CS 224S / LINGUIST 281
Speech Recognition, Synthesis, and Dialogue

Dan Jurafsky

Lecture 10: Acoustic Modeling

IP Notice:

Outline for Today

- Speech Recognition Architectural Overview
- Hidden Markov Models in general and for speech
  - Forward
  - Viterbi Decoding
- How this fits into the ASR component of course
  - Jan 27 HMMs, Forward, Viterbi,
  - Jan 29 Baum-Welch (Forward-Backward)
  - Feb 3: Feature Extraction, MFCCs, start of AM (VQ)
  - **Feb 5: Acoustic Modeling: GMMs**
  - Feb 10: N-grams and Language Modeling
  - Feb 24: Search and Advanced Decoding
  - Feb 26: Dealing with Variation
  - Mar 3: Dealing with Disfluencies

Outline for Today

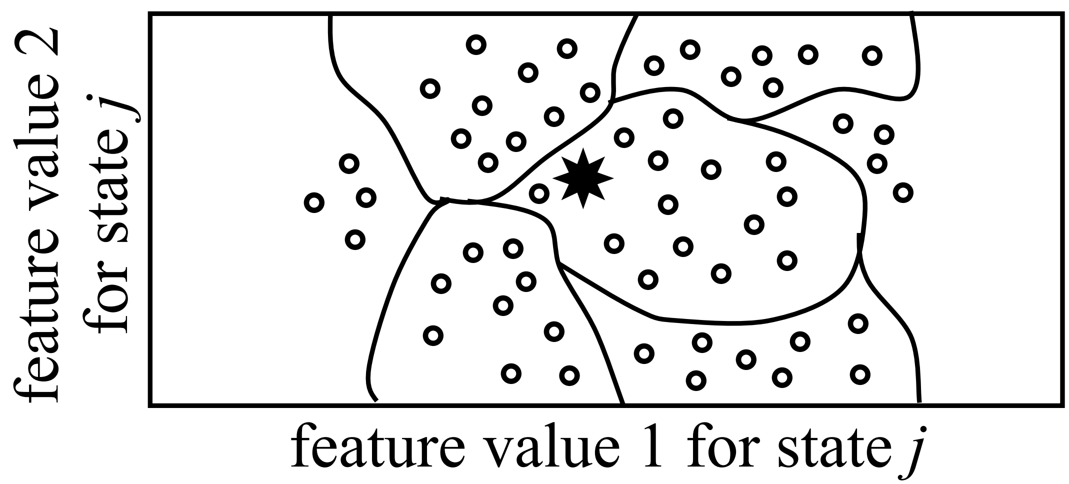
- Acoustic Model
  - Increasingly sophisticated models
  - Acoustic Likelihood for each state:
    - Gaussians
    - Multivariate Gaussians
    - Mixtures of Multivariate Gaussians
  - Where a state is progressively:
    - CI Subphone (3ish per phone)
    - CD phone (=triphones)
    - State-tying of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

Reminder: 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



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

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

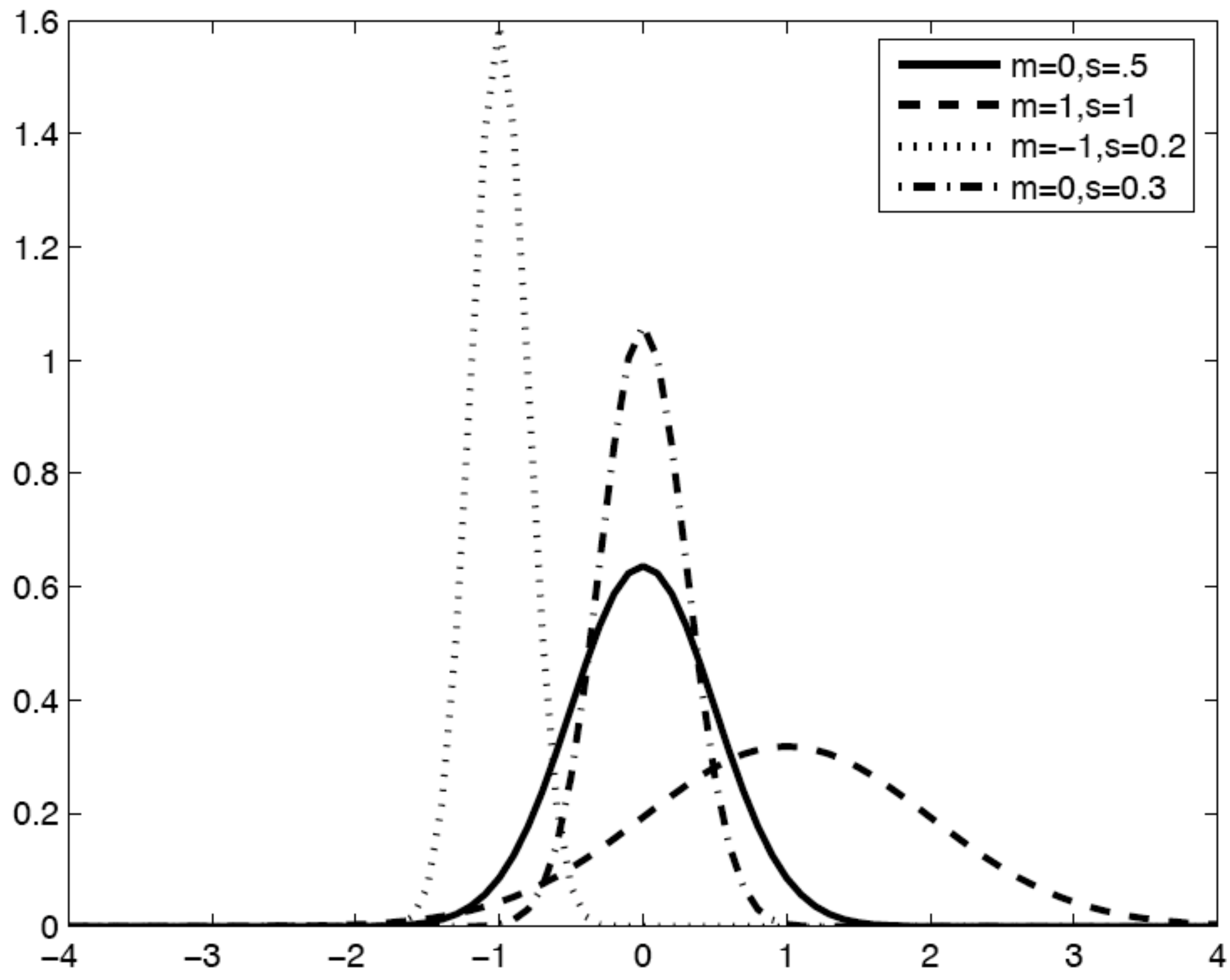
- Gaussians
  - Univariate Gaussians
    - Baum-Welch for univariate Gaussians
  - Multivariate Gaussians
    - Baum-Welch for multivariate Gausians
  - Gaussian Mixture Models (GMMs)
    - Baum-Welch for GMMs

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



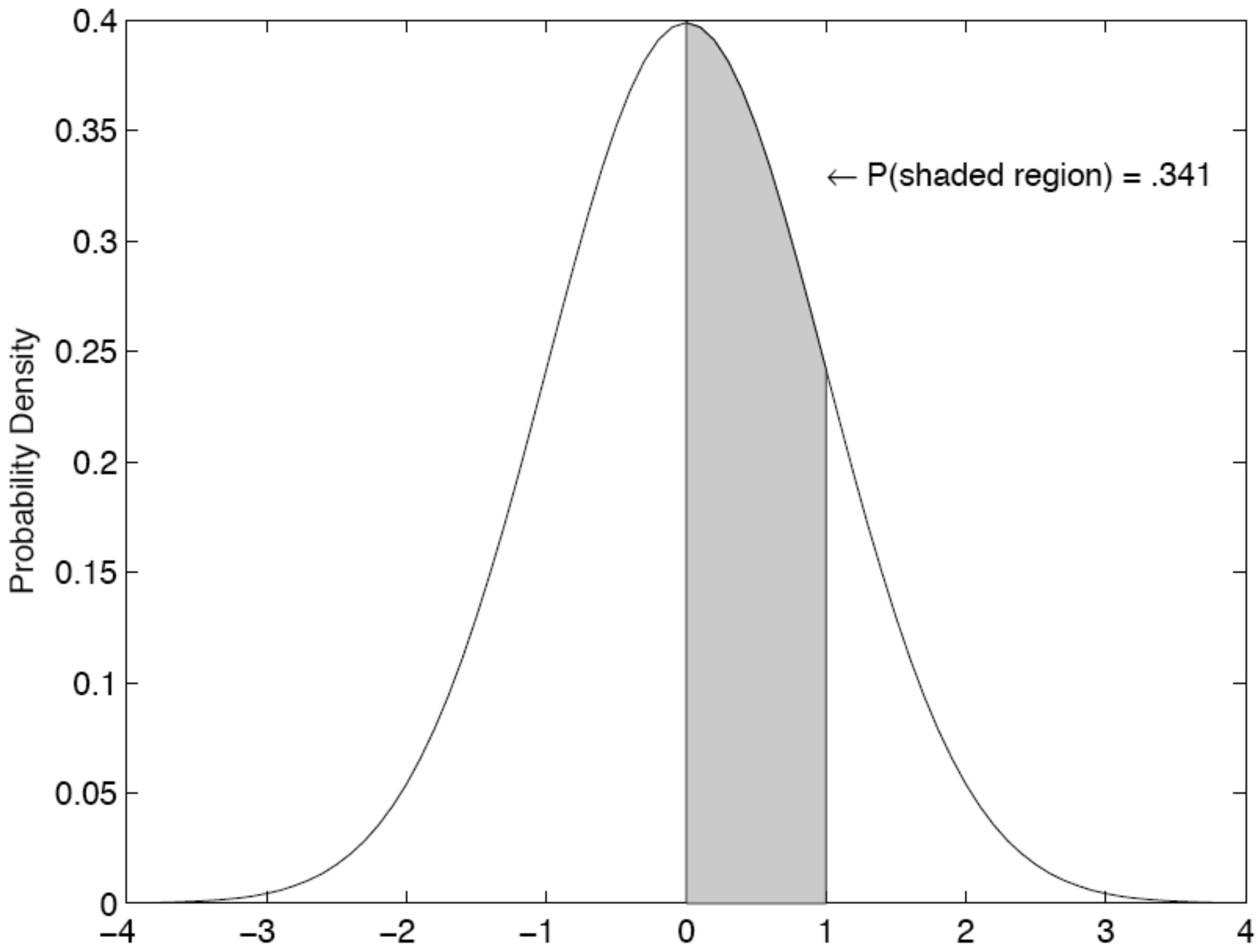
Reminder: means and variances

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

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

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

Gaussian as Probability Density Function

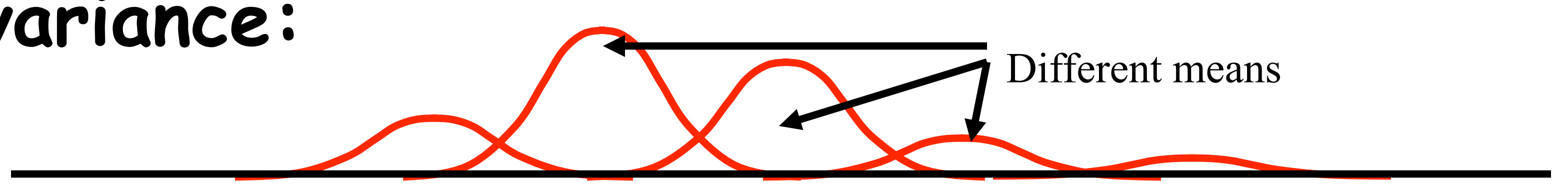


Gaussian PDFs

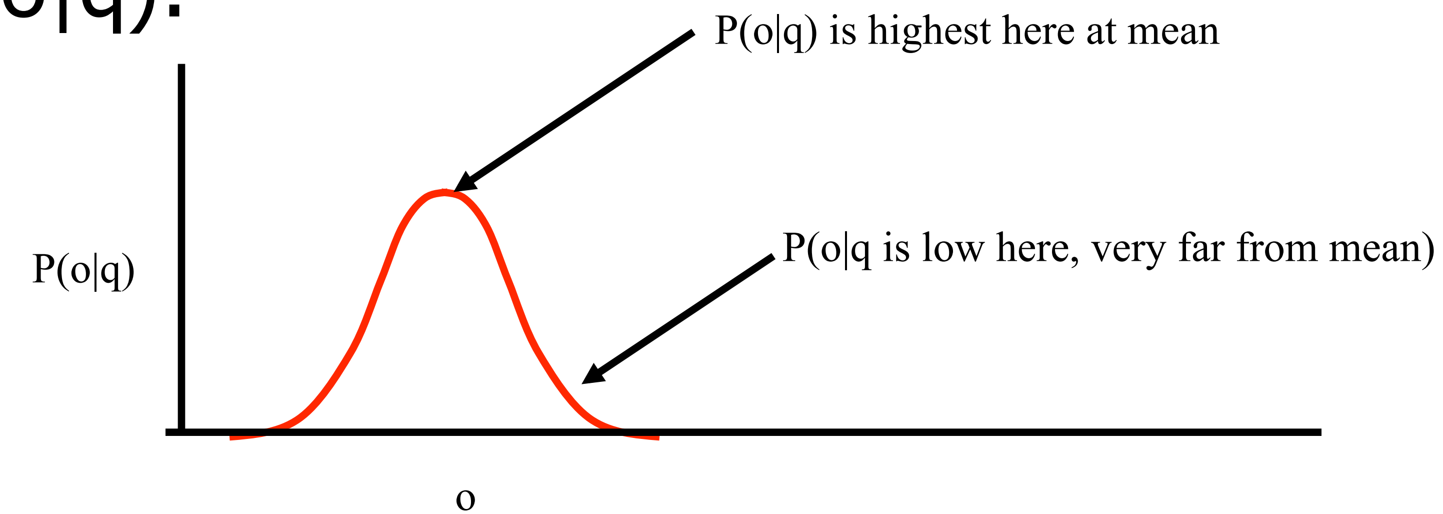
- A Gaussian is a probability density function; probability is area under curve.
- To make it a probability, we constrain area under curve = 1.
- BUT…
  - We will be using "point estimates"; value of Gaussian at point.
- Technically these are not probabilities, since a pdf gives a probability over a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

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

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

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 $\xi_t(i)$!!

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

Multivariate Gaussians

- Instead of a single mean μ and variance σ:

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

- Vector of observations *x* modeled by vector of means μ and covariance matrix Σ

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

Multivariate Gaussians

- Defining μ and Σ

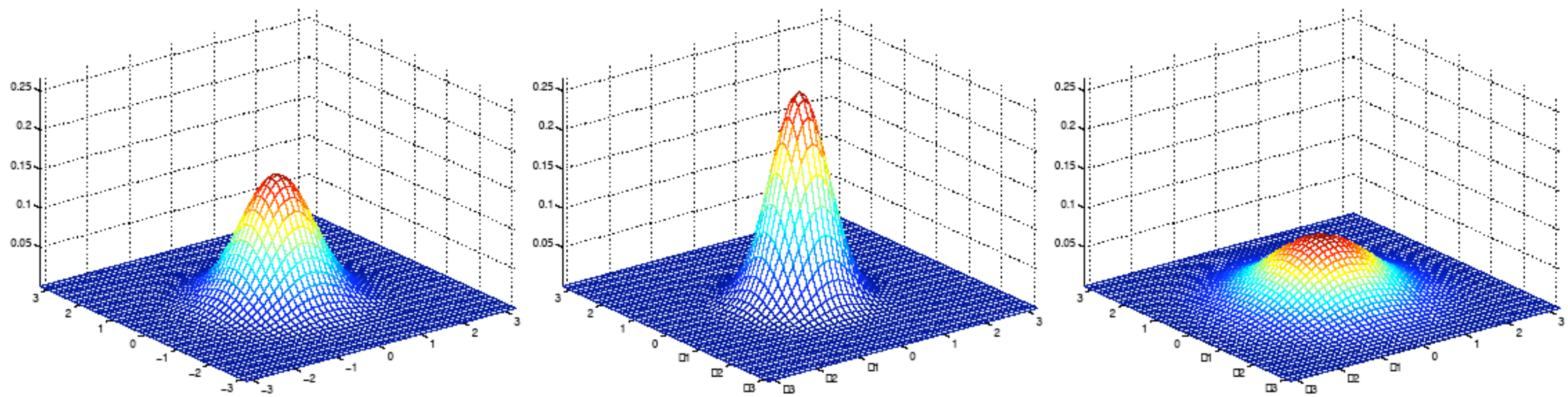
$$\mu = E(x)$$

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

- So the i-jth element of Σ is:

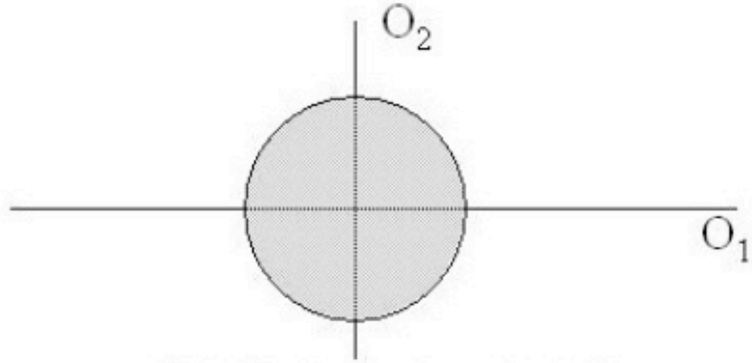
$$\sigma_{ij}^2 = E\left[(x_i-\mu_i)(x_j-\mu_j)\right]$$

Gaussian Intuitions: Size of Σ

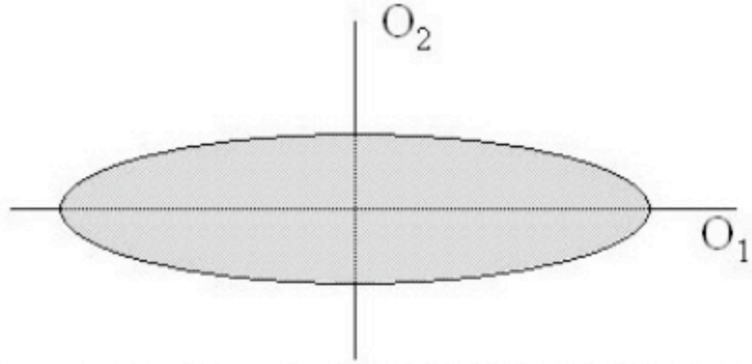


- $\mu = [0\ 0]$ $\qquad\qquad$ $\mu = [0\ 0]$ $\qquad\qquad$ $\mu = [0\ 0]$
- $\Sigma = I$ $\qquad\qquad$ $\Sigma = 0.6I$ $\qquad\qquad$ $\Sigma = 2I$
- As Σ becomes larger, Gaussian becomes more spread out; as Σ becomes smaller, Gaussian more compressed

Text and figures from Andrew Ng's lecture notes for CS229



O_1 and O_2 are uncorrelated – knowing O_1 tells you nothing about O_2

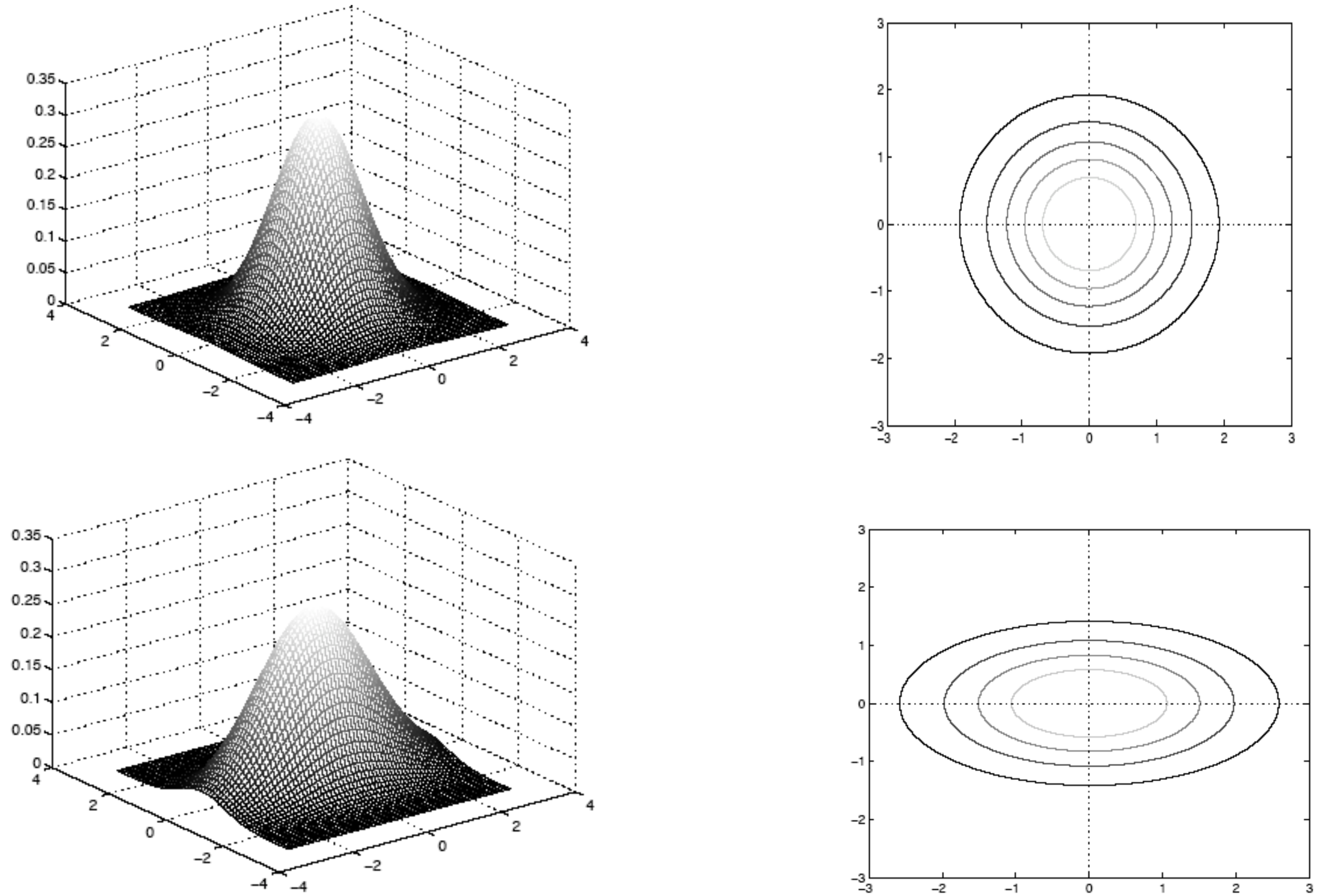


O_1 and O_2 can be uncorrelated without having equal variances

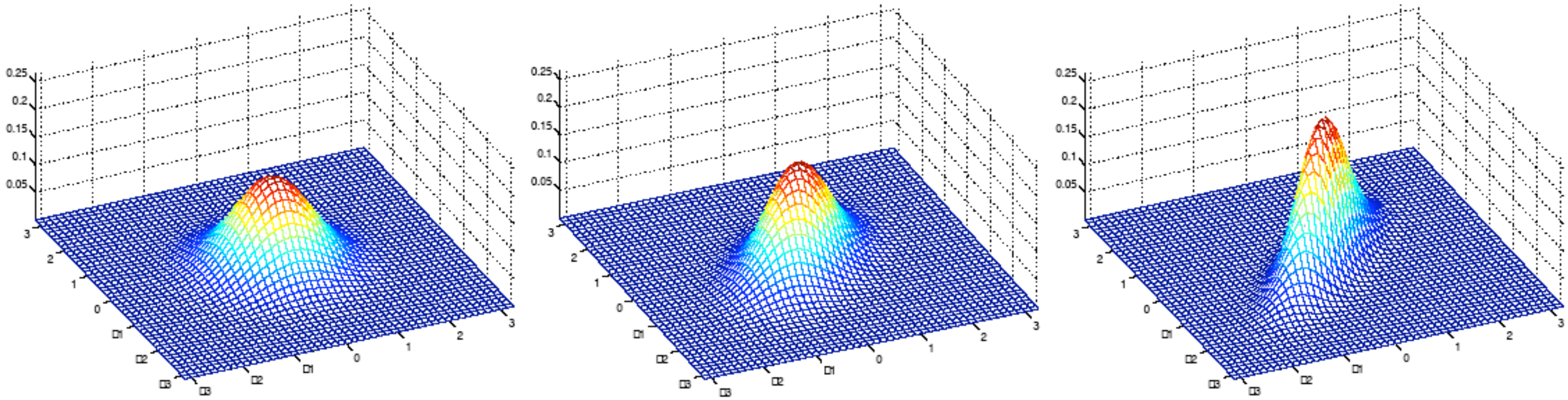
From Chen, Picheny et al lecture slides

$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$ $\begin{bmatrix}.6 & 0\\ 0 & 2\end{bmatrix}$

- Different variances in different dimensions



Gaussian Intuitions: Off-diagonal

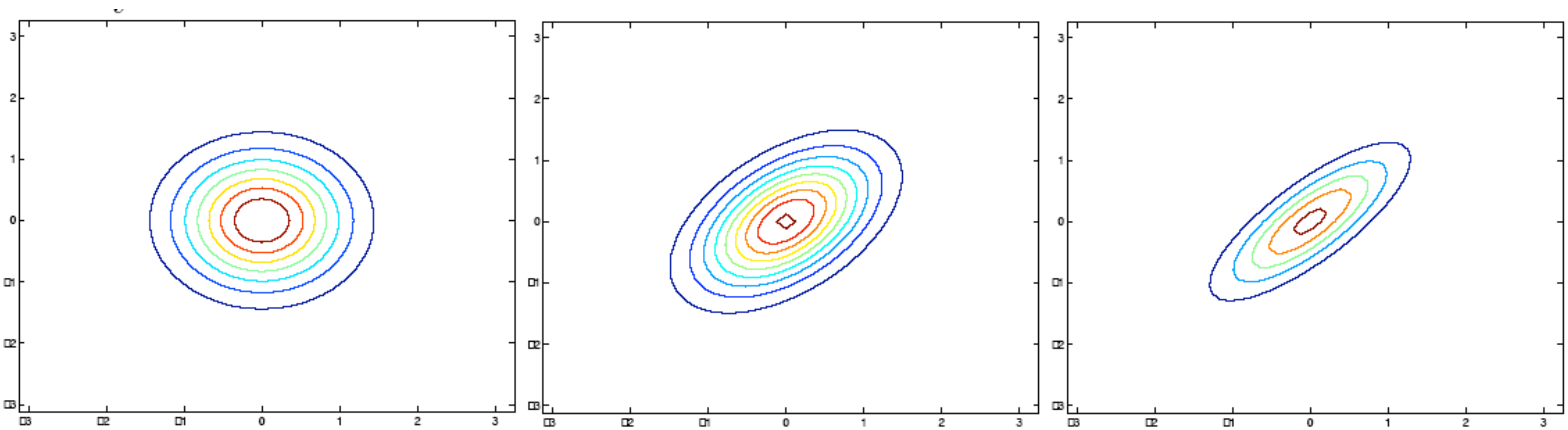


$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

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

Text and figures from Andrew Ng's lecture notes for CS229

Gaussian Intuitions: off-diagonal

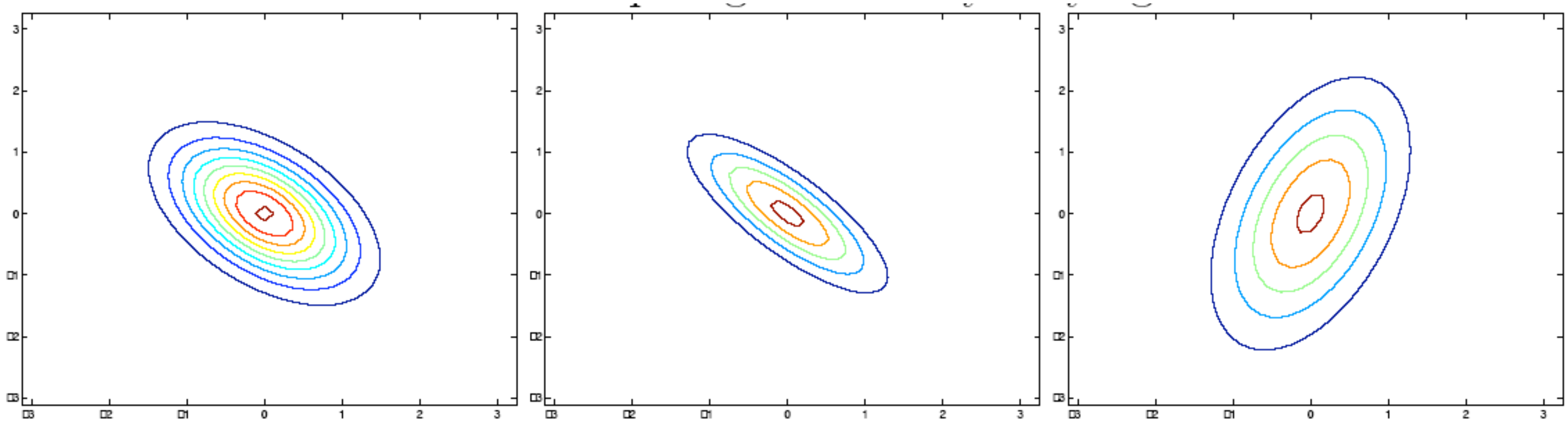


$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

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

Text and figures from Andrew Ng's lecture notes for CS229

Gaussian Intuitions: off-diagonal and diagonal

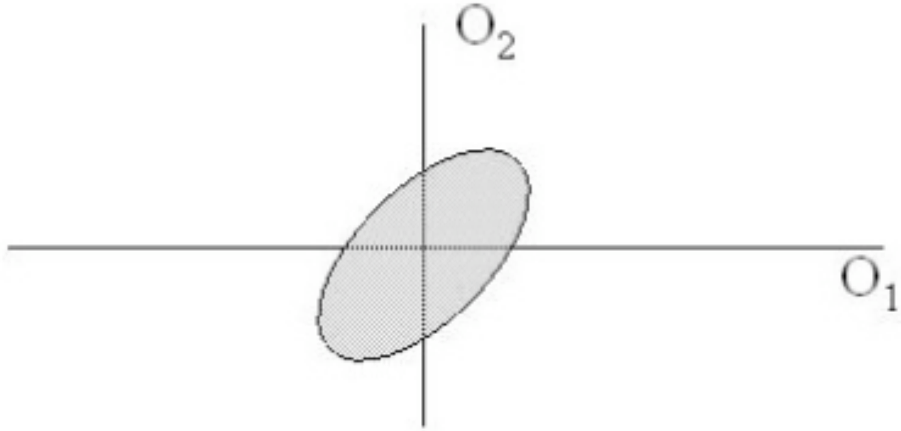


$$\Sigma = \begin{bmatrix} 1 & \text{-}0.5 \\ \text{-}0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & \text{-}0.8 \\ \text{-}0.8 & 1 \end{bmatrix}; \quad .\Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

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

Text and figures from Andrew Ng's lecture notes for CS229

In two dimensions



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

From Chen, Picheny et al lecture slides

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 saw las time
- 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

$$b_j(o_t) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{jd}^2}} \exp\left(-\frac{1}{2}\left(\frac{o_{td}-\mu_{jd}}{\sigma_{jd}}\right)^2\right)$$

$$b_j(o_t) = \frac{1}{2\pi^{D/2}\prod_{d=1}^{D}\sigma_{jd}^{\ 2}} \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^{\ 2}}\right)$$

Baum-Welch reestimation equations for multivariate Gaussians

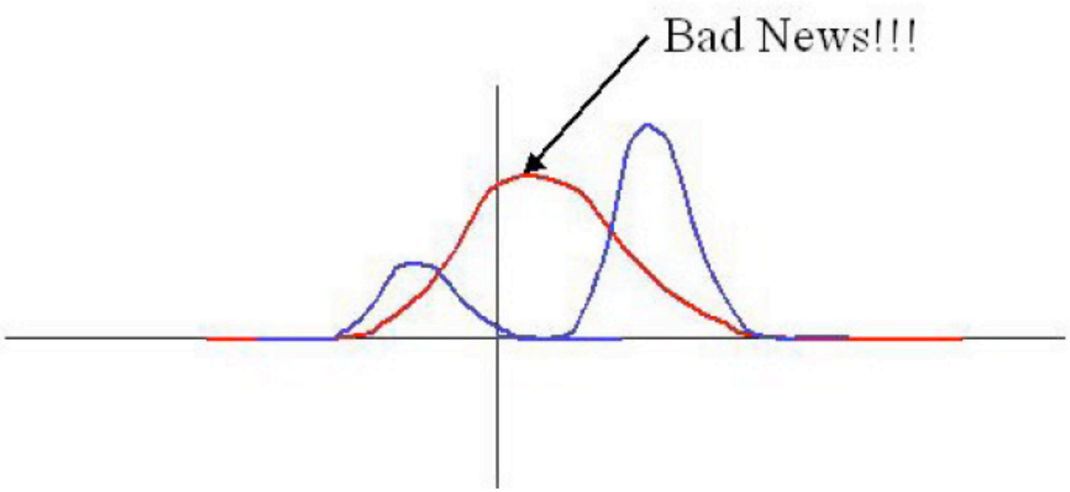
- Natural extension of univariate case, where now μ_i is mean vector for state i:

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

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

But we're not there yet

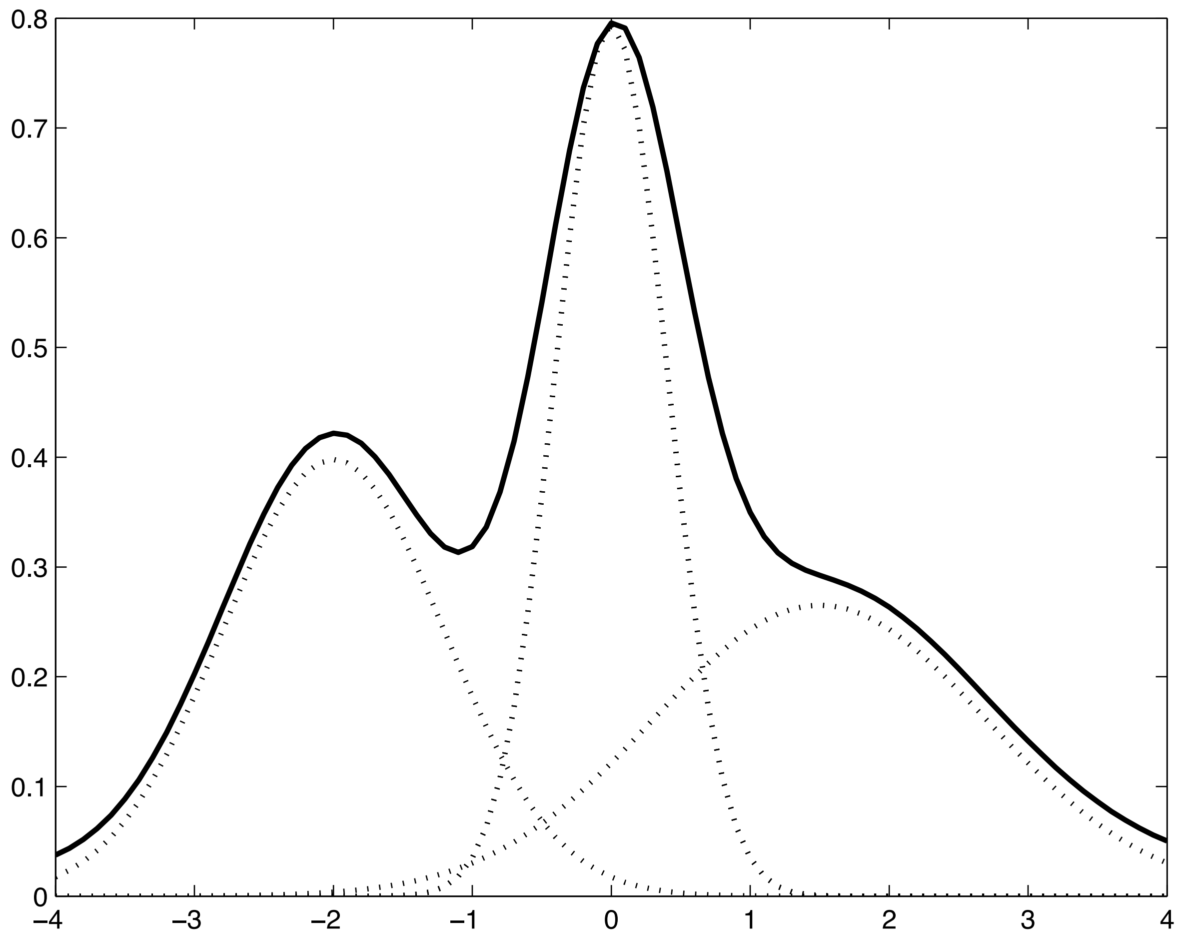
- Single Gaussian may do a bad job of modeling distribution in any dimension:



- Solution: Mixtures of Gaussians

Figure from Chen, Picheney et al slides

Mixture of Gaussians to model a function



Mixtures of Gaussians

- M mixtures of Gaussians:

$$f(x \mid \mu_{jk}, \Sigma_{jk}) = \sum_{k=1}^{M} c_{jk} \frac{1}{(2\pi)^{D/2} \mid \Sigma_{jk} \mid^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_{jk})^T \Sigma^{-1} (x - \mu_{jk}) \right)$$

- For diagonal covariance:

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

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

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

Training a GMM

- Problem: how do we train a GMM if we don't know what component is accounting for aspects of any particular observation?
- Intuition: we use Baum-Welch to find it for us, just as we did for finding hidden states that accounted for the observation

Baum-Welch for Mixture Models

- By analogy with ξ earlier, let's define the probability of being in state *j* at time *t* with the k^{th} mixture component accounting for o_t:

- Now,

$$\xi_{tm}(j) = \frac{\sum_{i=1}^{N} \alpha_{t-1}(j) a_{ij} c_{jm} b_{jm}(o_t) \beta_j(t)}{\alpha_F(T)}$$

$$\overline{\mu}_{jm} = \frac{\sum_{t=1}^{T} \xi_{tm}(j) o_t}{\sum_{t=1}^{T} \sum_{k=1}^{M} \xi_{tk}(j)} \qquad \overline{c}_{jm} = \frac{\sum_{t=1}^{T} \xi_{tm}(j)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \xi_{tk}(j)} \qquad \overline{\Sigma}_{jm} = \frac{\sum_{t=1}^{T} \xi_{tm}(j)(o_t - \mu_j)(o_t - \mu_j)^T}{\sum_{t=1}^{T} \sum_{k=1}^{M} \xi_{tm}(j)}$$

How to train mixtures?

- Choose M (often 16; or can tune M dependent on amount of training observations)
- Then can do various splitting or clustering algorithms
- One simple method for "splitting":

1) Compute global mean μ and global variance
2) Split into two Gaussians, with means $\mu \pm \varepsilon$ (sometimes ε is 0.2σ)
3) Run Forward-Backward to retrain
4) Go to 2 until we have 16 mixtures

Embedded Training

- Components of a speech recognizer:
  - Feature extraction: not statistical
  - Language model: word transition probabilities, trained on some other corpus
  - Acoustic model:
    - Pronunciation lexicon: the HMM structure for each word, built by hand
    - Observation likelihoods bj(ot)
    - Transition probabilities aij

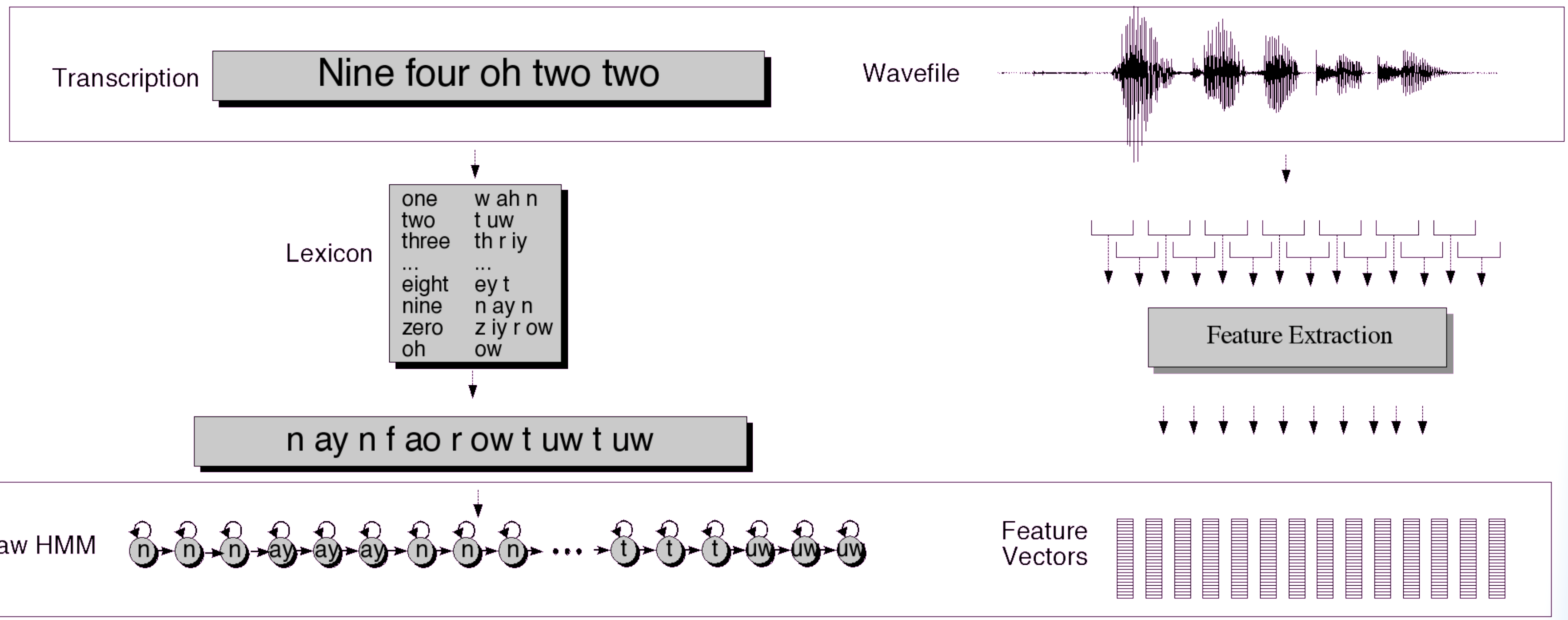
Embedded training of acoustic model

- If we had hand-segmented and hand-labeled training data
- With word and phone boundaries
- We could just compute the
  - B: means and variances of all our triphone gaussians
  - A: transition probabilities
- And we'd be done!
- But we don't have word and phone boundaries, nor phone labeling

Embedded training

- Instead:
  - We'll train each phone HMM embedded in an entire sentence
  - We'll do word/phone segmentation and alignment automatically as part of training process

Embedded Training



Initialization: "Flat start"

- Transition probabilities:
  - set to zero any that you want to be "structurally zero"
    - The γ probability computation includes previous value of a_{ij}, so if it's zero it will never change
  - Set the rest to identical values
- Likelihoods:
  - initialize μ and σ of each state to global mean and variance of all training data

Embedded Training

- Given: phoneset, pron lexicon, transcribed wavefiles
  - Build a whole sentence HMM for each sentence
  - Initialize A probs to 0.5, or to zero
  - Initialize B probs to global mean and variance
  - Run multiple iteractions of Baum Welch
    - During each iteration, we compute forward and backward probabilities
  - Use them to re-estimate A and B
  - Run Baum-Welch til converge

Viterbi training

- Baum-Welch training says:
  - We need to know what state we were in, to accumulate counts of a given output symbol o_t
  - We'll compute $\xi_I(t)$, the probability of being in state i at time t, by using forward-backward to sum over all possible paths that might have been in state i and output o_t.
- Viterbi training says:
  - Instead of summing over all possible paths, just take the single most likely path
  - Use the Viterbi algorithm to compute this "Viterbi" path
  - Via "forced alignment"

Forced Alignment

- Computing the "Viterbi path" over the training data is called "forced alignment"
- Because we know which word string to assign to each observation sequence.
- We just don't know the state sequence.
- So we use a_{ij} to constrain the path to go through the correct words
- And otherwise do normal Viterbi
- Result: state sequence!

Viterbi training equations

- Viterbi

Baum-Welch

$$\hat{a}_{ij} = \frac{n_{ij}}{n_i}$$

For all pairs of emitting states,
1 <= i, j <= N

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \gamma_t(i,j)}$$

$$\hat{b}_j(v_k) = \frac{n_j(s.t.o_t = v_k)}{n_j}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t = v_k}^{T} \xi_j(t)}{\sum_{t=1}^{T} \xi_j(t)}$$

Where n_{ij} is number of frames with transition from i to j in best path
And n_j is number of frames where state j is occupied

Viterbi Training

- Much faster than Baum-Welch
- But doesn't work quite as well
- But the tradeoff is often worth it.

Viterbi training (II)

- Equations for non-mixture Gaussians

$$\bar{\mu}_i = \frac{1}{N_i} \sum_{t=1}^{T} o_t \quad s.t.\ q_t = i$$

$$\bar{\sigma}_i^{\,2} = \frac{1}{N_i} \sum_{t=1}^{T} (o_t - \mu_i)^2 \quad s.t.\ q_t = i$$

- Viterbi training for mixture Gaussians is more complex, generally just assign each observation to 1 mixture

Log domain

- In practice, do all computation in log domain
- Avoids underflow
  - Instead of multiplying lots of very small probabilities, we add numbers that are not so small.
- Single multivariate Gaussian (diagonal Σ) compute:
- In log space:

$$b_j(o_t) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{jd}^2}} exp\left(-\frac{1}{2}\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^2}\right)$$

$$\log b_j(o_t) = -\frac{1}{2}\sum_{d=1}^{D}\left[log(2\pi)+\sigma_{jd}^2+\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^2}\right]$$

Log domain

- Repeating:

$$\log b_j(o_t) = -\frac{1}{2}\sum_{d=1}^{D}\left[log(2\pi) + \sigma_{jd}^2 + \frac{(o_{td} - \mu_{jd})^2}{\sigma_{jd}^2}\right]$$

- With some rearrangement of terms

$$\log b_j(o_t) = C - \frac{1}{2}\sum_{d=1}^{D}\frac{(o_{td} - \mu_{jd})^2}{\sigma_{jd}^2}$$

- Where:

$$C = -\frac{1}{2}\sum_{d=1}^{D}\left(\log(2\pi) + \sigma_{jd}^2\right)$$

- Note that this looks like a weighted Mahalanobis distance!!!
- Also may justify why we these aren't really probabilities (point estimates); these are really just distances.

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
```

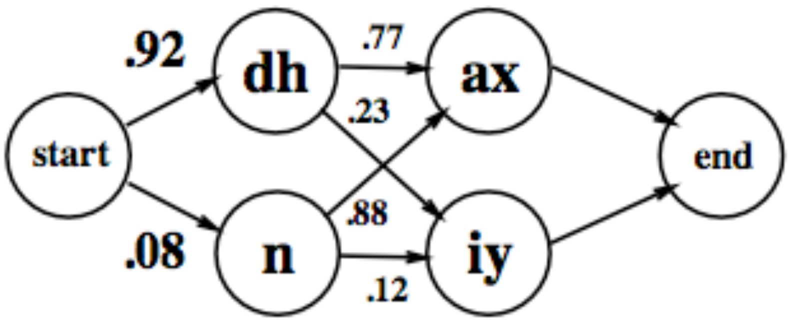
Better metrics than WER?

- 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

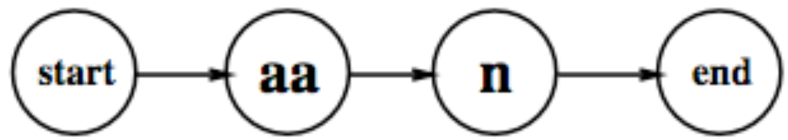
Summary: ASR Architecture

- 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!

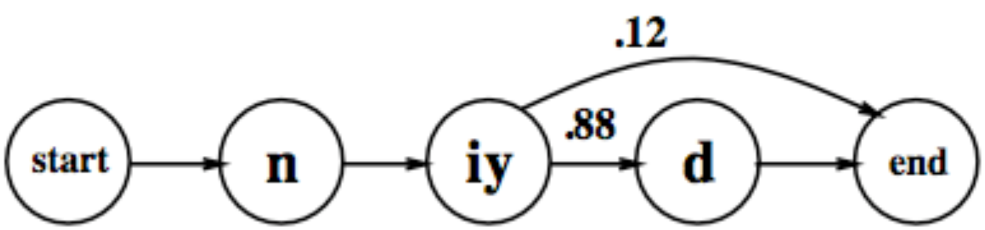
ASR Lexicon: Markov Models for pronunciation



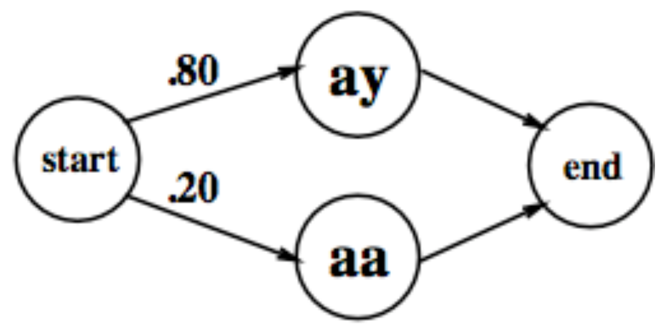
Word model for "the"



Word model for "on"



Word model for "need"



Word model for "I"

Pronunciation Modeling

Generating surface forms:

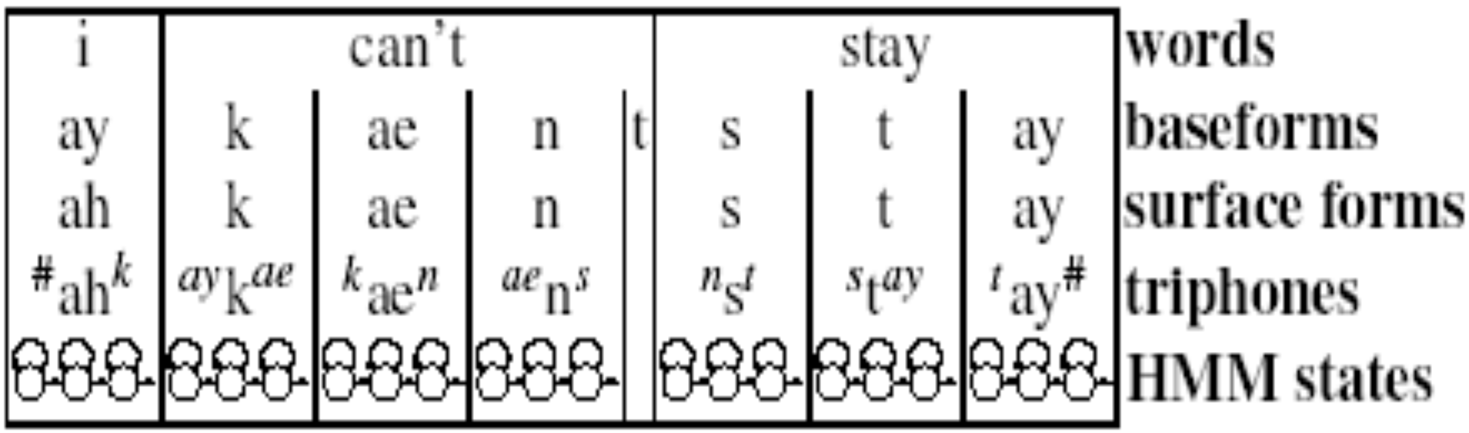


Fig. 4. Transforming from baseform pronunciations to surface pronunciations in ASR decoding.

Eric Fosler-Lussier slide

Dynamic Pronunciation Modeling

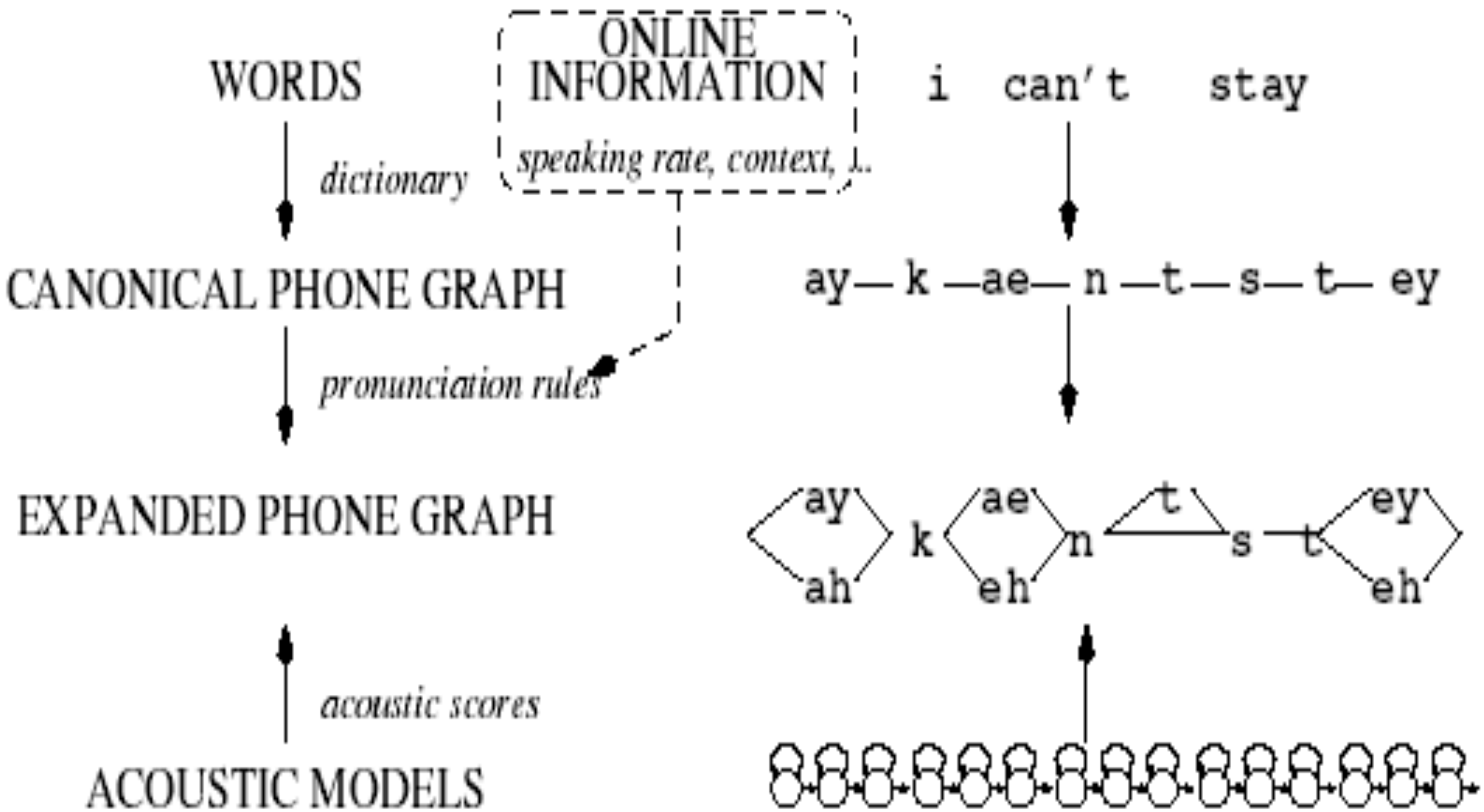


Fig. 10. A schematic view of transformation-based pronunciation modeling

Slide from Eric Fosler-Lussier

Summary

- Speech Recognition Architectural Overview
- Hidden Markov Models in general
  - Forward
  - Viterbi Decoding
- Hidden Markov models for Speech
- Evaluation