

CS 224S LING 281 Speech Recognition and Synthesis

Lecture 12: Search
Dan Jurafsky

CS 224S W2006 1

Part II: Search (= "Decoding")

- 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

CS 224S W2006 2

What we are searching for

- Given Acoustic Model (AM) and Language Model (LM):

$$(1) \hat{W} = \underset{W \in L}{\operatorname{argmax}} P(O|W)P(W)$$

AM (likelihood) LM (prior)
 ↓ ↓

CS 224S W2006 3

Combining Acoustic and Language Models

- We don't actually use equation (1)
 - (1) $\hat{W} = \underset{W \in L}{\operatorname{argmax}} P(O|W)P(W)$
 - AM underestimates acoustic probability
 - Why? Bad independence assumptions
 - Intuition: we compute (independent) AM probability estimates; but if we could look at context, we would assign a much higher probability. So we are underestimating
 - We do this every 10 ms, but LM only every word.
 - Besides: AM (as we've seen) isn't a true probability
 - AM and LM have vastly different dynamic ranges

CS 224S W2006 4

Language Model Scaling Factor

- Solution: add a language model weight (also called language weight LW or language model scaling factor LMSF)

$$(2) \hat{W} = \underset{W \in L}{\operatorname{argmax}} P(O|W)P(W)^{LMSF}$$

- Value determined empirically, is positive (why?)
- Often in the range 10 +- 5.

CS 224S W2006 5

Word Insertion Penalty

- But LM prob $P(W)$ also functions as penalty for inserting words
 - Intuition: when a uniform language model (every word has an equal probability) is used, LM prob is a $1/V$ penalty multiplier taken for each word
 - Each sentence of N words has penalty N/V
 - If penalty is large (smaller LM prob), decoder will prefer fewer longer words
 - If penalty is small (larger LM prob), decoder will prefer more shorter words
- When tuning LM for balancing AM, side effect of modifying penalty
- So we add a separate word insertion penalty to offset

$$(3) \hat{W} = \underset{W \in L}{\operatorname{argmax}} P(O|W)P(W)^{LMSF} WIP^{N(W)}$$

CS 224S W2006 6

Log domain

- We do everything in log domain
- So final equation:

$$(4) \hat{W} = \operatorname{argmax}_{W \in L} \log P(O|W) + LMSF \log P(W) + N \log WIP$$

CS 224S W2006

Language Model Scaling Factor

- As LMSF is increased:
 - More deletion errors (since increase penalty for transitioning between words)
 - Fewer insertion errors
 - Need wider search beam (since path scores larger)
 - Less influence of acoustic model observation probabilities

Text from John Hanjalic's slides 4

Word Insertion Penalty

- Controls trade-off between insertion and deletion errors
 - As penalty becomes larger (more negative)
 - More deletion errors
 - Fewer insertion errors
- Acts as a model of effect of length on probability
 - But probably not a good model (geometric assumption probably bad for short sentences)

CS 224S W2006
Text augmented from Bryan K. P. Colton's slides

Speeding things up

- Viterbi is $O(N^2T)$, where N is total number of HMM states, and T is length
- This is too large for real-time search
- A ton of work in ASR search is just to make search faster:
 - Beam search (pruning)
 - Fast match
 - Tree-based lexicons

CS 224S W2006

Beam search

- Instead of retaining all candidates (cells) at every time frame
- Use a threshold T to keep subset:
 - At each time t
 - Identify state with lowest cost D_{\min}
 - Each state with cost $> D_{\min} + T$ is discarded ("pruned") before moving on to time $t+1$

CS 224S W2006

Viterbi Beam search

- Is the most common and powerful search algorithm for LVCSR
- Note:
 - What makes this possible is time-synchronous
 - We are comparing paths of equal length
 - For two different word sequences W_1 and W_2 :
 - We are comparing $P(W_1|O_0^t)$ and $P(W_2|O_0^t)$
 - Based on same partial observation sequence O_0^t
 - So denominator is same, can be ignored
 - Time-asynchronous search (A*) is harder

CS 224S W2006

Viterbi Beam Search

- Empirically, beam size of 5-10% of search space
- Thus 90-95% of HMM states don't have to be considered at each time t
- Vast savings in time.

CS 224S W2006 14

Problems with Viterbi

- The forward probability of observation given word string

$$P(O|W) = \sum_{S \in S_1^T} P(O, S|W)$$

- The Viterbi algorithm makes the "Viterbi Approximation"

$$P(O|W) \approx \max_{S \in S_1^T} P(O, S|W)$$

- Approximates probability of observation given word, with prob of observation given only best state sequence.

CS 224S W2006 14

Problems with Viterbi and Solutions

- Best path can be very different than best word string if words have many possible pronunciations
- Impossible or expensive to take advantage of many useful knowledge sources
- Solutions
 - Modify Viterbi to return multiple paths, and use other knowledge to **rescore**.
 - Use a completely different decoding algorithm which both computes true Forward probability, and is easy to incorporate complex language models

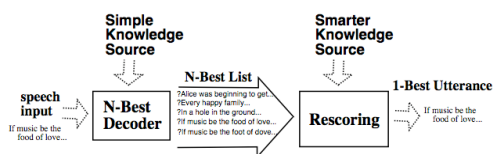
CS 224S W2006 15

N-best and multipass search algorithms

- The ideal search strategy would use every available knowledge source (KS)
- But is often difficult or expensive to integrate a very complex KS into first pass search
 - For example, parsers as a language model; have long-distance dependencies that violate dynamic programming assumptions
 - Other knowledge sources might not be left-to-right (knowledge of following words can help predict preceding words)
- Hence: **multipass search algorithms**

CS 224S W2006 16

Multipass Search



CS 224S W2006 17

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

CS 224S W2006 18

Sample N-best list

Rank	Path	AM logprob	LM logprob
1.	it's an area that's naturally sort of mysterious	-7193.53	-20.25
2.	that's an area that's naturally sort of mysterious	-7192.28	-21.11
3.	it's an area that's not really sort of mysterious	-7221.68	-18.91
4.	that scenario that's naturally sort of mysterious	-7189.19	-22.08
5.	there's an area that's naturally sort of mysterious	-7198.35	-21.34
6.	that's an area that's not really sort of mysterious	-7220.44	-19.77
7.	the scenario that's naturally sort of mysterious	-7205.42	-21.50
8.	so it's an area that's naturally sort of mysterious	-7195.92	-21.71
9.	that scenario that's not really sort of mysterious	-7217.34	-20.70
10.	there's an area that's not really sort of mysterious	-7226.51	-20.01

CS 224S W2006

19

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.
 - S. Young. 1984. "Generating Multiple Solutions from Connected Word DP Recognition Algorithms". Proc. of the Institute of Acoustics, 6:4, 351-354.
- 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!

CS 224S W2006

20

Computing N-best lists

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

CS 224S W2006

21

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

CS 224S W2006

22

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

CS 224S W2006

23

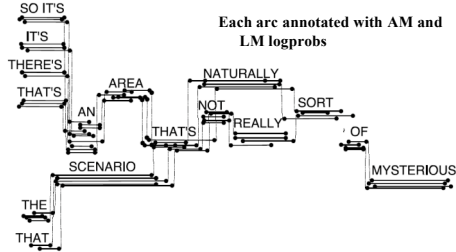
Exact N-best for time-synchronous Viterbi

- 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

CS 224S W2006

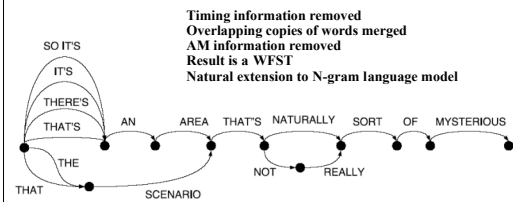
24

Word Lattice



CS 224S W2006 24

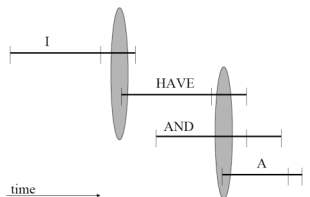
Word Graph



CS 224S W2006 25

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_{i+1} is one of the end-times of w_j



Bryan Felton's algorithm and figure from his slides

CS 224S W2006 27

Lattices

- 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

CS 224S W2006 28

Posterior lattices

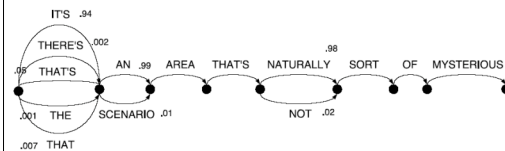
- We don't actually compute posteriors:

$$\hat{W} = \operatorname{argmax}_{W \in \mathcal{L}} \frac{P(O|W)P(W)}{P(O)} = \operatorname{argmax}_{W \in \mathcal{L}} P(O|W)P(W)$$

- 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 passing through word

CS 224S W2006 29

Mesh = Sausage = pinched lattice



CS 224S W2006 30

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

CS 224S W2006 31

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

CS 224S W2006 32

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 .

CS 224S W2006 33

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.

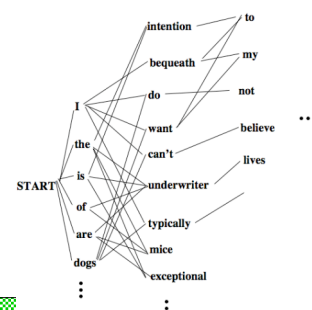
CS 224S W2006 34

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 $h(.)$

CS 224S W2006 35

A* Decoding (2)



36

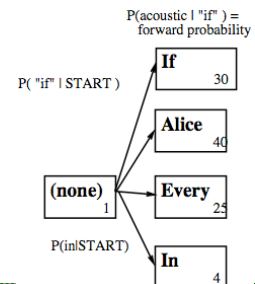
Stack decoding (A*) algorithm

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

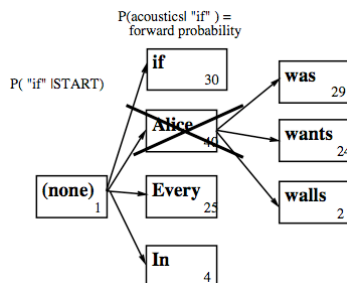
CS 224S W2006 37

A* Decoding (2)



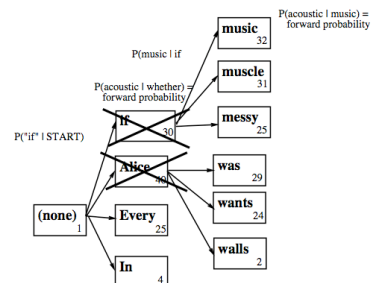
CS 224S W2006 38

A* Decoding (cont.)



CS 224S W2006 39

A* Decoding (cont.)



CS 224S W2006 40

Making A* work: $h(\cdot)$

- If $h(\cdot)$ is zero, breadth first search
- Stupid estimates of $h(\cdot)$:
 - 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!

CS 224S W2006 41

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

CS 224S W2006 42

Fast match

- 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

CS 224S W2006 44

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

CS 224S W2006 44

Forward-Backward Search

- First perform a forward search, computing partial forward scores α for each state
- Then do second pass search backwards
 - From last frame of speech back to first
- Using α 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

CS 224S W2006 45

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 Δ_t .
 - Score of final state of each word w in Δ_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 α^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

CS 224S W2006 46

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 ph_{w_i} , where w_i is first word of this partial path (last word expanded in backward search)
- Current score of path ph_{w_i} is $\beta_t(ph_{w_i})$
- We want to extend to next word w_j
- Two questions:
 - Find h^* heuristic for estimating future input stream
 - $\alpha_t(w_j)$!! So new score for word is $\alpha_t(w_j) + \beta_t(ph_{w_i})$
 - Find best crossing time t between w_i and w_j .
 - $t^* = \text{argmin}_t \{\alpha_t(w_j) + \beta_t(ph_{w_i})\}$

CS 224S W2006 47

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

CS 224S W2006 48