1 More about ASR

- Introduction
- Dynamic programming
- Viterbi algorithm
- Other approaches to decoding
- Training acoustic models
- FLOSS resources
- Assignment
- Bibliography
Two technologies are needed to make the HMM framework practical

- Decoder technology to find the
  \[
  \arg\max_{Words} P(Observation|Words) \cdot P(Words)
  \]

- Determining the stochastic parameters of the HMM state automaton, i.e. training

Many pictures (and their copyrights) are from [Jurafsky and Martin(2000)]
Dynamic programming

Look for best alignment: Minimum edit distance

- Delete
- Insert
- Substitute
Dynamic programming

function MIN-EDIT-DISTANCE(target, source) returns min-distance

\[ n \leftarrow \text{LENGTH}(\text{target}) \]
\[ m \leftarrow \text{LENGTH}(\text{source}) \]

Create a distance matrix \( \text{distance}[n+1,m+1] \)
\[ \text{distance}[0,0] \leftarrow 0 \]

for each column \( i \) from 0 to \( n \) do
  for each row \( j \) from 0 to \( m \) do
    \[ \text{distance}[i, j] \leftarrow \min( \text{distance}[i-1,j] + \text{ins-cost}(\text{target}_i), \]
    \[ \text{distance}[i-1,j-1] + \text{subst-cost}(\text{source}_j, \text{target}_i), \]
    \[ \text{distance}[i,j-1] + \text{del-cost}(\text{source}_j) ) \]

Fill a matrix with cumulative edit distances, \( \text{distance}[i, j] = \min \) of
- \( \text{distance}[i - 1, j] + \text{insert-cost}(\text{target}_i) \)
- \( \text{distance}[i - 1, j - 1] + \text{substitution-cost}(\text{source}_j, \text{target}_i) \)
- \( \text{distance}[i, j - 1] + \text{deletion-cost}(\text{source}_j) \)
Dynamic programming

<table>
<thead>
<tr>
<th>n</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>i</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>t</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>n</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>#</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>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Trace back the choices of the minimal distance (bold numbers)

- This finds the globally minimal cost path
- Full search unwieldy for large and complex matrices
- In general, searches are pruned to exclude paths that deviate far from the diagonal: Beam search
Viterbi algorithm

Simplified pronunciation networks  [Jurafsky and Martin(2000)]

- Each word is modeled as a Finite State Machine
- Individual phoneme HMMs are trained from a corpus that does not contain all the words
- A pronunciation dictionary contains all word models
- Transition probabilities are ”trained” from a transcribed speech corpus
Viterbi algorithm

Viterbi algorithm result “for I need a” [Jurafsky and Martin(2000)]

- Whole sequence on X axis
- All word models on the other axis
- Switch to (any) new word after reaching the end of the current word
- Word switching cost based on the language model
### Viterbi algorithm

<table>
<thead>
<tr>
<th>I need</th>
<th>0.0016</th>
<th>need need</th>
<th>0.000047</th>
<th># Need</th>
<th>0.000018</th>
</tr>
</thead>
<tbody>
<tr>
<td>I the</td>
<td>0.00018</td>
<td>need the</td>
<td>0.012</td>
<td># The</td>
<td>0.016</td>
</tr>
<tr>
<td>I on</td>
<td>0.000047</td>
<td>need on</td>
<td>0.000047</td>
<td># On</td>
<td>0.00077</td>
</tr>
<tr>
<td>I I</td>
<td>0.039</td>
<td>need I</td>
<td>0.000016</td>
<td># I</td>
<td>0.079</td>
</tr>
<tr>
<td>the need</td>
<td>0.00051</td>
<td>on need</td>
<td>0.000055</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the the</td>
<td>0.0099</td>
<td>on the</td>
<td>0.094</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the on</td>
<td>0.00022</td>
<td>on on</td>
<td>0.0031</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the I</td>
<td>0.00051</td>
<td>on I</td>
<td>0.00085</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Bigram probabilities [Jurafsky and Martin(2000)]

- Word switching in Viterbi searches uses probabilities
- Switch to a new word with bigram probability cost
- Does not work with trigram probabilities
Viterbi algorithm

Single pronunciation automaton for *I, need, on, and the*

[Jurafsky and Martin(2000)]

- Bigram probabilities connect the word models
- Merge **start** and **end** states of connected words
- Need for *pruning* is apparent
Viterbi algorithm

```plaintext
function VITERBI(observations of len $T$, state-graph) returns best-path

num-states ← NUM-OF-STATES(state-graph)
Create a path probability matrix $viterbi[num-states+2,T+2]$
$viterbi[0,0] ← 1.0$
for each time step $t$ from 0 to $T$ do
  for each state $s$ from 0 to num-states do
    for each transition $s'$ from $s$ specified by state-graph
      new-score ← $viterbi[s, t] * a[s,s'] * b_{s'}(o_t)$
      if ((viterbi[$s'$, $t+1$] = 0) || (new-score > viterbi[$s'$, $t+1$]))
        then
          $viterbi[s', t+1] ←$ new-score
          back-pointer[$s', t+1] ← $s$
  Backtrace from highest probability state in the final column of $viterbi[]$ and return path
```

Extended version of the edit distance [Jurafsky and Martin(2000)]

- $a[s, s'] = P(s \rightarrow s')$
- $b_{s'}(o_t) = P(o_t|s')$
Viterbi algorithm

Individual state columns in Viterbi algorithm [Jurafsky and Martin(2000)]

- The actual entries for the Automaton
- Note the problems for a 20,000 word dictionary
Viterbi algorithm: Subphones revisited [Jurafsky and Martin(2000)]

Use structured, context sensitive phone units

- Single phone units perform bad due to coarticulation
- \textit{Begin} differs from \textit{End} (eg, /d/)
- 60 context dependent triphones $\Rightarrow 60^3 = 216000$ models
- Cluster contexts, eg, on manner and place of articulation
Other approaches to decoding: Introduction

The standard HMM model has limitations

- Viterbi decoder penalizes multiple pronunciations
- Viterbi decoder does not work for anything more complex than bigram
- It is not possible to include other linguistic knowledge
  - Phoneme duration (HMM have a Poison distribution)
  - Intonation
  - Semantics
  - Speaker identification
  - Expressive speech tags
  - Task related knowledge
Two stage N-best decoding [Jurafsky and Martin(2000)]

- Keep N-best utterance list or word lattice
- Rescore the probabilities with the extra knowledge
  - A trigram or higher grammar
  - Phoneme duration probability
  - Parallel Intonation and Accent detector (HMM)
  - Include semantic or task related knowledge
  - Multiple speakers and expressive speech tags
- Look up best path through rescored word lattice
Other approaches to decoding: $A^*$

Stack, or $A^*$, decoding [Jurafsky and Martin(2000)]

- Viterbi uses best path upto position $t$ to get to $t + 1$
- $A^*$ uses complete forward algorithm (exact likelihoods)
- $A^*$ searches potential utterances best-first
Other approaches to decoding: \( A^\star \)

```function STACK-DECODING() returns min-distance

Initialize the priority queue with a null sentence.
Pop the best (highest score) sentence \( s \) off the queue.
If \( s \) is marked end-of-sentence (EOS) output \( s \) and terminate.
Get list of candidate next words by doing fast matches.
For each candidate next word \( w \):
    Create a new candidate sentence \( s + w \).
    Use forward algorithm to compute acoustic likelihood \( L \) of \( s + w \)
    Compute language model probability \( P \) of extended sentence \( s + w \)
    Compute “score” for \( s + w \) (a function of \( L \), \( P \), and ???)
    if (end-of-sentence) set EOS flag for \( s + w \).
    Insert \( s + w \) into the queue together with its score and EOS flag
```

Stack decoding [Jurafsky and Martin(2000)]

- At each point, the \( A^\star \) looks for the most likely next word
- Acoustic likelihood is part of the criterium
- Use the forward probability of preceding words
Other approaches to decoding: \( A^* \)

If *music* be the food of love \([\text{Jurafsky and Martin} (2000)]\)

- “*Start Alice*” has highest score: 40
- “*Start if*” has highest score: 30
- “*Start if music*” has highest score: 32
More about ASR

Other approaches to decoding: $A^*$

Remarks

- Use fast match heuristics for selecting next words
- Longer utterances have lower probabilities, score should correct for this
- $A^*$ evaluation function: $f^*(p) = g(p) + h^*(p)$
- $g(\text{partial path}) = P(O|\text{Words}) \cdot P(\text{Words})$, i.e. the likelihood until now
- $h^*(p)$ something that correlates with number of words in the rest of the utterance
- Defining a good $h^*(p)$ is an interesting (unsolved) problem
Other approaches to decoding: $A^*$ fast match

A tree structured lexicon from SPHINX [Gouvêa()] [Jurafsky and Martin(2000)]
- Need to get forward probabilities of potential continuations fast
- Tree lexicon shares forward probabilities between words
- Allows early pruning of search trees
Training acoustic models: Introduction

Determine $P(Observation|Words)$, i.e. the transition probability between phone states $a_{ij}$ and the acoustic likelihood of the speech vectors $b_j(o_k)$

- Large, “transcribed” speech corpus (on text level)
- Coverage of speakers and language types
- Recorded under the same conditions as intended use, eg, over the phone or in a driving car
- Use the same microphone etc.
- Using a simulated task (Wizard of Oz or Green curtain) to elicit the same kind of speech
Training acoustic models

If all states were known \[\text{[Jurafsky and Martin(2000)]}\]

- \(a_{ij} = \frac{\#S_{ij}}{\#S_i^*}\) (count transitions and states)
- \(b_i(O_k) = \frac{\#(O_k \& S_i)}{\#S_i}\) (for discrete \(O_k\))
If observations are continuous vectors \([\text{SPH()}]\)

- \(b_i(O_t) \Rightarrow N\{\hat{\mu}_i, \hat{\Sigma}_i\}\)
- \(\hat{\mu}_i = \frac{1}{T_i} \sum_{t=1}^{T_i} O_t\)
- \(\hat{\Sigma}_i = \frac{1}{T_i} \sum_{t=1}^{T_i} [(O_t - \hat{\mu}_i)'(O_t - \hat{\mu}_i)]\)
Training acoustic models

States have to be estimated. Use an iterative procedure [Jurafsky and Martin(2000)]

- Run the recognizer on the corpus with the known words
- Calculate $\hat{a}_{ij} = \frac{\text{expected } \# S_i \rightarrow S_j}{\text{expected } \# S_i \rightarrow S_{*}}$
- Calculate $\hat{b}_j(v_k) = \frac{\text{expected } \# S_j \text{ observing } v_k}{\text{expected } \# S_j}$
- Update all values and start again
FLOSS resources

Free and Open Source ASR systems

- SPHINX (CMU) [Gouvêa(2005)] [Singh(2005)]
- CMU Statistical Language Modeling Toolkit [Rosenfeld()]
- CMU Pronouncing Dictionary [Lenzo()]
- Internet-Accessible Speech Recognition Technology project (ISIP, Mississippi State University) [ISIP(2004)]
- Open Mind Speech [Valin()]
Assignment: Week 8 Statistical Language Models

Construct your own language model

- Download texts from the internet, eg, [Project Gutenberg(2005)]
- Use a single author or a single genre
- Use --help to see instructions of the programs
- Construct unigram and bigram word tables with \texttt{Ngramcount.pl}
  \url{http://www.fon.hum.uva.nl/rob/Courses/Taaltechnologien/Ngramcount.pl}
- \texttt{perl Ngramcount.pl 1 \langle\text{filename1}\rangle \langle\text{filename2}\rangle \ldots > \text{unigramtable.txt}}
- \texttt{perl Ngramcount.pl 2 \langle\text{filename1}\rangle \langle\text{filename2}\rangle \ldots > \text{bigramtable.txt}}
- Inspect the table files. What are the most frequent words and bigrams?
- Calculate the probabilities of sentences with \texttt{ngramprobability.pl}
  \url{http://www.fon.hum.uva.nl/rob/Courses/Taaltechnologien/ngramprobability.pl}
- \texttt{perl ngramprobability.pl --count 5 --verbose bigramtable.txt "\langle\text{sentence}\rangle"}
- Enter some sentences and inspect the resulting probabilities
- Experiment with the --count option. Try --count -1 on a sentence that contains unknown word combinations
Further Reading I

P. Boersma.
Praat, a system for doing phonetics by computer.

P. Boersma and D. Weenink.
Praat 4.2: doing phonetics by computer.

CSLU.
CSLU Toolkit.
Web.

FSF.
GNU General Public License.

Joshua T. Goodman.
A bit of progress in language modeling.
URL is extended preprint.
Further Reading II

E. Gouvêa.
The CMU Sphinx Group Open Source Speech Recognition Engines.
Web.

ISIP.
The Mississippi State ISIP public domain speech recognizer.

Daniel Jurafsky and James H. Martin.
*Speech and Language Processing*.
Updates at http://www.cs.colorado.edu/

Kevin Lenzo.
The CMU Pronouncing Dictionary.
Web.

Project Gutenberg.
Project gutenberg free ebook library.
Web, 2005.
URL http://www.gutenberg.org/.
Further Reading III

Roni Rosenfeld.
The CMU Statistical Language Modeling (SLM) Toolkit.
Web.

Rita Singh.
Robust group's open source tutorial learning to use the cmu sphinx automatic speech recognition system.
Web, 2005.

Manual for the Sphinx-III recognition system.
SPHINX-CMU.
URL http://fife.speech.cs.cmu.edu/sphinxman/.

Paul A. Taylor, S. King, S. D. Isard, and H. Wright.
Intonation and dialogue context as constraints for speech recognition.

Jean-Marc Valin.
Open mind speech.
Web.

Xue Wang.
incorporating knowledge on segmental duration in hmm-based continuous speech recognition.
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.