Continuous Automatic Speech Recognition

Professor Marie Roch

J&M 7.5, 9.5-9.8, 10.3

Problem

• Find words $W$ that max. observations $O$

$$\hat{W} = \arg \max_{W \in \mathcal{L}} P(O \mid W) P(W)$$

• How can we find $\hat{W}$ in a reasonable manner?
Probability imbalance

- Acoustic observations assumed independent
  - Clearly false
  - Underestimate of P(O|W)
- Language model scale factor (weight)
  \[ \hat{W} = \arg \max_{w \in L} P(O|W) P^{LMSF}(W) \]
  typical \( LMSF \in [5,15] \)

Probability and sentence length

- Each time we add a word to \( W \),
  \( P(W) \) decreases
- Large vocabulary language models tend to have lower probabilities, so the penalty for adding words becomes even greater.
- We can consider this to be a penalty for inserting words.
Insertion Penalty and Recognition Bias

• Search becomes biased:
  – Larger penalty $\rightarrow$ preference for shorter sentences with longer words
  – Smaller penalty $\rightarrow$ preference for longer sentences with shorter words

Word insertion penalty

• To avoid bias towards large or small words we use a tunable word insertion penalty parameter.

$$\hat{W} = \arg\max_{w \in L} P(O|W) \cdot \frac{P^{LMSF}(W) \cdot WIP^N}{P^{LM}} \quad 0 < WIP \leq 1$$

• High penalty $\rightarrow$ preference for longer words
• Low penalty $\rightarrow$ preference for shorter words
Decoding

- Decoders are used to determine the optimal word sequence.
- Typically use Viterbi algorithm with extensions
- We need to be able to expand words to subword units.

Lexical baseforms

- Describes the transcription of a word into subword units.
- Issues
  - pronunciations due to dialects, e.g. “tomato”
  - coarticulation
    - across words, “you” /y uw/ versus “did you ...” /jh uh/
  - common contractions
CMU Pronouncing Dictionary

- Over 100,000 entries
- 39 phonemes
- Transcription examples:
  
<table>
<thead>
<tr>
<th>Word</th>
<th>Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOLPHIN</td>
<td>D AA1 L F AH0 N</td>
</tr>
<tr>
<td>TOMATO</td>
<td>T AH0 M EY1 T OW2</td>
</tr>
<tr>
<td>TOMATO(2)</td>
<td>T AH0 M AA1 T OW2</td>
</tr>
<tr>
<td>YOU'VE</td>
<td>Y UW1 V</td>
</tr>
</tbody>
</table>

So far...

- We have balanced the differences between acoustic and language model weights...
- Determined how to build words from subunits...

\[
\hat{W} = \arg \max_{W \in \mathcal{L}} P(O \mid W) \cdot P^{LMSF}(W) \cdot WIP^N \quad \text{WIP} < 1
\]

but how can we systematically search for W in real-time?
Search

• A search is a systematic exploration of a state space.
• Example: The eight puzzle.

Exploration by depth

• We avoid any state which would repeat something previously seen.
• i.e. Moving the 4 into the blank space would repeat the root of the tree.
Exploration by breadth

- Similar restrictions to avoid repeated states.

Exploring search spaces

- When deciding how to search, we have several choices to make:
  - breadth vs. depth search
  - blind vs. heuristic search
  - optimal path vs. first/any path
Search algorithms

- By modifying the generic search algorithm, we can implement different types of search
  - depth first
  - breadth first
  - best first (A* if heuristic is guaranteed to not overestimate cost to goal)
  - beam search

Pruning

- Some partial paths not worth pursuing.
- Elimination of these paths called pruning.
- Likely to result in an inadmissible search.
Beam search – Example of pruning

- Breadth-first search with pruning.
- Results in an inadmissible search.
- Modifications required to generic search

Building words

words frequently constructed "on the fly"
Forward decode
pretending each state is a phoneme

\[ \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(x_i) \]

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>0</td>
<td>0</td>
<td>0.008</td>
<td>0.0093</td>
<td>0.0114</td>
<td>0.00703</td>
<td>0.00345</td>
<td>0.00306</td>
<td>0.00206</td>
<td>0.00117</td>
</tr>
<tr>
<td>AV</td>
<td>0</td>
<td>0.04</td>
<td>0.034</td>
<td>0.0654</td>
<td>0.0055</td>
<td>0.016</td>
<td>0.00576</td>
<td>0.000806</td>
<td>0.000208</td>
<td>0.000032</td>
</tr>
<tr>
<td>F</td>
<td>0.8</td>
<td>0.32</td>
<td>0.112</td>
<td>0.0224</td>
<td>0.00448</td>
<td>0.000896</td>
<td>0.000179</td>
<td>4.48e-05</td>
<td>1.12e-05</td>
<td>2.8e-06</td>
</tr>
</tbody>
</table>

\[ V_t(j) = \max_{i \in \mathcal{I}_S}(V_{t-1}(i) a_{ij} b_j(x_i)) \]

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>0</td>
<td>0</td>
<td>0.008</td>
<td>0.0093</td>
<td>0.0114</td>
<td>0.00703</td>
<td>0.00345</td>
<td>0.00306</td>
<td>0.00206</td>
<td>0.00117</td>
</tr>
<tr>
<td>AV</td>
<td>0</td>
<td>0.04</td>
<td>0.034</td>
<td>0.0654</td>
<td>0.0055</td>
<td>0.016</td>
<td>0.00576</td>
<td>0.000806</td>
<td>0.000208</td>
<td>0.000032</td>
</tr>
<tr>
<td>F</td>
<td>0.8</td>
<td>0.32</td>
<td>0.112</td>
<td>0.0224</td>
<td>0.00448</td>
<td>0.000896</td>
<td>0.000179</td>
<td>4.48e-05</td>
<td>1.12e-05</td>
<td>2.8e-06</td>
</tr>
</tbody>
</table>

Viterbi decode
pretending each state is a phoneme

\[ \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(x_i) \]

\[ V_t(j) = \max_{i \in \mathcal{I}_S}(V_{t-1}(i) a_{ij} b_j(x_i)) \]
Transitioning between words
bigram model

Decoders must be modified to track word history

Bigram model in a trellis
Forward or Viterbi algorithm

The Viterbi algorithm typically used

- **Pros**
  - No scaling for log implementation
  - No need to manage alternate word hypotheses

- **Cons**
  - Doesn’t yield all-path (full) probability

Regardless of choice, modifications to store word sequence necessary.
Time synchronous Viterbi beam search

- Real-time decoding is computationally expensive.
- Search space is pruned at each step
  - Take highest likelihood path
  - Keep any path with 90-95% of probability (beam size 5-10%), e.g.:

\[ I_{\text{keep}}(w^t_j, q^t_j) = n \]

\[ V(X_1 \ldots X_t \mid w^t_j, q^t_j) \geq \log(0.95) + \log V(X_1 \ldots X_t \mid w^t_j, q^t_j) = m \]

where \( w^t_j, q^t_j = m \) is the best hypothesis and \( w^t_j, q^t_j = n \) ranges across all hypotheses

Forced (Viterbi) alignment

- Special case of Viterbi decode
- Construct HMM for a single hypothesis \( w^N_i \)
- Decode

- Can be used to find boundaries of words and segments
Embedded training

• How do we train the large vocabulary speech recognizers?
• Hand labeled phonemes?
  – Typical annotation time 400:1 →
  – Humans performance on subphone boundaries is poor

Embedded training solution
(flat or dead start training)

• Label at word level
• Use lexical baseforms to expand to phoneme stream
• Flat start: Initialize single Gaussian models to global mean and variance, all other parameters except π to uniform distribution
Embedded/Flat start training

- Perform forced alignment
  - Sample forced alignment from Young et al. (2006)

- Use segmented data to retrain models
  - e.g. collect all /eh/ data and retrain /eh/

- Repeat until convergence occurs

Embedded training

- Partition / reestimate
- Repeat the process
Subsequent iterations...

Embedded training

• Why does it work?
• Suppose we want to train /ə/ as in /əbaut/ (about)
• Consider context of other words with /ə/ in them:
  /əbatəd/ /byuːtəfəl/ /mɑrdʒənəli:/
  These have many different contexts
• /ə/ is more likely to appear than other phones.
Evaluation

- Problem: Different types of errors
  - insertion: “It was” \(\rightarrow\) “It’s was”
  - substitution:
    “the portable phone” \(\rightarrow\) “the portable form”
  - deletion: “New York, the empire state” \(\rightarrow\)
    “New York, empire state”

Error rate

Word Error Rate = \(100 \times \frac{\text{insertions + substitutions + deletions}}{\text{#words in transcript}}\)

Sentence Error Rate = \(100 \times \frac{\text{sentences with } \geq 1 \text{ error}}{\text{#sentences in transcript}}\)
How are we doing?
NIST Speech benchmarks

Advanced topics
(time permitting)
Multipass decoders
(J&M 10.1)

- Viterbi decode $\rightarrow$
  best state sequence, not best *word* sequence.

- Solutions:
  - Multi-pass decoding
  - Stack decoders

N-best decoding

- Return N most likely candidates

<table>
<thead>
<tr>
<th>Rank</th>
<th>Path</th>
<th>AM logprob</th>
<th>LM logprob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>it’s an area that’s naturally sort of mysterious</td>
<td>-7193.53</td>
<td>-20.25</td>
</tr>
<tr>
<td>2.</td>
<td>that’s an area that’s naturally sort of mysterious</td>
<td>-7192.28</td>
<td>-21.11</td>
</tr>
<tr>
<td>3.</td>
<td>it’s an area that’s not really sort of mysterious</td>
<td>-7221.68</td>
<td>-18.91</td>
</tr>
<tr>
<td>4.</td>
<td>that scenario that’s naturally sort of mysterious</td>
<td>-7189.19</td>
<td>-22.08</td>
</tr>
<tr>
<td>5.</td>
<td>there’s an area that’s naturally sort of mysterious</td>
<td>-7198.35</td>
<td>-21.34</td>
</tr>
<tr>
<td>6.</td>
<td>that’s an area that’s not really sort of mysterious</td>
<td>-7220.44</td>
<td>-19.77</td>
</tr>
<tr>
<td>7.</td>
<td>the scenario that’s naturally sort of mysterious</td>
<td>-7205.42</td>
<td>-21.50</td>
</tr>
<tr>
<td>8.</td>
<td>so it’s an area that’s naturally sort of mysterious</td>
<td>-7195.92</td>
<td>-21.71</td>
</tr>
<tr>
<td>9.</td>
<td>that scenario that’s not really sort of mysterious</td>
<td>-7217.34</td>
<td>-20.70</td>
</tr>
<tr>
<td>10.</td>
<td>there’s an area that’s not really sort of mysterious</td>
<td>-7226.51</td>
<td>-20.01</td>
</tr>
</tbody>
</table>
N-best decoders

- Maintain N-best word hypothesis history

Arcs give information about:
- acoustic models & scores
- language model scores
- start, end times, etc.
Second pass

- Rescore N-best hypotheses with better sources, e.g.
  - language model
  - acoustic model
- Lattice error rate
  - Defines error as not being in top N
  - Why is this important?

Lattice density

- Many similar hypotheses
- Common to collapse these during rescoring
Word Graph/
Finite State Machine

- Lattice can be collapsed to a graph
  - Discard timing information
  - Retain N-gram probabilities

- Graph used as a constrained language model and rescored

Cleaning up the output

- Disfluency detection and repair
  - filler: It’s like you know uhm a really cool thing
  - fragments & restarts: I li- no I love to dive
  - repetition: We took the Chevy down to, down to the levee.

- Both high and low level features are used:
  - neighboring words, part of speech tags
  - pause time, F0 jitter, spectral tilt, glottal cycle analysis
Other types of post-processing

• Diarization
• Sentence segmentation and punctuation
• Duration

Scaling

• Remember: forward/backward algorithm likelihoods quickly underflow.
• Scaling coefficients keep likelihood within machine precision.

$$S_t = \frac{1}{\sum_{j=1}^{N} \alpha'_t(j)} \rightarrow \sum_{j=1}^{N} S_t \alpha'_t(j) = 1$$
Scaling the forward & backward alg.

- We can multiply each $\alpha_t$ and $\beta_t$ by the scale factor $S_t$ and use the scaled $\alpha_{t-1}$ and $\beta_{t+1}:

$$\hat{\alpha}_t(j) = \sum_{i=1}^{N} \alpha'_t(i) a_i b_j(X_i)$$

$$S_t = \frac{1}{\sum_{i=1}^{N} \hat{\alpha}_t(i)}$$

$$\alpha'_t(j) = S_t \hat{\alpha}_t(j)$$

- Prevents hardware underflow.

Scaling

- At each step, we scale.

$$\alpha'_t(j) = S_t \alpha_t(j) \quad \text{by definition}$$

$$\alpha'_2(j) = S_2 \sum_{i=1}^{N} \alpha'_t(i) a_i b_j(x_2)$$

$$\alpha'_2(j) = S_2 \sum_{i=1}^{N} S_t \alpha_t(i) a_i b_j(x_2)$$

$$\alpha'_2(i) = S_1 S_2 \sum_{i=1}^{N} \alpha_t(i) a_i b_j(x_2) \quad S_2 \text{ constant across } i$$

- Leads to:

$$\alpha'_t(j) = S_1 S_2 \ldots S_t \sum_{i=1}^{N} \alpha_t(i) a_i b_j(x_i)$$

$$= (\prod_{t=1}^{T} S_t) \alpha_t(j)$$

$$= \text{Scale}_{\alpha_t}(t) \alpha_t(j) \text{ where Scale}_{\alpha_t}(t) = (\prod_{t=1}^{T} S_t)$$
Recovering probability from scaling

\[ \sum_{i=1}^{N} \alpha_x(i) = \sum_{i=1}^{N} \text{Scale}_\alpha(T) \alpha_x(i) \]  from previous slide

\[ = \text{Scale}_\alpha(T) \sum_{i=1}^{N} \alpha_x(i) \]  \( \text{Scale}_\alpha(T) \) independent of \( i \)

\[ = \text{Scale}_\alpha(T) \mathbb{P}(X | \Phi) \]  \( \mathbb{P}(X | \Phi) = \sum_{i=1}^{N} \alpha_x(i) \)

\[ \mathbb{P}(X | \Phi) = \frac{\sum_{i=1}^{N} \alpha_x(i)}{\text{Scale}_\alpha(T)} = \frac{1}{\text{Scale}_\alpha(T)} \]  would still result in underflow

\[ \log \mathbb{P}(X | \Phi) = -\sum_{i=1}^{T} \log S_i \]