automatic speech recognition
Recall our input to ASR

Is the spectrum the best input for our ASR systems?
The Mel-scale

• Human hearing is **not** equally sensitive to **all** frequencies.
  • We are **less** sensitive to frequencies $> 1$ kHz.

• A **mel** is a unit of pitch. Pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of **mels**.

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

(No need to memorize this either)
The Mel-scale filter bank

- To **mimic** the response of the **human ear** (and because it *can* improve speech recognition), we often discretize the spectrum using $M$ triangular **filters**.
  - **Uniform** spacing before 1 kHz, **logarithmic** after 1 kHz
Aside - Mel-Frequency Cepstral Coefficients

• Earlier ASR required additional *Cepstral* processing on the Mel Spectrum
• Used to separate the **source** (glottal waveform) from **filter** (vocal tract resonances)
• MFCCs are used in Assignment 3
• Details on how to calculate them can be found in the appendices (not tested)
• Neural ASR usually uses the Mel-Spectrum as input
  • Good at **de-correlating** source and filter by itself
GAUSSIAN MIXTURES
Classifying speech sounds

- Speech sounds can cluster. This graph shows vowels, each in their own colour, according to the 1$\text{st}$ two formants.
Classifying speakers

• Similarly, all of the speech produced by one speaker will cluster differently in the Mel space than speech from another speaker.
• We can decide if a given observation comes from one speaker or another.

\[
P(\text{Speaker 1}) > P(\text{Speaker 2})
\]

<table>
<thead>
<tr>
<th>MFCC</th>
<th>Time, (t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>...</td>
</tr>
</tbody>
</table>

Observation matrix
Fitting continuous distributions

• Since we are operating with continuous variables, we need to fit continuous probability functions to a discrete number of observations.

• If we assume the 1-dimensional data in this histogram is Normally distributed, we can fit a continuous Gaussian function simply in terms of the mean $\mu$ and variance $\sigma^2$. 
(Aside) Univariate (1D) Gaussians

• Also known as **Normal** distributions, $N(\mu, \sigma)$

\[
P(x; \mu, \sigma) = \frac{\exp \left( -\frac{(x-\mu)^2}{2\sigma^2} \right)}{\sqrt{2\pi\sigma}}
\]

• The parameters we can modify are $\theta = \langle \mu, \sigma^2 \rangle$
  • $\mu = E(x) = \int x \cdot P(x)dx$ (**mean**)
  • $\sigma^2 = E((x - \mu)^2) = \int (x - \mu)^2 P(x)dx$ (**variance**)

But we don’t have samples for all $x$...
Maximum likelihood estimation

• Given data $X = \{x_1, x_2, \ldots, x_n\}$, MLE produces an estimate of the parameters $\hat{\theta}$ by maximizing the likelihood, $L(X, \theta)$:

$$\hat{\theta} = \text{argmax}_\theta L(X, \theta)$$

where $L(X, \theta) = P(X; \theta) = \prod_{i=1}^n P(x_i; \theta)$.

• Since $L(X, \theta)$ provides a surface over all $\theta$, in order to find the highest likelihood, we look at the derivative

$$\frac{\delta}{\delta \theta} L(X, \theta) = 0$$

to see at which point the likelihood stops growing.
MLE with univariate Gaussians

• Estimate $\mu$:

$$L(X, \mu) = P(X; \mu) = \prod_{i=1}^{n} P(x_i; \theta) = \prod_{i=1}^{n} \frac{\exp \left( -\frac{(x_i - \mu)^2}{2\sigma^2} \right)}{\sqrt{2\pi\sigma}}$$

$$\log L(X, \mu) = -\frac{\sum_i (x_i - \mu)^2}{2\sigma^2} - n \log(\sqrt{2\pi\sigma})$$

$$\frac{\delta}{\delta\mu} \log L(X, \mu) = \frac{\sum_i (x_i - \mu)}{\sigma^2} = 0$$

$$\mu = \frac{\sum_i x_i}{n}$$

• Similarly, $\sigma^2 = \frac{\sum_i (x_i - \mu)^2}{n}$
Multivariate Gaussians

- When data is $d$-dimensional, the input variable is
  \[ \mathbf{x} = \langle x[1], x[2], \ldots, x[d] \rangle \]
  the mean is
  \[ \mathbf{\mu} = E(\mathbf{x}) = \langle \mu[1], \mu[2], \ldots, \mu[d] \rangle \]
  the covariance matrix is
  \[ \Sigma[i, j] = E(x[i]x[j]) - \mu[i]\mu[j] \]
  and
  \[ P(\mathbf{x}) = \frac{\exp\left(-\frac{(\mathbf{x} - \mathbf{\mu})^\top \Sigma^{-1} (\mathbf{x} - \mathbf{\mu})}{2}\right)}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \]

\(A^\top\) is the transpose of \(A\)
\(A^{-1}\) is the inverse of \(A\)
\(|A|\) is the determinant of \(A\)
Intuitions of covariance

• As values in $\Sigma$ become larger, the Gaussian spreads out.
• ($I$ is the identity matrix)
Intuitions of covariance

- Different values on the diagonal result in different variances in their respective dimensions.

\[ \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 0.6 \end{bmatrix} \]
Non-Gaussian observations

- Speech data are generally *not* unimodal.
- The observations below are **bimodal**, so fitting one Gaussian would not be representative.
Mixtures of Gaussians

- Gaussian mixture models (GMMs) are a weighted linear combination of $M$ component Gaussians, $\langle \Gamma_1, \Gamma_2, ..., \Gamma_M \rangle$:

$$P(\tilde{x}) = \sum_{j=1}^{M} P(\Gamma_j)P(\tilde{x} | \Gamma_j)$$
Observation likelihoods

• Assuming MFCC dimensions are independent of one another, the covariance matrix is diagonal – i.e., 0 off the diagonal.
• Therefore, the probability of an observation vector given a Gaussian from slide 20 becomes

\[
P(\hat{x}|\Gamma_m) = \exp\left(-\frac{1}{2} \sum_{i=1}^{d} \frac{(x[i] - \mu_m[i])^2}{\Sigma_m[i]} \right)
\]

\[
\frac{1}{(2\pi)^{\frac{d}{2}} \left( \prod_{i=1}^{d} \Sigma_m[i] \right)^{\frac{1}{2}}}
\]

• We imagine a GMM first chooses a Gaussian, then emits an observation from that Gaussian.
Mixtures of Gaussians

• If we knew which Gaussian generated each sample, we could learn $P(\Gamma_j)$ with MLE, but that data is hidden, so we must use...

$$P(\tilde{x}) = \sum_{j=1}^{M} P(\Gamma_j)P(\tilde{x}|\Gamma_j)$$
Expectation-Maximization for GMMs

- If $\omega_m = P(\Gamma_m)$ and $b_m(x_t) = P(x_t | \Gamma_m)$, 
  \[
P_\theta(x_t) = \sum_{m=1}^{M} \omega_m b_m(x_t)
  \]
  where $\theta = \{\omega_m, \mu_m, \Sigma_m\}$ for $m = 1..M$

- To estimate $\theta$, we solve $\nabla_\theta \log L(X, \theta) = 0$ where
  \[
  \log L(X, \theta) = \sum_{t=1}^{T} \log P_\theta(x_t) = \sum_{t=1}^{T} \log \sum_{m=1}^{M} \omega_m b_m(x_t)
  \]
Expectation-Maximization for GMMs

- We differentiate the log likelihood function w.r.t. $\mu_m[n]$ and set this to 0 to find the value of $\mu_m[n]$ at which the likelihood stops growing.

$$\frac{\delta \log L(X, \theta)}{\delta \mu_m[n]} = \sum_{t=1}^{T} \frac{1}{P_\theta(x_t)} \left[ \frac{\delta}{\delta \mu_m[n]} \omega_m b_m(x_t) \right] = 0$$
Expectation-Maximization for GMMs

- The **expectation step** gives us:
  \[ b_m(\bar{x}_t) = P(\bar{x}_t|\Gamma_m) \]
  \[ P(\Gamma_m|\bar{x}_t; \theta) = \frac{\omega_m b_m(\bar{x}_t)}{P_\theta(\bar{x}_t)} \]

- The **maximization step** gives us:
  \[ \mu_m = \frac{\sum_t P(\Gamma_m|\bar{x}_t; \theta)\bar{x}_t}{\sum_t P(\Gamma_m|\bar{x}_t; \theta)} \]
  \[ \Sigma_m = \frac{\sum_t P(\Gamma_m|\bar{x}_t; \theta)\bar{x}_t^2}{\sum_t P(\Gamma_m|\bar{x}_t; \theta)} - \mu_m^2 \]
  \[ \hat{\omega}_m = \frac{1}{T} \sum_{t=1}^{T} P(\Gamma_m|\bar{x}_t; \theta) \]

Proportion of overall probability contributed by \( m \)

Recall from slide 13, MLE wants:
- \( \mu = \frac{\sum_i x_i}{n} \)
- \( \sigma^2 = \frac{\sum_i (x_i - \mu)^2}{n} \)
Some notes...

- In the previous slide, the square of a vector, \( \vec{a}^2 \), is elementwise (i.e., `numpy.multiply`)
  - E.g., \([2, 3, 4]^2 = [4, 9, 16]\)

- Since \( \Sigma \) is diagonal, it can be represented as a vector.

\[
\mathbf{\Sigma} \mathbf{\Sigma} = \sum_t \mathbf{P}(\mathbf{\Gamma}_m|\mathbf{x}_t; \theta) \mathbf{x}_t^2 - \mu_m^2
\]

- Can \( \sigma_m^2 = \sum_t \mathbf{P}(\mathbf{\Gamma}_m|\mathbf{x}_t; \theta) \mathbf{x}_t^2 - \mu_m^2 \) become negative?
  - No.
    - This is left as an exercise, but only if you’re interested.
Speaker recognition

• **Speaker recognition**: The identification of a speaker among several speakers given only acoustics.

• Each speaker will produce speech according to different probability distributions.
  • We train a Gaussian mixture model for each speaker, given annotated data (mapping utterances to speakers).
  • We choose the speaker whose model gives the highest probability for an observation.
Recipe for GMM EM

- For each speaker, we learn a GMM given all $T$ frames of their training data.

1. **Initialize:** Guess $\theta = \langle \omega_m, \mu_m, \Sigma_m \rangle$ for $m = 1..M$ either uniformly, randomly, or by $k$-means clustering.

2. **E-step:** Compute $b_m(x_t)$ and $P(\Gamma_m|x_t; \theta)$.

3. **M-step:** Update parameters for $\langle \omega_m, \mu_m, \Sigma_m \rangle$ as described on slide 21.
SPEECH RECOGNITION
Consider what we want speech to do

Buy ticket... AC490... yes

My hands are in the air.

Put this there.

Can we just use GMMs?
Aspects of ASR systems in the world

- **Speaking mode**: Isolated word (e.g., “yes”) vs. continuous (e.g., “Hey Siri, ask Cortana for the weather”)
- **Speaking style**: Read speech vs. spontaneous speech; the latter contains many dysfluencies (e.g., stuttering, *uh*, *like*, ...)
- **Enrolment**: Speaker-dependent (all training data from one speaker) vs. speaker-independent (training data from many speakers).
- **Vocabulary**: Small (<20 words) or large (>50,000 words).
- **Transducer**: Cell phone? Noise-cancelling microphone? Teleconference microphone?
Speech is dynamic

- Speech **changes** over time.
  - GMMs are good for high-level clustering, but they encode **no notion** of order, sequence, nor time.

- Speech is an expression of **language**.
  - We want to incorporate knowledge of how phonemes and words are ordered with **language models**.
Speech is sequences of phonemes

/ow p ah n dh ah p aa d b ey d ao r z/

We want to convert a series of (e.g.) MFCC vectors into a sequence of phonemes.
Continuous HMMs (CHMM)

- A **continuous HMM** has observations that are distributed over continuous variables.
  - Observation probabilities, $b_i$, are also continuous.
  - E.g., here $b_0(\vec{x})$ tells us the probability of seeing the (multivariate) continuous observation $\vec{x}$ while in state 0.

\[
\vec{x} = \begin{pmatrix}
4.32957 \\
2.48562 \\
1.08139 \\
...
\end{pmatrix}
\]
Defining CHMMs

• Continuous HMMs are very similar to discrete HMMs.
  • $S = \{s_1, \ldots, s_N\}$: set of states (e.g., subphones)
  • $X = \mathbb{R}^d$: continuous observation space
  • $\Pi = \{\pi_1, \ldots, \pi_N\}$: initial state probabilities
  • $A = \{a_{ij}\}, i, j \in S$: state transition probabilities
  • $B = b_i(\tilde{x}), i \in S, \tilde{x} \in X$: state output probabilities (i.e., Gaussian mixtures)

yielding

• $Q = \{q_0, \ldots, q_T\}, q_i \in S$: state sequence
• $\mathcal{O} = \{\sigma_0, \ldots, \sigma_T\}, \sigma_i \in X$: observation sequence
Using CHMMs

• As before, these HMMs are *generative* models that encode statistical knowledge of how output is *generated*.

• We **train** CHMMs with **Baum-Welch** (a type of Expectation-Maximization), as we did before with discrete HMMs.
  • Here, the observation parameters, \( b_i(\bar{x}) \), are adjusted using the GMM training ‘recipe’ from earlier.

• We find the best state sequences using **Viterbi**, as before.
  • Here, the best state sequence gives us a **sequence of phonemes**
Phoneme dictionaries

• How do we convert our phoneme sequence into **words**?
• There are many **phonemic dictionaries** that map words to pronunciations (i.e., lists of phoneme sequences).
• The **CMU dictionary** ([http://www.speech.cs.cmu.edu/cgi-bin/cmudict](http://www.speech.cs.cmu.edu/cgi-bin/cmudict)) is popular.
  • 127K words transcribed with the ARPABet.
  • Includes some rudimentary **prosody markers**.

...  
EVOLUTION        EH2 V AH0 L UW1 SH AH0 N  
EVOLUTION (2)    IY2 V AH0 L UW1 SH AH0 N  
EVOLUTION (3)    EH2 V OW0 L UW1 SH AH0 N  
EVOLUTION (4)    IY2 V OW0 L UW1 SH AH0 N  
EVOLUTIONARY     EH2 V AH0 L UW1 SH AH0 N EH2 R IY0
The noisy channel model for ASR

$$W^* = \arg\max_{W} P(X|W)P(W)$$

How to encode $P(X|W)$?
Putting it together

• How do we combine the language model, phonemic dictionary, and CHMM together? → Nest them!

• Full details are an aside – see Appendices and J&M 2nd Ed.
EVALUATING SPEECH RECOGNITION
Evaluating ASR accuracy

• How can you tell how well an ASR system recognizes speech?
  • E.g., if somebody said
    Reference:  *how to recognize speech*
    but an ASR system heard
    Hypothesis:  *how to wreck a nice beach*
    how do we quantify the error?

• One measure is **word accuracy**: \#CorrectWords/\#ReferenceWords
  • E.g., 2/4, above
  • This runs into problems similar to those we saw with SMT.
    • E.g., the hypothesis ‘*how to recognize speech boing boing boing boing boing boing boing boing*’ has 100% accuracy by this measure.
    • Normalizing by \#HypothesisWords also has problems...
Word-error rates (WER)

- ASR enthusiasts are often concerned with **word-error rate (WER)**, which counts different **kinds** of errors that can be made by ASR at the word-level.
  - **Substitution error**: One word being mistook for another e.g., ‘*shift*’ given ‘*ship*’
  - **Deletion error**: An input word that is ‘skipped’ e.g. ‘*I Torgo*’ given ‘*I am Torgo*’
  - **Insertion error**: A ‘hallucinated’ word that was not in the input.
    e.g., ‘*This Norwegian parrot is no more*’ given ‘*This parrot is no more*’
Levenshtein distance

- The **Levenshtein** distance (and WER) is straightforward to calculate using dynamic programming.

Allocate matrix $R[n + 2, m + 2]$  
// where $n$ is the number of reference words  
// and $m$ is the number of hypothesis words

Add `<s>` to beginning of each sequence, and `</s>` to their ends.

Fill [0:end] along the first row and column.

for $i := 1..n + 1$ // #ReferenceWords
  for $j := 1..m + 1$ // #Hypothesis words
    $R[i, j] := \min(\)
      \begin{align*}
      R[i - 1, j] + 1, & \quad \text{deletion} \\
      R[i - 1, j - 1], & \quad \text{if the } i^{th} \text{ reference word and} \\
      R[i - 1, j - 1] + 1, & \quad \text{if they differ, i.e., substitution} \\
      R[i, j - 1] + 1) & \quad \text{insertion}
      \end{align*}$

Return $100 \times R[n, m]/n$  // WER
Levenshtein distance – initialization

<table>
<thead>
<tr>
<th>Reference</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;s&gt;</td>
</tr>
<tr>
<td>&lt;s&gt;</td>
<td>0</td>
</tr>
<tr>
<td>how</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
</tr>
<tr>
<td>recognize</td>
<td>3</td>
</tr>
<tr>
<td>speech</td>
<td>4</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>5</td>
</tr>
</tbody>
</table>

The value at cell \((i, j)\) is the minimum number of errors necessary to align \(i\) with \(j\).
Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;s&gt;</td>
<td>0</td>
</tr>
<tr>
<td>how</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
</tr>
<tr>
<td>wreck</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
</tr>
<tr>
<td>nice</td>
<td>5</td>
</tr>
<tr>
<td>beach</td>
<td>6</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>7</td>
</tr>
</tbody>
</table>

- \[ R[1,1] = \min(LEFT + 1, (0), ABOVE + 1) = 0 \text{ (match)} \]
- We put a little arrow in place to indicate the choice.
  - ‘Arrows’ are normally stored in a backtrace matrix.
Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>hypothesis</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&lt;s&gt;</td>
<td>how</td>
<td>to</td>
<td>wreck</td>
<td>a</td>
<td>nice</td>
<td>beach</td>
</tr>
<tr>
<td>&lt;s&gt;</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>how</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>to</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We continue along for the first reference word...
- These are all insertion errors
Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;s&gt;  how   to  wreck  a  nice  beach  &lt;/s&gt;</td>
</tr>
<tr>
<td>&lt;s&gt;</td>
<td>0   1  2   3   4   5   6   7</td>
</tr>
<tr>
<td>how</td>
<td>1   0  1   2   3   4   5   6</td>
</tr>
<tr>
<td>to</td>
<td>2   1  0   1   2   3   4   5</td>
</tr>
<tr>
<td>recognize</td>
<td>3   2  1   1   2   3   4   5</td>
</tr>
<tr>
<td>speech</td>
<td>4</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>5</td>
</tr>
</tbody>
</table>

- Since `recognize ≠ wreck`, we have a substitution error.
- At some points, you have >1 possible path as indicated.
  - We can prioritize types of errors arbitrarily.
Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;s&gt;</td>
<td>&lt;s&gt;</td>
</tr>
<tr>
<td>how</td>
<td>how</td>
</tr>
<tr>
<td>to</td>
<td>to</td>
</tr>
<tr>
<td>wreck</td>
<td>wreck</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>nice</td>
<td>nice</td>
</tr>
<tr>
<td>beach</td>
<td>beach</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>&lt;/s&gt;</td>
</tr>
</tbody>
</table>

- And we finish the grid.
- There are $R[end, end] = 4$ word errors and a WER of $4/4 = 100\%$.
  - WER can be greater than 100\% (relative to the reference).
### Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>&lt;s&gt;</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
<th>&lt;/s&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;s&gt;</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>how</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>recognize</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>speech</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

- If we want, we can **backtrack** using our arrows (in a backtrace matrix).
- Here, we estimate 2 **substitution** errors and 2 **insertion** errors.
NEURAL SPEECH RECOGNITION
Remember Viterbi

The best path to state $s_j$ at time $t$, $\delta_j(t)$, depends on the best path to each possible previous state, $\delta_i(t-1)$, and their transitions to $j$, $a_{ij}$.

\[
\delta_j(t) = \max_i \left[ \delta_i(t-1) a_{ij} b_j(\sigma_t) \right]
\]

\[
\psi_j(t) = \text{argmax}_i \left[ \delta_i(t-1) a_{ij} \right]
\]

Do these probabilities need to be GMMs?

Observations, $\sigma_t$

$\sigma_0 = \text{ship}$

$\sigma_1 = \text{frock}$

$\sigma_2 = \text{tops}$
Replacing GMMs with DNNs

• Obtain $b_j(x) = p(x|s_j)$ with a neural network.
• Instead of learning a continuous distribution directly, we can use Bayes’ rule:

$$p(x|s_j) = \frac{p(s_j|x) \cdot p(x)}{p(s_j)}$$
Replacing GMMs with DNNs

- The probability of a word sequence $W$ comes *loosely* from $P(X|W)$

\[
\approx \max_{q_1 \cdots q_T} \prod_{t=1}^{T} P(q_t|q_{t-1}) P(x_t|q_t) \approx \max_{q_1 \cdots q_T} \prod_{t=1}^{T} P(q_t|q_{t-1}) \frac{P(q_t|x_t)}{P(q_t)}
\]

HMM

DNN
Training the DNN

• Maximize $P(q_t | x_t)$
• The order which we transition through states ($\approx$ phonemes) is known by the transcription (ignoring alternate pronunciations)
• At what frames these transitions happen are unknown
  • $\because q_t$ is unknown!
• Solution: bootstrapping
  • Use another model to determine $q_t$
  • Often argmax$_{q_1...T} P_{\text{CHMM}}(q_1...T, x_1...T)$ from GMM-HMM
    • $P_{\text{CHMM}}(q_t | q_{t-1})$ often stolen as well
    • ... and $P(q_t)$
• Other, advanced methods exist
Hybrid HMM and DNN

What are these DNNs learning?

- **t-SNE** (stochastic neighbour embedding using t-distribution) visualizations in 2D (colours=speakers).
- Deeper layers encode information about the **segment**

What are these DNNs learning?

- DNN trained to classify phonemes
- t-SNE visualizations of hidden layer.
- Lower layers detect manner of articulation

Figure 1: Multilingual BN features of five vowels from French (+), German (□) and Spanish (▽): /a/ (black), /i/ (blue), /e/ (green), /o/ (red), and /u/ (yellow)

End-to-end neural networks

• Neural networks are typically trained at the frame level.
  • This requires a separate training target for every frame, which in turn requires the alignment between the audio and transcription sequences to be known.
  • However, the alignment is only reliable once the classifier is trained.
• “End-to-end” ≈ an objective function that allows sequence transcription without requiring prior alignment between the input $X$ (frames of audio) and target $Y$ (output strings) sequences with arbitrary lengths, i.e.

$$P(Y|X)$$

• Target tokens can be words, sub-words, or just characters
• Two popular choices of $P(Y|X)$:
  1. Seq2seq (encoder/decoder, transformers)
  2. Connectionist Temporal Classification
Seq2seq architectures

• The **same architectures** we saw in NMT work for ASR!
• Replace source embedding vector $x_t$ with Mel spectrum vector
• Replace target sequence $E$ with transcription sequence $Y$

... 

That’s it.
Aside – Listen, Attend, and Spell

https://arxiv.org/abs/1508.01211
Connectionist Temporal Classification

• Consider alignment:

\[
x_1 x_2 x_3 x_4 x_5 x_6 \quad \text{input (} X \text{)}
\]
\[
\begin{array}{ccc}
  c & c & a & a & t \\
  c & a & t \\
\end{array} \quad \text{alignment}
\]
\[
\begin{array}{ccc}
  c & a & t \\
\end{array} \quad \text{output (} Y \text{)}
\]

• Not every input step needs an output. How can we collapse alignments for multi-character output (like, ‘his’ vs ‘hiss’)?
• CTC introduces ‘blank token’ $\epsilon$ as a placeholder

See: https://distill.pub/2017/ctc/
Connectionist Temporal Classification

We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.

The network gives $p_t(a | X)$, a distribution over the outputs \{h, e, l, o, $\epsilon$\} for each input step.

With the per time-step output distribution, we compute the probability of different sequences.

By marginalizing over alignments, we get a distribution over outputs.

This is computed by an RNN.

$$p(Y | X) = \sum_{A \in A_{x,y}} \prod_{t=1}^{T} p_t(a_t | X)$$

The CTC conditional probability marginalizes over the set of valid alignments computing the probability for a single alignment step-by-step.

See: [https://distill.pub/2017/ctc/](https://distill.pub/2017/ctc/)
Connectionist Temporal Classification

Summing over all alignments can be very expensive.

Dynamic programming merges alignments, so it's much faster.

See: https://distill.pub/2017/ctc/
Connectionist Temporal Classification

See: https://distill.pub/2017/ctc/
Connectionist Temporal Classification

- It is still expensive to consider all possible alignments, and it is naïve to merely pick the max probability at each time step.
- We therefore introduce a beam search (like in NMT)

See: https://distill.pub/2017/ctc/
End-to-end neural networks

State-of-the-art?

State-of-the-art?

- **Input**: spectrogram
- **Output**: characters (incl. space and null characters)
- No phonemes or vocabulary means no OOV words.

State-of-the-art?

<table>
<thead>
<tr>
<th>Model</th>
<th>SWB</th>
<th>CH</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vesely et al. (GMM-HMM BMMI) [44]</td>
<td>18.6</td>
<td>33.0</td>
<td>25.8</td>
</tr>
<tr>
<td>Vesely et al. (DNN-HMM sMBR) [44]</td>
<td>12.6</td>
<td>24.1</td>
<td>18.4</td>
</tr>
<tr>
<td>Maas et al. (DNN-HMM SWB) [28]</td>
<td>14.6</td>
<td>26.3</td>
<td>20.5</td>
</tr>
<tr>
<td>Maas et al. (DNN-HMM FSH) [28]</td>
<td>16.0</td>
<td>23.7</td>
<td>19.9</td>
</tr>
<tr>
<td>Seide et al. (CD-DNN) [39]</td>
<td>16.1</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Kingsbury et al. (DNN-HMM sMBR HF) [22]</td>
<td>13.3</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Sainath et al. (CNN-HMM) [36]</td>
<td>11.5</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Soltau et al. (MLP/CNN+I-Vector) [40]</td>
<td><strong>10.4</strong></td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Deep Speech SWB</td>
<td>20.0</td>
<td>31.8</td>
<td>25.9</td>
</tr>
<tr>
<td>Deep Speech SWB + FSH</td>
<td>12.6</td>
<td><strong>19.3</strong></td>
<td><strong>16.0</strong></td>
</tr>
</tbody>
</table>

Table 3: Published error rates (%WER) on Switchboard dataset splits. The columns labeled “SWB” and “CH” are respectively the easy and hard subsets of Hub5’00.

Summary

• We’ve seen how to:
  • extract useful speech features with Mel-scale filter banks
  • cluster multi-modal speech data with Gaussian mixture models.
  • recognize speech with hidden Markov models and neural networks.
  • Recognize speech using only end-to-end neural networks.
  • evaluate ASR performance with Levenshtein distance.

• Next, we’ll see how to synthesize artificial speech.
APPENDICES
(EVERYTHING THAT FOLLOWS IS AN ASIDE. NOT ON THE EXAM.)
APPENDIX: CEPSTRUM AND MFCCS
Source and filter

• The **acoustics** of speech are produced by a glottal pulse waveform (the **source**) passing through a vocal tract whose shape modifies that wave (the **filter**).

• The **shape** of the vocal tract is more important to phoneme recognition.
  • *We want to separate the source from the filter in the acoustics.*
Source and filter

• Since speech is assumed to be the output of a linear time invariant system, it can be described as a convolution.
  • Convolution, $x * y$, is beyond the scope of this course, but can be conceived as the modification of one signal by another.

• For speech signal $x[n]$, glottal signal $g[n]$, and vocal tract transfer $v[n]$ with spectra $X[z]$, $G[z]$, and $V[z]$, respectively:

$$x[n] = g[n] * v[n]$$

$$X[z] = G[z]V[z]$$

$$\log X[z] = \log G[z] + \log V[z]$$

We’ve separated the source and filter into two terms!
The cepstrum

• We separate the source and the filter by pretending the log of the spectrum is actually a time domain signal.
  • the log spectrum \( \log X[z] \) is a sum of the log spectra of the source and filter, i.e., a superposition; finding its spectrum will allow us to isolate these components.

• Cepstrum: \( n. \) the spectrum of the log of the spectrum.
  • Fun fact: ‘ceps’ is the reverse of ‘spec’. Instead of ‘filters’ we have ‘lifters’…

![Diagram showing the process of separating source and filter using cepstrum.](image)
The cepstrum

- The domain of the cepstrum is *quefreny* (a play on the word ‘frequency’).
The cepstrum

This is due to the vocal tract shape

This is due to the glottis

Pictures from John Coleman (2005)
Mel-frequency cepstral coefficients

- **Mel-frequency cepstral coefficients (MFCCs)** are a popular representation of speech used in ASR.
- They are the *spectra* of the logarithms of the *Mel-scaled filtered spectra* of the *windows* of the *waveform*.
MFCCs in practice

• An observation vector of MFCCs often consists of
  • The **first 13 cepstral coefficients** (i.e., the first 13 dimensions produced by this method),
  • An additional **overall energy** measure,
  • The **velocities** ($\delta$) of each of those 14 dimensions,
    • i.e., the rate of change of each coefficient at a given time
  • The **accelerations** ($\delta\delta$) of each of original 14 dimensions.

• The result is that at a timeframe $t$ we have an observation MFCC vector of $(13+1) \cdot 3 = 42$ dimensions.
  • This vector is what is used by our ASR systems...
Advantages of MFCCs

• The cepstrum produces highly uncorrelated features (every dimension is useful).
  • This includes a separation of the source and filter.

• Historically, the cepstrum has been easier to learn than the spectrum for phoneme recognition.

• “tl;dr: Use Mel-scaled filter banks if the [ML] algorithm is not susceptible to highly correlated input. Use MFCCs if the [ML] algorithm is susceptible to correlated input.” – Haytham Fayek
APPENDIX: PHONEME HMMS AND COMPOSITION
Phoneme HMMs

• Phonemes *change* over time – we model these dynamics by building one HMM for *each* phoneme.
  • Tristate phoneme models are popular.
    • The centre state is often the ‘steady’ part.

tristate phoneme model (e.g., /oi/)
Phoneme HMMs

- We train each phoneme HMM using all sequences of that phoneme.

<table>
<thead>
<tr>
<th>MFCC</th>
<th>Time, $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>85 96</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>...</td>
</tr>
</tbody>
</table>

$t_1$ $t_2$ phn

... 64 85 ae
85 96 sh
96 102 epi
102 106 m
...

Phoneme HMMs

/iy/
/ih/
/eh/
/s/
/sh/

annotation observations
Putting it together

“open the pod bay doors”

Language model

Acoustic model
Combining models

• We can learn an $N$-gram **language model** from word-level transcriptions of speech data.
  • These models are discrete and are trained using MLE.

• Our phoneme HMMs together constitute our **acoustic model**.
  • Each phoneme HMM tells us how a phoneme ‘sounds’.

• We can **combine** these models by **concatenating** phoneme HMMs together according to a known lexicon.
  • We use a word-to-phoneme dictionary.
Combining models

- If we know how phonemes combine to make words, we can simply **concatenate** together our phoneme models by inserting and **adjusting** transition weights.
  - e.g., *Zipf* is pronounced /z ih f/, so...

(It’s more complicated: 1) the HMMs are often more complex, 2) they often represent phonemes *in context* of other phonemes 3) ... )
Concatenating phoneme models

From Jurafsky & Martin text
Bigram models

From Jurafsky & Martin text
APPENDIX: OTHER NEURAL ARCHITECTURES AND IMPLEMENTATIONS
End-to-end hybrids

• Get word boundaries from some external tool.
• Train word/characters and acoustics simultaneously.
• Obtain up to 0.11% improvement in error rates

Convolutioonal Neural Networks

• Spectrograms are kinds of images, so let’s use the kinds of neural networks used in computer vision.
The open-source Kaldi ASR

• Kaldi is the *de-facto* open-source ASR toolkit: http://kaldi-asr.org
  • It has pretrained models, including the ASpIRE chain model trained on Fisher English, augmented with impulse responses and noises to create multi-condition training.
  • My favourite incarnation uses I-Vectors to account for the speaker.
  • It often (anecdotally) performs better than Google’s SpeechAPI.
  • It is originally in C++, but a wrapper (PyTorch-Kaldi) exists in the much easier Python.
  • Pro-sanity tip: don’t read news about its progenitor.
LAS, Transformers, and the RNN-T (extends CTC) are reaching state of the art, e.g.

For **HOT** news and architectures, see [https://github.com/syhw/wer_are_we](https://github.com/syhw/wer_are_we)
APPENDIX: SPEAKER ADAPTATION
Speaker adaptation

• Given a neural ASR system trained with many speakers, we want to adapt to the voice of a new individual.
• We know how to do this with HMMs
  • e.g., with interpolation, or (aside) with MAP or MLLR training.

• DNNs need *lots* of data to be useful, but we can adapt:
  • **Conservative:** re-train whole DNN, with some constraints
  • **Transformative:** only retrain one layer (or a few)
  • **Speaker-aware:** do not really train the parameters
Conservative speaker adaptation

- Stopping criteria can exist on output, parameters, or meta-aspects of training

- All of Amazon or Facebook’s secret recordings of billions of people in the bathroom

- Tiny database of you in the bathroom
Transformative speaker adaptation

- Insert a new layer.
- Keeping all other parameters fixed, train the new ones to normalize speaker information.

- There are many alternatives...

Tiny database of you in the bathroom
Speaker-aware training

• Fixed length low dimension vectors, obtained in a variety of ways.

• Note we can segment things by recording device, noise, etc.

• This can be used to remove the channel effect.

Speaker-aware training

Training data:

Speaker 1

Speaker 2

Acoustic features augmented with speaker vectors

Testing data:

All speakers use the same DNN model
Different speakers augmented by different features
APPENDIX: CLUSTERING
Clustering

• **Quantization** involves turning possibly *multi-variate* and *continuous* representations into *univariate discrete* symbols.
  • Reduced storage and computation costs.
  • Potentially tremendous loss of information.

• Observation $X$ is in Cluster One, so we replace it with $1$.

• Clustering is *unsupervised* learning.
  • Number and form of clusters often unknown.
Aspects of clustering

• What defines a particular cluster?
  • Is there some prototype representing each cluster?

• What defines membership in a cluster?
  • Usually, some distance metric $d(x, y)$ (e.g., Euclidean distance).

• How well do clusters represent unseen data?
  • How is a new point assigned to a cluster?
  • How do we modify that cluster as a result?
K-means clustering

• Used to group data into $K$ clusters, $\{C_1, \ldots, C_K\}$.

• Each cluster is represented by the mean of its assigned data.
  • (sometimes it’s called the cluster’s centroid).

• Iterative algorithm converges to local optimum:
  1. Select $K$ initial cluster means $\{\mu_1, \ldots, \mu_K\}$ from among data points.
  2. Until (stopping criterion),
     a) Assign each data sample to closest cluster
        $x \in C_i$ if $d(x, \mu_i) \leq d(x, \mu_j), \ \forall i \neq j$
     b) Update $K$ means from assigned samples
        $\mu_i = E(x) \ \forall \ x \in C_i, \ 1 \leq i \leq K$
**K-means example** \( (K = 3) \)

- Initialize with a random selection of 3 data samples.
- Euclidean distance metric \( d(x, \mu) \)

![Diagram showing the K-means algorithm process](image)
**K-means stopping condition**

- The total **distortion**, $\mathcal{D}$, is the sum of squared error,

\[
\mathcal{D} = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2
\]

- $\mathcal{D}$ decreases between $n^{th}$ and $(n + 1)^{th}$ iteration.

- We can stop training when $\mathcal{D}$ falls below some threshold $\mathcal{T}$.

\[
1 - \frac{\mathcal{D}(n + 1)}{\mathcal{D}(n)} < \mathcal{T}
\]
Acoustic clustering example

• 12 clusters of spectra, after training.
Number of clusters

- The number of true clusters is unknown.
- We can iterate through various values of $K$.
  - As $K$ approaches the size of the data, $D$ approaches 0...
Hierarchical clustering

- Hierarchical clustering clusters data into hierarchical ‘class’ structures.

- Two types: top-down (divisive) or bottom-up (agglomerative).

- Often based on greedy formulations.

- Hierarchical structure can be used for hypothesizing classes.
Divisive clustering

• Creates hierarchy by successively splitting clusters into smaller groups.
Agglomerative clustering

- **Agglomerative clustering** starts with $N$ ‘seed’ clusters and iteratively combines these into a hierarchy.

- On each iteration, the two most similar clusters are merged together to form a new meta-cluster.

- After $N - 1$ iterations, the hierarchy is complete.

- Often, when the similarity scores of new meta-clusters are tracked, the resulting graph (i.e., **dendogram**) can yield insight into the natural grouping of data.
Dendogram example
Speaker clustering

• 23 female and 53 male speakers from TIMIT.
• Data are vectors of average F1 and F2 for 9 vowels.
• Distance $d(C_i, C_j)$ is average of distances between members.
Acoustic-phonetic hierarchy

(this is basically an upside-down dendogram)
Word clustering