automatic speech recognition
Recall our input to ASR

Frame

Spectrum

Amplitude

Frequency (Hz)

Is the spectrum the best input for our ASR systems?
1. The Mel-scale filter bank

- To mimic the response of the human ear (and because it empirically improves speech recognition), we often discretize the spectrum using $M$ triangular filters.
  - Uniform spacing before 1 kHz, logarithmic after 1 kHz
2. 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 to separate the source from the filter in the acoustics.*
2. Source and filter (aside)

• Since speech is assumed to be the output of a linear time invariant system, it can be described as a convolution.
  • Convolution, $x \ast y$, 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:

$$x[n] = g[n] \ast v[n]$$
$$X[z] = G[z]V[z]$$
$$\log X[z] = \log G[z] + \log V[z]$$

We’ve separated the source and filter into two terms!
2. 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’...
2. The cepstrum

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

This is due to the vocal tract shape

This is due to the glottis

Spectrum

Cepstrum

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

- **Mel-frequency cepstral coefficients (MFCCs)** are the most 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**.
Advantages of MFCCs

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

• In practice, the cepstrum **has been easier to learn** than the spectrum for phoneme recognition.

• There is an efficient method to compute cepstra called the **discrete cosine transform**.
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 $(13+1)*3=42$ dimensions.
  • This vector is what is used by our ASR systems...
GAUSSIAN CLUSTERS
Classifying speech sounds

- Speech sounds tend to cluster. This graph shows vowels, each in their own colour, according to the 1st two formants.

Note: The vowel trapezoid’s dimensions were physical.
Classifying speakers

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

\[
P(\text{|
\begin{array}{c}\text{Observation matrix}
\end{array}\text{|
\begin{array}{c}P(0)\end{array}}) > P(\text{|
\begin{array}{c}\text{Observation matrix}
\end{array}\text{|
\begin{array}{c}P(1)\end{array}})
\]
Fitting continuous distributions

• In this course, we used **discrete** probability functions.
• Since we are now 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 $\sigma^2$. 
Comparing continuous distributions

• Moreover, if we observe a particular value in this univariate space, e.g., $x = 15$, we can say which of several distributions is most likely to have produced it.
  • Here, distribution $B$ is more likely to have produced $x = 15$ because $P(x; B) > P(x; A)$.
Good fits

• Given some fixed \textit{training data}, we want to be able to fit continuous probability functions that \textit{best match} our observations.
• The data in \textit{this histogram} are \textit{more likely} to have been produced from the parameterization on the \textit{left}.
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, $L(X, \theta)$:

$$\hat{\theta} = \arg\max_{\theta} L(X, \theta)$$

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

- \(A^\top\) 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 – 0 except for 1s on the diagonal)
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 is generally *not* unimodal – it’s more complex.
• The observations below are **bimodal**, so fitting one Gaussian would not be representative.
  • E.g., if you usually keep your phone in your desk or on your table, it makes no sense looking for them floating in the air between them.
Mixtures of Gaussians

• Gaussian mixture models (GMMs) are a weighted linear combination of $M$ component Gaussians, $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_M\}$:

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

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

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

• If we knew \textit{which} Gaussian generated each sample, we could learn \( P(\Gamma_j) \) with MLE, but that data is \textit{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 $b_m(x_t) = P(x_t|\Gamma_m)$, then

$$P_\theta(x_t) = \sum_{m=1}^{M} \omega_m b_m(x_t)$$

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.

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

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

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

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

• Since $\Sigma$ is diagonal, it can be represented as a vector.

• Can $\overrightarrow{\sigma_m^2} = \frac{\Sigma_t P(\Gamma_m|\vec{x}_t; \theta)\vec{x}_t^2}{\Sigma_t P(\Gamma_m|\vec{x}_t; \theta)} - \mu_m^2$ 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 some 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 30.

• (see the Reynolds & Rose (1995) paper on the course webpage for details)
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 $\mathbf{X}$ 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
        $$ x \in C_i \quad \text{if} \quad d(x, \mu_i) \leq d(x, \mu_j), \quad \forall i \neq j $$
     b) Update $K$ means from assigned samples
        $$ \mu_i = E(x) \quad \forall \ x \in C_i, \quad 1 \leq i \leq K $$
**K-means example \((K = 3)\)**

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

$$
1 - \frac{\mathcal{D}(n + 1)}{\mathcal{D}(n)} < T
$$
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...

![Diagrams showing clustering with different $K$ values: $K = 2$ and $K = 4$.]
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.
Dendrogram 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

city names

numbers

Time, price modifiers
SPEECH RECOGNITION
Consider what we want speech to do

Put this there.

My hands are in the air.

Buy ticket... AC490... yes

Put this there.

Can we just use GMMs?
Speech databases

• Large-vocabulary continuous ASR is meant to encode full conversational speech, with a vocabulary of >64K words.
  • This requires *lots* of data to train our models.

• The **Switchboard** corpus contains 2430 conversations spread out over about 240 hours of data (~14 GB).
• The **TIMIT** database contains 63,000 sentences from 630 speakers.
  • Relatively small (~750 MB), but very popular.
• Speech data from conferences (e.g., **TED**) or from broadcast news tends to be between 3 GB and 30 GB.
Aspects of ASR systems in the world

- **Speaking mode**: Isolated word (e.g., “yes”) vs. continuous (e.g., “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, or 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 (*)

We want to convert a series of MFCC vectors into a sequence of phonemes.

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

open(podBay.doors);

“open the pod bay doors”

(*) not really
Phoneme dictionaries

• 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  

Annotation/transcription

• Speech data must be **segmented** and **annotated** in order to be useful to an ASR learning component.
• Programs like Wavesurfer or Praat allow you to demarcate where a phoneme begins and ends in time.
Putting it together?

“open the pod bay doors”

Language model

Acoustic model
The noisy channel model for ASR

\[ W^* = \underset{W}{\text{argmax}} \ P(X|W)P(W) \]

How to encode \( P(X|W) \)?
Reminder – discrete HMMs

• Previously we saw **discrete HMMs**: at each state we observed a discrete symbol from a finite set of discrete symbols.

<table>
<thead>
<tr>
<th>word</th>
<th>P(word)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ship</td>
<td>0.1</td>
</tr>
<tr>
<td>pass</td>
<td>0.05</td>
</tr>
<tr>
<td>camp</td>
<td>0.05</td>
</tr>
<tr>
<td>flock</td>
<td>0.6</td>
</tr>
<tr>
<td>soccer</td>
<td>0.05</td>
</tr>
<tr>
<td>mother</td>
<td>0.1</td>
</tr>
<tr>
<td>tops</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>word</th>
<th>P(word)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ship</td>
<td>0.3</td>
</tr>
<tr>
<td>pass</td>
<td>0.0</td>
</tr>
<tr>
<td>camp</td>
<td>0.0</td>
</tr>
<tr>
<td>flock</td>
<td>0.2</td>
</tr>
<tr>
<td>soccer</td>
<td>0.05</td>
</tr>
<tr>
<td>mother</td>
<td>0.05</td>
</tr>
<tr>
<td>tops</td>
<td>0.4</td>
</tr>
</tbody>
</table>
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.

\[
\vec{x} = \begin{bmatrix} 4.32957 \\ 2.48562 \\ 1.08139 \\ \vdots \\ 0.45628 \end{bmatrix}
\]
Defining CHMMs

- Continuous HMMs are very similar to discrete HMMs.
  - $S = \{s_1, ..., s_N\}$: set of states (e.g., subphones)
  - $X = \mathbb{R}^{42}$: continuous observation space

- $\Pi = \{\pi_1, ..., \pi_N\}$: 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
- $Q = \{q_0, ..., q_T\}, q_i \in S$: state sequence
- $O = \{\sigma_0, ..., \sigma_T\}, \sigma_i \in X$: observation sequence
Word-level HMMs?

• Imagine that we want to learn an HMM for each word in our lexicon (e.g., 60K words → 60K HMMs).
• No, thank you! Zipf’s law tells us that many words occur very infrequently.
  • 1 (or a few) training examples of a word is not enough to train a model as highly parameterized as a CHMM.

• In a word-level HMM, each state might be a phoneme.
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.

![Diagram of tristate phoneme model](image)
Phoneme HMMs

- We train each phoneme HMM using *all* sequences of that phoneme.
- Even from different words.

<table>
<thead>
<tr>
<th>MFCC</th>
<th>64 85 ae</th>
<th>85 96 sh</th>
<th>96 102 epi</th>
<th>102 106 m</th>
<th>...</th>
</tr>
</thead>
</table>

Time, $t$

<table>
<thead>
<tr>
<th></th>
<th>85 96</th>
<th>85 96</th>
<th>85 96</th>
<th>85 96</th>
<th>85 96</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

annotation

observations
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 a bit more complicated than this – normally phoneme HMMs have special ‘handle’ states at either end that connect to other HMMs)
Co-articulation and triphones

- **Co-articulation**: *n.* When a phoneme is influenced by adjacent phonemes.

- **A triphone HMM** captures co-articulation.
  - Triphone model /a-b+c/ is phoneme *b* when preceded by *a* and followed by *c*.

Two (of many) triphone HMMs for /t/:
- /s-t+iy/
- /iy-t+eh/
Combining triphone HMMs

- Triphone models can only connect to other triphone models that ‘match’.

/z+ih/

/z-ih+f/

/ih-f/
Concatenating phoneme models

We can easily incorporate unigram probabilities through transitions, too.

From Jurafsky & Martin text
Bigram models

From Jurafsky & Martin text
**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, $b_i(\mathbf{x})$, 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* and *words*. 
Speech recognition architecture

- Cepstral feature extraction
- MFCC features
- Gaussian Mixture models
- Phoneme likelihoods
- HMM lexicon
- Viterbi decoder
- N-gram language model

Are there alternatives? Of course, e.g.:
1. Acoustic-articulatory hybrids
2. HMM-ANN hybrids
3. End-to-end neural networks

... a real poncho
1. Audio-visual speech methods

- Observing the **vocal tract** directly, rather than through inference, can be very helpful in automatic speech recognition.

- The shape and aperture of the mouth gives some clues as to the phoneme being uttered.
  - Depending on the level of invasiveness, we can even measure the glottis and tongue directly.
1. Example of articulatory data

• TORGO was built to train augmented ASR systems.
  • 9 subjects with **cerebral palsy** (1 with ALS), 9 matched controls.
  • Each reads 500—1000 prompts over **3 hours** that cover **phonemes** and **articulatory contrasts** (e.g., *meat* vs. *beat*).
  • **Electromagnetic articulography** (and video) track points to <1 mm.
1. Example – Lip aperture and nasals

Acoustic spectrograms

/m/
/n/
/ng/

Lip apertures over time
1. Coupled HMM

Where $Q_i$ is the HMM state, $m$ is the index into a GMM, and $o_i$ is the observation at time $i$.

2. 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 $j$, $a_{ij}$

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

$$\psi_j(t) = \arg\max_i \left[ \delta_i(t - 1) a_{ij} \right]$$

Do these probabilities need to be GMMs?

$\sigma_0 = \text{ship}$

$\sigma_1 = \text{frock}$

$\sigma_2 = \text{tops}$

Observations, $\sigma_t$
2. Replacing GMMs with ANNs

• Obtain \( b_j(x) = p(x|q_j) \) with a neural network.
• We can’t learn that continuous distribution directly, but we can use Bayes’ rule:

\[
p(x|q_j) = \frac{p(q_j|x) \cdot p(x)}{p(q_j)}
\]
2. Hybrid HMM and ANN

2. Hybrid HMM and ANN

Results: As always, it depends on the data and task.

Sometimes, the hybrid approach is best. Sometimes, HMM-GMMs are best, and sometimes...

3. End-to-end neural networks

- End-to-end neural network ASR often depends on two steps:
  1. A generalization of RNNs (e.g., GRUs) to be bi-directional. This allows us to use both Forward and Backward information, as in HMMs.

3. End-to-end neural networks

- Neural networks (feedforward or RNN) are typically trained as **frame-level** classifiers.
  - 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.
- ∴ the second step for end-to-end neural network ASR is:
  2. An objective function that allows sequence transcription without requiring prior alignment between the input and target sequences. E.g., **Connectionist Temporal Classification**:
    \[ CTC(x) = - \log P(y^*|x) \]  
    for desired transcription \( y^* \).
3. End-to-end neural networks


**Table 1. Wall Street Journal Results.** All scores are word error rate/character error rate (where known) on the evaluation set. ‘LM’ is the Language model used for decoding. ‘14 Hr’ and ‘81 Hr’ refer to the amount of data used for training.

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>LM</th>
<th>14 HR</th>
<th>81 HR</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN-CTC</td>
<td>NONE</td>
<td>74.2/30.9</td>
<td>30.1/9.2</td>
</tr>
<tr>
<td>RNN-CTC</td>
<td>DICTIONARY</td>
<td>69.2/30.0</td>
<td>24.0/8.0</td>
</tr>
<tr>
<td>RNN-CTC</td>
<td>MONOGRAM</td>
<td>25.8</td>
<td>15.8</td>
</tr>
<tr>
<td>RNN-CTC</td>
<td>BIGRAM</td>
<td>15.5</td>
<td>10.4</td>
</tr>
<tr>
<td>RNN-CTC</td>
<td>TRIGRAM</td>
<td>13.5</td>
<td>8.7</td>
</tr>
<tr>
<td>RNN-WER</td>
<td>NONE</td>
<td>74.5/31.3</td>
<td>27.3/8.4</td>
</tr>
<tr>
<td>RNN-WER</td>
<td>DICTIONARY</td>
<td>69.7/31.0</td>
<td>21.9/7.3</td>
</tr>
<tr>
<td>RNN-WER</td>
<td>MONOGRAM</td>
<td>26.0</td>
<td>15.2</td>
</tr>
<tr>
<td>RNN-WER</td>
<td>BIGRAM</td>
<td>15.3</td>
<td>9.8</td>
</tr>
<tr>
<td>RNN-WER</td>
<td>TRIGRAM</td>
<td>13.5</td>
<td>8.2</td>
</tr>
<tr>
<td>BASELINE</td>
<td>NONE</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>BASELINE</td>
<td>DICTIONARY</td>
<td>56.1</td>
<td>51.1</td>
</tr>
<tr>
<td>BASELINE</td>
<td>MONOGRAM</td>
<td>23.4</td>
<td>19.9</td>
</tr>
<tr>
<td>BASELINE</td>
<td>BIGRAM</td>
<td>11.6</td>
<td>9.4</td>
</tr>
<tr>
<td>BASELINE</td>
<td>TRIGRAM</td>
<td>9.4</td>
<td>7.8</td>
</tr>
<tr>
<td>COMBINATION</td>
<td>TRIGRAM</td>
<td>—</td>
<td>6.7</td>
</tr>
</tbody>
</table>

Here, **lower** scores are **better**, because they are **error rates**.

*But how to compute those error rates?*
EVALUATING SPEECH RECOGNITION
Evaluating ASR accuracy

• How can you tell how good an ASR system at recognizing 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*’ 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 mistaken 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’
Evaluating ASR accuracy

• But how to decide which errors are of each type?
• E.g.,
  Reference:  *how to recognize speech*
  Hypothesis:  *how to wreck a nice beach*,

• It’s not so simple: ‘*speech*’ seems to be mistaken for ‘*beach*’, except the /s/ phoneme is incorporated into the preceding hypothesis word, ‘*nice*’ (/n ay s/).
  • Here, ‘*recognize*’ seems to be mistaken for ‘*wreck a nice*’
    • Are each of ‘*wreck a nice*’ substitutions of ‘*recognize*’?
    • Is ‘*wreck*’ a substitution for ‘*recognize*’?
      • If so, the words ‘*a*’ and ‘*nice*’ must be insertions.
    • Is ‘*nice*’ a substitution for ‘*recognize*’?
      • If so, the words ‘*wreck*’ and ‘*a*’ must be insertions.
Levenshtein distance

• In practice, ASR people are often more concerned with overall WER, and don’t care about how those errors are partitioned.
  • E.g., 3 substitution errors are ‘equivalent’ to 1 substitution plus 2 insertions.

• The Levenshtein distance is a straightforward algorithm based on dynamic programming that allows us to compute overall WER.
Levenshtein distance

Allocate matrix $R[n + 1, m + 1]$  
\(//\) where $n$ is the number of reference words  
\(//\) and $m$ is the number of hypothesis words

Initialize $R[0, 0] := 0$, and $R[i, j] := \infty$ for all other $i = 0$ or $j = 0$

for $i := 1..n$  
\(//\) #ReferenceWords  
for $j := 1..m$  
\(//\) #Hypothesis words  
\[ R[i, j] := \min( R[i - 1, j] + 1, \quad // \text{deletion} \]
\[ R[i - 1, j - 1], \quad // \text{if the } i^{th} \text{ reference word and} \]
\[ R[i - 1, j - 1] + 1, \quad // \text{the } j^{th} \text{ hypothesis word match} \]
\[ R[i, j - 1] + 1 ) \quad // \text{insertion} \]

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

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>to</td>
<td>$\infty$</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td>$\infty$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>$\infty$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $R[1,1] = \min(\infty + 1, (0), \infty + 1) = 0$ (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>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</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>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</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>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- And onto the second reference word
Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>how</td>
</tr>
<tr>
<td>-</td>
<td>to</td>
</tr>
<tr>
<td>-</td>
<td>wreck</td>
</tr>
<tr>
<td>-</td>
<td>a</td>
</tr>
<tr>
<td>-</td>
<td>nice</td>
</tr>
<tr>
<td>-</td>
<td>beach</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>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</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>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- And we finish the grid.
- There are $R[n, m] = 4$ 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>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</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>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- If we want, we can **backtrack** using our arrows to find the proportion of substitution, deletion, and insertion errors.
Levenshtein distance

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</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>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- Here, we estimate 2 substitution errors and 2 insertion errors.
- Arrows can be encoded within a special backtrace matrix.
## Recent performance

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Speech type</th>
<th>Lexicon size</th>
<th>ASR WER (%)</th>
<th>Human WER (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td>Spontaneous</td>
<td>10</td>
<td>0.3 %</td>
<td>0.009 %</td>
</tr>
<tr>
<td>Phone directory</td>
<td>Read</td>
<td>1000</td>
<td>3.6 %</td>
<td>0.1 %</td>
</tr>
<tr>
<td>Wall Street Journal</td>
<td>Read</td>
<td>64,000</td>
<td>3.6 %</td>
<td>1 %</td>
</tr>
<tr>
<td>Radio news</td>
<td>Mixed</td>
<td>64,000</td>
<td>13.5 %</td>
<td>-</td>
</tr>
<tr>
<td>Switchboard (telephone)</td>
<td>conversation</td>
<td>10,000</td>
<td>6.3 %</td>
<td>4 %</td>
</tr>
</tbody>
</table>