Speech and Language Processing

Chapter 10 of SLP
Advanced Automatic Speech Recognition (I)
Search and Triphone Acoustic Modeling
Outline

- **Advanced in Search**
  - Multipass search
    - N-best lists
    - Lattices

- **Advances in Context**
  - Context-dependent (triphone) acoustic modeling

- **Metadata**
  - Disfluencies, etc
Multi-pass Search

Speech Input → Simple Knowledge Source → N-Best Decoder

N-Best List:
- Alice was beginning to get...
- Every happy family
- In a hole in the ground...
- If music be the food of love...
- If music be the foot of dove...

Smarter Knowledge Source → Rescoring

1-Best Utterance:
- If music be the food of love
Ways to represent multiple hypotheses

- **N-best list**
  - Instead of single best sentence (word string), return ordered list of N sentence hypotheses

- **Word lattice**
  - Compact representation of word hypotheses and their times and scores

- **Word graph**
  - FSA representation of lattice in which times are represented by topology
A sample N-best list
(From the CU-HTK BN system, thanks to Phil Woodland)

<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>
Computing N-best lists

- In the worst case, an admissible algorithm for finding the N most likely hypotheses is exponential in the length of the utterance.

- For example, if AM and LM score were nearly identical for all word sequences, we must consider all permutations of word sequences for whole sentence (all with the same scores).

- But of course if this is true, can’t do ASR at all!
Computing N-best lists

- Instead, various non-admissible algorithms:
  - (Viterbi) Exact N-best
  - (Viterbi) Word Dependent N-best
A* N-best

- A* (stack-decoding) is best-first search
- So we can just keep generating results until it finds N complete paths
- This is the N-best list
- But is inefficient
Exact N-best for time-synchronous Viterbi

- Due to Schwartz and Chow; also called “sentence-dependent N-best”
- Idea: maintain separate records for paths with distinct histories
  - History: whole word sequence up to current time $t$ and word $w$
  - When 2 or more paths come to the same state at the same time, merge paths w/same history and sum their probabilities.
  - Otherwise, retain only N-best paths for each state
Exact N-best for time-synchronous Viterbi

- **Efficiency:**
  - Typical HMM state has 2 or 3 predecessor states within word HMM
  - So for each time frame and state, need to compare/merge 2 or 3 sets of N paths into N new paths.
  - At end of search, N paths in final state of trellis reordered to get N-best word sequence
  - Complex is $O(N)$; this is too slow for practical systems
Word Lattice

Each arc annotated with AM and LM logprobs
Word Graph

Timing information removed
Overlapping copies of words merged
AM information removed
Result is a WFST
Natural extension to N-gram language model
Converting word lattice to word graph

- Word lattice can have range of possible end frames for word
- Create an edge from \((w_i, t_i)\) to \((w_j, t_j)\) if \(t_{j-1}\) is one of the end-times of \(w_i\)
Some researchers are careful to distinguish between word graphs and word lattices.

But we’ll follow convention in using “lattice” to mean both word graphs and word lattices.

Two facts about lattices:

- **Density**: the number of word hypotheses or word arcs per uttered word.
- **Lattice error rate** (also called “lower bound error rate”): the lowest word error rate for any word sequence in lattice.
  - Lattice error rate is the “oracle” error rate, the best possible error rate you could get from rescoring the lattice.
  - We can use this as an upper bound.
Posterior lattices

- We don’t actually compute posteriors:

\[
\hat{W} = \arg\max_{W \in \mathcal{L}} \frac{P(O|W)P(W)}{P(O)} = \arg\max_{W \in \mathcal{L}} P(O|W)P(W)
\]

- Why do we want posteriors?
  - Without a posterior, we can choose best hypothesis, but we can’t know how good it is!
  - In order to compute posterior, need to
    - Normalize over all different word hypothesis at a time
  - Align all the hypotheses, sum over all paths
Mesh = Confusion Network = Sausage = pinched lattice
One-pass vs. multipass

- Potential problems with multipass
  - Can’t use for real-time (need end of sentence)
    - (But can keep successive passes really fast)
  - Each pass can introduce inadmissible pruning
    - (But one-pass does the same w/beam pruning and fastmatch)

- Why multipass
  - Very expensive KSs. (NL parsing, higher-order n-gram, etc)
  - Spoken language understanding: N-best perfect interface
  - Research: N-best list very powerful offline tools for algorithm development
  - N-best lists needed for discriminant training (MMIE, MCE) to get rival hypotheses
A* Decoding

- Intuition:
  - If we had good heuristics for guiding decoding
  - We could do depth-first (best-first) search and not waste all our time on computing all those paths at every time step as Viterbi does.
- A* decoding, also called stack decoding, is an attempt to do that.
- A* also does not make the Viterbi assumption
- Uses the actual forward probability, rather than the Viterbi approximation
Reminder: A* search

- A search algorithm is “admissible” if it can guarantee to find an optimal solution if one exists.
- Heuristic search functions rank nodes in search space by $f(N)$, the goodness of each node $N$ in a search tree, computed as:
  \[ f(N) = g(N) + h(N) \]
  where
  - $g(N)$ = The distance of the partial path already traveled from root $S$ to node $N$
  - $h(N)$ = Heuristic estimate of the remaining distance from node $N$ to goal node $G$. 

7/30/08  Speech and Language Processing  Jurafsky and Martin 19
Reminder: A* search

- If the heuristic function \( h(N) \) of estimating the remaining distance from \( N \) to goal node \( G \) is an underestimate of the true distance, best-first search is admissible, and is called A* search.
A* search for speech

- The search space is the set of possible sentences
- The forward algorithm can tell us the cost of the current path so far $g(.)$
- We need an estimate of the cost from the current node to the end $h(.)$
A* Decoding (2)

START

- intention
- to
- my
- not
- believe
- lives
- to
- my
- not
- believe
- lives
- to
- my
- not
- believe
- lives

- bequeath
- want
- can't
- believe
- lives

- do
- not
- believe
- lives

- of
- underwriter
- typically
- believe
- lives

- are
- underwriter
- typically
- believe
- lives

- dogs
- exceptional
- believe
- lives

...
Stack decoding (A*) algorithm

function STACK-DECODING() returns min-distance

Initialize the priority queue with a null sentence.
Loop until queue is empty
    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 etc.).
    If (end-of-sentence) set EOS flag for s + w.
    Insert s + w into the queue together with its score and EOS flag.
A* Decoding (2)

\[ P(\text{"If"} \mid \text{START}) \]

\[ P(\text{"Every"} \mid \text{START}) \]

- If: 30
- Alice: 40
- In: 4
- Every: 25

[START] 1
A* Decoding (cont.)

\[ P(O \mid "if") = \text{forward probability} \]

\[ P("if" \mid \text{START}) \]

(a)

(b)

\[ P(O \mid "if") = \text{forward probability} \]

\[ P("music" \mid "if") \]

\[ P("if" \mid \text{START}) \]
Making A* work: \( h(.) \)

- If \( h(.) \) is zero, breadth first search
- Stupid estimates of \( h(.) \):
  - Amount of time left in utterance
- Slightly smarter:
  - Estimate expected cost-per-frame for remaining path
  - Multiply that by remaining time
  - This can be computed from the training set (how much was the average acoustic cost for a frame in the training set)
- Later: multi-pass decoding, can use backwards algorithm to estimate \( h^* \) for any hypothesis!
A*: When to extend new words

- Stack decoding is asynchronous
- Need to detect when a phone/word ends, so search can extend to next phone/word
- If we had a cost measure: how well input matches HMM state sequence so far
- We could look for this cost measure slowly going down, and then sharply going up as we start to see the start of the next word.
- Can’t use forward algorithm because can’t compare hypotheses of different lengths
- Can do various length normalizations to get a normalized cost
Efficiency: don’t want to expand to every single next word to see if it’s good.

Need a quick heuristic for deciding which sets of words are good expansions

“Fast match” is the name for this class of heuristics.

Can do some simple approximation to words whose initial phones seem to match the upcoming input
Forward-Backward Search

- Useful to know how well a given partial path will do in rest of the speech.
- But can’t do this in one-pass search
- Two-pass strategy: Forward-Backward Search
Forward-Backward Search

- First perform a forward search, computing partial forward scores $\alpha$ for each state
- Then do second pass search backwards
  - From last frame of speech back to first
- Using $\alpha$ as
  - Heuristic estimate for $h^*$ function for A* search
  - or Fast match score for remaining path
- Details:
  - Forward pass must be fast: Viterbi with simplified AM and LM
  - Backward pass can be A* or Viterbi
Forward-Backward Search

- **Forward pass: At each time t**
  - Record score of final state of each word ending.
  - Set of words whose final states are active (surviving in beam) at time t is $\Delta_t$.
  - Score of final state of each word $w$ in $\Delta_t$ is $\alpha_t(w)$
    - Sum of cost of matching utterance up to time t given most likely word sequence ending in word $w$ and cost of LM score for that word sequence
    - At end of forward search, best cost is $\alpha^T$.

- **Backward pass**
  - Run in reverse (backward) considering last frame $T$ as beginning one
    - Both AM and LM need to be reversed
    - Usually A* search
Forward-Backward Search: Backward pass, at each time $t$

- Best path removed from stack
- List of possible one-word extensions generated
- Suppose best path at time $t$ is $p_{hw_j}$, where $w_j$ is first word of this partial path (last word expanded in backward search)
- Current score of path $p_{hw_j}$ is $\beta_t(p_{hw})$
- We want to extend to next word $w_i$
- Two questions:
  - Find $h^*$ heuristic for estimating future input stream
    - $\alpha_t(w_i)!!$ So new score for word is $\alpha_t(w)+\beta_t(p_{hw})$
  - Find best crossing time $t$ between $w_i$ and $w_j$. 
    - $t^* = \text{argmin}_t[\alpha_t(w)+\beta_t(p_{hw})]$
Summary

- Search
  - Defining the goal for ASR decoding
  - Speeding things up: Viterbi beam decoding
  - Problems with Viterbi decoding
  - Multipass decoding
    - N-best lists
    - Lattices
    - Word graphs
    - Meshes/confusion networks
  - A* search
Context-Dependent Acoustic Modeling
Modeling phonetic context: different “eh”s

- w eh d y eh l b eh n

![Spectrogram showing different contexts for the "eh" sound]
Modeling phonetic context

- The strongest factor affecting phonetic variability is the neighboring phone.
- How to model that in HMMs?
- Idea: have phone models which are specific to context.
- Instead of Context-Independent (CI) phones
- We’ll have Context-Dependent (CD) phones
CD phones: triphones

- Triphones
- Each triphone captures facts about preceding and following phone
- Monophone:
  - p, t, k
- Triphone:
  - iy-p+aa
  - a-b+c means “phone b, preceding by phone a, followed by phone c”
“Need” with triphone models
Word-Boundary Modeling

- **Word-Internal Context-Dependent Models**
  ‘OUR LIST’:
  
  SIL AA+R AA-R L+IH L-IH+S IH-S+T S-T

- **Cross-Word Context-Dependent Models**
  ‘OUR LIST’:
  
  SIL- AA+R AA-R+L R-L+IH L-IH+S IH-S+T S-T+SIL

- Dealing with cross-words makes decoding harder! We will return to this.
Implications of Cross-Word Triphones

- Possible triphones: 50x50x50 = 125,000
- How many triphone types actually occur?
- 20K word WSJ Task, numbers from Young et al
- Cross-word models: need 55,000 triphones
- But in training data only 18,500 triphones occur!
- Need to generalize models.
Modeling phonetic context: some contexts look similar

W iy         r iy          m iy          n iy

7/30/08
Solution: State Tying

- Young, Odell, Woodland 1994
- Decision-Tree based clustering of triphone states
- States which are clustered together will share their Gaussians
- We call this “state tying”, since these states are “tied together” to the same Gaussian.
- Previous work: generalized triphones
  - Model-based clustering (‘model’ = ‘phone’)
  - Clustering at state is more fine-grained
Young et al state tying... etc.

... etc.
State tying/clustering

- How do we decide which triphones to cluster together?
- Use **phonetic features** (or ‘broad phonetic classes’)
  - Stop
  - Nasal
  - Fricative
  - Sibilant
  - Vowel
  - lateral
Decision tree for clustering triphones for tying

Phone /ih/ beg. state

Left nasal?

Yes

Right liquid?

Yes

Cluster A: n-ih+l, ng-ih+l, m-ih+l

No

Left fricative?

Yes

Cluster B: n-ih+r, ng-ih+r, m-ih+r, n-ih+w, ...

No
### Decision tree for clustering triphones for tying

<table>
<thead>
<tr>
<th>Feature</th>
<th>Phones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>b d g k p t</td>
</tr>
<tr>
<td>Nasal</td>
<td>m n ng</td>
</tr>
<tr>
<td>Fricative</td>
<td>ch dh f jh s sh th v z zh</td>
</tr>
<tr>
<td>Liquid</td>
<td>l r w y</td>
</tr>
<tr>
<td>Vowel</td>
<td>aa ae ah ao aw ax axr ay eh er ey ih ix iy ow oy uh uw</td>
</tr>
<tr>
<td>Front Vowel</td>
<td>ae eh ih ix iy</td>
</tr>
<tr>
<td>Central Vowel</td>
<td>aa ah ao axr er</td>
</tr>
<tr>
<td>Back Vowel</td>
<td>ax ow uh uw</td>
</tr>
<tr>
<td>High Vowel</td>
<td>ih ix iy uh uw</td>
</tr>
<tr>
<td>Rounded</td>
<td>ao ow oy uh uw w</td>
</tr>
<tr>
<td>Reduced</td>
<td>ax axr ix</td>
</tr>
<tr>
<td>Unvoiced</td>
<td>ch f hh k p s sh t th</td>
</tr>
<tr>
<td>Coronal</td>
<td>ch d dh jh l n r s sh t th z zh</td>
</tr>
</tbody>
</table>
Training a tied-mixture triphone AM:
Young, Odell, Woodland 1994

(1) Train monophone single Gaussian models

(2) Clone monophones to triphones

(3) Cluster and tie triphones

(4) Expand to GMMs

... etc.
Accents: An experiment

- A word by itself
- The word in context
Summary

- **Advanced in Search**
  - Multipass search
    - N-best lists
    - Lattices

- **Advances in Context**
  - Context-dependent (triphone) acoustic modeling

- **Metadata**
  - Disfluencies, etc