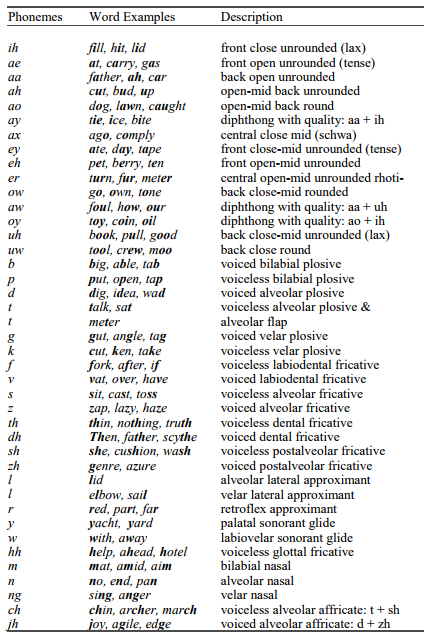
Phonetics is the part of linguistics that focuses on the study of the sounds produced by human speech. It encompasses their production (through the human vocal apparatus), their acoustic properties, and perception. There are three basic branches of phonetics, all of which are relevant to automatic speech recognition.

* *Articulatory phonetics* focuses on the production of speech sounds via the vocal tract, and various articulators
* *Acoustic phonetics* focuses on the transmission of speech sounds from a speaker to a listener
* *Auditory phonetics* focuses on the reception and perception of speech sounds by the listener.

The atomic unit of speech sound is called a *phoneme*. Words are comprised of one or more phonemes in sequence. The acoustic realization of a phoneme is called a *phone*. Below is a table of phonemes of U.S. English and common realizations.



One major way to categorize phonemes is into vowels and consonants.

**Vowels** can be distinguished by two attributes. First, they are voiced sounds, meaning that the airflow from the vocal chords into the mouth cavity is created by the vibration of the vocal chords at a particular fundamental frequency (or pitch). Second, the tongue does not in any way form a constriction of air flow during production. The placement of the tongue, lips, and jaw distinguishes different vowel sounds from each other. These different positions form different resonances inside the vocal tract called formants and the resonant frequencies of these formants characterizes the different vowel sounds.

**Consonants** are characterized by significant constriction of air flow in the airway or mouth. Like vowels, some consonants can be voiced, while others are unvoiced. Unvoiced phonemes do not engage the vocal cords and therefore do not have a fundamental frequency or pitch. Some consonant phonemes occur in pairs that differ only in wether they are voiced or unvoiced but are otherwise identical. For example, the sounds /b/ and /p/ are have identical articulatory characteristics (your mouth, tongue, jaw are in the same position for both) but the former is voiced and the latter is unvoiced. The sounds /d/ and /t/ are another such pair.

One important aspect of phonemes is that their realization can change dependent on the surrounding phones. This is called phonetic context and it caused by a phenomenon called coarticulation. The process of producing these sounds in succession changes their characteristics. Modified versions of a phoneme caused by coarticulation are called allophones.

All state of the art speech recognition systems use this context-dependent nature of phonemes to create a detailed model of phonemes in their various phonetic contexts.

**Syllables and words**

A syllable is a sequence of speech sounds, composed of a nucleus phone and optional initial and final phones. The nucleus is typically a vowel or syllabic consonant, and is the voiced sound that can be shouted or sung.

As an example, the English word “bottle” contains two syllables. The first syllable has three phones, which are “b aa t” in the Arpabet phonetic transcription code. The “aa” is the nucleus, the “b” is a voiced consonant initial phone, and the “t” is an unvoiced consonant final phone. The second syllable consists only of the syllabic cosonant “l”.

A word can also be composed of a single syllable which itself is a single phoneme, e.g. “Eye,” “uh,” or “eau.”

In speech recognition, syllable units are rarely considered and words are commonly tokenized into constituent phonemes for modeling.

**Syntax and Semantics**

Syntax describes how sentences can be put together given words and rules that define allowable grammatical constructs. Semantics generally refers to the way that meaning is attributed to the words or phrases in a sentence. Both syntax and semantics are a major part of natural language processing but neither plays a major role in speech recognition.

When we build and experiment with speech recognition systems is it obviously very important to measure performance. Because speech recognition is a sequence classification task (in constrast to image labeling where samples are independent), we must consider the entire sequence when we measure error.

The most common metric for speech recognition accuracy is the Word Error Rate (WER). There are three types of errors a system can make: a *substitution*, where one word is incorrectly recognized as a different word, a *deletion*, where no word is hypothesized when the reference transcription has one, and an *insertion* where the hypothesized transcription inserts extra words not present in the reference. The overall WER can be computed as

WER=Nsub+Nins+NdelNref

where Nsub,Nins,Ndel are the number of substitutions, insertions and deletions, respectively and Nref is the number of words in the reference transcription.

The WER is computed using a [string edit distance](https://en.wikipedia.org/wiki/Edit_distance) between the reference transcription and the hypothesized transcription. String edit distance can be efficiently computed using dynamic programming. Because string edit distance can be unreliable over a long body of text, we typically accumulate the error counts on a sentence-by-sentence basis and these counts are aggregated overall sentences in the test set to compute the overall WER.

In the example below, the hypothesis “how never a little later he had comfortable chat” is measured against the reference “however a little later we had a comfortable chat” to reveal two substitution errors, one insertion error, and one deletion error.

| **Reference** | **Hypothesis** | **Error** |
| --- | --- | --- |
| however | how | Substitution |
|  | never | Insertion |
| a | a |  |
| little | little |  |
| later | later |  |
| we | he | Substitution |
| had | had |  |
| a |  | Deletion |
| comfortable | comfortable |  |
| chat | chat |  |

In some cases, the cost of the three different types of errors may not be equivalent. In this case the edit distance computation can be adjusted accordingly.

Sentence error rate (SER) is a less commonly used evaluation metric which treats each sentence as a single sample that is either correct or incorrect. If any word in the sentence is hypothesized incorrectly, the sentence is judged incorrect. SER is computed simply as the proportion of incorrect sentences to total sentences.

Statistical significance testing involves measuring to what degree the difference between two experiments (or algorithms) can be attributed to actual differences in the two algorithms or are merely the result inherent variability in the data, experimental setup or other factors. The idea of statistical significance underlies all pattern classifications tasks. However, the way statistical significance is measure is task-dependent. At the center of most approaches is the notion of a “hypothesis test” in which there is a “null” hypothesis. The question then becomes with what confidence you can argue that the null hypothesis can be rejected.

For speech recognition the most commonly used measure to compare two experiments is called the Matched Pairs Sentence-Segment Word Error (MAPSSWE) Test, commonly shortened to just the Matched Pairs Test. It was suggested for speech recognition evalutions by [Gillick](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.296.4438" \t "_blank)*[et al.](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.296.4438" \t "_blank)* .

In this approach, the test set is divided up into segments with the assumption that errors in one segment are statistically independent from each other. This assumption is well-matched with typical speech recognition experiments where many test utterances are run through the recognizer one-by-one. Given the utterance-level error count from the WER computation described above, constructing a matched pairs test is straightforward. More details of the algorithm can be found in [Pallet *et al.*](https://doi.org/10.1109/ICASSP.1990.115546) .

Besides accuracy, there may be computational requirements that impact performance, such as processing speed or latency. Decoding speed is usually measured with respect to a real-time factor. A RTF of 1.0 means that the system processes the data in real-time, takes ten seconds to process ten seconds of audio.

Factors above 1.0 indicate that the system needs more time to process the data. For some applications, this may be acceptable. For instance, when creating a transcription of a meeting or lecture, it may be more important to take more time and produce accurate transcriptions than to get the transcriptions quickly.

When the RTF is below 1.0, the system processes the data more quickly than it arrives. This can be useful when more than one system runs on the same machine. In that case, multithreading can effectively use one machine to process multiple audio sources in parallel. RTF below 1.0 also indicates that the system can “catch up” to real-time in online streaming applications. For instance, when performing a remote voice query on the phone, network congestion can cause gaps and delays in receiving the audio at the server. If the ASR system can process data in faster than real-time, it can catch up after the data arrives, hiding the latency behind the speed of the recognition system.

In general, any ASR system can be tuned to tradeoff speed for accuracy. But, there is a limit. For a given model and test set, the speed-accuracy graph has an asymptote that is impossible to cross, even with unlimited computing power. The remaining errors can be entirely ascribed to modeling errors. Once the search finds the best result according to the model, further processing will not improve the accuracy.

**Bayes rule and the fundamental equation of speech recognition**

Speech recognition is cast as a statistical optimization problem. Specifically, for a given sequence of observations **O**={*O*1,…,*ON*}, we seek the most likely word sequence **W**={*W*1,…,*WM*}. That is, we are looking for the word sequence which maximizes the posterior probability *P*(**W**|**O**).\ Mathematically, this can be expressed as:

ˆ*W*=*argmaxWP*(*W*|*O*)

To solve this expression, we employ Bayes rule

*P*(*W*|*O*)=*P*(*O*|*W*)*P*(*W*)*P*(*O*)

Because the word sequence does not depend on the marginal probability of the observation *P*(*O*), this term can be ignored. This, we can rewrite this expression as

ˆ*W*=*argmaxWP*(*O*|*W*)*P*(*W*)

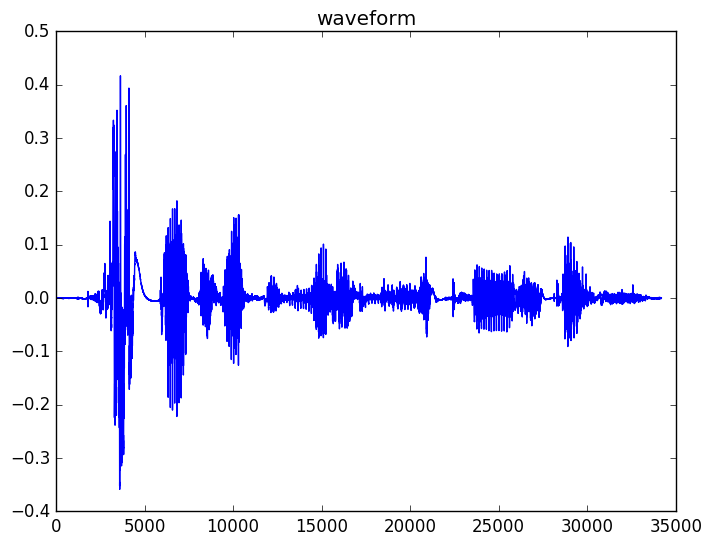
This is known as the *fundamental equation of speech recognition*. The speech recognition problem can be cast as a search over this joint model for the best word sequence.

The equation has a component P(*O*|*W*) known as an **acoustic model**, that describes the distribution over acoustic observations *O* given the word sequence *W*. The acoustic model is responsible for modeling how sequences of words are converted into acoustic realizations, and then into the acoustic observations presented to the ASR system. Acoustics and acoustic modeling are covered in Modules 2 and 3 of this course.

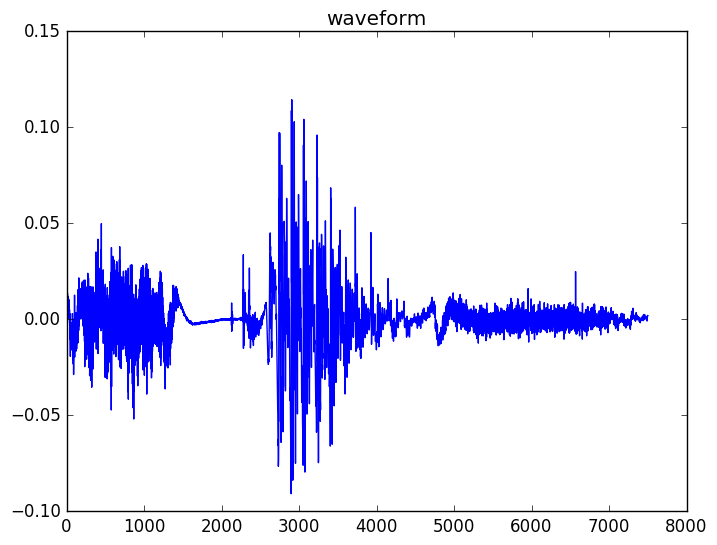
The equation has a component P(*W*) called a **language model** based solely on the word sequence *W*. The language model assigns a probability to every possible word sequence. It is trained on sequences of words that are expected to be like those the final system will encounter in everyday use. A language model trained on English text will probably assign a high value to the word sequence “I like turtles” and a low value to “Turtles sing table.” The language model steers the search towards word sequences that follow the same patterns as in the training data. Language models can also be seen in purely text-based applications, such as the autocomplete field in modern web browsers. Module 4 of this course is dedicated to language modeling.

Speech sound waves propagate through the air and are captured by a microphone which converts the pressure wave into electrical activity which can be captured. The electrical activity is sampled to create a sequence of waveform samples that describe the signal. Music signals are typically sampled at 44,100 Hz (or 44,100 samples per second). Due to the Nyquist theorem, this means that audio with frequencies of up to 22,050 Hz can be faithfully captured by sampling. Speech signals have less high frequency (only up to 8000 Hz) information so a sampling rate of 16,000 Hz is typically used. Speech over conventional telephone lines and most mobile phones is band-limited to about 3400 Hz, so a sampling rate of 8000 Hz is typically used for telephone speech.

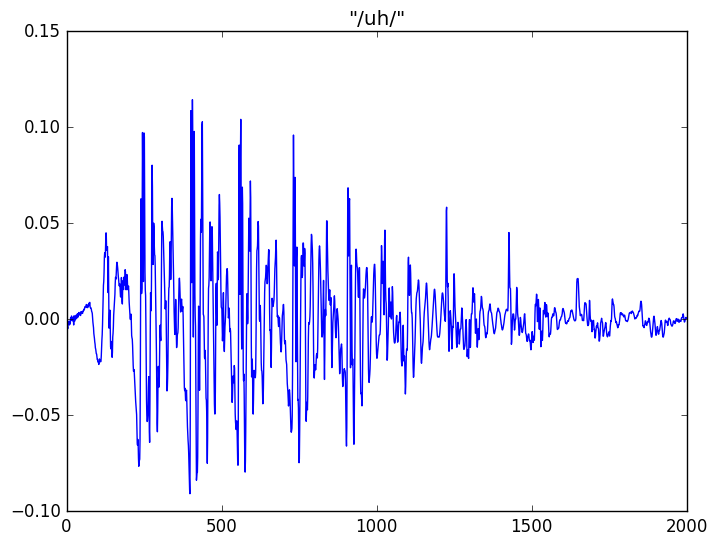
A typical waveform is plotted here, for the partial sentence "speech recognition is cool stuff".



Recall from module 1, where the concepts of voiced and unvoiced phonemes were discussed. If we focus on the waveform for the last word "stuff" we can see that the waveform has 3 distinct parts, the initial unvoiced sound 'st', the middle voiced vowel sound 'uh', and the final unvoiced 'f'. You can see that the unvoiced parts look noise-like and random which the voiced portion is periodic due to the vibration of the vocal chords.



If we zoom further into the voiced vowel segment, you can see the periodic nature more clearly. The periodic nature arises from the vibration of the vocal chords.



From observing these waveforms, it is apparent that two factors contribute to the characteristics of the waveform, 1) the excitation from the vocal chords that drives the air through the vocal tract and out the mouth and 2) the shape of the vocal tract itself when making a particular sound.

For example, we can see that both the 'st' and 'f' sounds are noise-like due to the unvoiced excitation but have different shapes because they are different sounds. The 'uh' sound is more periodic due to the voiced excitation and with its own shape due to the vocal tract. So, from the same speaker, a different vowel sound would have a similar periodicity but different overall shape because the same vocal chords are generating the excitation, but the shape of the vocal tract is different when producing a different sound.

This speech production process is most commonly modeled in signal processing using a source-filter model. The source is the excitation signal generated by the vocal chords that passes through the vocal tract, modeled as a time-varying linear filter. The source-filter model has many applications in speech recognition, synthesis, analysis and coding, and there are many ways of estimating the parameters of the source signal and the filter, such as the well-known linear predictive coding (LPC) approach.

For speech recognition, the phoneme classification is largely dependent on the vocal tract shape, and therefore, the filter portion of the source-filter model. The excitation or source signal is largely ignored or discarded. Thus, feature extraction process for speech recognition is largely designed for capturing the time-varying filter shapes over the course of an utterance.

## Short-time Fourier Analysis

One thing that is apparent from observing these waveforms is that speech is a non-stationary signal. That means its statistical properties change over time. Therefore, in order to properly analyze a speech signal, we need to examine the signal in chunks (also called windows or frames) that are small enough that the speech can be assumed to be stationary within those windows. Thus, we perform the analysis on a series of short, overlapping frames of audio. In speech recognition, we typically use windows of length 0.025 sec (25 ms) with an overlap of 0.01 (10 ms). This corresponds to a frame rate of 100 frames per second.

Because we are extracting a chunk from a longer continuous signal is it important to take care of edge effects by applying a window to the frame of data. Typically, a Hamming window is used although other windows may also be used.

If we let m be in the frame index, n is the sample index, and L is the frame size in samples and N is the frame shift in samples, each frame of audio is exacted from the original signal as

xm[n]=w[n]x[mN+n],n=0,1,…,L−1

where w[n] is the window function.

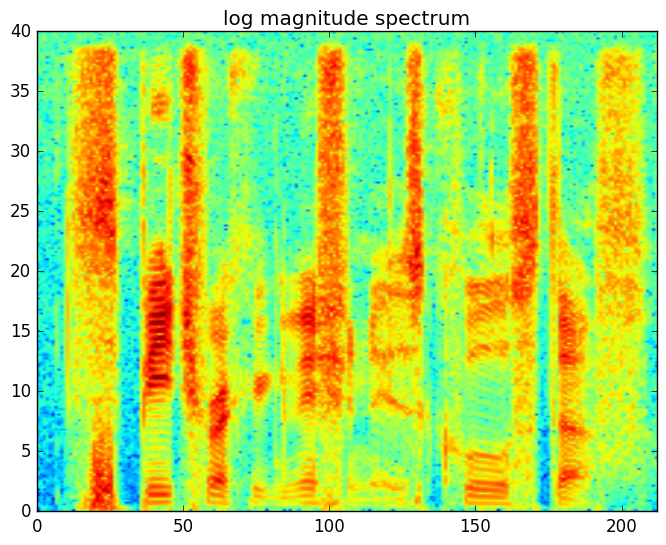
We then transform each frame of data into the frequency-domain using a discrete Fourier transform.

Xm[k]=∑n=0N−1xm[n]e−j2πknN

Note that all modern software packages have routines for efficiently computing the Fast Fourier Transform (FFT), which is an efficient way of computing the discrete Fourier transform.

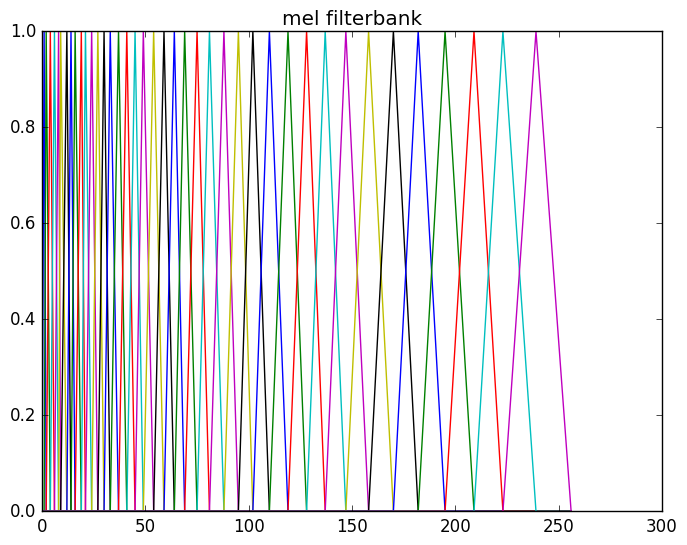
The Fourier representation XM[k] is a complex number which represents both the spectral magnitude (absolute amplitude) and phase each frame and frequency. For feature extraction purposes, we do not use the phase information so only consider the magnitude |Xm[k]|.

A spectrogram shows a 2D plot log-magnitude (or log-power) of the result of short-time fourier analysis of a speech signal. The horizontal axis shows the frame index (in 10 ms units) and the vertical axis shows the frequency axis from 0 Hz up to the Nyquist frequency with his one half of the sampling rate. For example, the spectrogram of the original waveform "speech recognition is cool stuff" is shown here. In the spectrogram, high energy regions are shown in orange and red.



From the spectrogram you can see high energy regions at the high frequencies (upper portion of the figure) which correspond roughly to unvoiced consonants and high energy regions at the lower frequencies which correspond roughly to voiced vowels. You'll also notice the horizonal lines in the voiced regions which signify the harmonic structure of voice speech.

To remove variability in the spectrogram caused by the harmonic structure in the voiced regions and the random noise in the unvoiced regions, we perform a spectral smoothing operation on the magnitude spectrum. We apply a filterbank which is motivated by the processing done by the auditory system. This filterbank applies an approximately logarithmic scale to the frequency axis. That is, the filters become wider and farther apart as frequency increases. The most common filterbank used for feature extraction is known as the **mel filterbank**. A mel filterbank of 40 filters is shown here. Each filter will average the power spectrogram across a different frequency range.



Observe that the filters are narrow and closely spaced on the left size of the figure and wider and farther apart on the right side of the figure.

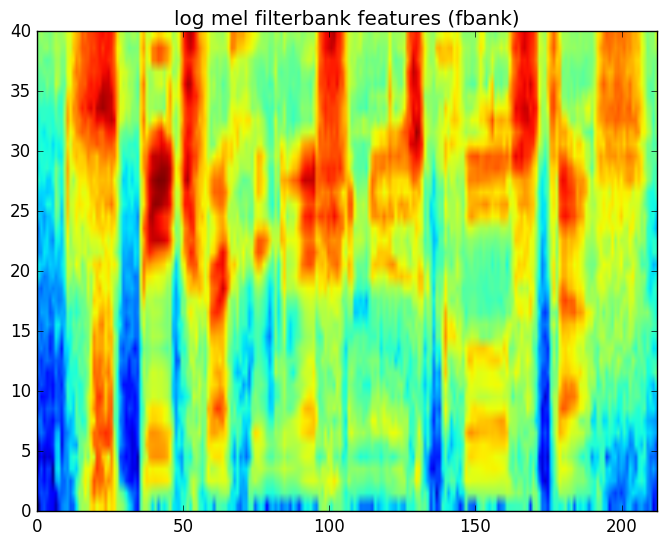
It is typical to represent the mel filterbank as a matrix, where each row corresponds to one filter in the filterbank. Thus, P-dimensional mel filterbank coefficients can be computed from the magnitude spectrum as

Xmel[p]=∑kM[p,k]|Xm[k]|,      p=0,1,…,P−1

A mel filterbank of length 40 is typical, though state of the art system have been built with fewer or more. Fewer results in more smoothing and more results in less smoothing.

The last step of the feature extraction process is to apply a logarithm operation. This helps compress the dynamic range of the signals and also closely models a nonlinear compression operation that occurs in the auditory system. We refer to the output of this logarithm operation as "filterbank" coefficients

The spectrogram-like view of the filterbank coefficients for the original waveform are shown here for a 40-dimensional filterbank. Compared to the original spectrogram, the filterbank coefficients are a much smoother version along the vertical (frequency) axis of the spectrogram where both the high frequency noise variability and pitch/harmonic structure has been removed.



There are other pre-processing steps that can be applied prior to feature extraction. These include

**Dithering:** adding a very small amount of noise to the signal to prevent mathematical issues during feature computation (in particular, taking the logarithm of 0)

**DC-removal:** removing any constant offset from the waveform

**Pre-emphasis:** applying a high pass filter to the signal prior to feature extraction to counteract that fact that typically the voiced speech at the lower frequencies has much high energy than the unvoiced speech at high frequencies. Pre-emphasis is performed with a simple linear filter

y[n]=x[n]−αx[n−1]

where a value of α=0.97 is commonly used.

It is possible that the communication channel will introduce some bias (constant filtering) on the captured speech signal. For example, a microphone may not have a flat frequency response. In addition, variations in signal gain can cause differences in the computed filterbank coefficients even though the underlying signals represent the same speech. These channel effects can be modeled as a convolution in time, which is equivalent to elementwise multiplication in the frequency domain representation of the signal.

Thus, we can model the channel effects as a consant filter

Xt,obs[k]=H[k]Xt[k]

And the magnitude of the observation as

|Xt,obs[k]|=|H[k]||Xt[k]|

If we take the log of both sides and compute the mean of all frames in the utterance, we have

μobs=1T∑tlog⁡(|Xt,obs[k]|)

=1T∑tlog⁡(|H[k]||Xt[k]|)

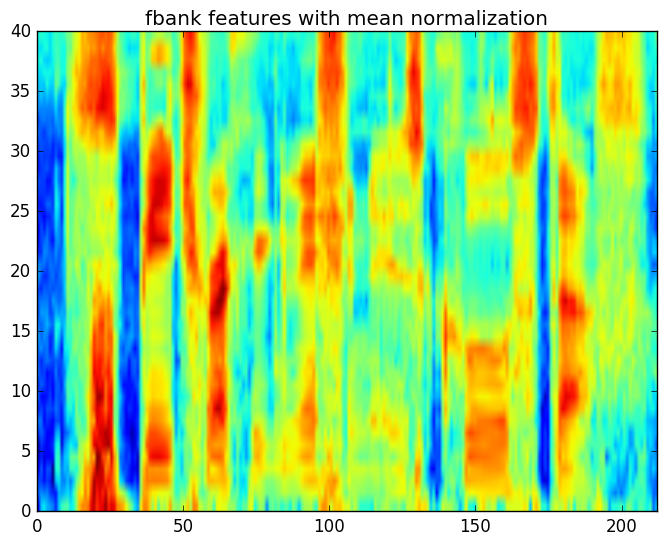
=1T∑tlog⁡(|H[k]|)+1T∑tlog⁡(|Xt[k]|)

Now, if we assume that the filter is constant over time and the log magnitude of the underlying speech signal has zero mean, this can be simplified to:

μttobs=log⁡(|H[k]|)

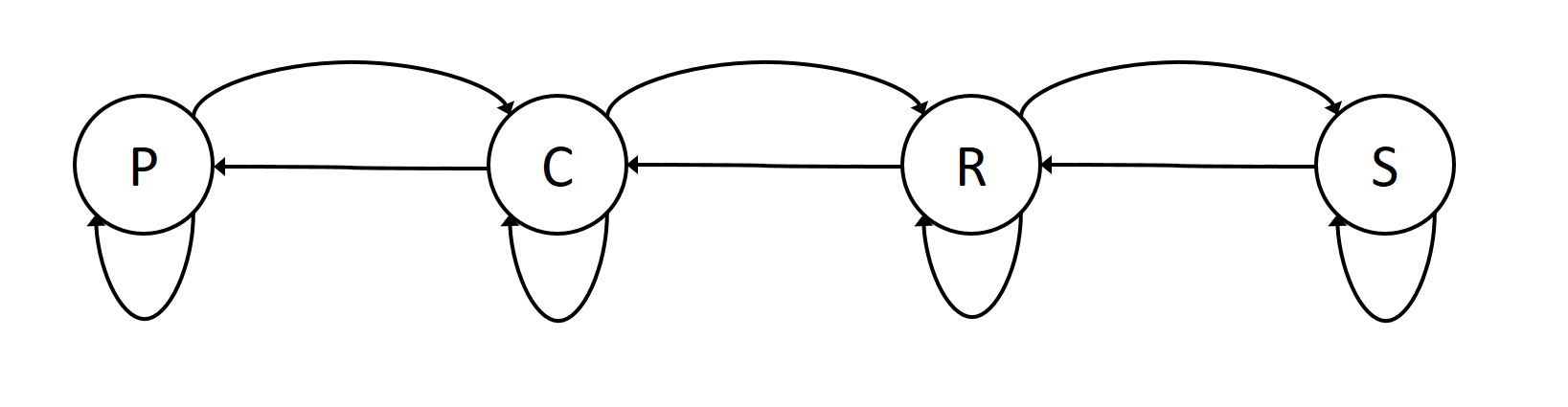
Thus, if we compute the mean of the log magnitude of the observed utterance and subtract it from every frame in the utterance, we’ll remove any constant channel effects from the signal.

For convenience, we perform this normalization on filterbank features directly, after the log operation. Below is a spectrogram of the previous filterbank coefficients after mean normalization.

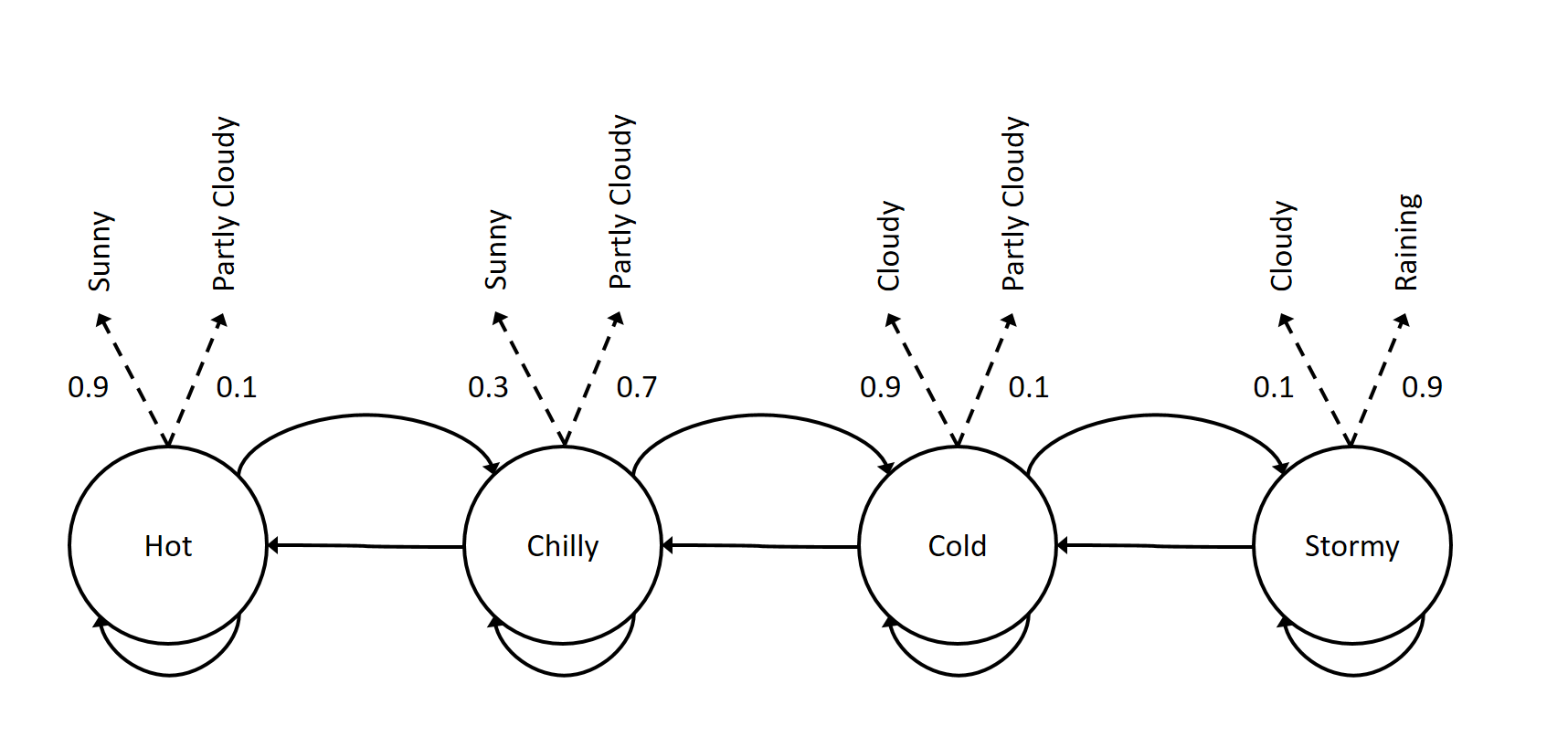


Summary:

To compute features for speech recognition from a speech signal, we are interested in extracting information about the time-varying spectral information that corresponds to the different underlying shapes of the vocal tract. These are modeled by a filter in the common source-filter model. The steps for computing the features for an utterance can be summarized as

1. Pre-process the signal, including pre-emphasis and dithering
2. Segment the signal into a series of overlapping frames, typically 25 ms frames with 10 ms frame shift
3. For each frame,
   * Apply a Hamming window function to the signal
   * Compute the Fourier transform using the FFT operation
   * Compute the magnitude of the spectrum
   * Apply the mel filterbank
   * Apply the log operation
4. If channel compensation is desired, apply mean normalize the frames of filterbank coefficients.
5. Before studying HMMs, it will be useful to briefly review Markov chains. Markov chains are a method for modeling random processes. In a Markov chains, discrete events are modeled with a number of states. The movement among states is governed by a random process.
6. Let’s consider an example. In a weather prediction application, the states could be " **S**unny", " **P**artly Cloud", " **C**loudy", and " **R**aining". If we wanted to consider the probability of a particular 5 day forecast, e.g. P(p,p,c,r,s), we would employ Bayes’ rule to break up this joint probability into a series of conditional probabilities.
7. p(X1,X2,X3,X4,X5)=p(X5|X4,X3,X2,X1)p(X4|X3,X2,X1)p(X3|X2,X1)p(X2|X1)p(X1)
8. This expression can be greatly simplified if we consider the first-order Markov assumption, which states that
9. p(Xi|X1,…,Xi−1)=p(Xi|Xi−1)
10. Under this assumption, the joint probability of a 5-day forecast can be written as
11. p(X1,X2,X3,X4,X5)=p(X5|X4)p(X4|X3)p(X3|X2)p(X2|X1)p(X1)
12. =p(X1)∏i=25p(Xi|Xi−1)
13. Thus, the key elements of a Markov chain are the state identities (weather forecast in this case) and the transition probabilities p(Xi|Xi−1) that express the probability of moving from one state to another (including back to the same state).
14. For example, a complete (though likely inaccurate) Markov chain for weather prediction can be depicted as
15. 
16. Note that in addition to the condidional probabilities
17. p(Xi|Xi−1)
18. in the equation above, there was also a probability associated with the first element of the sequence,
19. p(X1)
20. . So, in addition to the state inventory and the conditional transition probabilities, we also need a set of prior probabilities that indicate the probability of starting the chain in each of the states. Let us assume our prior probabilities are as follows:
21. p(p)=πp,p(c)=πc,p(r)=πr,p(s)=πs
22. Now, let us return to the example. We can now compute the probability of P(p,p,c,r,s) quite simply as
23. p(p,p,c,r,s)=p(s|r,c,p,p)p(r|c,p,p)p(c|p,p)p(p|p)p(p)
24. =p(s|r)p(r|c)p(c|p)p(p|p)p(p)

The Markov chains previously described are also known as *observable* Markov models. That is because once you land in a state, it is known what the outcome will be, e.g. it will rain. A *hidden* Markov model is different in that each state is defined not by a deterministic event or observation but by a probability distribution over events or observations. This makes the m,.odel *doubly stochastic.* The transitions between states are probabilistic and so are the observations in the states themselves. We could convert the Markov chain on weather to a hidden Markov model by replacing the states with distributions. Specifically, each state could have a different probability of seeing the various weather conditions, sun, partly cloudy, cloudy, or rainy.



Thus, a HMM is characterized by a set of *N* states along with

* A transition matrix that defines probabilities of transitioning among states A with elements aij
* A probability distribution for each state B={bi(x)},{i=1,2,…,N}
* A prior probability distribution over states π={π1,π2,…,πN}

This, we can summarize the parameters of an HMM compactly as Φ={A,B,π}

There are three fundamental problems for hidden Markov models each with well-known solutions. We will only briefly describe the problems and their solutions next. There are many good resources online and in the literature for additional details.

Given a model and an observation sequence, what is the probability that these observations were generated by the model?

This Evaluation problem can be solved summing up the probability over all possible values of the hidden state sequence. Implemented naively this can be quite expensive as there are an exponential number of states sequences (O(NT), where *N* is the number of states and *T* the number of time steps).

The *forward algorithm* is a far more efficient dynamic-programming solution. As its name implies, it processes the sequence in a single pass. It stores up to *N* values at each time step, and reduces the computational complexity to O(N2T).

Given a model and an observation sequence, what is the most likely sequence of states through the model that can explain the observations?

This problem can be solved using the well-known *Viterbi algorithm.* The application of this algorithm to the special case of large vocabulary speech recognition is discussed in Module 5, and an example of how it can be interated into the training criterion is discussed in Module 6.

Given a model and an observation sequence (or a set of observation sequences) how can we adjust the model parameters Φ?

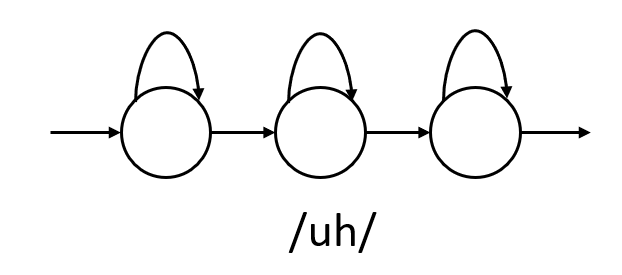
This problem can be efficiently solved using the *Baum-Welch* algorithm, which includes the *Forward-Backward* algorithm.

A byproduct of the forward algorithm mentioned earlier in this lession is that it computes the probability of being in a state *i* at time *t* given all observations up to and including time *t*. The backward algorithm has a similar structure, but computes the probability of being in state *i* at time *t*given all future observations starting at *t+1*. These two artifcats are combined in the forward-backward algorithm to produce the posterior probability of being in state *i* at time *t* given all of the observations.

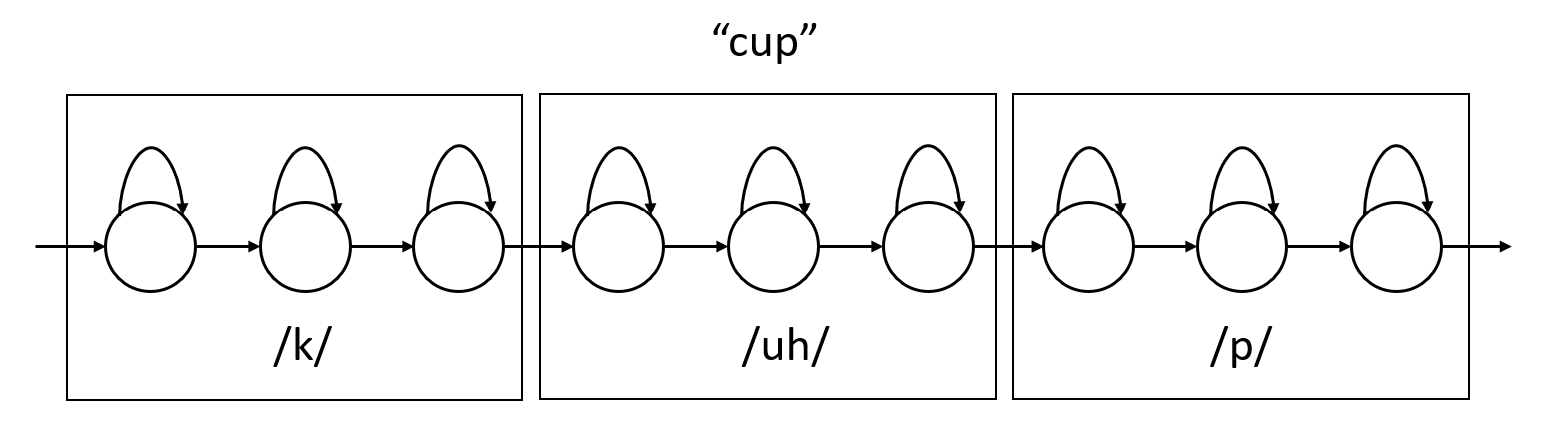
Once we know the posterior probability for each state at each time, the Baum-Welch algorithm acts as if these were direct observations of the hidden state sequence, and updates the model parameters to improve the objective function. An example of how this applies to acoustic modeling is covered in more depth in Module 6.

In speech recognition, hidden Markov models are used to model the acoustic observations (feature vectors) at the subword level, such as phonemes.

It is typically for each phoneme to be modeled with 3 states, to separately model the beginning, middle and end of the phoneme. Each state has a self-transition and a transition to the next state.



Word HMMs can be formed by concatenating its constituent phoneme HMMs. For example, the HMM word "cup" can be formed by concatenating the HMMs for its three phonemes.



Thus, a high-quality pronunciation dictionary which "spells" each word in the system by its phonemes is critically important for successful acoustic modeling.

Historically, each state in the HMM had a probability distribution defined by a Gaussian Mixture Model (GMM) which is defined as

p(x|s)=∑mwmN(x;μm,Σm)

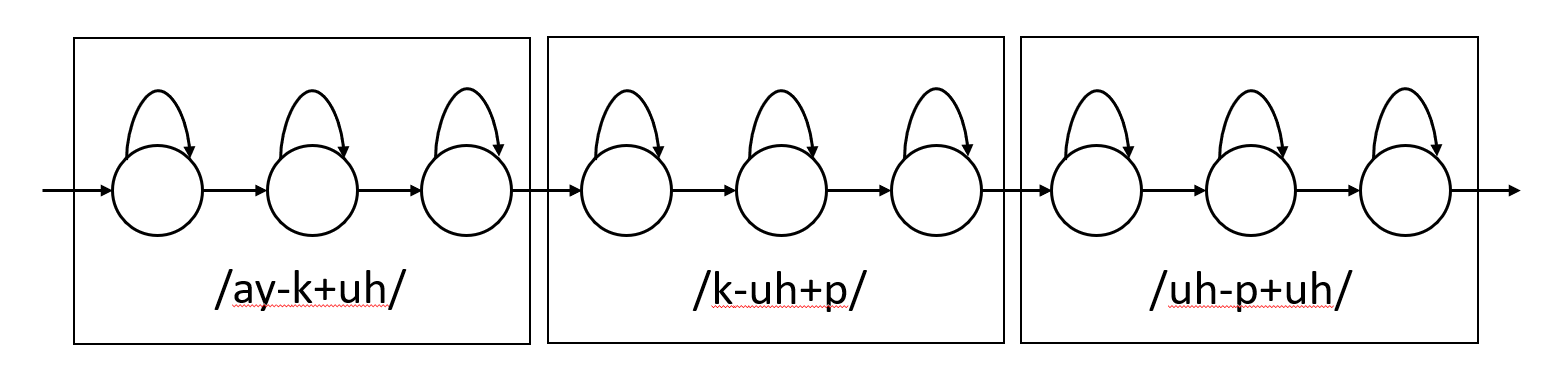
where N(x;μm,Σm) is a Gaussian distribution and wm is a mixture weight, with ∑mwm=1. Thus, each state of the model has its own GMM. The Baum-Welch training algorithm estimated all the transition probabilities as well as the means, variances, and mixture weights of all GMMs.

All modern speech recognition systems no longer model the observations using a collection of Gaussian mixture models but rather a single deep neural network that has output labels that represent the state labels of all HMMs states of all phonemes. For example, if there were 40 phonemes and each has a 3-state HMM, the neural network would have 40×3=120 output labels.

Such acoustic models are called "hybrid" systems or DNN-HMM systems to reflect the fact that the observation probability estimation formerly done by GMMs is now done by a DNN, but that the rest of the HMM framework, in particular the HMM state topologies and transition probabilities, are still used.

In the previous section, we described how word HMMs can be constructed by chaining the HMMs for the individual phones in a word according to the pronunciation dictionary. These phonemes are referred to as "context-independent" phones, or CI phones for short. It turns out that the realization of a phoneme is in fact heavily dependent on the phonemes can precede and follow it. For example, the /ah/ sound in "bat" is different from the /ah/ sound in "cap".

For this reason, higher accuracy can be achieved using "context-dependent" (CD) phones. Thus, to model "bat" we'd use an HMM representing the context dependent phone /b-ah+t/ for the middle /ah/ sound and for the word "cap" we'd use a separate HMM that modeled /k-ah+p/. So, imagine the word "cup" was in the utterance "a cup of coffee". Then cup would be modeled by the following context-dependent HMMs



Because this choice of context-dependent phones models 3 consecutive phones, they are referred to as "triphones". Though not common, some systems model an even longer phonetic context, such as "quinphones" which is a sequence of 5 consecutive phones.

When context-independent phones are used, there are a very managable number of states: *N* phones times *P* states per phone. U. S. English is typically represented using 40 phones, with three states per phone. This results in 120 context-independent states. As we move to context-dependent units, the number of triphones is N3. This leads to a significant increase in the number of states, for example: 403∗3=192,000.

This explosion of the label space leads to two major problems:

1. Far less data is available to train each triphone
2. Some triphones will not be observed in training but may occur in testing

A solution to these problems is in widespread use which involves pooling data associated with multiple context-dependent states that have similar properties and combining them into a single “tied” or “shared” HMM state. This tied state, known as a senone, is then used to compute the acoustic model scores for all of the original HMM states who’s data was pooled to create it.

Grouping a set of context-dependent triphone states into a collection of senones is performed using a *decision-tree clustering* process. A decision tree is constructed for every state of every context-independent phone.

The clustering process is performed as follows:

1. Merge all triphones with a common center phone from a particular state together to form the root node. For example, state 2 of all triphones of the form /\*-p+\*/
2. Grow the decision tree by asking a series of linguistic binary questions about the left or right context of the triphones. For example, *"Is the left context phone a back vowel?"* or "*Is the right context phone voiced?"* At each node, choose the question with results in the largest increase in likelihood of the training data.
3. Continue to grow the tree until the desired number of nodes are obtained or the likelihood increase of a further split is below a threshold.
4. The leaves of this tree define the senones for this context-dependent phone state.

This process solves both problems listed above. First, the data can now be shared among several triphone states so the parameters estimates are robust. Second, if a triphone is needed at test time that was unseen in training, it's corresponding senone can be found by walking the decision tree and answering the splitting questions appropriately.

Almost all modern speech recognition systems that use phone-based units utilize senones as the context-dependent unit. A production-grade large vocabulary recognizer can typically have about 10,000 senones in the model. Note that this is far more than the 120 context-independent states but far less than the 192,000 states in a untied context-dependent system.

One of the most significant advances in speech recognition in recent years is the use of deep neural network acoustic models. As mentioned earlier, the hybrid DNN systems replace a collection of GMMs (one for every senone) with a single deep neural network with output labels corresponding to senones.

The most common objective function used for training neural networks for classification tasks is *cross entropy*. For a *M*-way multi-class classification task such as senone classification, the objective function for a single sample can be written as

E=−∑i=1Mtmlog⁡(ym)

Where tm is the label (1 if the data is from class *m* and 0 otherwise) and ym is the output of the network which is a softmax layer over the output activations. Thus, for each frame, we need to generate a M-dimensional 1-hot vector, that consists of all zeros except for a single 1 corresponding to the true label. This means that we need to assign every frame of every utterance to a senone in order to generate these labels.