Spontaneous Speech Recognition Using HMMs

by

Benjamin W. Yoder

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2001

Author .....
Department of Electrical Engineering and Computer Science
June 11, 2001

Certified by .....
Deb Roy
Assistant Professor
Thesis Supervisor

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

Spontaneous Speech Recognition Using HMMs

by

Benjamin W. Yoder

Submitted to the Department of Electrical Engineering and Computer Science
on June 11, 2001, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Electrical Engineering

Abstract

This thesis describes a speech recognition system that was built to support spontaneous speech understanding. The system is composed of (1) a front end acoustic analyzer which computes Mel-frequency cepstral coefficients, (2) acoustic models of context-dependent phonemes (triphones), (3) a back-off bigram statistical language model, and (4) a beam search decoder based on the Viterbi algorithm. The context-dependent acoustic models resulted in 67.9% phoneme recognition accuracy on the standard TIMIT speech database.

Spontaneous speech was collected using a "Wizard of Oz" simulation of a simple spatial manipulation game. Naive subjects were instructed to manipulate blocks on a computer screen in order to solve a series of geometric puzzles using only spoken commands. A hidden human operator performed actions in response to each spoken command. The speech from thirteen subjects formed the corpus for the speech recognition results reported here. Using a task-specific bigram statistical language model and context-dependent acoustic models, the system achieved a word recognition accuracy of 67.6%. The recognizer operated using a vocabulary of 523 words. The recognition had a word perplexity of 36.

Thesis Supervisor: Deb Roy
Title: Assistant Professor

Acknowledgments

Thanks are due to Deb Roy for his help and advice throughout my research.

Also, thanks go to all members of the Cognitive Machines group, with special thanks to Peter Gorniak and Kai-Yuh Hsiao.

Contents

List of Figures

List of Tables

Chapter 1

Acoustic Modeling

In this section, the acoustic models will be explained and discussed. First, the mathematics behind the training of context independent phoneme units will be discussed, including some basic math for HMMs, initialization via Segmental K-means, and the training using an Expectation Maximization algorithm (Baum-Welch). Second, the extension of this to context-dependent phoneme units will be covered. Third, the evaluation method and results for the units as trained on the TIMIT database will be discussed.

1.1 Basic Math for HMMs

HMMs are often used for modeling Markov processes. A process is considered Markovian if:

$$P(o[n]|o[n-1], o[n-2], o[n-3], ...) = P(o[n]|o[n-1])$$

That is, if the probability of an observation at time n ($o[n]$) depends only on the observation at time $n-1$. If all of the information in a process can be contained in the current observation, the description of a process can be thought of in terms of transitions between states where each state is enough to represent everything known about the process. Although this is not true of speech, it can be used as a good simplifying assumption and has been successfully applied in state-of-the-art speech

recognition systems.

Following the conventions of [1], we introduce some basic math and terminology which is useful for dealing with HMMs. An HMM consists of a matrix of transition probabilities with elements a_{ij} being the probability of transitioning from state i to state j, and output probabilities $b_j(\mathbf{o})$ being the probability of outputing the observation vector $\mathbf{o}$ in state j. In a discrete density HMM, the HMM can output a number of discrete values. In this case, the output probabilities for each state specify the probability of outputing the discrete values. In the continuous density case, the probability density must be specified for the observation vector $\mathbf{o}$. This specifies the output probability for all possible values of $\mathbf{o}$.

The set of all of the HMM parameters is given by λ. Two basic problems of dealing with HMMs and their solutions are briefly discussed here to introduce the appropriate terminology (see [1] for a complete discussion). The two basic problems are determining the probability of an observation sequence given λ for an HMM, and determining the most probable state sequence for a given observation sequence (given λ). The additional problem of choosing model parameters so as to maximize the observation probability $P(\mathbf{O}|\lambda)$ is discussed in the section on training using the Baum-Welch method.

1.1.1 Determining the Probability of an Observation Sequence

Given an observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, .., \mathbf{o}_T)$ of length T and HMM parameters λ, the goal is to determine $P(\mathbf{O}|\lambda)$. One solution to this is known as the forward procedure. Defining the variable

$$\alpha_t(i) = P(\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, ..., \mathbf{o}_t, q_t = i|\lambda)$$

where q_t is the state at time t, and i is the current state. Given an initial probability distribution over the states π_i (π_i is the probability of the sequence starting in state i) and a final probability distribution over states ρ_i (ρ_i is the probability of the sequence ending in state i), $\alpha_t(i)$ can be solved inductively:

1. Initialize α at time $t = 1$:

$$\alpha_1(i) = \pi_i b_i(\mathbf{o}_1)$$

2. Inductive step

$$\alpha_t(i) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(\mathbf{o}_t)$$

To determine $P(\mathbf{O}|\lambda)$, all that remains is to sum the αs for the final observation over all states:

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \rho_i \alpha_T(i)$$

Thus the evaluation takes $\Theta(N^2T)$ operations which scales linearly with the length of the observation sequence.

Another similar way of solving the problem is to work backwards from the ending state. The variable $\beta_t(i)$ can be defined as

$$\beta_t(i) = P(\mathbf{o}_{t+1}, \mathbf{o}_{t+2}, \mathbf{o}_{t+3}, ..., \mathbf{o}_T, q_t = i|\lambda)$$

Thus $\beta_t(i)$ is the probability of the HMM accounting for observations after time t and being in state i at time t. Similar to the forward procedure given above, one can find an inductive solution which is given by:

1. Initialize β at time $t = T$:

$$\beta_T(i) = \rho_i$$

where again ρ_i is the probability of ending in state i.

2. Inductive step

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)$$

This induction, working backwards from a final distribution is typically called the "backward procedure". The HMMs used for the acoustic modeling are left-to-right. This means that the only legal transitions are to either the same state, or to a state further on (the transition matrix specifying a_{ij} is upper triangular) As the

HMMs used here are three state left-to-right and require the observation sequence to enter at the beginning and exit the HMM at the end, the values of π_i and ρ_i can be updated with the other HMM parameters using the Baum-Welch algorithm discussed in section 1.2.3.

There are also a few variables which are of use later and are derived from the forward and backward variables. One variable, $\xi_t(i,j)$ is the probability of being in state i at time t and state j at time $t+1$.

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda)$$

$$\xi_t(i,j) = \frac{P(q_t = i, q_{t+1} = j, \mathbf{O}|\lambda)}{P(\mathbf{O}|\lambda)}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(\mathbf{o}_{t+1})\beta_{t+1}(j)}{P(\mathbf{O}|\lambda)}$$

This is useful when reestimating transition probabilities with the Baum-Welch algorithm. Another useful variable, $\gamma_t(i)$ is the probability of being in state i at time t given the observations:

$$\gamma_t(i) = P(q_t = i|\mathbf{O}, \lambda)$$

$$\gamma_t(i) = \frac{P(\mathbf{O}, q_t = i|\lambda)}{P(\mathbf{O}|\lambda)}$$

$$\gamma_t(i) = \frac{P(\mathbf{O}, q_t = i|\lambda)}{\sum_{i=1}^{N} P(\mathbf{O}, q_t = i|\lambda)}$$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

The variable $\gamma_t(i,k)$, the probability of being in state i at time t with the kth mixture accounting for the observation, is also useful for continuous density HMMs:

$$\gamma_t(i,k) = P(q_t = i, \text{mixture} = k|\mathbf{O}, \lambda)$$

$$\gamma_t(i,k) = \gamma_t(i)\frac{c_{ik}N(\mathbf{o}_t, \mu_{ik}, \mathbf{U}_{ik})}{\sum_{m=1}^{M} c_{jm}N(\mathbf{o}_t, \mu_{im}, \mathbf{U}_{im})}$$

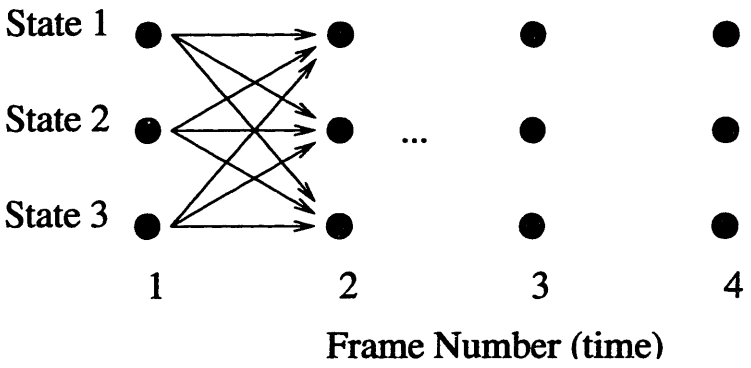

Figure 1-1: Trellis for a three state HMM

Where (as discussed later) μ_{ik} is the vector of means and $\mathbf{U}_{ik}$ is the matrix of variances for gaussian mixture k in state i of the HMM.

1.1.2 Finding the Optimal State Sequence

Now that several useful quantities can be calculated for HMMs, it is relatively easy to determine the most probable state sequence for an observation sequence **O**. This algorithm, called the Viterbi algorithm is one which will be used extensively for other tasks such as word and phoneme recognition. Consider a trellis which is essentially a matrix with the number of rows being the number of states (N) and the number of columns being the length of the observation sequence (T) shown in figure 1-1. For each element in the trellis, it is necessary to store its current probability and the state that it most likely transitioned from. Consider the states in frame one. Their probability is simply $\pi_i b_i(\mathbf{o}_1)$ or the probability of starting in state i and outputting the first observation. For the remaining frames, the probability for a state j is given by:

$$\max_i \delta_{t-1}(i) a_{ij} b_j(\mathbf{o}_t)$$

where $\delta_{t-1}(i)$ is the probability stored in the lattice at the previous time step. Only the maximum is required, because the state contains all of the information of the process. If the same state can be reached several different ways, only the most probable needs to be kept. When this maximum is found, the state that created the maximum is stored at the node as well as the maximum probability. Repeating this process until

Figure 1-2: Feature Extraction Method

the end of the observation sequence creates probability values and the previous state information (backtraces) for all elements of the trellis. It is then possible to read back the most probable state sequence by finding the most probable state at time T and following the backtraces until the first state is reached. The result is a time ordered sequence of states corresponding to the most likely state sequence. This algorithm also runs in $O(N^2T)$ time and so is not more costly than the calculation of the $P(\mathbf{O}|\lambda)$

It should be noted that in all of the calculations for the HMMs, the precision of the floating point values used is important. For long observation sequences, the probability of the observations will drop and eventually exceed the precision of the float. As a solution to this, the logarithms of probabilities are used for probabilities throughout. This also simplifies the evaluation of the probabilities for gaussians as the logarithm function cancels out the exponential greatly speeding up computation for decoding. Multiplications also become less costly, although the sums given in many of the previous formulae become more expensive. Details on this approach can be found in [1].

1.2 Training Context Independent Units

Training HMMs from speech consists of several steps. First, the data passes through a feature extractor which replaces a waveform with evenly spaced features at discrete times. Next, an HMM structure is assumed, and the model is initialized with a Segmental K-means algorithm. Finally, the model is trained with the Baum-Welch algorithm which increases the probability of observations given the model.

1.2.1 Feature Extraction .

Feature extraction takes place as shown in figure 1-2. First of all, the waveform is

pre-emphasized with a filter whose z-transform is $1 - .97z^{-1}$. This compensates for the "radiation load" which sound undergoes as it exits the mouth. The radiation load is an effect of the sound exiting from the vocal tract (which is often modeled as a tube) into an open space through the mouth. The result of the pre-emphasis filter is to accentuate high frequencies. The waveform is then windowed with a Hamming window of length 20 ms, and the window is moved in increments of 10 ms. across the length of the waveform. This creates a vector of features every 10 ms. The samples within this window are passed through a filter bank to determine the amount of each frequency range present in the signal. The filter bank consists of triangular filters spaced according to the Mel Scale (see [1], p. 190). The Mel scale on the filter bank is used because it mimics the design of the human ear: it provides good resolution at low frequencies and progressively coarser resolution as the frequency increases. Following this, the log of each filter bank coefficient is taken. This helps to separate out constant components of the cepstra, as it turns the multiplication in the frequency domain into an addition. As the average is subtracted off of each component at the end, this allows the feature extraction to concentrate mostly on feature differences. In this specific instance, 24 mel scale filters were used, and with the data given being sampled at 16 kHz, the 24 triangular filters in the filter bank were spread between 0 and 8 kHz.

Next, the output of the filter banks is transformed using a discrete Cosine Transform (DCT). This is useful later on as it removes some of the dependencies between outputs so that the covariance matrices can be assumed diagonal. The DCT is implemented using the following equation (see [3], p. 591):

$$X[k] = \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi k(n + 0.5)}{N}\right)$$

where N is the number of filter banks, and k is the index to the transformed coefficients. In this specific case, 12 transformed coefficients were used.

Third, liftering was done to attempt to smooth these coefficients (see [1], p. 169). The idea is that the first few cepstral coefficients are mostly due to variability in the

shape of an individual speakers glottal pulse and other speaker related factors [1]. Thus the liftering attempts to de-emphasize these cepstral coefficients as they do not contain much information. This liftering takes the form

$$X[k] = 1 + h \sin\left(\frac{n\pi}{L}\right)$$

where h is typically $L/2$ and L was chosen to be 22 for an 8 kHz bandwidth. As discussed later, in tests on phoneme recognition, this improved performance by about .2%, which is only a small improvement, but as the extra processing time is negligible, it was included anyway.

Next, log energy is included as an extra coefficient. The log energy was calculated before the window was applied, and was found by summing the square of all of the samples in the windowed region:

$$E = \log \sum_{n=0}^{N-1} x[n]^2$$

Calculating the energy using a windowed signal was also attempted. As figure 1-3 shows, the energy computed without windowing is more smoothed, as it depends equally on all samples within the window. It was found empirically that the phoneme recognition performance did not depend on whether the energy was first windowed or not.

As a final step, the average coefficient was computed over the utterance. This average was then subtracted from the coefficient at each frame. The goal of this step is to remove the constant factors of the speech so that the differences can become more noticeable. In addition, as the waveform is not stationary, the set of coefficients were augmented to include first and second order derivatives. The derivatives were calculated using the following regression equation:

$$x'[i] = \frac{\sum_{n=1}^{N} n(x[i+n] - x[i-n])}{2\sum_{n=1}^{N} n^2}$$

where N is the size of the regression radius (how many frames are included in cal-

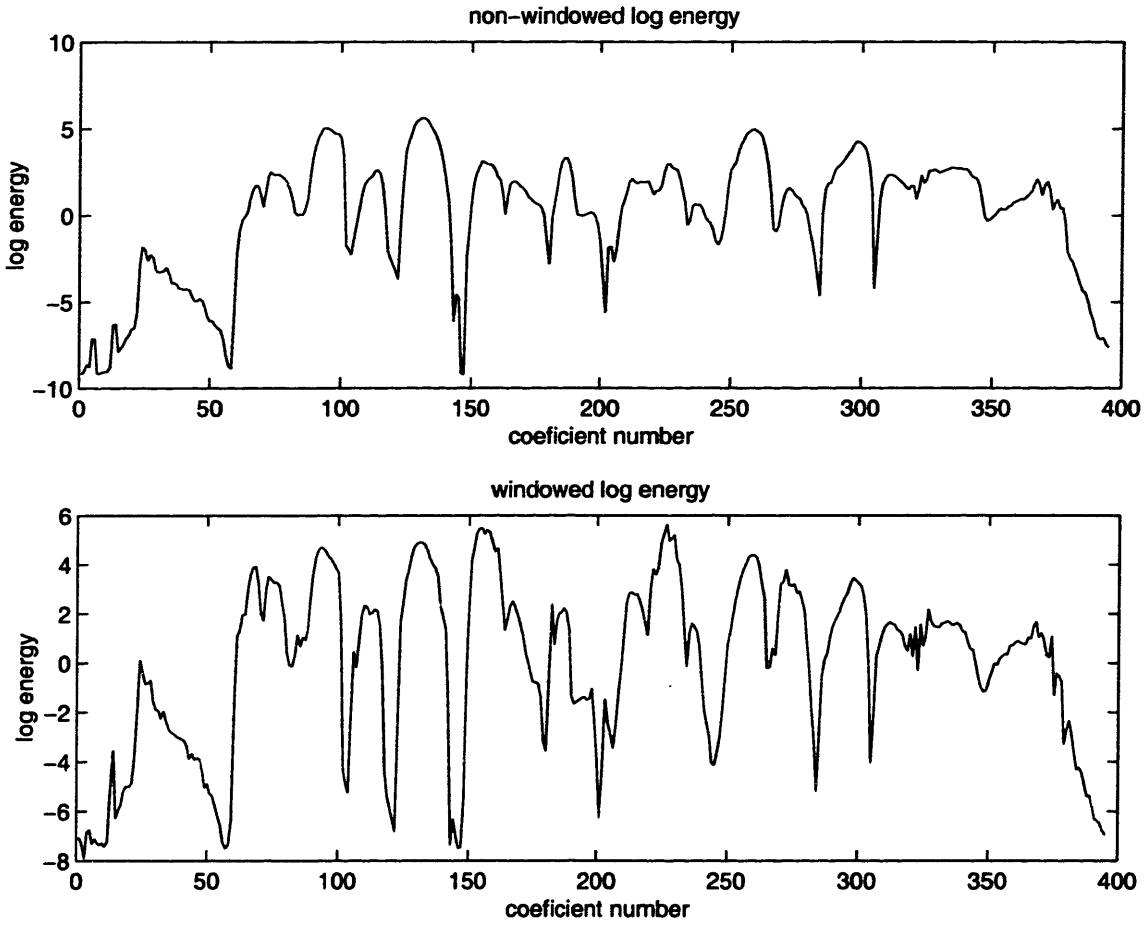

Figure 1-3: Comparison of windowed and non-windowed energy for the utterance "She had your dark suit in greasy wash water all year."

culating a derivative). A large regression radius will not be very sensitive to noise, but will lack time resolution whereas a small regression radius will have good time resolution but may be affected more by noise. In the current system, the regression radius is fixed at 2. The second derivative is calculated from the first derivative by use of the same regression formula. This produces three times the original number of features for a total of 39 features per frame for the current system.

1.2.2 HMM initialization through Segmental K-means

The next step after running the data through a feature extractor is to initialize the HMM through a Segmental K-means algorithm. The output and transition probabilities can be iteratively improved through the Baum-Welch method for Expectation

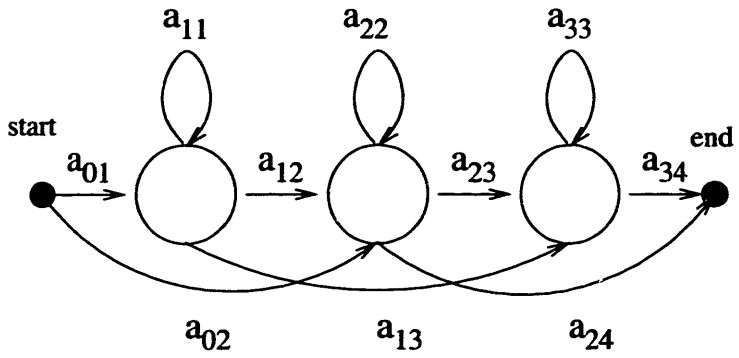

Figure 1-4: Left to right HMM structure with single state skips.

Maximization. However, this method may only reach a local maxima, so it is important to initialize these probabilities to reasonable estimates. The HMM structure assumed is a three state HMM with single state skips allowed (as shown in 1-4). This allows for greater variability in speaking rate, as a phoneme no longer has to last at least 30 ms. in order to pass through the HMM.

The HMM transition weights (a_{ij}) consist of the probability of transitioning from state i to state j, and are shown in figure 1-4. An observation sequence starts at the "start" label in 1-4 where it undergoes null transitions (not providing any output observations) into one of the first states. The observation sequence is also required to end by transitioning to the "end" state, so the transition matrix is actually of size $N+2$ where N is the number of states in the HMM. For recognition purposes, these HMMs can easily be chained together to form words or other units.

The HMM outputs are modeled as continuous density output probabilities by representing each output probability as a gaussian mixture. Thus the output probability for an HMM in state j is given by:

$$b_j(\mathbf{o}) = \sum_{k=1}^{M} c_{jk} N(\mathbf{o}, \mu_{\mathbf{jk}}, \mathbf{U_{jk}})$$

where $N(\mathbf{o}, \mu_{\mathbf{jk}}, \mathbf{U_{jk}})$ denotes a gaussian distribution for observation vector $\mathbf{o}$ with mean vector $\mu_{\mathbf{jk}}$ and covariance matrix $\mathbf{U_{jk}}$. Here the sum is over the mixtures (k), $\mathbf{o}$ is the observation vector, $\mathbf{c_{jk}}$ are the mixture weights for each state, and $\mu_{\mathbf{jk}}$ and $\mathbf{U_{jk}}$ are the mean vector and covariance matrix for the gaussians respectively. As the

probability distribution for the output probabilities must integrate to one,

$$\sum_{k=1}^{M} c_{jk} = 1$$

Also, because the DCT tends to remove dependencies between the MFCC coefficients, the gaussians are assumed to be diagonal. This simplifies many of the calculations which are described later.

The training data for the HMMs consists of a set of observation sequences $\mathbf{O} = [\mathbf{O_1}, \mathbf{O_2}, \mathbf{O_3}, \mathbf{O_4}, ...]$. Each observation sequence consists of a frames, $\mathbf{O_i} = (\mathbf{o}_1^{(i)}, \mathbf{o}_2^{(i)}, \mathbf{o}_3^{(i)}, ...)$. The more observation sequences available, the more accurately and robustly the HMM can be trained since the parameters will accurately reflect the variation in the observations.

The first step in initialization is to give reasonable values for the output and transition probabilities. This is done by taking all of the observation sequences $\mathbf{O}$ with more frames than states and segmenting them so that an equal number of frames lie in each state. This provides a set of observations points ($[\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, ...]$) for each state which are produced by the output probabilities. These observations can be used to initialize the output probability (a gaussian mixture) of a given state through a k-means procedure as given below:

1. Choose k of the observation points as the initial means of the gaussians.

2. Label all of the points with the mixture of the gaussian that is the closest (in Euclidean distance) to it.

3. Recalculate the means and variances for each mixture by calculating the mean and variance of all of the points with the same label as the mixture. Recalculate the mixture weights by taking the fraction of points which are associated with the mixture divided by the total. If a mixture weight falls below a threshold, than reinitialize the mixture with a new random point and return to Step 2.

4. Recalculate the labels. If they do not change then quit, otherwise return to Step 3.

This initializes the output probabilities. The transition probabilities can be set to reasonable default values and then updated iteratively along with the output probabilities.

Next, these initial guesses are iteratively refined by using a hard alignment of states. For each observation sequence, the Viterbi Algorithm is applied to find the most likely state sequence for the observation sequence. The transition weights are updating by counting the number of transitions from a given state to another state divided by the total number of transitions from the state:

$$a_{ij} = \frac{\text{number of transitions from } i \text{ to } j}{\sum_i \text{number of transitions from } i \text{ to } j}$$

The output probabilities are refined by getting the set of observations given by the Viterbi Algorithm to be most likely in that state. These output probabilities are then used to run the Segmental k-means algorithm but without the initialization of the means given by Step 1 of the procedure.

1.2.3 HMM Training Through Baum-Welch

While the above procedure will often improve the $P(\mathbf{O}|\lambda)$, it is not guaranteed to do so. Usually after several iterations of Segmental k-means the benefits are negligible. There exists an Expectation Maximization (EM) algorithm which is guaranteed to iteratively improve the $P(\mathbf{O}|\lambda)$. An instance of EM, the Baum-Welch algorithm provides a way of updating the HMM parameters iteratively so that either the $P(\mathbf{O}|\lambda)$ increases or a "stable point" has been reached so that further iterations will not improve the parameters.

Let $\gamma_t(i,k)$ be defined as before to be the probability of being in state i at time t with the kth mixture accounting for the observation given the observation sequence and the HMM parameters:

$$\gamma_t(i,k) = \left(\frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}\right)\left(\frac{c_{ik}N(\mathbf{o}_t,\mu_{ik},\mathbf{U}_{ik})}{\sum_{m=1}^{M}c_{jm}N(\mathbf{o}_t,\mu_{im},\mathbf{U}_{im})}\right)$$

The re-estimation formula for the mixture weights consists of the expected number of times the observation sequence was in the ith state with the kth mixture component accounting for the observation divided by the expected number of times in the ith state.

$$c_{ik} = \frac{\sum_{t=1}^{T} \gamma_t(i,k)}{\sum_{k=1}^{M} \sum_{t=1}^{T} \gamma_t(i,k)}$$

Similarly, the re-estimation formulas for the mean vector $\mu_i k$ and the variance matrix $\mathbf{U}_{ik}$ of the kth mixture component of the ith state can be given as follows:

$$\mathbf{u}_{ik} = \frac{\sum_{t=1}^{T} \gamma_t(i,k) \cdot \mathbf{o}_t}{\sum_{t=1}^{T} \gamma_t(i,k)}$$

$$\mathbf{U}_{ik} = \frac{\sum_{t=1}^{T} \gamma_t(i,k) \cdot (\mathbf{o}_t - \mu_{ik})(\mathbf{o}_t - \mu_{ik})}{\sum_{t=1}^{T} \gamma_t(i,k)}$$

An important issue in updating the variances of continuous density HMMs is that the variances of the gaussians must be appropriately floored such that they do not become too sharply peaked and account for only one data point. These floor variances are calculated by reading in all of the observation sequences for a particular HMM and calculating the mean for all of the dimensions of the observation, $\mu = \sum_{t=1}^{T} \mathbf{o}_t$. The variances are then estimated for each dimension by taking the difference from the mean and squaring, $\mathbf{U} = \sum_{t=1}^{T} (\mathbf{o}_t - \mu)(\mathbf{o}_t - \mu)$. These variances are then divided by a constant factor (10 in this case) to give the floor values of the variances reasonable values.

The transition weights are implemented by taking the expected number of transitions from state i to j divided by the expected number of transitions from state i. As the number of total transitions from a state is the same as the number of observations in the state, the formula can be given as:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

It is possible to implement the Baum-Welch algorithm according to the above formulas such that all of the observation data does not have to be in memory at one

time. This can be done by use of accumulators. Three accumulators must be stored for each state and mixture. The fourth must be stored for each combination of states. The accumulators are:

1. $\sum_t \gamma_t(i,k)$

2. $\sum_t \gamma_t(i,k) \cdot \mathbf{o}_t$

3. $\sum_t \gamma_t(i,k) \cdot (\mathbf{o}_t - \mu_{ik})(\mathbf{o}_t - \mu_{ik})$

4. $\sum_t \xi_t(i,j)$

As can be seen from the re-estimation formulas, these accumulators provide enough information to update the HMM parameters using the Baum-Welch algorithm. Observation sequences are read in one at a time, and $\gamma_t(i,k)$ and $\xi_t(i,j)$ are calculated for all mixtures and states. From these, $\sum_t \gamma_t(i,k) \cdot \mathbf{o}_t$ and $\sum_t \gamma_t(i,k) \cdot (\mathbf{o}_t - \mu_{ik})(\mathbf{o}_t - \mu_{ik})$ are calculated and then used to update the four accumulators by summing these values over time. A graph of the improvements per iteration for the Baum-Welch algorithm of the phoneme "aa" is shown in figure 1-5. This phoneme was initialized using Segmental k-means, and then trained for 10 iterations with the Baum-Welch algorithm described above. Notice that each iteration increases the observation probability.

1.3 Context Dependant Phoneme Units

Ideally the base units chosen for word recognition are independent of context, so that they don't depend on the other units around them. In practice, however, the way phonemes are pronounced varies depending on the phonemes around them. Much of this has to do with co-articulation, where the movements made by the mouth and vocal cords are coordinated to produce smooth speech. This coordination means that adjacent phonemes are produced from one fluent motion, and thus the pronunciation of any phoneme is tied to the other phonemes around it. In order to deal with this effect, models which depend on the context need to be developed. The typical

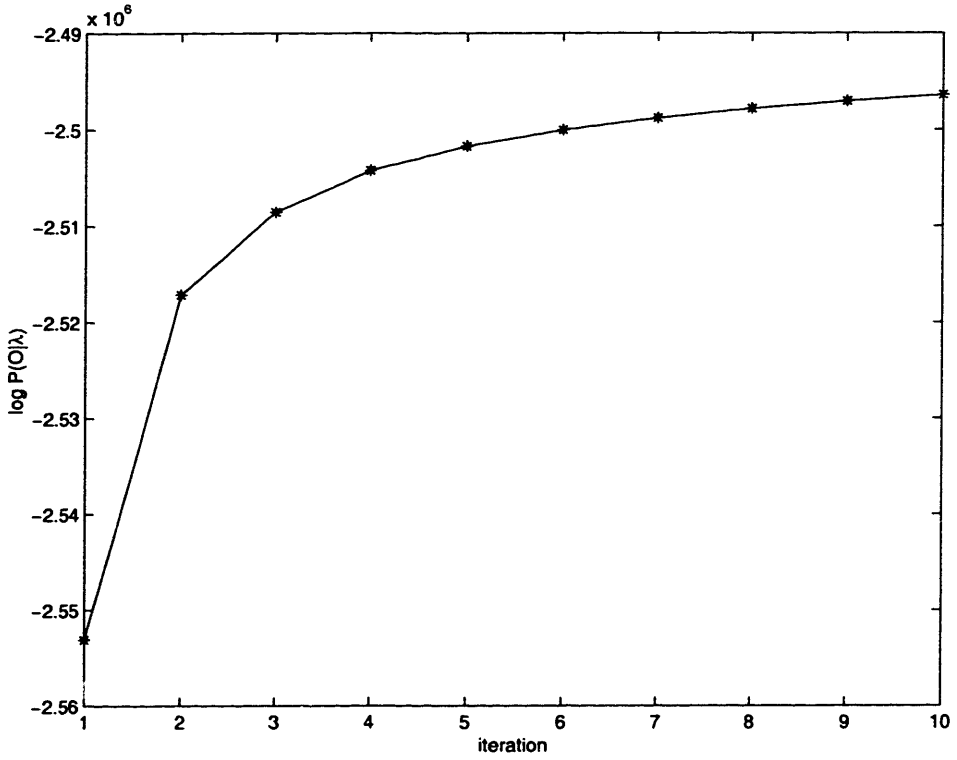

Figure 1-5: $P(\mathbf{O}|\lambda)$ vs. Baum-Welch iteration for phoneme "aa"

approach is to use "triphones", phoneme models that depend on the previous and following phonemes (context independent models are often referred to as "uniphones"). As there are usually around 50 phonemes to be trained, training an HMM for all possible contexts requires $50^3 = 125,000$ models. This is clearly impractical, especially since many phoneme contexts are rare and acquiring enough data to robustly train HMMs for rarely occurring contexts would take a very large training corpus. One approach is to cluster those contexts which have similar effects on the phoneme in question. For example, the "ey" phoneme in the words "came" and "game" may be similar even though it is preceded by different phonemes.

1.3.1 Clustering Algorithm

The clustering algorithm used here is similar to that used in [4]. This approach is appealing because it does not depend on any extra phonetic knowledge to do the clustering. If context dependent units need to be trained for new languages or

situations, no extra phonetic knowledge needs to be used. The algorithm also scales well: not all of the training data needs to be in memory at the same time and the length of time necessary to run depends primarily on the number of different contexts present for a given phoneme.

The clustering is done on a state level so as to better model the effects of the surrounding phonemes. For example, the first state will usually depend more upon the preceding phoneme than the following one, and the last state will depend more upon the following phoneme than the preceding one. The first step is to calculate the variance for each dimension of the observation vectors. This provides a good way of scaling the importance of the information of each dimension. This variance is used both in the distance metric between output distributions and for the floor variances assigned to each state. As before, $\mu = \sum_{t=1}^{T} \mathbf{o}_t$ is estimated, and from this $\mathbf{U} = \sum_{t=1}^{T} (\mathbf{o}_t - \mu)(\mathbf{o}_t - \mu)$ is estimated and $\mathbf{U}$ is assumed diagonal. The variances for each dimension, σ_k^2 are these diagonal elements of $\mathbf{U}$. The next step is to get a list of observations for each state (keeping track of the preceding and following phoneme for each observation). We do this by finding the optimal state sequence for each observation sequence by use of the Viterbi algorithm. All of the observation sequences have can then be split into observations occuring in each state. We proceed with the following sequence of steps for each state:

1. Separate all of the different contexts into different "bins".

2. Calculate the mean vector $\mu = \sum_n \mathbf{o}_n / N$ for each bin.

3. Find the smallest distance between all pairs of contexts. If this distance is lower than a threshold, merge these two bins. Here the distance between two groups is defined as:

$$d(i,j) = \sqrt{\frac{1}{K} \sum_k \frac{(\mu_{ik} - \mu_{jk})^2}{\sigma_k^2}}$$

   where k is the dimension being summed over and K is the number of dimensions. The two bins are merged by combining their observations and then creating a mean vector which is a weighted sum of the mean vectors of the bins being

merged.

4. Repeat step 2 until the distance exceeds the threshold.

5. Merge those bins with fewer counts than a minimum count threshold to the bins closest (in distance) to them.

The cutoff threshold was set to 0.3 and the minimum number of counts to 200. This gave an average of about 10 clusters per state for trigrams. Of course, more clusters were created for phonemes for which there were many observations, and fewer clusters were created for phonemes for which there were only a few available observations. The values for the cutoff threshold and the minimum number of counts were found empirically to give decent results, although changing these values (especially the cutoff threshold) did not produce noticeable changes.

As an example of this algorithm, consider three different contexts for the "ey" phoneme which will be used to train the "ey" acoustic model. Suppose there are several observation sequences all of which come from the words "came", "game", and "same". First of all, the variance for each dimension is calculated using all of the observation vectors to set the floor variance for all of the states. Second, the optimal state sequence for all of the observation vectors is found using the Viterbi algorithm. All of the observations in the first state are then used to find the clustering for the first state of the "ey" phoneme. As there are three separate contexts, "c ey m", "g ey m" and "s ey m", there are three different bins created, and the mean and the variance is calculated for each bin. If the variance for any of the bins is smaller than the floor variance, it is set to the floor variance. Next the distance metric is applied, and suppose contexts "c ey m" and "g ey m" are similar enough to be merged. The observations from the two bins are merged, and the mean and variances are updated. Now suppose that the "s ey m" context is different enough from the merged context "c ey m" and "g ey m" that it does not fall within the threshold of the distance metric and that both bins have enough data to train an output distribution. Two separate output distributions are created for the first state of the HMM, and the process is repeated for the remaining two states.

After the clustering algorithm is run, the new output distributions need to be retrained. This can be done with a modified Baum-Welch algorithm. The output distributions used in the re-estimation formulas depend on the phoneme context. When re-estimated using Baum-Welch, first the output distributions for each state are determined from the phoneme context. Next, the variables $\gamma_t(j,k)$ and $\xi_t(i,j)$ are calculated as before using these output distributions. Finally, the accumulators are updated. As there are now multiple output densities for each state (dependent on phoneme context), there are accumulators for each state, mixture, and output distribution. For a specific observation sequence, only those accumulators corresponding to the output distributions for the given phoneme context need to be updated.

At first, the output distributions for each phoneme context were initially set to those of the uniphone distributions. The output distributions were then updated using Baum-Welch for the triphones. This was found, however, to produce many mixtures of weight 0 (no counts were found to be accounted for this mixture). This occurred when the context dependent output distribution for a particular phoneme context differs from the context independent distribution significantly. The simple solution to this difficulty is to reinitialize the mixtures with further applications of a k-means algorithm. After this re-initialization, mixtures better modeled the output distributions, as there were many more non-zero mixtures. The re-initialization is similar to the segmental k-means procedure. The observation sequences are segmented into groups of observations for each output distribution using Viterbi alignment and then initialized using a k-means procedure. (There may be several output distributions for each context dependent cluster.) The Viterbi alignment for triphones only differs from that of uniphones in that the output distributions for each state must be determined from the phoneme context.

1.4 Evaluation and Testing of Acoustic Units

Once the phoneme models have been trained, it is important to evaluate their performance. First of all, a brief discussion of a common speech corpus (TIMIT) will

be presented. Next, methods for evaluating performance through a modified Viterbi search will be covered. Finally, the recognition results will be presented for several different choices of parameters.

1.4.1 The TIMIT database

A common benchmark for these results is their performance on the TIMIT database [8]. The TIMIT database consists of 6,300 utterances of roughly 5-10 sec. in length. These sentences consist of speakers from different dialect regions throughout the United States reading ten sentences. Of these ten sentences, two are identical across all speakers ("SA" sentences), three are "phonetically compact" sentences designed at MIT ("SI" sentences), and five are "phonetically diverse" sentences selected by TI ("SX" sentences). The TIMIT database comes sectioned into a "test" database and a "train" database so that results can be compared across different systems. The data provided includes the utterance, as well as phoneme and word level transcriptions. Because of the hand labeled phoneme level transcriptions, the TIMIT database provides a good way of creating initial phoneme models. Initial models can be trained from the phoneme level transcriptions. These models can then be used as a starting point to segment utterances from other speech corpa for which phoneme level transcriptions are not available. This segmentation is discussed later. A complete list of the phonemes used in the TIMIT database can be found in 4.

1.4.2 The Phoneme Recognizer

In order to evaluate performance, the acoustic models are used on unseen utterances in phoneme recognition. As shown in figure 1-6, the process begins with the feature extraction which takes the raw audio and extracts MFCCs as described earlier in Section 1.2.1. The MFCCs are fed into a phoneme recognizer which will produce the sequence of most likely phonemes. Finally, the generated list of phonemes are compared with the actual phonemes by use of a program by NIST which does the scoring.

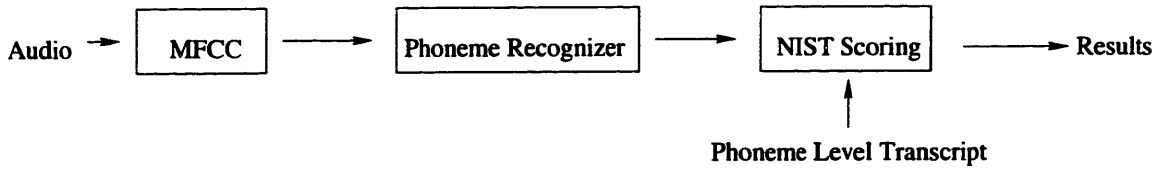

Figure 1-6: Phoneme Recognition Process

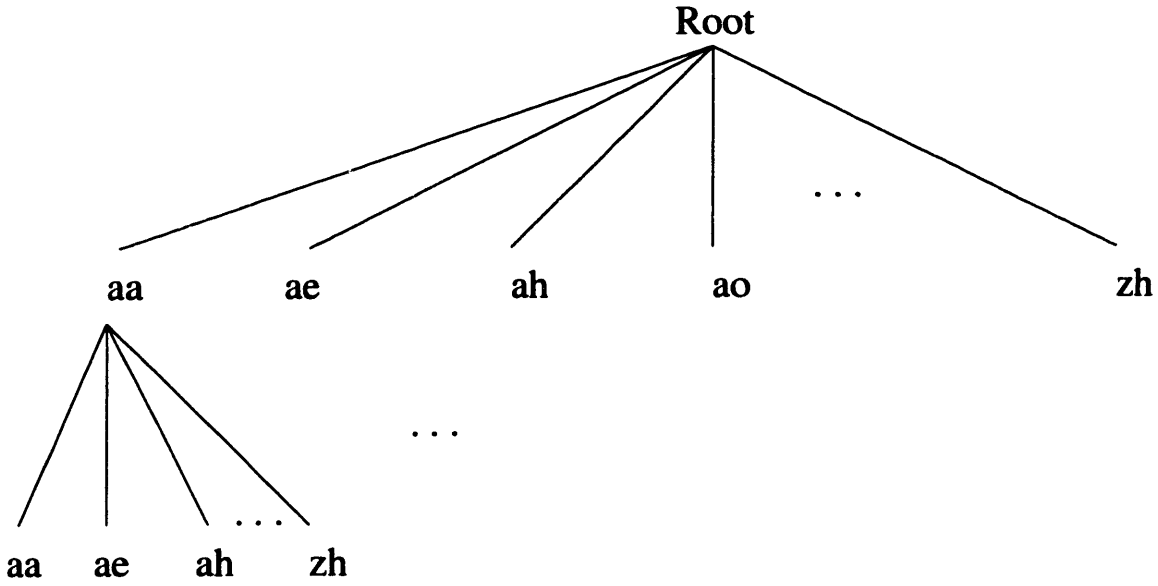

Figure 1-7: Tree for uniphone recognition

Phoneme recognition is achieved using a modified Viterbi algorithm. If the Viterbi algorithm were carried out unmodified (as discussed earlier), the trellis would consist of approximately 2,000 states (there are around 15 context dependent output distributions per state for about 50 three state HMMs). The Viterbi search runs in N^2T time, so decoding an utterance of 5 sec. duration would take on the order of 2 billion operations and is clearly unreasonable.

An alternative is to use what is called a "beam search" [1]. The idea is that many of the states present in the trellis are of low probability and are unlikely to be part of the decoded sequence. Instead of visualizing the decoding method as a trellis, it may be helpful to view as a tree search. As shown in figure 1-7, the tree can be expanded for each phoneme, and a full level search will provide the same results as the trellis implementation. The difficulty with this approach is that the number of leaves in the tree grows as N^t where t is the depth of the tree and N is the number of phonemes.

This ignores the fact that when two leafs at the same level are being evaluated for the same phoneme, only the most probable one needs to be evaluated, and the other one can be pruned. Thus one of the two leafs can be removed, keeping only the most likely one. To remove leafs which have a low probability, the beam is applied, and any leafs which are too unlikely are pruned. This is summarized in the following steps:

1. Expand the tree. For all of the current leafs, create new leafs that can be reached from the current state. Add the transition costs to the leafs. If the leaf to be created is the same as an existing state, keep only the most probable one.

2. Update the leaf probabilities by adding their probability of creating the current observation to their probability.

3. Renormalize the leaf probabilities by finding the probability of the most likely leaf, and subtracting this probability from all of the leafs. As probabilities are given in log terms, this corresponds to setting the most likely leaf to a probability of 0 (1 in non-log terms), and making all of the other nodes more probable by the same amount.

4. Apply the beam. Prune all leafs that fall below a threshold.

Note that this approach is still $O(N^2T)$ in the worst case, as N^2 new leafs may be created (and most of these merged) for any time frame. However, as many transitions have a probability of zero, many of these states will never be created, thus speeding up the process. Transitions have a probability of zero, if, for example, a node is in the first state of a phoneme, and the probability of it transitioning out of the phoneme is zero (as has been assumed in the HMM structure). In addition, this approach allows leafs with probabilities which fall below the beam width to be removed removed, thus allowing many more states never to be considered.

In order to effectively use the triphones generated earlier, the search needs to have knowledge of the phoneme following the current one. The search can thus be expanded slightly by considering the following phoneme as well. Figure 1-8 shows an expansion of this with unknown phonemes denoted with a "??" identifier.

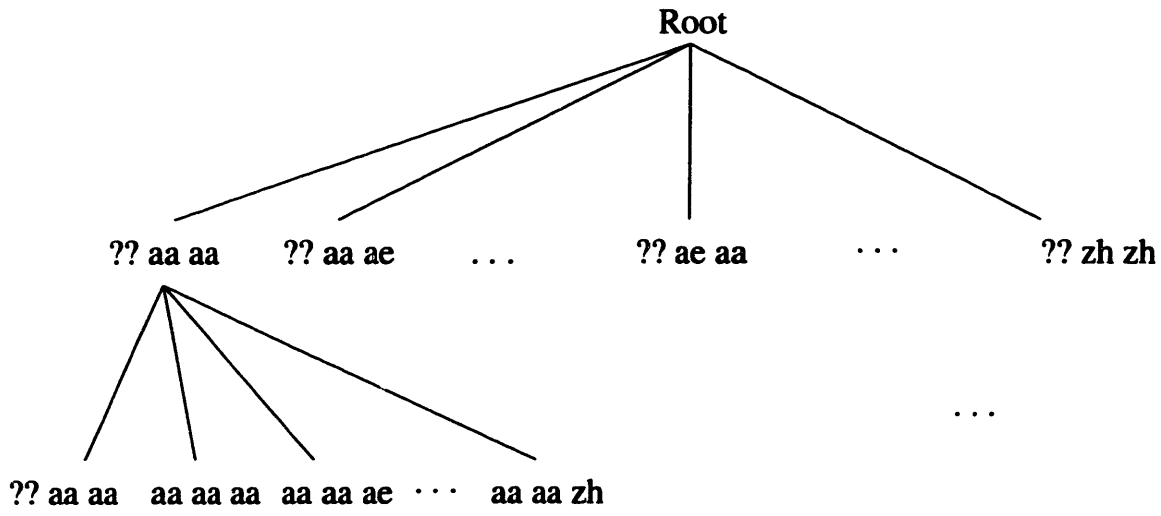

Figure 1-8: Tree for triphone recognition

Transition probabilities between phonemes can be included for better recognition. In this case, bigrams were used for the phoneme recognition, so $P(p_2|p_1)$ (the probability of phoneme p_2 following phoneme p_1) is calculated for all pairs of phonemes. This is calculated by taking all of the phoneme transcripts in the test section of the TIMIT database, calculating the number of times p_2 follows p_1, and dividing by the total number of times p_1 occurs:

$$P(p_2|p_1) = \frac{N(p_1, p_2)}{N(p_1)}.$$

where $N(p_1, p_2)$ is the number of times phoneme p_2 follows phoneme p_1 and $N(p_1)$ is the number of times phoneme p_1 occurs.

For word recognition, the number of words is usually large enough that not all possible word bigrams occur, and so a backoff needs to be done for the word bigrams from the word unigrams (see section 2.1.1). However, for phoneme level decoding, the number of phonemes (usually around 50) is small enough that there virtually all of the possible phoneme contexts occur. Thus such a backoff was not used.

In addition, a scaling factor is applied to the phoneme bigrams which allows them to be more comparable with the acoustic probabilities (see the section on language modeling in the next chapter). Otherwise, the recognition would be dominated by

the acoustic models, rather than a mix of the phoneme level language model and the acoustic model. The scaling factor exponentiates the probability of one phoneme following another:

$$P(p_2|p_1) \rightarrow P(p_2|p_1)^{\beta}$$

where β is the scaling factor. In this particular system, β was set to 8.0 which also seemed to balance insertion and deletion errors (insertion and deletion errors are covered in section 1.4.3).

1.4.3 The NIST scoring method

In order to ensure that the scoring of the recognizer is standardized, NIST (National Institute of Standards and Technology) provides an evaluation program [9]. As input, the scoring program takes a list of transcripts created by the recognizer (a "hypothesis" file), and a list of correct transcripts (a "reference" file) which are provided with the test data. Next, the program aligns the transcripts from the hypothesis and reference file in such a way that the total number of errors is minimized. Errors include insertion, deletion and substitution errors all of which are weighted equally. The program then calculates results for each speaker including the percent correct, the percent accuracy, and the percentage of insertion, deletion, and substitution errors. The total error is the sum of the deletion, insertion, and substitution errors. The percent correct is the number of phonemes which were the same after alignment, while the percent accuracy is the percent correct minus the insertion errors.

1.4.4 Recognition Results

The following results were run on the entire TIMIT "test" set excluding the "SA" sentences. Including the "SA" sentences produces abnormally good results as these sentences all have similar phonetic context. Phoneme bigrams were used as discussed previously and HMMs trained using the TIMIT training data. The baseline system was created with the following parameters:

Using these settings, the following phoneme level accuracies were reached: These

Table 1.1: Basic parameters for HMM training

Merge threshold for clustering algorithm	0.3
Minimum counts for clustering algorithm	200
Factor to divide overall variance by for floor variances	10.0
Phoneme language model scaling factor	8.0

Table 1.2: Baseline results

correct	accuracy	substitution	insertion	deletion	total error
72.4%	66.0%	20.6%	6.9%	6.4%	34.0%

results are competitive with other phoneme recognition results using continuous density context-dependent HMMs (for example, [4], [7], and [1]). In addition, several other runs were made to determine how parameter choices affected recognition.

One choice was to determine what effect the liftering (discussed earlier in the section on the front end) had on the recognition accuracy. Using the same set of parameters, the recognition was run with and without liftering, and the following results were obtained: As can be seen by this table in comparison to the baseline

Table 1.3: Comparison of recognition accuracy with and without liftering

	correct	accuracy	substitution	insertion	deletion	total error
with liftering	73.3%	66.9%	20.4%	6.3%	6.4%	33.1%
without liftering	73.1%	67.0%	20.6%	6.3%	6.1%	33.0%

results, the percentage correct increased by .2% while the accuracy decreased by .1%, showing that the liftering has little, if any effect. However, since the extra processing time was minimal, the liftering was left in by default.

Another exploration of parameters dealt with the effect of frame boundaries on phonemes. Before the baseline results were obtained, there was a bug in the assignment of frames into phonemes. If the phoneme boundary for the end of phoneme "aa" and the beginning of phoneme "b" was between frame 17 and 18, either frame

17 would be assigned to both phonemes or frame 18 would be assigned to both phonemes. The recognition accuracy of this system is given in table 1.4. Apparently,

Table 1.4: Results with some frames duplicated

correct	accuracy	substitution	insertion	deletion	total error
73.3%	66.9%	20.4%	6.3%	6.4%	33.1%

duplicating frames here improved recognition accuracy by almost 1%, increasing the percentage correct and decreasing the total errors. The reason that the results improved is believed to be because of the placement of the phoneme boundaries in the TIMIT database.

The TIMIT database consists of phoneme level information which is hand segmented. However, either these are a few frames off (a frame is only 10 ms., so this is entirely possible), or the boundaries are not modeled well by the HMMs being used. One way to alleviate this problem is to train models using the hand segmented data, re-segment the utterances using the trained models, and then to retrain the models using the new segmentation.

Re-segmenting the utterances is done using a forced alignment algorithm. This takes as input a set of precomputed models, an utterance to segment, and a list of the phonemes that occurred in the utterance (in order). Decoding is done similarly to the tree search described earlier, but the only phoneme it can transition to is the next phoneme in the sequence. Thus the output of this phoneme level forced alignment program is the sequence of phonemes given it with new frame numbers attached. In general, most boundaries stayed the same, but occasionally boundaries would shift by a few frames.

Using this program, all of the training utterances were re-segmented using the baseline models, and then new phoneme models were retrained from these alignments. These new phoneme models were then run on the "test" data and the following results from table 1.5 were obtained.

As shown in table 1.5, the results obtained here are slightly better than the results

Table 1.5: Results after re-segmenting training utterances

correct	accuracy	substitution	insertion	deletion	total error
73.9%	66.9%	20.2%	5.9%	6.8%	32.9%

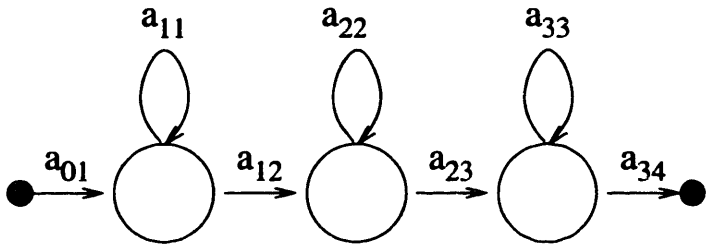

Figure 1-9: Left to right HMM structure with no state skips.

where duplicated frames were included (table 1.4). Thus a reasonable explanation to the increase in performance is that the duplicate frames slightly offset the slight inaccuracies in the phoneme boundaries, and that once these are fixed, the duplicate frames are no longer necessary. To test this theory, new models were trained using the same segmentation but with duplicate boundaries. The results are listed in table 1.6. This shows that recognition performance decreased when duplicate frames were

Table 1.6: Recognition accuracy with re-segmenting and then duplicating frames

correct	accuracy	substitution	insertion	deletion	total error
72.9%	67.0%	20.4%	6.7%	5.9%	33.0%

introduced when the utterances are properly segmented (as expected).

The last change of parameters was to change the HMM structure itself. A set of HMMs were trained with no state skips, as shown in figure 1-9. Training results were run with this HMM structure as opposed to that with state skips (see figure 1-4), and the results in table 1.7 were obtained. As can be seen from this, the main difference between the two is the insertion error dropped by almost 1% when no state skips were allowed. This shows that HMMs without state skips do a better job of modeling phone duration, which is often included in the decoding. In terms of substitution and deletion errors, the two were virtually identical.

Table 1.7: Comparison of recognition accuracy for HMMs with and without state skips

	correct	accuracy	substitution	insertion	deletion	total error
with state skips	73.9%	67.1%	20.2%	5.9%	6.8%	32.9%
without state skips	74.0%	67.9%	20.2%	5.9%	6.0%	32.1%

In order to compare the recognition of the triphones (context dependent) HMMs to the uniphones (context independent) HMMs, one final test was run with the uniphone HMMs. Run on the same task, the recognition results are listed in table 1.8. The

Table 1.8: Comparison of recognition accuracy for uniphone and triphone HMMs

	correct	accuracy	substitution	insertion	deletion	total error
triphone	74.0%	67.9%	20.2%	5.9%	6.0%	32.1%
uniphone	70.2%	64.2%	22.0%	7.7%	6.1%	35.8%

results show that phoneme recognition accuracy for uniphones was about 4% lower than that achieved by triphones. Thus training more model parameters using the contextual information of surrounding phonemes was beneficial.

Chapter 2

Extension to Word Recognition

In this chapter, the use of the acoustic models discussed in the first chapter are used to recognize English words. First, basic word recognition is discussed including the use of a basic language model. Next, this recognition is applied to a specific task, and descriptions of the language model and acoustic models used for recognition are discussed for the specific task. Finally, the recognition results and a discussion of the results are presented.

2.1 Basic word recognition

A reasonable goal for a word recognition system is to maximize $P(\mathbf{W}|\mathbf{O})$, that is the probability of $\mathbf{W}$, a word string, given $\mathbf{O}$ the acoustic observations. While this may not be optimal in all cases, it provides a reasonable starting point. For instance, this assumes that all words are equally important which is usually not the case. Using Bayes' law,

$$\max P(\mathbf{W}|\mathbf{O}) = \max P(\mathbf{O}|\mathbf{W})P(\mathbf{W})$$

The first term, $P(\mathbf{O}|\mathbf{W})$ can be calculated from the acoustic models described earlier, while the second term $P(\mathbf{W})$ is computed from the language model.

2.1.1 The language model

The goal of a language model is to provide the recognizer with estimates of $P(\mathbf{W})$ the probability of a word string. However, if there are N words in the vocabulary and the word string $\mathbf{W} = W_1, W_2, W_3, ..., W_M$ is of length M, then there are N^M possible word probabilities that would need to be stored. For vocabularies of at least a few hundred words, the storage space required quickly becomes immense. Another issue is that these probabilities are typically trained from a large corpus of text. There may be many word sequences that do not occur in the training text which are possible word sequences. For these reasons, it is necessary to develop an approximation which can be used to calculate the probability of a word sequence.

The approximation typically used is that of an n-gram model. For example, using a unigram model,

$$P(W_1, W_2, W_3, ..., W_M) = P(W_1)P(W_2)P(W_3)...P(W_M)$$

which assumes that the probability of any word does not depend on any of the words before it. While this is clearly a bad assumption for natural speech, it can be improved by using a higher level n-gram model such as a trigram or bigram model. The bigram model,

$$P(W_1, W_2, W_3, ..., W_M) = P(W_1)P(W_2|W_1)P(W_3|W_2)...P(W_M|W_{M-1})$$

assumes that the probability of a word depends only on the word directly before it. Trigrams and other higher level n-grams can be used to obtain more accurate word probabilities.

A difficulty with this approach is that it that the amount of information required for a language model is still quite large. For a bigram language model, it is still possible to have a bigram $P(W_2|W_1)$ which does not occur in the training corpus and yet is still a possible word combination. One way to address this problem is through the use of discounted probabilities.

Consider a vocabulary of five words $\mathbf{W} = \{A, B, C, D, E\}$ and their associated bigrams given the context A: $P(A|A)$, $P(B|A)$, $P(C|A)$, $P(D|A)$, $P(E|A)$. Suppose that three of the bigrams, $P(A|A)$, $P(B|A)$, and $P(C|A)$, all occur in the training corpus, but $P(D|A)$ and $P(E|A)$ do not. Discounting works by taking some of the probability mass from the given bigrams (P(Disc)) and assigning it to the unseen bigrams. As not much is known about the bigrams, it is logical to assign it in proportion to the unigram values:

$$P(\text{Disc}) = \alpha_A P(D) + \alpha_A P(E)$$

where α_A is the "back-off weight" for the unigram A. Thus the assigned bigram probabilities are given as $P(D|A) = \alpha_A P(D)$ and $P(E|A) = \alpha_A P(E)$, and the equation asserts that the probability mass taken from the known bigrams of context A (P(Disc)) is assigned to the unknown contexts.

The language models used here were calculated using Good Turning discounting [6]. The discounting applied here is calculated as follows:

$$d(r) = \frac{(r+1)n(r+1)}{rn(r)}$$

where $n(r)$ is the number of events which occur r times. There are several other methods for calculating the discounted probability, and these can be found in [6].

2.1.2 Word recognition, beam search

The approach implemented for word recognition is very similar to that developed for phoneme recognition in the previous chapter, except that the basic recognition unit is a word. Words are made from concatenation of phonemes as supplied by a phonetic dictionary. States in the tree consist of a word, a phoneme in the word, and a state within the phoneme. Transitions can then can take place on several different levels:

1. Between states in a phoneme model. The current state in a phoneme can change according to the states allowed by the transition matrix in the HMM.

The transition probability is that given by the transition matrix.

2. Between phonemes in a word. If the transition matrix allows a transition out of the phoneme model, it transitions to the next phoneme in the word. The possible starting states in the next phoneme are those allowed by its transition matrix. The transition probability is the sum of the transitions required (the transition to get out of the current phoneme, and the transition to get into the next phoneme).

3. Between different words. If the transition allows an exit from the last phoneme of a word, it can transition to a new word. The transition probability is the sum of the probability of transitioning out of the last phoneme, the probability of transitioning to the new word (supplied by the language model based on the current word), and the probability of transitioning into the first phoneme of the new word.

One difficulty immediately encountered in the recognition is the weighting of the acoustic and language models. The probabilities given by the language model and the acoustic model are typically on different levels. The acoustic model probabilities are typically much less than the language model probabilities, and thus count for much more than the language model. In order to counteract this, a weight is applied to the language model probabilities so that they are more important,

$$P(W_1|W_2) \rightarrow P(W_1|W_2)^{\alpha}$$

where here α is the language model scaling factor. As everything is done in log probabilities, this corresponds to multiplying the language model by a weight before combining it with the acoustic probabilities.

Another factor which is used is the "word insertion" penalty. This factor is applied primarily to adjust for changes in the speaking rate. Each time a transition to a new

word occurs, this factor is multiplied by the word insertion penalty, β:

$$P(W_1|W_2) \rightarrow \beta P(W_1|W_2)^{\alpha}$$

where β is the word insertion penalty. Low values of this parameter allow transitions to take place with fewer penalties for fast speaking rates. High values, on the other hand, condition the language model toward slow speakers. This parameter is usually adjusted by attempting to balance the word insertion and deletion errors after the recognizer is run on a test corpus.

Other than these parameters, the procedure used is basically the same as that for phoneme recognition:

1. Expand the tree. For all of the current leafs, create new leafs that can be reached from the current state. Add the transition costs to the leafs, including the cost from the language model if appropriate. If the leaf to be created is the same as an existing state, keep only the most probable one.

2. Update the leaf probabilities by adding their probability of creating the current observation to their probability.

3. Renormalize the leaf probabilities by assigning the most probable leaf a probability of 0 in log probability (corresponding to a probability of 1), and scaling all of the other nodes appropriately.

4. Apply the beam. Prune all leafs that fall below a threshold.

At the end of this procedure, a simple backtrace starting from the most probable leaf provides the best guess as to the most likely word sequence.

2.2 Task specific recognition

In order to evaluate the methods given above for word recognition, a specific task was created that would supply the recognizer with a corpus of medium vocabulary

spontaneous speech. In this task, subjects were given three puzzles consisting of several magnets and different colored balls which would move deterministically when the puzzle was activated. To solve a puzzle, magnets needed to be placed in locations around the screen so that when the simulation started, the balls would go to the places required by the solution. When the simulation stopped, the position of all objects would reset and the user would be able to adjust the position of the magnets. The physics simulation would cause the balls to drop as if under the presence of gravity when the puzzle was executed. Magnets would repel the balls as they dropped, with a force inversely proportional to their distance from the magnet. [1]

The experiment was a Wizard of Oz setup where commands given by the subject were responded to by an experimenter sitting in another room. Subjects were seated in a room and could were instructed to use voice commands to manipulate the objects. The setup allowed the collection of unscripted, unprompted speech as people attempted to solve the puzzles. An example screen shot is given in figure 2-1. This shows that even for this simple task many words occured only once. The data collected for use with the recognition system was taken for one hour per person in a single session, and 13 sessions were included. Utterances of single word commands (such as "start" and "stop") were thrown out so as not to artificially inflate the recognition accuracy. The vocabulary size consisted of 523 words, and a graph of the counts of the number of times words occur is given in figure 2-2. The complete word list including the number of word counts is given in 5. The collected speech generally included simple commands. Several successive commands from session 5 are listed below as a representative sample:

1. "OK, move the magnet to the left a little bit." (Wizard moves magnet to the left slightly).

2. "Move it to the right a little bit." (Magnet moves a small amount to the right)

3. "Put the second magnet in the upper right hand corner." (The second magnet moves to the upper right hand corner).

[1] Peter Gorniak developed the simulator and one of the puzzles used for the experiment.

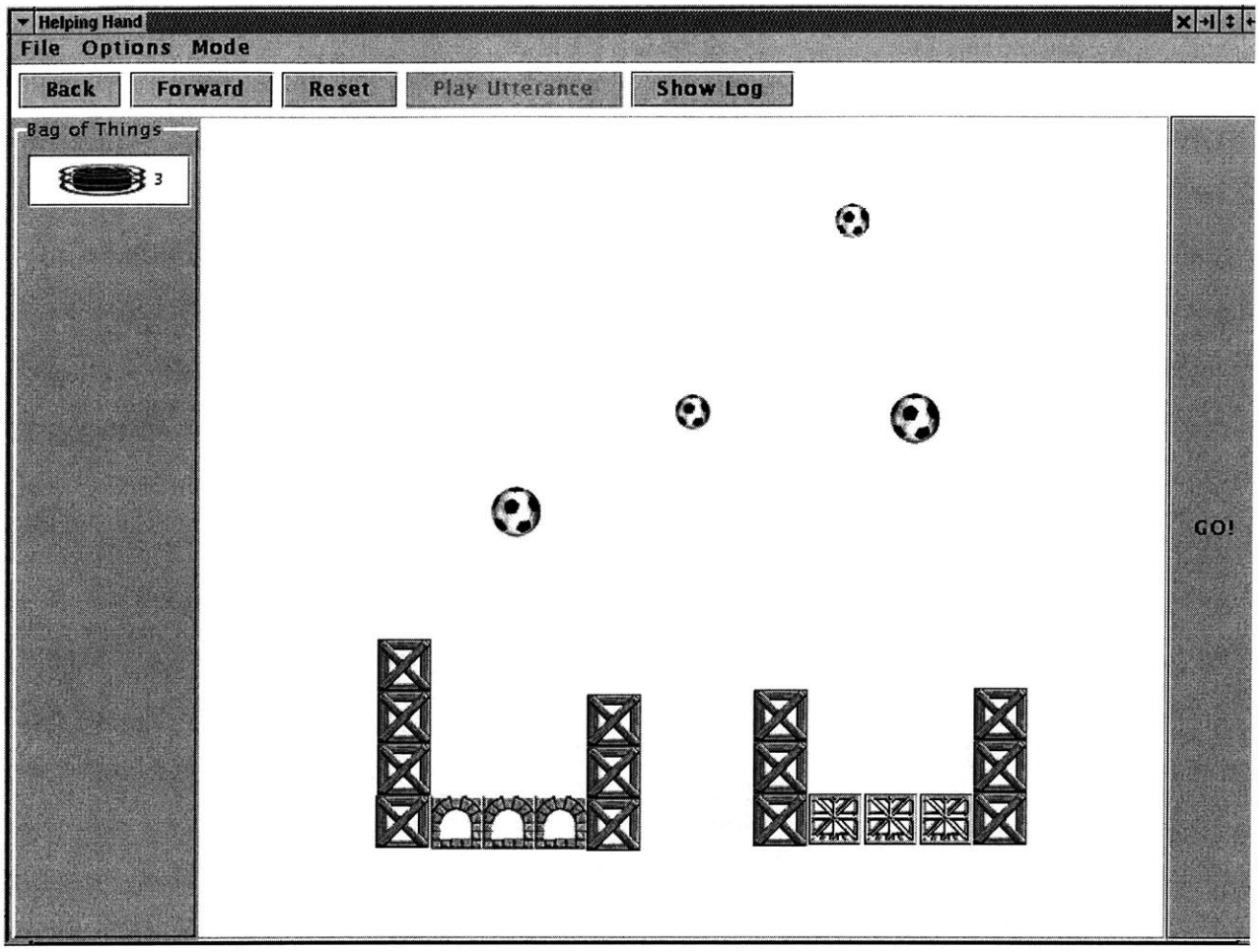

Figure 2-1: The first puzzle. The goal was to move the magnets so as to cause the red balls to fall into the right bin and the blue balls to fall into the left bin.

4. "Move it to the left a little bit." (The second magnet moves to the left slightly).

5. "Go." (Simulation starts).

6. "Stop." (Simulation stops and resets).

7. "Move the second magnet to the left a little more." (The second magnet moves to the left).

8. "Put the third magnet above the small blue ball." (The third magnet moves above the small blue ball).

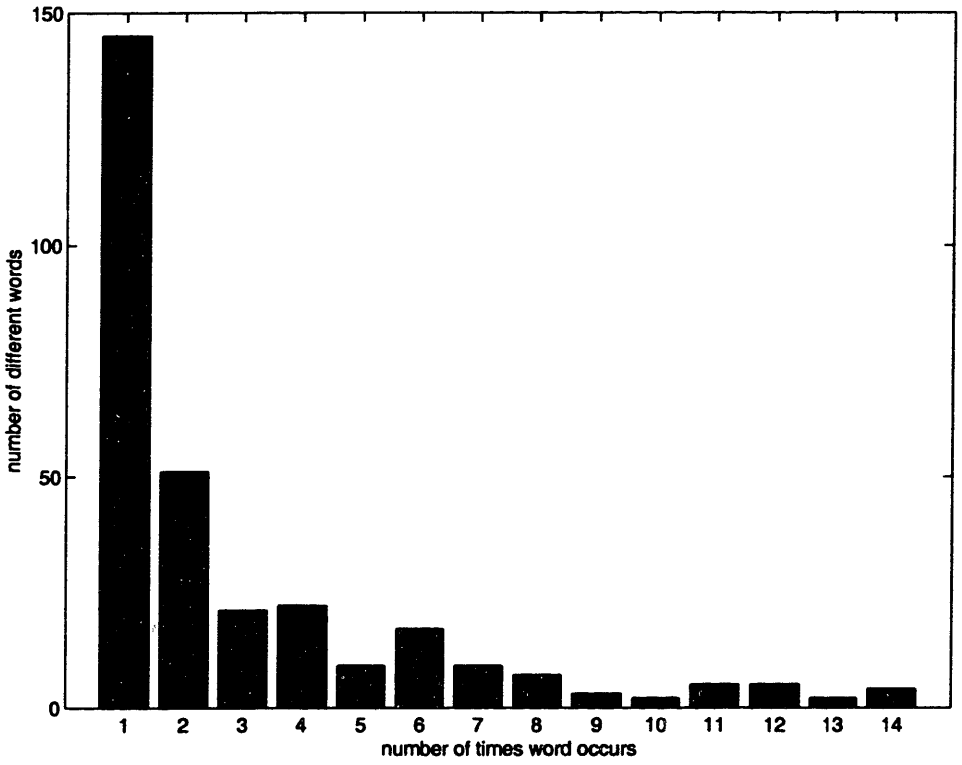

Figure 2-2: The number of times words occur. Most words occur only one or two times in the corpus.

2.2.1 Training of the acoustic models used for recognition

The acoustic models used for this task were trained using the same methods described earlier. However, for effective recognition, more data is needed to train the acoustic models (results in section 2.3 show a comparison between acoustic models trained only on TIMIT, and those trained on the Wall Street Journal database). As a result, the acoustic models used for this task were trained on the WSJ1 (Wall Street Journal Phase 1) corpus. This corpus consists of 78,000 training utterances (around 73 hours of speech) and is stored on 34 CDROMs. Unlike the TIMIT data, this data only comes with a word-level transcription, with no time boundaries. In order to train acoustic models using this corpus, such boundaries first had to be created.

The creation of phoneme level boundaries from the supplied transcript is done using a forced alignment procedure. This is done using the same tree based algorithm developed for the word and phoneme level recognition. The piece of extra

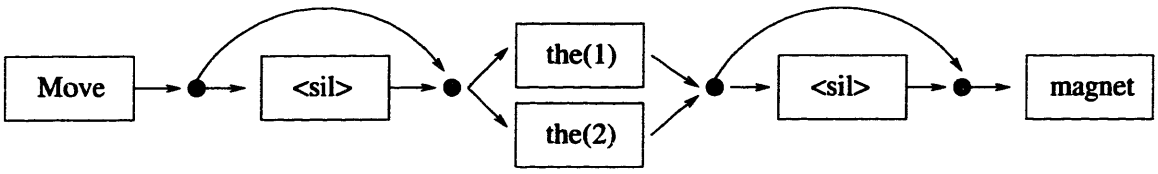

Figure 2-3: Possible word transitions for forced alignment with two possible pronunciations for the word "the", and optional silence (<sil>) between words

information is that the transcription provides the next possible word, or in the case of multiple pronunciations, several words. Optional silence is created between words, as depicted in figure 2-3. The transition constraints provided by the transcription allowed accurate determination of the phoneme boundaries. Although no scientific measure was used to determine the accuracy of boundaries, it was observed after listening to several utterances that the phonetic boundaries lined up reasonably well. The acoustic models used for the forced alignment were those trained on the TIMIT data. Thus the TIMIT database though it provides a limited amount of data, allowed the bootstrapping of more effective models.

After the forced alignment was completed for the utterances in the WSJ1 database, context dependent phonemes were trained from the accumulated information. The training included initialization via segmental k-means, as well as 5 rounds of EM (Baum-Welch) for the uniphones, left and right biphones, and triphones, as described previously.

2.2.2 Training of the Language Models used for recognition

The given task was specific enough that specific language models needed to be generated using the accumulated text. To accomplish this with a limited amount of data, the following technique was used:

1. Create a corpus of text from all of the subjects except for the one on which the recognizer is running.

2. Create a language model for the current subject using the collected text and the CMU statistical language modeling toolkit (see [5] for details).

3. Repeat for all of the subjects.

This method allowed the creation of speaker independent language models that were related well to the task. Originally, it was thought that a linear interpolation between this task specific language model, and a more general language model might provide better results.

For small vocabularies, this could be done by computing the word-level bigrams for all possible word pairs in the vocabulary, and mixing the two together:

$$P(W_1|W_2) = \alpha P^1(W_1|W_2) + (1-\alpha)P^2(W_1|W_2)$$

where $P^1(W_1|W_2)$ gives the bigram probability calculated from the general language model, and $P^2(W_1|W_2)$ gives the bigram probability from the task specific language model. Here α determines the amount of back-off. A large value of alpha will depend almost exclusively on the general language model, while a small value will depend mostly upon the task specific language model.

To evaluate this, the perplexity of the language models was evaluated. The perplexity of a language model depends on a supplied text which was held out from the training of a language model:

$$LP = \lim_{n\to\infty} \frac{1}{N} log P(W_1, W_2, W_3, ..., W_N)$$

where LP is the log of the perplexity. For a bigram language model,

$$LP = \lim_{n\to\infty} \frac{1}{N} log\left(P(W_1)P(W_2|W_1)P(W_3|W_2)...P(W_N|W_{N-1})\right)$$

The perplexity was computed for several different values of α, and the results can be seen in figure 2-4.

As shown in figure 2-4, the best weight to use is $\alpha = 0$, namely to depend only on the task specific language model. The general language model used for these calculations was also created by the CMU statistical language modeling toolkit, but was created from radio broadcast texts. As a result, the general language model was

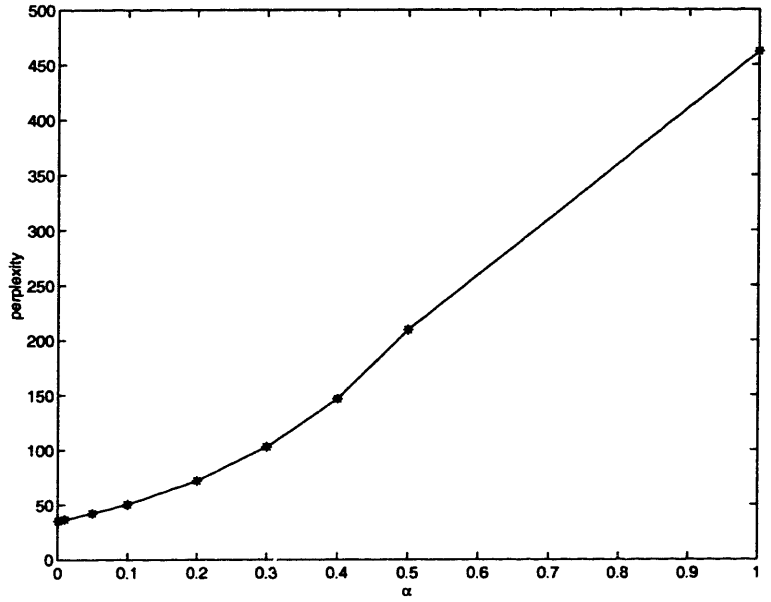

Figure 2-4: Graph of perplexity for varying levels of language model back-off (α).

a poor predictor of words, and thus did not help in reducing the perplexity as much as the task specific language models.

2.3 Recognition results

The recognizer was run on 11 of the one hour sessions using the language model and the acoustic models described above. The perplexity of the language model on the test set is 36, and the vocabulary consists of 523 words. In order to set the language model scaling factor, the system was run on the test set with various values of this constant. Figure 2-5 shows these results for different values of the language model scaling factor. The best recognition was acheived when the language scaling factor was set to 20. This value was somewhat higher than expected because of the natural constraints of the task. Although there were no instructions ahead of time to the subjects detailing what kind of language they should use (in fact, they were encouraged to use whatever came naturally to them), the task itself was constrained enough that similar constructs occurred.

It is impressive to note the effect with which the language model can have upon

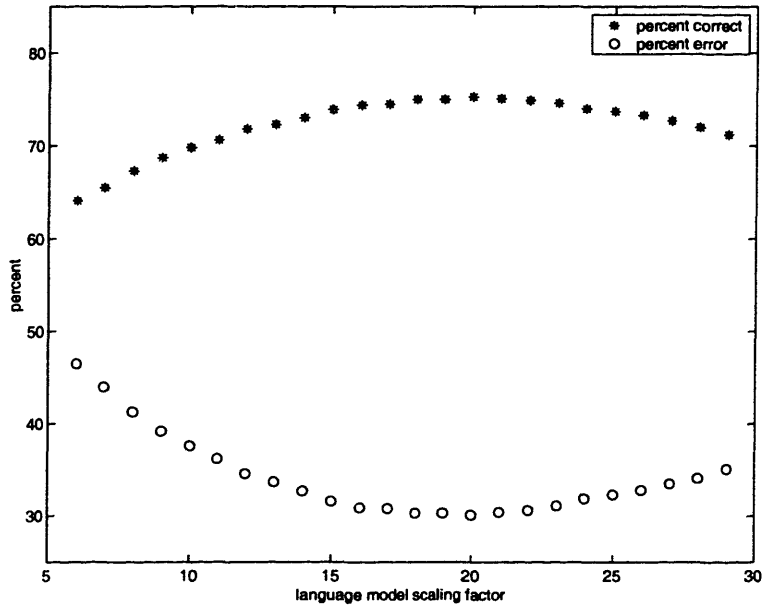

Figure 2-5: Recognition results for recognizer run at different language model scaling rates.

recognition for this specific task. With no language model, recognition has an error rate of around 50%, while with the scaling on the language model set correctly, recognition achieves an error rate of almost 30%. The full table of results is listed in table 2.1.

As can be seen from these results, there was a large variance in recognition accuracy ranging from 49.7% to 88.9%. Much of this is related to speaker dependent factors. While some speakers spoke quite clearly and evenly, others had large variations in speaking rate, as well as possibly backing up to correct themselves. In addition, some speakers may have been acoustically clear, and yet used vocabulary which was unique. Here the language model taken from the other speakers may have depressed their scores as well.

For example, speaker 10 (88.9% accuracy) spoke very clearly and evenly and also used relatively simple commands. The speech recorded here sounded more like dictated speech, and things were often overly annunciated. The commands used were often simpler than other users, allowing the language model to compute accurate probabilities for the speech. On the other hand, speaker 3 (55% accuracy) spoke with a wide variety of inflections and variability. The beginning of utterances often were

Table 2.1: Results for the recognition task

speaker number	correct	accuracy	substitution	insertion	deletion	total error
1	78.1	75.3	16.0	2.8	5.9	24.7
2	71.1	66.0	12.9	5.1	16.0	34.0
3	62.4	55.0	27.9	7.5	9.7	45.0
4	84.7	79.1	12.6	5.6	2.7	20.9
5	60.3	58.3	26.3	2.0	13.3	41.7
6	77.9	74.8	15.4	3.1	6.8	25.2
7	61.9	56.0	28.0	5.9	10.2	44.0
8	87.8	84.2	10.4	3.6	1.9	15.8
9	79.6	65.5	16.0	14.1	4.4	34.5
10	91.7	88.9	4.5	2.8	3.8	11.1
11	59.4	49.7	30.3	9.7	10.2	50.3
12	70.6	62.7	20.8	7.9	8.6	37.3
13	60.0	53.2	28.1	6.8	11.9	46.8
total	73.3	67.6	18.9	7.8	5.7	32.4

stretched out (as if to give time to think), while by the end the subject was speaking rapidly.

To compare the acoustic units used for this recognition with those trained on the TIMIT database, the recognition was run again with new acoustic models. These results are listed in table 2.2. This shows that the extra information from the WSJ

Table 2.2: Recognition results for acoustic units trained on the TIMIT and WSJ databases

database	correct	accuracy	substitution	insertion	deletion	total error
WSJ	73.3	67.6	18.9	7.8	5.7	32.4
TIMIT	57.8	49.0	32.8	9.4	8.8	51.0

database greatly improved recognition as the recognition accuracy improved by over 20%.

Chapter 3

Conclusions

In this thesis, a method for spontaneous speech recognition has been outlined and the results presented. The acoustic models were combined with the language model and word recognition was performed. In attempting to increase word recognition, several important features were required.

3.1 Importance of Context Dependant Acoustic Models and Task-Specific Language models

First of all, the use of context dependent phonemes increases phoneme recognition accuracy by about 4% for models trained and tested on the TIMIT database. While phoneme recognition results were not done with the WSJ database, the availability of more data should further increase the performance of the triphones compared to the uniphones, as the clustering algorithm used for triphones allows new model parameters to be trained dependent upon the available data. In contrast, the uniphones depend on only a fixed number of parameters independent of the data size.

In addition, use of an accurate language model for this task was very important. Recognition accuracy improved by almost 20% when the scaling of the language model was adjusted appropriately. Also, as can be seen in figure 2-4, the use of a task specific language model helped in reducing the perplexity from 460 to 36. Using

a more general language model computed from news broadcasts only increased the perplexity and would hurt recognition.

3.2 Future Directions

Many of the results presented here could benefit from further work. First of all, improvements in the recognizer could be very beneficial. Supporting a trigram language model instead of just bigrams could decrease the perplexity of the language models significantly. In addition, the dictionary which was used for the decoding of words into phoneme sequences could be augmented by including other possible pronunciations. This would better allow for speakers who may pronounce the same word slightly differently or with different accents. Another important change which should help recognition would be the acoustic modeling of function words such as "the", "a", and "and". These words occur so often that training HMMs for these specific words would be beneficial, rather than treating them as a concatenation of several phonemes.

Chapter 4

Appendix I: List of phonetic units used for the TIMIT database

Stops

Symbol	Example
b	bee
d	day
g	gay
p	pea
t	tea
k	key

Affricatives

Symbol	Example
jh	joke
ch	choke

Fricatives

Symbol	Example
s	sea
sh	she
z	zone
zh	azure
f	fin
th	thin
v	van
dh	then

Nasals

Symbol	Example
m	mom
n	noon
ng	sing
em	bottom
en	button

Semivowels and Glides

Symbol	Example
l	lay
r	ray
w	way
y	yacht
hh	hay
el	bottle

Vowels

Symbol	Example
iy	beet
ih	bit
eh	bet
ey	bait
ae	bat
aa	bott
aw	bout
ay	bite
ah	but
ao	bought
oy	boy
ow	boat
uh	book
uw	boot
er	bird
ax	about
ix	debit
axr	butter

Others

Symbol	Description
sil	silence
sp	short pause

Chapter 5

Appendix II: List of word frequencies for recognition task

1. THE (1782)
2. MAGNET (835)
3. MOVE (697)
4. TO (449)
5. LEFT (416)
6. RIGHT (367)
7. A (303)
8. BALL (281)
9. OF (257)
10. LITTLE (197)
11. DOWN (159)
12. IT (151)
13. BIT (147)
14. AND (143)
15. UP (138)
16. MIDDLE (138)
17. SMALL (125)
18. BOTTOM (124)
19. BLUE (124)
20. THAT (121)
21. OK (117)
22. PLACE (114)
23. ABOVE (111)
24. TOP (100)
25. GO (88)
26. RED (86)
27. IN (86)
28. PUT (83)
29. OBJECT (78)
30. PLEASE (77)
31. STOP (75)
32. ON (71)
33. SO (70)
34. SCREEN (69)
35. MOST (63)
36. TWO (59)
37. SLIGHTLY (59)
38. IS (57)
39. TARGET (56)
40. PUZZLE (56)
41. DIRECTLY (53)
42. INCH (52)
43. LOWER (50)
44. CORNER (50)
45. MAGNETS (48)
46. UNDER (45)
47. AN (45)
48. JUST (41)
49. BETWEEN (39)
50. BELOW (38)
51. MORE (37)
52. NEXT (35)
53. LARGE (35)
54. CURRENTLY (34)
55. I (33)
56. VERY (31)
57. THIRD (30)
58. SECOND (29)
59. ONE (29)
60. AT (29)
61. THAT'S (28)
62. HAND (28)
63. OKAY (27)
64. BACK (26)
65. WAY (25)
66. ALL (25)
67. NEW (24)
68. IT'S (24)
69. GET (23)
70. EXECUTE (23)
71. HALF (21)
72. FIRST (21)

73. BY (20)
74. AGAIN (20)
75. REMOVE (19)
76. AS (17)
77. QUARTER (16)
78. PINK (16)
79. NO (16)
80. LAST (16)
81. CENTER (16)
82. WALL (15)
83. UPPER (15)
84. TINY (15)
85. OTHER (15)
86. BORDER (15)
87. TAKE (14)
88. INCHES (14)
89. FAR (14)
90. BUT (14)
91. TOWARD (13)
92. FURTHER (13)
93. UNDERNEATH (12)
94. TARGETS (12)
95. LOWEST (12)
96. BASKETS (12)
97. ANOTHER (12)
98. WAS (11)
99. SIDE (11)
100. SAME (11)
101. RID (11)
102. CAN (11)
103. THIS (10)
104. OH (10)
105. WHERE (9)
106. LET'S (9)
107. ALONG (9)
108. YOU (8)
109. OUT (8)
110. NOT (8)
111. LEVEL (8)
112. FROM (8)
113. EDGE (8)
114. ARE (8)
115. TOUCHES (7)
116. RAISE (7)
117. OVER (7)
118. GOOD (7)
119. DO (7)
120. CLOSER (7)
121. CENTIMETER (7)
122. BIG (7)
123. BASKET (7)
124. WHAT (6)
125. UNDO (6)
126. THREE (6)
127. THEY (6)
128. THEN (6)
129. SOLVE (6)
130. SOCCER (6)
131. OR (6)
132. NOW (6)
133. NEAR (6)
134. MAKE (6)
135. L (6)
136. HAVE (6)
137. BLOCKS (6)
138. BALLS (6)
139. ALRIGHT (6)
140. ADD (6)
141. WAIT (5)
142. TOUCHING (5)
143. THEM (5)
144. SEE (5)
145. ME (5)
146. LIKE (5)
147. HIGHER (5)
148. EACH (5)
149. ABOUT (5)
150. YEAH (4)
151. WORK (4)
152. WHITE (4)
153. WALLS (4)
154. THATS (4)
155. START (4)
156. SQUARE (4)
157. SAY (4)
158. ONTO (4)
159. ITS (4)
160. HOW (4)
161. HIGHEST (4)
162. HALFWAY (4)
163. GRAY (4)
164. FIGURE (4)
165. FARTHER (4)
166. BRACE (4)
167. BOTH (4)
168. BIN (4)
169. BEFORE (4)
170. ALMOST (4)
171. ACTUALLY (4)
172. WITH (3)
173. UNTIL (3)
174. TOWARDS (3)
175. THANK (3)
176. SHAPE (3)
177. RUN (3)
178. ROTATE (3)
179. POSITION (3)
180. MOVING (3)
181. LOT (3)
182. LOAD (3)
183. LESS (3)
184. KEEP (3)
185. INTO (3)
186. IMMEDIATELY (3)
187. HIT (3)
188. HEIGHT (3)
189. GIVE (3)
190. FOUR (3)
191. EVEN (3)
192. AWFUL (3)
193. WHICH (2)
194. WHEN (2)
195. WELL (2)
196. VERTICAL (2)
197. UPSIDE (2)
198. TURN (2)
199. TRYING (2)
200. TOO (2)
201. THINK (2)
202. THESE (2)
203. TELL (2)
204. SUCH (2)
205. STUPID (2)
206. SOME (2)
207. SITS (2)

208. SHAPED (2)
209. SHAFT (2)
210. RESTART (2)
211. REALLY (2)
212. PULLS (2)
213. POSSIBLE (2)
214. PART (2)
215. ONLY (2)
216. OFF (2)
217. NEAREST (2)
218. MEAN (2)
219. LINE (2)
220. LETS (2)
221. JOB (2)
222. INSIDE (2)
223. I'M (2)
224. HELP (2)
225. GOT (2)
226. GOES (2)
227. GOAL (2)
228. DISTANCE (2)
229. DELETE (2)
230. COLUMN (2)
231. CLOSE (2)
232. CENTIMETERS (2)
233. CAN'T (2)
234. BOWLING (2)
235. BENEATH (2)
236. BARRIERS (2)
237. BARRICADES (2)
238. BARRICADE (2)
239. AWAY (2)
240. AMOUNT (2)
241. AM (2)
242. ACTION (2)
243. ACROSS (2)
244. YOU'RE (1)
245. YOUR (1)
246. YES (1)
247. WILL (1)
248. WHY (1)
249. WHOA (1)
250. WHAT'S (1)
251. WE'RE (1)
252. WENT (1)
253. WEIRD (1)
254. WANT (1)
255. UPWARD (1)
256. UNIT (1)
257. UM (1)
258. UH (1)
259. TRY (1)
260. 'TILL (1)
261. THROUGH (1)
262. THOUGHT (1)
263. THOUGH (1)
264. THOSE (1)
265. THING (1)
266. THEY'RE (1)
267. THERE (1)
268. THAN (1)
269. TEN (1)
270. TARGET'S (1)
271. TAKEN (1)
272. SURROUND (1)
273. STRONGER (1)
274. STRAIGHT (1)
275. STACK (1)
276. SORRY (1)
277. SOMETHING (1)
278. SOLVED (1)
279. SIMULATION (1)
280. SHUT (1)
281. SHALL (1)
282. SESSION (1)
283. SAYING (1)
284. SAVER (1)
285. ROW (1)
286. ROOM (1)
287. RETURN (1)
288. RESET (1)
289. REMOVED (1)
290. REFER (1)
291. READ (1)
292. PROGRAM (1)
293. PEOPLE (1)
294. PAIRS (1)
295. PAIR (1)
296. PAIN (1)
297. ORIGINAL (1)
298. ONCE (1)
299. NINETY (1)
300. NEED (1)
301. MY (1)
302. MOVES (1)
303. MOVED (1)
304. MONITOR (1)
305. MILLIMETER (1)
306. MEANING (1)
307. MAYBE (1)
308. LOW (1)
309. LIFE (1)
310. LENGTH (1)
311. LEFTHAND (1)
312. LEDGE (1)
313. LEAVING (1)
314. LEAVE (1)
315. KAIYUH (1)
316. K (1)
317. JOSH (1)
318. I'VE (1)
319. INVERTED (1)
320. IMAGE (1)
321. I'D (1)
322. HURRAY (1)
323. HOVERING (1)
324. HOLE (1)
325. HITS (1)
326. HIGH (1)
327. HERE (1)
328. HEAD (1)
329. HAPPENS (1)
330. HANG (1)
331. GUYS (1)
332. GREY (1)
333. GREAT (1)
334. GONNA (1)
335. GETTING (1)
336. GAP (1)
337. GAME (1)
338. FURTHERMOST (1)
339. FUDGE (1)
340. FORM (1)
341. FOR (1)
342. FIVE (1)

343. FINISH (1)
344. FIND (1)
345. FIGURED (1)
346. FIELD (1)
347. FEET (1)
348. EVIL (1)
349. EVERYTHING (1)
350. EQUAL (1)
351. END (1)
352. EITHER (1)
353. EIGHTH (1)
354. EIGHT (1)
355. DON'T (1)
356. DONE (1)
357. DOING (1)
358. DOESN'T (1)
359. DOES (1)
360. DIRECTION (1)
361. DID (1)
362. DEGREES (1)
363. CURRENT (1)
364. CORNERS (1)
365. COMPLETELY (1)
366. COMMAND (1)
367. COMING (1)
368. COMES (1)
369. COME (1)
370. CLOSEST (1)
371. CLOCKWISE (1)
372. BROUGHT (1)
373. BOXES (1)
374. BOO (1)
375. BLOCK (1)
376. BINS (1)
377. BEN (1)
378. BEING (1)
379. BEEN (1)
380. BASES (1)
381. BARS (1)
382. BACKWARDS (1)
383. APPEARING (1)
384. APPEAR (1)
385. ALREADY (1)
386. ALLRIGHT (1)
387. AGAINST (1)

Bibliography

[1] Rabiner and Juang. *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[2] Jelinek, Frederick *Statistical Methods for Speech Recognition.* MIT Press, 1997.

[3] Oppenheim, Schafer, and Buck *Discrete-Time Signal Processing.* Prentice Hall, 1999.

[4] S.J. Young and P.C. Woodland, "State clustering in HMM-based continuous speech recognition," *Computer Speech and Language*, vol. 8, no. 4, pp. 369-94, 1994.

[5] P. Clarkson and R. Rosenfeld, "Statistical Language Modeling using the CMU-Cambridge Toolkit," *Proceedings ESCA Eurospeech 1997*

[6] S.M. Katz, "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer". *IEEE Transactions of Acoustics, Speech, and Signal Processing*, vol. ASSP-35, pp. 400-401, March 1987.

[7] K.F. Lee and H.W. Hon, "Speaker-independent phone recognition using hidden Markov models." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, pp. 1641-1648, November 1989.

[8] J.S. Garofolo, et all. "TIMIT Acoustic-Phonetic Continuous Speech Corpus". National Institute of Standards and Technology (NIST), 1990.

[9] National Institute of Standards and Technology, Retrieved 20 August 2001 <http://www.nist.gov>.