

CS 224S / LINGUIST 236 Speech Recognition and Synthesis

Dan Jurafsky

Lecture 12: Advanced Issues in LVCSR Search

IP Notice:

2/10/05

CS 224S Winter 2005

1

Outline

- Computing Word Error Rate
- Goal of search: how to combine AM and LM
- Viterbi search
 - Review and adding in LM
 - Beam search
 - Silence models
- A* Search
 - Fast match
- Tree structured lexicons
- N-Best and multipass search
 - N-best
 - Word lattice and word graph
 - Forward-Backward search (not related to F-B training)

2/10/05

CS 224S Winter 2005

2

Evaluation

- How do we evaluate recognizers?
- Word error rate

2/10/05

CS 224S Winter 2005

3

Word Error Rate

- Word Error Rate =
- $$100 \frac{(\text{Insertions} + \text{Substitutions} + \text{Deletions})}{\text{Total Word in Correct Transcript}}$$

Alignment example:

REF: portable ***** PHONE UPSTAIRS last night so

HYP: portable FORM OF STORES last night so

Eval: I S S S

WER = 100 (1+2+0)/6 = 50%

2/10/05

CS 224S Winter 2005

4

NIST sctk-1.3 scoring software: Computing WER with sclite

- <http://www.nist.gov/speech/tools/>
- Sclite aligns a hypothesized text (HYP) (from the recognizer) with a correct or reference text (REF) (human transcribed)

id: (2347-b-013)

Scores: (#C #S #D #I) 9 3 1 2

REF: was an engineer SO I i was always with ***** MEN UM

and they

HYP: was an engineer ** AND i was always with THEM THEY ALL

THAT and they

Eval: D S I I S S

2/10/05

CS 224S Winter 2005

5

More on sclite

SYSTEM SUMMARY PERCENTAGES by SPEAKER

./csrnab.hyp									
SPEAKER	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err	
4t0	15	458	84.1	14.0	2.0	2.6	18.6	86.7	
4t1	21	544	93.6	5.9	0.6	0.7	7.2	57.1	
4t2	15	404	91.3	8.7	0.0	2.5	11.1	86.7	
Sum/Avg	51	1406	89.8	9.3	0.9	1.8	12.0	74.5	
Mean	17.0	468.7	89.7	9.5	0.8	1.9	12.3	76.8	
S.D.	3.5	70.6	5.0	4.1	1.0	1.0	5.8	17.0	
Median	15.0	458.0	91.3	8.7	0.6	2.5	11.1	86.7	

2/10/05

CS 224S Winter 2005

6

ScLite output for error analysis

```

CONFUSION PAIRS          Total      (972)
                          With >= 1 occurrences (972)

1: 6 -> (hesitation) ==> on
2: 6 -> the ==> that
3: 5 -> but ==> that
4: 4 -> a ==> the
5: 4 -> four ==> for
6: 4 -> in ==> and
7: 4 -> there ==> that
8: 3 -> (hesitation) ==> and
9: 3 -> (hesitation) ==> the
10: 3 -> (a-) ==> i
11: 3 -> and ==> i
12: 3 -> and ==> in
13: 3 -> are ==> there
14: 3 -> as ==> is
15: 3 -> have ==> that
16: 3 -> is ==> this
    
```

2/10/05

CS 224S Winter 2005

7

ScLite output for error analysis

```

17: 3 -> it ==> that
18: 3 -> mouse ==> most
19: 3 -> was ==> is
20: 3 -> was ==> this
21: 3 -> you ==> we
22: 2 -> (hesitation) ==> it
23: 2 -> (hesitation) ==> that
24: 2 -> (hesitation) ==> to
25: 2 -> (hesitation) ==> yeah
26: 2 -> a ==> all
27: 2 -> a ==> know
28: 2 -> a ==> you
29: 2 -> along ==> well
30: 2 -> and ==> it
31: 2 -> and ==> we
32: 2 -> and ==> you
33: 2 -> are ==> i
34: 2 -> are ==> were
    
```

2/10/05

CS 224S Winter 2005

8

Summary on WER

- WER is clearly better than metrics like e.g., perplexity
- But should we be more concerned with meaning ("semantic error rate")?
 - Good idea, but hard to agree on
 - Has been applied in dialogue systems, where desired semantic output is more clear
- Recent research: modify training to directly minimize WER instead of maximizing likelihood

2/10/05

CS 224S Winter 2005

9

Part II: Search

2/10/05

CS 224S Winter 2005

10

What we are searching for

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

$$(1) \hat{W} = \arg \max_{W \in L} P(O|W)P(W)$$

AM (likelihood) LM (prior)

2/10/05

CS 224S Winter 2005

11

Combining Acoustic and Language Models

- We don't actually use equation (1)

$$(1) \hat{W} = \arg \max_{W \in L} P(O|W)P(W)$$
 - AM underestimates acoustic probability
 - Why? Bad independence assumptions
 - Intuition: we compute (independent) AM probability estimates every 10 ms; but LM only every word.
 - AM and LM have vastly different dynamic ranges

2/10/05

CS 224S Winter 2005

12

Language Model Scaling Factor

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

$$(2) \hat{W} = \arg \max_{W \in L} P(O|W)P(W)^{LMSF}$$

- Value determined empirically, is positive (why?)
- For Sphinx, similar systems, generally in the range 10 +/- 3.

2/10/05

CS 224S Winter 2005

13

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 $1/N$ penalty multiplier taken for each word
- If penalty is large, decoder will prefer fewer longer words
- If penalty is small, decoder will prefer more shorter words
- When tuning LM for balancing AM, side effect of penalty
- So we add a separate word insertion penalty to offset

$$(3) \hat{W} = \arg \max_{W \in L} P(O|W)P(W)^{LMSF} WIP^{N(W)}$$

2/10/05

CS 224S Winter 2005

14

Log domain

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

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

2/10/05

CS 224S Winter 2005

15

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

2/10/05

CS 224S Winter 2005

16
Text from Bryan Pellom's slides

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)

2/10/05

CS 224S Winter 2005

17

Text augmented from Bryan Pellom's slides

Part III: More on Viterbi

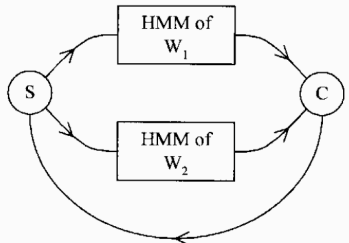
2/10/05

CS 224S Winter 2005

18

Adding LM probabilities to Viterbi: (1) Uniform LM

- Visualizing the search space for 2 words



2/10/05

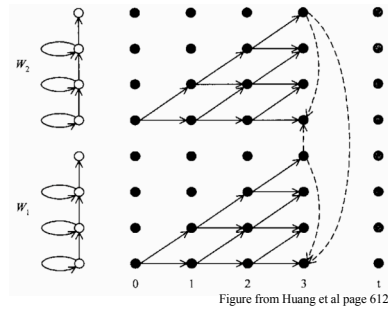
CS 224S Winter 2005

19

Figure from Huang et al page 611

Viterbi trellis with 2 words and uniform LM

- Null transition from the end-state of each word to start-state of all (both) words.

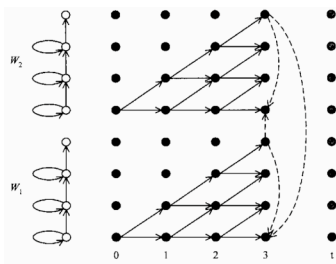


2/10/05

Figure from Huang et al page 612

Viterbi for 2 word continuous recognition

- Viterbi search: computations done *time-synchronously* from left to right, i.e. each cell for time t is computed before proceeding to time $t+1$



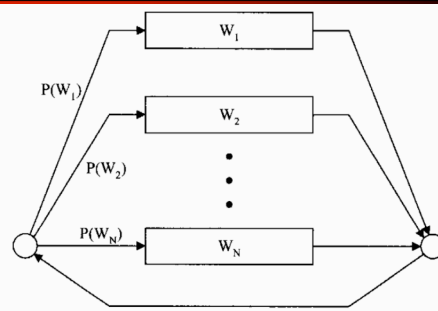
2/10/05

CS 224S Winter 2005

21

Text from Kjell Elenius course slides; figure from Huang page 612

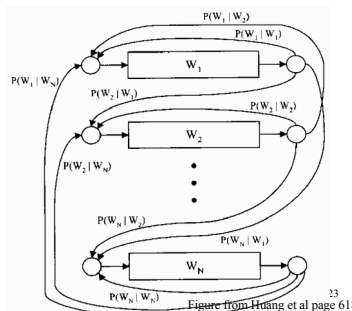
Search space for unigram LM



22

Figure from Huang et al page 617

Search space with bigrams

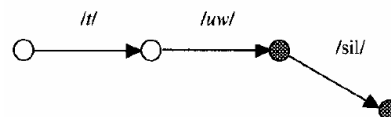


2/10/05

Figure from Huang et al page 618

Silences

- Each word HMM has optional silence at end
- Model for word "two" with two final states.



2/10/05

CS 224S Winter 2005

24

Reminder: Viterbi approximation

- Correct equation

$$(1) \hat{W} = \arg \max_{W \in L} P(O|W)P(W)$$

- We approximate $P(O|W)$:

$$(5) \hat{W} = \arg \max_{W \in L} P(W) \max_{s_0^T} P(O, s_0^T | W)$$

- Often called "the Viterbi approximation"
- "The most likely word sequence is approximated by the most likely state sequence"

2/10/05

CS 224S Winter 2005

25

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

2/10/05

CS 224S Winter 2005

26

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 $t+1$

2/10/05

CS 224S Winter 2005

27

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

2/10/05

CS 224S Winter 2005

28

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.

2/10/05

CS 224S Winter 2005

29

Part IV: A^* Search

2/10/05

CS 224S Winter 2005

30

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

2/10/05

CS 224S Winter 2005

31

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 .

2/10/05

CS 224S Winter 2005

32

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.

2/10/05

CS 224S Winter 2005

33

A* search for speech

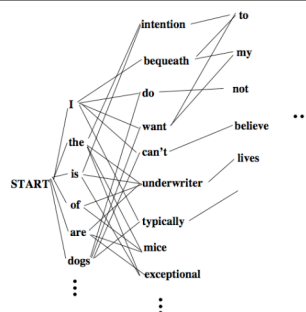
- The search space is the set of possible sentences
- The forward algorithm can tell us the cost of the current path so far $g(.)$
- We need an estimate of the cost from the current node to the end $h(.)$

2/10/05

CS 224S Winter 2005

34

A* Decoding (2)



2/10/05

Stack decoding (A*) algorithm

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

```

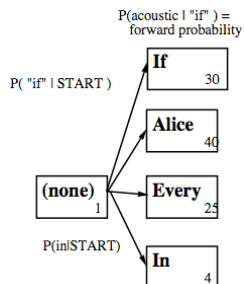
Initialize the priority queue with a null sentence.
Pop the best (highest score) sentence  $s$  off the queue.
If ( $s$  is marked end-of-sentence (EOS) ) output  $s$  and terminate.
Get list of candidate next words by doing fast matches.
For each candidate next word  $w$ :
  Create a new candidate sentence  $s + w$ .
  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
  
```

2/10/05

CS 224S Winter 2005

36

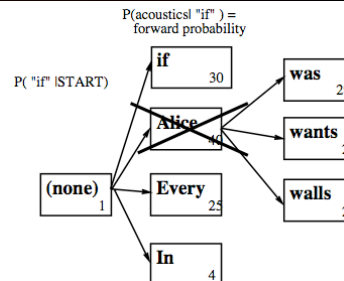
A* Decoding (2)



2/10/05

CS 224S Winter 2005

A* Decoding (cont.)

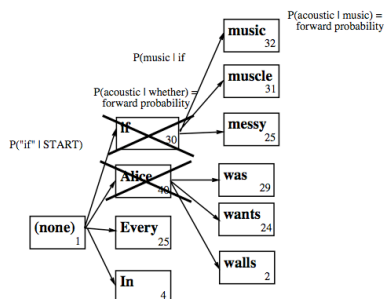


2/10/05

CS 224S Winter 2005

38

A* Decoding (cont.)



2/10/05

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!

2/10/05

CS 224S Winter 2005

40

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

2/10/05

CS 224S Winter 2005

41

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

2/10/05

CS 224S Winter 2005

42

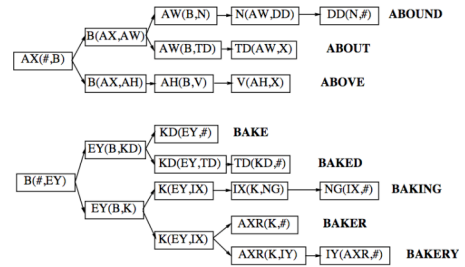
Part V: Tree structured lexicons

2/10/05

CS 224S Winter 2005

43

Tree structured lexicon



2/10/05

CS 224S Winter 2005

44

Part VI: N-best and multipass search

2/10/05

CS 224S Winter 2005

45

N-best and multipass search algorithms

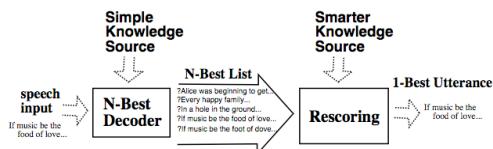
- The ideal search strategy would use every available knowledge source (KS)
- But is often difficult or expensive to integrate a very complex KS into first pass search
 - For example, parsers as a language model; have long-distance dependencies that violate dynamic programming assumptions
 - Other knowledge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

2/10/05

CS 224S Winter 2005

46

Multipass Search



2/10/05

CS 224S Winter 2005

47

Some definitions

- **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

2/10/05

CS 224S Winter 2005

48

N-best list

1. I will tell you would I think in my office
2. I will tell you what I think in my office
3. I will tell you when I think in my office
4. I would sell you would I think in my office
5. I would sell you what I think in my office
6. I would sell you when I think in my office
7. I will tell you would I think in my office
8. I will tell you why I think in my office
9. I will tell you what I think on my office
10. I Wilson you I think on my office

2/10/05

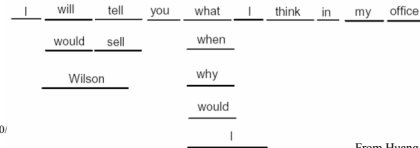
CS 224S Winter 2005

From Huang et al, page 664

49

Word lattice

- Encodes
 - Word
 - Starting/ending time(s) of word
 - Acoustic score of word
- More compact than N-best list:
 - Utterance with 10 words, 2 hypos per word
 - 1024 different sentences
 - Lattice with only 20 different hypotheses

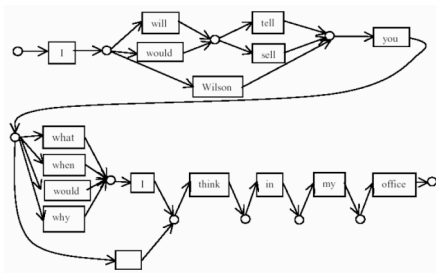


2/10/

50

From Huang et al, page 665

Word Graph



2/10/05

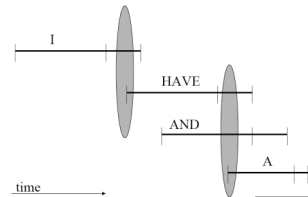
CS 224S Winter 2005

From Huang et al, page 665

51

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



2/10/05

CS 224S Winter 2005

52

Bryan Pellom's algorithm and figure, from his slides

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

2/10/05

CS 224S Winter 2005

53

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!

2/10/05

CS 224S Winter 2005

54

Computing N-best lists

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

2/10/05

CS 224S Winter 2005

55

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

2/10/05

CS 224S Winter 2005

56

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

2/10/05

CS 224S Winter 2005

57

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

2/10/05

CS 224S Winter 2005

58

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

2/10/05

CS 224S Winter 2005

59

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

2/10/05

CS 224S Winter 2005

60

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

2/10/05

CS 224S Winter 2005

61

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_j} , where w_j is first word of this partial path (last word expanded in backward search)
- Current score of path ph_{w_j} is $\beta_t(ph_{w_j})$
- 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 $\alpha_t(w_i) + \beta_t(ph_{w_j})$
 - Find best crossing time t between w_i and w_j .
 - $t^* = \text{argmin}_t (\alpha_t(w_i) + \beta_t(ph_{w_j}))$

2/10/05

CS 224S Winter 2005

62

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

2/10/05

CS 224S Winter 2005

63

Summary

- **Computing Word Error Rate**
- **Goal of search: how to combine AM and LM**
- **Viterbi search**
 - Review and adding in LM
 - Beam search
 - Silence models
- **A* Search**
 - Fast match
- **Tree structured lexicons**
- **N-Best and multipass search**
 - N-best
 - Word lattice and word graph
 - Forward-Backward search (not related to F-B training)

2/10/05

CS 224S Winter 2005

64