Doctoral Course
in
Speech Recognition

Part 3

Mats Blomberg, Kjell Elenius, Alec Seward

September-December 2003

General course info

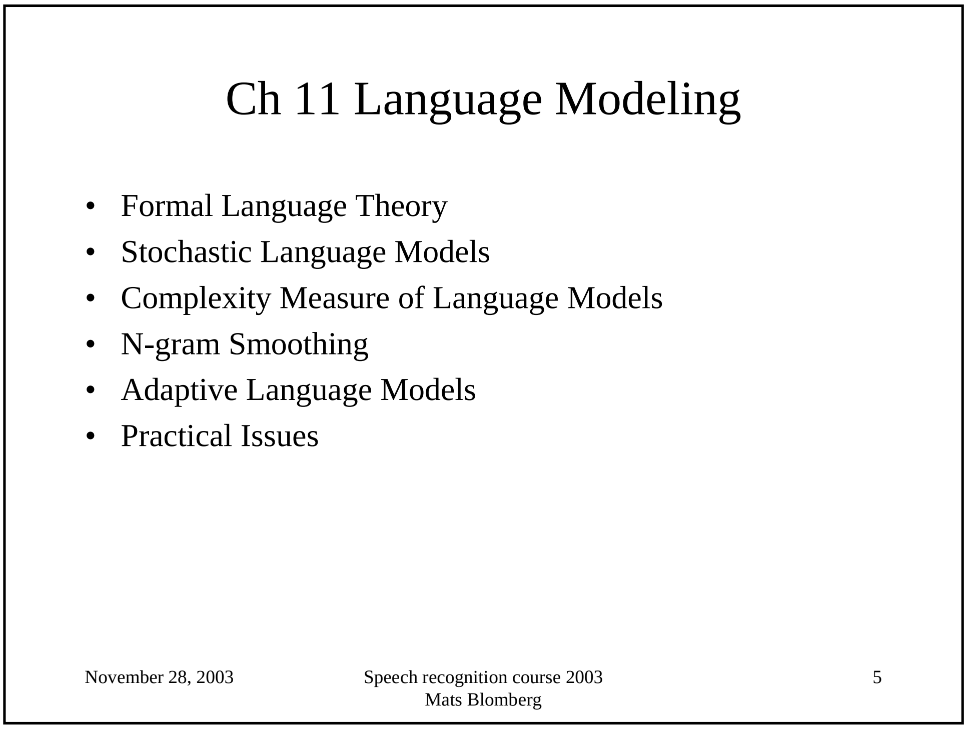
- Exercises
  - Return solutions by Dec. 5
- Term paper
  - Choose topic by Nov 21
  - Around 6 pages, max 10
  - Send to reviewers (2 course participants) by Dec 19
  - Reviewer return comments by Jan 10
  - Final paper to Mats by Jan. 24
- Closing seminar
  - Feb 6
  - Presentation of own paper
  - Active discussions

Exercises

- Questions?
- Corrections
  - Ex 1. VQ: Modified initialisation values
  - Ex. 3. Viterbi and **Forward** probabilities

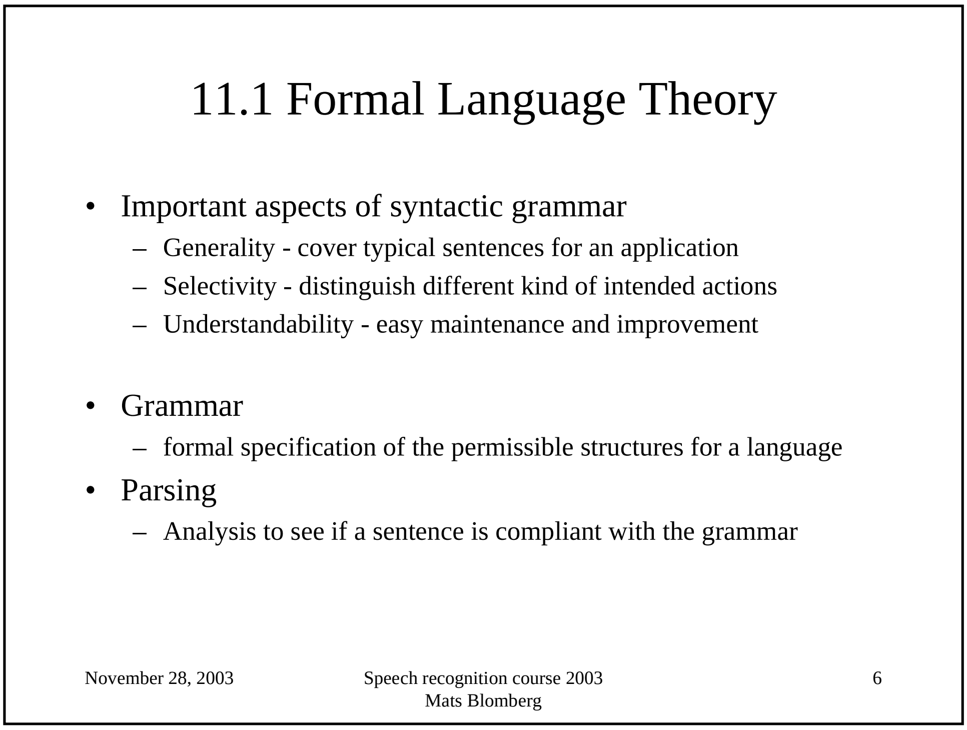
Course overview

- Day #1
  - Probability, Statistics and Information Theory (pp 73-131: 59 pages)
  - Pattern Recognition (pp 133-197: 65 pages)
  - Speech Signal Representations (pp 275-336 62 pages)
  - Hidden Markov Models (pp 377-413: 37 pages)
- Day #2
  - Hidden Markov Models (cont.)
  - Acoustic Modeling (pp 415-475: 61 pages)
  - Environmental Robustness (pp 477-544: 68 pages)
  - HTK tutorial (Giampi)
- Day #3
  - Language Modeling (pp 545-590: 46 pages) (Mats)
  - Basic Search Algorithms (pp 591-643: 53 pages) (Kjell)
  - ~~Large Vocabulary Search Algorithms~~
  - Finite State Transducers (Alec Seward)
  - (Applications and User Interfaces)
- Day #4 Closing seminar
  - Presentations of term papers



Ch 11 Language Modeling

- Formal Language Theory
- Stochastic Language Models
- Complexity Measure of Language Models
- N-gram Smoothing
- Adaptive Language Models
- Practical Issues

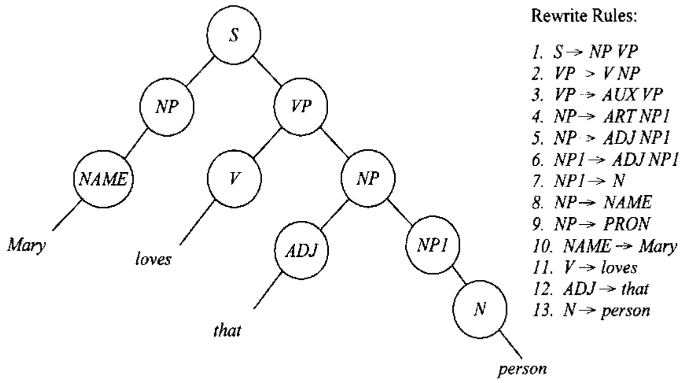


11.1 Formal Language Theory

- Important aspects of syntactic grammar
  - Generality - cover typical sentences for an application
  - Selectivity - distinguish different kind of intended actions
  - Understandability - easy maintenance and improvement
- Grammar
  - formal specification of the permissible structures for a language
- Parsing
  - Analysis to see if a sentence is compliant with the grammar

Tree representation

- The most common way to represent the grammatical structure of a sentence



Rewrite Rules:

1. S → NP VP
2. VP → V NP
3. VP → AUX VP
4. NP → ART NP1
5. NP → ADJ NP1
6. NP1 → ADJ NP1
7. NP1 → N
8. NP → NAME
9. NP → PRON
10. NAME → Mary
11. V → loves
12. ADJ → that
13. N → person

11.1.1 Chomsky Hierarchy

- Chomsky’s formal language theory
- A grammar is defined as G = (V,T, P, S)
  - V: non-terminal
  - T: terminal
  - P: Set of production rules
  - S: start symbol
- Analysis by sequential application of production rules
- Production rule type $\alpha \rightarrow \beta$, α, β strings of V and T
- Four major languages, hierarchically structured
- Major implementation tool in comp. linguistics
  - finite state automaton

Chomsky hierarchy and corresponding machines

Types	Constraints	Automata
Phrase structure grammar	$\alpha \rightarrow \beta$. The most general grammar. α, β: strings of non-terminals and terminals	Turing machine
Context-sensitive grammar	Subset of phrase structure grammar. $\|\alpha\| \leq \|\beta\|$	Linear bounded automata
Context-free grammar (	Subset of context-sensitive grammar $A \rightarrow \beta$, A: non-terminal, β: w or BC	Push down automata
Regular grammar	Subset of CFG $A \rightarrow w$ and $A \rightarrow wB$	Finite-state automata

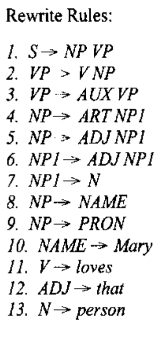
Push-down automata

- Also called Recursive Transition Network
- Transition Network: nodes and labeled arcs
- Parsing
  - Start at the initial state S
  - Traverse an arc if current word is in the arc category
  - If arc is followed, update current word
  - A phrase is parsed if there is a path from S to a *pop* (final) arc
  - More than one parse is possible

11.1.2 Chart Parsing for Context-Free Grammars

Top Down or Bottom Up Parsing?

- Top-down
  - Start from the root of the tree, successive rewrites into terminal symbols matching the input text
  - Goal-directed search
  - Example “Mary loves that person”
    - S
    - → NP VP
    - → NAME VP (rewrite S using S→NP)
    - → Mary VP (rewrite NP using NAME→Mary)
    - …
    - → Mary loves that person (rewrite N using N→person)

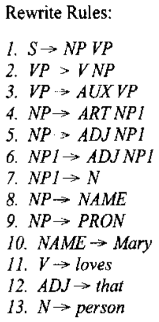


Rewrite Rules:

1. *S → NP VP*
2. *VP > V NP*
3. *VP → AUX VP*
4. *NP → ART NP1*
5. *NP → ADJ NP1*
6. *NP1 → ADJ NP1*
7. *NP1 → N*
8. *NP → NAME*
9. *NP → PRON*
10. *NAME → Mary*
11. *V → loves*
12. *ADJ → that*
13. *N → person*

Top Down or Bottom Up Parsing?

- Bottom-up
  - – Start with the words in the input text
  - – Use the rewrite rules backwards
  - – Example “Mary loves that person”
    - → NAME loves that person (rewrite Mary using NAME →Mary
    - → NAME V that person (rewrite loves using V →loves
    - …
    - →NP VP
    - →S (rewrite NP using S →NP VP)



Rewrite Rules:

1. *S → NP VP*
2. *VP → V NP*
3. *VP → AUX VP*
4. *NP → ART NP1*
5. *NP → ADJ NP1*
6. *NP1 → ADJ NP1*
7. *NP1 → N*
8. *NP → NAME*
9. *NP → PRON*
10. *NAME → Mary*
11. *V → loves*
12. *ADJ → that*
13. *N → person*

November 28, 2003 Speech recognition course 2003 Mats Blomberg 13

Top Down or Bottom Up Parsing?

- Top-down parsing features
  - – Very predictive
  - – Only considers grammatical combinations
  - – Predicts constituents that does not have a match in the text
- Bottom-up parsing features
  - – Checks input only once
  - – May build trees that can’t lead to full parse
  - – Suitable for robust language processing (see Ch. 17)
- Similar performance

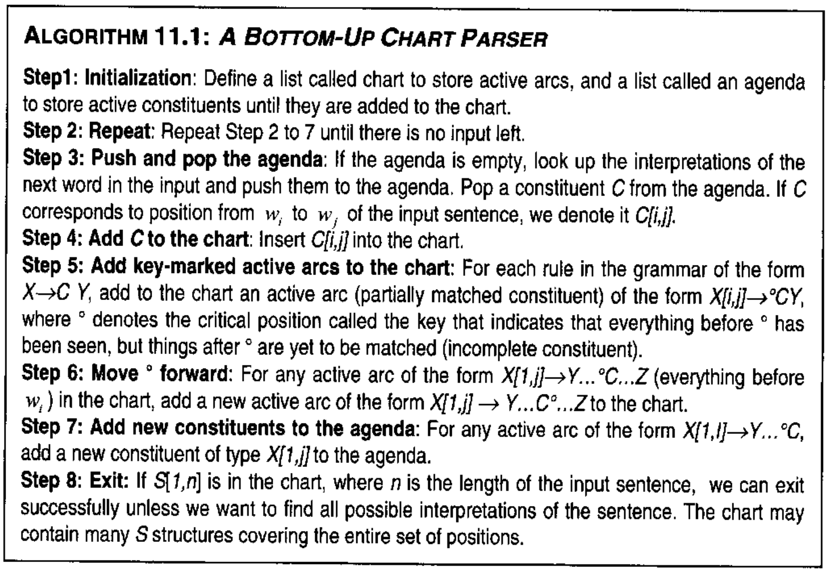
November 28, 2003 Speech recognition course 2003 Mats Blomberg 14

Bottom-Up Chart Parsing

- Basic principle: Store partial parsing results in a *chart* to eliminate duplicate work
- Parsing does not need to be left-to-right
- The chart maintains derived constituents and partially matched rules (*active arcs*)
- *Active constituents* represent subparts of the sentence according to the rewrite rules
- Active constituents are stored in an *agenda*

Bottom-Up Chart Parsing cont.

- Operation
  - Identify rules starting with the active constituent or rules that are partially identified and extend these
  - Combine partially matched records with completed constituent to form a new completed constituent or a new partially matched consitutent
  - Depth-first or breadth-first search
    - Breadth-first better if probabilities are used



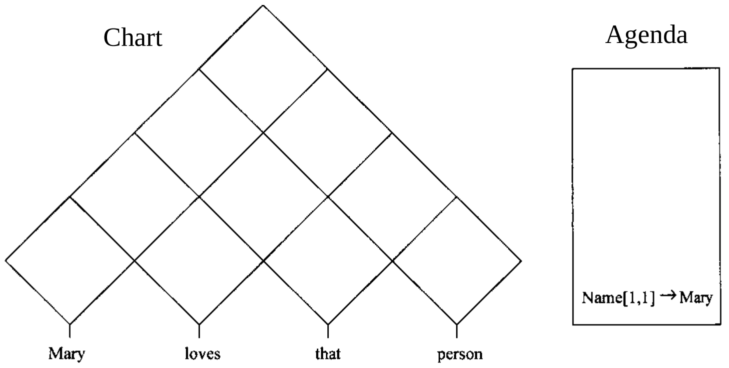
ALGORITHM 11.1: *A BOTTOM-UP CHART PARSER*

Step1: Initialization: Define a list called chart to store active arcs, and a list called an agenda to store active constituents until they are added to the chart.
Step 2: Repeat: Repeat Step 2 to 7 until there is no input left.
Step 3: Push and pop the agenda: If the agenda is empty, look up the interpretations of the next word in the input and push them to the agenda. Pop a constituent *C* from the agenda. If *C* corresponds to position from w_i to w_j of the input sentence, we denote it *C[i,j]*.
Step 4: Add *C* to the chart: Insert *C[i,j]* into the chart.
Step 5: Add key-marked active arcs to the chart: For each rule in the grammar of the form *X→C Y*, add to the chart an active arc (partially matched constituent) of the form *X[i,j]→°CY*, where ° denotes the critical position called the key that indicates that everything before ° has been seen, but things after ° are yet to be matched (incomplete constituent).
Step 6: Move ° forward: For any active arc of the form *X[1,j]→Y...°C...Z* (everything before w_i) in the chart, add a new active arc of the form *X[1,j] → Y...C°...Z* to the chart.
Step 7: Add new constituents to the agenda: For any active arc of the form *X[1,I]→Y...°C*, add a new constituent of type *X[1,j]* to the agenda.
Step 8: Exit: If *S*[*1,n*] is in the chart, where *n* is the length of the input sentence, we can exit successfully unless we want to find all possible interpretations of the sentence. The chart may contain many *S* structures covering the entire set of positions.

Algorithm: A Bottom-Up Chart Parser

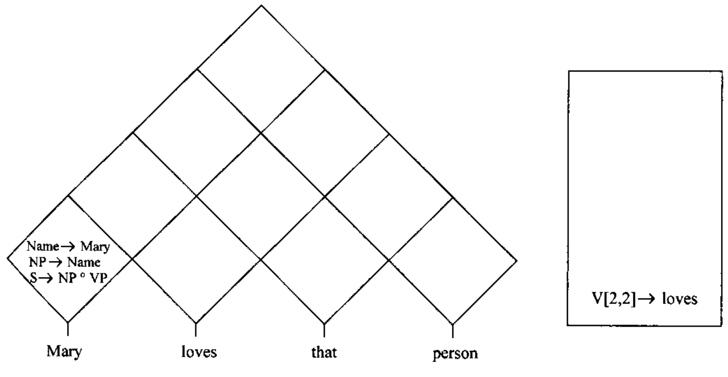
- 1. Initialization
- 2. Repeat 2 to 7 until all input words are processed
- 3. Push input word interpretation to, pop constituent from the agenda
- 4. Add the constituent to the chart
- 5. Find and add partial matches (key-marked) to the chart
- 6. Extend partial matches (Move the keys forward)
- 7. Put the partial matches to the agenda
- 8. Exit, successfully if the whole sentence is interpreted
  - continue if all sentence interpretations are required

Bottom-Up Chart Parsing example (1)



Look up interpretations of the next input word → push to Agenda
Pop constituent from Agenda, insert in the chart

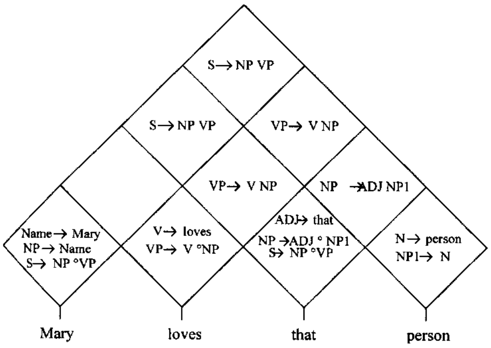
Bottom-Up Chart Parsing example (2)



(b) After Mary, the chart now has rules *Name→Mary*, *NP→Name*, and *S→NP°VP*.

Find partially matched rules

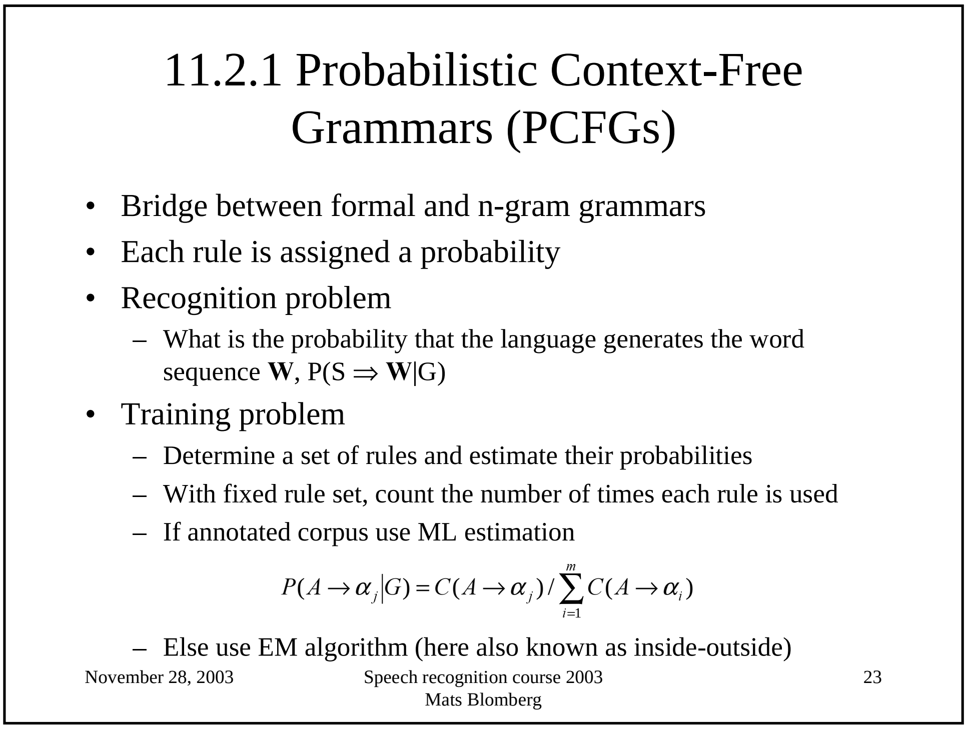
Bottom-Up Chart Parsing example (3)



(c) The chart after the whole sentence is parsed. S→ NP VP covers the whole sentence, indicating that the sentence is parsed successfully by the grammar.

11.2 Stochastic Language Models (SLM)

- In formal languages, P(**W**) = 1 or 0 for accept/reject
- Inappropriate for spoken language since
  - Incomplete grammar coverage
  - Speech is often ungrammatical
- Probabilistic Context-Free Grammars (PCFG)
- N-gram Language models

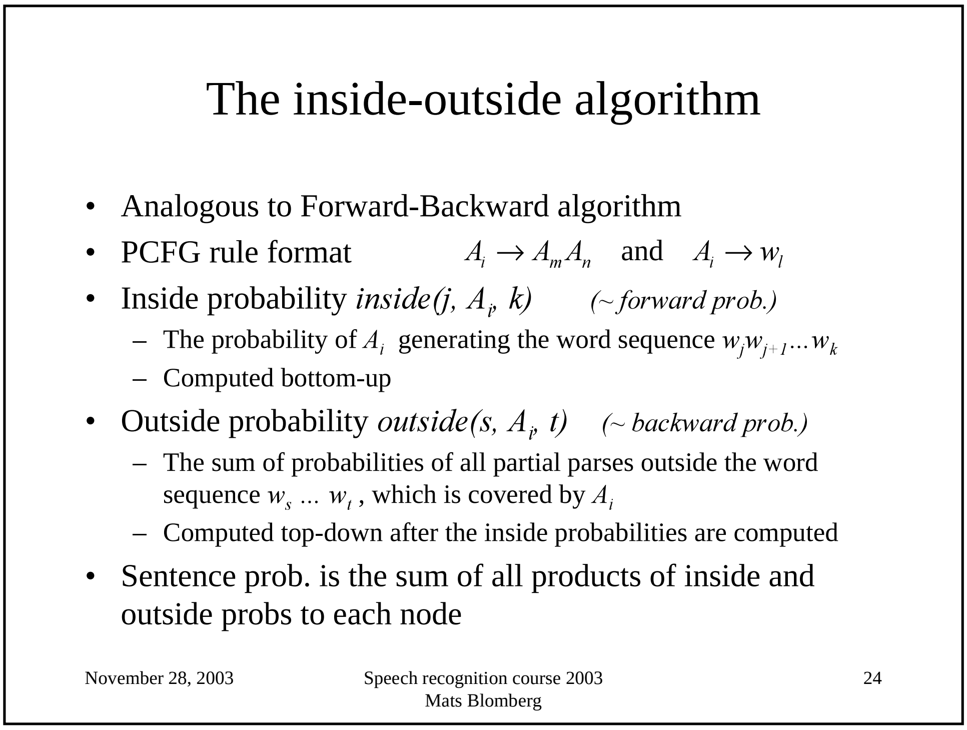


11.2.1 Probabilistic Context-Free Grammars (PCFGs)

- Bridge between formal and n-gram grammars
- Each rule is assigned a probability
- Recognition problem
  - What is the probability that the language generates the word sequence **W**, P(S ⇒ **W**|G)
- Training problem
  - Determine a set of rules and estimate their probabilities
  - With fixed rule set, count the number of times each rule is used
  - If annotated corpus use ML estimation

$$P(A \rightarrow \alpha_j | G) = C(A \rightarrow \alpha_j) / \sum_{i=1}^{m} C(A \rightarrow \alpha_i)$$

  - Else use EM algorithm (here also known as inside-outside)



The inside-outside algorithm

- Analogous to Forward-Backward algorithm
- PCFG rule format $A_i \rightarrow A_m A_n$ and $A_i \rightarrow w_l$
- Inside probability *inside(j, A_i, k)* *(~ forward prob.)*
  - The probability of A_i generating the word sequence $w_j w_{j+1} \ldots w_k$
  - Computed bottom-up
- Outside probability *outside(s, A_i, t)* *(~ backward prob.)*
  - The sum of probabilities of all partial parses outside the word sequence $w_s \ldots w_t$, which is covered by A_i
  - Computed top-down after the inside probabilities are computed
- Sentence prob. is the sum of all products of inside and outside probs to each node

The inside algorithm

$$inside(j,A_i,k) = P(A_i \Rightarrow w_j w_{j+1} \ldots w_k) = \sum_{n,m}\sum_{l=j}^{k-1} P(A_i \rightarrow A_m A_n) P(A_m \Rightarrow w_j \ldots w_l) P(A_n \Rightarrow w_{l+1} \ldots w_k)$$

$$= \sum_{n,m}\sum_{l=j}^{k-1} P(A_i \rightarrow A_m A_n) inside(j,A_m,l) inside(l+1,A_n,k)$$

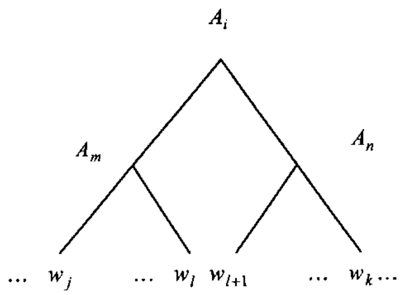
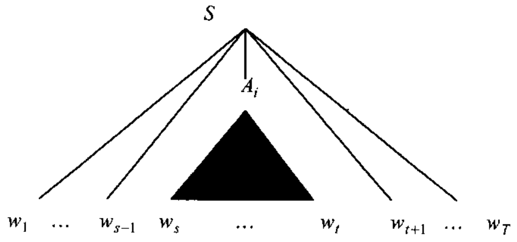


Figure 11.3 Inside probability is computed recursively as sum of all the derivations.

The outside algorithm

- Outside probability *outside(s, A_i, t)*
  - The sum of probabilities of all partial parses outside the word sequence w_s ... w_t , which is covered by A_i

$$outside(s,A_i,t) = P(S \Rightarrow w_1 \ldots w_{s-1} A_i w_{t+1} \ldots w_T) = \ldots$$



PCFG Rule probability

- Probability of rule $A_i \rightarrow A_m A_n$ covering words $w_s \ldots w_t$

$$\xi(i,m,n,s,t) = P(A_i \Rightarrow w_s ... w_t, A_i \rightarrow A_m A_n | S \Rightarrow \mathbf{W}, G)$$

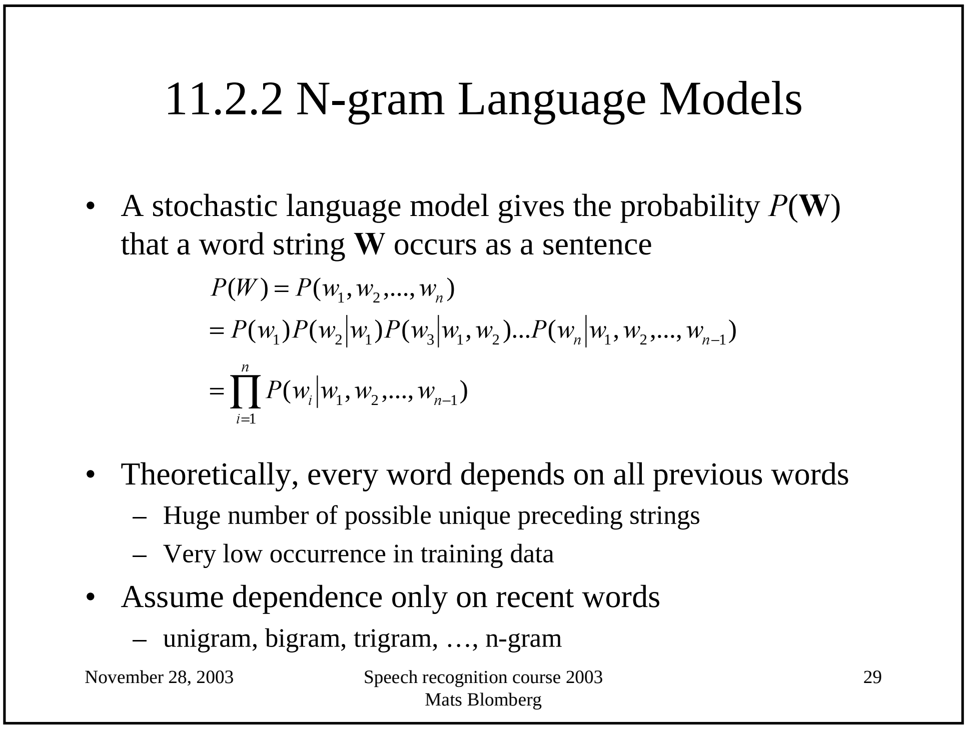
$$= \frac{1}{P(S \Rightarrow \mathbf{W}|G)} \sum_{k=s}^{t-1} P(A_i \rightarrow A_m A_n | G) inside(s, A_m, k) inside(k+1, A_n, t) outside(s, A_i, t)$$

- Probability on all word spans in the sentence

$$P(A_i \rightarrow A_m A_n \,\Big|\, G) = \frac{\sum_{s=1}^{T-1} \sum_{t=s+1}^{T} \xi(i,m,n,s,t)}{\sum_{m,n} \sum_{s=1}^{T-1} \sum_{t=s+1}^{T} \xi(i,m,n,s,t)}$$

PCFG Rule estimation aspects

- Only select rules with sufficient probabilities
  - Risk that low probability rules generate too many greedy symbols
- Only local maximum guaranteed (as in F-B)
- Problems
  - Assumes independence between the expansion of non-terminals
  - Lack of word sensitivity within word class

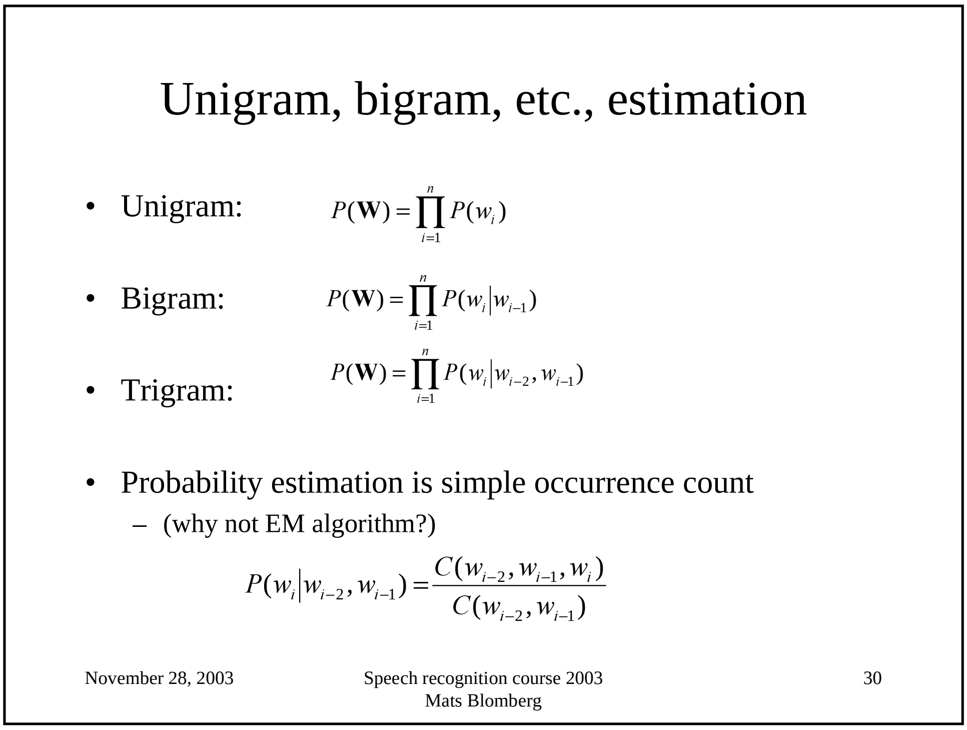


11.2.2 N-gram Language Models

- A stochastic language model gives the probability $P(\mathbf{W})$ that a word string $\mathbf{W}$ occurs as a sentence

$$
\begin{aligned}
&P(W) = P(w_1, w_2, ..., w_n) \\
&= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)...P(w_n \mid w_1, w_2, ..., w_{n-1}) \\
&= \prod_{i=1}^{n} P(w_i \mid w_1, w_2, ..., w_{n-1})
\end{aligned}
$$

- Theoretically, every word depends on all previous words
  - Huge number of possible unique preceding strings
  - Very low occurrence in training data
- Assume dependence only on recent words
  - unigram, bigram, trigram, …, n-gram



Unigram, bigram, etc., estimation

- Unigram: $P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i)$
- Bigram: $P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$
- Trigram: $P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$
- Probability estimation is simple occurrence count
  - (why not EM algorithm?)

$$
P(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}
$$

11.3 Complexity Measure of Language Models

- Test-set perplexity
  - Evaluates the generalization capability of the language model
- Training-set perplexity
  - Measures how the language model fits the training data
- Typical perplexity values
  - Digit strings: 10
  - n-gram on English text 50 - 1000
  - Wall Street Journal test set
    - trigram 128
    - bigram 176

11.4 N-Gram Smoothing

- Problem
  - Many very possible word sequences may have been observed in zero or very low numbers in the training data
  - Leads to extremely low probabilities, effectively disabling this word sequence, no matter how strong the acoustic evidence is
- Solution: smoothing
  - produce more robust probabilities for unseen data at the cost of modeling the training data slightly worse

N-gram Smoothing - simple technique

- Add constant (often 1) to all word sequence counts

Interpolation and Backoff Smoothing

- Interpolation models
  - Linear combination with lower order n-grams
  - Modifies the probabilities of *both* nonzero and zero count n-grams
- Backoff models
  - Use lower order n-grams when the requested n-gram has zero or very low count in the training data
  - Computes models with zero count from lower order n-grams.
  - Nonzero count n-grams are *unchanged*
  - *Discounting*
    - Reduce the probability of seen n-grams and distribute among unseen ones

11.4.1 Deleted Interpolation Smoothing

- Interpolation between n-grams of different length
- Example on combination of unigrams and bigrams

$$P_I(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)$$

- The optimal λ is specific for each word history
  - A high-frequent context generally gets higher weight
  - Requires enormous amount of training data
- Cluster into moderate number of weights

11.4.2 Backoff Smoothing

- **Good-Turing Estimate**
  - Partition n-grams into groups depending on their frequency in the training data
  - Change the number of occurrences of an n-gram according to

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

    - where r is the occurrence number
    - n_r is the number of n-grams that occur r times
- The **Katz smoothing** extends the Good-Turing estimate by combining higher and lower order models
- Bigram example:

$$C^*(w_{i-1}w_i) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1})P(w_i) & \text{if } r = 0 \end{cases} \qquad d_r \approx r^*/r$$

$\alpha(w_{i-1})$ is computed to satisfy the probability constraints

- Discount non-zero bigrams and distribute among zero-count bigrams

Alternative Backoff Models

- **Kneser-Ney smoothing**
  - Background
    - Lower order n-grams are often used as backoff model if the count of a higher-order n-gram is too low (e.g. unigram instead of bigram)
  - Problem example
    - Some words with relatively high unigram probability only occur in a few bigrams. E.g. *Francisco,* which is mainly found in *San Francisco*. However, infrequent word pairs, such as *New Francisco,* will be given too high probability if the unigram probabilities of *New* and *Francisco* are used. Maybe instead, the *Francisco* unigram should have a lower value to prevent it from occurring in other contexts.
  - Method
    - Instead of counting the occurrences of a unigram, count the number of *word identities* that it follows.
    - $P_{KN}(w_i)$ = (The number of *word identities* that it follows) / (The vocabulary size)
    - Discount and interpolate to estimate smoothed bigrams from KN unigrams and low-frequency bigrams

11.4.3 Class N-grams

- Group words into semantic or grammatical classes and build n-grams for class sequences

$$P(w_i \mid c_{i-n+1}...c_{i-1}) = P(w_i \mid c_i)P(c_i \mid c_{i-n+1}...c_{i-1})$$

- Benefits
  - rapid adaptation, small training sets, reduced memory requirement
- Very helpful for limited domain recognition
- Classes can be rule-based or data-driven
  - Rule-based classes useful in domain-specific systems
  - Data-driven in general-purpose systems

11.4.4 Performance of N-gram Smoothing

- Best: Kneser-Ney
- Next: Katz and Deleted Interpolation
- All three significantly better than No Smoothing

11.5 Adaptive Language Models

- Dynamic adjustment of the language model
  - Conversation topic is unstationary
  - Topic remains for some period of time
- Techniques
  - Cache Language Models
  - Topic-Adaptive Models
  - Maximum Entropy Models

11.5.1 Cache Language Models

- Basic idea
  - Accumulate n-grams spoken so far
  - Use these to create local (low-order) dynamic n-gram models
  - Interpolate with static n-gram

$$P_{cache}(w_i \mid w_{i-n+1} \ldots w_{i-1})$$
$$= \lambda_c P_{static}(w_i \mid w_{i-n+1} \ldots w_{i-1}) + (1-\lambda_c) P_{cache}(w_i \mid w_{i-2} w_{i-1})$$

  - Accounts for the fact that many words tend to be repeated during e.g. a conversation or dictation
  - But doesn’t account for higher probability of words in the same category (topic-specific words)

11.5.2 Topic-Adaptive Models

- Topic information can improve the static language model
  - The most probable word after “*the operating”* in a hospital is different from that in an office
- Topic-clustered language models
  - Manual or data-driven (better)
  - Use information retrieval techniques to find the appropriate documents in the training database
    - Step 1: Use what is recognized so far to find similar documents
    - Step 2: Adapt the topic-independent model to these documents
    - Retrieval measure: TFIDF (Term Frequency - Inverse Document Frequency) for determining document similarity

11.5.3 Maximum Entropy Models

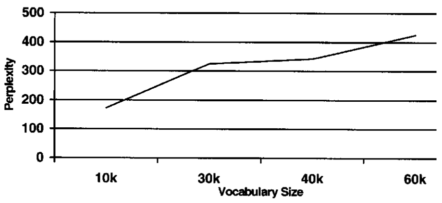
- Combine n-gram models with another method than linear interpolation

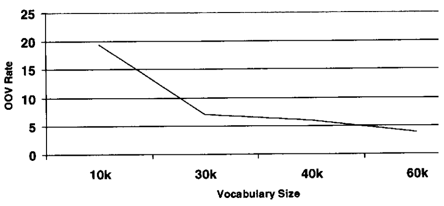
- … …   ?

- Has not offered significant improvement in comparison to linear interpolation

11.6 Practical Issues

- Vocabulary size
  - Conflict confusion rate vs. out-of-vocabulary (OOV) rate
  - For 99.5% English coverage 200 000 word vocabulary is required
  - Larger for inflectional languages
    (e.g. Swedish, German)
  - Combine fixed and personal vocabularies





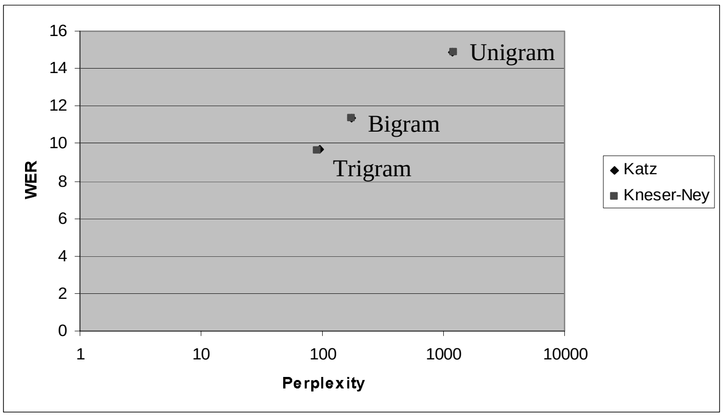
11.6.2 N-gram Pruning

- The n-gram model size becomes easily too large for practical applications
  - Pruning necessary
    - Remove low-count n-grams (those with lowest effect on entropy)
    - The remaining probabilities are unchanged
    - The backoff weights are recomputed
  - Pruning is effective
    - Trigrams can be compressed 25% with no performance degradation
    - Pruned 4-gram model better than unpruned (much larger) trigram model

11.6.3 CFG vs. N-gram Models

- Combine the portability of n-grams with the domain-specificity of CFG
  - Similar to class n-grams but the categories can be CFGs

Relation n-gram length and perplexity vs. word error rate

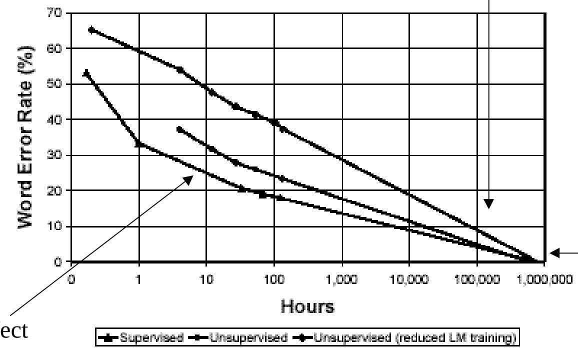


MS Whisper results

How large training data to reach human listening performance?

Extrapolated word error rates for increasing quantities of training data (Moore, Eurospeech 2003)

Heard during a life-time



Human performance

Saturation effect