Speech and Language Processing

Chapter 9 of SLP
Automatic Speech Recognition (II)
Outline for ASR

- ASR Architecture
  - The Noisy Channel Model
- Five easy pieces of an ASR system
  1) Language Model
  2) Lexicon/Pronunciation Model (HMM)
  3) Feature Extraction
  4) Acoustic Model
  5) Decoder
- Training
- Evaluation
Acoustic Modeling
(= Phone detection)

- Given a 39-dimensional vector corresponding to the observation of one frame $o_i$
- And given a phone $q$ we want to detect
- Compute $p(o_i|q)$
- Most popular method:
  - GMM (Gaussian mixture models)
- Other methods
  - Neural nets, CRFs, SVM, etc
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
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 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 $\sigma^2$

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

- 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)
Vector Quantization

• Example

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

- *Example*

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

- 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 $b_j(v_k)$

- $b_j(v_k) = \frac{\text{number of vectors with codebook index } k \text{ in state } j}{\text{number of vectors in state } j} = \frac{14}{56} = \frac{1}{4}$
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 Gaussians
  - 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 $\mu_j$ and variance $\sigma_j^2$

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

P(shaded region) = 0.341
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 same value is omitted from all Gaussians, so argmax is still correct.
A Gaussian is parameterized by a mean and a variance:

- \( P(o|q) \):
  - \( P(o|q) \) is highest here at mean
  - \( P(o|q) \) is low here, very far from mean

\( P(o|q) \) is the probability density function of a Gaussian distribution, where \( o \) is the observation and \( q \) is the parameter of the distribution. Gaussians are widely used in acoustic modeling for their ability to model continuous data.
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:

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

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

\[
\sigma_i^2 = \frac{1}{T} \sum_{t=1}^{T} (o_t - \mu_i)^2 \quad \text{s.t. } o_t \text{ is state } i
\]
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 $\mu$ and variance $\sigma$:
  \[ f(x \mid \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \]

- Vector of means $\mu$ and covariance matrix $\Sigma$:
  \[ f(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right) \]
Multivariate Gaussians

- Defining $\mu$ and $\Sigma$

$$\mu = E(x)$$

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

- So the i-jth element of $\Sigma$ is:

$$\sigma_{ij}^2 = E[(x_i - \mu_i)(x_j - \mu_j)]$$
Gaussian Intuitions: Size of $\Sigma$

- $\mu = [0 \ 0]$
- $\Sigma = I$
- $\Sigma = 0.6I$
- $\Sigma = 2I$

- As $\Sigma$ becomes larger, Gaussian becomes more spread out; as $\Sigma$ becomes smaller, Gaussian more compressed.
$O_1$ 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.
- Different variances in different dimensions
Gaussian Intuitions: Off-diagonal

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

\[ \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix} \]
Gaussian Intuitions: off-diagonal

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

\[
\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}
\]
Gaussian Intuitions: off-diagonal and diagonal

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

\[
\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}
\]
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 $\sigma_{ii}^2$
- So this means we consider the variance of each acoustic feature (dimension) separately
Diagonal covariance

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

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

$$f(x | \mu, \sigma) = \frac{1}{2\pi^{D/2} \prod_{d=1}^{D} \sigma_d^2} \exp \left( -\frac{1}{2} \sum_{d=1}^{D} \left( \frac{x_d - \mu_d}{\sigma_d^2} \right)^2 \right)$$
Baum-Welch reestimation equations for multivariate Gaussians

- Natural extension of univariate case, where now $\mu_i$ 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:

- Solution: Mixtures of Gaussians

7/30/08

Figure from Chen, Picheney et al slides
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 $D \times D$
  - Or more likely
    - $M$ Diagonal Covariance Matrices of $D \times D$
    - which is equivalent to
    - $M$ Variance Vectors of dimensionality $D$
Where we are

- Given: A wave file
- Goal: output a string of words
- What we know: the **acoustic model**
  - How to turn the wavefile into a sequence of acoustic feature vectors, one every 10 ms
  - If we had a complete phonetic labeling of the training set, we know how to train a gaussian “phone detector” for each phone.
  - We also know how to represent each word as a sequence of phones
- What we knew from Chapter 4: **the language model**
- Next:
  - Seeing all this back in the context of HMMs
  - Search: how to combine the language model and the acoustic model to produce a sequence of words
Decoding

- In principle:

\[ \hat{W} = \arg\max_{W \in \mathcal{L}} \text{likelihood} \frac{P(O|W)}{P(W)} \text{ prior} \]

- In practice:

\[ \hat{W} = \arg\max_{W \in \mathcal{L}} P(O|W)P(W)^{LMSF} \]

\[ \hat{W} = \arg\max_{W \in \mathcal{L}} P(O|W)P(W)^{LMSF} WIP^N \]

\[ \hat{W} = \arg\max_{W \in \mathcal{L}} \log P(O|W) + LMSF \times \log P(W) + N \times \log WIP \]
Why is ASR decoding hard?

[ay d ih s hh er d s ah m th ih ng ax b aw m uh v ih ng r ih s en l ih]
HMMs for speech

\[ Q = q_1 q_2 \ldots q_N \]  a set of states corresponding to subphones

\[ A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn} \]  a transition probability matrix \( A \), each \( a_{ij} \) representing the probability for each subphone of taking a self-loop or going to the next subphone. Together, \( Q \) and \( A \) implement a pronunciation lexicon, an HMM state graph structure for each word that the system is capable of recognizing.

\[ B = b_i(o_t) \]  A set of observation likelihoods, also called emission probabilities, each expressing the probability of a cepstral feature vector (observation \( o_t \)) being generated from subphone state \( i \).
HMM for digit recognition task

Lexicon

<table>
<thead>
<tr>
<th>Digit</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>w ah n</td>
</tr>
<tr>
<td>two</td>
<td>t uw</td>
</tr>
<tr>
<td>three</td>
<td>th r ly</td>
</tr>
<tr>
<td>four</td>
<td>f ao r</td>
</tr>
<tr>
<td>five</td>
<td>f ay v</td>
</tr>
<tr>
<td>six</td>
<td>s ih ks</td>
</tr>
<tr>
<td>seven</td>
<td>s eh v ax n</td>
</tr>
<tr>
<td>eight</td>
<td>ey t</td>
</tr>
<tr>
<td>nine</td>
<td>n ay n</td>
</tr>
<tr>
<td>zero</td>
<td>z ly r ow</td>
</tr>
<tr>
<td>oh</td>
<td>ow</td>
</tr>
</tbody>
</table>

Phone HMM

Start

p("one")

p("two")

p("zero")

p("oh")

End
The Evaluation (forward) problem for speech

- The observation sequence $O$ is a series of MFCC vectors
- The hidden states $W$ are the phones and words
- For a given phone/word string $W$, our job is to evaluate $P(O|W)$
- Intuition: how likely is the input to have been generated by just that word string $W$
Evaluation for speech: Summing over all different paths!

- f ay ay ay ay v v v v
- f f ay ay ay ay v v v
- f f f f ay ay ay ay v
- f f ay ay ay ay ay ay ay v
- f f ay ay ay ay ay ay ay ay ay v
- f f ay v v v v v v v
- f f ay v v v v v v v v
The forward lattice for “five”
The forward trellis for “five”

<table>
<thead>
<tr>
<th>V</th>
<th>0</th>
<th>0</th>
<th>0.008</th>
<th>0.0093</th>
<th>0.0114</th>
<th>0.00703</th>
<th>0.00345</th>
<th>0.00306</th>
<th>0.00206</th>
<th>0.00117</th>
</tr>
</thead>
<tbody>
<tr>
<td>AY</td>
<td>0</td>
<td>0.04</td>
<td>0.054</td>
<td>0.0664</td>
<td>0.0355</td>
<td>0.016</td>
<td>0.00676</td>
<td>0.000532</td>
<td>0.000109</td>
<td>0.000109</td>
</tr>
<tr>
<td>F</td>
<td>0.8</td>
<td>0.32</td>
<td>0.112</td>
<td>0.0224</td>
<td>0.00448</td>
<td>0.000896</td>
<td>0.000179</td>
<td>4.48e-05</td>
<td>1.12e-05</td>
<td>2.8e-06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>f</td>
<td>0.8</td>
<td>f</td>
<td>0.8</td>
<td>f</td>
<td>0.7</td>
<td>f</td>
<td>0.4</td>
<td>f</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>ay</td>
<td>0.1</td>
<td>ay</td>
<td>0.1</td>
<td>ay</td>
<td>0.3</td>
<td>ay</td>
<td>0.8</td>
<td>ay</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>v</td>
<td>0.6</td>
<td>v</td>
<td>0.6</td>
<td>v</td>
<td>0.4</td>
<td>v</td>
<td>0.3</td>
<td>v</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>0.4</td>
<td>p</td>
<td>0.4</td>
<td>p</td>
<td>0.2</td>
<td>p</td>
<td>0.1</td>
<td>p</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>iy</td>
<td>0.1</td>
<td>iy</td>
<td>0.1</td>
<td>iy</td>
<td>0.3</td>
<td>iy</td>
<td>0.6</td>
<td>iy</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Viterbi trellis for “five”
### Viterbi trellis for “five”

<table>
<thead>
<tr>
<th>Time</th>
<th>V</th>
<th>AY</th>
<th>F</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.008</td>
<td>f 0.8 ay 0.1 p 0.4 iy 0.1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.04</td>
<td>0.048</td>
<td>f 0.8 ay 0.1 p 0.4 iy 0.1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.32</td>
<td>0.112</td>
<td>f 0.7 ay 0.3 p 0.2 iy 0.3</td>
</tr>
<tr>
<td>4</td>
<td>0.0072</td>
<td>0.048</td>
<td>0.0269</td>
<td>f 0.4 ay 0.8 p 0.1 iy 0.6</td>
</tr>
<tr>
<td>5</td>
<td>0.00672</td>
<td>0.0224</td>
<td>0.0125</td>
<td>f 0.4 ay 0.8 p 0.1 iy 0.6</td>
</tr>
<tr>
<td>6</td>
<td>0.00403</td>
<td>0.00448</td>
<td>0.000896</td>
<td>f 0.4 ay 0.8 p 0.1 iy 0.6</td>
</tr>
<tr>
<td>7</td>
<td>0.00188</td>
<td>0.00538</td>
<td>0.000179</td>
<td>f 0.4 ay 0.8 p 0.1 iy 0.6</td>
</tr>
<tr>
<td>8</td>
<td>0.00161</td>
<td>0.00167</td>
<td>4.48e-05</td>
<td>f 0.5 ay 0.6 p 0.1 iy 0.5</td>
</tr>
<tr>
<td>9</td>
<td>0.000667</td>
<td>0.000428</td>
<td>1.12e-05</td>
<td>f 0.5 ay 0.6 p 0.1 iy 0.5</td>
</tr>
<tr>
<td>10</td>
<td>0.000493</td>
<td>8.78e-05</td>
<td>2.8e-06</td>
<td>f 0.5 ay 0.6 p 0.1 iy 0.5</td>
</tr>
</tbody>
</table>
Search space with bigrams
Viterbi trellis

\[ \begin{align*}
q_0 & \quad \cdots \quad \cdots \quad \cdots \\
\vdots & \quad \vdots \quad \vdots \quad \vdots \\
w_1 & \\
\vdots & \\
w_2 & \\
\vdots & \\
w_N & \\
\vdots & \\
q_c & \quad \cdots \quad \cdots \quad \cdots \\
\end{align*} \]

\[ P(w_N | w_1) \]

\[ P(w_2 | w_1) \]

\[ P(w_1 | w_1) \]
Viterbi backtrace
Evaluation

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

- Word Error Rate = 
  \[ \frac{100 \times (\text{Insertions} + \text{Substitutions} + \text{Deletions})}{\text{Total Word in Correct Transcript}} \]

Alignment example:

REF:  portable **** PHONE UPSTAIRS last night so
HYP:  portable FORM OF STORES last night so
Eval   I S S
WER = 100 \left(\frac{1+2+0}{6}\right) = 50\%
NIST sctk-1.3 scoring software:
Computing WER with sclite

- 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

<table>
<thead>
<tr>
<th>CONFUSION PAIRS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(972)</td>
</tr>
<tr>
<td>With &gt;= 1 occurances</td>
<td>(972)</td>
</tr>
</tbody>
</table>

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
Training
Reminder: Forward-Backward Algorithm

1) Initialize $\Phi=(A,B)$
2) Compute $\alpha, \beta, \xi$
3) Estimate new $\Phi'=(A,B)$
4) Replace $\Phi$ with $\Phi'$
5) If not converged go to 2
The Learning Problem: It’s not just Baum-Welch

- Network structure of HMM is always created by hand
  - no algorithm for double-induction of optimal structure and probabilities has been able to beat simple hand-built structures.
- Always Bakis network = links go forward in time
- Subcase of Bakis net: beads-on-string net:

  Baum-Welch only guaranteed to return local max, rather than global optimum
Complete Embedded Training

- Setting all the parameters in an ASR system
- Given:
  - training set: wavefiles & word transcripts for each sentence
  - Hand-built HMM lexicon
- Uses:
  - Baum-Welch algorithm
By analogy with $\xi$ earlier, let’s define the probability of being in state $j$ at time $t$ with the $k^{th}$ mixture component accounting for $o_t$:

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

Now,

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

$$\overline{c}_{jm} = \frac{\sum_{t=1}^{T} \xi_{tm}(j)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \xi_{tk}(j)}$$

$$\Sigma_{jm} = \frac{\sum_{t=1}^{T} \xi_{tm}(j)(o_t - \mu_j)(o_t - \mu_j)^T}{\sum_{t=1}^{T} \sum_{k=1}^{M} \xi_{tm}(j)}$$
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 $\mu$ and global variance
  2) Split into two Gaussians, with means $\mu \pm \epsilon$ (sometimes $\epsilon$ is 0.2$\sigma$)
  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 \( b_j(ot) \)
    - Transition probabilities \( a_{ij} \)
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

Transcription

Nine four oh two two

Lexicon

one  wah n
two  tuw
three thr iy
... ... 
eight ey t
nine  n ay n
zero  z i y r ow
oh  ow

Wavefile

Feature Extraction

n ay n f ao r ow t uw t uw

Raw HMM

Feature Vectors
 Initialization: “Flat start”

- Transition probabilities:
  - set to zero any that you want to be “structurally zero”
    - The $\gamma$ 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 $\mu$ and $\sigma$ of each state to global mean and variance of all training data
Embedded Training

- Now we have estimates for A and B
- So we just run the EM algorithm
- During each iteration, we compute forward and backward probabilities
- Use them to re-estimate A and B
- Run EM 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**
  
  \[
  \hat{a}_{ij} = \frac{n_{ij}}{n_i}
  \]
  
  For all pairs of emitting states, \(1 \leq i, j \leq N\)

- **Baum-Welch**
  
  \[
  \hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \gamma_t(i, j)}
  \]

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

  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

\[
\overline{\mu}_i = \frac{1}{N_i} \sum_{t=1}^{T} o_t \quad \text{s.t. } q_t = i
\]

\[
\overline{\sigma}_i^2 = \frac{1}{N_i} \sum_{t=1}^{T} (o_t - \mu_i)^2 \quad \text{s.t. } q_t = i
\]

- 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 $\Sigma$) compute:

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

- In log space:

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

- Where:
  - 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.
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: 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!
Summary

- ASR Architecture
  - The Noisy Channel Model
- Five easy pieces of an ASR system
  1) Language Model
  2) Lexicon/Pronunciation Model (HMM)
  3) Feature Extraction
  4) Acoustic Model
  5) Decoder
- Training
- Evaluation