LSA 352
Speech Recognition and Synthesis

Dan Jurafsky

Lecture 6: Feature Extraction and Acoustic Modeling

IP Notice: Various slides were derived from Andrew Ng's CS 229 notes, as well as lecture notes from Chen, Picheny et al, Yun-Hsuan Sung, and Bryan Pellom. I'll try to give correct credit on each slide, but I'll prob miss some.

Outline for Today

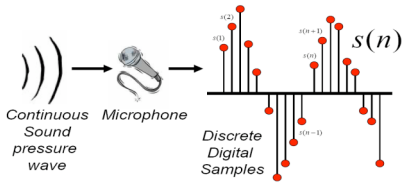
- Feature Extraction (MFCCs)
- The Acoustic Model: Gaussian Mixture Models (GMMs)
- Evaluation (Word Error Rate)
- How this fits into the ASR component of course
  - July 6: Language Modeling
  - July 19: HMMs, Forward, Viterbi,
  - **July 23: Feature Extraction, MFCCs, Gaussian Acoustic modeling, and hopefully Evaluation**
  - July 26: Spillover, Baum-Welch (EM) training

Outline for Today

- Feature Extraction
  - Mel-Frequency Cepstral Coefficients
- Acoustic Model
  - Increasingly sophisticated models
  - Acoustic Likelihood for each state:
    - Gaussians
    - Multivariate Gaussians
    - Mixtures of Multivariate Gaussians
  - Where a state is progressively:
    - CI Subphone (3ish per phone)
    - CD phone (=triphones)
    - State-tying of CD phone
- Evaluation
  - Word Error Rate

Discrete Representation of Signal

- Represent continuous signal into discrete form.



Thanks to Bryan Pellom for this slide

Digitizing the signal (A-D)

- **Sampling**:
  - measuring amplitude of signal at time t
  - 16,000 Hz (samples/sec) Microphone ("Wideband"):
  - 8,000 Hz (samples/sec) Telephone
  - Why?
    - Need at least 2 samples per cycle
    - max measurable frequency is half sampling rate
    - Human speech < 10,000 Hz, so need max 20K
    - Telephone filtered at 4K, so 8K is enough

Digitizing Speech (II)

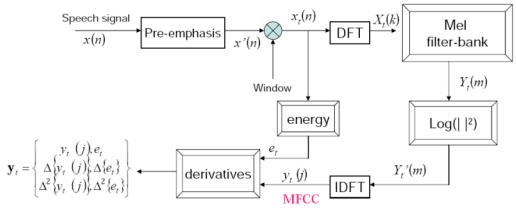
- **Quantization**
  - Representing real value of each amplitude as integer
  - 8-bit (-128 to 127) or 16-bit (-32768 to 32767)
- **Formats**:
  - 16 bit PCM
  - 8 bit mu-law; log compression
- LSB (Intel) vs. MSB (Sun, Apple)
- **Headers**:
  - Raw (no header)
  - Microsoft wav
  - Sun .au

Discrete Representation of Signal

- Byte swapping
  - Little-endian vs. Big-endian
- Some audio formats have headers
  - Headers contain meta-information such as sampling rates, recording condition
  - Raw file refers to 'no header'
  - Example: Microsoft wav, Nist sphere
- Nice sound manipulation tool: sox.
  - change sampling rate
  - convert speech formats

MFCC

- Mel-Frequency Cepstral Coefficient (MFCC)
  - Most widely used spectral representation in ASR

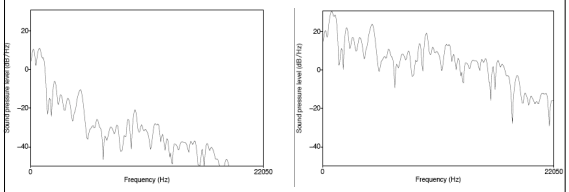


Pre-Emphasis

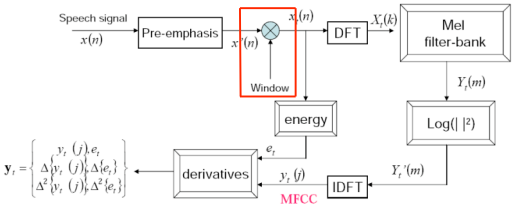
- Pre-emphasis: boosting the energy in the high frequencies
- Q: Why do this?
- A: The spectrum for voiced segments has more energy at lower frequencies than higher frequencies.
  - This is called **spectral tilt**
  - Spectral tilt is caused by the nature of the glottal pulse
- Boosting high-frequency energy gives more info to Acoustic Model
  - Improves phone recognition performance

Example of pre-emphasis

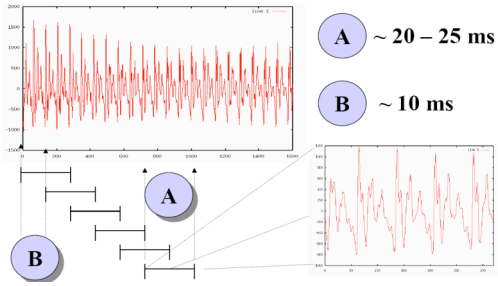
- Before and after pre-emphasis
  - Spectral slice from the vowel [aa]



MFCC



Windowing



Slide from Bryan Pellom

Windowing

- Why divide speech signal into successive overlapping frames?
  - Speech is not a stationary signal; we want information about a small enough region that the spectral information is a useful cue.
- Frames
  - Frame size: typically, 10-25ms
  - Frame shift: the length of time between successive frames, typically, 5-10ms

Common window shapes

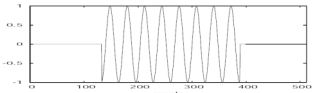
- Rectangular window:

$$w[n] = \begin{cases} 1 & 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases}$$

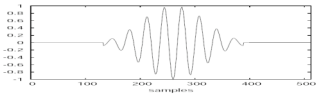
- Hamming window

$$w[n] = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right) & 0 \le n \le L-1 \\ 0 & \text{otherwise} \end{cases}$$

Window in time domain

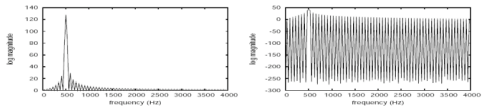


(a) Rectangular window

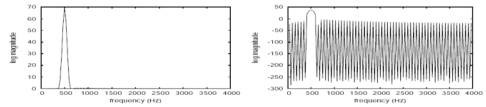


(c) Hamming window

Window in the frequency domain

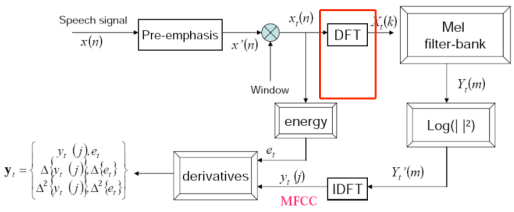


(a) Rectangular window (b) Rectangular window



(e) Hamming window (f) Hamming window

MFCC



Discrete Fourier Transform

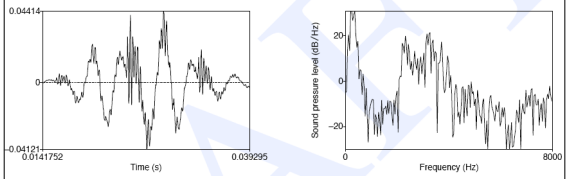
- Input:
  - Windowed signal x[n]…x[m]
- Output:
  - For each of N discrete frequency bands
  - A complex number X[k] representing magnidue and phase of that frequency component in the original signal
- Discrete Fourier Transform (DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\frac{\pi}{N}kn}$$

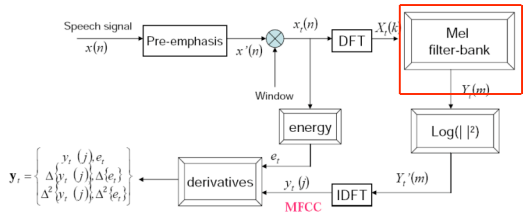
- Standard algorithm for computing DFT:
  - Fast Fourier Transform (FFT) with complexity N*log(N)
  - In general, choose N=512 or 1024

Discrete Fourier Transform computing a spectrum

- A 24 ms Hamming-windowed signal
  - And its spectrum as computed by DFT (plus other smoothing)

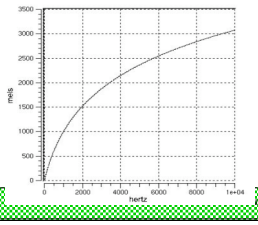


MFCC



Mel-scale

- Human hearing is not equally sensitive to all frequency bands
- Less sensitive at higher frequencies, roughly > 1000 Hz
- I.e. human perception of frequency is non-linear:



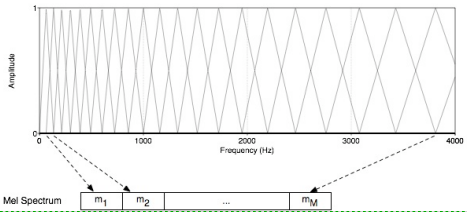
Mel-scale

- A **mel** is a unit of pitch
  - Definition:
    - Pairs of sounds perceptually equidistant in pitch
      - Are separated by an equal number of mels:
- Mel-scale is approximately linear below 1 kHz and logarithmic above 1 kHz
- Definition:

$$Mel(f) = 2595 \log_{10}(1 + \frac{f}{700})$$

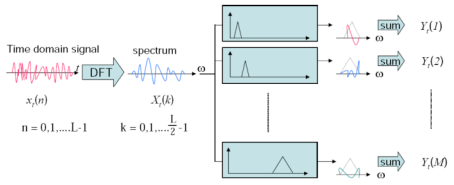
Mel Filter Bank Processing

- Mel Filter bank
  - Uniformly spaced before 1 kHz
  - logarithmic scale after 1 kHz

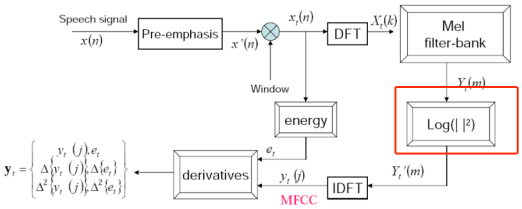


Mel-filter Bank Processing

- Apply the bank of filters according Mel scale to the spectrum
- Each filter output is the sum of its filtered spectral components

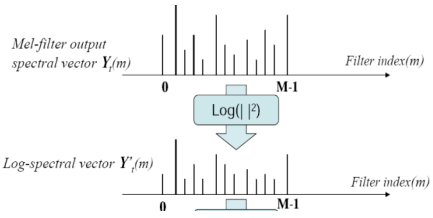


MFCC



Log energy computation

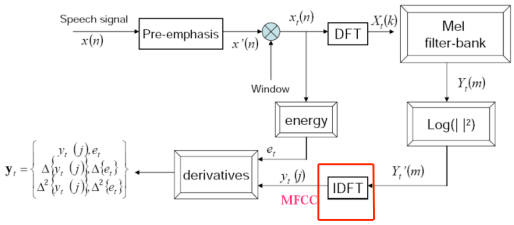
- Compute the logarithm of the square magnitude of the output of Mel-filter bank



Log energy computation

- Why log energy?
- Logarithm compresses dynamic range of values
  - Human response to signal level is logarithmic
  - humans less sensitive to slight differences in amplitude at high amplitudes than low amplitudes
- Makes frequency estimates less sensitive to slight variations in input (power variation due to speaker's mouth moving closer to mike)
- Phase information not helpful in speech

MFCC

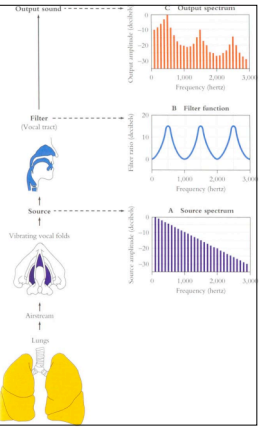


The Cepstrum

- One way to think about this
  - Separating the **source** and **filter**
  - Speech waveform is created by
    - A glottal source waveform
    - Passes through a vocal tract which because of its shape has a particular filtering characteristic
- Articulatory facts:
  - The vocal cord vibrations create harmonics
  - The mouth is an amplifier
  - Depending on shape of oral cavity, some harmonics are amplified more than others

Vocal Fold Vibration

George Miller figure

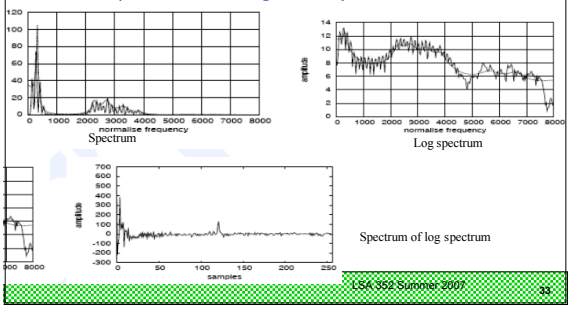


We care about the filter not the source

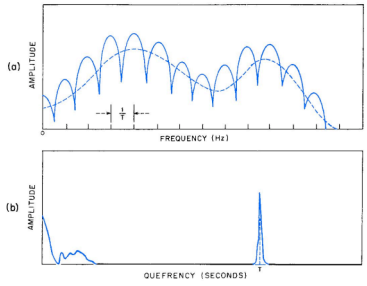
- Most characteristics of the source
  - F0
  - Details of glottal pulse
- Don't matter for phone detection
- What we care about is the **filter**
  - The exact position of the articulators in the oral tract
- So we want a way to separate these
  - And use only the filter function

The Cepstrum

- The spectrum of the log of the spectrum



Thinking about the Cepstrum



Mel Frequency cepstrum

- The cepstrum requires Fourier analysis
- But we're going from frequency space back to time
- So we actually apply inverse DFT

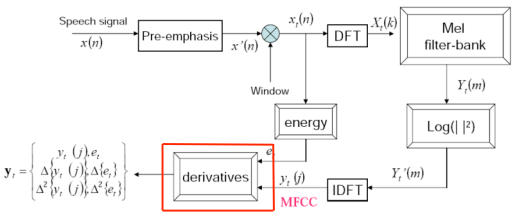
$$y_t[k] = \sum_{m=1}^{M} \log(|Y_t(m)|)\cos(k(m-0.5)\frac{\pi}{M}),\ \text{k=0,...,J}$$

- Details for signal processing gurus: Since the log power spectrum is real and symmetric, inverse DFT reduces to a Discrete Cosine Transform (DCT)

Another advantage of the Cepstrum

- DCT produces highly **uncorrelated** features
- We'll see when we get to acoustic modeling that these will be much easier to model than the spectrum
  - Simply modelled by linear combinations of Gaussian density functions with diagonal covariance matrices
- In general we'll just use the first 12 cepstral coefficients (we don't want the later ones which have e.g. the F0 spike)

MFCC

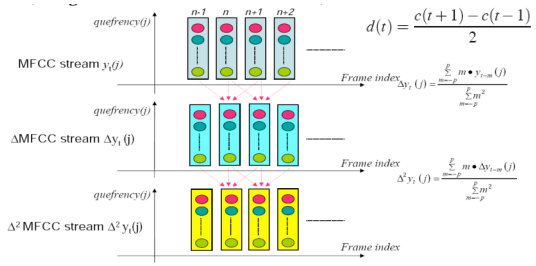


Dynamic Cepstral Coefficient

- The cepstral coefficients do not capture energy
- So we add an energy feature $Energy = \sum_{t=t_1}^{t_2} x^2[t]$
- Also, we know that speech signal is not constant (slope of formants, change from stop burst to release).
- So we want to add the changes in features (the slopes).
- We call these **delta** features
- We also add **double-delta** acceleration features

Delta and double-delta

- Derivative: in order to obtain temporal information



$$d(t) = \frac{c(t+1) - c(t-1)}{2}$$

Typical MFCC features

- Window size: 25ms
- Window shift: 10ms
- Pre-emphasis coefficient: 0.97
- MFCC:
  - 12 MFCC (mel frequency cepstral coefficients)
  - 1 energy feature
  - 12 delta MFCC features
  - 12 double-delta MFCC features
  - 1 delta energy feature
  - 1 double-delta energy feature
- Total 39-dimensional features

Why is MFCC so popular?

- Efficient to compute
- Incorporates a perceptual Mel frequency scale
- Separates the source and filter
- IDFT(DCT) decorrelates the features
  - Improves diagonal assumption in HMM modeling
- Alternative
  - PLP

Now on to Acoustic Modeling

Problem: how to apply HMM model to continuous observations?

- We have assumed that the output alphabet V has a finite number of symbols
- But spectral feature vectors are real-valued!
- How to deal with real-valued features?
  - Decoding: Given o_t, how to compute $P(o_t|q)$
  - Learning: How to modify EM to deal with real-valued features

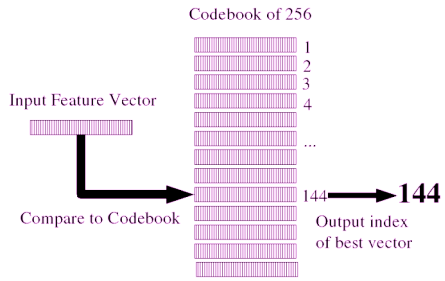
Vector Quantization

- Create a training set of feature vectors
- Cluster them into a small number of classes
- Represent each class by a discrete symbol
- For each class v_k, we can compute the probability that it is generated by a given HMM state using Baum-Welch as above

VQ

- We'll define a
  - Codebook, which lists for each symbol
  - A prototype vector, or codeword
- If we had 256 classes ('8-bit VQ'),
  - A codebook with 256 prototype vectors
  - Given an incoming feature vector, we compare it to each of the 256 prototype vectors
  - We pick whichever one is closest (by some 'distance metric')
  - And replace the input vector by the index of this prototype vector

VQ



VQ requirements

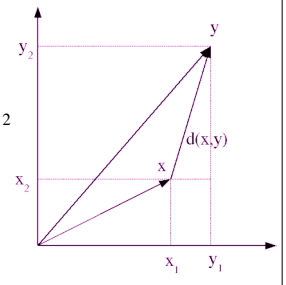
- A distance metric or distortion metric
  - Specifies how similar two vectors are
  - Used:
    - to build clusters
    - To find prototype vector for cluster
    - And to compare incoming vector to prototypes
- A clustering algorithm
  - K-means, etc.

Distance metrics

- Simplest:
  - (square of) Euclidean distance

$$d^2(x,y) = \sum_{i=1}^{D} (x_i - y_i)^2$$

  - Also called 'sum-squared error'



Distance metrics

- More sophisticated:
  - (square of) Mahalanobis distance
  - Assume that each dimension of feature vector has variance σ^2

$$d^2(x,y) = \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{\sigma_i^2}$$

  - Equation above assumes diagonal covariance matrix; more on this later

Training a VQ system (generating codebook): K-means clustering

1. Initialization
   choose M vectors from L training vectors (typically $M=2^B$)
   as initial code words… random or max. distance.
2. Search:
   for each training vector, find the closest code word, assign this training vector to that cell
3. Centroid Update:
   for each cell, compute centroid of that cell. The new code word is the centroid.
4. Repeat (2)-(3) until average distance falls below threshold (or no change)

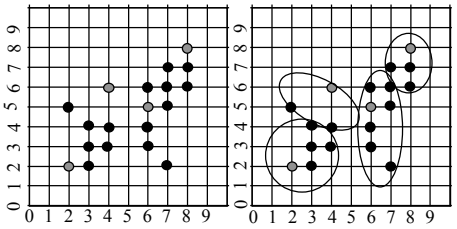
Slide from John-Paul Hosum, OHSU/OGI

Vector Quantization

Slide thanks to John-Paul Hosum, OHSU/OGI

• *Example*

Given data points, split into 4 codebook vectors with initial values at (2,2), (4,6), (6,5), and (8,8)

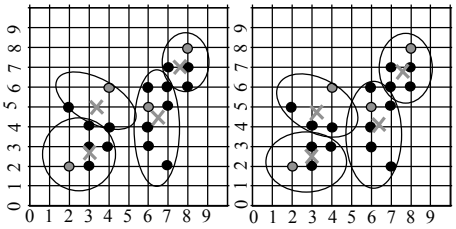


Vector Quantization

Slide from John-Paul Hosum, OHSU/OGI

• *Example*

compute centroids of each codebook, re-compute nearest neighbor, re-compute centroids...

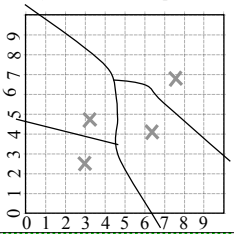


Vector Quantization

Slide from John-Paul Hosum, OHSU/OGI

• *Example*

Once there's no more change, the feature space will be partitioned into 4 regions. Any input feature can be classified as belonging to one of the 4 regions. The entire codebook can be specified by the 4 centroid points.

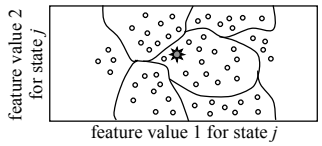


Summary: VQ

- To compute $p(o_t|q_j)$
  - Compute distance between feature vector o_t
    - and each codeword (prototype vector)
    - in a preclustered codebook
    - where distance is either
      - Euclidean
      - Mahalanobis
  - Choose the vector that is the closest to o_t
    - and take its codeword v_k
  - And then look up the likelihood of v_k given HMM state j in the B matrix
- $B_j(o_t)=b_j(v_k)$ s.t. v_k is codeword of closest vector to o_t
- Using Baum-Welch as above

Computing $b_j(v_k)$

Slide from John-Paul Hosum, OHSU/OGI



- $b_j(v_k) = \frac{\text{number of vectors with codebook index } k \text{ in state } j}{\text{number of vectors in state } j} = \frac{14}{56} = \frac{1}{4}$

Summary: VQ

- Training:
  - Do VQ and then use Baum-Welch to assign probabilities to each symbol
- Decoding:
  - Do VQ and then use the symbol probabilities in decoding

Directly Modeling Continuous Observations

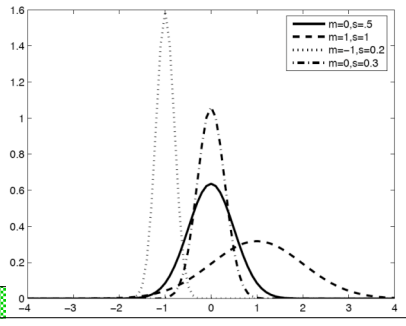
- Gaussians
  - Univariate Gaussians
    - Baum-Welch for univariate Gaussians
  - Multivariate Gaussians
    - Baum-Welch for multivariate Gausians
  - Gaussian Mixture Models (GMMs)
    - Baum-Welch for GMMs

Better than VQ

- VQ is insufficient for real ASR
- Instead: Assume the possible values of the observation feature vector o_t are normally distributed.
- Represent the observation likelihood function $b_j(o_t)$ as a Gaussian with mean μ_j and variance σ_j^2

$$f(x \mid \mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Gaussians are parameters by mean and variance



Reminder: means and variances

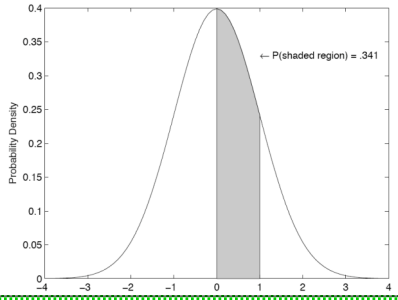
- For a discrete random variable X
- Mean is the expected value of X
  - Weighted sum over the values of X

$$\mu = E(X) = \sum_{i=1}^{N} p(X_i)X_i$$

- Variance is the squared average deviation from mean

$$\sigma^2 = E(X_i - E(X))^2) = \sum_{i=1}^{N} p(X_i)(X_i - E(X))^2$$

Gaussian as Probability Density Function

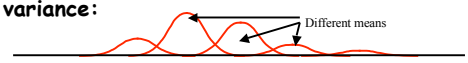


Gaussian PDFs

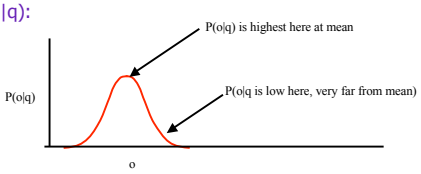
- A Gaussian is a probability density function; probability is area under curve.
- To make it a probability, we constrain area under curve = 1.
- BUT…
  - We will be using "point estimates"; value of Gaussian at point.
- Technically these are not probabilities, since a pdf gives a probability over a internvl, needs to be multiplied by dx
- As we will see later, this is ok since same value is omitted from all Gaussians, so argmax is still correct.

Gaussians for Acoustic Modeling

A Gaussian is parameterized by a mean and a variance:



- P(o|q):



Using a (univariate Gaussian) as an acoustic likelihood estimator

- Let's suppose our observation was a single real-valued feature (instead of 39D vector)
- Then if we had learned a Gaussian over the distribution of values of this feature
- We could compute the likelihood of any given observation o_t as follows:

$$b_j(o_t) = \frac{1}{\sqrt{2\pi\sigma_j^2}} exp\left(-\frac{(o_t-\mu_j)^2}{2\sigma_j^2}\right)$$

Training a Univariate Gaussian

- A (single) Gaussian is characterized by a mean and a variance
- Imagine that we had some training data in which each state was labeled
- We could just compute the mean and variance from the data:

$$\mu_i = \frac{1}{T}\sum_{t=1}^{T} o_t \quad s.t.\ o_t \ \ is \ \ state \ \ i$$

$$\sigma_i^2 = \frac{1}{T}\sum_{t=1}^{T} (o_t-\mu_i)^2 \quad s.t.\ o_t \ \ is \ \ state \ \ i$$

Training Univariate Gaussians

- But we don't know which observation was produced by which state!
- What we want: to assign each observation vector o_t to every possible state *i*, prorated by the probability the the HMM was in state *i* at time t.
- The probability of being in state i at time t is $\xi_t(i)$!!

$$\overline{\mu}_i = \frac{\sum_{t=1}^{T} \xi_t(i) o_t}{\sum_{t=1}^{T} \xi_t(i)}$$

$$\overline{\sigma^2}_i = \frac{\sum_{t=1}^{T} \xi_t(i)(o_t-\mu_i)^2}{\sum_{t=1}^{T} \xi_t(i)}$$

Multivariate Gaussians

- Instead of a single mean μ and variance σ:

$$f(x \mid \mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

- Vector of means μ and covariance matrix Σ

$$f(x \mid \mu,\Sigma) = \frac{1}{(2\pi)^{n/2} \mid \Sigma \mid^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

Multivariate Gaussians

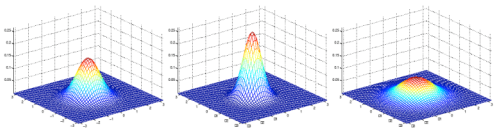
- Defining μ and Σ

$$\mu = E(x)$$
$$\Sigma = E\left[(x-\mu)(x-\mu)^T\right]$$

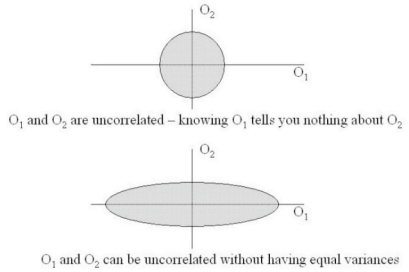
- So the i-jth element of Σ is:

$$\sigma_{ij}^2 = E\left[(x_i-\mu_i)(x_j-\mu_j)\right]$$

Gaussian Intuitions: Size of Σ

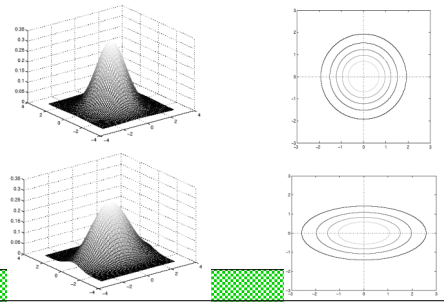


- μ = [0 0] μ = [0 0] μ = [0 0]
- Σ = I Σ = 0.6I Σ = 2I
- As Σ becomes larger, Gaussian becomes more spread out; as Σ becomes smaller, Gaussian more compressed

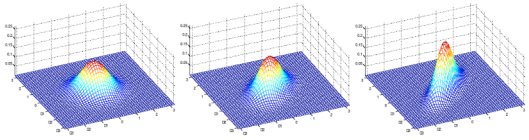


[1 0] [.6 0]
[0 1] [0 2]

- Different variances in different dimensions



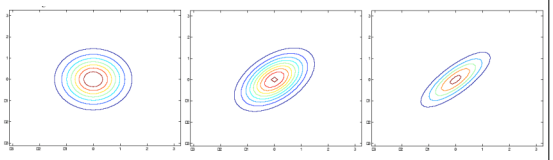
Gaussian Intuitions: Off-diagonal



$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

- As we increase the off-diagonal entries, more correlation between value of x and value of y

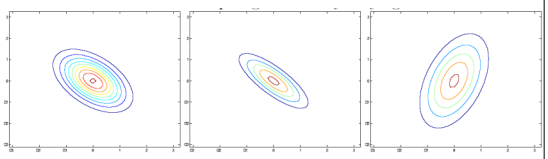
Gaussian Intuitions: off-diagonal



$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

- As we increase the off-diagonal entries, more correlation between value of x and value of y

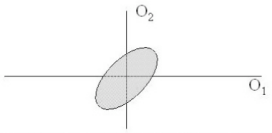
Gaussian Intuitions: off-diagonal and diagonal



$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

- Decreasing non-diagonal entries (#1-2)
- Increasing variance of one dimension in diagonal (#3)

In two dimensions



O_1 and O_2 are correlated – knowing O_1 tells you something about O_2

But: assume diagonal covariance

- I.e., assume that the features in the feature vector are uncorrelated
- This isn't true for FFT features, but is true for MFCC features, as we will see.
- Computation and storage much cheaper if diagonal covariance.
- I.e. only diagonal entries are non-zero
- Diagonal contains the variance of each dimension σ_{ii}^2
- So this means we consider the variance of each acoustic feature (dimension) separately

Diagonal covariance

- Diagonal contains the variance of each dimension σ_{ii}^2
- So this means we consider the variance of each acoustic feature (dimension) separately

$$f(x \mid \mu,\sigma) = \prod_{d=1}^{D} \frac{1}{\sigma_j \sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_d - \mu_d}{\sigma_d}\right)^2\right)$$

$$f(x \mid \mu,\sigma) = \frac{1}{2\pi^{D/2} \prod_{d=1}^{D} \sigma_d^{\,2}} \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - \mu_d)^2}{\sigma_d^{\,2}}\right)$$

Baum-Welch reestimation equations for multivariate Gaussians

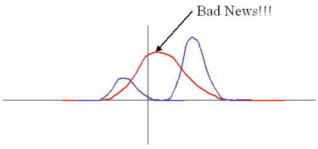
- Natural extension of univariate case, where now μ_i is mean vector for state i:

$$\bar{\mu}_i = \frac{\sum_{t=1}^{T} \xi_t(i) o_t}{\sum_{t=1}^{T} \xi_t(i)}$$

$$\bar{\Sigma}_i = \frac{\sum_{t=1}^{T} \xi_t(i)(o_t - \mu_i)(o_t - \mu_i)^T}{\sum_{t=1}^{T} \xi_t(i)}$$

But we're not there yet

- Single Gaussian may do a bad job of modeling distribution in any dimension:



- So

Mixtures of Gaussians

- M mixtures of Gaussians:

$$f(x \mid \mu_{jk}, \Sigma_{jk}) = \sum_{k=1}^{M} c_{jk} N(x, \mu_{jk}, \Sigma_{jk})$$

- For diagonal covariance:

$$b_j(o_t) = \sum_{k=1}^{M} c_{jk} N(o_t, \mu_{jk}, \Sigma_{jk})$$

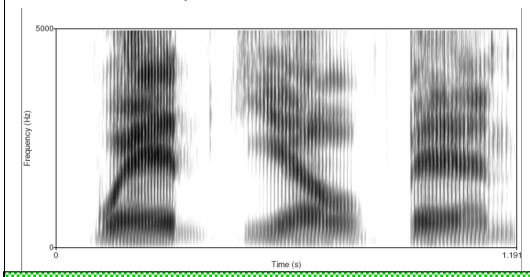
$$b_j(o_t) = \sum_{k=1}^{M} \frac{c_{jk}}{2\pi^{D/2} \prod_{d=1}^{D} \sigma_{jkd}^{\ 2}} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{jkd} - \mu_{jkd})^2}{\sigma_{jkd}^{\ 2}}\right)$$

GMMs

- Summary: each state has a likelihood function parameterized by:
  - M Mixture weights
  - M Mean Vectors of dimensionality D
  - Either
    - M Covariance Matrices of DxD
  - Or more likely
    - M Diagonal Covariance Matrices of DxD
    - which is equivalent to
    - M Variance Vectors of dimensionality D

Modeling phonetic context: different "eh"s

- w eh d y eh l b eh n



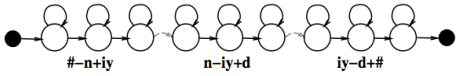
Modeling phonetic context

- The strongest factor affecting phonetic variability is the neighboring phone
- How to model that in HMMs?
- Idea: have phone models which are specific to context.
- Instead of Context-Independent (CI) phones
- We'll have Context-Dependent (CD) phones

CD phones: triphones

- Triphones
- Each triphone captures facts about preceding and following phone
- Monophone:
  - p, t, k
- Triphone:
  - iy-p+aa
  - a-b+c means "phone b, preceding by phone a, followed by phone c"

"Need" with triphone models



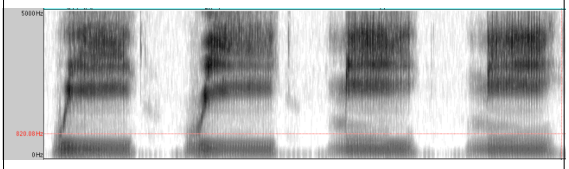
Word-Boundary Modeling

- Word-Internal Context-Dependent Models
  'OUR LIST':
  SIL AA+R AA-R L+IH L-IH+S IH-S+T S-T
- Cross-Word Context-Dependent Models
  'OUR LIST':
  SIL-AA+R AA-R**+L R-**L+IH L-IH+S IH-S+T S-T**+SIL**
- Dealing with cross-words makes decoding harder! We will return to this.

Implications of Cross-Word Triphones

- Possible triphones: 50x50x50=125,000
- How many triphone types actually occur?
- 20K word WSJ Task, numbers from Young et al
- Cross-word models: need 55,000 triphones
- But in training data only 18,500 triphones occur!
- Need to generalize models.

Modeling phonetic context: some contexts look similar

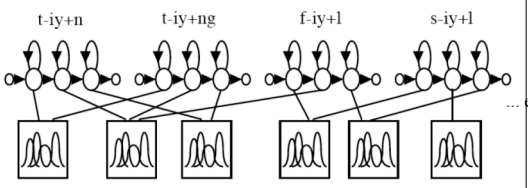


W iy r iy m iy n iy

Solution: State Tying

- Young, Odell, Woodland 1994
- Decision-Tree based clustering of triphone states
- States which are clustered together will share their Gaussians
- We call this "state tying", since these states are "tied together" to the same Gaussian.
- Previous work: generalized triphones
  - Model-based clustering ('model' = 'phone')
  - Clustering at state is more fine-grained

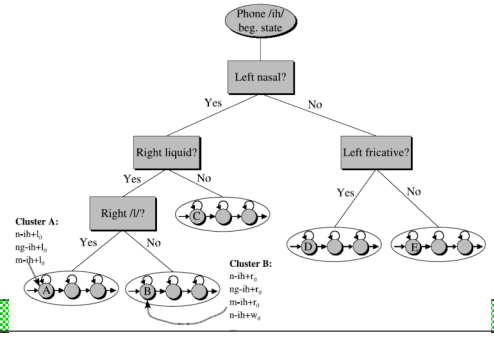
Young et al state tying



State tying/clustering

- How do we decide which triphones to cluster together?
- Use phonetic features (or 'broad phonetic classes')
  - Stop
  - Nasal
  - Fricative
  - Sibilant
  - Vowel
  - lateral

Decision tree for clustering triphones for tying



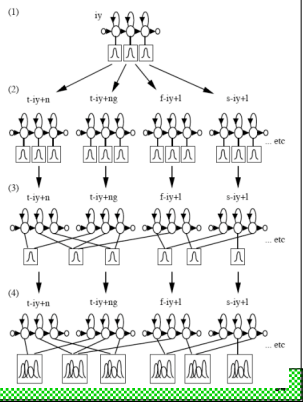
Decision tree for clustering triphones for tying

Feature	Phones
Stop	b d g k p t
Nasal	m n ng
Fricative	ch dh f jh s sh th v z zh
Liquid	l r w y
Vowel	aa ae ah ao aw ax axr ay eh er ey ih ix iy ow oy uh
Front Vowel	ae eh ih ix iy
Central Vowel	aa ah ao axr er
Back Vowel	ax ow uh uw
High Vowel	ih ix iy uh uw
Rounded	ao ow oy uh uw w
Reduced	ax axr ix
Unvoiced	ch f hh k p s sh t th
Coronal	ch d dh jh l n r s sh t th z zh

State Tying:

Young, Odell, Woodland 1994

- The steps in creating CD phones.
- Start with monophone, do EM training
- Then clone Gaussians into triphones
- Then build decision tree and cluster Gaussians
- Then clone and train mixtures (GMMs



Evaluation

- How to evaluate the word string output by a speech recognizer?

Word Error Rate

- Word Error Rate =

100 (Insertions+Substitutions + Deletions)

Total Word in Correct Transcript

Aligment example:

REF: portable **** PHONE UPSTAIRS last night so

HYP: portable FORM OF STORES last night so

Eval I S S

WER = 100 (1+2+0)/6 = 50%

NIST sctk-1.3 scoring softare: Computing WER with sclite

- http://www.nist.gov/speech/tools/
- Sclite aligns a hypothesized text (HYP) (from the recognizer) with a correct or reference text (REF) (human transcribed)

```
id: (2347-b-013)
Scores: (#C #S #D #I) 9 3 1 2
REF:  was an engineer SO I   i was always with **** **** MEN UM   and they
HYP:  was an engineer ** AND i was always with THEM THEY ALL THAT and they
Eval:                 D  S                      I    I    S   S
```


Sclite output for error analysis

```
CONFUSION PAIRS                 Total                (972)
                                With >=  1 occurances (972)

   1:    6  ->  (%hesitation) ==> on
   2:    6  ->  the ==> that
   3:    5  ->  but ==> that
   4:    4  ->  a ==> the
   5:    4  ->  four ==> for
   6:    4  ->  in ==> and
   7:    4  ->  there ==> that
   8:    3  ->  (%hesitation) ==> and
   9:    3  ->  (%hesitation) ==> the
  10:    3  ->  (a-) ==> i
  11:    3  ->  and ==> i
  12:    3  ->  and ==> in
  13:    3  ->  are ==> there
  14:    3  ->  as ==> is
  15:    3  ->  have ==> that
  16:    3  ->  is ==> this
```


Sclite output for error analysis

```
  17:    3  ->  it ==> that
  18:    3  ->  mouse ==> most
  19:    3  ->  was ==> is
  20:    3  ->  was ==> this
  21:    3  ->  you ==> we
  22:    2  ->  (%hesitation) ==> it
  23:    2  ->  (%hesitation) ==> that
  24:    2  ->  (%hesitation) ==> to
  25:    2  ->  (%hesitation) ==> yeah
  26:    2  ->  a ==> all
  27:    2  ->  a ==> know
  28:    2  ->  a ==> you
  29:    2  ->  along ==> well
  30:    2  ->  and ==> it
  31:    2  ->  and ==> we
  32:    2  ->  and ==> you
  33:    2  ->  are ==> i
  34:    2  ->  are ==> were
```

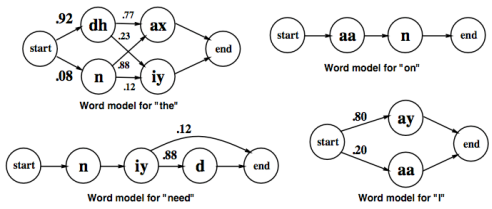

Better metrics than WER?

- WER has been useful
- But should we be more concerned with meaning ("semantic error rate")?
  - Good idea, but hard to agree on
  - Has been applied in dialogue systems, where desired semantic output is more clear

Summary: ASR Architecture

- Five easy pieces: ASR Noisy Channel architecture
  1) Feature Extraction:
     39 "MFCC" features
  2) Acoustic Model:
     Gaussians for computing p(o|q)
  3) Lexicon/Pronunciation Model
     - HMM: what phones can follow each other
  4) Language Model
     - N-grams for computing $p(w_i|w_{i-1})$
  5) Decoder
     - Viterbi algorithm: dynamic programming for combining all these to get word sequence from speech!

ASR Lexicon: Markov Models for pronunciation



Summary: Acoustic Modeling for LVCSR.

- Increasingly sophisticated models
- For each state:
  - Gaussians
  - Multivariate Gaussians
  - Mixtures of Multivariate Gaussians
- Where a state is progressively:
  - CI Phone
  - CI Subphone (3ish per phone)
  - CD phone (=triphones)
  - State-tying of CD phone
- Forward-Backward Training
- Viterbi training

Summary