automatic speech recognition 2
This lecture

- Automatic speech recognition (ASR)
  - Applying HMMs to ASR,
  - Practical aspects of ASR, and
  - Levenshtein distance.
Consider what we want speech to do

My hands are in the air.

Buy ticket... AC490... yes

Put this there.

Can we just use GMMs?
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 should be able 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 MFCC vectors into a sequence of phonemes.

\[
\text{open(podBay.doors);} \\
\]

(* 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

Language model

Source

\( P(W) \)

Acoustic model

Channel

\( P(X|W) \)

Decoder

\( W^* = \arg\max_W P(X|W)P(W) \)

Word sequence \( W \)

Acoustic sequence \( X \)

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 $b_0(\mathbf{x})$ tells us the probability of seeing the (multivariate) continuous observation $\mathbf{x}$ while in state 0.

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

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

\[ \theta \]

\[ \Pi = \{\pi_1, \ldots, \pi_N\} \] : 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
\[ O = \{\sigma_0, \ldots, \sigma_T\}, \sigma_i \in X \] : observation sequence
**Word-level HMMs?**

- Imagine that we want to learn an HMM for each word in our lexicon (e.g., 60K words $\rightarrow$ 60K HMMs).
- No, thank you! Zipf’s law tells us to expect *many* words to 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.

\[ b_0 \quad b_1 \quad b_2 \]

\[ /oi/ \_0 \quad /oi/ \_1 \quad /oi/ \_2 \]

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

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

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

<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>

Phoneme HMMs

/iy/

/ih/

/eh/

/s/

/sh/
Combining models

• We can learn an \( N \)-gram **language model** from word-level and phoneme-level annotations 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.
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 tiny 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

Lexicon

one  w ah n
two  t uw
three th r iy
four  f ao r
five f ay v
six  s ih k s
seven s eh v ax n
eight ey t
nine n ay n
zero z iy r ow
oh  ow

Phone HMM

We can easily incorporate unigram probabilities through transitions, too.

From Jurafsky & Martin text
Bigram models
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, $b_i(\hat{x})$, are adjusted using the GMM training ‘recipe’ from last week.

• 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

\[ P(X|W) \]

\[ P(W) \]
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, sell my Apple stock.”)

• 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?
Signal-to-noise ratio

• We are often concerned with the **signal-to-noise ratio** (SNR), which measures the ratio between the power of a desired **signal** within a recording ($P_{signal}$, e.g., the human speech) and additive **noise** ($P_{noise}$).
  
• Noise typically includes:
  
  • **Background noise** (e.g., people talking, wind),
  
  • **Signal degradation**. This is *normally* ‘white’ noise produced by the medium of transmission.

\[
SNR_{db} = 10 \log_{10} \left( \frac{P_{signal}}{P_{noise}} \right)
\]

You don’t have to memorize this formula.

• High $SNR_{db}$ is >30dB. Low $SNR_{db}$ is < 10 dB.
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.
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.
Example – Lip aperture and nasals

Acoustic spectrograms

Lip apertures over time
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 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 – 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>
## 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></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>

- $R[1,1] = \min(\infty + 1, 0, *, \infty + 1) = 0$ (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>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>how</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>how</td>
<td>∞</td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
</tr>
<tr>
<td>speech</td>
<td>∞</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.
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
        $R[i, j] := \min( R[i - 1, j] + 1, R[i - 1, j - 1], R[i - 1, j - 1] + 1 )$ // insertion
            // deletion
            // if the $i^{th}$ reference word and // if they differ, i.e., substitution
            // the $j^{th}$ hypothesis word match

Return $100 \times R[n, m]/n$
State-of-the-art 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>6.6 %</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>19.3 %</td>
<td>4 %</td>
</tr>
</tbody>
</table>