Applied Math 91r Research Paper – A Fine Speech Recognizer

Lukasz Strozek
Supervised by Paul Bamberg, Senior Lecturer in Mathematics

May 12, 2004

Team	***Members:***
Jennifer	*Chen*
Nick	*Elprin*
Aaron	*Greenspan*
Jeff	*Hammerbacher*
Cassia	*Martin*
Jack	*Miller*
Lukasz	*Strozek*
Mark	*Zuckerberg*

Twenty Reasons Why Dogs Don't Use Computers
7. Barking in next cube keeps activating YOUR voice recognition software.
Anonymous

Table of Contents

1 Introduction to Speech Recognition

Speech recognition is only a small subset of a much larger group of tasks called Signal Processing. All these tasks can be characterized with a "black box" approach: they perform a conversion between an analog signal (*e.g.* a pattern of vibrations in air generated mostly through resonating systems, or a pattern generated by light of varying intensity) and a much more abstract structure understandable by a human. In addition to Sound Processing, Computer Vision, a now-popular Computer Science topic, is another field embraced by Signal Processing.

Homing in to Sound Processing, we isolate two complementary processes: speech recognition and speech synthesis. The former transforms sound to language (and is thus called a Speech-to-Text process) while the latter does the opposite: transforms words into their respective utterances (Text-to-Speech). Both processes are thought to be rather difficult and, when done properly, involve complex software and hardware. The difficulty is in the inherent nondeterminism of language as a human creation (for instance, there is no formula which can fully describe pronunciation), which computers are yet not fully capable of dealing with.

1.1 Origins and History

It is mistakenly believed that sound processing did not exist in the pre-computing epoch. In fact, it all started with Bell's 1870 invention of a telephone, which is nothing more than an automaton converting sound to an intermediate form, transferring it through large distances, and then converting it back to sound.

More surprisingly, as early as in 1936 Bell labs developed a machine that was capable of synthesizing speech. Due to its many mechanical parts, the machine's utterances sounded tin-like and not human at all. For many decades, producers of sci-fi movies used this fact in developing their vision of robots.

By the 1960s, it became clear that due to the existing limits in computer hardware, an ambitious task of recognizing whole dictionaries-worth of speech was impossible. Researchers concentrated on small-vocabulary systems whose task was to recognize one utterance out of a very limited set. The first machine that can truly bear the name "speech recognizer" was the VIP 100 by Threshold Technology. Ever since, an increasing number of companies have devoted a great deal of research into the field, recognizing[†] that the ability to understand spoken commands is of high demand in a highly-computerized world.

[†]No pun intended.

1.2 Limitations of Speech Recognition Systems

There are a number of reasons why Speech Recognition is still not 100% accurate despite a decades-long research in this field.

- Since speech recognition is a deterministic process that uses statistical methods to discern words, like-sounding words are very likely to be frequently confused. For instance, saying "What are this hotel's night rates?" and "The lab is out of nitrates" is thus very likely to be recognized incorrectly.
- Speech as such is very context-sensitive. Homonymes (*i.e.* words having the same sound, but different meanings, and more importantly – spelling) means that an idea recognizer needs to "understand" what the speaking is saying (that is, be context-sensitive). A speech recognizer has no way of knowing whether the speaker refers to "bare" or "bear"; "in" or "inn"; "wait" or "weight".
- Background noise and speech distortions are very difficult to filter out, and they can grossly change the result of the recognition.
- Slurred speech, foreign accent, odd speech patterns are all factors which may decrease the accuracy of a recognizer
- Hardware that delivers the sound to the recognizer is also imperfect. For instance, the quality of a microphone may have a significant bearing on the results.

That said, voice recognition is given a great advantage – humans, aware that they are talking to a machine, subconsciously slow down and talk more precisely, willing to help the system. In conjunction with the specific purposes of the voice recognition systems (automated systems have a limited set of possible answers to check against – for instance, ZIP codes or names of the states – most of which are unequivocal), the performance of a well-constructed recognizer may reach 90-95%.

1.3 Speech Recognition As It Stands Today

Speech Recognition is now categorized according to one criterion: speaker dependence. Speaker dependent systems use designed for use by one specific user. These usually involve personal computers, PDAs, cell phones, and other devices likely to have one user.

They are easy to train and achieve high results due to low variation in test cases (it is assumed that every person is moderately consistent in their way of speaking). However, they are not scalable and are forever confined to a personal portable system.

Speaker adaptive systems are designed to adapt their operation to work with new speakers. There involve dictation software packages. Again, they are pretty easy to train, but require substantially more resources and are not as cheap to develop as speaker dependent systems. Apple's OS X now features Voice Recognition able to shut down your computer, launch an internet browser or even tell a joke (from built-in joke database, OS X is no Artificial Intelligence yet!). While still not 100% accurate, the system is an impressive addition to an easy-to-manage desktop environment.

Speaker independent systems can be used without any training whatsoever. This branch is, of course, where the demand lies, since systems that would perfectly recognize any speech would be ideal for voice-enabled telephone systems (more and more companies use these to make automated customer service more user-friendly and enhance its capabilities) like ones installed in the American Airlines or Expedia support systems. Tourists would also find such system handy as it would allow them to communicate with foreigners much more easily. Imagine a hand-held recognizer, which "listens" to what a French tour guide says, and displays the text it recognized – in English – on a display. Combined with a smart speech synthesis system, full translators seem to be an attractive option. Imagine the same hand-held recognizer actually *telling* you what that Frenchman said, in your native language!

The two leading manufacturers – IBM and Dragon Systems – managed to improve their products a lot since the early 1980s. Voice recognition products today can support continuous speech and large dictionaries, features especially useful for those who dictate text to be written on screen (*e.g.* writers). They are also able to adapt much faster, with much fewer training sessions. Today's accuracy rates for commercial systems top at 98%, which is a giant improvement over a mere 90% in the early 1990s. Most advanced packages switch between two modes – text dictation mode and command mode – seamlessly, with an accuracy rate of 90%. And even though they are no competition to a fast typist (who achieves 80-100 words per minute with an accuracy of 99.5%), systems such as this are gaining more and more favor each day.

It is obvious that voice recognition is a very useful process, which can not only improve the quality of life and help people with disabilities communicate more freely, but also help bring down language barriers and increase communication efficiency.

2 Speech Recognition – A High Level Approach and Prerequisites

Speech recognition is, in essence, a rather simple process. It consists of a series of processes tied together. These are:

- Digital sound processing – this process converts human speech into a set of data relevant to the recognizer
- Training – given the data, the training program takes a set of utterances of the same phrase to create a model (a representative) for each utterance.
- Recognition – the actual recognizer attempts to fit all models to an unknown utterance and returns the one that results in a best such fit

Variations on this scheme are possible, and we have indeed experimented with this simple design while undertaking more and more demanding tasks.

2.1 Abstraction Model

In very high-level terms, each utterance from our recognizer's vocabulary is assigned a model, which consists of a set of nodes, each node being a set of characteristics (**features**) representative of a fraction of the utterance. Training is the process, which, given a series of utterances of the same phrase (the **training set**), a model for the phrase is made. Given a set of models, the recognizer will attempt to see how well each of the utterances from the **test set** fits any one of these models, and return the one that fits the utterance best.

Let's trace a simple, (very) high-level example. Let our vocabulary consist of two phrases: "thanks" and "goodbye." Assume that our "features" consist of two data: which frequencies are emphasized (`L`, `M`, `H` for Low, Medium and High) and what the overall amplitude is (`L`, `M`, `H`). We can think of a suitable model for the former as a set of nodes

```
(M, L), (L, H), (H, M)
```

because we don't hear much of the "th" in the word "thanks" (hence low amplitude at the beginning) while the final "s" gives a high-frequency node; a suitable model for the latter could then be a set of nodes

```
(L, L) (L, H) (L, M)
```

If an utterance to recognize were of the form

```
(M, M), (L, H), (H, H)
```

the recognizer would fit the former model better and thus the phrase would be recognized as “thanks”.

The recognizer, in essence, abstracts from the actual waveform to create a set of nodes. It’s the nodes that are matched against each other. Throughout the process, a major data loss is incurred, but (at least in theory) the information lost accounts for nothing relevant to the recognizer’s task (for instance, information such as overall loudness and pitch is lost, but these should by no means decide which phrase a given utterance fits best!).

2.2 Basics of Digital Sound Processing

The utterances are all nothing more than analog signals, that is, sounds like many that surround us. In technical terms, a sound signal is a pattern of pressure changes in air – continually compressed and expanded air forces parts of the human ear to resonate, thus creating a phenomenon of hearing. When we speak, similar devices in our throat resonate and as a result produce those patterns, vibrations.

These analog signals are continuous, that is, the strength of the signal at any given time can be any non-negative number (is not quantized, or restricted to a multiple of some value) and changes continuously. Such signals are difficult for a computer to handle, since computers are nothing else than finite state machines that need to operate on finite sets. Devices exist that convert an analog signal into a digital one, *i.e.* one that consists of series of quantized data. Electrical components are made to resonate as sound passes through them thus creating voltage changes. These changes are then sampled (*i.e.* probed) with high frequency and quantized. Such signal is then easy to work on with a computer.

It has been proven that actual readings are of no use for speech recognition, since in their most basic form they don’t convey relevant information about the sound characteristics. Instead, it is argued, one should look at the frequency chart of a sound wave. What is a frequency chart? Every waveform can be represented as a sum of pure sine waves of different frequencies and amplitudes. In technical terms, the frequency chart is a decomposition of a waveform by frequency ranges. If the waveform is a set of amplitude readings as a function of time

$$w(t) = (V_1, V_2, \ldots) \tag{1}$$

its frequency chart is a set of energies characteristic for several frequency ranges as a function of time

$$\vec{f}(t) = ((E_1(f_1), E_1(f_2), \ldots, E_1(f_n)), (E_2(f_1), \ldots, E_2(f_n)), \ldots) \tag{2}$$

In fact, each of f_i is a range of frequencies. Each set of energies, for a given time unit (called a **frame**) becomes a **feature** of the frame. Given these features, models of speech can be made.

2.3 Theory of Hidden Markov Models (HMM)

A Markov process is a simple stochastic process in which the distribution of future states depends only on the present state and not on how it arrived in the present state. If, in addition, such process is given in discrete time values, such process is called a **Markov chain**.

A hidden Markov model (HMM) is a model used widely in AI research, as well as speech recognition, based on the Markov chain idea. Formally, it is a five-tuple $(\Omega_X, \Omega_O, A, B, \pi)$, where

Ω_X is a finite set of possible states $s_1, \ldots, s_n$
Ω_O is a finite set of possible observations $v_1, \ldots, v_M$
$A = \{a_{ij}\}$ are transition probabilities, *i.e.* $a_{ij} = Prob(X_{t+1} = s_j \mid X_t = s_i)$
$B = \{b_i\}$ are observation probabilities, *i.e.* $b_i(k) = Prob(O_t = v_k \mid X_t = s_i)$
$\pi = \{\pi_i\}$ is the initial state distribution, *i.e.* $\pi_i = Prob(X_0 = s_i)$

Let $\sigma = (f_1, \ldots, f_m)$ be the sequence of actual observations, $\sigma \subseteq \Omega_O$. Given a fixed Ω_X and Ω_O, we seek:

- $Prob(\sigma \mid \{A, B, \pi\})$, the probability of the observations given the model
- The most likely state trajectory (**path**) given the model and observations
- Adjust $\{A, B, \pi\}$ so as to maximize $Prob(\sigma \mid \{A, B, \pi\})$

In our case, Ω_X will correspond to a finite set of nodes while σ – to the frames of our utterance. We seek the probability that a given model $\{A, B, \pi\}$ fits our utterance and the path which represents the most likely assignment of states to the observations. Refer to the following two sections for actual application to a speech recognition task.

2.4 Basics of Training

The task of training is as follows: each phrase in a vocabulary needs to be assigned a model. Such model will be a set of nodes which best characterizes a given utterance. The training is performed in two stages:

- First, it needs to be decided where the speech begins and where it ends. Input signal may have plenty of "padding" silence surrounding the speech.
- Then, the program needs to create an actual model consisting of n nodes that best fits all of the utterances of a given phrase. An important performance question addressed in this paper is one of what n should be in order to create most representative models.

The two tasks are, surprisingly, very similar. The first one can be modeled as describing a finite state machine with three states: silence (s_1), followed by speech (s_2), followed by silence (s_3). Given an utterance consisting of a series of frames

$$f_1, f_2, f_3, \ldots, f_m \tag{3}$$

If a frame f_i belongs to a state s_j, we say $f_i \in s_j$. The algorithm needs to find two boundaries: one between s_1 and s_2 (let's call it b_1) and one between s_2 and s_3 (say, b_2). A **value** of a boundary b_j is defined as an index i such that $f_i \in s_j$ while $f_{i+1} \in s_{j+1}$.

First, these boundaries are assigned some initial values. Then, a Dynamic Programming method is used to determine how well given set of frames represent this initial breakdown.

Our state space is an ordered pair (f_i, s_j), *s.t.* $f_i \in s_j$. At any next frame f_{i+1}, we can either stay in the same state s_j or move to the next state s_{j+1}. With each pair there is a fitness value associated with it, which described how well a given state models a given frame. In fact, our algorithm assigns **penalties** to pairs instead of fitness values (we can think of penalties as being negative fitness values). A **path** is a series of legal moves from the initial state (f_1, s_1) to the final state (f_m, s_3). A path with the smallest penalty uniquely determines the boundaries: since there is exactly one i_1 such that (f_{i_1}, s_1) and (f_{i_1+1}, s_2) both belong to the best path, the value of b_1 is i_1. Similarly, since there is exactly one i_2 such that (f_{i_2}, s_2) and (f_{i_2+1}, s_3) both belong to the best path, the value of b_2 is i_2.

Given these boundaries, a model is created as follows: Each node corresponding to a state s_j is an average of all frames f_i such that $f_i \in s_i$. For the speech-silence model, states s_1 and s_3 are merged together in creating a single node called the **silence** node.

Given these new boundaries (closer to the actual model, but not necessarily defining the best possible model), a similar process can be performed again to refine the boundaries. Given enough iterations, the boundaries converge to their optimal values.

One should notice that the same procedure can be carried out for the second stage of training, *i.e.* creating an actual model. All frames between the two boundaries, that is, frames $(f_{b_1+1}, f_{b_1+1}+1, \ldots, f_{b_2}$, give us our new state space. Given that we need n nodes in our model (the value of n needs to be agreed upon), we have n states $s_1, \ldots, s_n$, and so these frames are divided up linearly into $n-1$ parts and the initial $n-1$ boundaries are drawn. Dynamic Programming method is used to determine the path of minimum penalty, and given this path, new state boundaries are determined. Again, iteration of the above process will eventually make the boundary values converge to some optimal values.

Because we have several utterances of one phrase, an average across all corresponding frames in all utterances is taken while performing the latter process. However, for the silence-speech-silence model, each utterance is treated individually as the amount of silence across utterances may vary significantly.

Really? Or do we take average of the nodes? Nick, Jen?

The following example may be helpful.

Example, Lucas

2.5 Basics of Recognition

Given a set of models $M_k = (F_{k,1}, \ldots, F_{k,n_k})$ consisting of n_k nodes, and a sequence of frames $f_1, \ldots, f_m$ of the unknown utterance, the recognizer will then attempt to find a model which fits the utterance best (*i.e.* a model which gives a minimum-penalty path). This is done using a Dynamic Programming method similar to one used for training. For each k, our state space is an ordered pair $(f_i, s_{k,j})$, which denotes frame i being in a state corresponding to $F_{k,j}$, the j^{th} node of a k^{th} model. At any next frame f_{i+1}, we can either stay in the same state $s_{k,j}$ or move to the next state $s_{k,j+1}$. With each pair there is a penalty value incurred, proportional to the misfit of the frame. The Dynamic Programming method finds the **path** of a smallest penalty from $(f_1, s_{k,1})$ to (f_m, s_{k,n_k}). A value of k for which such penalty is the minimum determines the model which the unknown utterance fits best (ties are broken arbitrarily).

Because, effectively, the entire state space, and thus the running time of the algorithm, is $O(\max_1^K(n_k)mK)$, a cubic-order, our focus lay in improving the existing algorithm with heuristic methods, outlined in the following chapters.

2.6 Introduction to Phonetics and IPA

Phonemes are smallest unit of speech. Every utterance is a set of phonemes, which correspond to different sounds in our speech. For example, the words "could" may be thought of as consisting of sounds like "$\mathcal{K}$", "$\mathcal{U}$" and "$\mathcal{D}$". In fact, there is a special alphabet consisting of hundreds of symbols, which makes transcribing words in most languages in the world possible. It is an International Phonetic Alphabet (IPA), whose chart is included below.

IPA chart, Lucas

One idea for a speech recognizer is to generate models for each phoneme as opposed to each utterance. The phrase to be recognized is then matched against a set of phonemes as models, and the best phoneme model for the phrase is found. We transcribed all the phrases in our test to an ASCII IPA system, which is a simplified IPA system using plain ASCII codes to make the transcriptions easier to encode in a computer-based speech recognition system. We first used an online English-IPA dictionary at

`http://www.foreignword.com/dictionary/IPA/`

and then followed ASCII IPA conventions at

`http://odur.let.rug.nl/ kleiweg/ascii-ipa/`

The transcriptions provided are not perfect, and, in particular, required much adaptation for casual American pronunciation, and word boundary ellisions.

2.7 Terms Used Throughout

To recap, below is a list of all terms that will be used throughout this paper. This section is primarily for reference.

- **Utterance** – a phrase recorded and stored in memory, which constitutes the unit of speech to recognize. The word "mathematics", for example, is an utterance because it exists in our vocabulary.
- **Vocabulary** – a set of all utterances whose models we have. In other words, a set of all phrases which the recognizer is able to discern
- **Phoneme** – an indivisible unit of speech. Utterances are made up of phonemes. Phonemes correspond to different sounds that we utter in order to say these phrases.
- **Sample** – a smallest unit of the digital waveform. Sampling is a process whereby the signal strength is probed with high frequency. Each time the strength is probed, its value is one sample.
- **Frame** – a smallest time chunk of the frequency chart. A frame corresponds to a range of samples, usually 120ms-worth of an utterance.
- **Frequency** – an utterance consists of a series of frames, each frame having characteristic layout of frequencies. Each frequency has a relative signal power associated with it. A set of all frequencies' powers in a given frame constitute this frame's features.
- **Amplitude** – the strength of each frame. The louder the frame is, the greater its amplitude.
- **Features** – a set of characteristics for a given frame, *i.e.* the powers for each frequency and the amplitude. Training and recognition treat these features as building blocks for the models.
- **Node** – a "representative" feature. Nearby features in each utterance are grouped together and averaged out, and their average across all similar utterances may be basis for a node.
- **Model** – a set of nodes representing a given utterance. Each model consists of a set of nodes. For example, there is a model for the utterance "applied math". These models are used in recognition.
- **Training Set** – a set of utterances of the same phrase used in training; these utterances will help create a model for the phrase
- **Test Set** – a set of utterances unknown to the recognizer, which the recognizer will attempt to match with one of the models in its memory and thus, one of the phrases

3 Our Implementation of a Speech Recognizer

Our team's goal was to design and implement a new voice recognizer from scratch, building upon existing ideas and changing the recognizer design through experiments and innovation. The following were our main goals, and we attempted them in this particular order:

- A discrete speech recognizer, able to discern between a very small vocabulary, consisting of the phrases "Mathematics", "Computer Science", "Applied Math" and "Linguistics". The recognizer was trained using five utterances of each phrase, and in the final testing it was supposed to recognize twenty shuffled utterances from the test set. All utterances were recorded by the same person. The training algorithm used a fixed number of nodes per model, and so the phrases had to be of similar lengths in order to achieve best results.
- We then decided to attempt a more useful task of recognizing vocabulary from a traveler's foreign language phrase book. We used Berlitz Hungarian For travelers, 1981, but the choice is entirely arbitrary. From that phrase book, we transcribed about 130 phrases into IPA. We ran simulated annealing over those phrases to generate a minimal phrase set that contained all the phonemes that appeared in our dataset. Running with 1000 iterations actually resulted in a smaller dataset than the 20 phrases we were looking for. We augmented the resulting 13 phrases with another 7 picked manually to represent less common phonemes. Since this expanded vocabulary increased the running time of our program, we introduced thresholding, that is, reducing the search to elements likely to produce the correct answer (more about thresholding in the following chapters).
- Duration model was introduced. So far, our model guaranteed nothing about how many frames each node consisted of. We decided to introduce duration modeling so that extra penalty would be incurred for nodes consisting of too few or too many frames.
- Phoneme model was then introduced. Given the IPA translations of each phrase, we decided to make each phoneme a node (or a couple of nodes) and perform the training by creating a model for each phoneme and then attempting to recognize the phrases by checking how close the recognized phonetic model fits any of the models given in the vocabulary. This task was obviously less accurate as we treated all phonemes the same way, and so thresholding had to be reduced in order to make sure we don't introduce false results.
- Finally, an attempt at connected digit recognition was made. We created a model for each phoneme in each context and then attempted to recognize series of three digits.

> This task would be impossible with discrete speech as the number of models under an old scheme equals a thousand (all possible three-digit combinations). With context-sensitive phoneme-based recognizer, substantially fewer models were necessary.

We decided to use PHP as our programming environment. Since the end goal of the project was to create a voice recognition system that would work on pre-recorded audio files, simple enough for the average end-user to work with, the simplicity of PHP as well as availability of different modules makes our task easier. Moreover, great computational power is not of prime importance as the task of speech recognition does not demand the speed premium offered by an ahead-of-time compilation. Moreover, creating the system required stringing together a number of interrelated but separate software components: Windows-based recording software, signal processing code, training algorithms, and the core of the software, the recognizer itself. PHP makes combining such components really easy. In this particular project, each component was assigned to a different team of programmers, where each team comprised one or two students. With the exception of the recording software, all of the components were written by students in PHP.

Since PHP is a scripting language that is popular for its ability to create web-based applications relatively quickly, the interface for the voice recognition software was also web-based, unlike most commercially-available voice recognition systems, which run on the Win32 platform. Also, since the code was written (almost) entirely in PHP, apart from the actual recording, it is completely portable and platform-independent. Since different team members had different operating systems, writing the code in PHP eliminated the problem of porting working code. Working modules could be immediately shared between all team members.

We also designed our own Bug Database and Version Control System. Members of the team checked out working versions and checked in code they were working on. This made the coding effort efficient, as well as enabled the teams to work on the project concurrently.

☢ *Aaron – more to write?*

4 From Speech to Digital Waveform

This (and the following ones) chapter will concentrate on our actual implementation. Details of the implementation as well as changes in the original design will be presented and discussed.

The first step to recognizing speech was to convert it to a digital form. As of today, there exists no possibility for PHP to record sound on a client's computer (PHP, by definition, is a processing language working on the server side), so we had to rely on a platform-dependent professional application. We used a `Win32` application called `NCollect`, which recorded the utterances and saved them in a NIST file.

NIST (`.nwv`) files are raw sound files enriched with headers preceding the actual data. These headers contained information about the length of an utterance and a prompt associated with each (it was possible to assign a prompt to each recorded phrase from within `NCollect`) – the two pieces of information relevant to us.

One enormous advantage of the NIST format is that NIST files allow multiple utterances to be combined into one file by simple file concatenation. This means that all utterances from one session were stored in a single file, which made data processing much simpler and cleaner, with fewer files that needed to be uploaded and downloaded by various team members.

More information can be found on this format from the Linguistic Data Consortium at:

`http://www.ldc.upenn.edu/Using/`

One important limitation we had to consider when choosing our sampling rate (the frequency with which probing is performed) is the so-called Nyquist frequency. Since sampling introduces data loss, it is impossible to exactly reconstruct the waveform from its sampled form. In essence, due to quantization in both the voltage and time, only an approximate form of the utterance is represented by the digital data. Data loss due to sampling is called **aliasing**. A particular form of aliasing occurs when the frequency of any wave component is *higher than 1/2 of the sampling rate* (this threshold value is called the Nyquist frequency). If this is the case, aliasing causes the information about the waveform to be entirely lost. Let's give an example.

Example of aliasing and Nyquist frequency, Lucas

Since frequencies found in speech rarely exceed 4 kHz, we decided to record data with a sampling rate of 11025 Hz (this is one-fourth of a professional sampling rate, but is still greates than the Nyquist frequency for speech, so it doesn't reduce the quality of our recordings). The data was a single-channel (Mono) 16-bit data, which means that the wave data was represented by integers ranging from -32767 to 32768.

The `nwv` files collected through `NCollect` were then uploaded to the server, which performed their spectral analysis.

5 From Waveform to Spectral Display

From this point on, every process was carried out with code written by us in PHP. First, a PHP module called `fft.php` performed a Fourier Analysis on the raw sound data in order to extract its spectrum information.

Fourier Analysis is a technique which allows a decomposition of a signal, $w(t)$, into its component frequencies, $\vec{f}(t) = (f, E(f))$. It assumes that $w(t)$ is periodic. To make $w(t)$ periodic, for a given **window** of size N, we construct

$$\begin{aligned} w'(0 \leqslant t < N) &= w(t) \\ w'(t < 0) &= w'(t + Nk), \text{ where } k = \left\lceil \frac{-t}{N} \right\rceil \\ w'(t \geqslant N) &= w'(t - Nk), \text{ where } k = \left\lfloor \frac{t}{N} \right\rfloor \end{aligned} \tag{4}$$

so that $w'(t) = w'(t + N)$ for all t. To a certain approximation, and with a sufficiently large window, this reflects true characteristics of the signal.

Fourier showed how, given $w'(t)$, one can find $F(t)$, defined as

$$F(f) = \sum_{-\infty}^{\infty} w'(t) e^{-2i\pi f t} dt = \sum_{-\infty}^{\infty} w'(t) \left(\cos(2\pi f t) - i \sin(2\pi f t)\right) dt \tag{5}$$

Since we're dealing with discrete values, a variation on this has also been developed called the Discrete Fourier Transform (DFT). For discrete-valued sequence of complex-valued data

$$x_0, x_1, \ldots, x_{N-1} \tag{6}$$

in a similar fashion to w', we *extend* x to all integer values:

$$\begin{aligned} x(i < 0) &= x(i + Nk), \text{ where } k = \left\lceil \frac{-i}{N} \right\rceil \\ x(i \geqslant N) &= x(i - Nk), \text{ where } k = \left\lfloor \frac{i}{N} \right\rfloor \end{aligned} \tag{7}$$

so that $x(i) = x(i + N)$ for all i. The Discrete Fourier Transform is defined as

$$X(n) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) e^{\frac{-2ik\pi n}{N}} \tag{8}$$

for $0 \leqslant n < N$. Since $X(n)$ is a complex series, we define

$$\text{amplitude} = A(n) = ||X(n)|| = \sqrt{\text{Re}(X)^2 + \text{Im}(x)^2} \tag{9}$$

$$\text{phase} = \tan^{-1}\left(\frac{\text{Im}(x)}{\text{Re}(x)}\right) \tag{10}$$

Since the original $x(i)$ comes from the real waveform $w(i)$, we have

$$\begin{aligned} \text{Im}(x(i)) &= 0 \\ \text{Re}(x(i)) &= w(i)\ \forall i \end{aligned} \tag{11}$$

It has also been shown that human ear is insensitive to phase (if the same signal is Fourier transformed into component waveforms, a random number of waveforms is phase shifted, the resultant sum sounds to a human ear exactly the same as the original waveform), so for the purposes of speech recognition, it is irrelevant information. The pairs $\{(n, A(n))\}$, where n is the so-called **Harmonic Number** and ranges from 0 to $N/2$, give us the frequency chart. The frequencies are not spread linearly; in fact, $A(0)$ is the DC component (the average of the input series), $A(1)$ is the amplitude of the first harmonic, which is the sine component whose period is N, *etc.* Finally, $A(N-1)$ is the amplitude of the Nyquist Frequency – the component whose period is 2. The following picture illustrates the relationship between the harmonics and the values of $X(n)$.

Picture from fft-good.pdf, Lucas

Performing Fourier Analysis on regularly translated windows gives us various values for $\{(n, A(n))\}$ as functions of time.

For the purposes of our implementation, we used an efficient version of DFT perfectly suited for the processing of digital data. A technique called Fast Fourier Transform does the analysis in time $O(n \log n)$ instead of the original $O(n^2)$. The implementation we're using in our project – `Radix-2 Cooley-Tukey` – has been developed by Cooley and Tukey in 1965 and requires that the number of data points be a power of 2. This condition can easily be satisfied by choosing a window size that is a power of 2 and padding the data with enough zeros at the end of a signal so that the last window fits entirely in the data.

Since cropping data into a window introduces an artifact at the boundary (the signal is continuous between $i = 0$ and $i = N - 1$, but, since $x(N) = x(0)$, there is a discontinuity between $i = N - 1$ and $i = N$ and this discontinuity repeats throughout all multiples of N), we introduced a technique called *windowing* to reduce the artifact. Essentially, multiplying the signal $x(i)$ by a continuous waveform that vanishes at the two ends causes the resultant signal to be continuous. We used a Hamming window

$$h(0 \leqslant i < n) = \sin \frac{i\pi}{N} \tag{12}$$

The following figure illustrates the effect of a Hamming window.

Hamming window, Lucas

Our module, `fft.php`, performed a Fast Fourier Transform on data read from the `nwv` file and stored the results in an `spp` file whose format we invented. Its format is as follows:

```
for each utterance from the nwv file,
    line 0:     header of the form
                Utterance <number> <prompt>:<IPA transcription>
    lines 1-n: n frames of the form
                Frame <number> <amplitude> <X(i) for F frequency ranges>
    at least one blank line
```

Where `amplitude` is the total frame amplitude, which is followed by F sets of data, one for each frequency group. A window of size 1024 gives us 512 possible frequencies, which then were combined in groups of 32 to give 16 output frequency ranges.

Further finetuning led to an introduction of partially overlapped windows. In order to make the output chart short enough, windows should not overlap in all frames apart from the first and the last, but in a much smaller fraction of frames. After certain experimentation, an overlap of 25% was introduced.

Due to the attenuation of high-frequency signals, a high-frequency filter has been introduced: replacing each data $x(i)$ with the first-difference $x(i)-x(i-1)$ acts as a pretty effective high-frequency filter. Further experimentation led to the final substitution

$$x(i) \;\longleftarrow\; x(i) - 0.9x(i-1) \tag{13}$$

The data is usually displayed as a spectrum graph, where the horizontal axis denotes time, the vertical axis – frequency ranges, while the intensity (energy) is represented with varying color of the points. PHP makes it possible to generate graphics on the fly and display it in the browser. Out display module takes an `spp` file and generates an image of the spectrum. Since the amplitude, as well as individual energies, are relative values, the display module transforms the data by adding on the amplitude to each of the energies, which in turn are scaled to fit around the average. The color values range from black to blue (default for silence) to yellow to white. The scale is asymptotic, so that arbitrarily large values of energies may be accommodated, though in real life (and with our microphone) values greater than 400 were never encountered.

The display module, `graphics.php`, can display graphs of an arbitrary size (the default size is directly proportional to the number of frequency ranges in the vertical direction and the number of frames in the horizontal direction). Bilinear and bicubic resizing of the display is possible for better visualization. The module also implements a cutoff filter, which both accelerates the display, and eliminates the random noise from the display. It is important to note that the actual *spp* data becomes unchanged.

Below is a sample waveform and its spectrum.

Sample waveform and its spectrum, Lucas

6 Training and Recognition

After generating all the `spp` files, the control is given to the training program, which takes an `spp` file for the training data. By default, this file consists of a sequence of utterances comprising of our vocabulary, five utterances per phrase in consecutive order. The training module then outputs a model file (`mdl`) of the following format:

```
for each model
    line 0:      blank
    line 1:      MODEL <model type> <model name>
    line 2:      blank
    line 3:      NODE\t\tPARAM\tMEANS(DEV)
    lines 4-n+3: <node name>\t<mean(s.d.) of amplitude>\t<data>
                 <data>: for each component: <mean(s.d.) of component>
```

☢ *Check format. Nick, Jen?*

The penalty function is based on the likelihood of a given value to be modeled by a Normal distribution with the above mean and standard deviation. Let the mean of a node be μ and the standard deviation – σ. Let the value of one component of a feature be x. Then the penalty is given by

$$p = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{x-\mu}{\sigma}} \tag{14}$$

In order to speed up calculations, we noticed that instead of attempting to find the probability of fitting the model, we can only look at likelihoods (the difference between likelihoods and probabilities is that with likelihoods, only relative quantites are meaningful, but this is exactly what we seek) and thus take the log of p. The actual penalty is then given by

$$p' = \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2}\frac{x-\mu}{\sigma} \tag{15}$$

Taking the logs then simplifies the matter even more since in order to find the total likelihood of the model fitting the data, all we need to do is to add all these **log-likelihoods**.

☢ *What is the actual penalty? Nick, Jen, Jack?*

The same penalty function is used in the recognizer.

While both sound processing and training can be done off-line, the actual recognition is rather time-critical and needs to be done as quickly as possible. The initial Dynamic Programming method has a cubic-order running time, which, for a large number of possible models, becomes infeasible. The following sections focus on the improvements and refinements of the original method, as well as changes in design necessary to accomplish the imposed set of goals.

6.1 Discrete Phrase Recognition

Recall that the first task was to perform a successful, 100% recognition of short phrases given a very small vocabulary. We used the phrases “Mathematics”, “Applied Math”, “Lignuistics” and “Computer Science” because these are the departments which approved of this 91r project. The training module created a model for each of the four utterances. We used a fixed number of nodes n for all models, though the value of n was experimentally chosen to be optimal (see following sections). The recognizer then used the four models to find the best fit of an occurrence it was given. This task was simple enough so that once all teams finished writing their modules it took a brief meeting to combine all modules together and perform the entire process of digital processing, spectrum analysis, training and recognition. Again, PHPs modular design paradigm proved to be of immense help here.

For the sake of organization, `lst` files were created, which are plain text prompts, including all phrases in the order in which they were recorded. We wrote a script, that given a vocabulary list, generated three `lst` files for training, practice, and testing.

The training `lst` file repeats each phrase five times before going on to the next one. The practice `lst` file repeats the entire vocabulary five times. The test `lst` file is completely randomized, depending on the Shuffler class, available online.

6.1.1 Exhaustive Search

The very first task only consisted of four phrases, each phrase was modeled with at most twenty nodes, and the number of frames of each utterance rarely exceeded one hundred. This allowed us to run the recognition basically as an exhaustive search procedure. Penalties were found for all models and the model with the smallest penalty was chosen to be the answer. Our recognizer achieved a 100% accuracy rate, correctly identifying 20 shuffled utterances, each corresponding to one out of four phrases in the vocabulary.

6.1.2 Thresholding

The next task was somewhat more resource-consuming. We aimed to expand our vocabulary to 20 phrases. It quickly became clear that some heuristics need to be implemented in order to shorten the running time of the recognizer.

The main idea is to eliminate all paths which are unlikely to give the minimum-penalty solution as early on as possible, given their current behavior, in order to prune the search space and achieve speed growth. An idea of thresholding was put forth: all paths whose current penalty is above a certain threshold will be instantly pruned. This threshold was chosen to be a certain margin above the up-till-now-minimal path. Let the threshold be, say, h. Let the penalty of the currently minimal path be P. This means that any state in the Dynamic Programming model whose current penalty is greater than $(1+h)P$ is immediately assigned a penalty of $+\infty$ and thus never considered again. Now, if this process were extended to all models we're currently looking at, the potential speedup benefits are enormous since we're pruning an entire two-dimensional space (all modes in all states in one time frame). Following chapter discusses the speed of the algorithm with various parameters and variable threshold.

With thresholding turned on and the vocabulary extended to 20 words, on a 100-phrase test set our recognizer achieved an astonishing 98% success rate. Curiously, the two phrases that the recognizer confused were "Hurry up" and "...".

☢ *...and what? Jack?*

6.1.3 Duration Modeling

Hidden Markov Model is a nice theoretical device allowing us to formulate the problem of speech recognition and seek its solution, but one feature that it's inherently missing is (by virtue of its definition) state memory, some mechanism for controlling how long, for example, the finite state machine should spend in each node.

As an experiment, we decided to add duration modeling to our recognizer. We decided to model the duration of a node with the Poisson distribution. Using this distribution makes sense, because the longer we stay in a node, the larger the variation in number of frames it will be assigned is. Also, Poisson distribution is an easy one to work with since its only parameter λ happens to be both the mean and the standard deviation. We can extract the mean quite easily from the presented models and plug this mean in as our λ value in the Poisson distribution.

To calculate the λ values for each node in each word, we just take all models of the word and average the number of frames assigned to the node. We can then store these values for the λs in a two-dimensional array (or other suitable data structure), indexed by word and node.

We experimented on two possibilities:

- Adding the negative log likelihoods upon exit from a node. We just needed to add

$$\lambda + \sum_{i=1}^{n} (\log n - \log \lambda) \tag{16}$$

  on exit from each node, where n is equal to the number of frames assigned to a node.

- Adding the negative log likelihoods after each frame. Here we need to add λ each time we enter a new, and add $\log n - \log \lambda$ each time a new frame is assigned to that node.

Obviously, both methods reduce to the same value on exit from the node. The latter, however, allows us to use thresholding for each frame (the former adds penalties only after a state change).

The results of using duration modeling, together with thresholding, are presented in the following chapter.

6.2 Phoneme-based Recognition

Up until this point we were still using a fixed number of nodes per phrase for all phrases. This clearly was a wrong approach as the phrases that we introduced into our vocabulary were no longer similar in length. We decided to use a variable number of nodes, and how many nodes a given phrase should have was deduced from the IPA transcription of the phrase. We decided to use δ nodes per phoneme, but we also added some phonemes, as outlined below. δ was then experimentally chosen.

Of course, in a real speech recognizer, we'd need an IPA dictionary in order to be able to transcribe users' utterances on the fly. For simplicity, we decided we happened to have IPA transcriptions for all utterances in our vocabulary.

Our standard transcriptions ignore strength and stress marks, but we used '`!`' for aspiration. In addition, we included a few new symbols for the phonemes that are affected by a neighboring "r":

`<r` – as in "air"
`~r` – as in "fur" without the "f"
`>r` – as in "more" without the "m"
`+r` – as in the "tr" part of "translate"

For instance, the phrase "Could you drive more slowly, please." has been transcribed as

```
:kUd ju draIv moUr sloUli p!liz
```

For this task, we wrote an additional script to generate our modified `spp` files to combine the English prompts from the `lst` files with the IPA transcriptions.

The graphics module was modified to allow for display of node boundaries *with* corresponding phonemes underneath the nodes, which became a neat debugging tool (it was easy to see if the training module went hayward by looking at the spectrogram and checking if the phonemes were correctly placed) as well as an illustration of a rather impressive feature of our recognizer.

Once this part was verified to work rather well, we moved on to phoneme-based recognition, where models for individual phonemes across all utterances of all phrases are found, and then the recognizer looks at the phonetic transcriptions of the phrases in the vocabulary and concatenates the models for the phrases by concatenating the phoneme models in real-time.

6.2.1 Basic Model

In a basic model, every occurrence of the same phoneme was treated as the same model. In other words, both "m" in "mathematics" and the "m" in "applied math" were all averaged out together to create a universal model for the phoneme "m". This was obviously bound to create a less accurate recognizer, since due to averaging of many nodes, each model suffered a great information loss. Moreover, we could not guarantee that all same phonemes were correctly identified, which introduced even greater possibility of misinterpretation. This stage was purely experimental, as we wanted to see how accurate our recognizer can get even with such small number of training data. Having a couple of occurrences of each phoneme is clearly not sufficient to fully reconstruct the phonetic model for all words in a language. The following chapter summarizes the work and the results we achieved.

6.2.2 Context-sensitive Model

A better idea is to use a context-sensitive model, where a model is created for each possible junction of two phonemes rather than phonemes themselves. This would involve a significantly greater amount of resources as the number of nodes potentially increases quadratically and the recognizer now has a much greater number of nodes to work with. However, if thresholding were to prove effective in pruning the search space, a context-sensitive model would be a much more accurate model to perform the recognition with.

6.3 Attempts at Connected Recognition

An additional interesting speech recognition task is connected number recognition, which is quite useful for recognizing phone numbers and zip codes. An arbitrary length number recognizer can be adequately modelled by learning only three digit numbers. The idea here is to create a model for each digit preceded with and followed by every other digit. This would create nodes that span across digits, as well as nodes that refer to actual digits. For example, a number 347 can be thought of as consisting of the following nodes:

☢ *Nodes: (-3)(3)(34)(4)(47)(7)(7-), Lucas*

The data generation task is slightly different here, because we do not know ahead of time exactly what sequences we will want to recognize. Creating a model for, say, all 1000 possible combinations of three-digit numbers is obviously too resource-consuming. We therefore had to be careful to maintain some scarcity.

We decided to generate five sets of one hundred numbers, using three of these sets for training, and one each for practice and testing. These sets are each generated semirandomly, with the property that each two digit sequence occurs once in the front position and once the back position. This assures that each digit shows up 10 times in each of the three positions, and that we will always have data on how that digit behaves in the context of other digits. For example, an arbitrary sequence “11” shows up in one of the training datasets as “115” and ”311.”

The code for this is available in the module “generate_numbers.php.” The basic algorithm operates as follows. First, we generate a random permutation of the 100 numbers (0-9 ten times each) that will appear as the first digit. Then, for the second digit, we generate 10 permutations of the 10 numbers, one to follow each possible first digit. Thus, we have 10 random arrays. If '0' is the first digit, then we will take the second digit from array '0'. If '1' is the first digit, we will take the second digiit from array '1'. The third digit is generated exactly like the second digit, except that numbers are grabbed from its arrays dependant on the second digit.

7 Working Recognizer in Action

This chapter describes the running process of our recognizer. Since it's been written entirely in PHP, the front-end was a website which we could access from any computer anywhere, connected to the network.

After recording the user's voice using a program provided by ScanSoft called NCollect, the user was prompted to complete a series of steps in a web browser. These in turn concatenated the recordings, processed them for frequency strength, created training models based on the energy at each of the levels, and then matched the training models against a second series of recordings to determine what the user had actually spoken. The end result appeared as a list, with the phrase determined to have been spoken at the top, followed by other possible matches, each corresponding to an indexed score generated by the software recognizer.

☢ *Aaron, more?*

8 Result Analysis

8.1 Accuracy

* 20 phrases, thresholding as a function of threshold
* With/without duration modeling
* With n nodes per phoneme

8.2 Speed

* Thresholding
* With/without duration modeling AND thresholding

We need some here, Jack, Anyone else?

9 Further Improvements

Our recognizer, though it works pretty well, is still far from being a commercial product. Many ideas we've had, we didn't have a chance to test out. Some more tempting improvements are given in the following sections.

9.1 Yet Faster FFT

Though the sound processing part of the entire process is not a bottleneck due to an incredibly fast FFT algorithm, one can say that what doesn't hurt much should just as well be implemented. One can make use of the fact that the imaginary part of the data is always zero and write a more efficient implementation of FFT, which can speed up all calculations by a factor of two. The same speedup can be accomplished with using bit-shifted integers instead of double-precision numbers or tabulating logarithms.

9.2 Pipelined Process and just-in-time training

This is not usually the case, but in rare cases where the recognition briefly follows the training, and both processes happen once only, the training could actually be postponed until the recognition started. Then the training and the recognition could be pipelined, so that the two processes take less time in total. There are, however, a couple of major difficulties with this approach as this would make thresholding very difficult, if not impossible (not all models are available at the time of recognizing).

9.3 Full Phoneme-Based Recognition

If we ever get hold of a large set of spoken data, all with IPA transcriptions, we can attempt a task to generate a library of phoneme models, which would include phoneme boundaries (perhaps only some more problematic ones). With such full set of models, a 100% phoneme-based recognition is possible, where the recognizer needs to know nothing about the vocabulary of the speaker. Perhaps a less extensive version of training could be used to "finetune" the existing phoneme set and adapt it to the speaker. This is all, we think, in the realm of possibility.

9.4 Continuous Recognition

One feat seems to be being able to perform a continuous, real-time recognition task. This is possible with a real-time processing language (PHP, unfortunately, such language is not), a lot of memory, threads running in parallel, processing the FFT and the recognition, and – most importantly – a very good thresholding algorithm, since the recognition part becomes the bottleneck of the entire process. Obviously, phoneme-based or word-based recognition that takes into account phoneme/word boundaries is a must.

Perhaps a smart duration model with good, accurate thresholding and an extensive library of phoneme models can accomplish the task. The author recommends that such task be taken up by the ever-ambitious 91r-*ers* of Spring 2005.

◊

10 Bibliography

Voice Recognition Technology,
`http://florin.stanford.edu/~t361/Fall2000/TWeston/consumer_products.html`

Wells, J., *IPA transcription systems for English*,
`http://www.phon.ucl.ac.uk/home/wells/ipa-english.htm`

Understanding FFT,
LDS Inc., 2003 (`www.lds-group.com`)

Ziemer, R., *Signals & Systems*,
Prentice Hall, New Jersey, 1998

Bourke, P., *Discrete Fourier Transform. Fast Fourier Transform*,
`astronomy.swin.edu.au/~pbourke/analysis/dft/`

Oppenheimer, A.V., Shaffer, R.W., *Digital Signal Processing*,
Prentice Hall, New Jersey, 1975

Rabiner, L.R., *A tutorial on hidden Markov Models and selected applications in speech recognition*, Proceedings of the IEEE, 1989

Charniak, E., *Statistical Language Learning*
MIT Press, 1993