 0.942652 AN 0.942652)
(3933 buy 0.009188 7 779 BUYING 0.500659 LYING 0.955875)
(3420 mine 0.0843125 12 676 I~LL 0.4023795 IN 0.741367)
(3420 planned 0.41799632 30 140 LAY 0.8991925 LAY 0.8991925)
(342 is 0.570024 2 18 IN 0.6154745 OR 0.882951)
(4617 every 0.9407805 2 2 EVER 0.9578305 EVER 0.9578305)
(3933 related 0.21486725 8 573 ILL 0.52027947 KNICK- 0.874421)
(3249 my 0.355187 2 31 MINE 0.511854 DOLL 0.7299915)
(2907 ability 0.8247472 1 12 ABILITY 0.8247472 ILL 0.994556)
(3933 is 0.918467 1 1 IS 0.918467 IS 0.918467)
(3762 and 0.474731 1 180 AND 0.474731 ARE 0.921025)
(4104 are 0.748858 1 5 ARE 0.748858 STAR 0.857826)
(4617 spend 0.52113867 8 89 PAN 0.9779655 PAN 0.9779655)
(2394 in 0.35642648 2 47 IS 0.4397125 AN 0.895222)
(3591 miami 0.61820334 1 41 MIAMI 0.61820334 AND 0.969271)
(2736 is 0.038666 36 641 DISH 0.48110098 SPHERE 0.6750593)
(3078 be 0.037473 6 414 EAT 0.3521325 KNEE 0.946342)
(5814 need 0.084226 5 628 LEAF 0.4095915 KNICK- 0.9554285)
(4617 for 0.116187505 9 359 ALL 0.4697865 SAW 0.7748395)
(3762 and 0.124147 8 292 AIR 0.42909652 RAY 0.785141)
(2565 in 0.491492 1 7 IN 0.491492 SAY 0.5981785)
(3249 big 0.32406166 5 85 BE 0.53815997 BUY 0.978667)
(4788 can 0.9533145 1 1 AN 0.9533145 AN 0.9533145)
(3933 be 0.135168 2 363 ILL 0.4198905 AN 0.9764135)
(2052 an 0.485607 1 12 AN 0.485607 IN 0.5858715)
(4446 sport 0.6145024 4 31 PA 0.9179115 KNEE 0.997736)
(0 for 0 1 720 nil 0 RIDE 0.656766)
(0 pretty 0 1 1555 nil 0 TEA 0.97172654)
(684 in 0.117268 17 144 WICK 0.523299 KNEE 0.705815)
(3249 dance 0.37358502 13 255 TAN 0.5558273 SET 0.9965135)
(3591 and 0.013089667 3 301 SALT 0.014706001 EARS 0.7093715)
(3249 dislikes 0.82079715 5 36 DATE 0.8628125 ILL 0.994719)
(4446 for 0.496189 5 61 RAW 0.5017405 OR 0.963705)

(0 pretty 0 1 1655 nil 0 RAKED 0.9972455)
(684 in 0.5634175 1 4 IN 0.5634175 OWN 0.8436115)
(0 dance 0 1 1606 nil 0 ACT 0.77698195)
(4104 and 0.17320566 1 186 AND 0.17320566 ACT 0.871338)
(3249 dislikes 0.64105636 9 105 DEAR 0.85585195 IN 0.9805435)
(5472 for 0.4007265 1 1 OR 0.4007265 OR 0.4007265)
(0 pretty 0 1 1529 nil 0 AM 0.9911685)
(2223 in 0.43675947 1 6 IN 0.43675947 RAIN 0.5242203)
(2736 dance 0.41422665 4 98 CAN 0.46373233 AM 0.709285)
(1368 and 0.042073 22 271 AT 0.49658298 IT 0.7090875)
(3591 dislikes 0.75019646 3 26 DISK 0.832177 LIKE 0.96909547)
(4446 of 0.006158 10 419 AIN~T 0.62361467 KNIT 0.7231297)
(342 and 0.095333666 26 384 LIGHT 0.6205415 ATE 0.93550396)
(3420 prevented 0.31241018 21 500 RUN 0.745318 IN 0.8815105)
(4275 from 0.310694 3 131 FRAME 0.49171934 TEA 0.949278)
(0 contains 0 1 1913 nil 0 AN 0.812647)
(171 is 0.0979255 9 320 IT 0.457066 STATUS 0.7587625)
(0 controlled 0 1 1865 nil 0 ROCK 0.9027503)
(0 from 0 1 928 nil 0 THE 0.9741485)
(1881 is 0.546827 1 12 EARS 0.546827 ROSE 0.7186857)
(2565 for 0.21192399 13 200 FOSTERS 0.571049 SCHOLARS 0.7843865)
(3762 buyer 0.9712105 1 1 BUYER 0.9712105 BUYER 0.9712105)
(3762 money 0.9487015 1 2 MONEY 0.9487015 KNEE 0.997469)
(0 for 0 1 488 nil 0 KNEE 0.994122)
(2907 is 0.5204265 3 6 IT 0.525493 TRUCK 0.582838)
(0 controlled 0 1 3133 nil 0 ILL 0.8999145)
(6156 from 0.5190243 5 59 RUN 0.8244155 SON 0.94738495)
(2907 is 0.7745025 1 1 IS 0.7745025 IS 0.7745025)
(3591 for 0.971299 1 3 OR 0.971299 HE 0.98626304)
(6156 buyer 0.9327045 4 4 BUY 0.993053 BUY 0.993053)
(3762 money 0.70696163 1 27 MONEY 0.70696163 KNEE 0.995953)
(5643 for 0.195388 4 153 RUST 0.33259448 US 0.897539)
(1710 are 0.3504355 1 86 ARE 0.3504355 WE 0.9162665)
(2394 or 0.21038951 4 55 ON 0.290086 IN 0.4052385)
(0 proceeding 0 1 1681 nil 0 SEA 0.995776)
(3591 grows 0.5214253 2 8 REST 0.69385195 REST 0.694663)
(0 of 0 1 568 nil 0 NUTS 0.64122397)
(2394 and 0.290285 15 158 IF 0.6820545 KNEE 0.72044504)
(2223 prevented 0.31159735 23 142 PRINT 0.6367115 TIN 0.87030554)
(3933 from 0.0714715 5 330 RIM 0.298973 TEN 0.660129)
(0 contains 0 1 2337 nil 0 TEA 0.9073005)
(4959 of 0.0348455 2 441 URGING 0.33254933 AL 0.5631115)
(3249 and 0.089605 19 460 RACK 0.5084225 ALL 0.847959)
(3591 prevented 0.10798683 17 683 MINUTE 0.27276176 CAN 0.55607)
(3762 from 0.007928333 15 524 AL 0.363823 ILL 0.662272)

(0 contains 0 1 2179 nil 0 ARC 0.8166905)

(0 retracted 0 1 2392 nil 0 CAT 0.862888)

(3249 an 0.4710205 2 40 AM 0.4973665 IN 0.892914)

(3591 michael 0.90742755 2 2 MY 0.962333 MY 0.962333)

(6156 of 0.14661 1 158 OF 0.14661 ACT 0.81954503)

(3591 is 0.958272 1 2 IS 0.958272 KNEES 0.9982645)

(4446 big 0.4738925 4 59 IN 0.5749535 KNIT 0.96743095)

(2907 in 0.70157254 3 12 ILL 0.853472 US 0.981916)

(3762 are 0.632435 4 18 ARM 0.7865765 KNOCKED 0.869299)

(4788 for 0.755607 1 7 OR 0.755607 ARE 0.993783)

(6156 quick 0.40744534 3 109 CALL 0.476751 ARE 0.969373)

(1881 in 0.267763 3 32 SWIG 0.32640833 DIG 0.934959)

(3078 in 0.584768 1 22 IN 0.584768 ARE 0.997551)

(0 consists 0 1 2455 nil 0 TEA 0.967476)

(3762 of 0.137881 6 194 DIM 0.144742 BUY 0.848095)

(4959 and 0.35286567 2 141 DELIGHT 0.48898974 LIE 0.92606)

(3591 are 0.17587 3 118 LOCK 0.19160901 PA 0.69188154)

(1881 are 0.164621 9 280 CAR 0.52196753 HE 0.8669315)

(4104 for 0.6777485 2 14 FORM 0.7003226 LIE 0.989879)

(4104 big 0.602522 5 19 IN 0.95049596 IN 0.95049596)

(4788 ambled 0.67039937 4 15 AM 0.987842 AM 0.987842)

(0 increases 0 1 1849 nil 0 SEA 0.997177)

(0 retracted 0 1 1989 nil 0 PARK 0.82062)

(2736 an 0.114817 8 182 AM 0.5384545 KEYED 0.8746815)

(3591 michael 0.612689 3 20 I~LL 0.617528 HE~LL 0.8983865)

(3078 of 0.16344 1 212 OF 0.16344 IT 0.621655)

(684 is 0.570161 3 8 AM 0.626615 EASE 0.81445396)

(2565 big 0.182936 30 193 COUNT 0.534191 ILL 0.717885)

(3420 in 0.1561875 2 67 ILL 0.180489 US 0.508708)

(0 retracted 0 1 2646 nil 0 READ~V_PRES 0.826543)

(1197 an 0.193885 11 240 AM 0.51815146 EAT 0.837273)

(3249 michael 0.98393154 1 1 MICHAEL 0.98393154 MICHAEL 0.98393154)

(2907 of 0.0856325 2 312 THE 0.099802 ALL 0.877445)

(2907 is 0.4241225 1 16 IS 0.4241225 NEED 0.803488)

(4446 big 0.4643455 12 38 IT 0.7826005 IT 0.7826005)

(4446 in 0.5072425 7 37 ILL 0.7653475 US 0.99567)

(2907 customer 0.2706992 13 654 COURSE 0.33289066 LEAP 0.9418515)

(4788 spilled 0.625782 2 12 PILL 0.84381104 DISK 0.8931926)

(2223 some 0.5282273 2 16 SLID 0.529021 ELM 0.7957475)

(1368 please 0.447996 12 54 PLACE 0.65568 LAST 0.74928933)

(3249 dig 0.088084 8 721 DECK 0.511532 IN 0.939704)

(0 my 0 1 977 nil 0 IN 0.7254715)

(0 ride 0 1 1227 nil 0 BE 0.905133)

(6669 in 0.043132503 6 273 STICK 0.181892 BEEP 0.88983846)

(3933 made 0.165121 8 461 MAKE 0.5227025 AM 0.892793)

(3420 of 0.397391 1 18 OF 0.397391 AM 0.678249)

(3762 are 0.639015 2 8 ARM 0.77709 BE 0.995787)

(3078 in 0.6274035 2 12 IT 0.664878 TEN 0.7760933)

(0 stockings 0 1 2202 nil 0 CEASE 0.9758805)

(2907 run 0.29501566 26 248 READ~V_PAST 0.6540343 LIE 0.970846)

(3591 dishes 0.5675715 2 48 DENSE 0.600734 IS 0.975152)

(4788 makes 0.948242 1 4 MAKES 0.948242 SEA 0.99613)

(171 and 0.15636049 16 338 NEXT 0.464755 TEXT 0.89443)

(3762 made 0.6140315 7 9 MAY 0.98558 MAY 0.98558)

(6327 of 0.074383 1 314 OF 0.074383 I~M 0.56355697)

(3078 are 0.1874045 12 248 ROAM 0.4428005 PUMPED 0.53283876)

(5301 in 0.929425 1 1 IN 0.929425 IN 0.929425)

(0 stockings 0 1 2456 nil 0 BOX 0.856112)

(4275 run 0.366367 4 192 OWN 0.49622452 AM 0.848271)

(5472 dishes 0.660367 1 8 DISHES 0.660367 HE 0.73520553)

(5301 makes 0.9825553 4 4 ACHES 0.993424 ACHES 0.993424)

(5985 and 0.1184175 5 381 KNOCKED 0.21454935 ARM 0.5451945)

(4617 for 0.27693734 2 85 RUN 0.501332 OF 0.956916)

(5130 pretty 0.6544657 4 45 READ~V_PRES 0.84816265 TEA 0.9867315)

(855 in 0.603928 3 20 IT 0.6808205 KNEE 0.95469)

(3249 dance 0.63984966 7 40 TEA 0.7941755 AN 0.96578455)

(684 and 0.1324185 32 292 ANT 0.586987 ACT 0.9515295)

(0 dislikes 0 1 1664 nil 0 ILL 0.9601395)

(4446 expense 0.7596216 5 13 SENSE 0.967382 SENSE 0.967382)

(4959 his 0.08355566 2 301 IS 0.10708 WAS 0.567628)

(2907 he 0.749623 2 9 HECK 0.760419 LESS 0.788133)

(2736 street 0.8731268 8 22 SAY 0.988127 ATE 0.990252)

(3762 needed 0.67678374 3 18 KNEE 0.827742 TIM 0.9020286)

(3933 be 0.95599 1 1 BE 0.95599 BE 0.95599)

(0 elegant 0 1 1414 nil 0 IT 0.77017)

(342 or 0.174021 9 161 ARM 0.5082715 RAW 0.59813)

(5472 from 0.9558115 1 3 FROM 0.9558115 SOME 0.96671903)

(2565 answer 0.7580135 8 14 AN 0.882728 AN 0.882728)

(0 forces 0 1 1315 nil 0 OR 0.987609)

(3933 from 0.073127 32 442 TRY 0.480334 SHE 0.978408)

(3249 lily 0.2679435 6 201 IN 0.5128795 ARE 0.851787)

(4104 are 0.014656001 12 522 ODD 0.36625248 ACT 0.830106)

(3591 of 0.31546 3 118 SAY 0.423462 RAY 0.878229)

(1026 come 0.77633595 1 2 COME 0.77633595 DUMB 0.858336)

(3933 makes 0.65662235 6 14 ACHES 0.9655995 ACHES 0.9655995)

(0 combine 0 1 1115 nil 0 BUY 0.994382)

(4959 all 0.958342 1 2 ALL 0.958342 LAW 0.97528946)

(3078 in 0.0237655 10 316 ILL 0.5043615 LIE 0.9072515)

(4104 by 0.964935 2 4 I~M 0.979793 SOME 0.9947105)

(4959 and 0.29446033 4 377 LED 0.433824 KICK 0.8618355)

303

(4275 stray 0.98699194 1 1 TRAY 0.98699194 TRAY 0.98699194)

(1368 for 0.30986765 16 120 ARE 0.997043 ARE 0.997043)

(5643 one 0.43633166 4 78 KNICK- 0.6675827 ON 0.8909795)

(3591 of 0.040011 5 235 DID 0.13183634 ROCK 0.6789573)

(4446 buy 0.956038 1 2 BUY 0.956038 ARE 0.977779)

(3591 be 0.742095 1 4 BE 0.742095 EASE 0.999831)

(4446 right 0.28837752 17 220 RAY 0.490272 LACK 0.885265)

(6498 hand 0.42817 9 112 AN 0.996281 AN 0.996281)

(4959 every 0.9990295 1 1 EVERY 0.9990295 EVERY 0.9990295)

(2907 related 0.44909462 21 133 REAL 0.76389736 LAY 0.984313)

(3933 my 0.926636 1 1 MY 0.926636 MY 0.926636)

(2394 ability 0.6061372 3 59 SALT 0.642286 TEA 0.974556)

(4446 is 0.98424 1 1 IS 0.98424 IS 0.98424)

(4617 cat 0.591491 9 49 AL 0.9338845 THEN 0.99337304)

(2565 my 0.4540415 10 66 MONEY 0.8731494 SON 0.92857754)

(3591 by 0.143182 23 367 I~D 0.30190098 PA 0.874269)

(3933 of 0.006348 16 944 VAIN 0.06384 RAW 0.644332)

(3762 may 0.771679 1 3 MAY 0.771679 KNEE 0.9851785)

(2736 money 0.781385 1 7 MONEY 0.781385 OR 0.950433)

(2907 by 0.727661 1 12 BUY 0.727661 ARM 0.94702697)

(0 hard 0 1 1379 nil 0 ARE 0.991848)

(4617 candy 0.68516165 3 13 AN 0.97107553 KNEE 0.988752)

(2736 did 0.2997867 4 87 DEEP 0.611251 BUY 0.979622)

(3933 buy 0.993957 1 1 BUY 0.993957 BUY 0.993957)

(5130 any 0.5710857 4 45 KNEE 0.916275 BUY 0.988418)

(4275 bright 0.70411235 1 6 RIGHT 0.70411235 ARM 0.975189)

(0 red 0 1 816 nil 0 ROCK 0.75791144)

(3420 his 0.024862334 13 459 STACKED 0.32552403 PECK 0.7506515)

(5472 for 0.22345366 1 149 DOOR 0.22345366 ANT 0.6002915)

(3762 is 0.316191 1 126 IS 0.316191 SIT 0.784167)

(4446 am 0.886832 1 1 AM 0.886832 AM 0.886832)

(1026 and 0.041617002 14 238 ANT 0.354968 WAS 0.901433)

(5472 are 0.2018305 1 84 ARE 0.2018305 OF 0.830437)

(1710 are 0.5438975 7 32 HE 0.9611795 HE 0.9611795)

(0 or 0 1 701 nil 0 ILL 0.976047)

(0 proceeding 0 1 1944 nil 0 US 0.979233)

(2736 grows 0.32468253 9 107 GRAB 0.61099535 ERA 0.78116155)

(3420 made 0.20508951 11 352 CRISS- 0.4166005 TEA 0.8238725)

(4275 of 0.010019 9 498 QUICK 0.061544668 NEAT 0.891552)

(2394 are 0.414552 3 34 LOCK 0.56370246 BE 0.978928)

(2736 in 0.675133 5 6 IT 0.745776 IT 0.745776)

(3078 stockings 0.62385064 17 74 STAY 0.89382404 ROCK 0.972577)

(3249 run 0.39825597 28 193 RICE 0.8414853 ICE 0.9786635)

(3762 dishes 0.3910802 5 327 DEANS 0.48454103 REST 0.7681855)

(2736 makes 0.6982025 6 46 AM 0.959641 SEA 0.969239)

(0 and 0 1 469 nil 0 TEA 0.992323)
(4104 are 0.35005 3 32 ROCK 0.37062252 IS 0.997174)
(4104 for 0.7095225 5 7 OUGHT 0.8044215 OUGHT 0.8044215)
(4104 quick 0.302011 3 100 KEY 0.461446 SING 0.84925497)
(2223 in 0.258143 1 18 IN 0.258143 PAW 0.39657003)
(3249 in 0.1890435 2 74 WITH 0.20109701 BUY 0.8049575)
(0 consists 0 1 1297 nil 0 FIST 0.99154997)
(0 of 0 1 438 nil 0 SIT 0.9704745)
(4275 and 0.19544698 4 154 DEAR 0.25310698 SING 0.88849497)
(5814 are 0.0049765 3 419 ART 0.0345735 PAW 0.694688)
(2907 call 0.875729 1 3 CALL 0.875729 ON 0.968277)
(171 an 0.548427 1 22 AN 0.548427 ARE 0.98519)
(4275 for 0.2557685 6 172 ALL 0.651551 FROM 0.900974)
(4788 can 0.9009615 1 1 AN 0.9009615 AN 0.9009615)
(4275 buy 0.710919 1 2 BUY 0.710919 OR 0.857033)
(4104 mine 0.9498195 2 3 MY 0.960671 ARE 0.988213)
(3762 planned 0.637724 9 62 LIKE 0.86116266 CRACK 0.968679)
(2736 is 0.4788475 1 13 IS 0.4788475 EASE 0.78007746)
(3420 expense 0.5947454 3 104 EXPECT 0.6229712 IT 0.914091)
(4788 his 0.13221733 3 285 IS 0.276094 ACE 0.7157975)
(0 he 0 1 908 nil 0 ERA 0.8466505)
(0 street 0 1 861 nil 0 WICK 0.8521465)
(0 needed 0 1 1124 nil 0 AND 0.96709)
(4275 be 0.025206 6 587 BEEP 0.1420555 TEA 0.9724725)
(0 elegant 0 1 1430 nil 0 AL 0.994593)
(0 or 0 1 444 nil 0 LIE 0.90619)
(4617 from 0.6088783 3 20 RUN 0.90785754 RUN 0.90785754)
(5301 answer 0.54885435 3 45 ANT 0.64589846 ARC 0.7649235)
(2907 he 0.6483935 1 6 HE 0.6483935 QUICK 0.77675563)
(4617 from 0.6272177 2 11 RUN 0.649046 THE 0.7917725)
(4275 every 0.42015398 2 33 EVER 0.42262998 KNEE 0.7668975)
(1197 in 0.4782535 4 54 IT 0.543578 ARE 0.992748)
(3762 of 0.004275 4 580 EVER 0.3160905 US 0.8969605)
(5130 cleaned 0.41090965 3 170 CLEANS 0.5567297 TEA 0.83958554)
(4104 for 0.8555175 1 2 OR 0.8555175 SCORE 0.8651135)
(0 combine 0 1 879 nil 0 MAIN 0.81911135)
(4446 all 0.373455 2 77 DOLL 0.42675567 MAY 0.94514)
(3078 in 0.020505 26 323 ILL 0.506971 TEA 0.583995)
(4275 by 0.039996 32 665 MAY 0.616976 MY 0.941262)
(3078 and 0.161797 20 438 KNEE 0.4649435 NEAT 0.9031915)
(3591 stray 0.905693 4 7 TRAIN 0.9265587 RAIN 0.9362985)
(513 for 0.35730067 3 71 ORIGIN 0.53328705 ARE 0.996502)
(3078 one 0.581821 7 20 WE 0.865209 WE 0.865209)
(3762 of 0.210603 3 128 RUN 0.41099 RAW 0.79410946)
(2907 buy 0.968355 2 4 I~LL 0.9695155 ARE 0.9896115)

305

(3762 be 0.965341 1 2 BE 0.965341 ILL 0.967346)

(4617 right 0.5010265 4 30 RIPE 0.6087665 CRY 0.889336)

(6156 hand 0.6371227 3 14 AN 0.982571 AN 0.982571)

(5301 are 0.119466 4 248 ERA 0.28535548 TRAIN 0.6937023)

(5472 all 0.926113 1 1 ALL 0.926113 ALL 0.926113)

(4104 errors 0.98899996 1 1 ERRORS 0.98899996 ERRORS 0.98899996)

(4104 nearest 0.7579281 5 10 IN 0.9966675 IN 0.9966675)

(3762 may 0.517657 1 8 MAY 0.517657 ARE 0.896779)

(3078 be 0.194545 1 157 BE 0.194545 ARM 0.641486)

(3420 distance 0.6459192 11 31 DEAR 0.881236 DEAR 0.881236)

(2907 buys 0.798103 3 4 BUY 0.962894 BUY 0.962894)

(0 catastrophic 0 1 3329 nil 0 TEA 0.9020335)

(1881 are 0.3630765 3 44 ODD 0.739289 THE 0.973252)

(0 or 0 1 579 nil 0 I~VE 0.96294403)

(0 proceeding 0 1 1601 nil 0 SEA 0.995391)

(4446 grows 0.41009966 17 90 ROLE 0.615552 OAK 0.76439154)

(2565 is 0.8741965 2 2 WICK 0.90106 WICK 0.90106)

(0 for 0 1 781 nil 0 SKI 0.845081)

(5301 ended 0.4128945 7 149 DIM 0.6221653 AM 0.93013597)

(4104 please 0.19960399 17 353 EAT 0.5560055 THE 0.967048)

(0 cleaners 0 1 1470 nil 0 CLEAN 0.97738004)

(0 for 0 1 630 nil 0 MAY 0.943681)

(4275 cleaners 0.43855467 6 89 CLEAN 0.5228985 TRAIN 0.702025)

(171 and 0.27715 14 231 IN 0.48654452 RUN 0.5500855)

(5472 are 0.019608 10 352 AID 0.09395 KNEE 0.748146)

(4275 spend 0.4836217 13 61 SPENT 0.7579193 SPENT 0.7579193)

(3078 in 0.348754 1 2 IN 0.348754 TIN 0.43380448)

(3762 miami 0.5164487 1 26 MIAMI 0.5164487 TEA 0.8574625)

(2736 is 0.6977865 1 1 IS 0.6977865 IS 0.6977865)

(3078 be 0.993753 1 1 BE 0.993753 BE 0.993753)

(2565 need 0.340403 4 58 KNEE 0.620178 RUN 0.69343996)

(7011 for 0.08312634 7 318 ADD 0.35571003 ILL 0.871887)

(1026 and 0.0968655 14 612 VAULTING 0.381881 TALL 0.877313)

(4275 in 0.367938 5 56 IT 0.4351335 KNEES 0.68775797)

(4446 big 0.681351 1 11 BIG 0.681351 SAY 0.95214105)

(0 can 0 1 685 nil 0 BE 0.959199)

(3420 be 0.98315 1 1 BE 0.98315 BE 0.98315)

(3591 an 0.189566 28 183 AND 0.567212 BE 0.949433)

(4788 sport 0.51418567 7 18 PACK 0.818434 KNACK 0.8256875)

(3420 made 0.95283103 1 1 AID 0.95283103 AID 0.95283103)

(3591 of 0.027146 15 407 ACT 0.08245 MAKE 0.9187205)

(3078 are 0.953252 1 1 ARE 0.953252 ARE 0.953252)

(171 in 0.562705 1 8 IN 0.562705 THE 0.7023735)

(0 stockings 0 1 2465 nil 0 STOCK 0.995908)

(4104 run 0.913673 1 1 RUN 0.913673 RUN 0.913673)

(2052 dishes 0.40857852 2 398 DISK 0.56346303 SOME 0.98439)

(4104 makes 0.59021235 15 46 ATE 0.934686 ATE 0.988361)

(0 and 0 1 340 nil 0 KNEE 0.8555905)

(171 he 0.9172605 1 1 HE 0.9172605 HE 0.9172605)

(4617 from 0.331668 2 224 FRAME 0.336658 ATE 0.934411)

(3762 every 0.49218902 2 18 KNEE 0.5030065 EASE 0.935871)

(1368 in 0.3063505 10 82 IT 0.4977535 AT 0.630111)

(3762 of 0.0130225 1 499 OF 0.0130225 ATE 0.74061203)

(3762 cleans 0.38865525 19 196 KEY 0.81421196 KEY 0.81421196)

(3933 for 0.419846 1 48 OR 0.419846 FISH 0.956377)

(2565 customer 0.696346 1 24 CUSTOMER 0.696346 OF 0.921176)

(4617 spilled 0.7269047 1 11 SPILLED 0.7269047 ELSE 0.92272496)

(1539 some 0.034870334 12 452 OF 0.4611685 SEX 0.986068)

(4275 please 0.20984267 19 470 LEAST 0.780599 LEAST 0.82355565)

(0 dig 0 1 897 nil 0 AIR 0.7668725)

(1710 my 0.196981 16 553 MARKETS 0.5662316 ARE 0.963941)

(4446 ride 0.4139885 8 271 ART 0.5896623 ATE 0.924378)

(4446 in 0.300778 1 83 IN 0.300778 EAT 0.6923665)

(4275 forces 0.6008395 15 68 OFFER 0.99699247 OFFER 0.99699247)

(4104 from 0.45025602 11 31 REAL 0.7079243 IS 0.775393)

(855 lily 0.67164963 1 12 LILY 0.67164963 LEAD 0.9863795)

(171 are 0.631936 3 23 ARC 0.79821 CAT 0.85508454)

(3420 of 0.025864 5 643 VAULT 0.105037995 BUY 0.85267)

(2736 come 0.4187875 10 80 KEY 0.876138 TEA 0.94527197)

(2565 makes 0.554786 7 32 AM 0.982953 AM 0.982953)

(2736 cat 0.79937667 4 11 CAN 0.8466533 BE 0.955984)

(3249 my 0.05082 36 517 MINE 0.44738498 KNEE 0.84395)

(3078 by 0.319113 6 62 I~VE 0.63817847 IS 0.7278055)

(3591 of 0.224802 2 210 LAMPS 0.298864 ANT 0.777408)

(3420 may 0.994372 1 1 MAY 0.994372 MAY 0.994372)

(2565 money 0.6219138 12 37 MAIN 0.87258697 OF 0.992532)

(4104 by 0.961597 2 2 I~M 0.97637403 I~M 0.97637403)

(0 hard 0 1 1453 nil 0 ARE 0.995585)

(4788 candy 0.7092225 13 51 AN 0.9659385 AN 0.9659385)

(4446 are 0.998812 1 1 ARE 0.998812 ARE 0.998812)

(5985 for 0.25463632 5 328 CHORUSED 0.37175626 NEAT 0.9625765)

(171 quick 0.7866073 1 12 QUICK 0.7866073 ARE 0.946069)

(2052 in 0.5334605 1 9 IN 0.5334605 END 0.731756)

(3420 in 0.676669 1 5 IN 0.676669 BUY 0.859683)

(0 consists 0 1 1245 nil 0 SPHERE 0.920451)

(3762 of 0.917667 1 1 OF 0.917667 OF 0.917667)

(2736 and 0.401804 4 128 AM 0.48402652 SING 0.930416)

(4275 are 0.905255 1 2 ARE 0.905255 PA 0.9336165)

(3420 intelligence 0.5118874 2 153 INTELLIGENT 0.53903586 EDGE 0.8861355)

(2736 is 0.605402 8 12 IF 0.94380903 IF 0.94380903)

(4275 for 0.83752096 1 13 OR 0.83752096 KEY 0.941649)
(0 become 0 1 1623 nil 0 BE 0.969887)
(0 artists 0 1 1912 nil 0 ARE 0.999065)
(4959 fish 0.76207745 1 4 FISH 0.76207745 THIS 0.9889115)
(4446 leap 0.9622715 1 1 LEAP 0.9622715 LEAP 0.9622715)
(5814 frantically 0.5103947 3 293 ANT 0.5835723 ON 0.97726345)
(2565 of 0.316217 1 45 OF 0.316217 ODD 0.7949635)
(0 forces 0 1 1238 nil 0 STRESS 0.93730706)
(4104 from 0.4906535 2 8 REAL 0.512767 IS 0.820791)
(3249 lily 0.54333335 1 87 LILY 0.54333335 LED 0.9229655)
(4617 are 0.830769 1 6 ARE 0.830769 TEA 0.934932)
(6498 of 0.010472 1 728 OF 0.010472 ILL 0.615679)
(2907 come 0.7918515 5 13 COW 0.8600515 TOWN 0.9834515)
(4446 makes 0.94201165 4 7 MAY 0.996417 MAY 0.996417)
(2907 cat 0.77059835 2 8 CAT~S 0.82733977 ELSE 0.938785)
(3249 my 0.063865 8 690 AIR 0.2639265 ARE 0.8623)
(3249 by 0.1223 5 313 I~LL 0.519367 IN 0.961836)
(3933 of 0.014204 14 377 ILL 0.173008 AM 0.7062485)
(3420 may 0.867208 1 1 MAY 0.867208 MAY 0.867208)
(2223 money 0.6645517 5 24 MY 0.787362 MY 0.963306)
(3591 by 0.811009 1 7 BUY 0.811009 NOR 0.93372047)
(0 hard 0 1 1355 nil 0 ARE 0.991059)
(4446 candy 0.388333 28 262 ADD 0.791114 TAD 0.881927)
(4959 for 0.1684645 7 321 ON 0.345944 OF 0.917015)
(3078 pretty 0.45062673 7 115 PRAYED 0.55536026 KNEE 0.94666946)
(2565 in 0.196995 8 109 LID 0.437546 KNEE 0.915486)
(0 dance 0 1 1479 nil 0 AT 0.9657485)
(0 and 0 1 457 nil 0 AM 0.79846)
(0 dislikes 0 1 2473 nil 0 IT 0.8793355)
(0 retracted 0 1 2674 nil 0 RACK 0.9876833)
(1197 an 0.0689115 10 187 AT 0.47751898 TEA 0.541239)
(3420 michael 0.7428125 6 11 I~M 0.8559135 THE 0.8988155)
(3249 of 0.409402 7 115 AT 0.53922 ART 0.9003095)
(3078 is 0.7876595 1 4 EARS 0.7876595 KNEES 0.9990305)
(2907 big 0.63857096 6 9 BE 0.868748 BE 0.868748)
(2736 in 0.38117698 1 39 IN 0.38117698 ELSE 0.9379525)
(0 customer 0 1 1927 nil 0 US 0.994631)
(4788 spilled 0.7035637 7 41 KILLED 0.91600066 ILL 0.989218)
(1368 some 0.037465002 7 535 LOCKED 0.35230434 LET 0.910957)
(1368 please 0.17267467 22 309 PLACED 0.54949224 KISSED 0.6640133)
(3591 dig 0.12736 18 501 ACT 0.6418087 LISTED 0.84387374)
(0 my 0 1 580 nil 0 GRINNED 0.66892654)
(2736 ride 0.54452664 7 27 ROAD 0.8167493 ROAD 0.9402815)
(2565 in 0.22683099 9 130 IT 0.617258 AT 0.723625)
(4104 are 0.563125 2 28 RAW 0.7813345 OR 0.999544)

(4788 for 0.995764 1 1 OR 0.995764 OR 0.995764)

(1026 big 0.5722765 7 23 ILL 0.8902275 ILL 0.899718)

(3420 ambled 0.54366124 14 26 CALF 0.9260225 CALF 0.9260225)

(1368 increases 0.6184137 2 98 INCREASE 0.65949774 LEAST 0.978867)

(2907 customer 0.76898956 5 14 TEA 0.947944 US 0.99869454)

(4617 spilled 0.8451383 2 8 PILL 0.865184 US 0.98971)

(2907 some 0.57434696 9 42 SEX 0.63510865 ELM 0.925004)

(3933 please 0.48703066 4 70 EASE 0.730546 EASE 0.990975)

(4104 dig 0.391167 9 134 DIM 0.6540757 MY 0.964345)

(3249 my 0.962133 1 1 MY 0.962133 MY 0.962133)

(2907 ride 0.574511 8 72 RACK 0.8173527 ADD 0.9679675)

(2907 in 0.201971 6 52 SOME 0.505221 SOME 0.505221)

(5130 are 0.2361745 5 210 RUN 0.9478655 RUN 0.9478655)

(4104 all 0.345022 9 128 MONEY 0.71685934 TEA 0.7587085)

(5472 errors 0.433757 3 149 ROAD 0.46167198 LEAD 0.9250475)

(4104 nearest 0.28203964 16 239 IN 0.577139 BE 0.740341)

(3591 may 0.892892 1 1 MAY 0.892892 MAY 0.892892)

(2907 be 0.336387 1 22 BE 0.336387 ROCK 0.702994)

(2223 distance 0.6228146 1 34 DISTANCE 0.6228146 TEA 0.9120685)

(3078 buys 0.692306 3 5 BUY 0.720777 EYES 0.72473097)

(0 catastrophic 0 1 3294 nil 0 STOCK 0.97606003)

(0 combine 0 1 1522 nil 0 BUY 0.967192)

(5472 all 0.994891 1 1 ALL 0.994891 ALL 0.994891)

(3078 in 0.1521735 4 168 TEAM 0.494284 TEAM 0.660109)

(3933 by 0.292749 5 199 I~M 0.5886125 AM 0.87930596)

(1710 and 0.331002 17 331 DISK 0.653731 CENT 0.7352383)

(4788 stray 0.7577805 4 14 TRAIN 0.81299734 CRY 0.8825245)

(513 for 0.112957336 5 206 ORIGIN 0.21438925 IF 0.6195735)

(5985 one 0.28148398 9 209 AND 0.49153 READ~V_PRES 0.6938086)

(684 of 0.5077385 1 12 OF 0.5077385 LAWS 0.7075705)

(3078 buy 0.777875 1 6 BUY 0.777875 PA 0.98566604)

(3420 be 0.990867 1 1 BE 0.990867 BE 0.990867)

(4446 right 0.6455705 3 9 I~D 0.661486 AN 0.810198)

(6498 hand 0.30770633 2 173 AND 0.4615595 IN 0.76886404)

(4959 and 0.18954599 16 340 AN 0.375376 TIM 0.760818)

(4104 sprained 0.82729375 4 6 PAIN 0.9443275 PAIN 0.9443275)

(4959 steep 0.5203943 10 49 TEA 0.9196435 TEA 0.9196435)

(2907 can 0.42971066 6 46 CARE 0.6598744 AIR 0.86047745)

(3420 be 0.999839 1 1 BE 0.999839 BE 0.999839)

(3420 diseases 0.76576793 1 18 DISEASES 0.76576793 EASE 0.998306)

(4617 are 0.646894 1 3 ARE 0.646894 EAR 0.792076)

(2052 in 0.87319005 1 1 IN 0.87319005 IN 0.87319005)

(3249 he 0.74578655 1 2 HE 0.74578655 THESE 0.777815)

(4788 from 0.60703564 1 10 FROM 0.60703564 AIN~T 0.80482304)

(513 every 0.35394 8 196 TELL 0.621548 IF 0.92564654)

(1710 in 0.4313595 4 42 IT 0.5647165 IT 0.5647165)

(2736 of 0.035078 8 324 HONEST 0.48596674 DARK 0.54200935)

(855 cleans 0.17036575 42 526 RAIN 0.4130055 BRAIN 0.83505404)

(0 for 0 1 704 nil 0 ROLE 0.7933965)

(3420 of 0.162165 1 210 OF 0.162165 MOUNT 0.7292686)

(3762 and 0.39425197 4 51 AN 0.76499 KNEE 0.8294545)

(0 prevented 0 1 1974 nil 0 ARE 0.975677)

(0 from 0 1 540 nil 0 ERA 0.93445253)

(0 contains 0 1 2166 nil 0 TEA 0.99684155)

(5130 intelligence 0.8044853 2 24 INTELLIGENT 0.81194705 EDGE 0.99919105)

(5472 is 0.968724 1 1 IS 0.968724 IS 0.968724)

(7182 for 0.36635366 4 218 REAL 0.664661 REAL 0.9362235)

(0 become 0 1 1967 nil 0 BE 0.982635)

(4788 artists 0.24544099 42 653 ARE 0.932992 BUY 0.937533)

(5301 fish 0.7248385 2 3 IN 0.915028 IN 0.915028)

(4446 leap 0.493816 1 24 LEAP 0.493816 TEA 0.96826)

(5301 frantically 0.37154865 24 333 RAN 0.587496 KEY 0.91041553)

(171 of 0.501342 2 13 WAS 0.6137665 THE 0.8197855)

(0 retracted 0 1 2336 nil 0 RACK 0.957482)

(2052 an 0.2162705 20 152 SICK 0.61050606 OF 0.832635)

(3249 michael 0.958744 1 1 MICHAEL 0.958744 MICHAEL 0.958744)

(0 of 0 1 356 nil 0 DIM 0.78959095)

(1881 is 0.37623 5 42 DREAM 0.49455667 OF 0.8292395)

(4275 big 0.94293153 1 1 BIG 0.94293153 BIG 0.94293153)

(4104 in 0.339775 3 43 NEAR 0.43979633 EVERY 0.8512975)

(4959 are 0.259575 9 287 ARC 0.4677135 CRASH 0.6077113)

(3420 or 0.580724 1 4 OR 0.580724 ARE 0.984356)

(0 proceeding 0 1 2005 nil 0 SEA 0.99811)

(3933 grows 0.5591417 23 51 GRILL 0.9464357 GRILL 0.9464357)

(3933 call 0.73034453 1 2 ALL 0.73034453 RUN 0.86439097)

(4788 an 0.92885053 1 1 AN 0.92885053 AN 0.92885053)

(0 for 0 1 1008 nil 0 IT 0.874666)

(3933 can 0.959235 1 1 CAN 0.959235 CAN 0.959235)

(4275 buy 0.853348 1 5 BUY 0.853348 ARE 0.985204)

(3762 mine 0.505879 1 25 MINE 0.505879 EASE 0.998016)

(1197 planned 0.08599333 15 867 MEANT 0.47092232 QUICK 0.6518717)

(5472 is 0.4919295 3 55 IT 0.50881 OUGHT 0.97591996)

(2565 is 0.706433 5 11 IT 0.9841635 ARE 0.993545)

(0 controlled 0 1 1591 nil 0 SAYS 0.9839845)

(3762 from 0.344589 4 66 RAIN 0.5358917 SON 0.658935)

(2736 is 0.20857549 13 238 IT 0.53426003 REST 0.7541354)

(4617 for 0.6953935 5 19 OUGHT 0.95354295 OUGHT 0.95354295)

(3933 buyer 0.9765045 3 3 BUY 0.999261 BUY 0.999261)

(3078 money 0.6815513 1 19 MONEY 0.6815513 MY 0.965821)

(171 for 0.31384933 8 149 RUNS 0.57116896 KNEE 0.919641)

(3762 are 0.102933 3 224 ROAD 0.11515299 TAN 0.794415)
(6840 all 0.988596 1 2 ALL 0.988596 SALT 0.99028164)
(0 errors 0 1 1272 nil 0 ALL 0.893047)
(4104 nearest 0.5596807 5 51 IT 0.7421895 SON 0.799916)
(3591 may 0.937341 1 1 MAY 0.937341 MAY 0.937341)
(3249 be 0.25528 13 229 BUILT 0.6265753 PILL 0.87550247)
(3762 distance 0.43542424 3 139 REST 0.5544107 SETS 0.83637196)
(3078 buys 0.46647102 11 52 BITES 0.8312975 BITES 0.8312975)
(0 catastrophic 0 1 2599 nil 0 LAST 0.98519444)
(2394 are 0.006498 14 485 ODD 0.421895 IS 0.937666)
(4104 for 0.5060835 5 44 ALL 0.8072635 PICK 0.913553)
(2736 quick 0.57827365 8 41 KILL 0.809182 OR 0.970962)
(3933 in 0.2490525 3 103 ILL 0.2541455 AM 0.988533)
(513 in 0.27841 3 40 EARS 0.4405765 BUY 0.995865)
(0 consists 0 1 1352 nil 0 IT 0.940386)
(2736 of 0.851429 1 2 OF 0.851429 SIT 0.93266404)
(2565 and 0.1344525 8 264 TELL 0.584324 STREETS 0.92195153)
(3591 are 0.832452 2 3 ARC 0.8809045 ARC 0.8809045)
(2736 customer 0.8599866 1 7 CUSTOMER 0.8599866 US 0.99727)
(4788 spilled 0.9325223 2 4 PILL 0.9482985 PILL 0.9482985)
(2907 some 0.447724 7 80 SEX 0.8478367 ILL 0.996894)
(1539 please 0.60795504 1 26 PLEASE 0.60795504 EASE 0.9112005)
(3420 dig 0.48332602 1 39 DIG 0.48332602 ATE 0.981412)
(3933 my 0.794769 1 1 MY 0.794769 MY 0.794769)
(2907 ride 0.6087447 3 8 WRIST 0.6714106 RAIN 0.820413)
(4275 in 0.536494 2 2 NIP 0.643829 NIP 0.643829)
(3420 of 0.554975 1 29 OF 0.554975 AND 0.90585697)
(2052 and 0.127388 21 250 LAUGH 0.538555 TREE 0.7854405)
(3420 prevented 0.22422881 55 886 RIM 0.92980963 TRIM 0.942167)
(5130 from 0.4207315 1 37 FROM 0.4207315 AM 0.978893)
(0 contains 0 1 1573 nil 0 TEA 0.881098)
(6498 are 0.2457265 1 191 ARE 0.2457265 KEY 0.896343)
(4275 for 0.957553 1 2 OR 0.957553 OR 0.965848)
(4446 big 0.700608 5 8 ILL 0.7912135 ILL 0.7912135)
(4275 ambled 0.64620626 7 13 AM 0.864462 AM 0.956882)
(0 increases 0 1 1569 nil 0 CEASE 0.965352)
(4959 for 0.287623 6 86 ON 0.44945902 NOW 0.796309)
(0 pretty 0 1 1590 nil 0 I~D 0.8393295)
(342 in 0.5563405 1 1 IN 0.5563405 IN 0.5563405)
(4104 dance 0.65918064 1 21 DANCE 0.65918064 AN 0.949515)
(684 and 0.5202895 14 52 ANT 0.842291 TEN 0.86545604)
(3762 dislikes 0.6922227 9 53 ILL 0.87714505 LIKE 0.9881125)
(3762 are 0.061782 22 247 ARM 0.521851 LESS 0.7447065)
(0 for 0 1 484 nil 0 EAR 0.783261)
(5130 quick 0.8222285 1 9 WICK 0.8222285 EAT 0.9561425)

311

(4788 in 0.105167 5 227 NIP 0.2075245 LAW 0.4420625)

(2052 in 0.572961 2 8 DIG 0.5801695 ARE 0.987601)

(0 consists 0 1 1373 nil 0 OF 0.943267)

(3591 of 0.942059 1 1 OF 0.942059 OF 0.942059)

(2394 and 0.24205601 4 221 AL 0.58170754 ARE 0.76434255)

(4959 are 0.0126705 2 629 ODD 0.021116 TEA 0.87823653)

(2394 is 0.2736015 11 252 IT 0.7267795 IT 0.7267795)

(0 controlled 0 1 2161 nil 0 ODD 0.983926)

(4104 from 0.75789803 1 1 FROM 0.75789803 FROM 0.75789803)

(2223 is 0.0807145 10 364 WICK 0.4238055 RAPE 0.640555)

(4788 for 0.6092925 3 3 ON 0.7343955 ON 0.7343955)

(4275 buyer 0.9718395 1 1 BUYER 0.9718395 BUYER 0.9718395)

(3420 money 0.708845 2 4 US 0.9327085 US 0.9327085)

(342 for 0.16528967 22 400 KNEES 0.53427 KNEE 0.833307)

(0 of 0 1 721 nil 0 TIM 0.6788815)

(3762 and 0.134182 31 270 ANT 0.3890175 LAY 0.950136)

(3591 prevented 0.084134206 16 1009 ROB 0.44178602 IT 0.7976395)

(0 from 0 1 829 nil 0 SEND 0.55283433)

(0 contains 0 1 1788 nil 0 STRAINED 0.9250145)

(4104 and 0.20510833 5 211 DID 0.25674367 TEA 0.91370046)

(4617 are 0.36019748 1 24 ARE 0.36019748 THREE 0.62333035)

(4275 spend 0.46326566 2 90 SPILLED 0.46927235 MY 0.978548)

(342 in 0.5909885 9 22 AM 0.74917805 MY 0.966139)

(3249 miami 0.9447363 3 4 I~D 0.991473 I~D 0.991473)

(684 is 0.42749602 14 44 IT 0.82759905 IT 0.82759905)

(3078 be 0.653656 2 12 BEING 0.681274 NEED 0.957286)

(2565 need 0.638063 2 6 KNEE 0.87804 IF 0.900002)

(4617 for 0.8652715 1 1 OR 0.8652715 OR 0.8652715)

(1368 and 0.25122398 3 281 NEARING 0.318531 ROLE 0.85107005)

(513 in 0.32452 2 34 IS 0.5336035 EASE 0.67721355)

(4275 big 0.999287 1 1 BIG 0.999287 BIG 0.999287)

(2736 can 0.454583 2 27 KEY 0.60292053 BE 0.973139)

(3420 be 0.923087 1 1 BE 0.923087 BE 0.923087)

(2223 an 0.268826 4 72 THAN 0.34334052 BE 0.956785)

(5301 sport 0.47390667 12 174 CART 0.6281127 ARE 0.892985)

(5472 all 0.647157 1 8 ALL 0.647157 SAW 0.810637)

(2565 are 0.4571155 6 136 LOCK 0.76775 ALL 0.948955)

(4617 expected 0.5492617 10 83 STACKED 0.7324807 ALL 0.804497)

(2907 comes 0.121709995 18 604 COURSE 0.37037733 REST 0.7497035)

(5130 for 0.26589867 6 226 BORING 0.4284985 MAY 0.933095)

(2052 his 0.5696475 1 5 HIS 0.5696475 IS 0.68533)

(0 broken 0 1 1684 nil 0 KIN 0.635677)

(4446 all 0.901629 1 1 ALL 0.901629 ALL 0.901629)

(3591 are 0.972313 2 2 ARC 0.98222196 ARC 0.98222196)

(4959 expected 0.53827965 21 118 CENT 0.8555587 ARC 0.985645)

(171 comes 0.52415234 12 56 REST 0.7411865 SOME 0.851967)

(0 for 0 1 386 nil 0 RUN 0.812273)

(1881 his 0.5154885 1 4 HIS 0.5154885 THE 0.5859575)

(2907 broken 0.4907278 6 44 BELT 0.57648134 RENT 0.7241013)

(2736 call 0.70741934 4 6 CORN 0.7425173 OR 0.825639)

(1539 an 0.375216 3 52 RUN 0.60828197 AM 0.997437)

(0 for 0 1 421 nil 0 MEN 0.6114845)

(4617 can 0.60615647 1 1 AN 0.60615647 AN 0.60615647)

(2907 buy 0.47503302 8 83 BALL 0.71575403 ARE 0.991208)

(3591 mine 0.5331815 1 78 MINE 0.5331815 TRIED 0.814265)

(3420 planned 0.086512335 26 702 TEN 0.3818285 HEN 0.7003375)

(4104 is 0.4446375 1 59 IS 0.4446375 OR 0.979491)

(4617 every 0.844344 2 2 EVER 0.99694896 EVER 0.99694896)

(0 related 0 1 1265 nil 0 AIN~T 0.811343)

(3933 my 0.985477 1 1 MY 0.985477 MY 0.985477)

(3249 ability 0.362378 2 129 ABYSS 0.38104436 IN 0.8585845)

(2052 is 0.5198515 1 6 IS 0.5198515 HELL 0.7518625)

(3249 is 0.46853197 1 13 IS 0.46853197 ARE 0.9082)

(0 controlled 0 1 1569 nil 0 ELSE 0.9256515)

(5472 from 0.31060448 4 155 ROB 0.50110066 RAY 0.922059)

(1539 is 0.51759946 2 4 RIPE 0.58055604 RIPE 0.58055604)

(6498 for 0.6307355 5 7 OUGHT 0.96566653 OUGHT 0.96566653)

(4104 buyer 0.5623455 1 27 BUYER 0.5623455 DIM 0.93507254)

(4104 money 0.8814573 1 5 MONEY 0.8814573 KNEE 0.999766)

(3249 for 0.35635433 3 126 CHORUSED 0.3884872 KNEE 0.99935)

(2565 intelligence 0.52076185 5 51 ENGINE 0.67599326 TEN 0.9033555)

(1026 is 0.30459398 8 109 IF 0.5357565 LEAF 0.5639385)

(0 for 0 1 773 nil 0 KNEE 0.982102)

(0 become 0 1 1625 nil 0 BE 0.987363)

(0 artists 0 1 2152 nil 0 SHE 0.8535655)

(4446 fish 0.9354415 1 2 FISH 0.9354415 BE 0.983377)

(3933 leap 0.089551 16 633 EAT 0.4401735 TEA 0.940344)

(0 frantically 0 1 2890 nil 0 AN 0.834304)

(3591 of 0.959988 1 1 OF 0.959988 OF 0.959988)

(6156 and 0.040183667 4 255 ILL 0.05457 ACTS 0.67574733)

(4104 sprained 0.78803277 3 18 PAIN 0.98850703 PAIN 0.98850703)

(5301 steep 0.50072736 2 34 TEA 0.6404525 KNEE 0.9408915)

(0 can 0 1 742 nil 0 ARC 0.875213)

(3762 be 0.986445 1 1 BE 0.986445 BE 0.986445)

(3762 diseases 0.62148863 1 16 DISEASES 0.62148863 IT 0.89462197)

(1881 are 0.34065098 5 67 SOME 0.39545548 RAY 0.796336)

(3933 in 0.3540065 7 152 RISKS 0.6405156 SEA 0.970379)

(342 for 0.365057 6 151 ON 0.5054515 ALL 0.993864)

(4788 pretty 0.435367 12 106 RENT 0.6633123 KNEE 0.9692675)

(171 in 0.232987 9 101 EAR 0.479839 KNEE 0.6929615)

313

(0 dance 0 1 1032 nil 0 HOW 0.847875)
(2223 and 0.20224732 26 237 KNICK- 0.73492897 CENT 0.9029353)
(0 dislikes 0 1 1663 nil 0 ACT 0.9327345)
(2394 is 0.5131435 1 4 IS 0.5131435 WAS 0.57172596)
(4788 for 0.0159135 22 454 ROAM 0.503335 KEY 0.98454547)
(4275 ended 0.23631726 14 212 DIM 0.7827057 DIM 0.98747504)
(2052 please 0.39836535 4 131 PICK 0.5465825 WE 0.974495)
(4788 cleaners 0.36770633 2 97 IF 0.5494435 IF 0.64037)
(4104 for 0.9355215 3 3 OUGHT 0.9671535 OUGHT 0.9671535)
(2223 cleaners 0.2655725 30 312 KEY 0.7671885 TEA 0.846433)
(4275 the 0.454218 2 10 LED 0.692657 PECK 0.79161704)
(2565 the 0.745192 1 4 THE 0.745192 OR 0.853809)
(2565 the 0.902124 1 1 THE 0.902124 THE 0.902124)
(3762 the 0.96634 1 1 THE 0.96634 THE 0.96634)
(2736 the 0.25974947 7 123 THIS 0.668189 THIS 0.668189)
(2223 the 0.1079595 3 172 THUS 0.468537 US 0.741017)
(2736 sun 0.7762487 4 5 US 0.968899 US 0.968899)
(1368 the 0.3803755 3 49 THUS 0.628791 NECK 0.73513603)
(0 worn 0 1 1019 nil 0 RAIN 0.9902245)
(171 the 0.9818715 1 1 THE 0.9818715 THE 0.9818715)
(1368 the 0.005612 27 393 ROB 0.19858234 PARK 0.60054034)
(2223 the 0.450119 1 39 THE 0.450119 MY 0.968003)
(2223 will 0.31016633 2 58 KNEE 0.519023 BE 0.988622)
(3078 the 0.42693752 1 2 THE 0.42693752 OF 0.845197)
(2565 the 0.95262 1 1 THE 0.95262 THE 0.95262)
(2223 the 0.098076 2 176 WAS 0.2025785 WISH 0.659184)
(3078 the 0.089277 1 237 THE 0.089277 IN 0.472511)
(4617 the 0.473477 1 10 THE 0.473477 RENT 0.602642)
(3249 was 0.8800345 1 2 WAS 0.8800345 IS 0.91925)
(171 the 0.041274 19 269 ILL 0.8286385 BUY 0.999808)
(2394 the 0.575898 3 9 SOME 0.6027925 KNIT 0.78812504)
(0 treats 0 1 2297 nil 0 TEA 0.8071135)
(2565 the 0.82877004 1 1 THE 0.82877004 THE 0.82877004)
(3762 was 0.99162304 1 1 WAS 0.99162304 WAS 0.99162304)
(684 the 0.2619755 3 55 THEN 0.40894452 MEN 0.789021)
(2907 the 0.527025 1 14 THE 0.527025 CEASE 0.5825963)
(0 the 0 1 430 nil 0 ATE 0.843047)
(3933 we 0.0542905 16 572 CLASS 0.48572233 TELL 0.84219646)
(171 the 0.77612746 1 3 THE 0.77612746 BUY 0.853089)
(3591 wore 0.22754 12 191 WARN 0.44194567 IN 0.74606645)
(3078 the 0.5477635 1 34 THE 0.5477635 ALL 0.889541)
(2394 the 0.5104145 1 25 THE 0.5104145 AM 0.805595)
(171 this 0.500427 5 37 THE 0.7626285 WHISPER 0.8215667)
(171 the 0.442171 3 47 WITH 0.5069965 AM 0.9849055)
(684 the 0.027683001 5 519 DRINK 0.1115665 SKI 0.9797725)

(342 we 0.227364 7 208 ILL 0.4663275 AL 0.914455)
(0 was 0 1 584 nil 0 IT 0.866098)
(3591 the 0.943863 1 1 THE 0.943863 THE 0.943863)
(2736 the 0.504444 1 21 THE 0.504444 ARC 0.806921)
(2565 the 0.875511 1 1 THE 0.875511 THE 0.875511)
(2736 sun 0.84010935 1 1 SON 0.84010935 SON 0.84010935)
(2394 the 0.70100904 1 2 THE 0.70100904 RUN 0.775074)
(0 worn 0 1 790 nil 0 OWN 0.9890425)
(2565 the 0.5653075 1 1 THE 0.5653075 THE 0.5653075)
(2565 the 0.52978903 1 16 THE 0.52978903 ART 0.8994985)
(4275 this 0.47602868 8 89 SIT 0.714043 ATE 0.998719)
(3078 the 0.1697205 1 170 THE 0.1697205 KEY 0.677721)
(684 the 0.578504 1 3 THE 0.578504 KEY 0.657517)
(2223 the 0.812001 2 5 THEN 0.9888485 THEN 0.9888485)
(684 the 0.0254405 5 508 US 0.5077695 TEA 0.968306)
(2736 the 0.8278585 1 1 THE 0.8278585 THE 0.8278585)
(4104 the 0.2726575 2 41 THEN 0.43012297 TEN 0.5696705)
(2736 the 0.947765 1 1 THE 0.947765 THE 0.947765)
(3591 the 0.700835 1 2 THE 0.700835 TORE 0.749406)
(2907 the 0.4788925 1 6 THE 0.4788925 UP 0.609221)
(3591 will 0.43275467 9 56 WE 0.997546 WE 0.997546)
(3249 the 0.759889 1 2 THE 0.759889 OF 0.779293)
(2736 the 0.8803475 1 2 THE 0.8803475 FISH 0.98429)
(2736 the 0.620148 1 22 THE 0.620148 KNICK- 0.89830697)
(3078 the 0.49229702 1 22 THE 0.49229702 SOME 0.961711)
(3591 wore 0.1744375 21 258 I~M 0.846501 ARM 0.9558705)
(342 the 0.67778146 1 1 THE 0.67778146 THE 0.67778146)
(2736 than 0.144986 26 394 TREE 0.77871203 IT 0.9576535)
(5985 than 0.6719605 1 5 THAN 0.6719605 AN 0.802022)
(3078 was 0.51489 1 17 WAS 0.51489 US 0.964689)
(0 the 0 1 472 nil 0 CAR 0.651604)
(2907 the 0.44461998 3 45 AIM 0.5159605 ARE 0.98403)
(855 the 0.790439 1 6 THE 0.790439 HE~LL 0.8535045)
(3591 was 0.9885875 1 1 WAS 0.9885875 WAS 0.9885875)
(2907 wore 0.76599896 1 4 WAR 0.76599896 LOCK 0.8586675)
(2565 the 0.6881505 1 1 THE 0.6881505 THE 0.6881505)
(1539 than 0.067943 4 283 ANT 0.2357715 BUY 0.821261)
(1881 the 0.947352 1 1 THE 0.947352 THE 0.947352)
(4104 was 0.401308 5 210 LIGHT 0.49929652 SIT 0.8648545)
(2052 the 0.57808053 2 21 THESE 0.600977 ARE 0.880982)
(7353 wall 0.5117893 7 14 ALL 0.830434 ALL 0.830434)
(5130 the 0.22679001 1 136 THE 0.22679001 CAR 0.8500055)
(2736 the 0.9093465 1 1 THE 0.9093465 THE 0.9093465)
(3078 the 0.33977002 1 22 THE 0.33977002 KEY 0.9144565)
(3249 the 0.302076 3 85 COW 0.501854 LAY 0.8230875)

(0 the 0 1 331 nil 0 IT 0.87952554)

(171 the 0.8373985 1 3 THE 0.8373985 HAM 0.86587)

(171 the 0.15303999 9 385 EAR 0.42136252 TEA 0.9124595)

(3420 the 0.195832 1 111 THE 0.195832 DIM 0.7437995)

(4275 we 0.881949 1 3 WE 0.881949 ARE 0.933848)

(3933 was 0.4865135 1 72 WAS 0.4865135 TEA 0.92392254)

(0 the 0 1 555 nil 0 IS 0.999555)

(0 unlimited 0 1 1958 nil 0 ERA 0.836439)

(0 the 0 1 208 nil 0 ATE 0.990514)

(5643 wall 0.084955 22 502 LOCKED 0.419337 LAST 0.9872987)

(0 the 0 1 356 nil 0 SON 0.89699)

(2736 the 0.1134265 5 421 DANCE 0.4980867 KNEE 0.93237)

(3078 wore 0.18724449 17 294 OR 0.7527705 OR 0.7527705)

(2565 the 0.24662651 2 70 CAT 0.25278297 DONE 0.86283946)

(0 than 0 1 692 nil 0 RAY 0.653137)

(0 task 0 1 1065 nil 0 SAW 0.81003594)

(2394 will 0.771551 3 4 EAR 0.833609 EAR 0.833609)

(2565 this 0.4865255 3 33 THE 0.97639 THE 0.97639)

(171 the 0.75475645 1 1 THE 0.75475645 THE 0.75475645)

(4275 task 0.381098 21 486 TELLS 0.823551 CELLS 0.9142103)

(3249 will 0.6232765 3 15 EAR 0.665694 AIR 0.963568)

(0 this 0 1 755 nil 0 SEA 0.918846)

(2736 the 0.5582475 1 1 THE 0.5582475 THE 0.5582475)

(3078 the 0.5606495 2 7 THUS 0.7237975 US 0.932176)

(684 will 0.4549725 1 50 ILL 0.4549725 CALL 0.96442103)

(2736 trish 0.6122495 9 83 TRICK 0.7439185 EDGE 0.989782)

(3420 the 0.8483325 1 3 THE 0.8483325 IN 0.88464653)

(342 the 0.94074655 1 1 THE 0.94074655 THE 0.94074655)

(3591 was 0.0752195 16 347 LIKED 0.43082923 NEAT 0.934113)

(3078 the 0.17501 5 157 AND 0.24300967 READ~V_PRES 0.62223667)

(7695 wall 0.282456 5 272 LED 0.4796063 RUN 0.8712685)

(0 the 0 1 385 nil 0 EVERY 0.92699397)

(0 the 0 1 655 nil 0 BE 0.985547)

(3249 the 0.55738354 1 9 THE 0.55738354 MY 0.822937)

(0 the 0 1 533 nil 0 KNICK- 0.559777)

(3762 these 0.2589925 2 191 EASE 0.29168552 BE 0.872134)

(0 than 0 1 418 nil 0 EASE 0.959365)

(3249 the 0.26742 1 64 THE 0.26742 EVER 0.7157455)

(342 the 0.932883 1 1 THE 0.932883 THE 0.932883)

(2394 the 0.7092755 1 1 THE 0.7092755 THE 0.7092755)

(2565 the 0.91542447 1 1 THE 0.91542447 THE 0.91542447)

(1368 will 0.64829135 5 15 WE 0.97806096 WE 0.97806096)

(2907 trish 0.84094745 1 5 TRISH 0.84094745 PLAY 0.93975246)

(1197 the 0.5186895 2 23 STAB 0.5315227 AN 0.8217035)

(342 the 0.6890155 1 1 THE 0.6890155 THE 0.6890155)

316

(3078 the 0.878139 1 2 THE 0.878139 TORE 0.9235275)

(2565 the 0.760522 1 1 THE 0.760522 THE 0.760522)

(2907 will 0.208685 6 199 VAULT 0.5231597 TEA 0.978969)

(3078 the 0.7097615 1 1 THE 0.7097615 THE 0.7097615)

(3249 the 0.603384 1 1 THE 0.603384 THE 0.603384)

(2907 the 0.7708205 1 3 THE 0.7708205 ERA 0.884822)

(2907 the 0.9070835 1 2 THE 0.9070835 OF 0.9169755)

(2907 the 0.6081685 2 5 THESE 0.60943866 IN 0.877829)

(2736 the 0.716003 1 1 THE 0.716003 THE 0.716003)

(5985 the 0.0043555 3 644 LED 0.17543 ARE 0.919311)

(0 unlimited 0 1 2224 nil 0 IN 0.883306)

(4788 the 0.370967 3 47 THIS 0.4145955 EASE 0.7495205)

(5130 wall 0.22401601 16 364 US 0.5144835 SLEIGH 0.797277)

(3420 the 0.19431551 1 115 THE 0.19431551 IS 0.74428)

(2736 the 0.3941055 9 123 DUMB 0.6693135 SON 0.95748055)

(684 the 0.973424 1 1 THE 0.973424 THE 0.973424)

(2565 the 0.377638 3 50 ACT 0.48215303 TEA 0.615911)

(0 this 0 1 752 nil 0 ATE 0.986086)

(3420 the 0.050598 2 347 SILLY 0.26855233 STILL 0.85328054)

(171 the 0.5140325 1 1 THE 0.5140325 THE 0.5140325)

(2736 the 0.9737675 1 1 THE 0.9737675 THE 0.9737675)

(171 the 0.7379085 1 1 THE 0.7379085 THE 0.7379085)

(2223 the 0.377769 3 77 SLEIGH 0.57948697 SLEIGH 0.9008975)

(3933 this 0.41908264 1 32 MISS 0.41908264 ACE 0.624796)

(3420 the 0.0547825 1 148 THE 0.0547825 BIKE 0.8845985)

(2565 the 0.55574 1 1 THE 0.55574 THE 0.55574)

(3762 will 0.59611833 2 37 WALL 0.887725 ALL 0.992549)

(2907 trish 0.8684265 1 14 TRISH 0.8684265 EAT 0.96680796)

(0 the 0 1 565 nil 0 BE 0.864258)

(513 the 0.6295435 1 2 THE 0.6295435 HA 0.6311115)

(3078 was 0.5228165 1 94 WAS 0.5228165 WISH 0.9261615)

(855 the 0.1358285 15 185 ILL 0.6711355 ILL 0.807344)

(513 the 0.367979 1 19 THE 0.367979 RAY 0.708896)

(2736 treats 0.66461104 19 33 TREES 0.89496154 EARS 0.910582)

(6498 than 0.83598197 1 7 THAN 0.83598197 AN 0.968299)

(3078 was 0.918036 1 1 WAS 0.918036 WAS 0.918036)

(2394 the 0.3632665 1 32 THE 0.3632665 ARE 0.752489)

(2394 the 0.528739 1 16 THE 0.528739 DEEP 0.9178345)

(1881 the 0.1385545 1 297 THE 0.1385545 ILL 0.980646)

(3591 was 0.5198695 1 15 WAS 0.5198695 ON 0.918822)

(7011 will 0.4034745 1 35 ILL 0.4034745 ALL 0.770763)

(2565 trish 0.585051 10 23 TEA 0.862162 TEA 0.862162)

(513 the 0.50764847 1 17 THE 0.50764847 BE 0.957186)

(2907 the 0.999707 1 1 THE 0.999707 THE 0.999707)

(2736 the 0.9075155 1 1 THE 0.9075155 THE 0.9075155)

(2907 the 0.946249 1 2 THE 0.946249 IN 0.96514153)
(5130 sun 0.71402 3 7 ON 0.7286515 IN 0.9005995)
(2907 the 0.3672275 1 52 THE 0.3672275 RACK 0.823037)
(0 worn 0 1 1185 nil 0 OR 0.987955)
(4104 wore 0.491969 5 51 WARM 0.63495564 ARM 0.87829196)
(171 the 0.7414875 1 1 THE 0.7414875 THE 0.7414875)
(2394 than 0.133799 7 201 LAID 0.32271135 IS 0.785685)
(342 the 0.604874 1 3 THE 0.604874 QUICK 0.61750335)
(2394 the 0.486619 1 9 THE 0.486619 OF 0.763142)
(4275 this 0.32955334 8 264 MR. 0.4968015 TITTER 0.9013066)
(3078 the 0.1266185 1 210 THE 0.1266185 CRY 0.8856445)
(342 the 0.4891925 5 10 WITH 0.6598315 WITH 0.6598315)
(0 the 0 1 359 nil 0 KNEE 0.86991596)
(3078 the 0.7683375 1 4 THE 0.7683375 BUY 0.999634)
(3078 the 0.414173 1 51 THE 0.414173 SIT 0.727611)
(5472 these 0.0053806663 1 726 CHEESE 0.0053806663 ARE 0.995406)
(0 than 0 1 590 nil 0 BE 0.999217)
(0 the 0 1 279 nil 0 IN 0.75825155)
(684 task 0.15120566 37 821 START 0.4555007 PART 0.72664636)
(2907 will 0.35023698 10 41 WE~RE 0.6189813 ARE 0.9031225)
(0 this 0 1 650 nil 0 SEA 0.747749)
(342 the 0.551514 1 2 THE 0.551514 QUICK 0.5647867)
(0 task 0 1 1099 nil 0 SICK 0.820452)
(2736 will 0.5615515 1 8 ILL 0.5615515 THE 0.8113395)
(0 this 0 1 611 nil 0 ADD 0.63919747)
(171 the 0.683971 1 1 THE 0.683971 THE 0.683971)
(2394 the 0.865638 1 1 THE 0.865638 THE 0.865638)
(3249 the 0.62504 1 2 THE 0.62504 WICK 0.645445)
(2565 the 0.8065915 1 1 THE 0.8065915 THE 0.8065915)
(3420 will 0.040328667 17 305 LOCK 0.43002 BE 0.887826)
(3591 the 0.6385975 1 3 THE 0.6385975 ALL 0.804233)
(3591 the 0.97034 1 1 THE 0.97034 THE 0.97034)
(2907 the 0.6542155 1 2 THE 0.6542155 OF 0.737128)
(3420 the 0.8365265 2 2 THUS 0.852963 THUS 0.852963)
(0 the 0 1 497 nil 0 ALL 0.935096)
(3420 the 0.6601355 1 6 THE 0.6601355 OF 0.821017)
(2565 the 0.5435635 1 6 THE 0.5435635 CAMPUS 0.5656137)
(0 these 0 1 976 nil 0 ARE 0.999778)
(4959 than 0.079916 18 420 KNEE 0.4675295 KEY 0.894155)
(3249 the 0.254316 1 60 THE 0.254316 EVER 0.70559454)
(0 the 0 1 590 nil 0 LIE 0.94849)
(4104 we 0.948938 1 2 WE 0.948938 ALL 0.951947)
(4275 was 0.190098 2 132 CUISINE 0.35340834 IS 0.985833)
(171 the 0.6675005 1 1 THE 0.6675005 THE 0.6675005)
(5472 the 0.4215855 1 47 THE 0.4215855 KISSED 0.68443567)

(342 the 0.72726953 1 8 THE 0.72726953 WICK 0.97505605)
(11286 was 0.0451375 1 944 WAS 0.0451375 OFF 0.99909496)
(684 the 0.091617 3 228 THAN 0.22057751 BUY 0.989939)
(2907 the 0.928851 1 1 THE 0.928851 THE 0.928851)
(5472 treats 0.54150504 24 142 TRICK 0.77524203 TEA 0.96630645)
(171 the 0.8175795 1 9 THE 0.8175795 HE 0.974051)
(3249 the 0.12563601 1 423 THE 0.12563601 STICK 0.96876)
(2907 the 0.861535 1 1 THE 0.861535 THE 0.861535)
(0 the 0 1 578 nil 0 IN 0.899881)
(0 unlimited 0 1 2227 nil 0 EAT 0.8252105)
(0 the 0 1 318 nil 0 TIM 0.736835)
(5301 wall 0.028018 8 617 LATE 0.18272734 LOT 0.91990197)
(3249 the 0.14722951 7 167 TEA 0.5551365 AT 0.942549)
(2565 the 0.083315 4 220 THEN 0.183807 RUN 0.72707)
(0 the 0 1 628 nil 0 ILL 0.581221)
(0 unlimited 0 1 2232 nil 0 EAT 0.977455)
(3249 the 0.238753 1 115 THE 0.238753 EASE 0.747286)
(2394 wall 0.136533 24 482 LOSS 0.5021535 COME 0.919701)
(3249 the 0.7424165 1 5 THE 0.7424165 REST 0.98474735)
(2907 the 0.3277725 5 22 THESE 0.57103765 EAT 0.664331)
(3420 the 0.184094 2 157 EAT 0.448576 IT 0.804107)
(2907 the 0.5786055 1 6 THE 0.5786055 HE 0.95671654)
(1710 the 0.23177901 14 143 SAY 0.7617705 PAY 0.9295455)
(2736 the 0.045766503 6 344 CEASE 0.3783015 DISH 0.855828)
(171 the 0.563142 2 8 THESE 0.66874903 DIG 0.78841054)
(2907 was 0.77284294 1 1 WAS 0.77284294 WAS 0.77284294)
(5643 the 0.008665999 3 670 THAN 0.031233 AND 0.84428847)
(3078 the 0.522916 1 2 THE 0.522916 TEA 0.53317)
(0 treats 0 1 1698 nil 0 US 0.9660955)
(3420 the 0.0555795 8 265 SOLID 0.22996876 LIVE~V 0.6756325)
(4275 we 0.992549 1 1 WE 0.992549 WE 0.992549)
(3591 was 0.1103855 10 240 ABYSS 0.234242 NEED 0.6386025)
(513 the 0.826066 1 1 THE 0.826066 THE 0.826066)
(2565 the 0.0648645 2 233 WICK 0.2388435 ATE 0.965426)
(2736 the 0.4616065 4 46 ACE 0.6350225 KNEES 0.94346297)
(171 the 0.7417825 1 1 THE 0.7417825 THE 0.7417825)
(1026 will 0.2440545 9 133 WICK 0.5475285 HE 0.7620995)
(2907 trish 0.6295907 8 29 TRIP 0.9229546 TRIP 0.9229546)
(0 the 0 1 416 nil 0 BEER 0.62897867)
(2736 the 0.87374854 1 3 THE 0.87374854 BUY 0.922286)
(4275 was 0.49964 1 5 WAS 0.49964 ARE 0.942503)
(3762 the 0.773248 1 2 THE 0.773248 CALF 0.831834)
(3933 the 0.2477015 14 261 DIM 0.444474 ROAM 0.94182366)
(2565 the 0.304686 1 18 THE 0.304686 OF 0.4582065)
(5301 we 0.047595 1 432 WE 0.047595 LET 0.53787297)

319

(2736 the 0.530248 1 12 THE 0.530248 LINE 0.92718947)
(2736 the 0.7808835 1 3 THE 0.7808835 BUY 0.86987)
(4104 was 0.748237 1 1 WAS 0.748237 WAS 0.748237)
(684 the 0.128176 12 167 THAN 0.463631 AN 0.906402)
(3762 the 0.35616 1 100 THE 0.35616 IF 0.863239)
(2565 the 0.705183 1 1 THE 0.705183 THE 0.705183)
(5301 we 0.0277665 13 329 TAN 0.3483285 ILL 0.800423)
(2565 the 0.9637505 1 1 THE 0.9637505 THE 0.9637505)
(0 wore 0 1 756 nil 0 ARE 0.933507)
(2565 the 0.090088 8 225 EAR 0.363051 COST 0.7579125)
(2736 the 0.9881815 1 1 THE 0.9881815 THE 0.9881815)
(2907 this 0.7345645 1 1 THIS 0.7345645 THIS 0.7345645)
(0 the 0 1 859 nil 0 ILL 0.498484)
(513 the 0.39701098 1 41 THE 0.39701098 BUY 0.992124)
(3078 the 0.61956847 1 7 THE 0.61956847 SWISS 0.72284967)
(5472 these 0.2369235 1 338 THESE 0.2369235 NUT 0.760216)
(6156 than 0.51579297 2 17 TEA 0.6244575 TEA 0.849308)
(2736 the 0.0464105 2 254 OF 0.0464495 TEAM 0.62928265)
(2736 the 0.84915555 1 1 THE 0.84915555 THE 0.84915555)
(2907 the 0.4007385 1 60 THE 0.4007385 AIM 0.74190354)
(6156 the 0.015384 7 598 LID 0.035053 SEA 0.9563885)
(4104 unlimited 0.11347401 56 877 KNEE 0.5010205 BE 0.996057)
(3420 the 0.55884004 1 21 THE 0.55884004 AIN~T 0.944098)
(3249 wall 0.544594 1 79 WALL 0.544594 BLOCK 0.98272467)
(3078 the 0.060517997 1 233 THE 0.060517997 CAST 0.927329)
(342 the 0.17281699 8 265 KEY 0.466686 AT 0.9217)
(2907 the 0.922265 1 1 THE 0.922265 THE 0.922265)
(3420 was 0.69346803 1 10 WAS 0.69346803 ILL 0.8634765)
(2907 the 0.248667 1 73 THE 0.248667 IT~S 0.547809)
(2907 the 0.39461002 1 66 THE 0.39461002 IF 0.965405)
(3249 the 0.762645 1 1 THE 0.762645 THE 0.762645)
(2907 we 0.5809535 2 4 WE~LL 0.709991 HE~LL 0.71309)
(3591 will 0.3941975 1 15 ILL 0.3941975 LIE 0.5468085)
(5130 trish 0.8704037 2 6 TRICK 0.8798137 LAY 0.969091)
(1026 the 0.6899405 1 7 THE 0.6899405 BE 0.997985)
(342 will 0.3346825 13 55 WICK 0.641755 KEY 0.8612755)
(1881 trish 0.44326332 30 251 TEA 0.9446345 KNEE 0.972317)
(684 the 0.5195885 1 20 THE 0.5195885 BE 0.977411)
(2736 the 0.6839325 1 1 THE 0.6839325 THE 0.6839325)
(2736 the 0.541995 1 19 THE 0.541995 AN 0.86701)
(3591 the 8.59e-4 14 509 ALL 0.3633925 ELSE 0.8800665)
(3591 we 0.914141 1 2 WE 0.914141 ILL 0.9192935)
(3762 was 0.53423804 2 23 OF 0.546967 KNEE 0.8964095)
(2223 under 0.101728 36 450 ILL 0.44404498 LEAVE 0.821977)
(2565 the 0.40935 1 17 THE 0.40935 WITH 0.49506152)

(0 under 0 1 1034 nil 0 RAY 0.927307)

(171 the 0.644195 1 5 THE 0.644195 RAY 0.764272)

(3933 under 0.322684 7 350 NEAR 0.53844064 LEAD 0.9329965)

(2565 the 0.1400865 1 91 THE 0.1400865 BIKE 0.799939)

(342 the 0.4812945 2 23 IS 0.492935 LET 0.6978255)

(2223 the 0.34521002 3 63 THESE 0.53299564 BE 0.929587)

(0 the 0 1 235 nil 0 NEAT 0.9254805)

(2565 the 0.6228265 1 1 THE 0.6228265 THE 0.6228265)

(3249 the 0.996065 1 1 THE 0.996065 THE 0.996065)

(3591 wore 0.00825 31 900 ALL 0.447578 ILL 0.76312196)

(2565 the 0.466874 4 39 LET~S 0.52638966 ROAST 0.9610065)

(2565 the 0.91719496 1 1 THE 0.91719496 THE 0.91719496)

(171 this 0.5684395 2 5 THE 0.8452945 THE 0.8452945)

(513 the 0.0510915 5 235 DUMPED 0.3636655 IN 0.861077)

(0 wore 0 1 931 nil 0 OF 0.765463)

(2736 the 0.8338095 1 1 THE 0.8338095 THE 0.8338095)

(0 the 0 1 656 nil 0 RUN 0.622997)

(5130 this 0.38540533 6 75 GUESS 0.5864833 DESK 0.848067)

(4275 than 0.23620832 21 158 KNEE 0.55023396 KNEE 0.906168)

(3249 was 0.88548696 1 1 WAS 0.88548696 WAS 0.88548696)

(4446 the 0.1149365 4 216 MICHAEL 0.49387452 LIVE~V 0.8558395)

(0 the 0 1 558 nil 0 CANDY 0.5451047)

(171 the 0.720538 1 5 THE 0.720538 HE~LL 0.79148996)

(3933 was 0.998739 1 1 WAS 0.998739 WAS 0.998739)

(2052 the 0.76296103 1 2 THE 0.76296103 SPHERE 0.76542497)

(2736 the 0.26055 2 83 SILLY 0.31010064 AND 0.7256235)

(2907 the 0.293741 2 92 HONEST 0.45881552 ALL 0.7884)

(2565 the 0.978893 1 1 THE 0.978893 THE 0.978893)

(171 the 0.564294 2 2 THESE 0.590451 THESE 0.590451)

(0 the 0 1 463 nil 0 AN 0.598126)

(171 the 0.0529015 13 232 PIG 0.46038198 IS 0.977372)

(2736 the 0.779508 1 1 THE 0.779508 THE 0.779508)

(1197 the 0.53478 1 15 THE 0.53478 DID 0.81160796)

(2736 was 0.3627725 12 101 IT 0.851107 IT 0.851107)

(0 the 0 1 395 nil 0 EAR 0.835273)

(7524 wall 0.21853699 21 262 ALL 0.629417 AIR 0.7910735)

(513 the 0.1132985 2 114 THOU 0.139966 SEA 0.796709)

(2907 the 0.569126 1 9 THE 0.569126 SIT 0.6418495)

(4446 the 0.3367365 3 14 MICHAEL 0.4360955 MICHAEL 0.4360955)

(2565 the 0.8846445 1 5 THE 0.8846445 BUY 0.987337)

(2736 the 0.81858397 1 3 THE 0.81858397 BUY 0.969478)

(4275 was 0.977543 1 1 WAS 0.977543 WAS 0.977543)

(3249 the 0.879111 1 2 THE 0.879111 ERA 0.98118055)

(3249 the 0.5387715 1 5 THE 0.5387715 OF 0.961559)

(2565 the 0.34568697 1 19 THE 0.34568697 COUNT 0.59965533)

(2736 we 0.513766 3 8 WE~RE 0.61372036 EAR 0.6751985)
(2565 the 0.5324095 2 4 THIS 0.7468715 HIS 0.873003)
(3249 was 0.47012052 1 22 WAS 0.47012052 ARM 0.8285225)
(2565 the 0.40560502 7 21 STILL 0.4694635 IT 0.60864)
(6156 wall 0.19532901 1 233 BALL 0.19532901 RAN 0.62830335)
(2736 the 0.456375 1 1 THE 0.456375 THE 0.456375)
(2565 the 0.66212 1 2 THE 0.66212 MAY 0.664955)
(3078 was 0.5509845 1 12 WAS 0.5509845 IT 0.8173045)
(2565 the 0.0783265 4 126 ACT 0.38905367 ART 0.667902)
(4275 wall 0.545503 1 7 WALL 0.545503 ALL 0.999688)
(0 the 0 1 358 nil 0 TRAY 0.879961)
(2565 the 0.580187 1 4 THE 0.580187 IT 0.6887785)
(2565 the 0.49063 1 13 THE 0.49063 WICK 0.65976346)
(2565 the 0.510821 3 21 THOU 0.7228735 COME 0.8583255)
(3420 the 0.062272 1 244 THE 0.062272 AIN~T 0.5020493)
(2736 the 0.872897 1 1 THE 0.872897 THE 0.872897)
(2907 the 0.5485095 2 18 THIS 0.6900365 ICE 0.9056915)
(2907 the 0.348525 3 65 US 0.51483697 IT~S 0.8443765)
(5130 sun 0.97477055 1 1 SON 0.97477055 SON 0.97477055)
(3249 the 0.29002398 2 126 SPHERE 0.30418935 READ~V_PAST 0.8179465)
(0 worn 0 1 1525 nil 0 RIM 0.896011)
(513 the 0.8925515 1 2 THE 0.8925515 HIM 0.8942175)
(2565 the 0.478013 5 17 IT~S 0.55102897 ARE 0.711627)
(2394 the 0.8770805 1 1 THE 0.8770805 THE 0.8770805)
(2907 sun 0.7374893 3 7 SORRY 0.80078197 THE 0.89327097)
(2394 the 0.8465765 1 1 THE 0.8465765 THE 0.8465765)
(0 worn 0 1 1278 nil 0 ERA 0.79170704)
(3420 under 0.41614333 1 198 UNDER 0.41614333 RUN 0.8709607)
(855 the 0.85718 1 1 THE 0.85718 THE 0.85718)
(2736 the 0.0128285 3 508 THESE 0.12273934 TEA 0.92637706)
(0 unlimited 0 1 1900 nil 0 AT 0.957312)
(3249 the 0.508462 3 35 TELL 0.53018063 AIN~T 0.792299)
(3762 wall 0.1073705 15 493 DOLL 0.44354752 DOLL 0.89343655)
(3420 the 0.076607 5 138 TEA 0.305827 LIST 0.8943275)
(1881 the 0.039697 12 810 KNEE 0.573208 AN 0.977269)
(3249 task 0.31153068 12 307 TORE 0.6106235 SAW 0.90540004)
(2052 will 0.40695348 3 45 EAR 0.55227953 EAR 0.831273)
(0 this 0 1 565 nil 0 MUST 0.814269)
(2565 the 0.649042 1 1 THE 0.649042 THE 0.649042)
(0 the 0 1 427 nil 0 ALL 0.99274)
(171 the 0.355927 2 63 AM 0.50849754 BUY 0.999531)
(1026 the 0.494549 4 49 SPHERE 0.58674335 TRAY 0.91108453)
(3762 these 0.38062197 2 175 EAT 0.5328275 ARE 0.995955)
(0 than 0 1 339 nil 0 EASE 0.99983)
(2736 the 0.2910715 1 19 THE 0.2910715 BUY 0.746633)

(0 the 0 1 577 nil 0 OR 0.924294)
(3762 we 0.904988 1 1 WE 0.904988 WE 0.904988)
(3591 was 0.696374 1 5 WAS 0.696374 ALL 0.951223)
(2736 the 0.8847255 1 1 THE 0.8847255 THE 0.8847255)
(3591 was 0.648481 4 5 WISH 0.902261 WISH 0.902261)
(513 the 0.16047199 6 279 SOLID 0.350133 TEN 0.71072996)
(342 the 0.367831 1 40 THE 0.367831 RAY 0.8856255)
(0 treats 0 1 1507 nil 0 TEA 0.9799825)
(4104 wore 0.504042 5 72 WARM 0.66869336 ARM 0.930055)
(2736 the 0.3791865 1 23 THE 0.3791865 GUM 0.5295945)
(3078 than 0.060658 14 347 DECK 0.1210285 ROAM 0.6681755)
(5472 than 0.57458097 1 3 THAN 0.57458097 EYES 0.735682)
(3762 was 0.323644 1 37 WAS 0.323644 US 0.693246)
(0 the 0 1 573 nil 0 ARE 0.97406)
(2565 the 0.61632645 1 12 THE 0.61632645 ILL 0.972592)
(2565 the 0.516213 1 35 THE 0.516213 HE 0.90698755)
(3591 was 0.94465554 1 1 WAS 0.94465554 WAS 0.94465554)
(2736 the 0.8844875 1 1 THE 0.8844875 THE 0.8844875)
(2907 the 0.82390654 1 6 THE 0.82390654 ICE 0.940022)
(3249 the 0.7257505 1 5 THE 0.7257505 IT 0.7508135)
(5130 sun 0.986646 1 1 SON 0.986646 SON 0.986646)
(2736 the 0.22590101 2 157 THESE 0.254147 RIGHT 0.98105097)
(4275 worn 0.34513667 1 292 WARN 0.34513667 AM 0.997683)
(342 the 0.3665235 4 42 KEY 0.50267 IN 0.7698285)
(4104 the 0.0301695 7 266 IT 0.045665 LEAD 0.497779)
(3078 the 0.69218 1 5 THE 0.69218 BUY 0.92516)
(3078 the 0.057348497 11 362 SEA 0.503851 IS 0.998231)
(0 unlimited 0 1 1615 nil 0 TEA 0.9905995)
(3249 the 0.6444045 1 3 THE 0.6444045 ODD 0.6794715)
(5985 wall 0.45591798 4 132 ALL 0.768154 SAW 0.87883604)
(4788 the 0.5741235 1 24 THE 0.5741235 REST 0.884682)
(2565 the 0.4997175 3 57 THAN 0.95463455 MAN 0.9604145)
(0 the 0 1 574 nil 0 ALL 0.867901)
(171 the 0.317881 2 45 THUS 0.42208648 MY 0.958849)
(2907 the 0.19056699 1 143 THE 0.19056699 SINCE 0.696683)
(3420 these 0.110496506 9 310 OAK 0.3581985 ARE 0.95186245)
(684 than 0.26863867 3 114 KNEE 0.3464675 BE 0.952617)
(3249 the 0.1473715 4 193 AND 0.15726833 RAIN 0.585546)
(684 the 0.885381 1 1 THE 0.885381 THE 0.885381)
(2736 the 0.3114515 1 72 THE 0.3114515 CAMP 0.59570533)
(2736 than 0.24102099 10 180 DID 0.47575533 IN 0.799693)
(2907 was 0.4602595 1 16 WAS 0.4602595 TEA 0.7049305)
(3078 the 0.2591405 5 219 ENDED 0.37910974 AT 0.8729615)
(0 the 0 1 414 nil 0 TEA 0.9590955)
(1026 the 0.378219 1 84 THE 0.378219 HE 0.9451825)

(3249 was 0.5215455 1 4 WAS 0.5215455 IS 0.7262085)
(171 the 0.7184425 2 2 THESE 0.732487 THESE 0.732487)
(2565 the 0.2598435 1 56 THE 0.2598435 TRISH 0.585104)
(0 this 0 1 542 nil 0 IT 0.74026954)
(4617 the 0.27820697 1 82 THE 0.27820697 IT 0.6266525)
(171 the 0.591943 1 1 THE 0.591943 THE 0.591943)
(2394 the 0.97185445 1 1 THE 0.97185445 THE 0.97185445)
(2736 the 0.078334495 1 194 THE 0.078334495 ICE 0.917104)
(2394 the 0.8094075 1 1 THE 0.8094075 THE 0.8094075)
(4959 sun 0.889925 1 4 SON 0.889925 I~D 0.8919385)
(2394 the 0.518238 1 24 THE 0.518238 DISH 0.80822647)
(5130 worn 0.38571966 5 297 IRATE 0.632631 AID 0.951686)
(3078 the 0.570474 1 6 THE 0.570474 WITH 0.79149854)
(3249 the 0.3032795 1 45 THE 0.3032795 KNEE 0.741986)
(3591 the 0.6634635 1 5 THE 0.6634635 TEA 0.83302104)
(2565 the 0.5287935 1 14 THE 0.5287935 ARE 0.872629)
(3078 task 0.5400837 17 102 TOWN 0.8113575 OFF 0.9610625)
(2394 will 0.1946595 12 148 EAR 0.4307685 THE 0.75644195)
(0 this 0 1 737 nil 0 SEA 0.752483)
(2565 the 0.521795 1 13 THE 0.521795 WITH 0.7529865)
(171 the 0.614286 1 1 THE 0.614286 THE 0.614286)
(3249 was 0.5069605 2 10 OF 0.5078665 WE 0.981282)
(1197 the 0.397429 1 32 THE 0.397429 OF 0.720441)
(3762 the 0.444929 2 45 TRAY 0.6226835 RAY 0.949067)
(0 treats 0 1 2146 nil 0 TAN 0.644902)
(2565 the 0.6274515 1 1 THE 0.6274515 THE 0.6274515)
(2565 the 0.18962 4 128 EAR 0.526505 KNEE 0.772608)
(2565 the 0.873248 1 3 THE 0.873248 BUY 0.886979)
(3078 the 0.9262695 1 1 THE 0.9262695 THE 0.9262695)
(3420 the 0.598792 1 3 THE 0.598792 ILL 0.667543)
(2394 the 0.5439255 2 15 THESE 0.578362 ALL 0.996771)
(1368 will 0.4198297 5 62 ILL 0.715159 BE 0.925489)
(3249 the 0.748232 1 6 THE 0.748232 RIM 0.851109)
(2736 the 0.81524503 1 1 THE 0.81524503 THE 0.81524503)
(2565 the 0.787722 1 4 THE 0.787722 ANT 0.822792)
(2736 the 0.637925 1 6 THE 0.637925 EAR 0.770397)
(0 task 0 1 1445 nil 0 LAUGH 0.912956)
(2394 will 0.912969 3 3 EAR 0.9555435 EAR 0.9555435)
(0 this 0 1 528 nil 0 SEA 0.933863)
(2907 the 0.6822185 1 2 THE 0.6822185 TEA 0.6936805)
(2394 the 0.58695203 1 12 THE 0.58695203 CAR 0.943241)
(3591 was 0.3391335 6 206 ATE 0.47452098 IN 0.950606)
(0 the 0 1 601 nil 0 ILL 0.732722)
(2394 the 0.1526845 3 49 TIN 0.178903 RAY 0.5830445)
(0 treats 0 1 1549 nil 0 ATE 0.986179)

324

(0 under 0 1 1077 nil 0 LAY 0.764654)
(2736 the 0.3798635 2 9 THESE 0.38235402 HE 0.4852555)
(2736 the 0.599431 1 3 THE 0.599431 HIM 0.641549)
(2907 was 0.89277 1 3 WAS 0.89277 ILL 0.923845)
(2565 the 0.2499485 2 79 TEA 0.3125885 LID 0.945893)
(5301 wall 0.48219267 3 25 ALL 0.619242 OF 0.852644)
(2736 the 0.072713494 1 83 THE 0.072713494 IT~S 0.64876866)
(2565 the 0.713495 1 1 THE 0.713495 THE 0.713495)
(2394 the 0.0380145 16 237 WITH 0.568051 HE 0.67684)
(2907 the 0.044775 7 316 KNEE 0.458973 TEA 0.887658)
(2394 the 0.4737945 4 54 AND 0.6074353 US 0.86330354)
(3420 the 0.68855155 1 1 THE 0.68855155 THE 0.68855155)
(4788 was 0.49388748 1 42 WAS 0.49388748 US 0.957709)
(0 the 0 1 446 nil 0 ERRORS 0.815972)
(0 the 0 1 497 nil 0 IF 0.645151)
(2394 the 0.730102 1 2 THE 0.730102 OF 0.802235)
(2394 we 0.041088 18 342 THE 0.639168 OF 0.988577)
(171 the 0.7944405 1 1 THE 0.7944405 THE 0.7944405)
(2223 the 0.3192675 1 50 THE 0.3192675 HE 0.7575605)
(3249 the 0.037294 4 536 THESE 0.260927 WICK 0.7434525)
(2736 the 0.0903705 10 278 TEST 0.46350667 IS 0.85717297)
(2736 the 0.720841 1 1 THE 0.720841 THE 0.720841)
(2736 the 0.93147 1 1 THE 0.93147 THE 0.93147)
(3078 the 0.38100052 2 65 CEASE 0.411668 ALL 0.9821)
(3078 the 0.648998 1 2 THE 0.648998 ALL 0.866767)
(3420 the 0.076109 5 183 THESE 0.1524515 ROAST 0.541912)
(3762 these 0.0900105 2 304 MILK 0.10504267 ARE 0.931357)
(5814 than 0.13678633 9 427 KNEE 0.4808355 BE 0.994737)
(1881 the 0.65347254 1 2 THE 0.65347254 EVER 0.9357265)
(2565 the 0.54824 1 5 THE 0.54824 ARE 0.837643)
(2736 the 0.9329845 1 1 THE 0.9329845 THE 0.9329845)
(2394 the 0.1741075 6 382 THAN 0.668051 AN 0.994553)
(2223 the 0.6594045 1 1 THE 0.6594045 THE 0.6594045)
(3249 the 0.656476 1 2 THE 0.656476 IF 0.6960795)
(684 the 0.544517 1 1 THE 0.544517 THE 0.544517)
(342 the 0.3412185 4 51 RAKED 0.452368 TEA 0.688361)
(2565 the 0.587179 1 8 THE 0.587179 KNEE 0.8451785)
(342 will 0.3283435 2 28 ILL 0.4054775 BE 0.985708)
(3249 the 0.5566595 1 9 THE 0.5566595 NUMB 0.83983505)
(171 the 0.7084625 1 1 THE 0.7084625 THE 0.7084625)
(2907 the 0.595182 1 1 THE 0.595182 THE 0.595182)
(2736 the 0.7985785 1 1 THE 0.7985785 THE 0.7985785)
(4275 the 0.758478 2 3 THIS 0.90650046 THIS 0.90650046)
(2565 was 0.5711213 7 14 WE 0.927827 WE 0.927827)
(2736 the 0.4730235 1 7 THE 0.4730235 EAR 0.6674705)

(7353 wall 0.15435965 2 246 KNEE 0.2691325 RAY 0.990993)
(1026 the 0.2649295 1 136 THE 0.2649295 EAR 0.6789345)
(9405 than 0.41239634 1 84 CAN 0.41239634 SOME 0.744279)
(3933 was 0.85799754 1 1 WAS 0.85799754 WAS 0.85799754)
(3420 the 0.267525 1 59 THE 0.267525 OF 0.7039045)
(2394 the 0.66391253 1 10 THE 0.66391253 ARE 0.986791)
(171 the 0.49842697 6 25 WITH 0.631356 ILL 0.844844)
(3249 was 0.3979935 1 84 WAS 0.3979935 US 0.935705)
(2223 the 0.47616702 1 7 THE 0.47616702 DRESS 0.621867)
(0 wore 0 1 1035 nil 0 ARE 0.995171)
(2565 the 0.5496825 3 45 LESS 0.5513585 LAW 0.87691545)
(2907 the 0.7680865 1 1 THE 0.7680865 THE 0.7680865)
(342 this 0.68866146 2 15 THE 0.868927 EAR 0.90557647)
(342 the 0.51773846 1 4 THE 0.51773846 EAT 0.64194)
(3420 the 0.3489655 2 40 DIM 0.48127002 CAMP 0.61487037)
(171 the 0.674151 1 3 THE 0.674151 BUY 0.84176)
(0 the 0 1 391 nil 0 LAKE 0.6043825)
(2565 the 0.7273745 1 6 THE 0.7273745 BUY 0.994386)
(4617 the 0.284348 1 77 THE 0.284348 HE 0.978692)
(2565 the 0.4124895 4 50 EGG 0.46949401 HE 0.9799795)
(3420 the 0.4917135 4 45 THOU 0.5817265 AL 0.8604095)
(2736 the 0.4286735 3 9 US 0.5552965 US 0.5552965)
(2565 the 0.97063446 1 1 THE 0.97063446 THE 0.97063446)
(2565 the 0.033820502 23 342 TAN 0.654716 TAN 0.7684565)
(3078 the 0.48459 1 43 THE 0.48459 ROSE 0.88223004)
(513 the 0.89774346 1 2 THE 0.89774346 OF 0.954969)
(2565 the 0.739455 1 3 THE 0.739455 BUY 0.829665)
(4104 was 0.5651245 1 6 WAS 0.5651245 RUN 0.80427456)
(3078 the 0.4023545 2 89 DISH 0.602806 CASH 0.988222)
(3249 the 0.383938 2 46 LET 0.4291895 SAW 0.603505)
(2565 the 0.5155515 2 16 THEN 0.59611803 DOWN 0.63386565)
(3591 we 0.4164355 1 6 WE 0.4164355 AM 0.789955)
(0 under 0 1 1171 nil 0 KNEE 0.915064)
(171 the 0.53724897 1 1 THE 0.53724897 THE 0.53724897)
(513 the 0.3899035 5 54 WE 0.6852085 KEYED 0.9386215)
(2907 the 0.33946902 1 4 THE 0.33946902 GRIP 0.34845567)
(3078 the 0.658926 1 7 THE 0.658926 BUY 0.978749)
(3249 the 0.899697 1 1 THE 0.899697 THE 0.899697)
(3933 wore 0.618259 2 22 WARM 0.6944683 OR 0.950359)
(2565 the 0.87570655 1 1 THE 0.87570655 THE 0.87570655)
(513 the 0.066413 13 328 IN 0.476162 RUN 0.6687617)
(4275 this 0.48304 6 29 US 0.846438 US 0.846438)
(171 the 0.926408 1 1 THE 0.926408 THE 0.926408)
(2394 the 0.1291385 3 234 THAN 0.6124715 ANT 0.9857515)
(2907 the 0.026342 7 366 EASE 0.165407 NEED 0.6587335)

(171 the 0.57189 1 1 THE 0.57189 THE 0.57189)

(171 the 0.50918 1 9 THE 0.50918 KEY 0.917469)

(0 the 0 1 415 nil 0 SING 0.80050254)

(342 the 0.586075 3 18 THEN 0.695367 KEY 0.94819653)

(0 the 0 1 431 nil 0 SEA 0.794628)

(3420 the 0.70653546 1 1 THE 0.70653546 THE 0.70653546)

(4104 we 0.769319 1 1 WE 0.769319 WE 0.769319)

(3933 was 0.50662696 1 38 WAS 0.50662696 AN 0.949837)

(2394 the 0.8151225 1 1 THE 0.8151225 THE 0.8151225)

(0 wore 0 1 835 nil 0 CEASE 0.6690585)

(2394 the 0.780717 1 2 THE 0.780717 US 0.899222)

(2394 the 0.622519 1 3 THE 0.622519 PLAY 0.64398)

(171 this 0.61273754 1 13 THIS 0.61273754 IS 0.796198)

(2394 the 0.9222505 1 1 THE 0.9222505 THE 0.9222505)

(2565 the 0.391264 1 19 THE 0.391264 LIE 0.614771)

(2565 the 0.424284 1 1 THE 0.424284 THE 0.424284)

(3591 will 0.33069268 4 100 ILL 0.554969 BE 0.998243)

(3249 the 0.432243 1 25 THE 0.432243 KNEE 0.857263)

(342 the 0.575597 1 7 THE 0.575597 EAT 0.743773)

(2736 the 0.201754 1 108 THE 0.201754 ICE 0.5176735)

(2907 the 0.720897 1 4 THE 0.720897 OF 0.990163)

(2394 the 0.848517 1 1 THE 0.848517 THE 0.848517)

(2052 under 0.39828435 1 53 UNDER 0.39828435 IN 0.7187875)

(2736 the 0.946296 1 3 THE 0.946296 BUY 0.994639)

(2736 the 0.546872 1 3 THE 0.546872 WITH 0.689233)

(2565 the 0.1846725 1 150 THE 0.1846725 IF 0.682848)

(4104 this 0.286569 2 167 SIT 0.4298535 KISSED 0.7499335)

(4446 the 0.0431965 5 247 SWIMMING 0.14181349 TRICK 0.56302196)

(3078 the 0.5112615 1 6 THE 0.5112615 WITH 0.665482)

# APPENDIX H. FREQUENCY NORMALIZATION DATA

## A. DESCRIPTION

The following table lists data used to develop the frequency normalization method described in Chapter IV.

## B. DATA

| TIMIT Speech File | Sample Number | F1 Freq (Hz) | F2 Freq (Hz) | F3 Freq (Hz) | Pitch (Hz) |
|---|---|---|---|---|---|
| F:/TIMIT/TRAIN/DR2/FAEM0/SX132.WAV | 23,911 | 333 | 2,369 | 3,026 | 172 |
| F:/TIMIT/TRAIN/DR2/FAEM0/SX132.WAV | 24,167 | 349 | 2,361 | 2,856 | 117 |
| F:/TIMIT/TRAIN/DR2/FAJW0/SX183.WAV | 23,287 | 479 | 2,540 | 3,075 | 160 |
| F:/TIMIT/TRAIN/DR2/FAJW0/SX183.WAV | 23,543 | 503 | 2,726 | 3,164 | 162 |
| F:/TIMIT/TRAIN/DR2/FCAJ0/SX129.WAV | 33,179 | 495 | 2,523 | 3,124 | 158 |
| F:/TIMIT/TRAIN/DR2/FCAJ0/SX129.WAV | 33,435 | 471 | 2,588 | 3,172 | 158 |
| F:/TIMIT/TRAIN/DR2/FCAJ0/SX129.WAV | 33,947 | 479 | 2,596 | 2,986 | 154 |
| F:/TIMIT/TRAIN/DR2/FCMM0/SX183.WAV | 13,497 | 568 | 2,418 | 3,181 | 190 |
| F:/TIMIT/TRAIN/DR2/FCMM0/SX183.WAV | 13,753 | 519 | 2,475 | 3,043 | 190 |
| F:/TIMIT/TRAIN/DR2/FCYL0/SX127.WAV | 17,323 | 300 | 2,548 | 2,937 | 198 |
| F:/TIMIT/TRAIN/DR2/FCYL0/SX127.WAV | 18,091 | 365 | 2,767 | 3,140 | 205 |
| F:/TIMIT/TRAIN/DR2/FDAS1/SX21.WAV | 21,065 | 446 | 2,207 | 2,872 | 195 |
| F:/TIMIT/TRAIN/DR2/FDAS1/SX21.WAV | 21,321 | 381 | 2,345 | 3,002 | 182 |
| F:/TIMIT/TRAIN/DR2/FDAS1/SX21.WAV | 21,577 | 406 | 2,426 | 3,034 | 178 |
| F:/TIMIT/TRAIN/DR2/FDAS1/SX21.WAV | 22,089 | 406 | 2,491 | 2,848 | 184 |
| F:/TIMIT/TRAIN/DR2/FDNC0/SX18.WAV | 12,455 | 519 | 2,402 | 3,043 | 193 |
| F:/TIMIT/TRAIN/DR2/FDXW0/SX161.WAV | 16,961 | 373 | 2,548 | 3,083 | 178 |
| F:/TIMIT/TRAIN/DR2/FDXW0/SX161.WAV | 17,217 | 422 | 2,475 | 3,018 | 178 |
| F:/TIMIT/TRAIN/DR2/FDXW0/SX161.WAV | 17,473 | 511 | 2,442 | 3,002 | 178 |
| F:/TIMIT/TRAIN/DR2/FEAC0/SX255.WAV | 36,743 | 389 | 2,199 | 2,718 | 152 |
| F:/TIMIT/TRAIN/DR2/FHLM0/SX120.WAV | 38,551 | 365 | 2,791 | 3,383 | 198 |
| F:/TIMIT/TRAIN/DR2/FHLM0/SX120.WAV | 38,807 | 389 | 2,799 | 3,083 | 200 |
| F:/TIMIT/TRAIN/DR2/FHLM0/SX120.WAV | 39,319 | 414 | 2,580 | 3,262 | 200 |
| F:/TIMIT/TRAIN/DR2/FHLM0/SX120.WAV | 39,575 | 406 | 2,604 | 3,132 | 195 |
| F:/TIMIT/TRAIN/DR2/FHLM0/SX120.WAV | 39,831 | 406 | 2,645 | 3,099 | 190 |
| F:/TIMIT/TRAIN/DR2/FKAA0/SX128.WAV | 17,335 | 454 | 2,320 | 2,929 | 184 |
| F:/TIMIT/TRAIN/DR2/FKAA0/SX128.WAV | 17,847 | 414 | 2,385 | 3,132 | 184 |

| | | | | |
|---|---|---|---|---|
| F:/TIMIT/TRAIN/DR2/FKAA0/SX128.WAV | 18,103 | 341 | 2,361 | 3,156 | 182 |
| F:/TIMIT/TRAIN/DR2/FLMA0/SX253.WAV | 5,811 | 365 | 2,507 | 3,018 | 239 |
| F:/TIMIT/TRAIN/DR2/FLMA0/SX253.WAV | 6,067 | 487 | 2,556 | 3,043 | 235 |
| F:/TIMIT/TRAIN/DR2/FLMA0/SX253.WAV | 6,323 | 479 | 2,718 | 3,083 | 232 |
| F:/TIMIT/TRAIN/DR2/FMJB0/SX187.WAV | 24,783 | 763 | 2,450 | 3,010 | 180 |
| F:/TIMIT/TRAIN/DR2/FMJB0/SX187.WAV | 25,039 | 771 | 2,540 | 3,034 | 180 |
| F:/TIMIT/TRAIN/DR2/FMJB0/SX187.WAV | 25,807 | 260 | 2,540 | 3,043 | 184 |
| F:/TIMIT/TRAIN/DR2/FMKF0/SX276.WAV | 16,302 | 414 | 2,426 | 2,970 | 193 |
| F:/TIMIT/TRAIN/DR2/FMKF0/SX276.WAV | 16,558 | 365 | 2,418 | 2,913 | 193 |
| F:/TIMIT/TRAIN/DR2/FMMH0/SX187.WAV | 20,961 | 333 | 2,734 | 3,286 | 178 |
| F:/TIMIT/TRAIN/DR2/FMMH0/SX187.WAV | 21,217 | 381 | 2,702 | 3,043 | 176 |
| F:/TIMIT/TRAIN/DR2/FPJF0/SX236.WAV | 31,758 | 462 | 2,426 | 3,132 | 186 |
| F:/TIMIT/TRAIN/DR2/FPJF0/SX236.WAV | 32,014 | 503 | 2,580 | 3,018 | 184 |
| F:/TIMIT/TRAIN/DR2/FRLL0/SX164.WAV | 26,551 | 438 | 2,329 | 2,986 | 213 |
| F:/TIMIT/TRAIN/DR2/FRLL0/SX164.WAV | 27,063 | 381 | 2,507 | 2,864 | 203 |
| F:/TIMIT/TRAIN/DR2/FRLL0/SX164.WAV | 27,575 | 454 | 2,467 | 2,775 | 208 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 69,415 | 495 | 2,101 | 3,010 | 186 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 69,671 | 479 | 2,199 | 2,856 | 178 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 69,927 | 495 | 2,207 | 2,921 | 172 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 70,183 | 487 | 2,256 | 2,986 | 172 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 70,439 | 495 | 2,280 | 2,986 | 170 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 71,463 | 438 | 2,442 | 2,945 | 168 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 71,719 | 438 | 2,499 | 2,929 | 167 |
| F:/TIMIT/TRAIN/DR2/FSCN0/SX176.WAV | 71,975 | 438 | 2,450 | 2,937 | 167 |
| F:/TIMIT/TRAIN/DR2/FSKL0/SX179.WAV | 39,738 | 438 | 2,012 | 2,385 | 142 |
| F:/TIMIT/TRAIN/DR2/FSKL0/SX179.WAV | 40,250 | 349 | 2,199 | 2,686 | 119 |
| F:/TIMIT/TRAIN/DR2/FSKL0/SX179.WAV | 40,506 | 341 | 2,166 | 2,621 | 122 |
| F:/TIMIT/TRAIN/DR2/FSRH0/SX221.WAV | 17,597 | 487 | 2,320 | 3,099 | 262 |
| F:/TIMIT/TRAIN/DR2/FTMG0/SX272.WAV | 10,147 | 414 | 2,621 | 3,205 | 211 |
| F:/TIMIT/TRAIN/DR2/FTMG0/SX272.WAV | 11,427 | 422 | 2,564 | 2,978 | 229 |
| F:/TIMIT/TRAIN/DR2/MARC0/SX18.WAV | 9,362 | 422 | 1,955 | 2,410 | 133 |
| F:/TIMIT/TRAIN/DR2/MARC0/SX18.WAV | 9,618 | 341 | 2,037 | 2,572 | 132 |
| F:/TIMIT/TRAIN/DR2/MARC0/SX18.WAV | 9,874 | 325 | 2,118 | 2,572 | 129 |
| F:/TIMIT/TRAIN/DR2/MBJV0/SX167.WAV | 38,187 | 316 | 1,663 | 3,010 | 101 |
| F:/TIMIT/TRAIN/DR2/MBJV0/SX167.WAV | 38,443 | 325 | 1,769 | 3,034 | 102 |
| F:/TIMIT/TRAIN/DR2/MBJV0/SX167.WAV | 38,699 | 341 | 1,761 | 3,051 | 103 |
| F:/TIMIT/TRAIN/DR2/MBJV0/SX167.WAV | 38,955 | 438 | 1,615 | 3,059 | 102 |
| F:/TIMIT/TRAIN/DR2/MBJV0/SX167.WAV | 39,211 | 511 | 1,469 | 3,026 | 101 |
| F:/TIMIT/TRAIN/DR2/MCEW0/SX272.WAV | 8,192 | 430 | 1,631 | 2,345 | 178 |

| | | | | | |
|---|---|---|---|---|---|
| F:/TIMIT/TRAIN/DR2/MCEW0/SX272.WAV | 8,704 | 308 | 2,296 | 3,440 | 190 |
| F:/TIMIT/TRAIN/DR2/MCEW0/SX272.WAV | 9,216 | 268 | 2,288 | 3,497 | 203 |
| F:/TIMIT/TRAIN/DR2/MCEW0/SX272.WAV | 9,472 | 243 | 2,223 | 3,497 | 195 |
| F:/TIMIT/TRAIN/DR2/MDBP0/SX258.WAV | 45,484 | 300 | 1,826 | 2,580 | 105 |
| F:/TIMIT/TRAIN/DR2/MDBP0/SX258.WAV | 46,252 | 260 | 2,085 | 2,621 | 93 |
| F:/TIMIT/TRAIN/DR2/MDBP0/SX258.WAV | 46,508 | 316 | 2,045 | 2,629 | 101 |
| F:/TIMIT/TRAIN/DR2/MDEM0/SX248.WAV | 6,398 | 292 | 2,045 | 2,572 | 203 |
| F:/TIMIT/TRAIN/DR2/MDEM0/SX248.WAV | 6,654 | 381 | 2,101 | 2,491 | 198 |
| F:/TIMIT/TRAIN/DR2/MDLB0/SX136.WAV | 62,057 | 519 | 2,353 | 2,775 | 105 |
| F:/TIMIT/TRAIN/DR2/MDLB0/SX136.WAV | 62,569 | 503 | 2,418 | 2,686 | 95 |
| F:/TIMIT/TRAIN/DR2/MDLC2/SX174.WAV | 35,635 | 316 | 2,045 | 2,450 | 95 |
| F:/TIMIT/TRAIN/DR2/MDLC2/SX174.WAV | 36,147 | 308 | 1,801 | 2,215 | 91 |
| F:/TIMIT/TRAIN/DR2/MDMT0/SX122.WAV | 13,575 | 325 | 2,223 | 2,759 | 125 |
| F:/TIMIT/TRAIN/DR2/MDMT0/SX122.WAV | 13,831 | 268 | 2,215 | 2,864 | 120 |
| F:/TIMIT/TRAIN/DR2/MDMT0/SX122.WAV | 14,087 | 268 | 2,191 | 2,815 | 116 |
| F:/TIMIT/TRAIN/DR2/MDMT0/SX122.WAV | 14,343 | 292 | 2,166 | 2,726 | 114 |
| F:/TIMIT/TRAIN/DR2/MDPS0/SX179.WAV | 41,994 | 292 | 1,858 | 2,515 | 69 |
| F:/TIMIT/TRAIN/DR2/MDPS0/SX179.WAV | 42,250 | 284 | 1,850 | 2,385 | 74 |
| F:/TIMIT/TRAIN/DR2/MDPS0/SX179.WAV | 42,506 | 252 | 1,842 | 2,320 | 78 |
| F:/TIMIT/TRAIN/DR2/MDSS0/SX261.WAV | 22,443 | 292 | 2,037 | 2,824 | 119 |
| F:/TIMIT/TRAIN/DR2/MDWD0/SX450.WAV | 19,936 | 398 | 1,874 | 3,270 | 133 |
| F:/TIMIT/TRAIN/DR2/MDWD0/SX450.WAV | 20,448 | 446 | 1,931 | 2,556 | 127 |
| F:/TIMIT/TRAIN/DR2/MEFG0/SX105.WAV | 23,069 | 462 | 1,477 | 2,377 | 96 |
| F:/TIMIT/TRAIN/DR2/MEFG0/SX105.WAV | 23,325 | 430 | 1,655 | 2,385 | 96 |
| F:/TIMIT/TRAIN/DR2/MEFG0/SX105.WAV | 23,581 | 300 | 1,793 | 2,304 | 96 |
| F:/TIMIT/TRAIN/DR2/MEFG0/SX105.WAV | 23,837 | 308 | 1,882 | 2,377 | 97 |
| F:/TIMIT/TRAIN/DR2/MEFG0/SX105.WAV | 24,093 | 292 | 1,931 | 2,369 | 94 |
| F:/TIMIT/TRAIN/DR2/MHRM0/SX148.WAV | 8,634 | 260 | 2,239 | 2,580 | 168 |
| F:/TIMIT/TRAIN/DR2/MJAE0/SX174.WAV | 40,071 | 446 | 1,817 | 3,116 | 120 |
| F:/TIMIT/TRAIN/DR2/MJAE0/SX174.WAV | 40,327 | 422 | 1,834 | 3,124 | 122 |
| F:/TIMIT/TRAIN/DR2/MJBG0/SX152.WAV | 6,855 | 284 | 2,166 | 2,921 | 150 |
| F:/TIMIT/TRAIN/DR2/MJBG0/SX152.WAV | 7,111 | 284 | 2,183 | 2,937 | 158 |
| F:/TIMIT/TRAIN/DR2/MJBG0/SX152.WAV | 7,367 | 300 | 2,191 | 2,953 | 162 |
| F:/TIMIT/TRAIN/DR2/MJBG0/SX152.WAV | 7,623 | 300 | 2,191 | 2,767 | 163 |
| F:/TIMIT/TRAIN/DR2/MJDE0/SX130.WAV | 3,351 | 252 | 1,939 | 2,856 | 140 |
| F:/TIMIT/TRAIN/DR2/MJDE0/SX130.WAV | 3,607 | 252 | 1,972 | 2,832 | 144 |
| F:/TIMIT/TRAIN/DR2/MJDE0/SX130.WAV | 3,863 | 276 | 1,972 | 2,864 | 150 |
| F:/TIMIT/TRAIN/DR2/MJDE0/SX130.WAV | 4,375 | 284 | 2,020 | 2,791 | 160 |
| F:/TIMIT/TRAIN/DR2/MJDE0/SX130.WAV | 4,631 | 316 | 1,963 | 2,759 | 163 |

| | | | | |
|---|---|---|---|---|
| F:/TIMIT/TRAIN/DR2/MJDE0/SX130.WAV | 4,887 | 333 | 2,020 | 2,718 | 160 |
| F:/TIMIT/TRAIN/DR2/MJEB0/SX170.WAV | 40,797 | 325 | 2,045 | 2,369 | 128 |
| F:/TIMIT/TRAIN/DR2/MJHI0/SX248.WAV | 4,897 | 292 | 2,045 | 2,442 | 111 |
| F:/TIMIT/TRAIN/DR2/MJMA0/SX145.WAV | 17,935 | 406 | 1,923 | 2,588 | 113 |
| F:/TIMIT/TRAIN/DR2/MJMA0/SX145.WAV | 18,191 | 398 | 2,020 | 2,653 | 114 |
| F:/TIMIT/TRAIN/DR2/MJMA0/SX145.WAV | 18,447 | 308 | 2,101 | 2,783 | 114 |
| F:/TIMIT/TRAIN/DR2/MJMA0/SX145.WAV | 18,703 | 300 | 2,174 | 2,702 | 115 |
| F:/TIMIT/TRAIN/DR2/MJMA0/SX145.WAV | 18,959 | 292 | 2,272 | 2,775 | 114 |
| F:/TIMIT/TRAIN/DR2/MJMD0/SX128.WAV | 17,489 | 292 | 2,077 | 2,442 | 110 |
| F:/TIMIT/TRAIN/DR2/MJMD0/SX128.WAV | 17,745 | 284 | 2,158 | 2,499 | 107 |
| F:/TIMIT/TRAIN/DR2/MJMD0/SX128.WAV | 18,001 | 284 | 2,126 | 2,507 | 106 |
| F:/TIMIT/TRAIN/DR2/MJPM0/SX18.WAV | 9,796 | 462 | 1,874 | 2,207 | 123 |
| F:/TIMIT/TRAIN/DR2/MJPM0/SX18.WAV | 10,564 | 487 | 1,874 | 2,215 | 117 |
| F:/TIMIT/TRAIN/DR2/MJRP0/SX135.WAV | 25,860 | 284 | 2,093 | 2,588 | 163 |
| F:/TIMIT/TRAIN/DR2/MJRP0/SX135.WAV | 26,116 | 276 | 2,101 | 2,734 | 165 |
| F:/TIMIT/TRAIN/DR2/MKAH0/SX178.WAV | 44,599 | 243 | 2,101 | 2,686 | 119 |
| F:/TIMIT/TRAIN/DR2/MKAJ0/SX154.WAV | 17,815 | 503 | 1,817 | 2,613 | 104 |
| F:/TIMIT/TRAIN/DR2/MKDT0/SX173.WAV | 47,535 | 260 | 2,004 | 2,458 | 113 |
| F:/TIMIT/TRAIN/DR2/MKDT0/SX173.WAV | 47,791 | 252 | 2,061 | 2,523 | 112 |
| F:/TIMIT/TRAIN/DR2/MKDT0/SX173.WAV | 48,047 | 284 | 2,028 | 2,556 | 110 |
| F:/TIMIT/TRAIN/DR2/MKDT0/SX173.WAV | 48,303 | 268 | 1,955 | 2,394 | 110 |
| F:/TIMIT/TRAIN/DR2/MKJO0/SX167.WAV | 32,469 | 260 | 2,012 | 2,385 | 116 |
| F:/TIMIT/TRAIN/DR2/MKJO0/SX167.WAV | 32,725 | 243 | 2,053 | 2,467 | 115 |
| F:/TIMIT/TRAIN/DR2/MKJO0/SX167.WAV | 32,981 | 252 | 2,061 | 2,540 | 112 |
| F:/TIMIT/TRAIN/DR2/MKJO0/SX167.WAV | 33,237 | 252 | 1,996 | 2,621 | 114 |
| F:/TIMIT/TRAIN/DR2/MMAG0/SX136.WAV | 57,570 | 430 | 2,215 | 2,613 | 93 |
| F:/TIMIT/TRAIN/DR2/MMAG0/SX136.WAV | 57,826 | 438 | 2,264 | 2,645 | 90 |
| F:/TIMIT/TRAIN/DR2/MMDS0/SX263.WAV | 10,155 | 308 | 1,696 | 2,402 | 130 |
| F:/TIMIT/TRAIN/DR2/MMDS0/SX263.WAV | 10,411 | 308 | 1,777 | 2,353 | 128 |
| F:/TIMIT/TRAIN/DR2/MMDS0/SX263.WAV | 10,667 | 316 | 1,866 | 2,272 | 128 |
| F:/TIMIT/TRAIN/DR2/MMGK0/SX152.WAV | 5,231 | 308 | 2,012 | 2,807 | 151 |
| F:/TIMIT/TRAIN/DR2/MMGK0/SX152.WAV | 5,487 | 325 | 2,028 | 2,556 | 150 |
| F:/TIMIT/TRAIN/DR2/MMGK0/SX152.WAV | 5,743 | 316 | 1,842 | 2,556 | 144 |
| F:/TIMIT/TRAIN/DR2/MMGK0/SX152.WAV | 5,999 | 300 | 1,834 | 2,450 | 145 |
| F:/TIMIT/TRAIN/DR2/MMGK0/SX152.WAV | 6,255 | 381 | 1,761 | 2,434 | 151 |
| F:/TIMIT/TRAIN/DR2/MMXS0/SX336.WAV | 48,172 | 373 | 1,915 | 2,231 | 144 |
| F:/TIMIT/TRAIN/DR2/MMXS0/SX336.WAV | 48,428 | 365 | 1,761 | 2,085 | 144 |
| F:/TIMIT/TRAIN/DR2/MPPC0/SX152.WAV | 6,547 | 316 | 2,337 | 2,702 | 172 |
| F:/TIMIT/TRAIN/DR2/MPPC0/SX152.WAV | 6,803 | 349 | 2,247 | 2,718 | 176 |

| | | | | |
|---|---|---|---|---|
| F:/TIMIT/TRAIN/DR2/MPRB0/SX125.WAV | 27,706 | 454 | 2,304 | 2,994 | 99 |
| F:/TIMIT/TRAIN/DR2/MPRB0/SX125.WAV | 27,962 | 430 | 2,426 | 2,953 | 100 |
| F:/TIMIT/TRAIN/DR2/MPRB0/SX125.WAV | 28,218 | 414 | 2,499 | 3,010 | 98 |
| F:/TIMIT/TRAIN/DR2/MRAB0/SX144.WAV | 9,581 | 349 | 1,460 | 2,264 | 133 |
| F:/TIMIT/TRAIN/DR2/MRFK0/SX176.WAV | 48,852 | 300 | 1,882 | 2,548 | 96 |
| F:/TIMIT/TRAIN/DR2/MRFK0/SX176.WAV | 49,108 | 300 | 1,907 | 2,507 | 100 |
| F:/TIMIT/TRAIN/DR2/MRFK0/SX176.WAV | 49,364 | 292 | 1,963 | 2,499 | 99 |
| F:/TIMIT/TRAIN/DR2/MRFK0/SX176.WAV | 49,620 | 276 | 1,972 | 2,499 | 94 |
| F:/TIMIT/TRAIN/DR2/MRFK0/SX176.WAV | 49,876 | 276 | 2,004 | 2,475 | 92 |
| F:/TIMIT/TRAIN/DR2/MRFK0/SX176.WAV | 50,132 | 284 | 1,996 | 2,450 | 87 |
| F:/TIMIT/TRAIN/DR2/MRGS0/SX276.WAV | 20,098 | 316 | 1,955 | 2,653 | 122 |
| F:/TIMIT/TRAIN/DR2/MRGS0/SX276.WAV | 20,354 | 316 | 2,028 | 2,694 | 123 |
| F:/TIMIT/TRAIN/DR2/MRGS0/SX276.WAV | 20,610 | 300 | 2,061 | 2,694 | 123 |
| F:/TIMIT/TRAIN/DR2/MRGS0/SX276.WAV | 20,866 | 292 | 2,037 | 2,483 | 122 |
| F:/TIMIT/TRAIN/DR2/MRHL0/SX165.WAV | 12,625 | 446 | 1,501 | 2,596 | 150 |
| F:/TIMIT/TRAIN/DR2/MRHL0/SX165.WAV | 12,881 | 422 | 2,101 | 2,604 | 150 |
| F:/TIMIT/TRAIN/DR2/MRJH0/SX307.WAV | 6,615 | 365 | 2,345 | 2,775 | 150 |
| F:/TIMIT/TRAIN/DR2/MRJH0/SX307.WAV | 6,871 | 316 | 2,353 | 2,742 | 138 |
| F:/TIMIT/TRAIN/DR2/MRJH0/SX307.WAV | 7,127 | 316 | 2,361 | 2,751 | 128 |
| F:/TIMIT/TRAIN/DR2/MRJH0/SX307.WAV | 7,639 | 284 | 2,377 | 2,791 | 115 |
| F:/TIMIT/TRAIN/DR2/MRJM0/SX148.WAV | 10,407 | 284 | 2,110 | 2,467 | 155 |
| F:/TIMIT/TRAIN/DR2/MRJM0/SX148.WAV | 10,663 | 316 | 2,174 | 2,613 | 147 |
| F:/TIMIT/TRAIN/DR2/MRJM1/SX128.WAV | 17,671 | 300 | 2,045 | 2,320 | 126 |
| F:/TIMIT/TRAIN/DR2/MRJM1/SX128.WAV | 17,927 | 276 | 2,053 | 2,434 | 123 |
| F:/TIMIT/TRAIN/DR2/MRJM1/SX128.WAV | 18,183 | 276 | 1,963 | 2,475 | 120 |
| F:/TIMIT/TRAIN/DR2/MRJM1/SX128.WAV | 18,439 | 292 | 1,882 | 2,515 | 114 |
| F:/TIMIT/TRAIN/DR2/MRJT0/SX148.WAV | 9,401 | 325 | 1,785 | 2,134 | 97 |
| F:/TIMIT/TRAIN/DR2/MRJT0/SX148.WAV | 9,657 | 300 | 1,826 | 2,215 | 96 |
| F:/TIMIT/TRAIN/DR2/MRLJ0/SX160.WAV | 40,543 | 276 | 2,304 | 2,783 | 92 |
| F:/TIMIT/TRAIN/DR2/MRLJ0/SX160.WAV | 40,799 | 276 | 2,288 | 2,783 | 92 |
| F:/TIMIT/TRAIN/DR2/MRLJ0/SX160.WAV | 41,055 | 284 | 2,215 | 2,742 | 92 |
| F:/TIMIT/TRAIN/DR2/MRLR0/SX206.WAV | 7,415 | 325 | 2,012 | 2,653 | 147 |
| F:/TIMIT/TRAIN/DR2/MRLR0/SX206.WAV | 7,671 | 414 | 1,834 | 2,645 | 148 |
| F:/TIMIT/TRAIN/DR2/MRLR0/SX206.WAV | 7,927 | 479 | 1,436 | 2,629 | 150 |
| F:/TIMIT/TRAIN/DR2/MRMS0/SX120.WAV | 25,325 | 292 | 2,150 | 2,418 | 110 |
| F:/TIMIT/TRAIN/DR2/MSAT0/SX176.WAV | 41,655 | 308 | 1,777 | 2,531 | 92 |
| F:/TIMIT/TRAIN/DR2/MSAT0/SX176.WAV | 41,911 | 316 | 1,793 | 2,531 | 87 |
| F:/TIMIT/TRAIN/DR2/MSAT0/SX176.WAV | 42,167 | 308 | 1,817 | 2,548 | 86 |
| F:/TIMIT/TRAIN/DR2/MSAT0/SX176.WAV | 42,423 | 284 | 1,842 | 2,523 | 85 |

| | | | | |
|---|---|---|---|---|
| F:/TIMIT/TRAIN/DR2/MSAT0/SX176.WAV | 42,679 | 292 | 1,850 | 2,515 | 81 |
| F:/TIMIT/TRAIN/DR2/MTAT1/SX149.WAV | 55,655 | 430 | 1,963 | 2,312 | 118 |
| F:/TIMIT/TRAIN/DR2/MTAT1/SX149.WAV | 55,911 | 325 | 1,980 | 2,613 | 118 |
| F:/TIMIT/TRAIN/DR2/MTAT1/SX149.WAV | 56,167 | 325 | 2,004 | 2,629 | 116 |
| F:/TIMIT/TRAIN/DR2/MTAT1/SX149.WAV | 56,423 | 333 | 1,955 | 2,531 | 114 |
| F:/TIMIT/TRAIN/DR2/MTBC0/SX183.WAV | 15,252 | 487 | 2,045 | 2,661 | 110 |
| F:/TIMIT/TRAIN/DR2/MTBC0/SX183.WAV | 15,508 | 479 | 2,126 | 2,669 | 110 |
| F:/TIMIT/TRAIN/DR2/MTBC0/SX183.WAV | 15,764 | 479 | 2,191 | 2,621 | 109 |
| F:/TIMIT/TRAIN/DR2/MTBC0/SX183.WAV | 16,276 | 462 | 2,150 | 2,499 | 107 |
| F:/TIMIT/TRAIN/DR2/MTBC0/SX183.WAV | 16,532 | 471 | 2,053 | 2,434 | 101 |
| F:/TIMIT/TRAIN/DR2/MTBC0/SX183.WAV | 16,788 | 495 | 1,817 | 2,361 | 99 |
| F:/TIMIT/TRAIN/DR2/MTDB0/SX321.WAV | 44,565 | 398 | 1,744 | 2,442 | 100 |
| F:/TIMIT/TRAIN/DR2/MTDB0/SX321.WAV | 44,821 | 333 | 1,874 | 2,353 | 97 |
| F:/TIMIT/TRAIN/DR2/MTDB0/SX321.WAV | 45,077 | 373 | 1,834 | 2,385 | 98 |
| F:/TIMIT/TRAIN/DR2/MTDB0/SX321.WAV | 45,333 | 430 | 1,761 | 2,385 | 98 |
| F:/TIMIT/TRAIN/DR2/MTJG0/SX170.WAV | 10,855 | 292 | 2,061 | 2,669 | 148 |
| F:/TIMIT/TRAIN/DR2/MTJG0/SX170.WAV | 11,111 | 260 | 2,077 | 2,637 | 150 |
| F:/TIMIT/TRAIN/DR2/MTJG0/SX170.WAV | 11,367 | 316 | 2,045 | 2,645 | 148 |
| F:/TIMIT/TRAIN/DR2/MTJG0/SX170.WAV | 11,623 | 325 | 1,939 | 2,548 | 147 |
| F:/TIMIT/TRAIN/DR2/MWSB0/SX276.WAV | 14,088 | 292 | 1,712 | 3,108 | 94 |
| F:/TIMIT/TRAIN/DR2/MZMB0/SX176.WAV | 50,855 | 300 | 2,061 | 2,856 | 103 |
| F:/TIMIT/TRAIN/DR2/MZMB0/SX176.WAV | 51,111 | 292 | 2,093 | 2,897 | 102 |
| F:/TIMIT/TRAIN/DR2/MZMB0/SX176.WAV | 51,367 | 292 | 2,101 | 2,840 | 95 |
| F:/TIMIT/TRAIN/DR2/MZMB0/SX176.WAV | 51,623 | 292 | 2,053 | 2,815 | 90 |
| F:/TIMIT/TRAIN/DR2/MZMB0/SX176.WAV | 51,879 | 292 | 2,110 | 2,832 | 88 |
| F:/TIMIT/TRAIN/DR2/MZMB0/SX176.WAV | 52,391 | 284 | 2,077 | 2,767 | 97 |

# LIST OF REFERENCES

1. Deller, J. R., J. G. Proakis, and J. H. L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan Publishing Company, 1993.

2. Garafolo, J. S., L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, *DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus CD-ROM*, U. S. Department of Commerce, National Institute of Standards and Technology, 1993.

3. Parsons, T. W., *Voice and Speech Processing*, McGraw-Hill Book Company, 1987.

4. Thompson, R. F., *The Brain, an Introduction to Neuroscience*, W. H. Freeman and Company, 1985.

5. Sivakumar, S. C., and W. Robertson, "Enhancing Neural Network Based Consonant Recognition Using Formant Transitions in Neighboring Vowels," from Communications on the Move, ICCS/ISITA, 1992.

6. Kent, E. W., *The Brains of Men and Machines*, BYTE/McGraw Hill, 1981.

7. Ainsworth, W. A., *Speech Recognition by Machine*, Peter Peregrinus Ltd., 1988.

8. Witbrock, M., and P. Haffner, "Rapid Connectionist Speaker Adaption," IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.

9. Demuth, H., and M. Beale, *Neural Network Toolbox User's Guide*, The Math Works Inc., 1994.

10. Brown, D. W., *SPC Toolbox*, Naval Postgraduate School, 1993.

11. Jain, A. K., and J. Mao, "Artificial Neural Networks: a Tutorial," IEEE *Computer*, March 1996.

12. Wasserman, P. D., *Neural Computing, Theory and Practice*, Van Nostrand Reinhold, 1989.

13. Haykin, S., *Neural Networks, a Comprehensive Foundation*, Macmillan College Publishing Company, 1994.

14. Dayhoff, J., *Neural Network Architectures*, Van Nostrand Reinhold, 1990.

15. Morgan, N., and H. Bourlard, "Continuous Speech Recognition," IEEE Signal Processing Magazine, May 1995.

16. Waibel, A., and A. Hirai, Phoneme-based Word Recognition by Neural Network - a Step Toward Large Vocabulary Recognition," IJCNN International Joint Conference on Neural Networks, 1990.

17. Wendell, B. S. and K. Abdelhamied, "A Phoneme Recognition System Using Modular Construction of Time-Delay Neural Networks," Proceedings, Fifth Annual IEEE Symposium on Computer-Based Medical Systems, 1992.

18. Matsuura, Y., H. Miyazawa, and T. E. Skinner, "Word Recognition Using a Neural Network and a Phonetically Based DTW," Neural Networks for Dignal Processing IV, Proceedings of the 1994 IEEE Workshop, 1994.

19. Iso, K., and T. Watanabe, "Speech Recognition Using Demi-Syllable Neural Prediction Model," *Advances in Neural Information Processing Systems , Volume 3*, Morgan Kaufmann Publishers, 1991.

20. Dalsgaard, P., O. Anderson, and R. Jorgensen, "On the Identification of Phonemes Using Acoustic-Phonetic Features Derived by a Self-Organizing Neural Network," Neural Networks for Signal Processing II, Proceedings of the IEEE-SP Workshop, 1992.

21. Clayman, B., ed. *The American Medical Association Encyclopedia of Medicine*, Random House, 1989.

22. Zwicker, E., and E. Terhardt, "Analytical Expressions for Critical-Band Rate and Critical Bandwidth as a Function of Frequency," The Journal of the Acoustical Society of America, Vol. 68, No. 4, October 1980.

23. Kamm, C. A., and S. Singhal, "Effect of Neural Network Input Span on Phoneme Classification," IJCNN International Joint Conference on Neural Networks, IEEE, 1990.

24. Zue, V. W., and S. Seneff, "Transcription and Alignment of the TIMIT Database," The Second Symposium on Advanced Man-Machine Interface through Spoken Language, 1988.

25. Hornik, K., "Approximation Capabilities of Multi-layer Feedforward Networks," *Neural Networks,* Vol. 4, 1991.

26. Abe, S., M. Kayama, H. Takenaga, and T. Kitamura, "Extracting Algorithms from Pattern Classification Neural Networks," *Neural Networks*, Vol. 6, 1993.

27. Caudill, M., "Neural Network Training Tips and Techniques," AI Expert, January 1991.

28. Sietsma, J., and R. J. F. Dow, "Creating Artificial Neural Networks that Generalize," *Neural Networks*, Vol. 4, 1991.

29. Hamming, R. W., EC-4000 Lecture Notes, Naval Postgraduate School, 1996.

30. Basu, A., and T. Svendsen, "A Time-Frequency Segmental Neural Network for Phoneme Recognition," ICASSP-93, 1993 IEEE International Conference on Acoustics, Speech and Signal Processing, 1993.

# INITIAL DISTRIBUTION LIST

1.  Defense Technical Information Center      2
    8725 John J. Kingman Rd, STE 0944
    Ft. Belvoir, Virginia 22060-6218

2.  Dudley Knox Library      2
    Naval Postgraduate School
    411 Dyer Rd.
    Monterey, California 93943-5101

3.  Director, Training and Education      1
    MCCDC, Code C46
    1019 Elliot Road
    Quantico, Virginia 22134-5027

4.  Director, Marine Corps Research Center      1
    MCCDC, Code C40RC
    2040 Broadway Street
    Quantico, Virginia 22134-5107

5.  Director, Studies and Analysis Division      1
    MCCDC, Code C45
    3300 Russell Road
    Quantico, Virginia 22134-5130

6.  Commandant of the Marine Corps      1
    C4I Directorate, Code CS
    Washington, DC 20380-0001

7.  Commanding General, Marine Corps Systems Command      1
    C4I/COMM-S
    2033 Barnett Ave. Suite 315
    Quantico, Virginia 22134-5080

8.  Commanding Officer, Marine Corps Tactical Systems Support Activity      1
    Communications Systems Division
    Camp Pendleton, California 92055-5000

9.  Professor Dan C. Boger      1
    Naval Postgraduate School, Code CC
    Monterey, California 93943

10. Professor Robert B. McGhee              1
    Department of Computer Science, Code CS/Mz
    Naval Postgraduate School
    Monterey, California 93943

11. Professor Monique P. Farques        1
    Department of Electrical & Computer Engineering, Code EC
    Naval Postgraduate School
    Monterey, California 93943

12. Major Mark E. Cantrell, USMC         1
    6860 S.W. Norse Hall Road
    Tualatin, Oregon, 97083