

Video Lectures

[Help Center](#)

Having trouble viewing lectures? Try changing players. Your current player format is html5. [Change to flash.](#)

> Week 1 - Introduction to Natural Language Processing

- ✓ [Introduction \(Part 1\) \(11:17\)](#)



- ✓ [Introduction \(Part 2\) \(10:28\)](#)



> Week 1 - The Language Modeling Problem

- ✓ [Introduction to the Language Modeling Problem \(Part 1\) \(6:17\)](#)



- ✓ [Introduction to the Language Modeling Problem \(Part 2\) \(7:12\)](#)



- ✓ [Markov Processes \(Part 1\) \(8:56\)](#)



- ✓ [Markov Processes \(Part 2\) \(6:28\)](#)



- ✓ [Trigram Language Models \(9:40\)](#)



- ✓ [Evaluating Language Models: Perplexity \(12:36\)](#)



> Week 1 - Parameter Estimation in Language Models

- ✓ [Linear Interpolation \(Part 1\) \(7:46\)](#)



- ✓ [Linear Interpolation \(Part 2\) \(11:35\)](#)



- ✓ [Discounting Methods \(Part 1\) \(9:26\)](#)



- ✓ [Discounting Methods \(Part 2\) \(3:34\)](#)



> Week 1 - Summary

- ✓ [Summary \(2:31\)](#)



> Week 2 - Tagging Problems, and Hidden Markov Models

- ✓ [The Tagging Problem \(10:01\)](#)



- ✓ [Generative Models for Supervised Learning \(8:57\)](#)



- ✓ [Hidden Markov Models \(HMMs\): Basic Definitions \(12:00\)](#)



- ✓ [Parameter Estimation in HMMs \(13:16\)](#)



- ✓ The Viterbi Algorithm for HMMs (Part 1) (14:07)   
- ✓ The Viterbi Algorithm for HMMs (Part 2) (3:31)   
- ✓ The Viterbi Algorithm for HMMs (Part 3) (7:33)   
- ✓ Summary (1:50)   

› Week 3 - Parsing, and Context-Free Grammars

- ✓ Introduction (0:28)   
- ✓ Introduction to the Parsing Problem (Part 1) (10:37)    
- ✓ Introduction to the Parsing Problem (Part 2) (4:20)   
- ✓ Context-Free Grammars (Part 1) (12:11)   
- ✓ Context-Free Grammars (Part 2) (2:22)   
- ✓ A Simple Grammar for English (Part 1) (10:32)   
- ✓ A Simple Grammar for English (Part 2) (5:30)   
- ✓ A Simple Grammar for English (Part 3) (11:21)   
- ✓ A Simple Grammar for English (Part 4) (2:20)   
- ✓ Examples of Ambiguity (5:56)   

› Week 3 - Probabilistic Context-Free Grammars (PCFGs)

- ✓ Introduction (1:12)     
- ✓ Basics of PCFGs (Part 1) (9:43)   
- ✓ Basics of PCFGs (Part 2) (8:26)   
- ✓ The CKY Parsing Algorithm (Part 1) (7:31)   
- ✓ The CKY Parsing Algorithm (Part 2) (13:22)   
- ✓ The CKY Parsing Algorithm (Part 3) (10:07)   

› Week 4 - Weaknesses of PCFGs

- ✓ Weaknesses of PCFGs (14:59)     

› Week 4 - Lexicalized PCFGs

- ✓ Introduction (00:17)     
- ✓ Lexicalization of a Treebank (10:44)   
- ✓ Lexicalized PCFGs: Basic Definitions (12:40)   

- ✓ Parameter Estimation in Lexicalized PCFGs (Part 1) (5:28)   
- ✓ Parameter Estimation in Lexicalized PCFGs (Part 2) (9:08)   
- ✓ Evaluation of Lexicalized PCFGs (Part 1) (9:32)   
- ✓ Evaluation of Lexicalized PCFGs (Part 2) (11:28)   

› Week 5 - Introduction to Machine Translation (MT)

- ✓ Opening Comments (0:25)    
- ✓ introduction (2:03)   
- ✓ Challenges in MT (8:06)   
- ✓ Classical Approaches to MT (Part 1) (8:02)   
- ✓ Classical Approaches to MT (Part 2) (5:56)   
- ✓ Introduction to Statistical MT (12:31)   

› Week 5 - The IBM Translation Models

- ✓ Introduction (3:24)     
- ✓ IBM Model 1 (Part 1) (13:06)   
- ✓ IBM Model 1 (Part 2) (9:01)   
- ✓ IBM Model 2 (11:27)   
- ✓ The EM Algorithm for IBM Model 2 (Part 1) (5:09)   
- ✓ The EM Algorithm for IBM Model 2 (Part 2) (8:37)   
- ✓ The EM Algorithm for IBM Model 2 (Part 3) (9:28)   
- ✓ The EM Algorithm for IBM Model 2 (Part 4) (4:52)   
- ✓ Summary (1:48)   

› Week 6 - Phrase-based Translation Models

- ✓ Introduction (0:41)     
- ✓ Learning Phrases from Alignments (Part 1) (9:18)   
- ✓ Learning Phrases from Alignments (Part 2) (7:01)   
- ✓ Learning Phrases from Alignments (Part 3) (8:47)   
- ✓ A Sketch of Phrase-based Translation (8:17)   

› Week 6 - Decoding of Phrase-based Translation Models

- [Definition of the Decoding Problem \(Part 1\) \(9:12\)](#)
- [Definition of the Decoding Problem \(Part 2\) \(13:00\)](#)
- [Definition of the Decoding Problem \(Part 3\) \(10:43\)](#)
- [The Decoding Algorithm \(Part 1\) \(14:39\)](#)
- [The Decoding Algorithm \(Part 2\) \(6:23\)](#)
- [The Decoding Algorithm \(Part 3\) \(12:29\)](#)

➤ Week 7 - Log-linear Models

- [Introduction \(0:47\)](#)
- [Two Example Problems \(11:19\)](#)
- [Features in Log-Linear Models \(Part 1\) \(13:56\)](#)
- [Features in Log-Linear Models \(Part 2\) \(10:13\)](#)
- [Definition of Log-linear Models \(Part 1\) \(11:50\)](#)
- [Definition of Log-linear Models \(Part 2\) \(3:45\)](#)
- [Parameter Estimation in Log-linear Models \(Part 1\) \(12:44\)](#)
- [Parameter Estimation in Log-linear Models \(Part 2\) \(4:13\)](#)
- [Smoothing/Regularization in Log-linear Models \(15:12\)](#)

➤ Week 8 - Log-linear Models for Tagging (MEMMs)

- [Introduction \(1:41\)](#)
- [Recap of the Tagging Problem \(3:15\)](#)
- [Independence Assumptions in Log-linear Taggers \(8:32\)](#)
- [Features in Log-Linear Taggers \(13:21\)](#)
- [Parameters in Log-linear Models \(3:59\)](#)
- [The Viterbi Algorithm for Log-linear Taggers \(9:37\)](#)
- [An Example Application \(9:28\)](#)
- [Summary \(2:45\)](#)

➤ Week 8 - Log-Linear Models for History-based Parsing

- [Introduction \(0:47\)](#)
- [Conditional History-based Models \(7:14\)](#)

- ✓ Representing Trees as Decision Sequences (Part 1) (7:23)   
- ✓ Representing Trees as Decision Sequences (Part 2) (10:20)   
- ✓ Features, and Beam Search (12:10)   
- ✓ Summary (1:12)   

› Week 9 - Unsupervised Learning: Brown Clustering

- ✓ Introduction (0:36)   
- ✓ Word Cluster Representations (8:36)     
- ✓ The Brown Clustering Algorithm (Part 1) (11:50)   
- ✓ The Brown Clustering Algorithm (Part 2) (8:30)   
- ✓ The Brown Clustering Algorithm (Part 3) (9:18)   
- ✓ Clusters in NE Recognition (Part 1) (11:33)   
- ✓ Clusters in NE Recognition (Part 2) (7:28)   

› Week 9 - Global Linear Models (GLMs)

- ✓ Introduction (0:30)     
- ✓ Recap of History-based Models (7:11)   
- ✓ Motivation for GLMs (6:34)   
- ✓ Three Components of GLMs (14:39)   
- ✓ GLMs for Parse Reranking (10:36)   
- ✓ Parameter Estimation with the Perceptron Algorithm (6:11)   
- ✓ Summary (3:01)   

› Week 10 - GLMs for Tagging

- ✓ Introduction (1:02)     
- ✓ Recap of GLMs (7:40)   
- ✓ GLMs for Tagging (Part 1) (5:26)   
- ✓ GLMs for Tagging (Part 2) (7:35)   
- ✓ GLMs for Tagging (Part 3) (7:06)   
- ✓ GLMs for Tagging (Part 4) (6:00)   

› Week 10 - GLMs for Dependency Parsing

-  [Introduction \(0:37\)](#)      
-  [The Dependency Parsing Problem \(Part 1\) \(5:21\)](#)   
-  [The Dependency Parsing Problem \(Part 2\) \(13:53\)](#)   
-  [GLMs for Dependency Parsing \(Part 1\) \(11:59\)](#)   
-  [GLMs for Dependency Parsing \(Part 2\) \(8:28\)](#)   
-  [Experiments with GLMs for Dep. Parsing \(5:38\)](#)   
-  [Summary \(2:50\)](#)   

Natural Language Processing

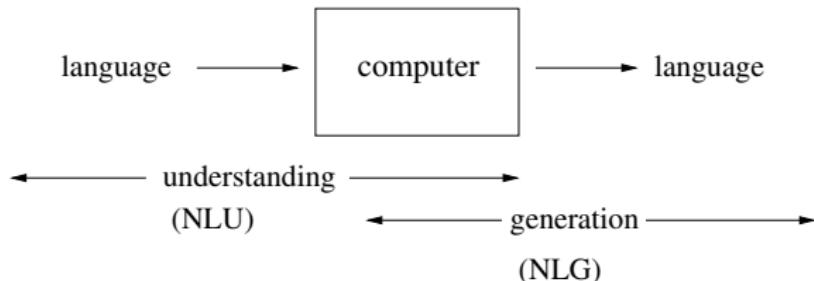
Michael Collins, Columbia University

Overview

- ▶ What is Natural Language Processing (NLP)?
- ▶ Why is NLP hard?
- ▶ What will this course be about?

What is Natural Language Processing?

computers using natural language as input and/or output



Machine Translation: e.g., Google Translation from Arabic

Stock prices retreated in the stock markets again with increasing concern about the circumstances surrounding the credit markets in the world, due mostly to the problems it faces American mortgage lending market, which raised concern among investors.

The index retreated Vuciji / 100 on the London Stock Exchange at the beginning of a percentage point in the dealings of up to 6082 points, while the Nikkei index retreated / 225 Japanese rate of 2.2% to close at the lowest level in eight months.

The American Jones index has lost about 1.6 points Tuesday to reach 13029 points, the Nasdaq index had lost 1.7 of its value.

These declines came despite statements by the American Federal Reserve Bank (Central Bank), in which he said that the process of pumping more funds into capital markets when necessary.

Information Extraction

10TH DEGREE is a full service advertising agency specializing in direct and interactive marketing. Located in Irvine CA, 10TH DEGREE is looking for an Assistant Account Manager to help manage and coordinate interactive marketing initiatives for a marquee automotive account. Experience in online marketing, automotive and/or the advertising field is a plus. Assistant Account Manager Responsibilities Ensures smooth implementation of programs and initiatives Helps manage the delivery of projects and key client deliverables ... Compensation: \$50,000-\$80,000
Hiring Organization: 10TH DEGREE



INDUSTRY	Advertising
POSITION	Assistant Account Manager
LOCATION	Irvine, CA
COMPANY	10TH DEGREE
SALARY	\$50,000-\$80,000

Information Extraction

- ▶ Goal: Map a document collection to structured database
- ▶ Motivation:
 - ▶ Complex searches (“Find me all the jobs in advertising paying at least \$50,000 in Boston”)
 - ▶ Statistical queries (“How has the number of jobs in accounting changed over the years?”)

Text Summarization

Agency Suspends Smallpox Vaccines for People With Heart Disease

Summary from the U.S.

A second health care worker has died of a heart attack (3) after receiving a smallpox vaccination (9) and officials are investigating whether vaccinations are to blame (3) for cardiac problems. (6) The vaccine never has been associated with heart trouble but as a precaution (3) the U.S. centers for Disease Control and Prevention (14) is advising people with a history of heart disease to be vaccinated (3) until further notice. (14) Strom suggested that the Bush administration reassess whether it necessary and safe to continue with its aggressive plan to inoculate millions of health care workers and emergency responders. (1)



Story keywords

vaccine, Heart, Smallpox, vaccinated, Disease

Source articles

1. [Vaccination program in peril after second death](#) (seattletimes.newssource.com, 03/28/2003, 319 words)
2. [Wired News: Smallpox Shots: Proceed With Care](#) (Wired, 03/27/2003, 559 words)
3. [2nd worker dies after smallpox vaccination](#) (suntimes.com, 03/28/2003, 358 words)
4. [2nd worker dies after smallpox vaccine](#) (dallasnews.com, 03/28/2003, 499 words)
5. [Smallpox vaccine is reviewed after second fatal heart attack](#) (boston.com, 03/28/2003, 732 words)
6. [Second Smallpox Vaccine Death Evens](#) (CBS News 03/28/2003, 865 words)

Dialogue Systems

User: I need a flight from Boston to Washington, arriving by 10 pm.

System: What day are you flying on?

User: Tomorrow

System: Returns a list of flights

Basic NLP Problems: Tagging

TAGGING: Strings to Tagged Sequences

a b e e a f h j ⇒ a/C b/D e/C e/C a/D f/C h/D j/C

Example 1: Part-of-speech tagging

Profits/N soared/V at/P Boeing/N Co./N ,/
easily/ADV topping/V forecasts/N on/P Wall/N
Street/N ./.

Example 2: Named Entity Recognition

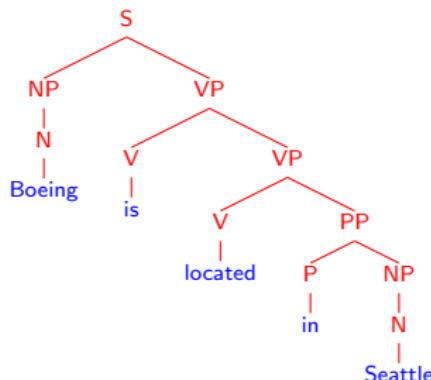
Profits/NA soared/NA at/NA Boeing/SC Co./CC
,/NA easily/NA topping/NA forecasts/NA on/NA
Wall/SL Street/CL ./.

Basic NLP Problems: Parsing

INPUT:

Boeing is located in Seattle.

OUTPUT:



Overview

- ▶ What is Natural Language Processing (NLP)?
- ▶ Why is NLP hard?
- ▶ What will this course be about?

Why is NLP Hard?

[example from L.Lee]

“At last, a computer that understands you like your mother”

Ambiguity

“At last, a computer that understands you like your mother”

1. (*) It understands you as well as your mother understands you
2. It understands (that) you like your mother
3. It understands you as well as it understands your mother

1 and 3: Does this mean well, or poorly?

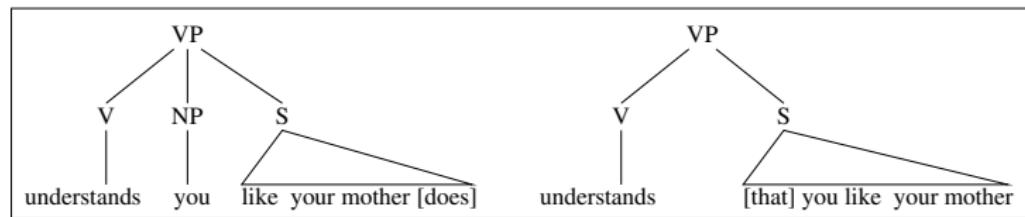
Ambiguity at Many Levels

At the **acoustic** level (speech recognition):

1. “... a computer that understands you **like your** mother”
2. “... a computer that understands you **lie cured** mother”

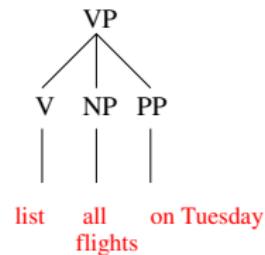
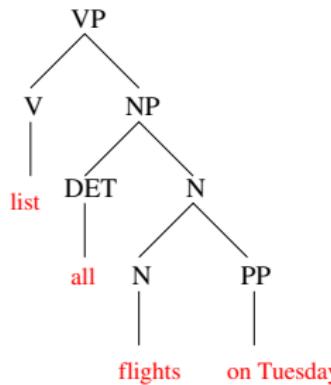
Ambiguity at Many Levels

At the **syntactic** level:



Different structures lead to different interpretations.

More Syntactic Ambiguity



Ambiguity at Many Levels

At the **semantic** (meaning) level:

Two definitions of “mother”

- ▶ a woman who has given birth to a child
- ▶ a stringy slimy substance consisting of yeast cells and bacteria; is added to cider or wine to produce vinegar

This is an instance of **word sense ambiguity**

More Word Sense Ambiguity

At the **semantic** (meaning) level:

- ▶ They put money in the *bank*
= buried in mud?
- ▶ I saw her duck with a telescope

Ambiguity at Many Levels

At the **discourse** (multi-clause) level:

- ▶ Alice says they've built a computer that understands you like your mother
- ▶ But she ...
 - ... doesn't know any details
 - ... doesn't understand me at all

This is an instance of **anaphora**, where she co-refers to some other discourse entity

Overview

- ▶ What is Natural Language Processing (NLP)?
- ▶ Why is NLP hard?
- ▶ What will this course be about?

Course Coverage

- ▶ NLP sub-problems: part-of-speech tagging, parsing, word-sense disambiguation, etc.
- ▶ Machine learning techniques: probabilistic context-free grammars, hidden markov models, estimation/smoothing techniques, the EM algorithm, log-linear models, etc.
- ▶ Applications: information extraction, machine translation, natural language interfaces...

A Syllabus

- ▶ Language modeling, smoothed estimation
- ▶ Tagging, hidden Markov models
- ▶ Statistical parsing
- ▶ Machine translation
- ▶ Log-linear models, discriminative methods
- ▶ Semi-supervised and unsupervised learning for NLP

Prerequisites

- ▶ Basic linear algebra, probability, algorithms
- ▶ Programming skills

Assessment

- ▶ Questions during the lectures
- ▶ 3 homeworks

Books

Comprehensive notes for the course will be provided at
<http://www.cs.columbia.edu/~mcollins>

Additional useful background:

Jurafsky and Martin:

Speech and Language Processing (2nd Edition)

Lecture Questions

1 Language Model 1

1.1 Question (time: 6:17)

Say we have a vocabulary $\mathcal{V} = \{\text{the}\}$ and a constant $N \geq 1$.

For any $x_1 \dots x_n$ such that $x_i \in \mathcal{V}$ for $i = 1 \dots (n-1)$ and $x_n = \text{STOP}$, we define $p(x_1, \dots, x_n) = \begin{cases} \frac{1}{N} & \text{if } n \leq N \\ 0 & \text{otherwise} \end{cases}$

Is this a valid language model?

- (a) True
- (b) False

1.2 Question (time: 6:17)

Say we have a vocabulary $\mathcal{V} = \{\text{the}, \text{dog}\}$.

For any $x_1 \dots x_n$ such that $x_i \in \mathcal{V}$ for $i = 1 \dots (n-1)$ and $x_n = \text{STOP}$, we define $p(x_1, \dots, x_n) = \begin{cases} \frac{1}{2} & \text{if } n = 2 \\ 0 & \text{otherwise} \end{cases}$

Is this a valid language model?

- (a) True
- (b) False

2 Markov Process 1

2.1 Question (time: 2:47)

Consider a Markov process with states $\mathcal{V} = \{0, 1, 2\}$ and length $n = 10$.

How many different sequences can be generated by this process?

- (a) 2^{10}
- (b) 10^2
- (c) 3^{10}
- (d) 10^3

3 Trigram

3.1 Question (time: 5:12)

Say we have a language model with $\mathcal{V} = \{\text{the}, \text{dog}, \text{runs}\}$, and the following parameters:

- $q(\text{the}|*, *) = 1$
- $q(\text{dog}|*, \text{the}) = 0.5$
- $q(\text{STOP}|*, \text{the}) = 0.5$
- $q(\text{runs}|\text{the}, \text{dog}) = 0.5$
- $q(\text{STOP}|\text{the}, \text{dog}) = 0.5$
- $q(\text{STOP}|\text{dog}, \text{runs}) = 1$

How many sentences have non-zero probability under this model?

3.2 Question (time: 7:01)

Consider the following corpus of sentences:

- the dog walks STOP
- walks the dog STOP
- dog walks fast STOP

Let q_{ML} be the maximum-likelihood parameters of a trigram language model trained on this corpus. Which of the following parameters have a value that is both well-defined and non zero?

- (a) $q_{\text{ML}}(\text{walks}|\text{dog}, \text{the})$
- (b) $q_{\text{ML}}(\text{fast}|\text{dog}, \text{the})$
- (c) $q_{\text{ML}}(\text{walks}|*, \text{dog})$
- (d) $q_{\text{ML}}(\text{STOP}|\text{walks}, \text{dog})$
- (e) $q_{\text{ML}}(\text{dog}|\text{walks}, \text{the})$
- (f) $q_{\text{ML}}(\text{walks}|\text{the}, \text{dog})$

4 Perplexity

4.1 Question (time: 6:37)

Define a trigram language model with the following parameters:

- $q(\text{the}|*, *) = 1, q(\text{dog}|*, \text{the}) = 0.5$
- $q(\text{cat}|*, \text{the}) = 0.5, q(\text{walks}|\text{the}, \text{cat}) = 1$
- $q(\text{STOP}|\text{cat}, \text{walks}) = 1, q(\text{runs}|\text{the}, \text{dog}) = 1$
- $q(\text{STOP}|\text{dog}, \text{runs}) = 1$

Now consider a test corpus with the following sentences:

- the dog runs STOP, the cat walks STOP, the dog runs STOP

What is the perplexity of the language model on this test corpus to three decimal places? (Note: use \log_2 for your calculations. Note that the number of words in this corpus, M , is equal to 12)

5 Linear Interpolation 2

5.1 Question (time: 2:21)

We are given the following corpus:

- the green book STOP
- my blue book STOP
- his green house STOP
- book STOP

Assume we compute a language model based on this corpus using linear interpolation with $\lambda_i = 1/3$ for all $i \in \{1, 2, 3\}$.

What is the value of the parameter $q_{LI}(\text{book}|\text{the, green})$ in this model to three decimal places?

(Note: please include STOP words in your unigram model.)

5.2 Question (time: 5:07)

Say that we train a language model using linear interpolation with $\lambda_1 = -0.5$, $\lambda_2 = 0.5$, and $\lambda_3 = 1.0$. Note that these values satisfy the constraint $\sum_i \lambda_i = 1$ but violate the constraint $\lambda_i \geq 0$.

What problems might occur in the resulting language model? Check all that apply.

- (a) we may have a bigram u, v such that $\sum_{w \in \mathcal{V}} q(w|u, v) \neq 1$
- (b) we may have a trigram u, v, w such that $q(w|u, v) < 0$
- (c) we may have a trigram u, v, w such that $q(w|u, v) > 1$

6 Discounting Methods 1

6.1 Question (time: 6:09)

Assume that we are given a corpus with the following properties:

- $\text{Count}(\text{the}) = 70$
- $|\{w : c(\text{the}, w) > 0\}| = 15$, i.e. there are 15 different words that follow "the".

Furthermore assume that discounted counts are defined as $c^*(\text{the}, w) = c(\text{the}, w) - 0.3$.

Under this corpus, what is the missing probability mass, $\alpha(\text{the})$, to three decimal places?

6.2 Question (time: 9:27)

Let's return to a smaller version of our corpus.

- the book STOP
- his house STOP

This time we compute a bigram language model using Katz back-off with $c^*(v, w) = c(v, w) - 0.5$.

What is the value of $q_{\text{BO}}(\text{book}|\text{his})$ estimated from this corpus?

A Answers

- (1.1) a
- (1.2) a
- (2.1) c
- (3.1) 3
- (3.2) c e f
- (4.1) 1.189
- (5.1) 0.571
- (5.2) b c
- (6.1) 0.064
- (6.2) 0.1

Lecture Questions

1 The Tagging Problem

1.1 Question (time: 6:15)

Say we are given the following sentence with named-entity boundaries

- (Person Jane Smith) lives in (Location England)

If we encode these boundaries as a tag sequence, what is the tag for the word "Jane"?

- (a) CP
- (b) SP
- (c) NA
- (d) P

1.2 Question (time: 6:15)

Say we are given the following sentence

- Profits are topping all estimates

We also are told that

- "Profits" has 2 possible tags: N and V
- "are" has 1 possible tag: V
- "topping" has 3 possible tags: N, ADJ, and V
- "all" has 3 possible tags: DT, ADV, and N
- "estimates" has 2 possible tags: N and V

How many tag sequences are possible for this sentence?

2 HMMs

2.1 Question (time: 7:12)

Say we are given a tagset $\mathcal{S} = \{D, N\}$, a vocabulary $\mathcal{V} = \{\text{the}, \text{dog}\}$, and a hidden Markov model. The HMM has transition parameters

- $q(D|*, *) = 1$
- $q(N|*, D) = 1$
- $q(\text{STOP}|D, N) = 1$
- $q(s|u, v) = 0$ for all other q params

and emission parameters

- $e(\text{the}|D) = 0.9$
- $e(\text{dog}|D) = 0.1$
- $e(\text{dog}|N) = 1$

Under this model, how many pairs of sequences $x_1 \dots x_n, y_1 \dots y_{n+1}$ satisfy $p(x_1, \dots, x_n, y_1, \dots, y_{n+1}) > 0$?

2.2 Question (time: 9:20)

Say we have a tag set $\mathcal{S} = \{D, N, V\}$, a vocabulary $\mathcal{V} = \{\text{the}, \text{cat}, \text{drinks}, \text{milk}, \text{dog}\}$, and a hidden Markov model with parameters $q(s|u, v) = 1/4$ for all s, u, v and $e(x|s) = 1/5$ for all tags s and words x .

What is the value of $p(\text{the cat drinks milk, D N V N STOP})$ under this model?

- (a) $(1/4)^4(1/5)^5$
- (b) $(1/4)^5(1/5)^5$
- (c) $(1/4)^5(1/5)^4$
- (d) $(1/4)^4(1/5)^4$

2.3 Question (time: 12:00)

Which of the following is a suitable definition for $p(x_1 \dots x_n, y_1 \dots y_{n+1})$ under a $\lfloor b \rfloor$ -gram $\rfloor b \rfloor$ -hidden Markov model?

- (a) $\prod_{i=1}^n q(y_i|y_{i-1}) \prod_{i=1}^n e(x_i|y_i)$
- (b) $\prod_{i=1}^{n+1} q(y_i|y_{i-1}) \prod_{i=1}^n e(x_i|y_i)$
- (c) $\prod_{i=1}^{n+1} q(y_i|y_{i-1}) \prod_{i=1}^n e(x_i, x_{i-1}|y_i)$

3 Estimation

3.1 Question (time: 5:02)

Consider the following training corpus of tagged sentences

- the dog barks → D N V STOP
- the cat sings → D N V STOP

Say we compute the maximum-likelihood estimates of a trigram hidden Markov model from this data. What is the value for the parameter $e(\text{cat}|N)$ of this HMM?

3.2 Question (time: 5:02)

Consider the following training corpus of tagged sentences

- the dog barks → D N V STOP
- the cat sings → D N V STOP

Say we estimate the parameters for a hidden Markov model from this data using linear interpolation with $\lambda_i = 1/3$ for $i = 1 \dots 3$.

What is the value of the parameter $q(\text{STOP}|N, V)$ under this model?

4 Viterbi 1

4.1 Question (time: 12:00)

We are given a hidden Markov model with transition parameters

- $q(D|*, *) = 1, q(N|*, D) = 1$
- $q(V|D, N) = 1, q(\text{STOP}|N, V) = 1$

and emission parameters

- $e(\text{the}|D) = 0.8, e(\text{dog}|D) = 0.2$
- $e(\text{dog}|N) = 0.8, e(\text{the}|N) = 0.2$
- $e(\text{barks}|V) = 1.0$

Say we have the sentence

- the dog barks

What is the value of $\pi(3, N, V)$?

5 Viterbi 2

5.1 Question (time: 3:31)

Say we are given a tag set $\mathcal{S} = \{D, N, V, P\}$ and a hidden Markov model with parameters

- $q(D|N, P) = 0.4$
- $q(D|w, P) = 0$ for $w \neq N$
- $e(\text{the}|D) = 0.6$

We are also given the sentence

- Ella walks to the red house

Say the dynamic programming table for this sentence has the following entries

- $\pi(3, D, P) = 0.1, \pi(3, N, P) = 0.2$
- $\pi(3, V, P) = 0.01, \pi(3, P, P) = 0.5$

What will be the value of $\pi(4, P, D)$?

A Answers

- (1.1) b
- (1.2) 36
- (2.1) 2
- (2.2) c
- (2.3) b
- (3.1) 0.5
- (3.2) 0.75
- (4.1) 0.64
- (5.1) 0.048

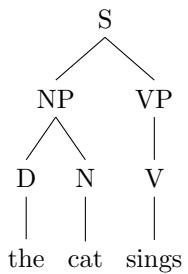
1 Basics of PCFGs (Part 1)

1.1 Question (time: 6:42, slide: 4)

Consider the following PCFG

$$\begin{array}{llll} q(S \rightarrow NP VP) = 0.9 & q(S \rightarrow NP) = 0.1 & q(NP \rightarrow D N) = 1 & q(VP \rightarrow V) = 1 \\ q(D \rightarrow \text{the}) = 0.8 & q(D \rightarrow a) = 0.2 & q(N \rightarrow \text{cat}) = 0.5 & q(N \rightarrow \text{dog}) = 0.5 \\ q(V \rightarrow \text{sings}) = 1 & & & \end{array}$$

Say we are given a parse tree



What is the probability of this parse?

1.2 Question (time: 9:42, slide: 5)

Consider the following PCFG with start symbol "S"

$$\begin{array}{llll} q(S \rightarrow NP VP) = 1.0 & q(VP \rightarrow VP PP) = 0.9 & q(VP \rightarrow V NP) = 0.1 \\ q(NP \rightarrow NP PP) = 0.5 & q(NP \rightarrow N) = 0.5 & q(PP \rightarrow P NP) = 1.0 \\ q(N \rightarrow \text{Ted}) = 0.2 & q(N \rightarrow \text{Jill}) = 0.2 & q(N \rightarrow \text{town}) = 0.6 \\ q(V \rightarrow \text{saw}) = 1.0 & q(P \rightarrow \text{in}) = 1.0 & \end{array}$$

Say we are given the sentence

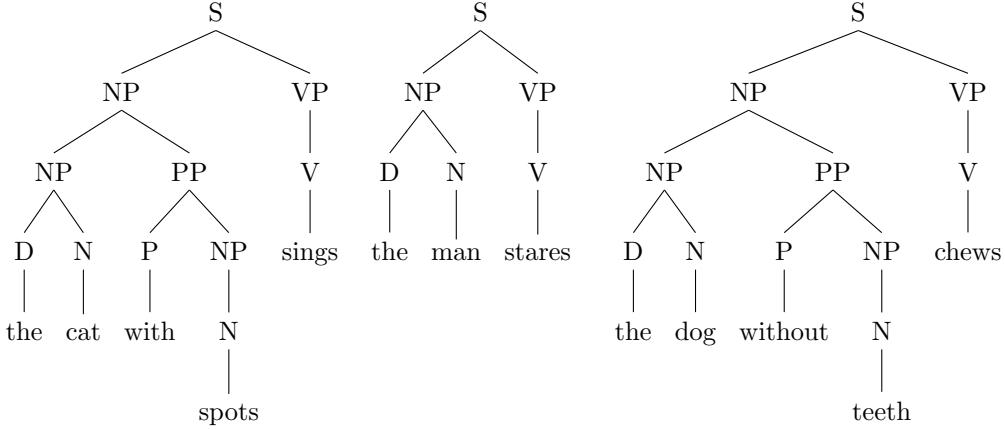
- Ted saw Jill in town.

What is the highest probability for any parse tree under this PCFG?

2 Basics of PCFGs (Part 2)

2.1 Question (time: 4:06, slide: 7)

Consider the following treebank



If we compute the maximum-likelihood PCFG from this treebank, what is the value of $q_{ML}(NP \rightarrow NP\ PP)$ to three decimal places?

3 The CKY Parsing Algorithm (Part 1)

3.1 Question (time: 7:31, slide: 10)

Consider the following PCFG not in Chomsky normal form

$$\begin{aligned}
 q(S \rightarrow N\ VP) &= 0.5 & q(VP \rightarrow V\ N) &= 0.6 \\
 q(VP \rightarrow V\ N\ PP) &= 0.4 & q(PP \rightarrow P\ N) &= 1.0 \\
 q(N \rightarrow \text{dog}) &= 1.0 & q(P \rightarrow \text{in}) &= 1.0 \\
 q(V \rightarrow \text{saw}) &= 1.0
 \end{aligned}$$

We then convert to Chomsky normal form by changing the rules to

$$\begin{aligned}
 S &\rightarrow N\ VP & VP &\rightarrow V\ N \\
 VP &\rightarrow V\ N/PP & N/PP &\rightarrow N\ PP \\
 PP &\rightarrow P\ N & N &\rightarrow \text{dog} \\
 P &\rightarrow \text{in} & V &\rightarrow \text{saw}
 \end{aligned}$$

If we want the new PCFG to be equivalent to the old PCFG and we are told $q(N/PP \rightarrow N\ PP) = 1.0$, what should be the value of $q(VP \rightarrow V\ N/P)$?

4 The CKY Parsing Algorithm (Part 2)

4.1 Question (time: 5:05, slide: 12)

Consider the sentence from the previous slide

- the dog saw the man with the telescope

Say we are given a PCFG with the rules

- $q(NP \rightarrow D\ N) = 0.5$
- $q(NP \rightarrow N) = 0.5$

- $q(D \rightarrow \text{the}) = 1.0$
- $q(N \rightarrow \text{dog}) = 0.1$
- $q(N \rightarrow \text{cat}) = 0.9$

What is the value for $\pi(1, 2, NP)$?

4.2 Question (time: 13:21, slide: 14)

Consider the sentence from the previous slide

- the dog saw the man with the telescope

Assume that we have π values such that

$$\begin{aligned}\pi(3, 3, V) \times \pi(4, 8, NP) &= 0.01 & \pi(3, 5, VP) \times \pi(6, 8, PP) &= 0.1 \\ \pi(3, 6, VP) \times \pi(7, 8, NP) &= 0.1 & \pi(3, 7, VP) \times \pi(8, 8, N) &= 0.01\end{aligned}$$

- For all other values of $s \in \{3 \dots 7\}$ and $X \in N, Y \in N$, assume that $\pi(3, s, Y) \times \pi(s + 1, 8, X) = 0$

Also assume that the PCFG has the following parameters

$$\begin{aligned}q(VP \rightarrow V NP) &= 0.2 & q(VP \rightarrow VP PP) &= 0.5 \\ q(VP \rightarrow VP NP) &= 0.2 & q(VP \rightarrow VP N) &= 0.1\end{aligned}$$

What is the value for $\pi(3, 8, VP)$?

A Answers

- 0.36

The answer is 0.36, which is the product of the rules used in the tree.

- 0.00027

There are two possible parse trees for this sentence, one attaches the preposition to the verb "saw" and the other attaches it to the noun "Jill". The verbal attachment has higher probability which is 0.00027.

- 0.286

The non-terminal NP is seen seven times. Two times it has NP PP as children, so the correct value rounds to 0.286.

- 0.4

The rule $VP \rightarrow V N/PP$ is the CNF form of the rule $VP \rightarrow V N PP$, so it should have the same value as the original rule, 0.4, after the conversion.

- 0.05

There is only one parse of the first two words of the sentence. Its value is $q(D \rightarrow \text{the}) \times q(N \rightarrow \text{dog}) \times q(NP \rightarrow D N) = 0.05$.

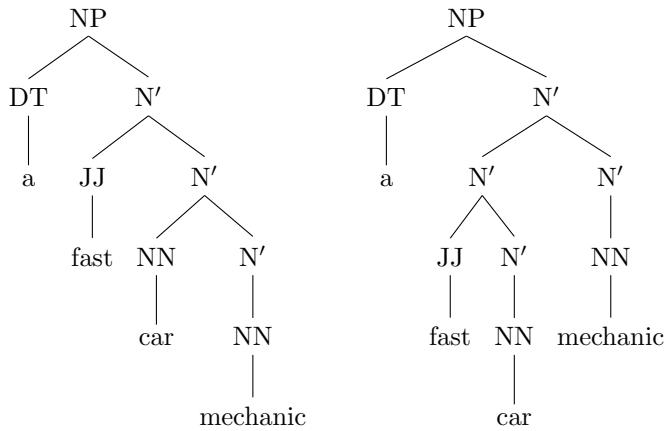
- 0.05

The highest scoring of the four rules is $VP \rightarrow VP PP$. Its score is $\pi(3, 5, VP) \times \pi(6, 8, PP) \times q(VP \rightarrow VP PP) = 0.05$.

1 Weaknesses of PCFGs

1.1 Question (time: 9:15, slide: 5)

Consider the following two parse trees



Which of the following statements is true?

- (a) The two parse trees receive the same probability under any PCFG.
- (b) The first parse tree receives higher probability if $q(N' \rightarrow NN\ N') > q(N' \rightarrow N'\ N')$.
- (c) Neither of the above.

A Answers

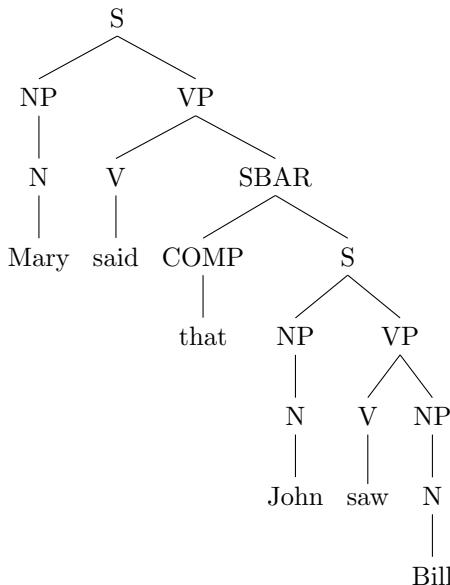
- (c)

Let the probability of the first tree be $q(N' \rightarrow NN)N' \times c$ for some value c , then the probability of the second tree is $q(N' \rightarrow N'N') \times q(N' \rightarrow N) \times c$. Condition 1 is definitely not true, and condition 2 is not enough to ensure that the first tree has higher probability.

1 Lexicalization of a Treebank

1.1 Question (time: 10:44, slide: 8)

Say we have the sentence “Mary said that John saw Bill” with the parse tree



We are also given the head rules (where * indicates the head)

$$S \rightarrow NP \mathbf{VP^*} \quad NP \rightarrow N^* \quad VP \rightarrow V^* NP$$

$$VP \rightarrow V^* SBAR \quad SBAR \rightarrow COMP^* S$$

List the head words (separated by a space) of the following non-terminals

1. the “SBAR”
2. the “S” spanning “Mary … Bill”
3. the “VP” spanning “said… Bill”

2 Lexicalized PCFGs

2.1 Question (time: 4:54, slide: 11)

Say we are constructing a lexicalized PCFG with $|N| = 10$ and $|\Sigma| = 1000$. How many possible rule pairs are there of the form

$$X(h) \rightarrow_1 Y_1(h)Y_2(w)$$

$$X(h) \rightarrow_2 Y_1(w)Y_2(h)$$

where $X, Y_1, Y_2 \in N$ and $h, w \in \Sigma$?

- (a) $10^3 \times 1000^2$
- (b) $10^2 \times 1000^3$
- (c) 10×1000
- (d) 10^3

2.2 Question (time: 5:59, slide: 12)

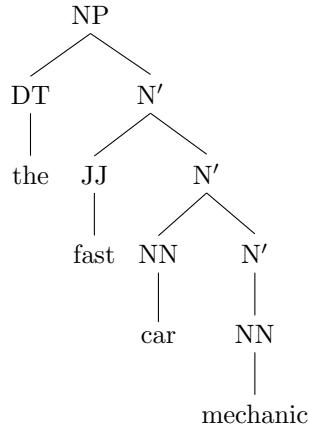
We are given a lexicalized grammar with some valid rules and some invalid rules. Which of the following rules are **valid**?

- (a) DT(the) \rightarrow a
- (b) DT(a) \rightarrow a
- (c) SBAR(that) \rightarrow_1 COMP(that) S(was)
- (d) SBAR(was) \rightarrow_2 COMP(that) S(was)
- (e) SBAR(was) \rightarrow_1 COMP(that) S(was)
- (f) PP(in) \rightarrow_1 IN(of) NP(company)

3 Parameter Estimation in Lexicalized PCFGs (Part 1)

3.1 Question (time: 2:57, slide: 16)

Say we have the sentence “the fast car mechanic” and the parse tree



We are also given the following head rules (where * indicates the head)

$$NP \rightarrow DT \mathbf{N'^*} \quad N' \rightarrow JJ \mathbf{N'^*} \quad N' \rightarrow NN \mathbf{N'^*} \quad N' \rightarrow \mathbf{NN^*}$$

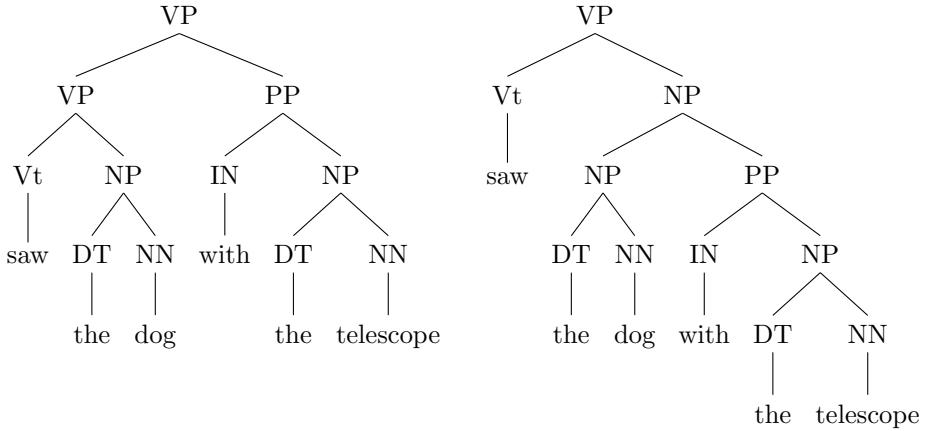
Which of the following parameters are used in calculating the probability of this parse tree?

- (a) $q(N' \text{ (mechanic)} \rightarrow_2 JJ(\text{fast}) N' \text{ (mechanic)})$
- (b) $q(N' \text{ (fast)} \rightarrow_1 JJ(\text{fast}) N' \text{ (mechanic)})$
- (c) $q(N' \text{ (mechanic)} \rightarrow_2 NN(\text{car}) N' \text{ (mechanic)})$
- (d) $q(N' \text{ (car)} \rightarrow_1 NN(\text{car}) N' \text{ (mechanic)})$
- (e) $q(NP(\text{the}) \rightarrow_1 DT(\text{the}) N' \text{ (mechanic)})$
- (f) $q(NP(\text{mechanic}) \rightarrow_2 DT(\text{the}) N' \text{ (mechanic)})$

4 Evaluation of Lexicalized PCFGs (Part 1)

4.1 Question (time: 5:46, slide: 22)

Say we have the phrase “saw the dog with the telescope” and we are given the gold parse tree (left) and a test parse tree (right)

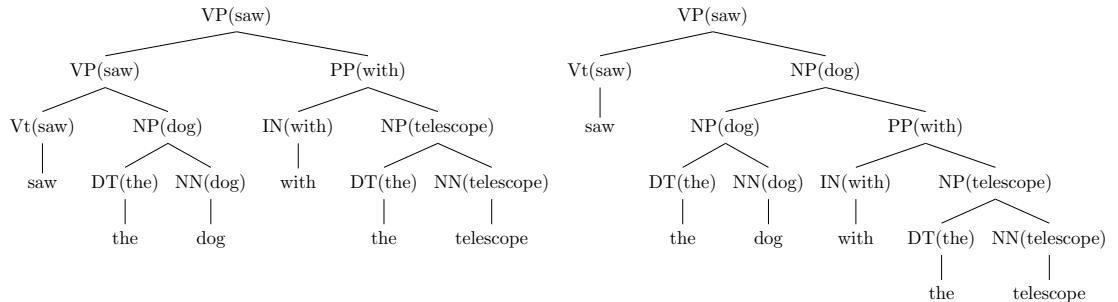


What is the **precision** of this test parse tree?

5 Evaluation of Lexicalized PCFGs (Part 2)

5.1 Question (time: 6:35, slide: 25)

Say we have the fragment “saw the dog with the telescope” and we are given the gold parse tree (left) and a test parse tree (right)



What is the **dependency accuracy** of this test parse tree (to three decimal places)?

A Answers

- that said said

Trace the head word up from the bottom of the tree to the indicated non-terminal.

- (a)

There are 10 non-terminals, so there are 10^3 possible unlexicalized rules of the form $X \rightarrow Y_1 Y_2$. Each one of these rules can have any word as its head h and any other word as w yielding 1000^2 lexicalized variants. The final grammar has $10^3 \times 1000^2$ rules of this form.

- (b) (c) (d)

Invalid rules have a head word that (1) does not match any right-hand side head word, (2) matches the wrong right-hand side word, i.e. SBAR(was) \rightarrow_1 COMP(that) S(was).

- (a) (c) (f)

Note that these rules always have the right-most noun as the head word. The incorrect parameters have a non-noun as head or the left noun in the case of car.

- 0.8

The answer is 0.8. There are 5 constituents in both parse trees, and 4 of them are in both trees. The incorrect constituent is (NP, 2, 6), and the missing constituent is (VP, 2, 4).

- 0.833

The answer is 0.833. There are 6 words in this sentence and the test parse gets 5 of the dependencies correct. The one incorrect dependency is “with” modifying “dog” instead of “with” modifying “saw”.

1 IBM Model 1 (Part 1)

1.1 Question (time: 5:32, slide: 6)

Say we have the sentence pair, $e = \text{the dog saw the cat}$ and $f = \text{adog asaw acat}$.

How many possible alignments are there between the sentences?

- (a) 5^3
- (b) 4^5
- (c) 6^3
- (d) 3^6

2 IBM Model 1 (Part 2)

2.1 Question (time: 1:42, slide: 10)

Consider the following sentence pair

- $e = \text{the dog barks}$
- $f = \text{abarks adog athe}$

and say we have the alignment $a_1 = 3, a_2 = 2, a_3 = 1$.

What is the value of $p(a|e, m)$ for this example under IBM Model 1?

2.2 Question (time: 4:20, slide: 11)

Consider the following sentence pair

- $e = \text{the dog barks}$
- $f = \text{abarks adog athe}$

Say we are given the alignment $a_1 = 3, a_2 = 2, a_3 = 1$ and the parameters

$$\begin{array}{lll} t(\text{athe}|\text{the}) = 0.5 & t(\text{abarks}|\text{the}) = 0.5 & t(\text{adog}|\text{the}) = 0.0 \\ t(\text{athe}|\text{dog}) = 0.8 & t(\text{abarks}|\text{dog}) = 0.0 & t(\text{adog}|\text{dog}) = 0.2 \\ t(\text{athe}|\text{barks}) = 0.0 & t(\text{abarks}|\text{barks}) = 0.1 & t(\text{adog}|\text{barks}) = 0.9 \end{array}$$

What is the value of $p(f|a, e, m)$ for this example under IBM Model 1?

3 IBM Model 2

3.1 Question (time: 4:50, slide: 17)

Consider the sentence pair

- $e = \text{the dog barks}$
- $f = \text{abarks adog athe}$

Say we have the alignment $a_1 = 3, a_2 = 2, a_3 = 1$ and the parameters

$$q(0|1, 3, 3) = 0.0 \quad q(1|1, 3, 3) = 0.0 \quad q(2|1, 3, 3) = 0.4 \quad q(3|1, 3, 3) = 0.6$$

$$q(0|2, 3, 3) = 0.0 \quad q(1|2, 3, 3) = 0.1 \quad q(2|2, 3, 3) = 0.9 \quad q(3|2, 3, 3) = 0.0$$

$$q(0|3, 3, 3) = 0.0 \quad q(1|3, 3, 3) = 0.2 \quad q(2|3, 3, 3) = 0.4 \quad q(3|3, 3, 3) = 0.4$$

What is the value of $p(a|e, m)$ for this example under IBM Model 2?

3.2 Question (time: 11:27, slide: 20)

Consider the sentence pair

- $e = \text{the dog barks}$
- $f = \text{abarks adog athe}$

and say we have the parameters

$$t(\text{abarks}|\text{the}) = 0.2 \quad t(\text{abarks}|\text{dog}) = 0.5 \quad t(\text{abarks}|\text{barks}) = 0.2 \quad t(\text{abarks}|\text{NULL}) = 0.1$$

$$q(3|1, 3, 3) = 0.3 \quad q(2|1, 3, 3) = 0.2 \quad q(1|1, 3, 3) = 0.4 \quad q(0|1, 3, 3) = 0.1$$

Define $a_1^* = \arg \max_{a \in \{0, \dots, l\}} q(a|1, 3, 3) \times t(f_1|e_a)$. What is the value of a_1^* for this example?

4 The EM Algorithm for IBM Model 2 (Part 1)

4.1 Question (time: 5:09, slide: 23)

Consider the sentence pair from the last slide

- $e = \text{And the program has been implemented}$
- $f = \text{Le programme a ete mis en application}$

and say we are given the alignment $a = \langle 2, 3, 1, 0, 6, 6, 6 \rangle$.

List the maximum-likelihood estimates (to three decimal places, separated by a space) for the parameters $t_{ML}(\text{Le}|\text{the})$, $t_{ML}(\text{Le}|\text{And})$, $t_{ML}(\text{mis}|\text{implemented})$, $t_{ML}(\text{en}|\text{implemented})$, $t_{ML}(\text{application}|\text{implemented})$, and $t_{ML}(\text{application}|\text{NULL})$.

5 The EM Algorithm for IBM Model 2 (Part 3)

5.1 Question (time: 8:02, slide: 28)

Consider the sentence pair

- $e^{(1)} = \text{the dog barks}$
- $f^{(1)} = \text{abarks adog athe}$

and say we have the following parameters

$$\begin{array}{llll} t(\text{abarks}|\text{the}) = 0.1 & t(\text{abarks}|\text{dog}) = 0.4 & t(\text{abarks}|\text{barks}) = 0.3 & t(\text{abarks}|\text{NULL}) = 0.2 \\ q(3|1, 3, 3) = 0.1 & q(2|1, 3, 3) = 0.4 & q(1|1, 3, 3) = 0.4 & q(0|1, 3, 3) = 0.1 \end{array}$$

What is the value of $\delta(1, 1, 3)$?

A Answers

- (c)

There are $l = 5$ English words plus the NULL word and $m = 3$ French words. The total number of alignments is $(l + 1)^m = 6^3$.

- 0.01563

Under IBM Model 1, all alignments are equally likely. The alignment probability is $p(a|e, m) = 1/(l + 1)^m = 1/64$.

- 0.01

The answer is 0.01. The probability calculation is $p(f|a, e, m) = \prod_{j=1}^m t(f_j|e_{a_j}) = t(\text{abarks}|\text{barks}) \times t(\text{adog}|\text{dog}) \times t(\text{athe}|\text{the}) = 0.01$.

- 0.108

The answer is 0.108. The probability calculation is $p(a|e, m) = \prod_{j=1}^m q(a_j|j, l, m) = q(3|1, 3, 3) \times t(2|2, 3, 3) \times t(1|3, 3, 3) = 0.108$.

- 2

There are four possibilities in the maximization,

- $t(\text{abarks}|\text{NULL}) \times q(0|1, 3, 3) = 0.01$
- $t(\text{abarks}|\text{the}) \times q(1|1, 3, 3) = 0.08$
- $t(\text{abarks}|\text{dog}) \times q(2|1, 3, 3) = 0.10$
- $t(\text{abarks}|\text{barks}) \times q(3|1, 3, 3) = 0.06$

The best choice is $a_1 = 2$.

- 1 0 0.333 0.333 0.333 0

The maximum-likelihood estimation is $t_{\text{ML}}(f|e) = \text{Count}(e, f)/\text{Count}(e)$ where $\text{Count}(e, f)$ is based on the word alignment pairs.

- 0.12

The formula for calculating δ values is $\delta(k, i, j) = \frac{q(j|i, l_k, m_k) \times t(f_i^{(k)}|e_j^{(k)})}{\sum_{j=0}^{l_k} q(j|i, l_k, m_k) \times t(f_i^{(k)}|e_j^{(k)})}$. The numerator is $q(3|1, 3, 3) \times t(\text{abarks}|\text{barks}) = 0.03$ and the denominator is $\sum_{j=0}^{l_k} q(j|1, 3, 3) \times t(\text{abarks}|e_j^{(k)}) = 0.02 + 0.04 + 0.16 + 0.03 = 0.25$. The final value is 0.12.

1 Learning Phrases from Alignments (Part 1)

1.1 Question (time: 9:18, slide: 7)

Consider the sentence pair

e = Mary did not slap the green witch

f = Maria no daba una bofetada a la bruja verde

With the alignment matrix

	Maria	no	daba	una	b of'	a	la	bruja	verde
Mary	●								
did						●			
not		●							
slap			●	●	●				
the						●			
green								●	
witch							●		

Under this alignment, what is the value for a_3 ?

2 Learning Phrases from Alignments (Part 2)

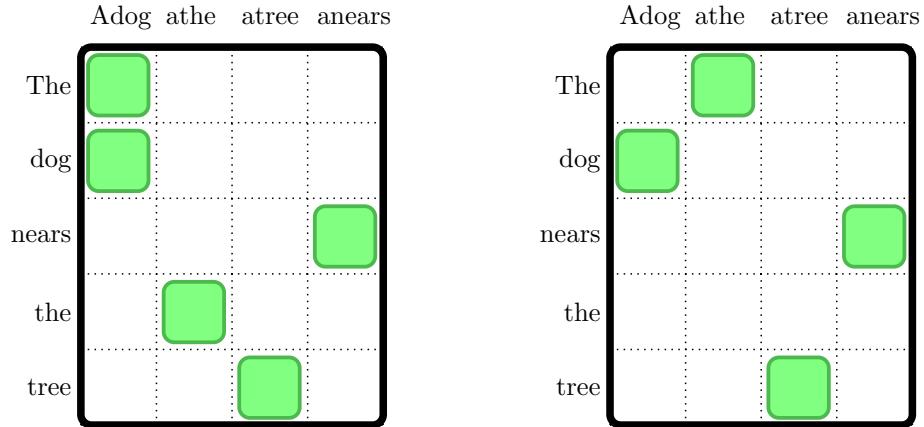
2.1 Question (time: 5:24, slide: 10)

Consider the sentence pairs

• e = The dog nears the tree

• f = adog athe atree anears

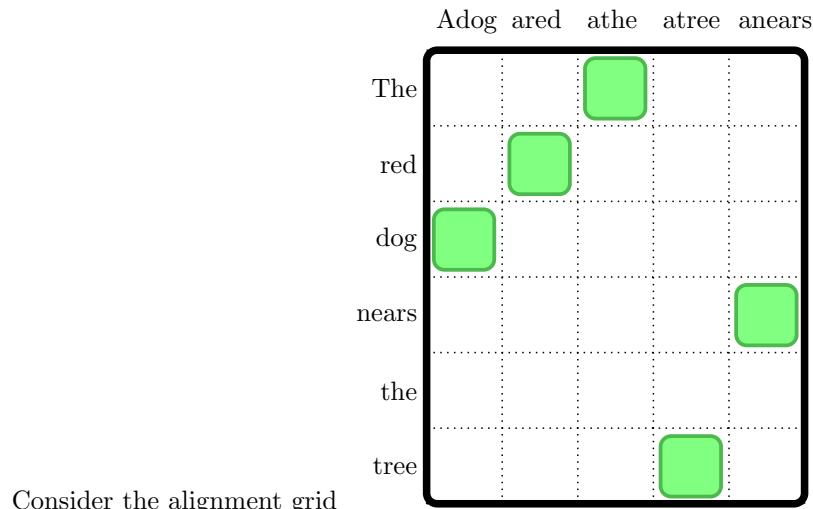
Say we have the following $p(e|f)$ and $p(f|e)$ alignment grids



How many points are in the intersection of the two alignments?

3 Learning Phrases from Alignments (Part 3)

3.1 Question (time: 6:13, slide: 13)



Consider the alignment grid

Which of the following phrase pairs can be extracted from this alignment?

- (a) (Adog ared athe, the red dog)
- (b) (Adog ared, red dog)
- (c) (nears tree, atree anear)
- (d) (the tree, athe atree)

(e) (nears the tree, atree anears)

(f) (dog nears the tree, atree anears)

A Answers

- 4

The answer is 4. Look for the point in the third column of the grid. It is in the fourth row.

- 3

The intersection consists of locations where there is a point in both alignment grids. There are three such points.

- (a) (b) (e)

The incorrect options fail because the foreign word is aligned to a word outside the phrase e.g. "athe atree" or the English words are not contiguous e.g. "nears" and "tree".

1 Definition of the Decoding Problem (Part 1)

1.1 Question (time: 9:12, slide: 4)

Consider the sentence "wir mussen auch diese kritik ernst nehmen" and the phrasal lexicon

- (wir mussen , we must)
- (wir mussen auch, we must also)
- (ernst, seriously)
- (diese kritik, this criticism)

Which of the following phrases is in \mathcal{P} ?

- (a) (1, 2, we must)
- (b) (1, 1, we)
- (c) (4, 5, this criticism)
- (d) (6, 6, seriously)
- (e) (1, 2, we must also)
- (f) (4, 6, this criticism seriously)

2 Definition of the Decoding Problem (Part 2)

2.1 Question (time: 13:00, slide: 7)

Consider the source sentence "wir mussen auch diese kritik ernst nehmen" If we have distortion limit $d = 4$, which of the following derivations are valid?

- (a) $y = (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$
- (b) $y = (6, 6, \text{seriously}), (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism})$
- (c) $y = (1, 3, \text{we must also}), (4, 5, \text{this criticism}), (7, 7, \text{take}), (6, 6, \text{seriously})$
- (d) $y = (1, 3, \text{we must also}), (4, 5, \text{this criticism}), (6, 6, \text{seriously}), (7, 7, \text{take})$

3 The Decoding Algorithm (Part 2)

3.1 Question (time: 3:03, slide: 13)

Consider the source sentence "wir mussen auch diese kritik ernst nehmen" with the distortion limit $d = 4$.

Say we are currently at state $q = (\text{also}, \text{seriously}, 0110010, 6, 0.1)$. Assuming all of the following phrases are in \mathcal{P} , which are also in $ph(q)$?

- (a) (1, 2, we must)
- (b) (7, 7, take)
- (c) (4, 5, this criticism)
- (d) (6, 6, seriously)
- (e) (1, 1, we)
- (f) (4, 4, this)

A Answers

- (a) (c) (d)

The incorrect phrases either are not in the phrasal lexicon, combine multiple phrase pairs, or are aligned to the incorrect source sentence indices.

- (a) (c) (d)

The incorrect derivations either have $|t(p_k) + 1s(p_{k+1})| > 4$ or $|1s(p_1)| > 4$ which violates the distortion limit.

- (b) (c) (f)

This incorrect phrases either violate the distortion limit d or translate source words multiple times.

1 Features in Log-Linear Models (Part 1)

1.1 Question (time: 10:48, slide: 14)

Say we have a set of words \mathcal{Y} with $|\mathcal{Y}| = 1000$, word history $x = \langle w_1 \dots w_{i-1} \rangle$, and let the size of our alphabet be 26 letters.

Define the following features for each word u and suffix s of length 4:

$$f_{N(u,s)}(x, y) = \begin{cases} 1 & \text{if } y = u \text{ and } w_{i-1} \text{ ends with } s \\ 0 & \text{otherwise} \end{cases} \quad \text{where } N \text{ maps word/suffix pairs to integers.}$$

How many features are there in this model?

- (a) 1000×26^4
- (b) 26^4
- (c) 1000
- (d) 1000×26

1.2 Question (time: 13:56, slide: 15)

Say we have a set of words \mathcal{Y} with $|\mathcal{Y}| = 1000$, word history $x = \langle w_1 \dots w_{i-1} \rangle$, and let the size of our alphabet be 26 letters.

Define the following features for each word u and suffix s of length 4:

$$f_{N(u,s)}(x, y) = \begin{cases} 1 & \text{if } y = u \text{ and } w_{i-1} \text{ ends with } s \\ 0 & \text{otherwise} \end{cases} \quad \text{where } N \text{ maps word/suffix pairs to integers.}$$

Now consider a training set of size $n = 10000$. What is a good upper bound on the number of features introduced for this training set?

- (a) 10000
- (b) 10000×26^4
- (c) 1000×26^4
- (d) 1000

2 Features in Log-Linear Models (Part 2)

2.1 Question (time: 7:45, slide: 17)

Consider the label set \mathcal{Y} consisting of part-of-speech tags with $|\mathcal{Y}| = 50$, and the set \mathcal{X} consisting of histories of the form $\langle t_1, \dots, t_i, w_1 \dots w_n, i \rangle$. Also let the number of words in the vocabulary be 1000.

Say that our features are of the form

$$f_{N(u,t)}(x, y) = \begin{cases} 1 & \text{if } w_i = u \text{ and } y = t \\ 0 & \text{otherwise} \end{cases} \quad \text{where } N \text{ maps a word/tag pair}$$

to an integer.

How many possible features are there in this model?

2.2 Question (time: 7:46, slide: 17)

Consider the label set \mathcal{Y} consisting of part-of-speech tags with $|\mathcal{Y}| = 50$, and the set \mathcal{X} consisting of histories of the form $\langle t_1, \dots, t_i, w_1 \dots w_n, i \rangle$. Also let the number of words in the vocabulary be 1000.

Say that our features are of the form

$$f_{N(u,t)}(x, y) = \begin{cases} 1 & \text{if } w_i = u \text{ and } y = t \\ 0 & \text{otherwise} \end{cases} \quad \text{where } N \text{ maps a word/tag pair}$$

to an integer.

Our training set consists of the following word/tag pairs

- the/DT man/NN fishes/V for/ADP the/DT fishes/NN

How many different features are introduced in this training set?

3 Definition of Log-linear Models (Part 1)

3.1 Question (time: 4:49, slide: 21)

Consider the label set $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{hat}\}$ with three simple features

- $f_1(x, y) = \begin{cases} 1 & \text{if } x = \text{the} \text{ and } y \text{ ends with at} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } x = \text{the} \text{ and } y \text{ starts with c} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if } x = \text{the} \text{ and } y \text{ has second letter o} \\ 0 & \text{otherwise} \end{cases}$

Say we are given the weight vector $v = \langle 1, 2, 3 \rangle$. What is the score of the (x, y) pair (the, cat) ?

4 Definition of Log-linear Models (Part 2)

4.1 Question (time: 0:54, slide: 22)

Consider the label set $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{hat}, \text{cot}\}$ with three simple features

- $f_1(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ ends with at} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ starts with c} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ has second letter o} \\ 0 & \text{otherwise} \end{cases}$

Say we are given the weight vector $v = \langle 0, 0, 0 \rangle$. What is the value of $p(\text{cat}|\text{the}; v)$?

4.2 Question (time: 0:54, slide: 22)

Consider the label set $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{hat}, \text{cot}\}$ with three simple features

- $f_1(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ ends with at} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ starts with c} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ has second letter o} \\ 0 & \text{otherwise} \end{cases}$

Say we are given the weight vector $v = \langle 3, 1, 1 \rangle$. What is the value of $p(\text{hat}|\text{the}; v)$ to three decimal places?

5 Parameter Estimation in Log-linear Models (Part 1)

5.1 Question (time: 12:44, slide: 26)

Consider the label set $\mathcal{Y} = \{\text{cat}, \text{dog}\}$ with three simple features

- $f_1(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ ends with at} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ starts with c} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if } x = \text{the and } y \text{ has second letter o} \\ 0 & \text{otherwise} \end{cases}$

Say we are also given two training examples $(x^{(1)}, y^{(1)}) = (\text{the}, \text{cat})$ and $(x^{(2)}, y^{(2)}) = (\text{the}, \text{dog})$. If our current weight vector is $v = \langle 0, 0, 0 \rangle$ and $L(v)$ is defined as in the slides, what is the value of $\frac{dL(v)}{dv_2}$?

6 Smoothing Regularization in Log-linear Models

6.1 Question (time: 10:07, slide: 31)

Consider the label set $\mathcal{Y} = \{\text{cat}, \text{dog}\}$ with three simple features

- $f_1(x, y) = \begin{cases} 1 & \text{if } x = \text{the} \text{ and } y \text{ ends with at} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } x = \text{the} \text{ and } y \text{ starts with c} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if } x = \text{the} \text{ and } y \text{ has second letter o} \\ 0 & \text{otherwise} \end{cases}$

Say we are also given two training examples $(x^{(1)}, y^{(1)}) = (\text{the}, \text{cat})$ and $(x^{(2)}, y^{(2)}) = (\text{the}, \text{dog})$. If our current weight vector is $v = \langle 1, 1, 1 \rangle$ and $L(v)$ is now defined to use regularization with $\lambda = 1$, what is the value of $\frac{dL(v)}{dv_2}$ to three decimal places?

A Answers

- (a)

There are 1000 possible values for w that could be in \mathcal{Y} and there are 26^4 possible English suffixes of length four. This gives 1000×26^4 features.

- (a)

The important new piece of information is the size of the training set. Since there are only 10000 training examples, we will see *at most* 10000 word and suffix pairs, even though 1000×26^4 are possible.

- 50000

There is one feature for each word and part-of-speech tag. This gives $50 \times 1000 = 50000$ features.

- 5

There are 5 unique word/tag pairs in the sentence. The pair **the**/DT is used twice. Note that **fishes**/V and **fishes**/NN introduce different features.

- 3

The first two features are 1 and the third feature is 0. The total score is $\langle 1, 2, 3 \rangle \cdot \langle 1, 1, 0 \rangle = 3$.

- 0.25

Since the weight vector is 0 all output words have the same score. Therefore all $y \in \mathcal{Y}$, $p(y|\text{the}) = 0.25$ and $p(\text{cat}|\text{the}) = 0.25$

- 0.237

The score of **(the, hat)** is $v \cdot f(\text{the}, \text{hat}) = 3$. The sum of exponential scores is $\sum_{y' \in \mathcal{Y}} v \cdot f(\text{the}, y') = e^4 + e^1 + e^3 + e^2$. Therefore the conditional probability is $p(\text{hat}|\text{the}) = e^3 / (e^4 + e^1 + e^3 + e^2) = 0.237$.

- 0

Starting from the definition of the gradient from the slides: $\frac{dL(v)}{dv_2} = \sum_{i=1}^n f_2(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \sum_{y' \in \mathcal{Y}} f_2(x^{(i)}, y') p(y'|x^{(i)}; v) = 1 - (1 \times 0.5 + 0 \times 0.5) - (1 \times 0.5 + 0 \times 0.5) = 0$.

- -1.462

Note first that $p(\text{cat}|\text{the}) = e^2/(e^2 + e)$ and $p(\text{dog}|\text{the}) = e/(e^2 + e)$. Starting from the definition of the gradient from the slides:

$$\frac{dL(v)}{dv_2} = \sum_{i=1}^n f_2(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \sum_{y' \in \mathcal{Y}} f_2(x^{(i)}, y') p(y'|x^{(i)}; v) - \lambda v_2 = 1 - \frac{e^2}{e^2 + e} - \frac{e^2}{e^2 + e} - (1 \times 1) = -\frac{2e^2}{e^2 + e} = -1.462$$

1 Independence Assumptions in Log-linear Taggers

1.1 Question (time: 8:32, slide: 8)

Say we have $w_1 \dots w_3 = \text{the dog barks}$. We would like

- $p(\text{D N V} \mid \text{the dog barks}) = 0.5$
- $p(\text{D N N} \mid \text{the dog barks}) = 0.5$

What should be the value for the following probabilities

- $p(\text{D} \mid \text{the dog barks}, \ast \ast)$
- $p(\text{N} \mid \text{the dog barks}, \ast \text{D})$
- $p(\text{V} \mid \text{the dog barks}, \text{D N})$
- $p(\text{N} \mid \text{the dog barks}, \text{D N})$

Write your answers to one decimal place separated by a space, e.g. 0.0 0.9
1.0 0.1

2 Features in Log-Linear Taggers

2.1 Question (time: 3:41, slide: 10)

Say we are given the following sentence with partial tag sequence

- the/DT mat/NN saw/VBD the/DT dog/NN in the park

Which of the following tuples is the correct history at this point?

- (a) (DT, NN, the man saw the dog, 6)
- (b) (NN, DT, the man saw the dog, 6)
- (c) (DT, NN, the man saw the dog in the park, 6)
- (d) (NN, IN, the man saw the dog in the park, 6)

2.2 Question (time: 13:21, slide: 14)

Say we are given a history

- $h = (\text{VBD}, \text{DT}, \text{the man saw the dog in the park}, 5)$

Which of the following features have $f(h, \text{NN}) = 1$?

(a) $f(h, t) = \begin{cases} 1 & \text{if } w_i = \text{dog and } t = \text{NN} \\ 0 & \text{otherwise} \end{cases}$

(b) $f(h, t) = \begin{cases} 1 & \text{if } w_i = \text{dog and } t = \text{DT} \\ 0 & \text{otherwise} \end{cases}$

(c) $f(h, t) = \begin{cases} 1 & \text{if } w_{i+1} = \text{dog and } t = \text{NN} \\ 0 & \text{otherwise} \end{cases}$

(d) $f(h, t) = \begin{cases} 1 & \text{if } y_{-1} = \text{DT and } t = \text{NN} \\ 0 & \text{otherwise} \end{cases}$

3 Parameters in Log-Linear Taggers

3.1 Question (time: 3:59, slide: 16)

Consider the training example

- the dog saw the man, DT NN VBD DT NN

Say we convert this example to (x_i, y_i) pairs for training a log-linear model.
Which of the following items are in the training set?

- (a) $x = (\text{DT NN}, \text{the dog saw the man}, 3), y = \text{NN}$
- (b) $x = (\text{VBD DT}, \text{the dog saw the man}, 3), y = \text{VBD}$
- (c) $x = (\text{DT NN}, \text{the dog saw the man}, 3), y = \text{VBD}$
- (d) $x = (\text{DT NN}, \text{the dog saw the man}, 4), y = \text{NN}$

4 The Viterbi Algorithm for Log-linear Taggers

4.1 Question (time: 9:37, slide: 20)

Consider the sentence $w_1 \dots w_n = \text{the dog saw the cat.}$

Say we have the π values

$$\pi(3, \text{NN}, \text{VBD}) = 0.02 \quad \pi(3, \text{NN}, \text{NN}) = 0.01 \quad \pi(3, \text{NN}, \text{DT}) = 0.03$$

$\pi(3, u, v) = 0.0$ for all other values of u and v . And say our log-linear model gives

- $q(\text{DT} | \text{NN}, \text{VBD}, \text{the dog saw the cat}, 4) = 0.4$

- $q(\text{DT} \mid \text{NN}, \text{NN}, \text{the dog saw the cat}, 4) = 0.5$

- $q(\text{DT} \mid \text{DT}, \text{NN}, \text{the dog saw the cat}, 4) = 0.3$

What is the value for $\pi(4, \text{VBD}, \text{DT})$?

A Answers

- 1.0 1.0 0.5 0.5

The two taggings, D N V and D N N, differ only in the last tag and are the only possible taggings for this sentence (probability sums to 1). Therefore we set the last decision probabilities to 0.5 each. The correct answer is 1.0 1.0 0.5 0.5.

- (c)

The incorrect answers either do not have the full sentence in the history or have the part-of-speech tags in the wrong order.

- (a) (d)

The incorrect answers either have $t \neq \text{NN}$ or have w_{i+1} equal to the wrong word.

- (c)

The incorrect responses either have y not equal to the tag at position i or have t_{i-1} or t_{i-2} not equal to the same tags as in the correct tagging.

- 0.008

The answer is 0.008. The value for $\pi(4, \text{VBD}, \text{DT}) = \max_x q(\text{DT} | x, \text{VBD}, \text{the dog saw the cat}, 4) \times \pi(3, x, \text{VBD}) = q(\text{DT} | \text{NN}, \text{VBD}, \text{the dog saw the cat}, 4) \times \pi(3, \text{NN}, \text{VBD}) = 0.008$.

1 Representing Trees as Decision Sequences (Part 1)

1.1 Question (time: 7:23, slide: 9)

Say we have the sentence "the dog is curious" with the chunking
[NP the/DT dog/NN] is [ADJP curious/JJ]

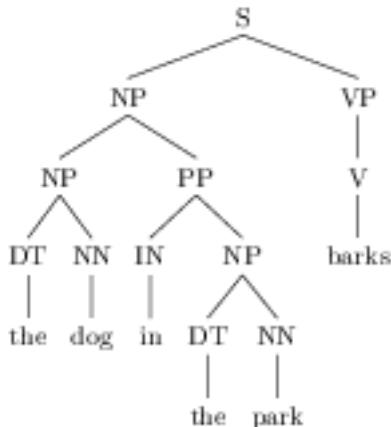
The sentence has four words, so the chunker makes four decisions. What are the correct chunking decisions for this sentence?

(Write in upper case separated by a space, e.g. START(S) OTHER START(NP)
OTHER.)

2 Representing Trees as Decision Sequences (Part 2)

2.1 Question (time: 3:04, slide: 11)

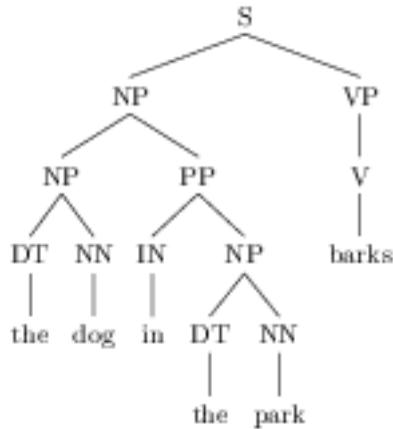
Say we are given the following parse tree



What is the correct first decision in the second layer of this parse? (Write in upper case, e.g. START(S).)

2.2 Question (time: 8:52, slide: 25)

Say we are given the following parse tree



We have just made a JOIN(NP) decision, giving

[START(NP) [NP the dog]] [JOIN(NP) [PP in the park]] [Vi barks]
What is the next decision?

- (a) CHECK=NO
- (b) CHECK=YES

A Answers

- START(NP) JOIN(NP) OTHER START(ADJP)

The answer is START(NP) JOIN(NP) OTHER START(ADJP). We begin the chunk with "the", then join on the word "dog", then place "is" outside the chunk, and end by starting a new chunk with "curious".

- START(NP)

The answer is START(NP). The first layer adds the part-of-speech tags, and the second layer begins by starting the NP over "the dog".

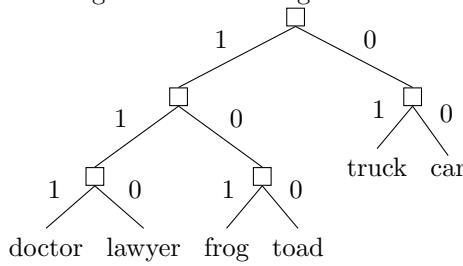
- (b)

Choosing CHECK=YES will merge [START(NP) [NP the dog]] and [JOIN(NP) [PP in the park]] to create the NP spanning "the dog in the park" as in the correct tree.

1 Word Cluster Representations

1.1 Question (time: 8:35, slide: 3)

Say we are given the following word cluster hierarchy



Which words are in the "10" cluster?

- (a) doctor
- (b) frog
- (c) truck
- (d) toad

2 The Brown Clustering Algorithm (Part 1)

2.1 Question (time: 11:50, slide: 7)

Say we are given a deterministic class function C with

$$C(\text{dog}) = 1 \quad C(\text{man}) = 2 \quad C(\text{woman}) = 2 \quad C(\text{walk}) = 3$$

Which of the following are possible definitions of e for a model with this class function?

- (a) $e(\text{dog} | 1) = 1, e(\text{man} | 2) = 0.5, e(\text{woman} | 2) = 0.5, e(\text{walk} | 3) = 1$
- (b) $e(\text{dog} | 1) = 1, e(\text{man} | 2) = 1, e(\text{woman} | 3) = 0.5, e(\text{walk} | 3) = 0.5$
- (c) $e(\text{dog} | 1) = 1, e(\text{man} | 2) = 0.5, e(\text{woman} | 2) = 0.5, e(\text{man} | 3) = 0.5, e(\text{walk} | 3) = 0.5$
- (d) $e(\text{dog} | 1) = 1, e(\text{man} | 2) = 0.9, e(\text{woman} | 2) = 0.1, e(\text{walk} | 3) = 1$

3 The Brown Clustering Algorithm (Part 2)

3.1 Question (time: 8:30, slide: 9)

Say we have estimated a clustering with two classes ($k = 2$) and that we have counts n such that

$$n(1) = 30 \quad n(2) = 10$$

$$n(1, 1) = 25 \quad n(2, 2) = 5 \quad n(1, 2) = 5 \quad n(2, 1) = 5$$

We now want to compute the quality of our clustering. What is the value of the expression $\sum_{c=1}^k \sum_{c'=1}^k p(c, c') \log \frac{p(c, c')}{p(c)p(c')}$?

A Answers

- (b) (d)

Starting from the top of the tree we traverse the left edge (1) and then the right edge (0). The two words underneath this node are "frog" and "toad".

- (a) (d)

Since the class function is deterministic, words should have zero probability of being emitted from a different class, e.g. $e(\text{man} \mid 3)$ must be 0.

- -0.341

The answer is -0.341. First note that $p(1) = 3/4$, $p(2) = 1/4$, $p(1,1) = 0.625$, and $p(1,2) = p(2,1) = p(2,2) = 0.125$. So the sum is $0.625 \log_{9/16}^{0.625} + 2 \times 0.125 \log_{3/16}^{0.125} + 0.125 \log_{1/16}^{0.125} = -0.341$.

1 Three Components of GLMs

1.1 Question (time: 8:46, slide: 17)

Consider a set of tags $\mathcal{T} = \{ \text{DT}, \text{V}, \text{NN} \}$.

Say we are given the sentence $x = \text{the dog walked to the store}$ and asked to construct a global linear model for tagging.

What is an upper bound on the size of $\text{GEN}(x)$?

1.2 Question (time: 8:46, slide: 17)

Consider a set of tags $\mathcal{T} = \{ \text{DT}, \text{V}, \text{NN} \}$.

Say we are given the sentence $x = \text{the dog walked to the store}$ and asked to construct a global linear model for tagging.

If we define $\text{GEN}(x)$ to generate the top N tag sequences where $N = 100$, what is the size of $\text{GEN}(x)$?

1.3 Question (time: 14:39, slide: 0)

Consider $x = \text{the dog laughs}$ and $\text{GEN}(x)$ made up of the tag sequences

DT NN V DT NN DT DT DT DT

Now say we are given f consisting of the following three feature functions

- $f_1(x, y) = \begin{cases} 1 & \text{if } y \text{ starts with DT and ends with V} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } y \text{ contains two DTs} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if all tags the same in } y \\ 0 & \text{otherwise} \end{cases}$

What is the feature vector of the first tagging in $\text{GEN}(x)$? (Write each value in the vector separated by a space, e.g. 1 1 0).

1.4 Question (time: 14:39, slide: 20)

Consider $x = \text{the dog laughs}$ and $\text{GEN}(x)$ made up of the tag sequences

DT NN V DT NN DT DT DT DT

Now say we are given f consisting of the following three feature functions

- $f_1(x, y) = \begin{cases} 1 & \text{if } y \text{ starts with DT and ends with V} \\ 0 & \text{otherwise} \end{cases}$
- $f_2(x, y) = \begin{cases} 1 & \text{if } y \text{ contains two DTs} \\ 0 & \text{otherwise} \end{cases}$
- $f_3(x, y) = \begin{cases} 1 & \text{if all tags the same in } y \\ 0 & \text{otherwise} \end{cases}$

If we are given the weight vector $v = \langle 10, 2, 9 \rangle$, what is $\max_{y \in \text{GEN}(x)} f(x, y) \cdot v$?

2 Parameter Estimation with the Perceptron Algorithm

2.1 Question (time: 6:11, slide: 30)

Say we are running the perceptron algorithm. We have reached input x_i and the set $\{f(x_i, y) : y \in \text{GEN}(x_i)\}$ is made up of the vectors

- $\langle 0, 1, 0, 1 \rangle$
- $\langle 0, 1, 1, 1 \rangle$
- $\langle 1, 1, 0, 1 \rangle$

Also we know that $f(x_i, y_i) = \langle 1, 1, 0, 1 \rangle$ and that our current parameters are $v = \langle -2, 5, 2, 0 \rangle$.

What will be the value of v at the end of this iteration? (Write each value in the vector separated by a space, e.g. 0 1 1 0).

2.2 Question (time: 6:11, slide: 30)

Say we are running the perceptron algorithm. We have reached input x_i and the set $\{f(x_i, y) : y \in \text{GEN}(x_i)\}$ is made up of the vectors

- $\langle 0, 1, 0, 1 \rangle$
- $\langle 0, 1, 1, 1 \rangle$
- $\langle 1, 1, 0, 1 \rangle$

Also we know that $f(x_i, y_i) = \langle 1, 1, 0, 1 \rangle$ and that our current $v = \langle 2, 5, 1, 0 \rangle$.

What will be the value of v at the end of this iteration? (Write each value in the vector separated by a space, e.g. 0 1 1 0).

A Answers

- 729

The answer is 729. At its largest, $\text{GEN}(x)$ is a set containing all possible tag sequences. There are 3 tags and 6 words, which gives $\text{GEN}(x) = 729$.

- 100

The answer is 100. $\text{GEN}(x)$ is a set containing only the top 100 possible tag sequences. Even though there are 729 different tag sequences, we only consider the top 100 within $\text{GEN}(x)$.

- 1 0 0

The answer is 1 0 0. The sentence is the/DT dog/NN laughs/V. Of the three features only the first is 1, and the other two are 0.

- 11

The answer is 11. The last sentence $y = \text{DT DT DT}$ has feature vector $f(x, y) = \langle 0, 1, 1 \rangle$, and so $f(x, y) \cdot v = 11$.

- -1 5 1 0

The answer is -1 5 1 0. First we compute the highest scoring vector z_i which is $f(x_i, z_i) = \langle 0, 1, 1, 1 \rangle$. Then we update the parameters $v = v + f(x_i, y_i) - f(x_i, z_i) = \langle -1, 5, 1, 0 \rangle$

- 2 5 1 0

The answer is 2 5 1 0. First we compute the highest scoring vector z_i which is $f(x_i, z_i) = \langle 1, 1, 0, 1 \rangle$. Then we update the parameters $v = v + f(x_i, y_i) - f(x_i, z_i) = \langle 2, 5, 1, 0 \rangle$. Since the correct answer has the same feature vector as the selected answer, the parameters do not change.

1 GLMs for Tagging (Part 2)

1.1 Question (time: 2:16, slide: 8)

Consider a set of tags $\mathcal{T} = \{ \text{DT}, \text{V}, \text{NN}, \text{ADV} \}$ and sentence $x = \text{the dog walked to the park.}$

If a history is defined as $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$ how many histories are there with $i = 3$?

1.2 Question (time: 4:40, slide: 8)

Given that a history is defined as $h = \langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$ and say $n = 10$ which of the following are valid local features?

- (a) $g_1(h, t) = \begin{cases} 1 & \text{if } t = \text{VBG and } t_{-1} = \text{IN} \\ 0 & \text{otherwise} \end{cases}$
- (b) $g_2(h, t) = \begin{cases} 1 & \text{if } t = \text{VBG and } w_{n-2} = \text{dog} \\ 0 & \text{otherwise} \end{cases}$
- (c) $g_3(h, t) = \begin{cases} 1 & \text{if } t = \text{VBG and } t_{-3} = \text{NN} \\ 0 & \text{otherwise} \end{cases}$
- (d) $g_4(h, t) = \begin{cases} 1 & \text{if } t = \text{VBG and } t_{-1} = \text{IN and } w_2 = \text{dog} \\ 0 & \text{otherwise} \end{cases}$

2 GLMs for Tagging (Part 3)

2.1 Question (time: 3:45, slide: 12)

Consider a set of tags $\mathcal{T} = \{ \text{DT}, \text{V}, \text{NN}, \text{ADV}, \text{IN} \}$, sentence $x = \text{the dog walked to the park,}$ and tag sequence $y = \text{DT NN V IN DT NN}$

Say we define local features

- $g_1(h, t) = \begin{cases} 1 & \text{if } t = \text{NN and } w_i = \text{dog} \\ 0 & \text{otherwise} \end{cases}$
- $g_2(h, t) = \begin{cases} 1 & \text{if } t = \text{NN and } t_{-1} = \text{DT} \\ 0 & \text{otherwise} \end{cases}$

- $g_3(h, t) = \begin{cases} 1 & \text{if } t = \text{NN and } t_{-1} = \text{DT and } w_{i-1} = \text{the} \\ 0 & \text{otherwise} \end{cases}$

What is the global feature vector $f(x, y)$? (Write each value separated by a space e.g. 2 0 0.)

3 GLMs for Tagging (Part 4)

3.1 Question (time: 3:51, slide: 14)

Say we are running the perceptron algorithm for a model with the following local features

- $g_1(h, t) = \begin{cases} 1 & \text{if } t = \text{DT and } w_i = \text{the} \\ 0 & \text{otherwise} \end{cases}$
- $g_2(h, t) = \begin{cases} 1 & \text{if } t = \text{NN and } t_{-1} = \text{DT} \\ 0 & \text{otherwise} \end{cases}$
- $g_3(h, t) = \begin{cases} 1 & \text{if } t = \text{NN and } t_{-1} = \text{DT and } w_i = \text{dog} \\ 0 & \text{otherwise} \end{cases}$

Our current sentence is $x_i = \text{the dog walks to the park}$. The correct tagging is $y_i = \text{DT NN V IN DT NN}$, and the current best tagging is $z_{[1:n_i]} = \text{NN NN V IN DT NN}$.

If the current weight vector is $v = \langle -1, -2, 2 \rangle$, what will the new weight be after the perceptron update? (Write each value in the vector separated by a space, e.g. 0 1 1 0).

A Answers

- 16

The answer is 16. Since w and i are fixed, we need to consider all values for t_{-2} and t_{-1} . There are 4^2 possible values.

- (a) (b) (d)

Local features can be based on any of the source words, but may only look at the two previous tags and the current tag. The incorrect local features look at tags outside this range.

- 1 2 2

The answer is 1 2 2. The global feature vector is formed by summing up the counts of the features over the entire sentence. In this examples g_1 is used once, g_2 is used twice and g_3 is used twice.

- None

The answer is 0 -1 3. The correct global feature vector is $f(x_i, y_i) = \langle 2, 2, 1 \rangle$. The predicted global feature vector is $f(x_i, z) = \langle 1, 1, 0 \rangle$. Therefore after the update the weight vector $v = \langle 0, -1, 3 \rangle$.

1 The Dependency Parsing Problem (Part 2)

1.1 Question (time: 3:45, slide: 6)

Consider the sentence "John saw a movie".

Draw the following dependency parses. Which are valid (projective) parses?

- (a) (2, 1), (0, 2), (1, 3), (3,4)
- (b) (2, 1), (0, 2), (2, 3), (3,4)
- (c) (2, 1), (0, 2), (2, 4), (3,4)
- (d) (0, 1), (1, 2), (2, 3), (3,4)

2 GLMs for Dependency Parsing (Part 1)

2.1 Question (time: 4:01, slide: 12)

Say we have a sentence "John saw a movie" and we are computing features.
How many possible arcs (h, m) are there for this sentence?

A Answers

- (b) (d)

The incorrect parses either have a crossing dependencies or have a word used multiple times as a modifier.

- 20

The answer is 20. There are four words that are possible modifiers and five words that are possible heads. The total is $5 \times 4 = 20$.

Global Linear Models

Michael Collins, Columbia University

Overview

- ▶ A brief review of history-based methods
- ▶ A new framework: Global linear models
- ▶ Parsing problems in this framework:
Reranking problems
- ▶ Parameter estimation method 1:
A variant of the perceptron algorithm

Techniques

- ▶ So far:
 - ▶ Smoothed estimation
 - ▶ Probabilistic context-free grammars
 - ▶ Log-linear models
 - ▶ Hidden markov models
 - ▶ The EM Algorithm
 - ▶ History-based models
- ▶ Today:
 - ▶ Global linear models

Supervised Learning in Natural Language

- ▶ General task: induce a function F from members of a set \mathcal{X} to members of a set \mathcal{Y} . e.g.,

Problem	$x \in \mathcal{X}$	$y \in \mathcal{Y}$
Parsing	sentence	parse tree
Machine translation	French sentence	English sentence
POS tagging	sentence	sequence of tags

- ▶ Supervised learning:
we have a *training set* (x_i, y_i) for $i = 1 \dots n$

The Models so far

- ▶ Most of the models we've seen so far are **history-based models**:
 - ▶ We break structures down into a **derivation**, or sequence of decisions
 - ▶ Each decision has an associated conditional probability
 - ▶ Probability of a structure is a product of decision probabilities
 - ▶ Parameter values are estimated using variants of maximum-likelihood estimation
 - ▶ Function $F : \mathcal{X} \rightarrow \mathcal{Y}$ is defined as

$$F(x) = \operatorname{argmax}_y p(x, y; \Theta) \quad \text{or} \quad F(x) = \operatorname{argmax}_y p(y|x; \Theta)$$

Example 1: PCFGs

- ▶ We break structures down into a derivation, or sequence of decisions
We have a top-down derivation, where each decision is to expand some non-terminal α with a rule $\alpha \rightarrow \beta$
- ▶ Each decision has an associated conditional probability
 $\alpha \rightarrow \beta$ has probability $q(\alpha \rightarrow \beta)$
- ▶ Probability of a structure is a product of decision probabilities

$$p(T, S) = \prod_{i=1}^n q(\alpha_i \rightarrow \beta_i)$$

where $\alpha_i \rightarrow \beta_i$ for $i = 1 \dots n$ are the n rules in the tree

- ▶ Parameter values are estimated using variants of maximum-likelihood estimation

$$q(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

- ▶ Function $F : \mathcal{X} \rightarrow \mathcal{Y}$ is defined as

$$F(x) = \operatorname{argmax}_y p(y, x; \Theta)$$

Example 2: Log-linear Taggers

- ▶ We break structures down into a derivation, or sequence of decisions
For a sentence of length n we have n tagging decisions, in left-to-right order
- ▶ Each decision has an associated conditional probability

$$p(t_i \mid t_{i-1}, t_{i-2}, w_1 \dots w_n)$$

where t_i is the i 'th tagging decision, w_i is the i 'th word

- ▶ Probability of a structure is a product of decision probabilities

$$p(t_1 \dots t_n \mid w_1 \dots w_n) = \prod_{i=1}^n p(t_i \mid t_{i-1}, t_{i-2}, w_1 \dots w_n)$$

- ▶ Parameter values are estimated using variants of maximum-likelihood estimation
 $p(t_i \mid t_{i-1}, t_{i-2}, w_1 \dots w_n)$ is estimated using a log-linear model
- ▶ Function $F : \mathcal{X} \rightarrow \mathcal{Y}$ is defined as

$$F(x) = \operatorname{argmax}_y p(y \mid x; \Theta)$$

A New Set of Techniques: Global Linear Models

Overview of today's lecture:

- ▶ Global linear models as a framework
- ▶ Parsing problems in this framework:
 - ▶ Reranking problems
- ▶ A variant of the perceptron algorithm

Global Linear Models as a Framework

- ▶ We'll move away from history-based models
No idea of a “derivation”, or attaching probabilities to “decisions”
- ▶ Instead, we'll have feature vectors over entire structures
“Global features”
- ▶ First piece of motivation:
Freedom in defining features

A Need for Flexible Features

Example 1 Parallelism in coordination [Johnson et. al 1999]

Constituents with similar structure tend to be coordinated

⇒ how do we allow the parser to learn this preference?

Bars in New York and pubs in London
vs. Bars in New York and pubs

A Need for Flexible Features (continued)

Example 2 Semantic features

We might have an ontology giving properties of various nouns/verbs

⇒ how do we allow the parser to use this information?

pour the **cappuccino**

vs. pour the **book**

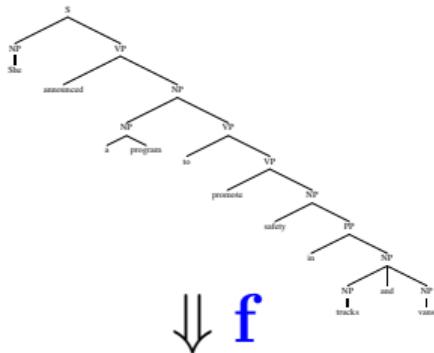
Ontology states that **cappuccino** has the +liquid feature, **book** does not.

Three Components of Global Linear Models

- ▶ \mathbf{f} is a function that maps a structure (x, y) to a **feature vector** $\mathbf{f}(x, y) \in \mathbb{R}^d$
- ▶ **GEN** is a function that maps an input x to a set of **candidates** $\text{GEN}(x)$
- ▶ \mathbf{v} is a parameter vector (also a member of \mathbb{R}^d)
- ▶ Training data is used to set the value of \mathbf{v}

Component 1: f

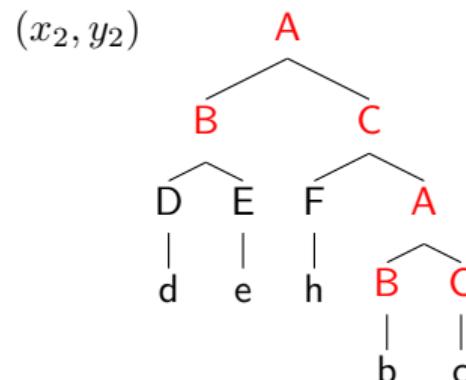
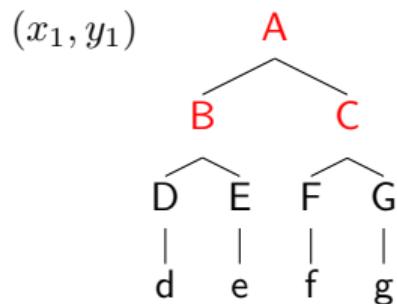
- ▶ f maps a candidate to a **feature vector** $\in \mathbb{R}^d$
 - ▶ f defines the **representation** of a candidate
-


$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$$

Features

- ▶ A “feature” is a function on a structure, e.g.,

$h(x, y) = \text{Number of times } \boxed{\begin{array}{c} \text{A} \\ \diagup \quad \diagdown \\ \text{B} \quad \text{C} \end{array}} \text{ is seen in } (x, y)$



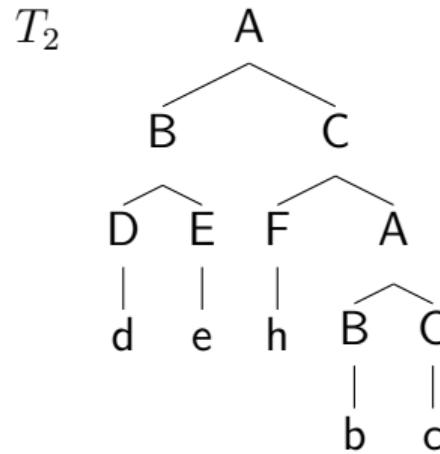
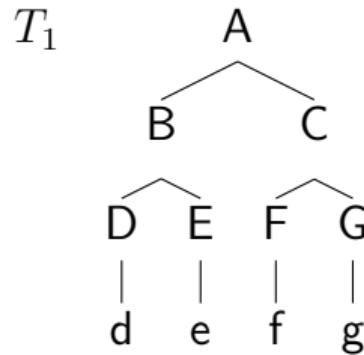
$$h(x_1, y_1) = 1$$

$$h(x_2, y_2) = 2$$

Feature Vectors

- ▶ A set of functions $h_1 \dots h_d$ define a **feature vector**

$$\mathbf{f}(x) = \langle h_1(x), h_2(x) \dots h_d(x) \rangle$$



$$\mathbf{f}(T_1) = \langle 1, 0, 0, 3 \rangle$$

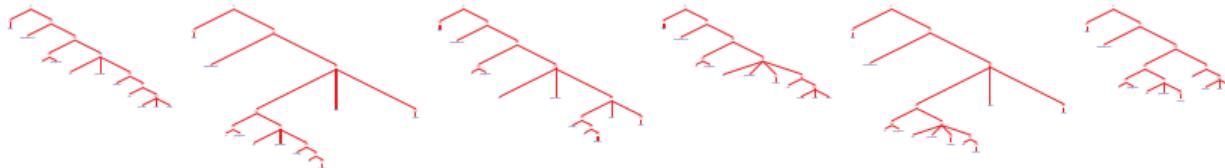
$$\mathbf{f}(T_2) = \langle 2, 0, 1, 1 \rangle$$

Component 2: GEN

- ▶ GEN enumerates a set of **candidates** for a sentence
-

She announced a program to promote safety in trucks and vans

↓ GEN

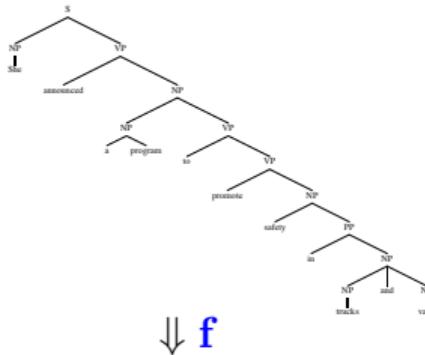


Component 2: **GEN**

- ▶ **GEN** enumerates a set of **candidates** for an input x
- ▶ Some examples of how **GEN**(x) can be defined:
 - ▶ Parsing: **GEN**(x) is the set of parses for x under a grammar
 - ▶ Any task: **GEN**(x) is the top N most probable parses under a history-based model
 - ▶ Tagging: **GEN**(x) is the set of all possible tag sequences with the same length as x
 - ▶ Translation: **GEN**(x) is the set of all possible English translations for the French sentence x

Component 3: \mathbf{v}

- ▶ \mathbf{v} is a **parameter vector** $\in \mathbb{R}^d$
 - ▶ \mathbf{f} and \mathbf{v} together map a candidate to a real-valued score
-



↓ \mathbf{f}

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle$$

↓ $\mathbf{f} \cdot \mathbf{v}$

$$\langle 1, 0, 2, 0, 0, 15, 5 \rangle \cdot \langle 1.9, -0.3, 0.2, 1.3, 0, 1.0, -2.3 \rangle = 5.8$$

Putting it all Together

- ▶ \mathcal{X} is set of sentences, \mathcal{Y} is set of possible outputs (e.g. trees)
- ▶ Need to learn a function $\textcolor{magenta}{F} : \mathcal{X} \rightarrow \mathcal{Y}$
- ▶ $\textcolor{red}{\text{GEN}}$, $\textcolor{blue}{f}$, \mathbf{v} define

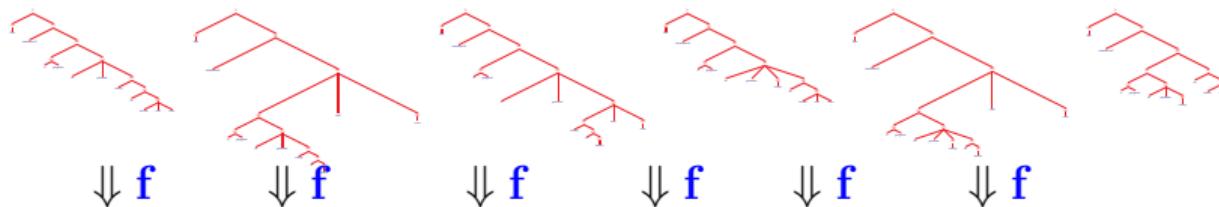
$$\textcolor{magenta}{F}(x) = \arg \max_{y \in \textcolor{red}{\text{GEN}}(x)} \textcolor{blue}{f}(x, y) \cdot \mathbf{v}$$

Choose the highest scoring candidate as the most plausible structure

- ▶ Given examples (x_i, y_i) , how to set \mathbf{v} ?

She announced a program to promote safety in trucks and vans

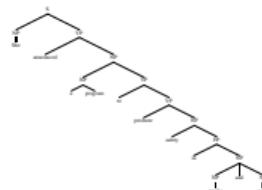
↓ GEN



$\langle 1, 1, 3, 5 \rangle$ $\langle 2, 0, 0, 5 \rangle$ $\langle 1, 0, 1, 5 \rangle$ $\langle 0, 0, 3, 0 \rangle$ $\langle 0, 1, 0, 5 \rangle$ $\langle 0, 0, 1, 5 \rangle$

↓ $f \cdot v$ ↓ $f \cdot v$
13.6 12.2 12.1 3.3 9.4 11.1

↓ arg max



Overview

- ▶ A brief review of history-based methods
- ▶ A new framework: Global linear models
- ▶ Parsing problems in this framework:
Reranking problems
- ▶ Parameter estimation method 1:
A variant of the perceptron algorithm

Reranking Approaches to Parsing

- ▶ Use a **baseline** parser to produce top N parses for each sentence in training and test data
GEN(x) is the top N parses for x under the baseline model
- ▶ One method: use a lexicalized PCFG to generate a number of parses
(in our experiments, around 25 parses on average for 40,000 training sentences, giving \approx 1 million training parses)
- ▶ **Supervision:** for each x_i take y_i to be the parse that is “closest” to the treebank parse in **GEN(x_i)**

The Representation \mathbf{f}

- ▶ Each component of \mathbf{f} could be essentially *any* feature over parse trees
- ▶ For example:

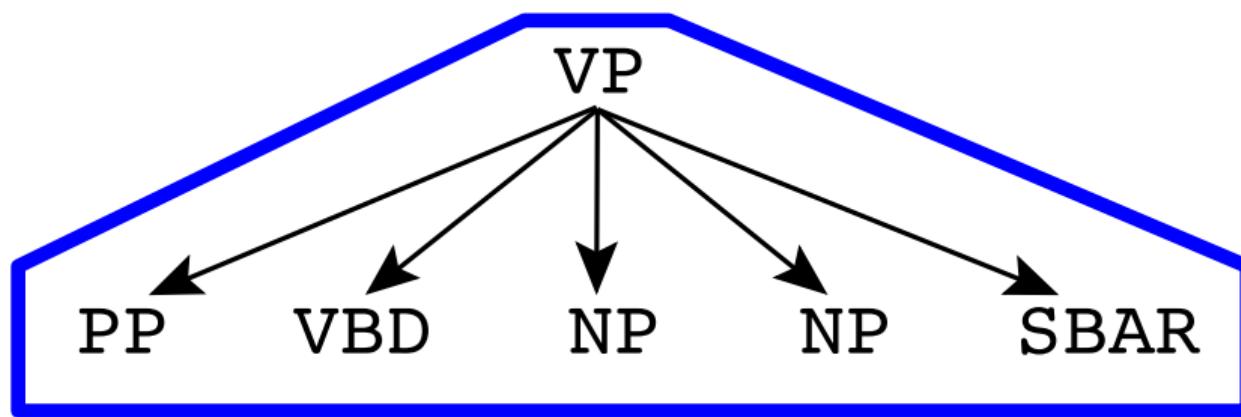
$$\textcolor{blue}{f}_1(x, y) = \log \text{ probability of } (x, y) \text{ under the baseline model}$$

$$\textcolor{blue}{f}_2(x, y) = \begin{cases} 1 & \text{if } (x, y) \text{ includes the rule VP} \rightarrow \text{PP VBD NP} \\ 0 & \text{otherwise} \end{cases}$$

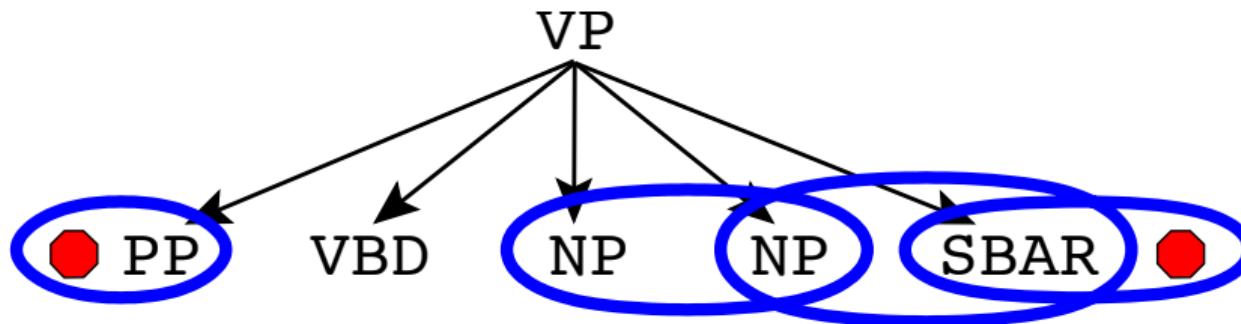
From [Collins and Koo, 2005]:

The following types of features were included in the model. We will use the rule $VP \rightarrow PP\ VBD\ NP\ NP\ SBAR$ with head VBD as an example. Note that the output of our baseline parser produces syntactic trees with headword annotations.

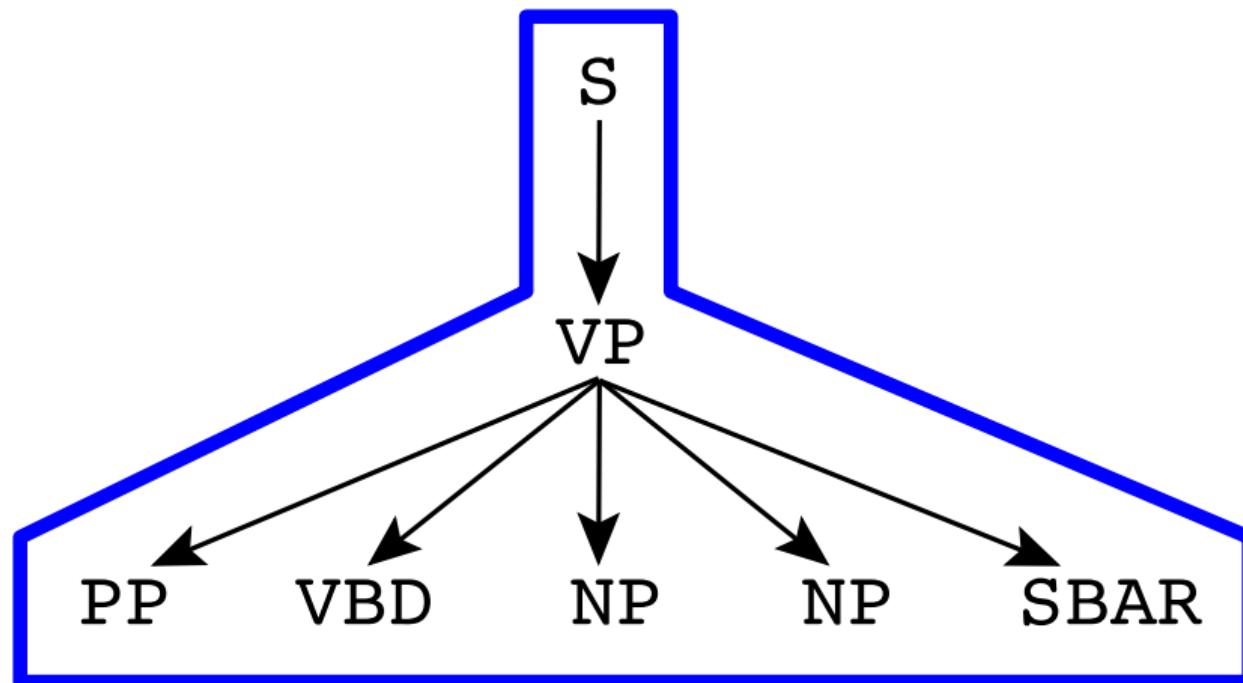
Rules These include all context-free rules in the tree, for example
 $VP \rightarrow PP\ VBD\ NP\ NP\ SBAR.$



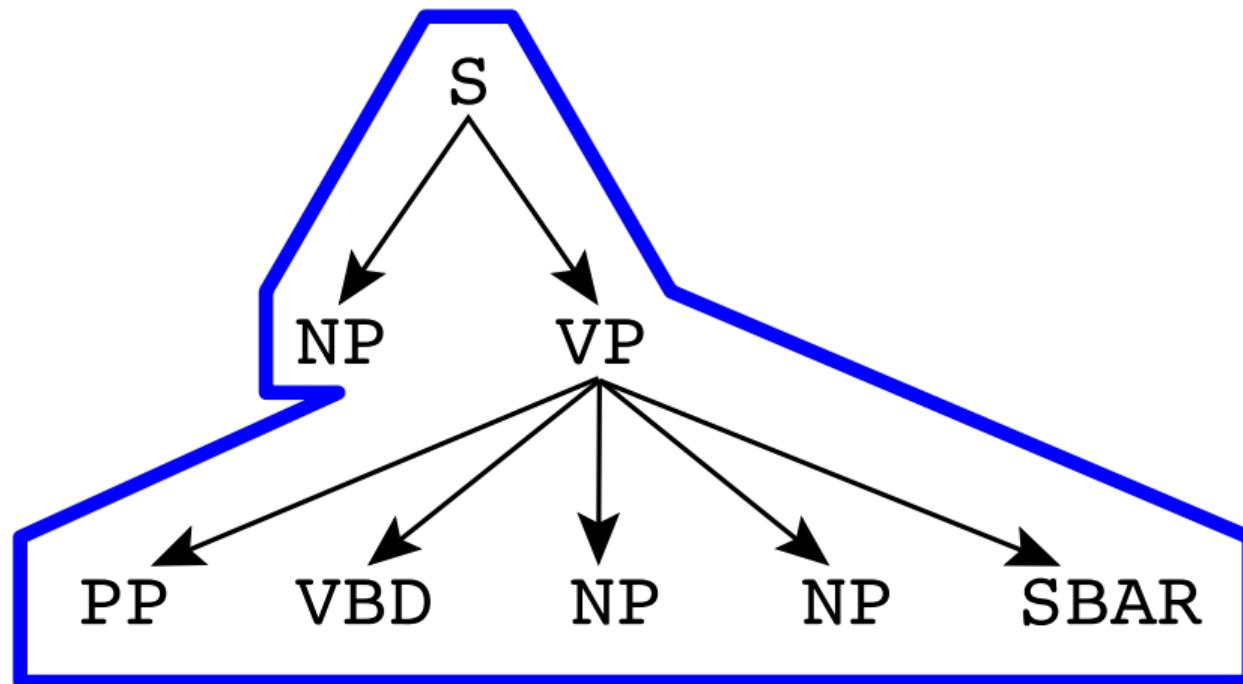
Bigrams These are adjacent pairs of non-terminals to the left and right of the head. As shown, the example rule would contribute the bigrams (Right, VP, NP, NP), (Right, VP, NP, SBAR), (Right, VP, SBAR, STOP), and (Left, VP, PP, STOP) to the left of the head.



Grandparent Rules Same as **Rules**, but also including the non-terminal above the rule.



Two-level Rules Same as **Rules**, but also including the entire rule above the rule.



Overview

- ▶ A brief review of history-based methods
- ▶ A new framework: Global linear models
- ▶ Parsing problems in this framework:
Reranking problems
- ▶ Parameter estimation method 1:
A variant of the perceptron algorithm

A Variant of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \dots n$

Initialization: $\mathbf{v} = 0$

Define: $F(x) = \operatorname{argmax}_{y \in \text{GEN}(x)} \mathbf{f}(x, y) \cdot \mathbf{v}$

Algorithm:

For $t = 1 \dots T$, $i = 1 \dots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\mathbf{v} = \mathbf{v} + \mathbf{f}(x_i, y_i) - \mathbf{f}(x_i, z_i)$

Output: Parameters \mathbf{v}

Perceptron Experiments: Parse Reranking

Parsing the Wall Street Journal Treebank

Training set = 40,000 sentences, test = 2,416 sentences

Generative model (Collins 1999): 88.2% F-measure

Reranked model: 89.5% F-measure (**11% relative error reduction**)

- ▶ Results from Charniak and Johnson, 2005:
 - ▶ Improvement from 89.7% (baseline generative model) to 91.0% accuracy
 - ▶ Gains from improved n-best lists, better features, better baseline model

Summary

- ▶ A new framework: **global linear models**
GEN, f, v
- ▶ There are several ways to train the parameters **v**:
 - ▶ Perceptron
 - ▶ Boosting
 - ▶ Log-linear models (maximum-likelihood)
- ▶ Applications:
 - ▶ Parsing
 - ▶ Generation
 - ▶ Machine translation
 - ▶ Tagging problems
 - ▶ Speech recognition

Global Linear Models Part 2: The Perceptron Algorithm for Tagging

Michael Collins, Columbia University

Recap: Three Components of Global Linear Models

- ▶ \mathbf{f} is a function that maps a structure (x, y) to a **feature vector** $\mathbf{f}(x, y) \in \mathbb{R}^d$
- ▶ **GEN** is a function that maps an input x to a set of **candidates** $\text{GEN}(x)$
- ▶ \mathbf{v} is a parameter vector (also a member of \mathbb{R}^d)
- ▶ Training data is used to set the value of \mathbf{v}

Recap: Putting it all Together

- ▶ \mathcal{X} is set of sentences, \mathcal{Y} is set of possible outputs (e.g. trees)
- ▶ Need to learn a function $\textcolor{magenta}{F} : \mathcal{X} \rightarrow \mathcal{Y}$
- ▶ **GEN**, \mathbf{f} , \mathbf{v} define

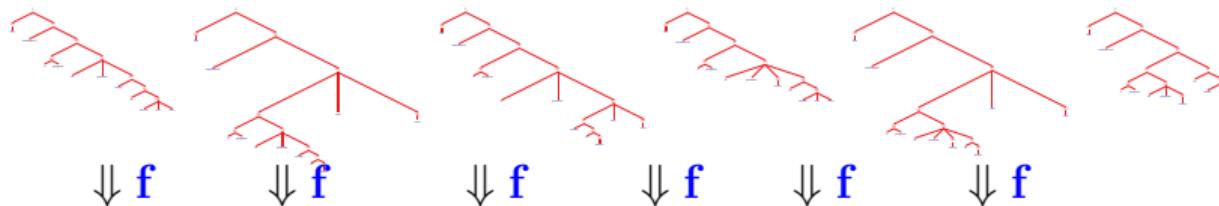
$$\textcolor{magenta}{F}(x) = \arg \max_{y \in \textcolor{red}{\text{GEN}}(x)} \mathbf{f}(x, y) \cdot \mathbf{v}$$

Choose the highest scoring candidate as the most plausible structure

- ▶ Given examples (x_i, y_i) , how to set \mathbf{v} ?

She announced a program to promote safety in trucks and vans

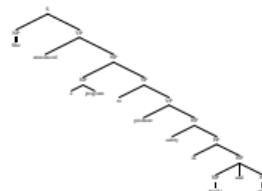
↓ GEN



$\langle 1, 1, 3, 5 \rangle$ $\langle 2, 0, 0, 5 \rangle$ $\langle 1, 0, 1, 5 \rangle$ $\langle 0, 0, 3, 0 \rangle$ $\langle 0, 1, 0, 5 \rangle$ $\langle 0, 0, 1, 5 \rangle$

↓ $f \cdot v$ ↓ $f \cdot v$
13.6 12.2 12.1 3.3 9.4 11.1

↓ arg max



Recap: A Variant of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \dots n$

Initialization: $\mathbf{v} = 0$

Define: $F(x) = \operatorname{argmax}_{y \in \text{GEN}(x)} \mathbf{f}(x, y) \cdot \mathbf{v}$

Algorithm:

For $t = 1 \dots T$, $i = 1 \dots n$

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\mathbf{v} = \mathbf{v} + \mathbf{f}(x_i, y_i) - \mathbf{f}(x_i, z_i)$

Output: Parameters \mathbf{v}

Tagging Problems

TAGGING: Strings to Tagged Sequences

a b e e a f h j ⇒ a/C b/D e/C e/C a/D f/C h/D j/C

Example 1: Part-of-speech tagging

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV
topping/V forecasts/N on/P Wall/N Street/N ,/, as/P
their/POSS CEO/N Alan/N Mulally/N announced/V first/ADJ
quarter/N results/N ./.

Example 2: Named Entity Recognition

Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA
topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA
their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA
quarter/NA results/NA ./NA

Tagging using Global Linear Models

- ▶ Inputs x are sentences $w_{[1:n]} = \{w_1 \dots w_n\}$
- ▶ Define \mathcal{T} to be the set of possible tags
- ▶ **GEN**($w_{[1:n]}$) = \mathcal{T}^n i.e. all tag sequences of length n
- ▶ Note: The size of **GEN** is exponential in the sentence length
- ▶ How do we define **f**?

Representation: Histories

- ▶ A **history** is a 4-tuple $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
 - ▶ t_{-2}, t_{-1} are the previous two tags.
 - ▶ $w_{[1:n]}$ are the n words in the input sentence.
 - ▶ i is the index of the word being tagged
-

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ
base/?? from which Spain expanded its empire into the rest of the
Western Hemisphere .

- ▶ $t_{-2}, t_{-1} = \text{DT, JJ}$
- ▶ $w_{[1:n]} = \langle \text{Hispaniola, quickly, became, \dots, Hemisphere, .} \rangle$
- ▶ $i = 6$

Local Feature-Vector Representations

- ▶ Take a history/tag pair (h, t) .
 - ▶ $g_s(h, t)$ for $s = 1 \dots d$ are **local features** representing tagging decision t in context h .
-

Example: POS Tagging

- **Word/tag features**

$$g_{100}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } t = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$g_{101}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in ing and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

- **Contextual Features**

$$g_{103}(h, t) = \begin{cases} 1 & \text{if } \langle t_{-2}, t_{-1}, t \rangle = \langle \text{DT}, \text{JJ}, \text{VB} \rangle \\ 0 & \text{otherwise} \end{cases}$$

A tagged sentence with n words has n history/tag pairs

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/NN

History			Tag	
t_{-2}	t_{-1}	$w_{[1:n]}$	i	t
*	*	$\langle Hispaniola, quickly, \dots, \rangle$	1	NNP
*	NNP	$\langle Hispaniola, quickly, \dots, \rangle$	2	RB
NNP	RB	$\langle Hispaniola, quickly, \dots, \rangle$	3	VB
RB	VB	$\langle Hispaniola, quickly, \dots, \rangle$	4	DT
VP	DT	$\langle Hispaniola, quickly, \dots, \rangle$	5	JJ
DT	JJ	$\langle Hispaniola, quickly, \dots, \rangle$	6	NN

A tagged sentence with n words has n history/tag pairs

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/NN

History			Tag	
t_{-2}	t_{-1}	$w_{[1:n]}$	i	t
*	*	$\langle \text{Hispaniola}, \text{quickly}, \dots, \rangle$	1	NNP
*	NNP	$\langle \text{Hispaniola}, \text{quickly}, \dots, \rangle$	2	RB
NNP	RB	$\langle \text{Hispaniola}, \text{quickly}, \dots, \rangle$	3	VB
RB	VB	$\langle \text{Hispaniola}, \text{quickly}, \dots, \rangle$	4	DT
VP	DT	$\langle \text{Hispaniola}, \text{quickly}, \dots, \rangle$	5	JJ
DT	JJ	$\langle \text{Hispaniola}, \text{quickly}, \dots, \rangle$	6	NN

Define global features through local features:

$$\mathbf{f}(t_{[1:n]}, w_{[1:n]}) = \sum_{i=1}^n g(h_i, t_i)$$

where t_i is the i 'th tag, h_i is the i 'th history

Global and Local Features

- Typically, local features are indicator functions, e.g.,

$$g_{101}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \text{ing and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

- and global features are then counts,

$f_{101}(w_{[1:n]}, t_{[1:n]})$ = Number of times a word ending in ing is tagged as VBG in $(w_{[1:n]}, t_{[1:n]})$

Putting it all Together

- ▶ $\text{GEN}(w_{[1:n]})$ is the set of all tagged sequences of length n
- ▶ GEN , \mathbf{f} , \mathbf{v} define

$$\begin{aligned} F(w_{[1:n]}) &= \arg \max_{t_{[1:n]} \in \text{GEN}(w_{[1:n]})} \mathbf{v} \cdot \mathbf{f}(w_{[1:n]}, t_{[1:n]}) \\ &= \arg \max_{t_{[1:n]} \in \text{GEN}(w_{[1:n]})} \mathbf{v} \cdot \sum_{i=1}^n g(h_i, t_i) \\ &= \arg \max_{t_{[1:n]} \in \text{GEN}(w_{[1:n]})} \sum_{i=1}^n \mathbf{v} \cdot g(h_i, t_i) \end{aligned}$$

Dynamic programming can be used to find the argmax!

A Variant of the Perceptron Algorithm

Inputs: Training set (x_i, y_i) for $i = 1 \dots n$

Initialization: $\mathbf{v} = 0$

Define: $F(x) = \operatorname{argmax}_{y \in \text{GEN}(x)} \mathbf{f}(x, y) \cdot \mathbf{v}$

Algorithm:

- For $t = 1 \dots T$, $i = 1 \dots n$
- $z_i = F(x_i)$
- If $(z_i \neq y_i)$ $\mathbf{v} = \mathbf{v} + \mathbf{f}(x_i, y_i) - \mathbf{f}(x_i, z_i)$

Output: Parameters \mathbf{v}

Training a Tagger Using the Perceptron Algorithm

Inputs: Training set $(w_{[1:n_i]}^i, t_{[1:n_i]}^i)$ for $i = 1 \dots n$.

Initialization: $\mathbf{v} = 0$

Algorithm: For $t = 1 \dots T, i = 1 \dots n$

$$z_{[1:n_i]} = \arg \max_{u_{[1:n_i]} \in \mathcal{T}^{n_i}} \mathbf{v} \cdot \mathbf{f}(w_{[1:n_i]}^i, u_{[1:n_i]})$$

$z_{[1:n_i]}$ can be computed with the dynamic programming (Viterbi) algorithm

If $z_{[1:n_i]} \neq t_{[1:n_i]}^i$ then

$$\mathbf{v} = \mathbf{v} + \mathbf{f}(w_{[1:n_i]}^i, t_{[1:n_i]}^i) - \mathbf{f}(w_{[1:n_i]}^i, z_{[1:n_i]})$$

Output: Parameter vector \mathbf{v} .

An Example

Say the correct tags for i 'th sentence are

the/DT man/NN bit/VBD the/DT dog/NN

Under current parameters, output is

the/DT man/NN bit/NN the/DT dog/NN

Assume also that features track: (1) all bigrams; (2) word/tag pairs

Parameters incremented:

$\langle \text{NN}, \text{VBD} \rangle, \langle \text{VBD}, \text{DT} \rangle, \langle \text{VBD} \rightarrow \text{bit} \rangle$

Parameters decremented:

$\langle \text{NN}, \text{NN} \rangle, \langle \text{NN}, \text{DT} \rangle, \langle \text{NN} \rightarrow \text{bit} \rangle$

Experiments

- ▶ Wall Street Journal part-of-speech tagging data

Perceptron = 2.89% error, Log-linear tagger = 3.28% error

- ▶ [Ramshaw and Marcus, 1995] NP chunking data

Perceptron = 93.63% accuracy, Log-linear tagger = 93.29% accuracy

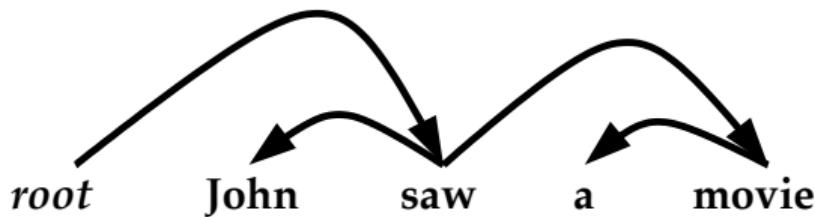
Global Linear Models Part 3: The Perceptron Algorithm for Dependency Parsing

Michael Collins, Columbia University

Overview

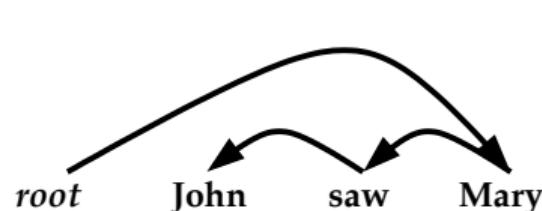
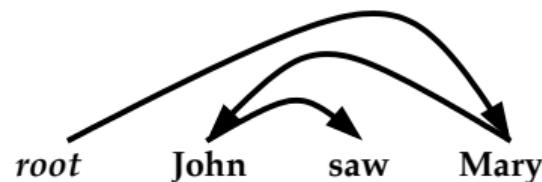
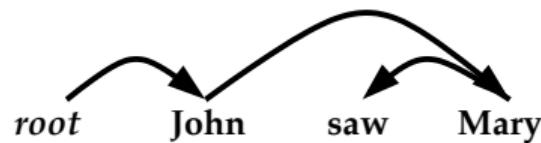
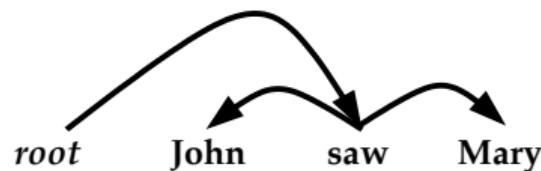
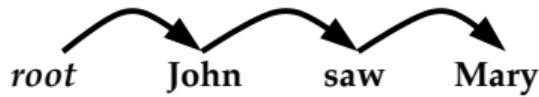
- ▶ Dependency parsing
- ▶ GLMs for dependency parsing
- ▶ Results from McDonald (2005)

Unlabeled Dependency Parses

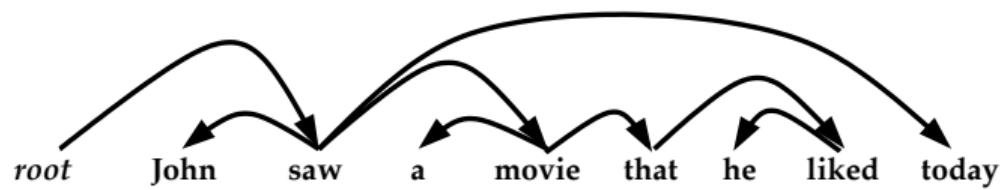


- ▶ root is a special *root* symbol
- ▶ Each dependency is a pair (h, m) where h is the index of a head word, m is the index of a modifier word. In the figures, we represent a dependency (h, m) by a directed edge from h to m .
- ▶ Dependencies in the above example are $(0, 2)$, $(2, 1)$, $(2, 4)$, and $(4, 3)$. (We take 0 to be the root symbol.)

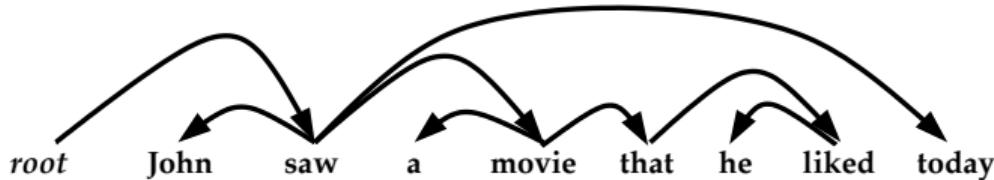
All Dependency Parses for *John saw Mary*



A More Complex Example



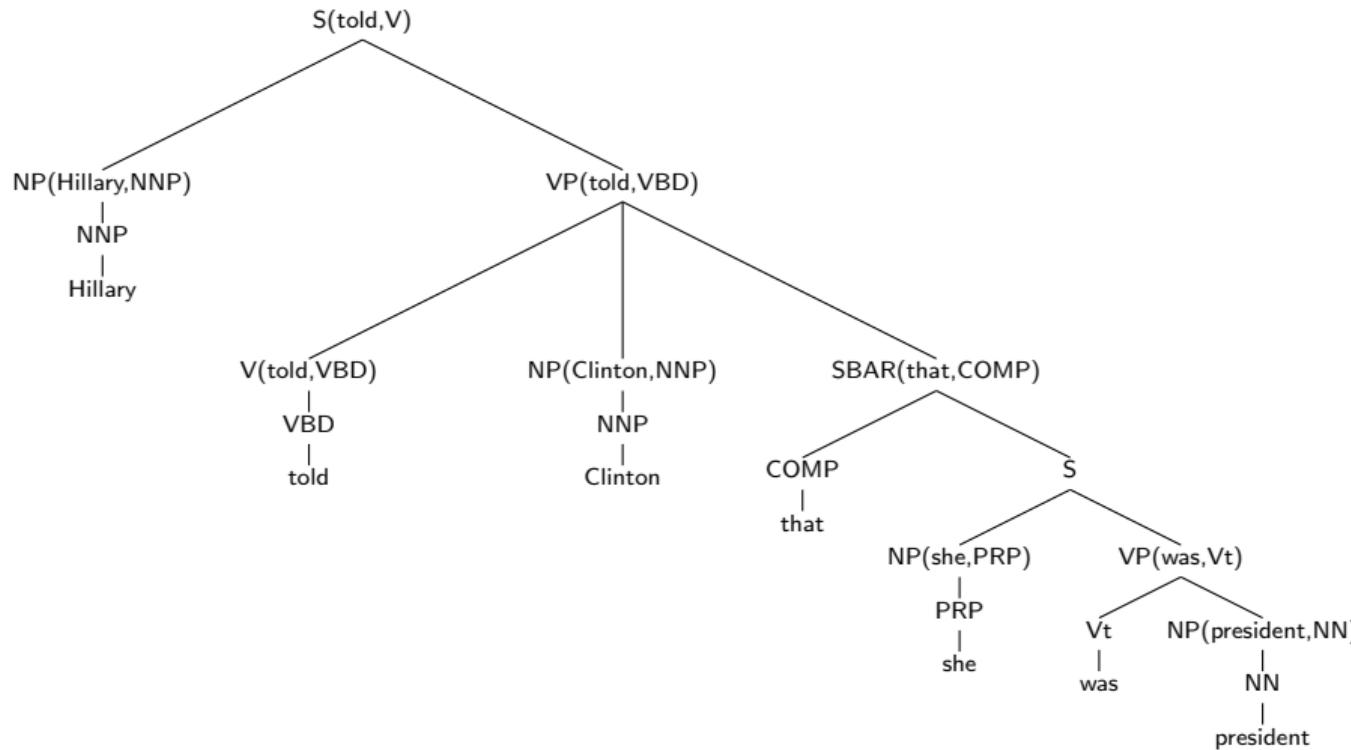
Conditions on Dependency Structures



- ▶ The dependency arcs form a *directed tree*, with the root symbol at the root of the tree.
(Definition: A directed tree rooted at *root* is a tree, where for every word *w* other than the root, there is a directed path from *root* to *w*.)
- ▶ There are no “crossing dependencies”.
Dependency structures with no crossing dependencies are sometimes referred to as **projective** structures.

Dependency Parsing Resources

- ▶ CoNLL 2006 conference had a “shared task” with dependency parsing of 12 languages (Arabic, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish, Turkish). 19 different groups developed dependency parsing systems. (See also CoNLL 2007).
- ▶ PhD thesis on the topic: Ryan McDonald, *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*, University of Pennsylvania.
- ▶ For some languages, e.g., Czech, there are “dependency banks” available which contain training data in the form of sentences paired with dependency structures
- ▶ For other languages, we can extract dependency structures from treebanks



Unlabeled Dependencies:

- | | | | |
|-------|----------------------|-------|-----------------------|
| (0,2) | (for root → told) | (4,6) | (for that → was) |
| (2,1) | (for told → Hillary) | (6,5) | (for was → she) |
| (2,3) | (for told → Clinton) | (6,7) | (for was → president) |
| (2,4) | (for told → that) | | |

Efficiency of Dependency Parsing

- ▶ PCFG parsing is $O(n^3G^3)$ where n is the length of the sentence, G is the number of non-terminals in the grammar
- ▶ Lexicalized PCFG parsing is $O(n^5G^3)$ where n is the length of the sentence, G is the number of non-terminals in the grammar.
- ▶ Unlabeled dependency parsing is $O(n^3)$.

Overview

- ▶ Dependency parsing
- ▶ Global Linear Models (GLMs) for dependency parsing
- ▶ Results from McDonald (2005)

GLMs for Dependency parsing

- ▶ x is a sentence
- ▶ $\text{GEN}(x)$ is set of all dependency structures for x
- ▶ $\mathbf{f}(x, y)$ is a feature vector for a sentence x paired with a dependency parse y

GLMs for Dependency parsing

- ▶ To run the perceptron algorithm, we must be able to efficiently calculate

$$\arg \max_{y \in \text{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y)$$

- ▶ Local feature vectors: define

$$\mathbf{f}(x, y) = \sum_{(h,m) \in y} \mathbf{g}(x, h, m)$$

where $\mathbf{g}(x, h, m)$ maps a sentence x and a dependency (h, m) to a local feature vector

- ▶ Can then use dynamic programming to calculate

$$\arg \max_{y \in \text{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y) = \arg \max_{y \in \text{GEN}(x)} \sum_{(h,m) \in y} \mathbf{w} \cdot \mathbf{g}(x, h, m)$$

Definition of Local Feature Vectors

- ▶ Features from McDonald et al. (2005):
 - ▶ Note: define w_i to be the i 'th word in the sentence, t_i to be the part-of-speech (POS) tag for the i 'th word.
 - ▶ *Unigram* features: Identity of w_h . Identity of w_m . Identity of t_h . Identity of t_m .
 - ▶ *Bigram* features: Identity of the 4-tuple $\langle w_h, w_m, t_h, t_m \rangle$. Identity of sub-sets of this 4-tuple, e.g., identity of the pair $\langle w_h, w_m \rangle$.
 - ▶ *Contextual features*: Identity of the 4-tuple $\langle t_h, t_{h+1}, t_{m-1}, t_m \rangle$. Similar features which consider t_{h-1} and t_{m+1} , giving 4 possible feature types.
 - ▶ *In-between features*: Identity of triples $\langle t_h, t, t_m \rangle$ for any tag t seen between words h and m .

Overview

- ▶ Dependency parsing
- ▶ Global Linear Models (GLMs) for dependency parsing
- ▶ Results from McDonald (2005)

Results from McDonald (2005)

Method	Accuracy
Collins (1997)	91.4%
1st order dependency	90.7%
2nd order dependency	91.5%

- ▶ Accuracy is percentage of correct unlabeled dependencies
- ▶ Collins (1997) is result from a lexicalized context-free parser, with dependencies extracted from the parser's output
- ▶ 1st order dependency is the method just described.
2nd order dependency is a model that uses richer representations.
- ▶ Advantages of the dependency parsing approaches: simplicity, efficiency ($O(n^3)$ parsing time).

Language Modeling

Michael Collins, Columbia University

Overview

- ▶ The language modeling problem
- ▶ Trigram models
- ▶ Evaluating language models: perplexity
- ▶ Estimation techniques:
 - ▶ Linear interpolation
 - ▶ Discounting methods

The Language Modeling Problem

- ▶ We have some (finite) vocabulary,
say $\mathcal{V} = \{\text{the, a, man, telescope, Beckham, two, ...}\}$
- ▶ We have an (infinite) set of strings, \mathcal{V}^\dagger

the STOP

a STOP

the fan STOP

the fan saw Beckham STOP

the fan saw saw STOP

the fan saw Beckham play for Real Madrid STOP

The Language Modeling Problem (Continued)

- ▶ We have a *training sample* of example sentences in English

The Language Modeling Problem (Continued)

- ▶ We have a *training sample* of example sentences in English
- ▶ We need to “learn” a probability distribution p
i.e., p is a function that satisfies

$$\sum_{x \in \mathcal{V}^\dagger} p(x) = 1, \quad p(x) \geq 0 \text{ for all } x \in \mathcal{V}^\dagger$$

The Language Modeling Problem (Continued)

- ▶ We have a *training sample* of example sentences in English
- ▶ We need to “learn” a probability distribution p
i.e., p is a function that satisfies

$$\sum_{x \in \mathcal{V}^\dagger} p(x) = 1, \quad p(x) \geq 0 \text{ for all } x \in \mathcal{V}^\dagger$$

$$p(\text{the STOP}) = 10^{-12}$$

$$p(\text{the fan STOP}) = 10^{-8}$$

$$p(\text{the fan saw Beckham STOP}) = 2 \times 10^{-8}$$

$$p(\text{the fan saw saw STOP}) = 10^{-15}$$

...

$$p(\text{the fan saw Beckham play for Real Madrid STOP}) = 2 \times 10^{-9}$$

...

Why on earth would we want to do this?!

- ▶ **Speech recognition** was the original motivation.
(Related problems are optical character recognition,
handwriting recognition.)

Why on earth would we want to do this?!

- ▶ **Speech recognition** was the original motivation.
(Related problems are optical character recognition,
handwriting recognition.)
- ▶ The estimation techniques developed for this problem will
be **VERY** useful for other problems in NLP

A Naive Method

- ▶ We have N training sentences
- ▶ For any sentence $x_1 \dots x_n$, $c(x_1 \dots x_n)$ is the number of times the sentence is seen in our training data
- ▶ A naive estimate:

$$p(x_1 \dots x_n) = \frac{c(x_1 \dots x_n)}{N}$$

Overview

- ▶ The language modeling problem
- ▶ Trigram models
- ▶ Evaluating language models: perplexity
- ▶ Estimation techniques:
 - ▶ Linear interpolation
 - ▶ Discounting methods

Markov Processes

- ▶ Consider a sequence of random variables X_1, X_2, \dots, X_n .
Each random variable can take any value in a finite set \mathcal{V} .
For now we assume the length n is fixed (e.g., $n = 100$).
- ▶ Our goal: model

$$P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

First-Order Markov Processes

$$P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

First-Order Markov Processes

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ = & P(X_1 = x_1) \prod_{i=2}^n P(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}) \end{aligned}$$

First-Order Markov Processes

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ &= P(X_1 = x_1) \prod_{i=2}^n P(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}) \\ &= P(X_1 = x_1) \prod_{i=2}^n P(X_i = x_i | X_{i-1} = x_{i-1}) \end{aligned}$$

First-Order Markov Processes

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ &= P(X_1 = x_1) \prod_{i=2}^n P(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}) \\ &= P(X_1 = x_1) \prod_{i=2}^n P(X_i = x_i | X_{i-1} = x_{i-1}) \end{aligned}$$

The first-order Markov assumption: For any $i \in \{2 \dots n\}$, for any $x_1 \dots x_i$,

$$P(X_i = x_i | X_1 = x_1 \dots X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

Second-Order Markov Processes

$$P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

Second-Order Markov Processes

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ = & P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \\ & \times \prod_{i=3}^n P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \end{aligned}$$

Second-Order Markov Processes

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ = & P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \\ & \times \prod_{i=3}^n P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \\ = & \prod_{i=1}^n P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \end{aligned}$$

(For convenience we assume $x_0 = x_{-1} = *$, where * is a special “start” symbol.)

Modeling Variable Length Sequences

- ▶ We would like the length of the sequence, n , to also be a random variable
- ▶ A simple solution: always define $X_n = \text{STOP}$ where STOP is a special symbol

Modeling Variable Length Sequences

- ▶ We would like the length of the sequence, n , to also be a random variable
- ▶ A simple solution: always define $X_n = \text{STOP}$ where STOP is a special symbol
- ▶ Then use a Markov process as before:

$$\begin{aligned} & P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) \\ &= \prod_{i=1}^n P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \end{aligned}$$

(For convenience we assume $x_0 = x_{-1} = *$, where * is a special “start” symbol.)

Trigram Language Models

- ▶ A trigram language model consists of:
 1. A finite set \mathcal{V}
 2. A parameter $q(w|u, v)$ for each trigram u, v, w such that $w \in \mathcal{V} \cup \{\text{STOP}\}$, and $u, v \in \mathcal{V} \cup \{*\}$.

Trigram Language Models

- ▶ A trigram language model consists of:
 1. A finite set \mathcal{V}
 2. A parameter $q(w|u, v)$ for each trigram u, v, w such that $w \in \mathcal{V} \cup \{\text{STOP}\}$, and $u, v \in \mathcal{V} \cup \{*\}$.
- ▶ For any sentence $x_1 \dots x_n$ where $x_i \in \mathcal{V}$ for $i = 1 \dots (n - 1)$, and $x_n = \text{STOP}$, the probability of the sentence under the trigram language model is

$$p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-2}, x_{i-1})$$

where we define $x_0 = x_{-1} = *$.

An Example

For the sentence

the dog barks STOP

we would have

$$\begin{aligned} p(\text{the dog barks STOP}) &= q(\text{the}|*, *) \\ &\quad \times q(\text{dog}|*, \text{the}) \\ &\quad \times q(\text{barks}|\text{the}, \text{dog}) \\ &\quad \times q(\text{STOP}|\text{dog}, \text{barks}) \end{aligned}$$

The Trigram Estimation Problem

Remaining estimation problem:

$$q(w_i \mid w_{i-2}, w_{i-1})$$

For example:

$$q(\text{laughs} \mid \text{the, dog})$$

The Trigram Estimation Problem

Remaining estimation problem:

$$q(w_i \mid w_{i-2}, w_{i-1})$$

For example:

$$q(\text{laughs} \mid \text{the, dog})$$

A natural estimate (the “maximum likelihood estimate”):

$$q(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

$$q(\text{laughs} \mid \text{the, dog}) = \frac{\text{Count}(\text{the, dog, laughs})}{\text{Count}(\text{the, dog})}$$

Sparse Data Problems

A natural estimate (the “maximum likelihood estimate”):

$$q(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

$$q(\text{laughs} \mid \text{the, dog}) = \frac{\text{Count}(\text{the, dog, laughs})}{\text{Count}(\text{the, dog})}$$

Say our vocabulary size is $N = |\mathcal{V}|$, then there are N^3 parameters in the model.

e.g., $N = 20,000 \Rightarrow 20,000^3 = 8 \times 10^{12}$ parameters

Overview

- ▶ The language modeling problem
- ▶ Trigram models
- ▶ Evaluating language models: perplexity
- ▶ Estimation techniques:
 - ▶ Linear interpolation
 - ▶ Discounting methods

Evaluating a Language Model: Perplexity

- ▶ We have some test data, m sentences

$$s_1, s_2, s_3, \dots, s_m$$

Evaluating a Language Model: Perplexity

- ▶ We have some test data, m sentences

$$s_1, s_2, s_3, \dots, s_m$$

- ▶ We could look at the probability under our model $\prod_{i=1}^m p(s_i)$. Or more conveniently, the *log probability*

$$\log \prod_{i=1}^m p(s_i) = \sum_{i=1}^m \log p(s_i)$$

Evaluating a Language Model: Perplexity

- ▶ We have some test data, m sentences

$$s_1, s_2, s_3, \dots, s_m$$

- ▶ We could look at the probability under our model $\prod_{i=1}^m p(s_i)$. Or more conveniently, the *log probability*

$$\log \prod_{i=1}^m p(s_i) = \sum_{i=1}^m \log p(s_i)$$

- ▶ In fact the usual evaluation measure is *perplexity*

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \frac{1}{M} \sum_{i=1}^m \log p(s_i)$$

and M is the total number of words in the test data.

Some Intuition about Perplexity

- ▶ Say we have a vocabulary \mathcal{V} , and $N = |\mathcal{V}| + 1$ and model that predicts

$$q(w|u, v) = \frac{1}{N}$$

for all $w \in \mathcal{V} \cup \{\text{STOP}\}$, for all $u, v \in \mathcal{V} \cup \{*\}$.

- ▶ Easy to calculate the perplexity in this case:

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \log \frac{1}{N}$$

⇒

$$\text{Perplexity} = N$$

Perplexity is a measure of effective “branching factor”

Typical Values of Perplexity

- ▶ Results from Goodman (“A bit of progress in language modeling”), where $|\mathcal{V}| = 50,000$
- ▶ A trigram model: $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-2}, x_{i-1})$.
Perplexity = 74

Typical Values of Perplexity

- ▶ Results from Goodman (“A bit of progress in language modeling”), where $|\mathcal{V}| = 50,000$
- ▶ A trigram model: $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-2}, x_{i-1})$.
Perplexity = 74
- ▶ A bigram model: $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-1})$.
Perplexity = 137

Typical Values of Perplexity

- ▶ Results from Goodman (“A bit of progress in language modeling”), where $|\mathcal{V}| = 50,000$
- ▶ A trigram model: $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-2}, x_{i-1})$.
Perplexity = 74
- ▶ A bigram model: $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-1})$.
Perplexity = 137
- ▶ A unigram model: $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i)$.
Perplexity = 955

Some History

- ▶ Shannon conducted experiments on entropy of English i.e., how good are people at the perplexity game?
C. Shannon. Prediction and entropy of printed English. Bell Systems Technical Journal, 30:50–64, 1951.

Some History

Chomsky (in *Syntactic Structures* (1957)):

Second, the notion “grammatical” cannot be identified with “meaningful” or “significant” in any semantic sense.

Sentences (1) and (2) are equally nonsensical, but any speaker of English will recognize that only the former is grammatical.

(1) *Colorless green ideas sleep furiously.*

(2) *Furiously sleep ideas green colorless.*

...

... Third, the notion “grammatical in English” cannot be identified in any way with the notion “high order of statistical approximation to English”. It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out on identical grounds as equally ‘remote’ from English. Yet (1), though nonsensical, is grammatical, while (2) is not. ...

Overview

- ▶ The language modeling problem
- ▶ Trigram models
- ▶ Evaluating language models: perplexity
- ▶ Estimation techniques:
 - ▶ Linear interpolation
 - ▶ Discounting methods

Sparse Data Problems

A natural estimate (the “maximum likelihood estimate”):

$$q(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

$$q(\text{laughs} \mid \text{the, dog}) = \frac{\text{Count}(\text{the, dog, laughs})}{\text{Count}(\text{the, dog})}$$

Say our vocabulary size is $N = |\mathcal{V}|$, then there are N^3 parameters in the model.

e.g., $N = 20,000 \Rightarrow 20,000^3 = 8 \times 10^{12}$ parameters

The Bias-Variance Trade-Off

- ▶ Trigram maximum-likelihood estimate

$$q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

- ▶ Bigram maximum-likelihood estimate

$$q_{\text{ML}}(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i)}{\text{Count}(w_{i-1})}$$

- ▶ Unigram maximum-likelihood estimate

$$q_{\text{ML}}(w_i) = \frac{\text{Count}(w_i)}{\text{Count}()}$$

Linear Interpolation

- ▶ Take our estimate $q(w_i \mid w_{i-2}, w_{i-1})$ to be

$$\begin{aligned} q(w_i \mid w_{i-2}, w_{i-1}) = & \lambda_1 \times q_{\text{ML}}(w_i \mid w_{i-2}, w_{i-1}) \\ & + \lambda_2 \times q_{\text{ML}}(w_i \mid w_{i-1}) \\ & + \lambda_3 \times q_{\text{ML}}(w_i) \end{aligned}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i .

Linear Interpolation (continued)

Our estimate correctly defines a distribution (define $\mathcal{V}' = \mathcal{V} \cup \{\text{STOP}\}$):

$$\sum_{w \in \mathcal{V}'} q(w | u, v)$$

Linear Interpolation (continued)

Our estimate correctly defines a distribution (define $\mathcal{V}' = \mathcal{V} \cup \{\text{STOP}\}$):

$$\sum_{w \in \mathcal{V}'} q(w | u, v)$$

$$= \sum_{w \in \mathcal{V}'} [\lambda_1 \times q_{\text{ML}}(w | u, v) + \lambda_2 \times q_{\text{ML}}(w | v) + \lambda_3 \times q_{\text{ML}}(w)]$$

Linear Interpolation (continued)

Our estimate correctly defines a distribution (define $\mathcal{V}' = \mathcal{V} \cup \{\text{STOP}\}$):

$$\begin{aligned}& \sum_{w \in \mathcal{V}'} q(w | u, v) \\&= \sum_{w \in \mathcal{V}'} [\lambda_1 \times q_{\text{ML}}(w | u, v) + \lambda_2 \times q_{\text{ML}}(w | v) + \lambda_3 \times q_{\text{ML}}(w)] \\&= \lambda_1 \sum_w q_{\text{ML}}(w | u, v) + \lambda_2 \sum_w q_{\text{ML}}(w | v) + \lambda_3 \sum_w q_{\text{ML}}(w)\end{aligned}$$

Linear Interpolation (continued)

Our estimate correctly defines a distribution (define $\mathcal{V}' = \mathcal{V} \cup \{\text{STOP}\}$):

$$\begin{aligned}& \sum_{w \in \mathcal{V}'} q(w | u, v) \\&= \sum_{w \in \mathcal{V}'} [\lambda_1 \times q_{\text{ML}}(w | u, v) + \lambda_2 \times q_{\text{ML}}(w | v) + \lambda_3 \times q_{\text{ML}}(w)] \\&= \lambda_1 \sum_w q_{\text{ML}}(w | u, v) + \lambda_2 \sum_w q_{\text{ML}}(w | v) + \lambda_3 \sum_w q_{\text{ML}}(w) \\&= \lambda_1 + \lambda_2 + \lambda_3\end{aligned}$$

Linear Interpolation (continued)

Our estimate correctly defines a distribution (define $\mathcal{V}' = \mathcal{V} \cup \{\text{STOP}\}$):

$$\begin{aligned}& \sum_{w \in \mathcal{V}'} q(w | u, v) \\&= \sum_{w \in \mathcal{V}'} [\lambda_1 \times q_{\text{ML}}(w | u, v) + \lambda_2 \times q_{\text{ML}}(w | v) + \lambda_3 \times q_{\text{ML}}(w)] \\&= \lambda_1 \sum_w q_{\text{ML}}(w | u, v) + \lambda_2 \sum_w q_{\text{ML}}(w | v) + \lambda_3 \sum_w q_{\text{ML}}(w) \\&= \lambda_1 + \lambda_2 + \lambda_3 \\&= 1\end{aligned}$$

Linear Interpolation (continued)

Our estimate correctly defines a distribution (define $\mathcal{V}' = \mathcal{V} \cup \{\text{STOP}\}$):

$$\begin{aligned}& \sum_{w \in \mathcal{V}'} q(w | u, v) \\&= \sum_{w \in \mathcal{V}'} [\lambda_1 \times q_{\text{ML}}(w | u, v) + \lambda_2 \times q_{\text{ML}}(w | v) + \lambda_3 \times q_{\text{ML}}(w)] \\&= \lambda_1 \sum_w q_{\text{ML}}(w | u, v) + \lambda_2 \sum_w q_{\text{ML}}(w | v) + \lambda_3 \sum_w q_{\text{ML}}(w) \\&= \lambda_1 + \lambda_2 + \lambda_3 \\&= 1\end{aligned}$$

(Can show also that $q(w | u, v) \geq 0$ for all $w \in \mathcal{V}'$)

How to estimate the λ values?

- ▶ Hold out part of training set as “validation” data

How to estimate the λ values?

- ▶ Hold out part of training set as “validation” data
- ▶ Define $c'(w_1, w_2, w_3)$ to be the number of times the trigram (w_1, w_2, w_3) is seen in validation set

How to estimate the λ values?

- ▶ Hold out part of training set as “validation” data
- ▶ Define $c'(w_1, w_2, w_3)$ to be the number of times the trigram (w_1, w_2, w_3) is seen in validation set
- ▶ Choose $\lambda_1, \lambda_2, \lambda_3$ to maximize:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3} c'(w_1, w_2, w_3) \log q(w_3 | w_1, w_2)$$

such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i , and where

$$\begin{aligned} q(w_i | w_{i-2}, w_{i-1}) = & \lambda_1 \times q_{\text{ML}}(w_i | w_{i-2}, w_{i-1}) \\ & + \lambda_2 \times q_{\text{ML}}(w_i | w_{i-1}) \\ & + \lambda_3 \times q_{\text{ML}}(w_i) \end{aligned}$$

Allowing the λ 's to vary

- Take a function Π that partitions histories
e.g.,

$$\Pi(w_{i-2}, w_{i-1}) = \begin{cases} 1 & \text{If } \text{Count}(w_{i-1}, w_{i-2}) = 0 \\ 2 & \text{If } 1 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 2 \\ 3 & \text{If } 3 \leq \text{Count}(w_{i-1}, w_{i-2}) \leq 5 \\ 4 & \text{Otherwise} \end{cases}$$

- Introduce a dependence of the λ 's on the partition:

$$q(w_i | w_{i-2}, w_{i-1}) = \lambda_1^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i | w_{i-2}, w_{i-1}) + \lambda_2^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i | w_{i-1}) + \lambda_3^{\Pi(w_{i-2}, w_{i-1})} \times q_{\text{ML}}(w_i)$$

where $\lambda_1^{\Pi(w_{i-2}, w_{i-1})} + \lambda_2^{\Pi(w_{i-2}, w_{i-1})} + \lambda_3^{\Pi(w_{i-2}, w_{i-1})} = 1$,
and $\lambda_i^{\Pi(w_{i-2}, w_{i-1})} \geq 0$ for all i .

Overview

- ▶ The language modeling problem
- ▶ Trigram models
- ▶ Evaluating language models: perplexity
- ▶ Estimation techniques:
 - ▶ Linear interpolation
 - ▶ Discounting methods

Discounting Methods

- ▶ Say we've seen the following counts:

x	Count(x)	$q_{\text{ML}}(w_i \mid w_{i-1})$
the	48	
the, dog	15	15/48
the, woman	11	11/48
the, man	10	10/48
the, park	5	5/48
the, job	2	2/48
the, telescope	1	1/48
the, manual	1	1/48
the, afternoon	1	1/48
the, country	1	1/48
the, street	1	1/48

- ▶ The maximum-likelihood estimates are high
(particularly for low count items)

Discounting Methods

- ▶ Now define “discounted” counts,

$$\text{Count}^*(x) = \text{Count}(x) - 0.5$$

- ▶ New estimates:

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(\text{the})}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Discounting Methods (Continued)

- We now have some “missing probability mass”:

$$\alpha(w_{i-1}) = 1 - \sum_w \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

e.g., in our example, $\alpha(\text{the}) = 10 \times 0.5 / 48 = 5/48$

Katz Back-Off Models (Bigrams)

- ▶ For a bigram model, define two sets

$$\begin{aligned}\mathcal{A}(w_{i-1}) &= \{w : \text{Count}(w_{i-1}, w) > 0\} \\ \mathcal{B}(w_{i-1}) &= \{w : \text{Count}(w_{i-1}, w) = 0\}\end{aligned}$$

- ▶ A bigram model

$$q_{BO}(w_i \mid w_{i-1}) = \begin{cases} \frac{\text{Count}^*(w_{i-1}, w_i)}{\text{Count}(w_{i-1})} & \text{If } w_i \in \mathcal{A}(w_{i-1}) \\ \alpha(w_{i-1}) \frac{q_{ML}(w_i)}{\sum_{w \in \mathcal{B}(w_{i-1})} q_{ML}(w)} & \text{If } w_i \in \mathcal{B}(w_{i-1}) \end{cases}$$

where

$$\alpha(w_{i-1}) = 1 - \sum_{w \in \mathcal{A}(w_{i-1})} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

Katz Back-Off Models (Trigrams)

- ▶ For a trigram model, first define two sets

$$\mathcal{A}(w_{i-2}, w_{i-1}) = \{w : \text{Count}(w_{i-2}, w_{i-1}, w) > 0\}$$

$$\mathcal{B}(w_{i-2}, w_{i-1}) = \{w : \text{Count}(w_{i-2}, w_{i-1}, w) = 0\}$$

- ▶ A trigram model is defined in terms of the bigram model:

$$q_{BO}(w_i | w_{i-2}, w_{i-1}) = \begin{cases} \frac{\text{Count}^*(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})} & \text{If } w_i \in \mathcal{A}(w_{i-2}, w_{i-1}) \\ \frac{\alpha(w_{i-2}, w_{i-1}) q_{BO}(w_i | w_{i-1})}{\sum_{w \in \mathcal{B}(w_{i-2}, w_{i-1})} q_{BO}(w | w_{i-1})} & \text{If } w_i \in \mathcal{B}(w_{i-2}, w_{i-1}) \end{cases}$$

where

$$\alpha(w_{i-2}, w_{i-1}) = 1 - \sum_{w \in \mathcal{A}(w_{i-2}, w_{i-1})} \frac{\text{Count}^*(w_{i-2}, w_{i-1}, w)}{\text{Count}(w_{i-2}, w_{i-1})}$$

Summary

- ▶ Three steps in deriving the language model probabilities:
 1. Expand $p(w_1, w_2 \dots w_n)$ using **Chain rule**.
 2. Make **Markov Independence Assumptions**
$$p(w_i | w_1, w_2 \dots w_{i-2}, w_{i-1}) = p(w_i | w_{i-2}, w_{i-1})$$
 3. **Smooth** the estimates using low order counts.
- ▶ Other methods used to improve language models:
 - ▶ “Topic” or “long-range” features.
 - ▶ Syntactic models.

It's generally hard to improve on trigram models though!!

Log-Linear Models

Michael Collins, Columbia University

The Language Modeling Problem

- ▶ w_i is the i 'th word in a document
- ▶ Estimate a distribution $p(w_i|w_1, w_2, \dots, w_{i-1})$ given previous “history” w_1, \dots, w_{i-1} .
- ▶ E.g., $w_1, \dots, w_{i-1} =$

Third, the notion “grammatical in English” cannot be identified in any way with the notion “high order of statistical approximation to English”. It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical

Trigram Models

- ▶ Estimate a distribution $p(w_i|w_1, w_2, \dots, w_{i-1})$ given previous “history” $w_1, \dots, w_{i-1} =$

Third, the notion “grammatical in English” cannot be identified in any way with the notion “high order of statistical approximation to English”. It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical

- ▶ **Trigram estimates:**

$$\begin{aligned} q(\text{model}|w_1, \dots, w_{i-1}) &= \lambda_1 q_{ML}(\text{model}|w_{i-2} = \text{any}, w_{i-1} = \text{statistical}) + \\ &\quad \lambda_2 q_{ML}(\text{model}|w_{i-1} = \text{statistical}) + \\ &\quad \lambda_3 q_{ML}(\text{model}) \end{aligned}$$

where $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$, $q_{ML}(y|x) = \frac{\text{Count}(x,y)}{\text{Count}(x)}$

Trigram Models

$$\begin{aligned} q(\text{model} | w_1, \dots, w_{i-1}) &= \lambda_1 q_{ML}(\text{model} | w_{i-2} = \text{any}, w_{i-1} = \text{statistical}) + \\ &\quad \lambda_2 q_{ML}(\text{model} | w_{i-1} = \text{statistical}) + \\ &\quad \lambda_3 q_{ML}(\text{model}) \end{aligned}$$

- ▶ Makes use of only bigram, trigram, unigram estimates
- ▶ Many other “features” of w_1, \dots, w_{i-1} may be useful, e.g.,:

$q_{ML}(\text{model} \mid w_{i-2} = \text{any})$

$q_{ML}(\text{model} \mid w_{i-1} \text{ is an adjective})$

$q_{ML}(\text{model} \mid w_{i-1} \text{ ends in "ical"})$

$q_{ML}(\text{model} \mid \text{author} = \text{Chomsky})$

$q_{ML}(\text{model} \mid \text{"model" does not occur somewhere in } w_1, \dots, w_{i-1})$

$q_{ML}(\text{model} \mid \text{"grammatical" occurs somewhere in } w_1, \dots, w_{i-1})$

A Naive Approach

$$\begin{aligned} q(\text{model} | w_1, \dots, w_{i-1}) = \\ \lambda_1 q_{ML}(\text{model} | w_{i-2} = \text{any}, w_{i-1} = \text{statistical}) + \\ \lambda_2 q_{ML}(\text{model} | w_{i-1} = \text{statistical}) + \\ \lambda_3 q_{ML}(\text{model}) + \\ \lambda_4 q_{ML}(\text{model} | w_{i-2} = \text{any}) + \\ \lambda_5 q_{ML}(\text{model} | w_{i-1} \text{ is an adjective}) + \\ \lambda_6 q_{ML}(\text{model} | w_{i-1} \text{ ends in "ical"}) + \\ \lambda_7 q_{ML}(\text{model} | \text{author} = \text{Chomsky}) + \\ \lambda_8 q_{ML}(\text{model} | \text{"model" does not occur somewhere in } w_1, \dots, w_{i-1}) + \\ \lambda_9 q_{ML}(\text{model} | \text{"grammatical" occurs somewhere in } w_1, \dots, w_{i-1}) \end{aligned}$$

This quickly becomes very unwieldy...

A Second Example: Part-of-Speech Tagging

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street,
as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V
forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N
Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

N = Noun

V = Verb

P = Preposition

Adv = Adverb

Adj = Adjective

...

A Second Example: Part-of-Speech Tagging

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ
base/?? from which Spain expanded its empire into the rest of the
Western Hemisphere .

- There are many possible tags in the position ??
 $\{\text{NN, NNS, Vt, Vi, IN, DT, ...}\}$
- The task: model the distribution

$$p(t_i | t_1, \dots, t_{i-1}, w_1 \dots w_n)$$

where t_i is the i 'th tag in the sequence, w_i is the i 'th word

A Second Example: Part-of-Speech Tagging

Hispaniola/**NNP** quickly/**RB** became/**VB** an/**DT** important/**JJ**
base/**??** from which Spain expanded its empire into the rest of the
Western Hemisphere .

- The task: model the distribution

$$p(t_i | t_1, \dots, t_{i-1}, w_1 \dots w_n)$$

where t_i is the i 'th tag in the sequence, w_i is the i 'th word

- Again: many “features” of $t_1, \dots, t_{i-1}, w_1 \dots w_n$ may be relevant

$$q_{ML}(\text{NN} \mid w_i = \text{base})$$

$$q_{ML}(\text{NN} \mid t_{i-1} \text{ is JJ})$$

$$q_{ML}(\text{NN} \mid w_i \text{ ends in "e"})$$

$$q_{ML}(\text{NN} \mid w_i \text{ ends in "se"})$$

$$q_{ML}(\text{NN} \mid w_{i-1} \text{ is "important"})$$

$$q_{ML}(\text{NN} \mid w_{i+1} \text{ is "from"})$$

Overview

- ▶ Log-linear models
- ▶ Parameter estimation in log-linear models
- ▶ Smoothing/regularization in log-linear models

The General Problem

- ▶ We have some **input domain** \mathcal{X}
- ▶ Have a finite **label set** \mathcal{Y}
- ▶ Aim is to provide a **conditional probability** $p(y | x)$ for any x, y where $x \in \mathcal{X}, y \in \mathcal{Y}$

Language Modeling

- ▶ x is a “history” w_1, w_2, \dots, w_{i-1} , e.g.,

Third, the notion “grammatical in English” cannot be identified in any way with the notion “high order of statistical approximation to English”.

It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse.

Hence, in any statistical

- ▶ y is an “outcome” w_i

Feature Vector Representations

- ▶ Aim is to provide a conditional probability $p(y | x)$ for “decision” y given “history” x
- ▶ A **feature** is a function $f_k(x, y) \in \mathbb{R}$
(Often **binary features** or **indicator functions**
 $f_k(x, y) \in \{0, 1\}\text{).}$
- ▶ Say we have m features f_k for $k = 1 \dots m$
⇒ A **feature vector** $f(x, y) \in \mathbb{R}^m$ for any x, y

Language Modeling

- ▶ x is a “history” w_1, w_2, \dots, w_{i-1} , e.g.,

Third, the notion “grammatical in English” cannot be identified in any way with the notion “high order of statistical approximation to English”.

It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse.

Hence, in any statistical

- ▶ y is an “outcome” w_i

- ▶ Example features:

$$f_1(x, y) = \begin{cases} 1 & \text{if } y = \text{model} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x, y) = \begin{cases} 1 & \text{if } y = \text{model} \text{ and } w_{i-1} = \text{statistical} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, w_{i-2} = \text{any}, w_{i-1} = \text{statistical} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, w_{i-2} = \text{any} \\ 0 & \text{otherwise} \end{cases}$$

$$f_5(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, w_{i-1} \text{ is an adjective} \\ 0 & \text{otherwise} \end{cases}$$

$$f_6(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, w_{i-1} \text{ ends in "ical"} \\ 0 & \text{otherwise} \end{cases}$$

$$f_7(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, \text{author} = \text{Chomsky} \\ 0 & \text{otherwise} \end{cases}$$

$$f_8(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, \text{"model" is not in } w_1, \dots, w_{i-1} \\ 0 & \text{otherwise} \end{cases}$$

$$f_9(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, \text{"grammatical" is in } w_1, \dots, w_{i-1} \\ 0 & \text{otherwise} \end{cases}$$

Defining Features in Practice

- ▶ We had the following “trigram” feature:

$$f_3(x, y) = \begin{cases} 1 & \text{if } y = \text{model}, w_{i-2} = \text{any}, w_{i-1} = \text{statistical} \\ 0 & \text{otherwise} \end{cases}$$

- ▶ In practice, we would probably introduce one trigram feature for every trigram seen in the training data: i.e., for all trigrams (u, v, w) seen in training data, create a feature

$$f_{N(u,v,w)}(x, y) = \begin{cases} 1 & \text{if } y = w, w_{i-2} = u, w_{i-1} = v \\ 0 & \text{otherwise} \end{cases}$$

where $N(u, v, w)$ is a function that maps each (u, v, w) trigram to a different integer

The POS-Tagging Example

- ▶ Each x is a “history” of the form $\langle t_1, t_2, \dots, t_{i-1}, w_1 \dots w_n, i \rangle$
 - ▶ Each y is a POS tag, such as NN, NNS, Vt, Vi, IN, DT, ...
 - ▶ We have m features $f_k(x, y)$ for $k = 1 \dots m$
-

For example:

$$f_1(\textcolor{green}{x}, \textcolor{red}{y}) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } y = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(\textcolor{green}{x}, \textcolor{red}{y}) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in ing and } y = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

...

The Full Set of Features in Ratnaparkhi, 1996

- ▶ Word/tag features for all word/tag pairs, e.g.,

$$f_{100}(x, y) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } y = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

- ▶ Spelling features for all prefixes/suffixes of length ≤ 4 , e.g.,

$$f_{101}(x, y) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in ing and } y = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{102}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ starts with pre and } y = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

The Full Set of Features in Ratnaparkhi, 1996

- Contextual Features, e.g.,

$$f_{103}(x, y) = \begin{cases} 1 & \text{if } \langle t_{i-2}, t_{i-1}, y \rangle = \langle \text{DT}, \text{JJ}, \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{104}(x, y) = \begin{cases} 1 & \text{if } \langle t_{i-1}, y \rangle = \langle \text{JJ}, \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{105}(x, y) = \begin{cases} 1 & \text{if } \langle y \rangle = \langle \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{106}(x, y) = \begin{cases} 1 & \text{if previous word } w_{i-1} = \text{the and } y = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{107}(x, y) = \begin{cases} 1 & \text{if next word } w_{i+1} = \text{the and } y = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

The Final Result

- ▶ We can come up with practically any questions (*features*) regarding history/tag pairs.
- ▶ For a given history $x \in \mathcal{X}$, each label in \mathcal{Y} is mapped to a different feature vector

$$\begin{aligned} f(\langle \text{JJ}, \text{DT}, \langle \text{Hispaniola}, \dots \rangle, 6 \rangle, \text{Vt}) &= 1001011001001100110 \\ f(\langle \text{JJ}, \text{DT}, \langle \text{Hispaniola}, \dots \rangle, 6 \rangle, \text{JJ}) &= 0110010101011110010 \\ f(\langle \text{JJ}, \text{DT}, \langle \text{Hispaniola}, \dots \rangle, 6 \rangle, \text{NN}) &= 0001111101001100100 \\ f(\langle \text{JJ}, \text{DT}, \langle \text{Hispaniola}, \dots \rangle, 6 \rangle, \text{IN}) &= 0001011011000000010 \end{aligned}$$

...

Parameter Vectors

- ▶ Given features $f_k(x, y)$ for $k = 1 \dots m$,
also define a **parameter vector** $v \in \mathbb{R}^m$
- ▶ Each (x, y) pair is then mapped to a “score”

$$v \cdot f(x, y) = \sum_k v_k f_k(x, y)$$

Language Modeling

- ▶ x is a “history” w_1, w_2, \dots, w_{i-1} , e.g.,
Third, the notion “grammatical in English” cannot be identified in any way with the notion “high order of statistical approximation to English”. It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse.
Hence, in any statistical
- ▶ Each possible y gets a different score:

$$v \cdot f(x, \text{model}) = 5.6 \qquad v \cdot f(x, \text{the}) = -3.2$$

$$v \cdot f(x, \text{is}) = 1.5 \qquad v \cdot f(x, \text{of}) = 1.3$$

$$v \cdot f(x, \text{models}) = 4.5 \qquad \dots$$

Log-Linear Models

- ▶ We have some input domain \mathcal{X} , and a finite label set \mathcal{Y} . Aim is to provide a conditional probability $p(y | x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- ▶ A feature is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$
(Often binary features or indicator functions
 $f_k : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$).
- ▶ Say we have m features f_k for $k = 1 \dots m$
⇒ A feature vector $f(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- ▶ We also have a **parameter vector** $v \in \mathbb{R}^m$
- ▶ We define

$$p(y | x; v) = \frac{e^{v \cdot f(x, y)}}{\sum_{y' \in \mathcal{Y}} e^{v \cdot f(x, y')}}$$

Why the name?

$$\log p(y \mid x; v) = \underbrace{v \cdot f(x, y)}_{\text{Linear term}} - \underbrace{\log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x, y')}}_{\text{Normalization term}}$$

Overview

- ▶ Log-linear models
- ▶ Parameter estimation in log-linear models
- ▶ Smoothing/regularization in log-linear models

Maximum-Likelihood Estimation

- ▶ Maximum-likelihood estimates given training sample $(x^{(i)}, y^{(i)})$ for $i = 1 \dots n$, each $(x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$:

$$v_{ML} = \operatorname{argmax}_{v \in \mathbb{R}^m} L(v)$$

where

$$L(v) = \sum_{i=1}^n \log p(y^{(i)} \mid x^{(i)}; v) = \sum_{i=1}^n v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')}$$

Calculating the Maximum-Likelihood Estimates

- ▶ Need to maximize:

$$L(v) = \sum_{i=1}^n v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')}$$

- ▶ Calculating gradients:

$$\begin{aligned}\frac{dL(v)}{dv_k} &= \sum_{i=1}^n f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \frac{\sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') e^{v \cdot f(x^{(i)}, y')}}{\sum_{z' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, z')}} \\ &= \sum_{i=1}^n f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \frac{e^{v \cdot f(x^{(i)}, y')}}{\sum_{z' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, z')}} \\ &= \underbrace{\sum_{i=1}^n f_k(x^{(i)}, y^{(i)})}_{\text{Empirical counts}} - \underbrace{\sum_{i=1}^n \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' \mid x^{(i)}; v)}_{\text{Expected counts}}\end{aligned}$$

Gradient Ascent Methods

- ▶ Need to maximize $L(v)$ where

$$\frac{dL(v)}{dv} = \sum_{i=1}^n f(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \sum_{y' \in \mathcal{Y}} f(x^{(i)}, y') p(y' | x^{(i)}; v)$$

Initialization: $v = 0$

Iterate until convergence:

- ▶ Calculate $\Delta = \frac{dL(v)}{dv}$
- ▶ Calculate $\beta_* = \operatorname{argmax}_\beta L(v + \beta \Delta)$ (Line Search)
- ▶ Set $v \leftarrow v + \beta_* \Delta$

Conjugate Gradient Methods

- ▶ (Vanilla) gradient ascent can be very slow
- ▶ Conjugate gradient methods require calculation of gradient at each iteration, but do a line search in **a direction which is a function of the current gradient, and the previous step taken.**
- ▶ Conjugate gradient packages are widely available
In general: they require a function

$$\text{calc_gradient}(v) \rightarrow \left(L(v), \frac{dL(v)}{dv} \right)$$

and that's about it!

Overview

- ▶ Log-linear models
- ▶ Parameter estimation in log-linear models
- ▶ Smoothing/regularization in log-linear models

Smoothing in Log-Linear Models

- ▶ Say we have a feature:

$$f_{100}(x, y) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } y = Vt \\ 0 & \text{otherwise} \end{cases}$$

- ▶ In training data, base is seen 3 times, with Vt every time
- ▶ Maximum likelihood solution satisfies

$$\sum_i f_{100}(x^{(i)}, y^{(i)}) = \sum_i \sum_y p(y | x^{(i)}; v) f_{100}(x^{(i)}, y)$$

- $\Rightarrow p(Vt | x^{(i)}; v) = 1$ for any history $x^{(i)}$ where $w_i = \text{base}$
- $\Rightarrow v_{100} \rightarrow \infty$ at maximum-likelihood solution (most likely)
- $\Rightarrow p(Vt | x; v) = 1$ for any test data history x where $w = \text{base}$

Regularization

- ▶ Modified loss function

$$L(v) = \sum_{i=1}^n v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^n \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')} - \frac{\lambda}{2} \sum_{k=1}^m v_k^2$$

- ▶ Calculating gradients:

$$\frac{dL(v)}{dv_k} = \underbrace{\sum_{i=1}^n f_k(x^{(i)}, y^{(i)})}_{\text{Empirical counts}} - \underbrace{\sum_{i=1}^n \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' | x^{(i)}; v)}_{\text{Expected counts}} - \lambda v_k$$

- ▶ Can run conjugate gradient methods as before
- ▶ Adds a penalty for large weights

Experiments with Regularization

- ▶ [Chen and Rosenfeld, 1998]: apply log-linear models to language modeling: Estimate $q(w_i | w_{i-2}, w_{i-1})$
- ▶ Unigram, bigram, trigram features, e.g.,

$$f_1(w_{i-2}, w_{i-1}, w_i) = \begin{cases} 1 & \text{if trigram is (the, dog, laughs)} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(w_{i-2}, w_{i-1}, w_i) = \begin{cases} 1 & \text{if bigram is (dog, laughs)} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(w_{i-2}, w_{i-1}, w_i) = \begin{cases} 1 & \text{if unigram is (laughs)} \\ 0 & \text{otherwise} \end{cases}$$

$$q(w_i | w_{i-2}, w_{i-1}) = \frac{e^{f(w_{i-2}, w_{i-1}, w_i) \cdot v}}{\sum_w e^{f(w_{i-2}, w_{i-1}, w) \cdot v}}$$

Experiments with Gaussian Priors

- ▶ In regular (unregularized) log-linear models, if all n-gram features are included, then it's equivalent to maximum-likelihood estimates!

$$q(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

- ▶ [Chen and Rosenfeld, 1998]: with regularization, get very good results. Performs as well as or better than standardly used “discounting methods” (see lecture 2).
- ▶ Downside: computing $\sum_w e^{f(w_{i-2}, w_{i-1}, w) \cdot v}$ is **SLOW**.

Log-Linear Models for History-Based Parsing

Michael Collins, Columbia University

Log-Linear Taggers: Summary

- ▶ The input sentence is $w_{[1:n]} = w_1 \dots w_n$
- ▶ Each tag sequence $t_{[1:n]}$ has a conditional probability

$$\begin{aligned} p(t_{[1:n]} \mid w_{[1:n]}) &= \prod_{j=1}^n p(t_j \mid w_1 \dots w_n, t_1 \dots t_{j-1}) && \text{Chain rule} \\ &= \prod_{j=1}^n p(t_j \mid w_1 \dots w_n, t_{j-2}, t_{j-1}) && \text{Independence} \\ &&& \text{assumptions} \end{aligned}$$

- ▶ Estimate $p(t_j \mid w_1 \dots w_n, t_{j-2}, t_{j-1})$ using log-linear models
- ▶ Use the Viterbi algorithm to compute

$$\operatorname{argmax}_{t_{[1:n]}} \log p(t_{[1:n]} \mid w_{[1:n]})$$

A General Approach: (Conditional) History-Based Models

- ▶ We've shown how to define $p(t_{[1:n]} \mid w_{[1:n]})$ where $t_{[1:n]}$ is a tag sequence
- ▶ How do we define $p(T \mid S)$ if T is a parse tree (or another structure)? (We use the notation $S = w_{[1:n]}$)

A General Approach: (Conditional) History-Based Models

- ▶ Step 1: represent a tree as a sequence of **decisions** $d_1 \dots d_m$

$$T = \langle d_1, d_2, \dots, d_m \rangle$$

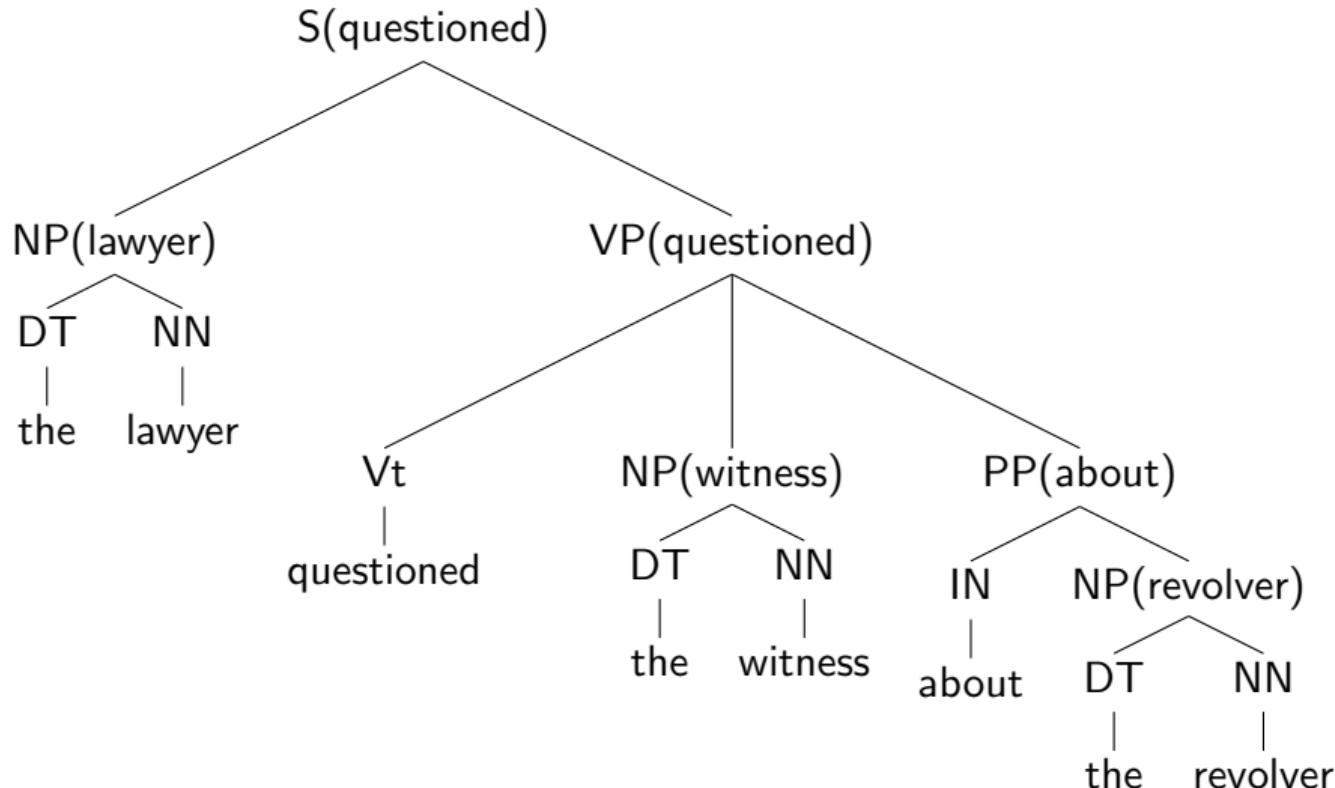
m is **not** necessarily the length of the sentence

- ▶ Step 2: the probability of a tree is

$$p(T \mid S) = \prod_{i=1}^m p(d_i \mid d_1 \dots d_{i-1}, S)$$

- ▶ Step 3: Use a log-linear model to estimate
 $p(d_i \mid d_1 \dots d_{i-1}, S)$
- ▶ Step 4: Search?? (answer we'll get to later: beam or heuristic search)

An Example Tree



Ratnaparkhi's Parser: Three Layers of Structure

1. Part-of-speech tags
2. Chunks
3. Remaining structure

Layer 1: Part-of-Speech Tags

DT	NN	Vt	DT	NN	IN	DT	NN
the	lawyer	questioned	the	witness	about	the	revolver

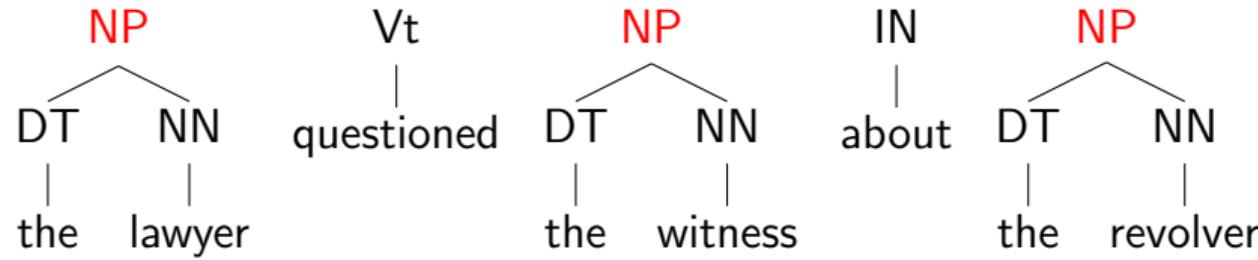
- ▶ Step 1: represent a tree as a sequence of **decisions** $d_1 \dots d_m$

$$T = \langle d_1, d_2, \dots, d_m \rangle$$

- ▶ First n decisions are tagging decisions

$$\langle d_1 \dots d_n \rangle = \langle \text{DT}, \text{NN}, \text{Vt}, \text{DT}, \text{NN}, \text{IN}, \text{DT}, \text{NN} \rangle$$

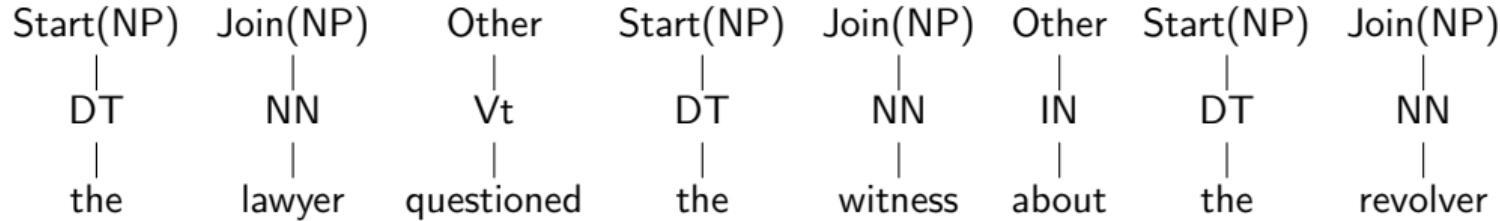
Layer 2: Chunks



Chunks are defined as any phrase where all children are part-of-speech tags

(Other common chunks are ADJP, QP)

Layer 2: Chunks



- ▶ Step 1: represent a tree as a sequence of **decisions** $d_1 \dots d_m$

$$T = \langle d_1, d_2, \dots, d_m \rangle$$

- ▶ First n decisions are tagging decisions
Next n decisions are chunk tagging decisions

$$\langle d_1 \dots d_{2n} \rangle = \langle \text{DT, NN, Vt, DT, NN, IN, DT, NN, } \\ \text{Start(NP), Join(NP), Other, Start(NP), Join(NP), } \\ \text{Other, Start(NP), Join(NP)} \rangle$$

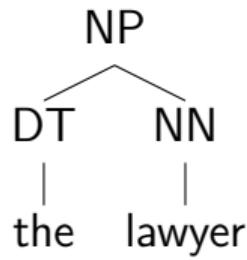
Layer 3: Remaining Structure

Alternate Between Two Classes of Actions:

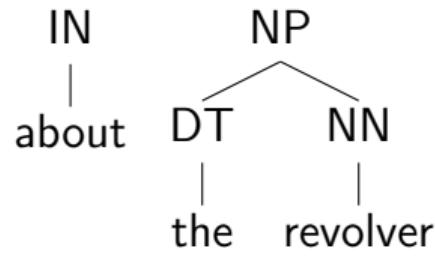
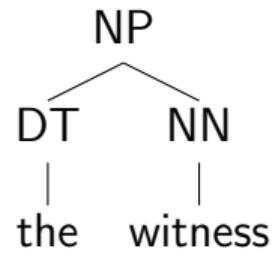
- ▶ Join(X) or Start(X), where X is a label (NP, S, VP etc.)
- ▶ Check=YES or Check=NO

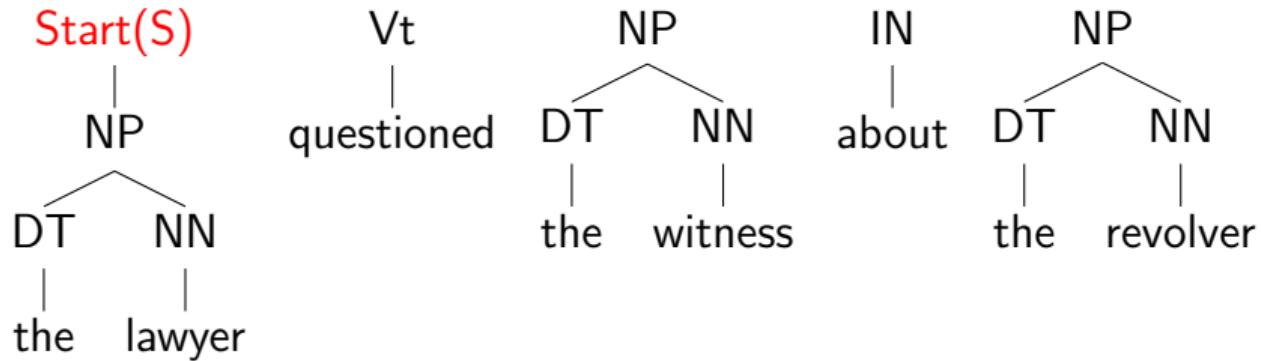
Meaning of these actions:

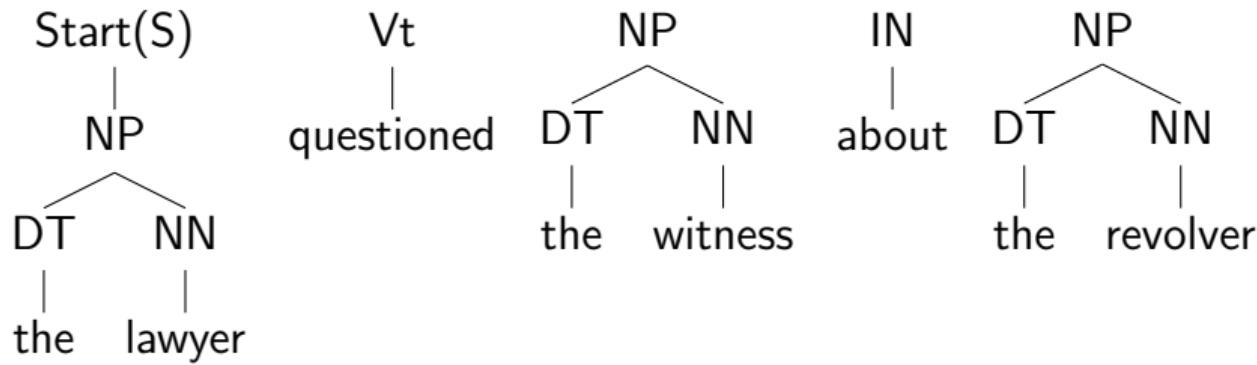
- ▶ Start(X) starts a new constituent with label X
(always acts on leftmost constituent with no start or join label above it)
- ▶ Join(X) continues a constituent with label X
(always acts on leftmost constituent with no start or join label above it)
- ▶ Check=NO does nothing
- ▶ Check=YES takes previous Join or Start action, and converts it into a completed constituent



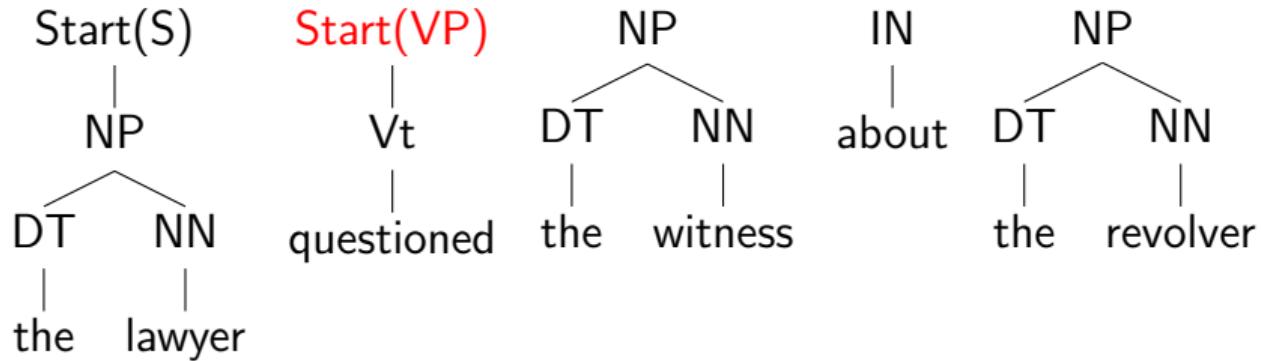
Vt
questioned

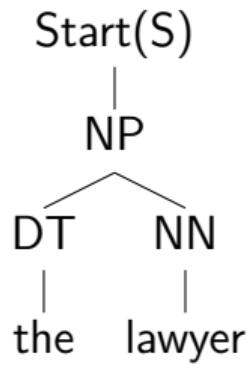






Check=NO





Start(VP)

```
graph TD; StartVP[Start(VP)] --- Vt1[Vt]; Vt1 --- questioned[questioned]
```

NP

```
graph TD; NP1[NP] --- DT1[DT]; NP1 --- NN1[NN]; DT1 --- the1[the]; NN1 --- witness[witness]
```

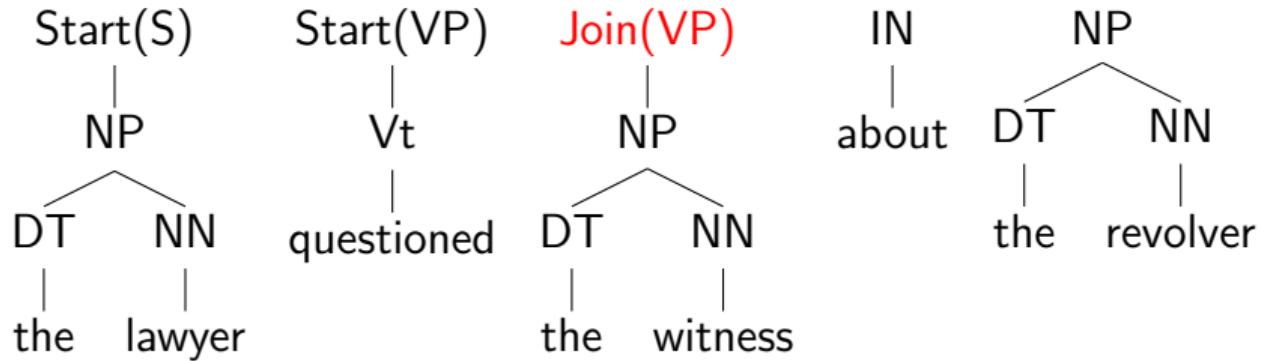
IN

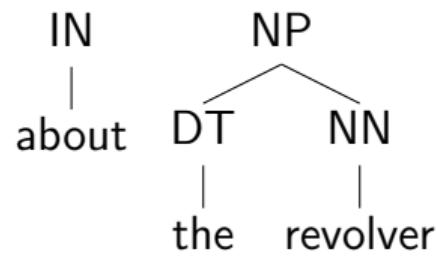
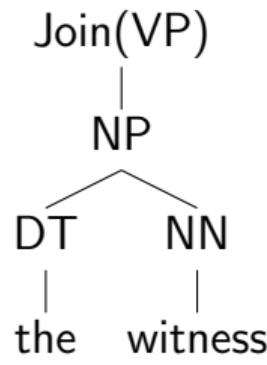
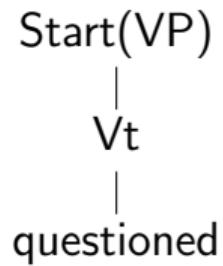
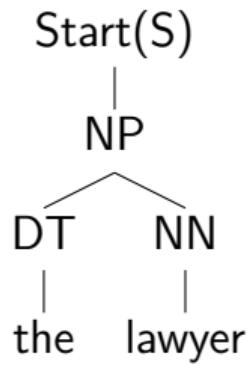
```
graph TD; IN1[IN] --- about1[about]
```

NP

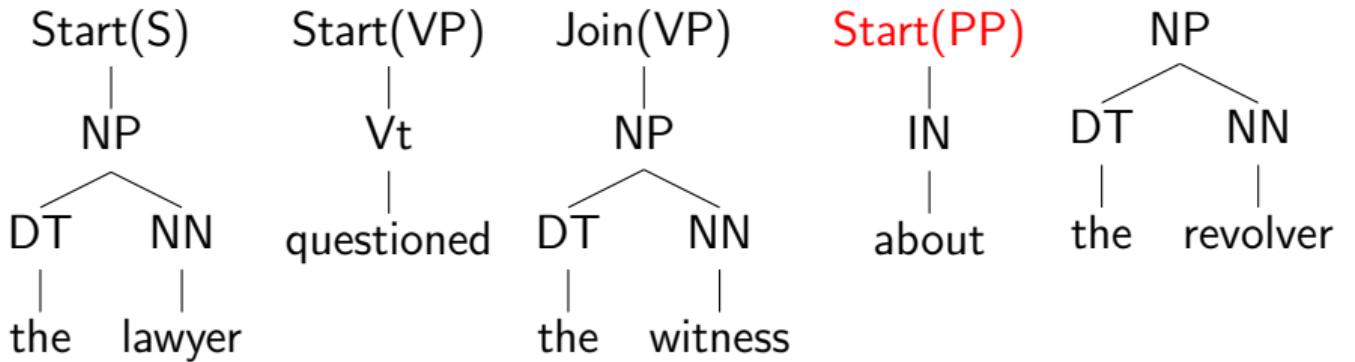
```
graph TD; NP1[NP] --- DT1[DT]; NP1 --- NN1[NN]; DT1 --- the1[the]; NN1 --- revolver[revolver]
```

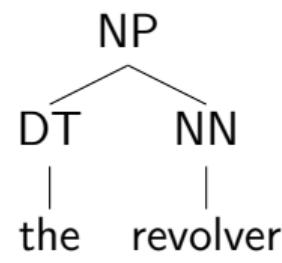
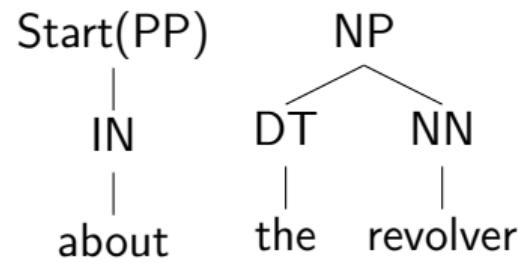
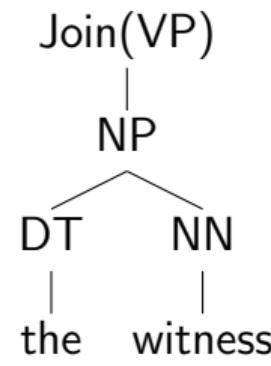
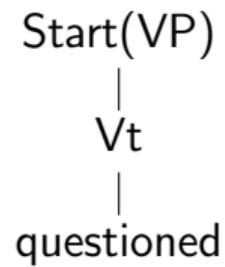
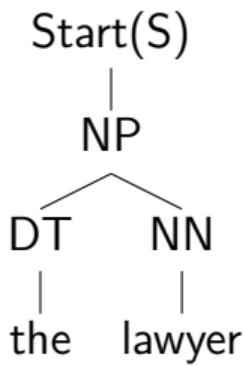
Check=NO



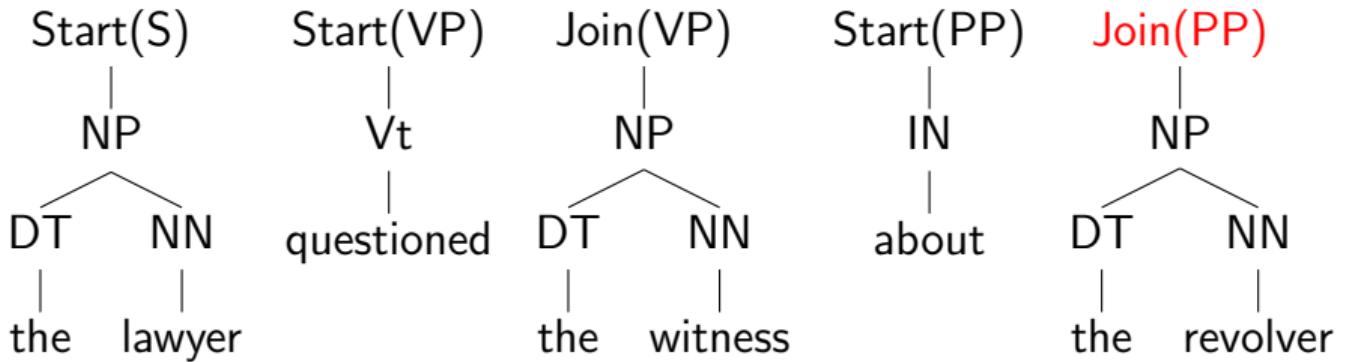


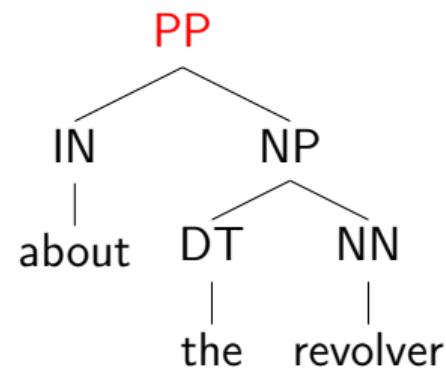
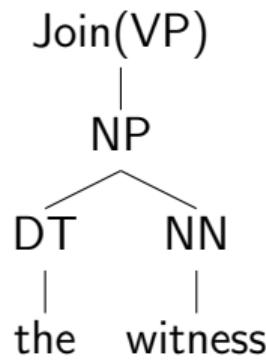
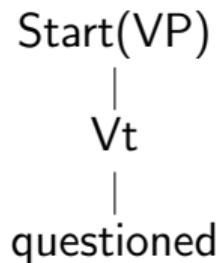
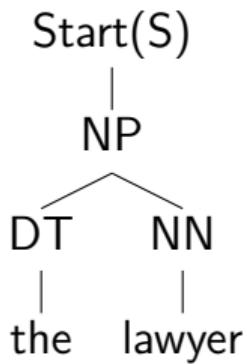
Check=NO



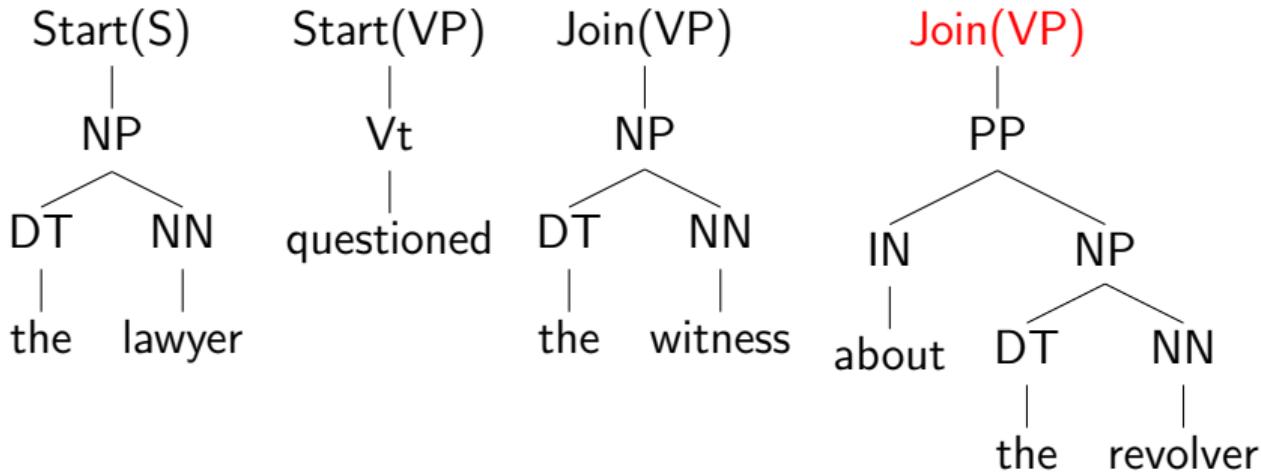


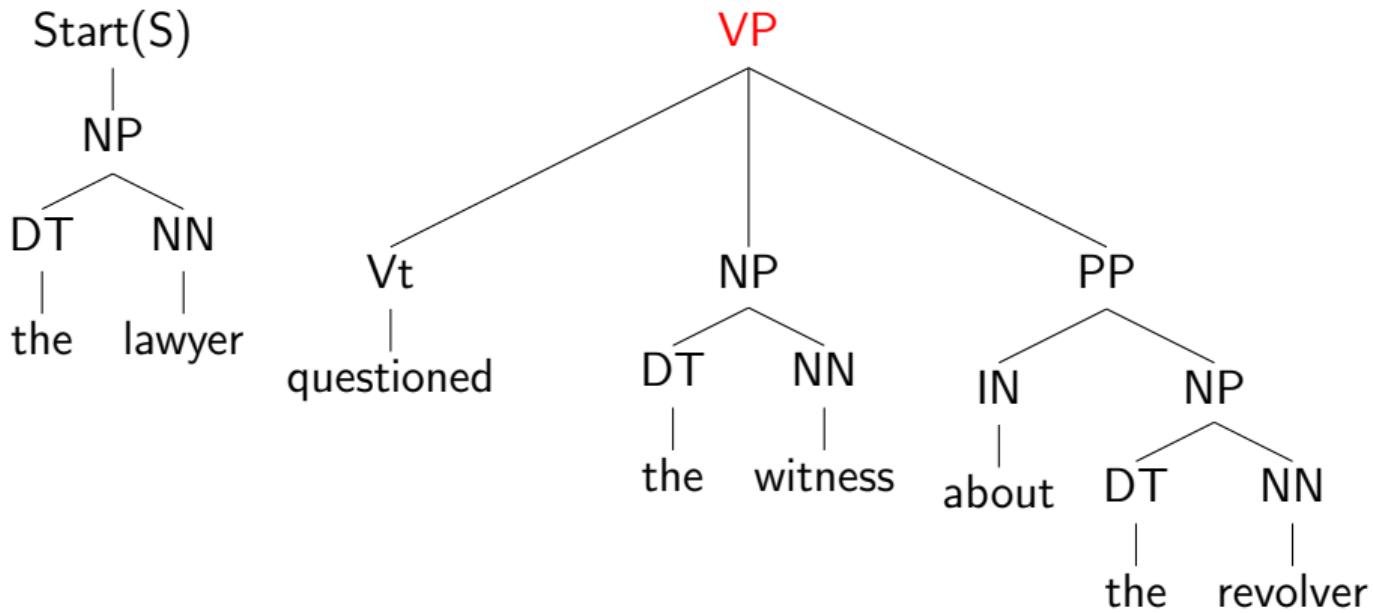
Check=NO



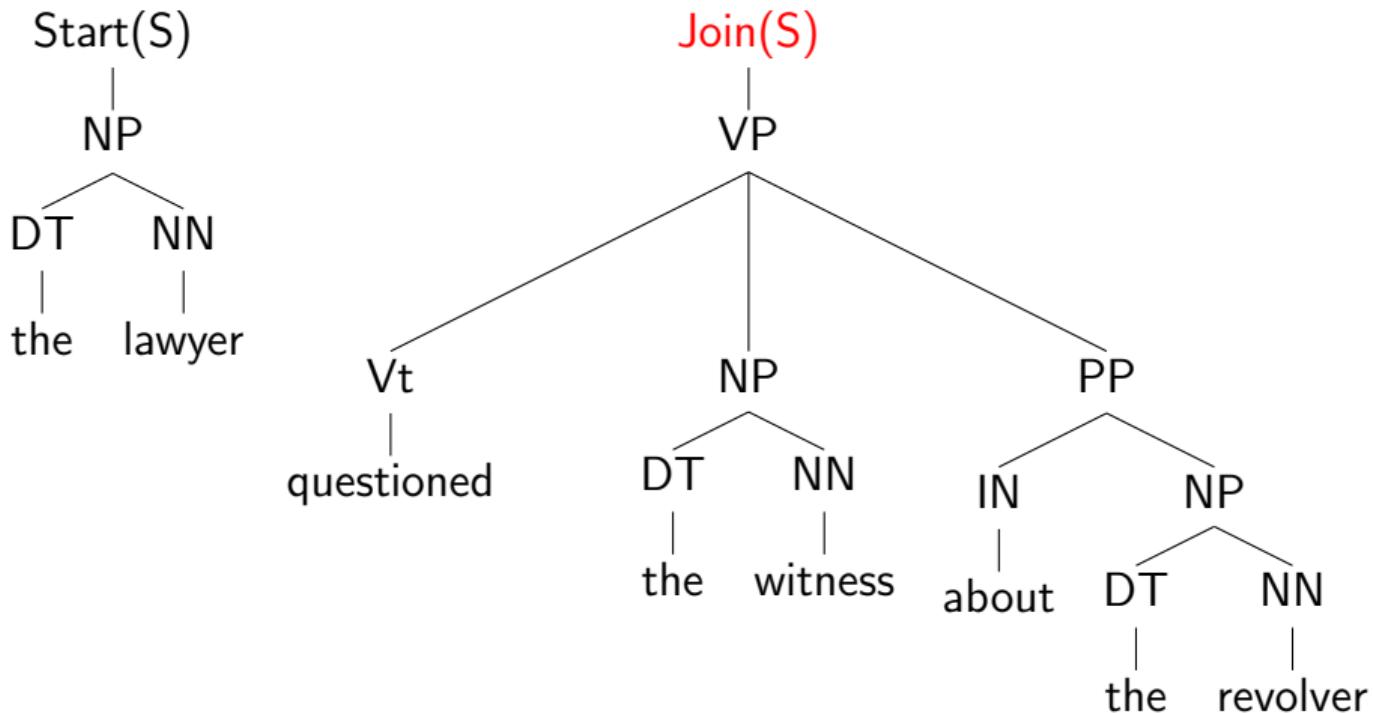


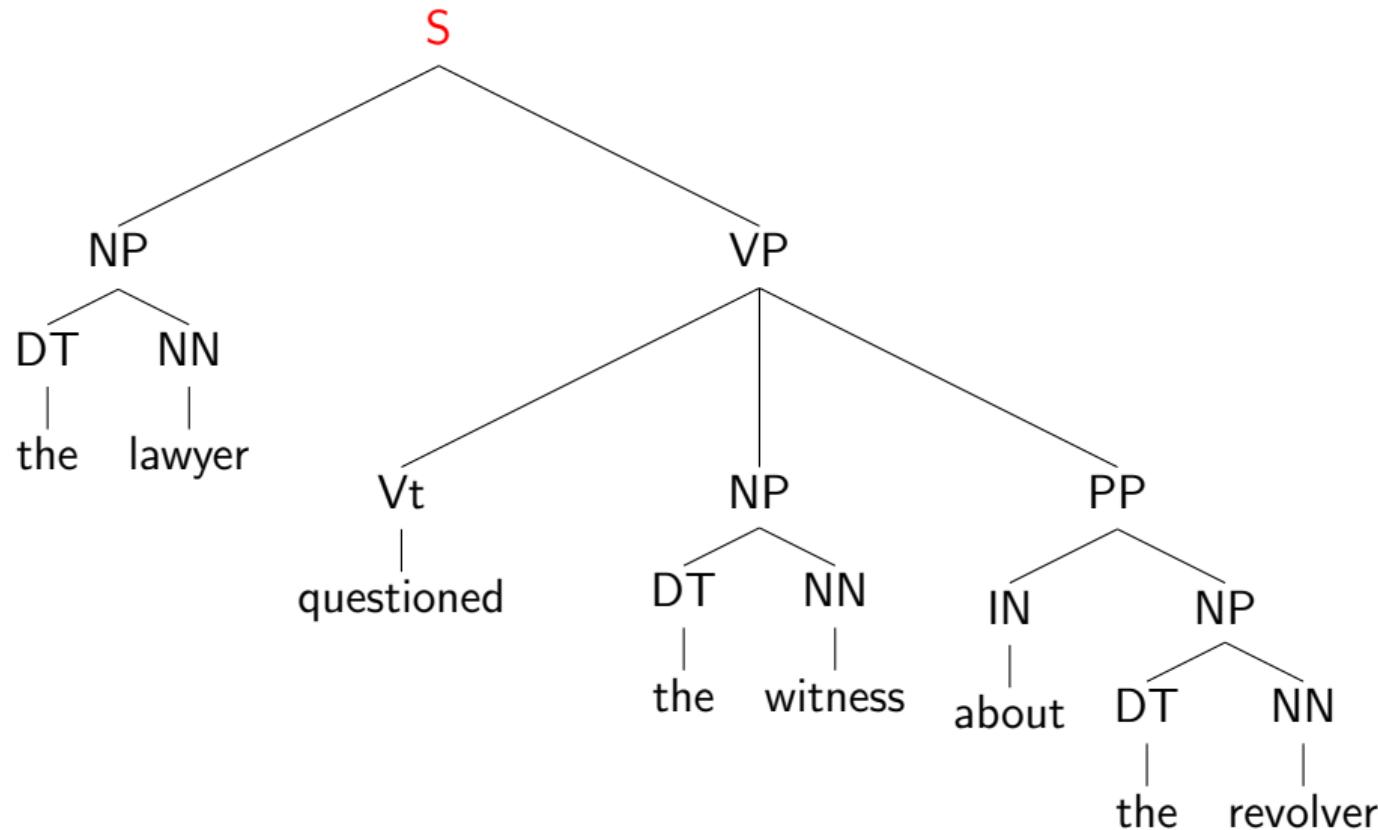
Check=YES





Check=YES





Check=YES

The Final Sequence of decisions

$$\langle d_1 \dots d_m \rangle = \langle \text{DT, NN, Vt, DT, NN, IN, DT, NN,} \\ \text{Start(NP), Join(NP), Other, Start(NP), Join(NP),} \\ \text{Other, Start(NP), Join(NP),} \\ \text{Start(S), Check=NO, Start(VP), Check=NO,} \\ \text{Join(VP), Check=NO, Start(PP), Check=NO,} \\ \text{Join(PP), Check=YES, Join(VP), Check=YES,} \\ \text{Join(S), Check=YES } \rangle$$

A General Approach: (Conditional) History-Based Models

- ▶ Step 1: represent a tree as a sequence of **decisions** $d_1 \dots d_m$

$$T = \langle d_1, d_2, \dots, d_m \rangle$$

m is **not** necessarily the length of the sentence

- ▶ Step 2: the probability of a tree is

$$p(T \mid S) = \prod_{i=1}^m p(d_i \mid d_1 \dots d_{i-1}, S)$$

- ▶ Step 3: Use a log-linear model to estimate

$$p(d_i \mid d_1 \dots d_{i-1}, S)$$

- ▶ Step 4: Search?? (answer we'll get to later: beam or heuristic search)

Applying a Log-Linear Model

- ▶ Step 3: Use a log-linear model to estimate

$$p(d_i \mid d_1 \dots d_{i-1}, S)$$

- ▶ A reminder:

$$p(d_i \mid d_1 \dots d_{i-1}, S) = \frac{e^{f(\langle d_1 \dots d_{i-1}, S \rangle, d_i) \cdot v}}{\sum_{d \in \mathcal{A}} e^{f(\langle d_1 \dots d_{i-1}, S \rangle, d) \cdot v}}$$

where:

$\langle d_1 \dots d_{i-1}, S \rangle$ is the history

d_i is the outcome

f maps a history/outcome pair to a feature vector

v is a parameter vector

\mathcal{A} is set of possible actions

Applying a Log-Linear Model

- ▶ Step 3: Use a log-linear model to estimate

$$p(d_i \mid d_1 \dots d_{i-1}, S) = \frac{e^{f(\langle d_1 \dots d_{i-1}, S \rangle, d_i) \cdot v}}{\sum_{d \in \mathcal{A}} e^{f(\langle d_1 \dots d_{i-1}, S \rangle, d) \cdot v}}$$

- ▶ The big question: how do we define f ?
- ▶ Ratnaparkhi's method defines f differently depending on whether next decision is:
 - ▶ A tagging decision
(same features as before for POS tagging!)
 - ▶ A chunking decision
 - ▶ A start/join decision after chunking
 - ▶ A check=no/check=yes decision

Layer 3: Join or Start

- ▶ Looks at head word, constituent (or POS) label, and start/join annotation of n 'th tree relative to the decision, where $n = -2, -1$
- ▶ Looks at head word, constituent (or POS) label of n 'th tree relative to the decision, where $n = 0, 1, 2$
- ▶ Looks at bigram features of the above for (-1,0) and (0,1)
- ▶ Looks at trigram features of the above for (-2,-1,0), (-1,0,1) and (0, 1, 2)
- ▶ The above features with all combinations of head words excluded
- ▶ Various punctuation features

Layer 3: Check=NO or Check=YES

- ▶ A variety of questions concerning the proposed constituent

The Search Problem

- ▶ In POS tagging, we could use the Viterbi algorithm because

$$p(t_j \mid w_1 \dots w_n, j, t_1 \dots t_{j-1}) = p(t_j \mid w_1 \dots w_n, j, t_{j-2} \dots t_{j-1})$$

- ▶ Now: Decision d_i could depend on arbitrary decisions in the “past” \Rightarrow no chance for dynamic programming
- ▶ Instead, Ratnaparkhi uses a beam search method

Log-Linear Models for Tagging (Maximum-entropy Markov Models (MEMMs))

Michael Collins, Columbia University

Part-of-Speech Tagging

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street,
as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V
forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N
Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

N = Noun

V = Verb

P = Preposition

Adv = Adverb

Adj = Adjective

...

Named Entity Recognition

INPUT: Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT: Profits soared at [Company Boeing Co.], easily topping forecasts on [Location Wall Street], as their CEO [Person Alan Mulally] announced first quarter results.

Named Entity Extraction as Tagging

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street,
as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA
topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA
their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA
quarter/NA results/NA ./NA

NA = No entity

SC = Start Company

CC = Continue Company

SL = Start Location

CL = Continue Location

Our Goal

Training set:

- 1 Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./.
- 2 Mr./NNP Vinken/NNP is/VBZ chairman/NN of/IN Elsevier/NNP N.V./NNP ,/, the/DT Dutch/NNP publishing/VBG group/NN ./.
- 3 Rudolph/NNP Agnew/NNP ,/, 55/CD years/NNS old/JJ and/CC chairman/NN of/IN Consolidated/NNP Gold/NNP Fields/NNP PLC/NNP ,/, was/VBD named/VBN a/DT nonexecutive/JJ director/NN of/IN this/DT British/JJ industrial/JJ conglomerate/NN ./.

...

- 38,219 It/PRP is/VBZ also/RB pulling/VBG 20/CD people/NNS out/IN of/IN Puerto/NNP Rico/NNP ,/, who/WP were/VBD helping/VBG Hurricane/NNP Hugo/NNP victims/NNS ,/, and/CC sending/VBG them/PRP to/TO San/NNP Francisco/NNP instead/RB ./.

- ▶ From the training set, induce a function/algorithm that maps new sentences to their tag sequences.

Overview

- ▶ Recap: The Tagging Problem
- ▶ Log-linear taggers

Log-Linear Models for Tagging

- ▶ We have an input sentence $w_{[1:n]} = w_1, w_2, \dots, w_n$
(w_i is the i 'th word in the sentence)

Log-Linear Models for Tagging

- ▶ We have an input sentence $w_{[1:n]} = w_1, w_2, \dots, w_n$
(w_i is the i 'th word in the sentence)
- ▶ We have a tag sequence $t_{[1:n]} = t_1, t_2, \dots, t_n$
(t_i is the i 'th tag in the sentence)

Log-Linear Models for Tagging

- ▶ We have an input sentence $w_{[1:n]} = w_1, w_2, \dots, w_n$
(w_i is the i 'th word in the sentence)
- ▶ We have a tag sequence $t_{[1:n]} = t_1, t_2, \dots, t_n$
(t_i is the i 'th tag in the sentence)
- ▶ We'll use an log-linear model to define

$$p(t_1, t_2, \dots, t_n | w_1, w_2, \dots, w_n)$$

for any sentence $w_{[1:n]}$ and tag sequence $t_{[1:n]}$ of the same length.
(Note: contrast with HMM that defines $p(t_1 \dots t_n, w_1 \dots w_n)$)

Log-Linear Models for Tagging

- ▶ We have an input sentence $w_{[1:n]} = w_1, w_2, \dots, w_n$ (w_i is the i 'th word in the sentence)
- ▶ We have a tag sequence $t_{[1:n]} = t_1, t_2, \dots, t_n$ (t_i is the i 'th tag in the sentence)
- ▶ We'll use an log-linear model to define

$$p(t_1, t_2, \dots, t_n | w_1, w_2, \dots, w_n)$$

for any sentence $w_{[1:n]}$ and tag sequence $t_{[1:n]}$ of the same length.
(Note: contrast with HMM that defines $p(t_1 \dots t_n, w_1 \dots w_n)$)

- ▶ Then the most likely tag sequence for $w_{[1:n]}$ is

$$t_{[1:n]}^* = \operatorname{argmax}_{t_{[1:n]}} p(t_{[1:n]} | w_{[1:n]})$$

How to model $p(t_{[1:n]} | w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$$p(t_{[1:n]} | w_{[1:n]}) = \prod_{j=1}^n p(t_j | w_1 \dots w_n, t_1 \dots t_{j-1}) \quad \text{Chain rule}$$

How to model $p(t_{[1:n]} | w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$$p(t_{[1:n]} | w_{[1:n]}) = \prod_{j=1}^n p(t_j | w_1 \dots w_n, t_1 \dots t_{j-1}) \quad \text{Chain rule}$$

$$= \prod_{j=1}^n p(t_j | w_1, \dots, w_n, t_{j-2}, t_{j-1})$$

Independence assumptions

- We take $t_0 = t_{-1} = *$

How to model $p(t_{[1:n]} | w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$$\begin{aligned} p(t_{[1:n]} | w_{[1:n]}) &= \prod_{j=1}^n p(t_j | w_1 \dots w_n, t_1 \dots t_{j-1}) && \text{Chain rule} \\ &= \prod_{j=1}^n p(t_j | w_1, \dots, w_n, t_{j-2}, t_{j-1}) && \text{Independence assumptions} \end{aligned}$$

- ▶ We take $t_0 = t_{-1} = *$
- ▶ Independence assumption: each tag only depends on previous two tags

$$p(t_j | w_1, \dots, w_n, t_1, \dots, t_{j-1}) = p(t_j | w_1, \dots, w_n, t_{j-2}, t_{j-1})$$

An Example

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ
base/?? from which Spain expanded its empire into the rest of the
Western Hemisphere .

- There are many possible tags in the position ??

$$\mathcal{Y} = \{\text{NN, NNS, Vt, Vi, IN, DT, ...}\}$$

Representation: Histories

- ▶ A **history** is a 4-tuple $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
 - ▶ t_{-2}, t_{-1} are the previous two tags.
 - ▶ $w_{[1:n]}$ are the n words in the input sentence.
 - ▶ i is the index of the word being tagged
 - ▶ \mathcal{X} is the set of all possible histories
-

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ
base/?? from which Spain expanded its empire into the rest of the
Western Hemisphere .

- ▶ $t_{-2}, t_{-1} = \text{DT, JJ}$
- ▶ $w_{[1:n]} = \langle \text{Hispaniola, quickly, became, \dots, Hemisphere, .} \rangle$
- ▶ $i = 6$

Recap: Feature Vector Representations in Log-Linear Models

- ▶ We have some input domain \mathcal{X} , and a finite label set \mathcal{Y} . Aim is to provide a conditional probability $p(y | x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- ▶ A **feature** is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$
(Often **binary features** or **indicator functions**
 $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$).
- ▶ Say we have m features f_k for $k = 1 \dots m$
⇒ A **feature vector** $f(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

An Example (continued)

- ▶ \mathcal{X} is the set of all possible histories of form $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
 - ▶ $\mathcal{Y} = \{\text{NN}, \text{NNS}, \text{Vt}, \text{Vi}, \text{IN}, \text{DT}, \dots\}$
 - ▶ We have m features $f_k : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ for $k = 1 \dots m$
-

For example:

$$f_1(\textcolor{green}{h}, \textcolor{red}{t}) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$
$$f_2(\textcolor{green}{h}, \textcolor{red}{t}) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in ing and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$
$$\dots$$

$$f_1(\langle \text{JJ}, \text{DT}, \langle \text{Hispaniola}, \dots \rangle, 6 \rangle, \text{Vt}) = 1$$

$$f_2(\langle \text{JJ}, \text{DT}, \langle \text{Hispaniola}, \dots \rangle, 6 \rangle, \text{Vt}) = 0$$

...

The Full Set of Features in [(Ratnaparkhi, 96)]

- ▶ Word/tag features for all word/tag pairs, e.g.,

$$f_{100}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

- ▶ Spelling features for all prefixes/suffixes of length ≤ 4 , e.g.,

$$f_{101}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in ing and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{102}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ starts with pre and } t = \text{NN} \\ 0 & \text{otherwise} \end{cases}$$

The Full Set of Features in [(Ratnaparkhi, 96)]

- Contextual Features, e.g.,

$$f_{103}(h, t) = \begin{cases} 1 & \text{if } \langle t_{-2}, t_{-1}, t \rangle = \langle \text{DT}, \text{JJ}, \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{104}(h, t) = \begin{cases} 1 & \text{if } \langle t_{-1}, t \rangle = \langle \text{JJ}, \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{105}(h, t) = \begin{cases} 1 & \text{if } \langle t \rangle = \langle \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{106}(h, t) = \begin{cases} 1 & \text{if previous word } w_{i-1} = \text{the and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{107}(h, t) = \begin{cases} 1 & \text{if next word } w_{i+1} = \text{the and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

Log-Linear Models

- ▶ We have some input domain \mathcal{X} , and a finite label set \mathcal{Y} . Aim is to provide a conditional probability $p(y | x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- ▶ A feature is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$
(Often binary features or indicator functions
 $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$).
- ▶ Say we have m features f_k for $k = 1 \dots m$
⇒ A feature vector $f(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- ▶ We also have a **parameter vector** $v \in \mathbb{R}^m$
- ▶ We define

$$p(y | x; v) = \frac{e^{v \cdot f(x, y)}}{\sum_{y' \in \mathcal{Y}} e^{v \cdot f(x, y')}}$$

Training the Log-Linear Model

- To train a log-linear model, we need a training set (x_i, y_i) for $i = 1 \dots n$. Then search for

$$v^* = \operatorname{argmax}_v \left(\underbrace{\sum_i \log p(y_i | x_i; v)}_{\text{Log-Likelihood}} - \frac{\lambda}{2} \underbrace{\sum_k v_k^2}_{\text{Regularizer}} \right)$$

(see last lecture on log-linear models)

- Training set is simply all history/tag pairs seen in the training data

The Viterbi Algorithm

Problem: for an input $w_1 \dots w_n$, find

$$\arg \max_{t_1 \dots t_n} p(t_1 \dots t_n \mid w_1 \dots w_n)$$

We assume that p takes the form

$$p(t_1 \dots t_n \mid w_1 \dots w_n) = \prod_{i=1}^n q(t_i \mid t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

(In our case $q(t_i \mid t_{i-2}, t_{i-1}, w_{[1:n]}, i)$ is the estimate from a log-linear model.)

The Viterbi Algorithm

- ▶ Define n to be the length of the sentence
- ▶ Define

$$r(t_1 \dots t_k) = \prod_{i=1}^k q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

- ▶ Define a dynamic programming table

$\pi(k, u, v) =$ maximum probability of a tag sequence ending
in tags u, v at position k

that is,

$$\pi(k, u, v) = \max_{\langle t_1, \dots, t_{k-2} \rangle} r(t_1 \dots t_{k-2}, u, v)$$

A Recursive Definition

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition:

For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{t \in \mathcal{S}_{k-2}} (\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k))$$

where \mathcal{S}_k is the set of possible tags at position k

The Viterbi Algorithm with Backpointers

Input: a sentence $w_1 \dots w_n$, log-linear model that provides $q(v|t, u, w_{[1:n]}, i)$ for any tag-trigram t, u, v , for any $i \in \{1 \dots n\}$

Initialization: Set $\pi(0, *, *) = 1$.

Algorithm:

- ▶ For $k = 1 \dots n$,

- ▶ For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\pi(k, u, v) = \max_{t \in \mathcal{S}_{k-2}} (\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k))$$

$$bp(k, u, v) = \arg \max_{t \in \mathcal{S}_{k-2}} (\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k))$$

- ▶ Set $(t_{n-1}, t_n) = \arg \max_{(u,v)} \pi(n, u, v)$
- ▶ For $k = (n-2) \dots 1$, $t_k = bp(k+2, t_{k+1}, t_{k+2})$
- ▶ **Return** the tag sequence $t_1 \dots t_n$

FAQ Segmentation: McCallum et. al

- ▶ McCallum et. al compared HMM and log-linear taggers on a *FAQ Segmentation* task
- ▶ Main point: in an HMM, modeling

$$p(\text{word}|\text{tag})$$

is difficult in this domain

FAQ Segmentation: McCallum et. al

```
<head>X-NNTP-POSTER: NewsHound v1.33
<head>
<head>Archive name: acorn/faq/part2
<head>Frequency: monthly
<head>

<question>2.6) What configuration of serial cable should I use
<answer>
<answer> Here follows a diagram of the necessary connections
<answer>programs to work properly. They are as far as I know t
<answer>agreed upon by commercial comms software developers fo
<answer>
<answer> Pins 1, 4, and 8 must be connected together inside
<answer>is to avoid the well known serial port chip bugs. The
```

FAQ Segmentation: Line Features

begins-with-number
begins-with-ordinal
begins-with-punctuation
begins-with-question-word
begins-with-subject
blank
contains-alphanum
contains-bracketed-number
contains-http
contains-non-space
contains-number
contains-pipe
contains-question-mark
ends-with-question-mark
first-alpha-is-capitalized
indented-1-to-4

FAQ Segmentation: The Log-Linear Tagger

```
<head>X-NNTP-POSTER: NewsHound v1.33
<head>
<head>Archive name: acorn/faq/part2
<head>Frequency: monthly
<head>
<question>2.6) What configuration of serial cable should I use
```

Here follows a diagram of the necessary connections

- ⇒ “tag=question;prev=head;begins-with-number”
- “tag=question;prev=head;contains-alphanum”
- “tag=question;prev=head;contains-nonspace”
- “tag=question;prev=head;contains-number”
- “tag=question;prev=head;prev-is-blank”

FAQ Segmentation: An HMM Tagger

<question>2.6) What configuration of serial cable should I use

- ▶ First solution for $p(\text{word} \mid \text{tag})$:

$p(\text{"2.6) What configuration of serial cable should I use"} \mid \text{question}) =$
 $e(\text{ 2.6}) \times$
 $e(\text{What} \mid \text{question}) \times$
 $e(\text{configuration} \mid \text{question}) \times$
 $e(\text{of} \mid \text{question}) \times$
 $e(\text{serial} \mid \text{question}) \times$
...

- ▶ i.e. have a **language model** for each tag

FAQ Segmentation: McCallum et. al

- ▶ Second solution: first map each sentence to string of features:

<question>2.6) What configuration of serial cable should I use

⇒

<question>begins-with-number contains-alphanum contains-nonspace
contains-number prev-is-blank

- ▶ Use a language model again:

$$p(\text{"2.6) What configuration of serial cable should I use"} \mid \text{question}) = \\ e(\text{begins-with-number} \mid \text{question}) \times \\ e(\text{contains-alphanum} \mid \text{question}) \times \\ e(\text{contains-nonspace} \mid \text{question}) \times \\ e(\text{contains-number} \mid \text{question}) \times \\ e(\text{prev-is-blank} \mid \text{question}) \times$$

FAQ Segmentation: Results

Method	Precision	Recall
ME-Stateless	0.038	0.362
TokenHMM	0.276	0.140
FeatureHMM	0.413	0.529
MEMM	0.867	0.681

- ▶ Precision and recall results are for recovering segments

FAQ Segmentation: Results

Method	Precision	Recall
ME-Stateless	0.038	0.362
TokenHMM	0.276	0.140
FeatureHMM	0.413	0.529
MEMM	0.867	0.681

- ▶ Precision and recall results are for recovering segments
- ▶ ME-stateless is a log-linear model that treats every sentence separately (no dependence between adjacent tags)

FAQ Segmentation: Results

Method	Precision	Recall
ME-Stateless	0.038	0.362
TokenHMM	0.276	0.140
FeatureHMM	0.413	0.529
MEMM	0.867	0.681

- ▶ Precision and recall results are for recovering segments
- ▶ ME-stateless is a log-linear model that treats every sentence separately (no dependence between adjacent tags)
- ▶ TokenHMM is an HMM with first solution we've just seen

FAQ Segmentation: Results

Method	Precision	Recall
ME-Stateless	0.038	0.362
TokenHMM	0.276	0.140
FeatureHMM	0.413	0.529
MEMM	0.867	0.681

- ▶ Precision and recall results are for recovering segments
- ▶ ME-stateless is a log-linear model that treats every sentence separately (no dependence between adjacent tags)
- ▶ TokenHMM is an HMM with first solution we've just seen
- ▶ FeatureHMM is an HMM with second solution we've just seen

FAQ Segmentation: Results

Method	Precision	Recall
ME-Stateless	0.038	0.362
TokenHMM	0.276	0.140
FeatureHMM	0.413	0.529
MEMM	0.867	0.681

- ▶ Precision and recall results are for recovering segments
- ▶ ME-stateless is a log-linear model that treats every sentence separately (no dependence between adjacent tags)
- ▶ TokenHMM is an HMM with first solution we've just seen
- ▶ FeatureHMM is an HMM with second solution we've just seen
- ▶ MEMM is a log-linear trigram tagger (MEMM stands for "Maximum-Entropy Markov Model")

Summary

- ▶ Key ideas in log-linear taggers:

- ▶ Decompose

$$p(t_1 \dots t_n | w_1 \dots w_n) = \prod_{i=1}^n p(t_i | t_{i-2}, t_{i-1}, w_1 \dots w_n)$$

- ▶ Estimate

$$p(t_i | t_{i-2}, t_{i-1}, w_1 \dots w_n)$$

using a log-linear model

- ▶ For a test sentence $w_1 \dots w_n$, use the Viterbi algorithm to find

$$\arg \max_{t_1 \dots t_n} \left(\prod_{i=1}^n p(t_i | t_{i-2}, t_{i-1}, w_1 \dots w_n) \right)$$

- ▶ Key advantage over HMM taggers: **flexibility in the features they can use**

Machine Translation

Michael Collins, Columbia University

Sign in

Try a new browser with automatic translation. [Download Google Chrome](#) [Dismiss](#)

Translate

From: Arabic - detected

To: English

Translate

English Spanish French Arabic - detected

كما أوضح أن الإنفاق الاستهلاكي كان المحرك الرئيسي للاقتصاد الذي تضرر جراء عوامل من الاضطرابات السياسية

وأشار إلى أن هناك شبه غياب للاستثمارات الأجنبية المباشرة في النصف الأول من السنة المالية، وأنه لتحقيق نمو اقتصادي بنسبة 7% تحتاج البلاد إلى معدل استثمار لا يقل عن 22%

English Spanish Arabic

He also explained that consumer spending was the main engine of the economy that has been hit by two years of political turmoil

He pointed out that there is a near absence of foreign direct investment (FDI) in the first half of the fiscal year, and that to achieve economic growth of 7% country needs investment rate of at least 22%

Overview

- ▶ Challenges in machine translation
- ▶ Classical machine translation
- ▶ A brief introduction to statistical MT

Challenges: Lexical Ambiguity

(Example from Dorr et. al, 1999)

Example 1:

book the flight ⇒ reservar

read the book ⇒ libro

Example 2:

the box was in the pen

the pen was on the table

Example 3:

kill a man ⇒ matar

kill a process ⇒ acabar

Challenges: Differing Word Orders

- ▶ English word order is *subject – verb – object*
- ▶ Japanese word order is *subject – object – verb*

English: IBM bought Lotus

Japanese: *IBM Lotus bought*

English: Sources said that IBM bought Lotus yesterday

Japanese: *Sources yesterday IBM Lotus bought that said*

Syntactic Structure is not Preserved Across Translations (Example from Dorr et. al, 1999)

The bottle floated into the cave



La botella entro a la cuerva flotando
(the bottle entered the cave floating)

Syntactic Ambiguity Causes Problems

(Example from Dorr et. al, 1999)

John hit the dog with the stick



John golpeo el perro con el palo/que tenia el palo

Pronoun Resolution (Example from Dorr et. al, 1999)

The computer outputs the data; it is fast.



La computadora imprime los datos; **es** rápida

The computer outputs the data; it is stored in ascii.



La computadora imprime los datos; **están** almacenados en ascii

Overview

- ▶ Challenges in machine translation
- ▶ Classical machine translation
- ▶ A brief introduction to statistical MT

Direct Machine Translation

- ▶ Translation is word-by-word
- ▶ Very little analysis of the source text (e.g., no syntactic or semantic analysis)
- ▶ Relies on a large bilingual dictionary. For each word in the source language, the dictionary specifies a set of rules for translating that word
- ▶ After the words are translated, simple reordering rules are applied (e.g., move adjectives after nouns when translating from English to French)

An Example of a set of Direct Translation Rules

(From Jurafsky and Martin, edition 2, chapter 25. Originally from a system from Panov 1960)

Rules for translating *much* or *many* into Russian:

if preceding word is *how* **return** *skol'ko*

else if preceding word is *as* **return** *stol'ko zhe*

else if word is *much*

if preceding word is *very* **return** *nil*

else if following word is a noun **return** *mnogo*

else (word is many)

if preceding word is a preposition and following word is noun **return** *mnogii*

else return *mnogo*

Some Problems with Direct Machine Translation

- ▶ Lack of any analysis of the source language causes several problems, for example:

- ▶ Difficult or impossible to capture long-range reorderings

English: Sources said that IBM bought Lotus yesterday

Japanese: *Sources yesterday IBM Lotus bought that said*

- ▶ Words are translated without disambiguation of their syntactic role

e.g., *that* can be a complementizer or determiner, and will often be translated differently for these two cases

They said *that* ...

They like *that* ice-cream

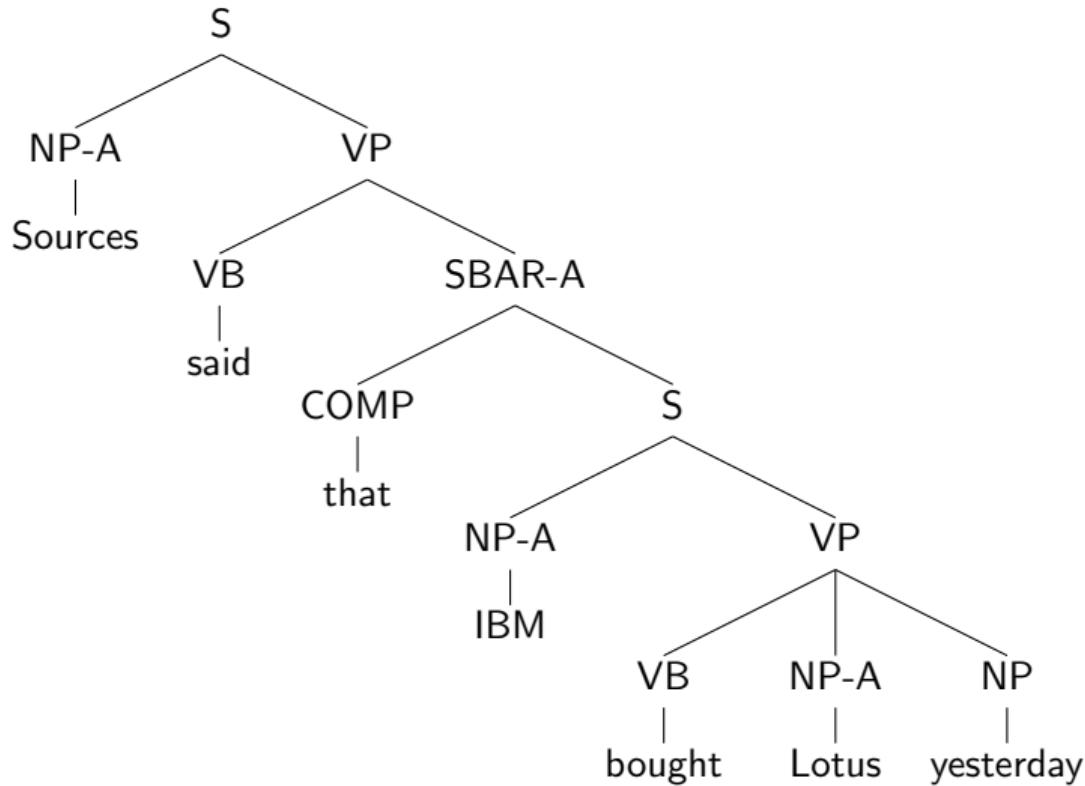
Transfer-Based Approaches

Three phases in translation:

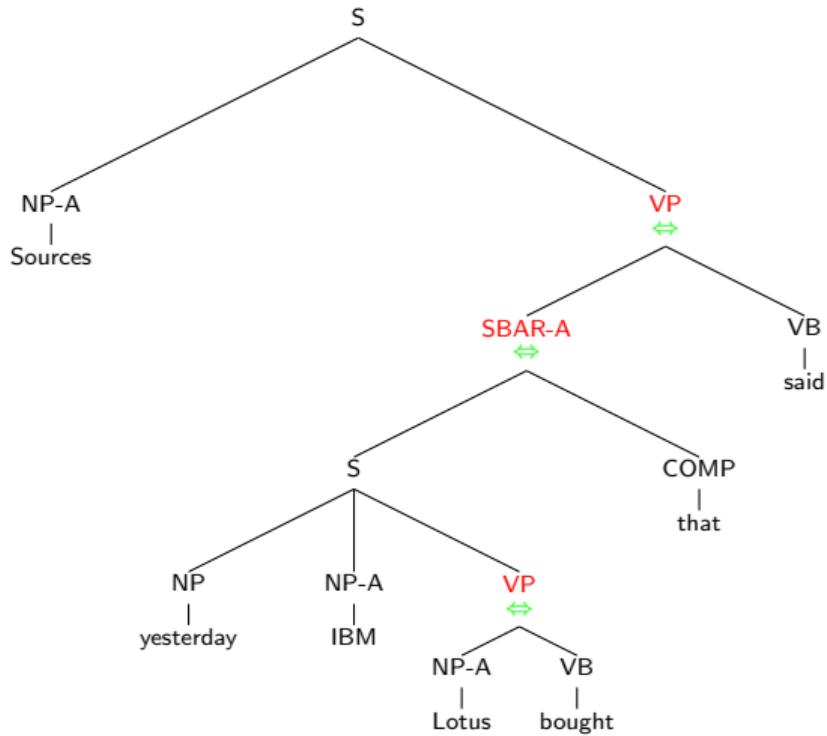
- ▶ **Analysis:** Analyze the source language sentence; for example, build a syntactic analysis of the source language sentence.
- ▶ **Transfer:** Convert the source-language parse tree to a target-language parse tree.
- ▶ **Generation:** Convert the target-language parse tree to an output sentence.

Transfer-Based Approaches

- ▶ The “parse trees” involved can vary from shallow analyses to much deeper analyses (even semantic representations).
- ▶ The transfer rules might look quite similar to the rules for direct translation systems. But they can now operate on syntactic structures.
- ▶ It’s easier with these approaches to handle long-distance reorderings
- ▶ The *Systran* systems are a classic example of this approach



⇒ Japanese: *Sources yesterday IBM Lotus bought that said*



Interlingua-Based Translation

Two phases in translation:

- ▶ **Analysis:** Analyze the source language sentence into a (language-independent) representation of its meaning.
- ▶ **Generation:** Convert the meaning representation into an output sentence.

Interlingua-Based Translation

One Advantage: If we want to build a translation system that translates between n languages, we need to develop n analysis and generation systems. With a transfer based system, we'd need to develop $O(n^2)$ sets of translation rules.

Disadvantage: What would a language-independent representation look like?

Interlingua-Based Translation

- ▶ How to represent different concepts in an interlingua?
- ▶ Different languages break down concepts in quite different ways:

German has two words for *wall*: one for an internal wall, one for a wall that is outside

Japanese has two words for *brother*: one for an elder brother, one for a younger brother

Spanish has two words for *leg*: *pierna* for a human's leg, *pata* for an animal's leg, or the leg of a table

- ▶ An interlingua might end up simple being an intersection of these different ways of breaking down concepts, but that doesn't seem very satisfactory...

Overview

- ▶ Challenges in machine translation
- ▶ Classical machine translation
- ▶ A brief introduction to statistical MT

A Brief Introduction to Statistical MT

- ▶ Parallel corpora are available in several language pairs
- ▶ Basic idea: use a parallel corpus as a training set of translation examples
- ▶ Classic example: IBM work on French-English translation, using the Canadian Hansards. (1.7 million sentences of 30 words or less in length).
- ▶ Idea goes back to Warren Weaver (1949): suggested applying statistical and cryptanalytic techniques to translation.

... one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: "This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode."

(Warren Weaver, 1949, in a letter to Norbert Wiener)

The Noisy Channel Model

- ▶ Goal: translation system from French to English
- ▶ Have a model $p(e | f)$ which estimates conditional probability of any English sentence e given the French sentence f . Use the training corpus to set the parameters.
- ▶ A Noisy Channel Model has two components:

$p(e)$ **the language model**

$p(f | e)$ **the translation model**

- ▶ Giving:

$$p(e | f) = \frac{p(e, f)}{p(f)} = \frac{p(e)p(f | e)}{\sum_e p(e)p(f | e)}$$

and

$$\operatorname{argmax}_e p(e | f) = \operatorname{argmax}_e p(e)p(f | e)$$

More About the Noisy Channel Model

- ▶ The **language model** $p(e)$ could be a trigram model, estimated from any data (parallel corpus not needed to estimate the parameters)
- ▶ The **translation model** $p(f | e)$ is trained from a parallel corpus of French/English pairs.
- ▶ Note:
 - ▶ The translation model is backwards!
 - ▶ The language model can make up for deficiencies of the translation model.
 - ▶ Later we'll talk about how to build $p(f | e)$
 - ▶ Decoding, i.e., finding

$$\operatorname{argmax}_e p(e)p(f | e)$$

is also a challenging problem.

Example from Koehn and Knight tutorial

Translation from Spanish to English, candidate translations based on $p(\text{Spanish} \mid \text{English})$ alone:

Que hambre tengo yo

→

What hunger have $p(s|e) = 0.000014$

Hungry I am so $p(s|e) = 0.000001$

I am so hungry $p(s|e) = 0.0000015$

Have i that hunger $p(s|e) = 0.000020$

...

Example from Koehn and Knight tutorial (continued)

With $p(\text{Spanish} \mid \text{English}) \times p(\text{English})$:

Que hambre tengo yo

→

What hunger have $p(s|e)p(e) = 0.000014 \times 0.000001$

Hungry I am so $p(s|e)p(e) = 0.000001 \times 0.0000014$

I am so hungry $p(s|e)p(e) = 0.0000015 \times 0.0001$

Have i that hunger $p(s|e)p(e) = 0.000020 \times 0.00000098$

...

The IBM Translation Models

Michael Collins, Columbia University

Recap: The Noisy Channel Model

- ▶ Goal: translation system from French to English
- ▶ Have a model $p(e | f)$ which estimates conditional probability of any English sentence e given the French sentence f . Use the training corpus to set the parameters.
- ▶ A Noisy Channel Model has two components:

$p(e)$ **the language model**

$p(f | e)$ **the translation model**

- ▶ Giving:

$$p(e | f) = \frac{p(e, f)}{p(f)} = \frac{p(e)p(f | e)}{\sum_e p(e)p(f | e)}$$

and

$$\operatorname{argmax}_e p(e | f) = \operatorname{argmax}_e p(e)p(f | e)$$

Roadmap for the Next Few Lectures

- ▶ IBM Models 1 and 2
- ▶ *Phrase-based* models

Overview

- ▶ IBM Model 1
- ▶ IBM Model 2
- ▶ EM Training of Models 1 and 2

IBM Model 1: Alignments

- ▶ How do we model $p(f \mid e)$?
- ▶ English sentence e has l words $e_1 \dots e_l$,
French sentence f has m words $f_1 \dots f_m$.
- ▶ An alignment a identifies which English word each French word originated from
- ▶ Formally, an alignment a is $\{a_1, \dots, a_m\}$, where each $a_j \in \{0 \dots l\}$.
- ▶ There are $(l + 1)^m$ possible alignments.

IBM Model 1: Alignments

- ▶ e.g., $l = 6, m = 7$

$e =$ And the program has been implemented

$f =$ Le programme a ete mis en application

- ▶ One alignment is
 $\{2, 3, 4, 5, 6, 6, 6\}$

- ▶ Another (bad!) alignment is
 $\{1, 1, 1, 1, 1, 1, 1\}$

Alignments in the IBM Models

- ▶ We'll define models for $p(a | e, m)$ and $p(f | a, e, m)$, giving

$$p(f, a | e, m) = p(a | e, m)p(f | a, e, m)$$

- ▶ Also,

$$p(f | e, m) = \sum_{a \in \mathcal{A}} p(a | e, m)p(f | a, e, m)$$

where \mathcal{A} is the set of all possible alignments

A By-Product: Most Likely Alignments

- Once we have a model $p(f, a | e, m) = p(a | e)p(f | a, e, m)$ we can also calculate

$$p(a | f, e, m) = \frac{p(f, a | e, m)}{\sum_{a \in \mathcal{A}} p(f, a | e, m)}$$

for any alignment a

- For a given f, e pair, we can also compute the most likely alignment,

$$a^* = \arg \max_a p(a | f, e, m)$$

- Nowadays, the original IBM models are rarely (if ever) used for translation, but they are used for recovering alignments

An Example Alignment

French:

le conseil a rendu son avis , et nous devons à présent adopter un nouvel avis sur la base de la première position .

English:

the council has stated its position , and now , on the basis of the first position , we again have to give our opinion .

Alignment:

the/le council/conseil has/à stated/rendu its/son position/avis ,/, and/et now/présent ,/NULL on/sur the/le basis/base of/de the/la first/première position/position ,/NULL we/nous again/NULL have/devons to/a give/adopter our/nouvel opinion/avis ./.

IBM Model 1: Alignments

- ▶ In IBM model 1 all alignments a are equally likely:

$$p(a \mid e, m) = \frac{1}{(l + 1)^m}$$

- ▶ This is a **major** simplifying assumption, but it gets things started...

IBM Model 1: Translation Probabilities

- ▶ Next step: come up with an estimate for

$$p(f \mid a, e, m)$$

- ▶ In model 1, this is:

$$p(f \mid a, e, m) = \prod_{j=1}^m t(f_j \mid e_{a_j})$$

- ▶ e.g., $l = 6$, $m = 7$

$e = \text{And the program has been implemented}$

$f = \text{Le programme a ete mis en application}$

- ▶ $a = \{2, 3, 4, 5, 6, 6, 6\}$

$$\begin{aligned} p(f \mid a, e) &= t(\text{Le} \mid \text{the}) \times \\ &\quad t(\text{programme} \mid \text{program}) \times \\ &\quad t(a \mid \text{has}) \times \\ &\quad t(\text{ete} \mid \text{been}) \times \\ &\quad t(\text{mis} \mid \text{implemented}) \times \\ &\quad t(\text{en} \mid \text{implemented}) \times \\ &\quad t(\text{application} \mid \text{implemented}) \end{aligned}$$

IBM Model 1: The Generative Process

To generate a French string f from an English string e :

- ▶ **Step 1:** Pick an alignment a with probability $\frac{1}{(l+1)^m}$
- ▶ **Step 2:** Pick the French words with probability

$$p(f \mid a, e, m) = \prod_{j=1}^m t(f_j \mid e_{a_j})$$

The final result:

$$p(f, a \mid e, m) = p(a \mid e, m) \times p(f \mid a, e, m) = \frac{1}{(l+1)^m} \prod_{j=1}^m t(f_j \mid e_{a_j})$$

An Example Lexical Entry

English	French	Probability
position	position	0.756715
position	situation	0.0547918
position	mesure	0.0281663
position	vue	0.0169303
position	point	0.0124795
position	attitude	0.0108907

... de la **situation** au niveau des négociations de l'OMPI ...

... of the current **position** in the WIPO negotiations ...

nous ne sommes pas en **mesure** de décider , ...

we are not in a **position** to decide , ...

... le **point de vue** de la commission face à ce problème complexe .

... the commission's **position** on this complex problem .

Overview

- ▶ IBM Model 1
- ▶ IBM Model 2
- ▶ EM Training of Models 1 and 2

IBM Model 2

- ▶ Only difference: we now introduce **alignment** or **distortion** parameters

$\mathbf{q}(i \mid j, l, m)$ = Probability that j 'th French word is connected to i 'th English word, given sentence lengths of e and f are l and m respectively

- ▶ Define

$$p(a \mid e, m) = \prod_{j=1}^m \mathbf{q}(a_j \mid j, l, m)$$

where $a = \{a_1, \dots, a_m\}$

- ▶ Gives

$$p(f, a \mid e, m) = \prod_{j=1}^m \mathbf{q}(a_j \mid j, l, m) \mathbf{t}(f_j \mid e_{a_j})$$

An Example

$$l = 6$$

$$m = 7$$

e = And the program has been implemented

f = Le programme a ete mis en application

$$a = \{2, 3, 4, 5, 6, 6, 6\}$$

$$\begin{aligned} p(a \mid e, 7) &= \mathbf{q}(2 \mid 1, 6, 7) \times \\ &\quad \mathbf{q}(3 \mid 2, 6, 7) \times \\ &\quad \mathbf{q}(4 \mid 3, 6, 7) \times \\ &\quad \mathbf{q}(5 \mid 4, 6, 7) \times \\ &\quad \mathbf{q}(6 \mid 5, 6, 7) \times \\ &\quad \mathbf{q}(6 \mid 6, 6, 7) \times \\ &\quad \mathbf{q}(6 \mid 7, 6, 7) \end{aligned}$$

An Example

$l = 6$

$m = 7$

$e = \text{And the program has been implemented}$

$f = \text{Le programme a ete mis en application}$

$a = \{2, 3, 4, 5, 6, 6, 6\}$

$$\begin{aligned} p(f | a, e, 7) &= \mathbf{t}(Le | the) \times \\ &\quad \mathbf{t}(\text{programme} | \text{program}) \times \\ &\quad \mathbf{t}(a | \text{has}) \times \\ &\quad \mathbf{t}(\text{ete} | \text{been}) \times \\ &\quad \mathbf{t}(\text{mis} | \text{implemented}) \times \\ &\quad \mathbf{t}(\text{en} | \text{implemented}) \times \\ &\quad \mathbf{t}(\text{application} | \text{implemented}) \end{aligned}$$

IBM Model 2: The Generative Process

To generate a French string f from an English string e :

- ▶ **Step 1:** Pick an alignment $a = \{a_1, a_2 \dots a_m\}$ with probability

$$\prod_{j=1}^m \mathbf{q}(a_j \mid j, l, m)$$

- ▶ **Step 3:** Pick the French words with probability

$$p(f \mid a, e, m) = \prod_{j=1}^m \mathbf{t}(f_j \mid e_{a_j})$$

The final result:

$$p(f, a \mid e, m) = p(a \mid e, m)p(f \mid a, e, m) = \prod_{j=1}^m \mathbf{q}(a_j \mid j, l, m)\mathbf{t}(f_j \mid e_{a_j})$$

Recovering Alignments

- ▶ If we have parameters q and t , we can easily recover the most likely alignment for any sentence pair
- ▶ Given a sentence pair $e_1, e_2, \dots, e_l, f_1, f_2, \dots, f_m$, define

$$a_j = \arg \max_{a \in \{0 \dots l\}} q(a|j, l, m) \times t(f_j|e_a)$$

for $j = 1 \dots m$

e = And the program has been implemented

f = Le programme a ete mis en application

Overview

- ▶ IBM Model 1
- ▶ IBM Model 2
- ▶ EM Training of Models 1 and 2

The Parameter Estimation Problem

- ▶ Input to the parameter estimation algorithm: $(e^{(k)}, f^{(k)})$ for $k = 1 \dots n$. Each $e^{(k)}$ is an English sentence, each $f^{(k)}$ is a French sentence
- ▶ Output: parameters $t(f|e)$ and $q(i|j, l, m)$
- ▶ A key challenge: **we do not have alignments on our training examples**, e.g.,

$e^{(100)} =$ And the program has been implemented

$f^{(100)} =$ Le programme a ete mis en application

Parameter Estimation if the Alignments are Observed

- ▶ First: case where alignments are observed in training data.

E.g.,

$$e^{(100)} = \text{And the program has been implemented}$$

$$f^{(100)} = \text{Le programme a ete mis en application}$$

$$a^{(100)} = \langle 2, 3, 4, 5, 6, 6, 6 \rangle$$

- ▶ Training data is $(e^{(k)}, f^{(k)}, a^{(k)})$ for $k = 1 \dots n$. Each $e^{(k)}$ is an English sentence, each $f^{(k)}$ is a French sentence, each $a^{(k)}$ is an alignment
- ▶ Maximum-likelihood parameter estimates in this case are trivial:

$$t_{ML}(f|e) = \frac{\text{Count}(e, f)}{\text{Count}(e)} \quad q_{ML}(j|i, l, m) = \frac{\text{Count}(j|i, l, m)}{\text{Count}(i, l, m)}$$

Input: A training corpus $(f^{(k)}, e^{(k)}, a^{(k)})$ for $k = 1 \dots n$, where $f^{(k)} = f_1^{(k)} \dots f_{m_k}^{(k)}$, $e^{(k)} = e_1^{(k)} \dots e_{l_k}^{(k)}$, $a^{(k)} = a_1^{(k)} \dots a_{m_k}^{(k)}$.

Algorithm:

- ▶ Set all counts $c(\dots) = 0$
- ▶ For $k = 1 \dots n$
 - ▶ For $i = 1 \dots m_k$, For $j = 0 \dots l_k$,

$$\begin{aligned}
 c(e_j^{(k)}, f_i^{(k)}) &\leftarrow c(e_j^{(k)}, f_i^{(k)}) + \delta(k, i, j) \\
 c(e_j^{(k)}) &\leftarrow c(e_j^{(k)}) + \delta(k, i, j) \\
 c(j|i, l, m) &\leftarrow c(j|i, l, m) + \delta(k, i, j) \\
 c(i, l, m) &\leftarrow c(i, l, m) + \delta(k, i, j)
 \end{aligned}$$

where $\delta(k, i, j) = 1$ if $a_i^{(k)} = j$, 0 otherwise.

Output: $t_{ML}(f|e) = \frac{c(e,f)}{c(e)}$, $q_{ML}(j|i, l, m) = \frac{c(j|i,l,m)}{c(i,l,m)}$

Parameter Estimation with the EM Algorithm

- ▶ Training examples are $(e^{(k)}, f^{(k)})$ for $k = 1 \dots n$. Each $e^{(k)}$ is an English sentence, each $f^{(k)}$ is a French sentence
- ▶ The algorithm is related to algorithm when alignments are observed, but two key differences:
 1. The algorithm is *iterative*. We start with some initial (e.g., random) choice for the q and t parameters. At each iteration we compute some “counts” based on the data together with our current parameter estimates. We then re-estimate our parameters with these counts, and iterate.
 2. We use the following definition for $\delta(k, i, j)$ at each iteration:

$$\delta(k, i, j) = \frac{q(j|i, l_k, m_k) t(f_i^{(k)}|e_j^{(k)})}{\sum_{j=0}^{l_k} q(j|i, l_k, m_k) t(f_i^{(k)}|e_j^{(k)})}$$

Input: A training corpus $(f^{(k)}, e^{(k)})$ for $k = 1 \dots n$, where $f^{(k)} = f_1^{(k)} \dots f_{m_k}^{(k)}$, $e^{(k)} = e_1^{(k)} \dots e_{l_k}^{(k)}$.

Initialization: Initialize $t(f|e)$ and $q(j|i, l, m)$ parameters (e.g., to random values).

For $s = 1 \dots S$

- ▶ Set all counts $c(\dots) = 0$
- ▶ For $k = 1 \dots n$
 - ▶ For $i = 1 \dots m_k$, For $j = 0 \dots l_k$

$$\begin{aligned} c(e_j^{(k)}, f_i^{(k)}) &\leftarrow c(e_j^{(k)}, f_i^{(k)}) + \delta(k, i, j) \\ c(e_j^{(k)}) &\leftarrow c(e_j^{(k)}) + \delta(k, i, j) \\ c(j|i, l, m) &\leftarrow c(j|i, l, m) + \delta(k, i, j) \\ c(i, l, m) &\leftarrow c(i, l, m) + \delta(k, i, j) \end{aligned}$$

where

$$\delta(k, i, j) = \frac{q(j|i, l_k, m_k) t(f_i^{(k)}|e_j^{(k)})}{\sum_{j=0}^{l_k} q(j|i, l_k, m_k) t(f_i^{(k)}|e_j^{(k)})}$$

- ▶ Recalculate the parameters:

$$t(f|e) = \frac{c(e, f)}{c(e)} \quad q(j|i, l, m) = \frac{c(j|i, l, m)}{c(i, l, m)}$$

$$\delta(k, i, j) = \frac{q(j|i, l_k, m_k) t(f_i^{(k)}|e_j^{(k)})}{\sum_{j=0}^{l_k} q(j|i, l_k, m_k) t(f_i^{(k)}|e_j^{(k)})}$$

$e^{(100)}$ = And the program has been implemented

$f^{(100)}$ = Le programme a ete mis en application

Justification for the Algorithm

- ▶ Training examples are $(e^{(k)}, f^{(k)})$ for $k = 1 \dots n$. Each $e^{(k)}$ is an English sentence, each $f^{(k)}$ is a French sentence
- ▶ The log-likelihood function:

$$L(t, q) = \sum_{k=1}^n \log p(f^{(k)}|e^{(k)}) = \sum_{k=1}^n \log \sum_a p(f^{(k)}, a|e^{(k)})$$

- ▶ The maximum-likelihood estimates are

$$\arg \max_{t,q} L(t, q)$$

- ▶ The EM algorithm will converge to a *local maximum* of the log-likelihood function

Summary

- ▶ Key ideas in the IBM translation models:
 - ▶ Alignment variables
 - ▶ Translation parameters, e.g., $t(\text{chien}|\text{dog})$
 - ▶ Distortion parameters, e.g., $q(2|1, 6, 7)$
- ▶ The EM algorithm: an iterative algorithm for training the q and t parameters
- ▶ Once the parameters are trained, we can recover the most likely alignments on our training examples

e = And the program has been implemented

f = Le programme a ete mis en application

Phrase-Based Translation

Michael Collins, Columbia University

Roadmap for the Next Few Lectures

- ▶ Last time: IBM Models 1 and 2
- ▶ Today: *phrase-based* models

Overview

- ▶ Learning phrases from alignments
- ▶ A phrase-based model
- ▶ Decoding in phrase-based models

Phrase-Based Models

- ▶ First stage in training a phrase-based model is extraction of a *phrase-based (PB) lexicon*
- ▶ A PB lexicon pairs strings in one language with strings in another language, e.g.,

nach Kanada \leftrightarrow in Canada

zur Konferenz \leftrightarrow to the conference

Morgen \leftrightarrow tomorrow

fliege \leftrightarrow will fly

...

An Example (from tutorial by Koehn and Knight)

- ▶ A training example (Spanish/English sentence pair):

Spanish: Maria no daba una bofetada a la bruja verde

English: Mary did not slap the green witch

- ▶ Some (not all) phrase pairs extracted from this example:

(Maria ↔ Mary), (bruja ↔ witch), (verde ↔ green),
(no ↔ did not), (no daba una bofetada ↔ did not slap),
(daba una bofetada a la ↔ slap the)

- ▶ We'll see how to do this using *alignments* from the IBM models (e.g., from IBM model 2)

Recap: IBM Model 2

- ▶ IBM model 2 defines a distribution $p(a, f|e, m)$ where f is foreign (French) sentence, e is an English sentence, a is an *alignment*, m is the length of the foreign sentence
- ▶ A useful by-product: once we've trained the model, for any (f, e) pair, we can calculate

$$a^* = \arg \max_a p(a|f, e, m) = \arg \max_a p(a, f|e, m)$$

under the model. a^* is the **most likely alignment**

English: Mary did not slap the green witch

Spanish: Maria no daba una bofetada a la bruja verde

Representation as Alignment Matrix

	Maria	no	daba	una	bof'	a	la	bruja	verde
Mary	●								
did						●			
not		●							
slap			●	●	●				
the							●		
green								●	
witch								●	

(Note: “bof”’ = “bofetada”)

In IBM model 2, each foreign (Spanish) word is aligned to exactly one English word. The matrix shows these alignments.

Finding Alignment Matrices

- ▶ Step 1: train IBM model 2 for $p(f | e)$, and come up with most likely alignment for each (e, f) pair
- ▶ Step 2: train IBM model 2 for $p(e | f)$ and come up with most likely alignment for each (e, f) pair
- ▶ We now have two alignments:
take intersection of the two alignments as a starting point

Alignment from $p(f | e)$ model:

	Maria	no	daba	una	bof'	a	la	bruja	verde
Mary	●								
did						●			
not		●							
slap			●	●	●				
the						●			
green								●	
witch							●		

Alignment from $p(e | f)$ model:

	Maria	no	daba	una	bof'	a	la	bruja	verde
Mary	●								
did		●							
not		●							
slap				●					
the						●			
green								●	
witch							●		

Intersection of the two alignments:

	Maria	no	daba	una	b of'	a	la	bruja	verde
Mary	●								
did									
not		●							
slap					●				
the						●			
green								●	
witch							●		

The intersection of the two alignments has been found to be a very reliable starting point

Heuristics for Growing Alignments

- ▶ Only explore alignment in **union** of $p(f \mid e)$ and $p(e \mid f)$ alignments
- ▶ Add one alignment point at a time
- ▶ Only add alignment points which align a word that currently has no alignment
- ▶ At first, restrict ourselves to alignment points that are “neighbors” (adjacent or diagonal) of current alignment points
- ▶ Later, consider other alignment points

The final alignment, created by taking the intersection of the two alignments, then adding new points using the growing heuristics:

	Maria	no	daba	una	bof'	a	la	bruja	verde
Mary	●								
did		●							
not		●							
slap			●	●	●				
the						●	●		
green								●	
witch								●	

Note that the alignment is no longer many-to-one: potentially multiple Spanish words can be aligned to a single English word, and vice versa.

Extracting Phrase Pairs from the Alignment Matrix

	Maria	no	daba	una	bof'	a	la	bruja	verde
Mary	●								
did		●							
not		●							
slap			●	●	●				
the						●	●		
green								●	
witch								●	

- ▶ A phrase-pair consists of a sequence of English words, e , paired with a sequence of foreign words, f
- ▶ A phrase-pair (e, f) is *consistent* if: 1) there is at least one word in e aligned to a word in f ; 2) there are no words in f aligned to words outside e ; 3) there are no words in e aligned to words outside f
e.g., (**Mary did not**, **Maria no**) is consistent. (**Mary did**, **Maria no**) is *not* consistent
- ▶ We extract all consistent phrase pairs from the training example.

Probabilities for Phrase Pairs

- ▶ For any phrase pair (f, e) extracted from the training data, we can calculate

$$t(f|e) = \frac{\text{Count}(f, e)}{\text{Count}(e)}$$

e.g.,

$$t(\text{daba una bofetada} | \text{slap}) = \frac{\text{Count}(\text{daba una bofetada, slap})}{\text{Count}(\text{slap})}$$

An Example Phrase Translation Table

An example from Koehn, EACL 2006 tutorial. (Note that we have $t(e|f)$ not $t(f|e)$ in this example.)

- ▶ Phrase Translations for *den Vorschlag*

English	$t(e f)$	English	$t(e f)$
the proposal	0.6227	the suggestions	0.0114
's proposal	0.1068	the proposed	0.0114
a proposal	0.0341	the motion	0.0091
the idea	0.0250	the idea of	0.0091
this proposal	0.0227	the proposal ,	0.0068
proposal	0.0205	its proposal	0.0068
of the proposal	0.0159	it	0.0068
the proposals	0.0159

Overview

- ▶ Learning phrases from alignments
- ▶ A phrase-based model
- ▶ Decoding in phrase-based models

Phrase-Based Systems: A Sketch

Today

Heute werden wir über die Wiedereröffnung
des Mont-Blanc-Tunnels diskutieren

$$\text{Score} = \underbrace{\log q(\text{Today} \mid *, *)}_{\text{Language model}}$$

$$+ \underbrace{\log t(\text{Heute} \mid \text{Today})}_{\text{Phrase model}}$$

$$+ \underbrace{\eta \times 0}_{\text{Distortion model}}$$

Phrase-Based Systems: A Sketch

Today we shall be

Heute werden wir über die Wiedereröffnung
des Mont-Blanc-Tunnels diskutieren

$$\text{Score} = \underbrace{\log q(\text{we}^*, \text{Today}) + \log q(\text{shall} | \text{Today, we}) + \log q(\text{be} | \text{we, shall})}_{\text{Language model}} \\ + \underbrace{\log t(\text{werden wir} | \text{we shall be})}_{\text{Phrase model}} \\ + \underbrace{\eta \times 0}_{\text{Distortion model}}$$

Phrase-Based Systems: A Sketch

Today we shall be debating

Heute werden wir über die Wiedereröffnung
des Mont-Blanc-Tunnels diskutieren

$$\text{Score} = \underbrace{\log q(\text{debating} | \text{shall, be})}_{\text{Language model}}$$

$$+ \underbrace{\log t(\text{diskutieren} | \text{debating})}_{\text{Phrase model}}$$

$$+ \underbrace{\eta \times 6}_{\text{Distortion model}}$$

Phrase-Based Systems: A Sketch

Today we shall be debating the reopening

Heute werden wir über die Wiedereröffnung
des Mont-Blanc-Tunnels diskutieren

Phrase-Based Systems: A Sketch

Today we shall be debating the reopening
of the Mont Blanc tunnel

Heute werden wir über die Wiedereröffnung
des Mont-Blanc-Tunnels diskutieren

Decoding with Phrase-Based Translation Models

Michael Collins, Columbia University

Phrase-based Translation

An example sentence:

wir müssen auch diese kritik ernst nehmen

A phrase-based lexicon contains phrase entries (f, e) where f is a sequence of one or more foreign words, e is a sequence of one or more English words.

Example phrase entries that are relevant to our example:

(wir müssen, we must)

(wir müssen auch, we must also)

(ernst, seriously)

Each phrase (f, e) has a score $g(f, e)$. E.g.,

$$g(f, e) = \log \left(\frac{\text{Count}(f, e)}{\text{Count}(e)} \right)$$

Phrase-based Models: Definitions

- ▶ A phrase-based model consists of:
 1. A phrase-based lexicon, consisting of entries (f, e) such as

(wir müssen, we must)

Each lexical entry has a score $g(f, e)$, e.g.,

$$g(\text{wir müssen, we must}) = \log \left(\frac{\text{Count}(\text{wir müssen, we must})}{\text{Count}(\text{we must})} \right)$$

2. A trigram language model, with parameters $q(w|u, v)$. E.g., $q(\text{also}|\text{we, must})$.
3. A “distortion parameter” η (typically negative).

Phrase-based Translation: Definitions

An example sentence:

wir müssen auch diese kritik ernst nehmen

- ▶ For a particular input (source-language) sentence $x_1 \dots x_n$, a phrase is a tuple (s, t, e) , signifying that the subsequence $x_s \dots x_t$ in the source language sentence can be translated as the target-language string e , using an entry from the phrase-based lexicon. E.g., (1, 2, we must)
- ▶ \mathcal{P} is the set of all phrases for a sentence.
- ▶ For any phrase p , $s(p)$, $t(p)$ and $e(p)$ are its three components. $g(p)$ is the score for a phrase.

Definitions

- ▶ A derivation y is a finite sequence of phrases, p_1, p_2, \dots, p_L , where each p_j for $j \in \{1 \dots L\}$ is a member of \mathcal{P} .
- ▶ The length L can be any positive integer value.
- ▶ For any derivation y we use $e(y)$ to refer to the underlying translation defined by y . E.g.,

$y = (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

and

$e(y) = \text{we must also take this criticism seriously}$

Valid Derivations

- ▶ For an input sentence $x = x_1 \dots x_n$, we use $\mathcal{Y}(x)$ to refer to the set of valid derivations for x .
- ▶ $\mathcal{Y}(x)$ is the set of all finite length sequences of phrases $p_1 p_2 \dots p_L$ such that:
 - ▶ Each p_k for $k \in \{1 \dots L\}$ is a member of the set of phrases \mathcal{P} for $x_1 \dots x_n$.
 - ▶ Each word in x is translated exactly once.
 - ▶ For all $k \in \{1 \dots (L - 1)\}$, $|t(p_k) + 1 - s(p_{k+1})| \leq d$ where $d \geq 0$ is a parameter of the model. In addition, we must have $|1 - s(p_1)| \leq d$

Examples

wir müssen auch diese kritik ernst nehmen

$y = (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

Examples

wir müssen auch diese kritik ernst nehmen

$y = (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

$y = (1, 3, \text{we must also}), (1, 2, \text{we must}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

Examples

wir müssen auch diese kritik ernst nehmen

$y = (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

$y = (1, 3, \text{we must also}), (1, 2, \text{we must}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

$y = (1, 2, \text{we must}), (7, 7, \text{take}), (3, 3, \text{also}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

Scoring Derivations

The optimal translation under the model for a source-language sentence x will be

$$\arg \max_{y \in \mathcal{Y}(x)} f(y)$$

In phrase-based systems, the score for any derivation y is calculated as follows:

$$h(e(y)) + \sum_{k=1}^L g(p_k) + \sum_{k=0}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

where the parameter η is the distortion penalty (typically negative). (We define $t(p_0) = 0$).

$h(e(y))$ is the trigram language model score. $g(p_k)$ is the phrase-based score for p_k .

An Example

wir müssen auch diese kritik ernst nehmen

$y = (1, 3, \text{we must also}), (7, 7, \text{take}), (4, 5, \text{this criticism}), (6, 6, \text{seriously})$

Decoding Algorithm: Definitions

- ▶ A state is a tuple

$$(e_1, e_2, b, r, \alpha)$$

where e_1, e_2 are English words, b is a bit-string of length n , r is an integer specifying the end-point of the last phrase in the state, and α is the score for the state.

- ▶ The initial state is

$$q_0 = (*, *, 0^n, 0, 0)$$

where 0^n is bit-string of length n , with n zeroes.

States, and the Search Space

wir müssen auch diese kritik ernst nehmen

(*, *, 0000000, 0, 0)

Transitions

- ▶ We have $ph(q)$ for any state q , which returns set of phrases that are allowed to follow state $q = (e_1, e_2, b, r, \alpha)$.
- ▶ For a phrase p to be a member of $ph(q)$, it must satisfy the following conditions:
 - ▶ p must not overlap with the bit-string b . I.e., we need $b_i = 0$ for $i \in \{s(p) \dots t(p)\}$.
 - ▶ The distortion limit must not be violated. More formally, we must have $|r + 1 - s(p)| \leq d$ where d is the distortion limit.

An Example of the Transition Function

wir müssen auch diese kritik ernst nehmen

(must, also, 1110000, 3, -2.5)

An Example of the Transition Function

wir müssen auch diese kritik ernst nehmen

(must, also, 1110000, 3, -2.5)

In addition, we define $\text{next}(q, p)$ to be the state formed by combining state q with phrase p .

The *next* function

Formally, if $q = (e_1, e_2, b, r, \alpha)$, and $p = (s, t, \epsilon_1 \dots \epsilon_M)$, then $\text{next}(q, p)$ is the state $q' = (e'_1, e'_2, b', r', \alpha')$ defined as follows:

- ▶ First, for convenience, define $\epsilon_{-1} = e_1$, and $\epsilon_0 = e_2$.
- ▶ Define $e'_1 = \epsilon_{M-1}$, $e'_2 = \epsilon_M$.
- ▶ Define $b'_i = 1$ for $i \in \{s \dots t\}$. Define $b'_i = b_i$ for $i \notin \{s \dots t\}$
- ▶ Define $r' = t$
- ▶ Define

$$\alpha' = \alpha + g(p) + \sum_{i=1}^M \log q(\epsilon_i | \epsilon_{i-2}, \epsilon_{i-1}) + \eta \times |r + 1 - s|$$

The Equality Function

- ▶ The function

$$\text{eq}(q, q')$$

returns true or false.

- ▶ Assuming $q = (e_1, e_2, b, r, \alpha)$, and $q' = (e'_1, e'_2, b', r', \alpha')$,
 $\text{eq}(q, q')$ is true if and only if $e_1 = e'_1$, $e_2 = e'_2$, $b = b'$ and
 $r = r'$.

The Decoding Algorithm

- ▶ Inputs: sentence $x_1 \dots x_n$. Phrase-based model $(\mathcal{L}, h, d, \eta)$.
The phrase-based model defines the functions $ph(q)$ and $\text{next}(q, p)$.
- ▶ Initialization: set $Q_0 = \{q_0\}$, $Q_i = \emptyset$ for $i = 1 \dots n$.
- ▶ For $i = 0 \dots n - 1$
 - ▶ For each state $q \in \text{beam}(Q_i)$, for each phrase $p \in ph(q)$:
 - (1) $q' = \text{next}(q, p)$
 - (2) Add(Q_i, q', q, p) where $i = \text{len}(q')$
 - ▶ Return: highest scoring state in Q_n . Backpointers can be used to find the underlying sequence of phrases (and the translation).

An Example

wir müssen auch diese kritik ernst nehmen

(*, *, 0000000, 0, 0)

Definition of Add(Q, q', q, p)

- ▶ If there is some $q'' \in Q$ such that $eq(q'', q') = \text{True}$:
 - ▶ If $\alpha(q') > \alpha(q'')$
 - ▶ $Q = \{q'\} \cup Q \setminus \{q''\}$
 - ▶ set $bp(q') = (q, p)$
 - ▶ Else return
- ▶ Else
 - ▶ $Q = Q \cup \{q'\}$
 - ▶ set $bp(q') = (q, p)$

Definition of beam(Q)

Define

$$\alpha^* = \arg \max_{q \in Q} \alpha(q)$$

i.e., α^* is the highest score for any state in Q .

Define $\beta \geq 0$ to be the *beam-width* parameter

Then

$$\text{beam}(Q) = \{q \in Q : \alpha(q) \geq \alpha^* - \beta\}$$

Parsing, and Context-Free Grammars

Michael Collins, Columbia University

Overview

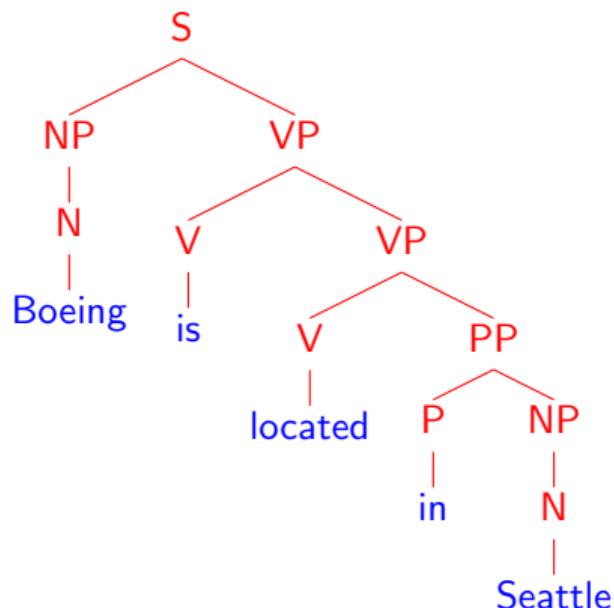
- ▶ An introduction to the parsing problem
- ▶ Context free grammars
- ▶ A brief(!) sketch of the syntax of English
- ▶ Examples of ambiguous structures

Parsing (Syntactic Structure)

INPUT:

Boeing is located in Seattle.

OUTPUT:



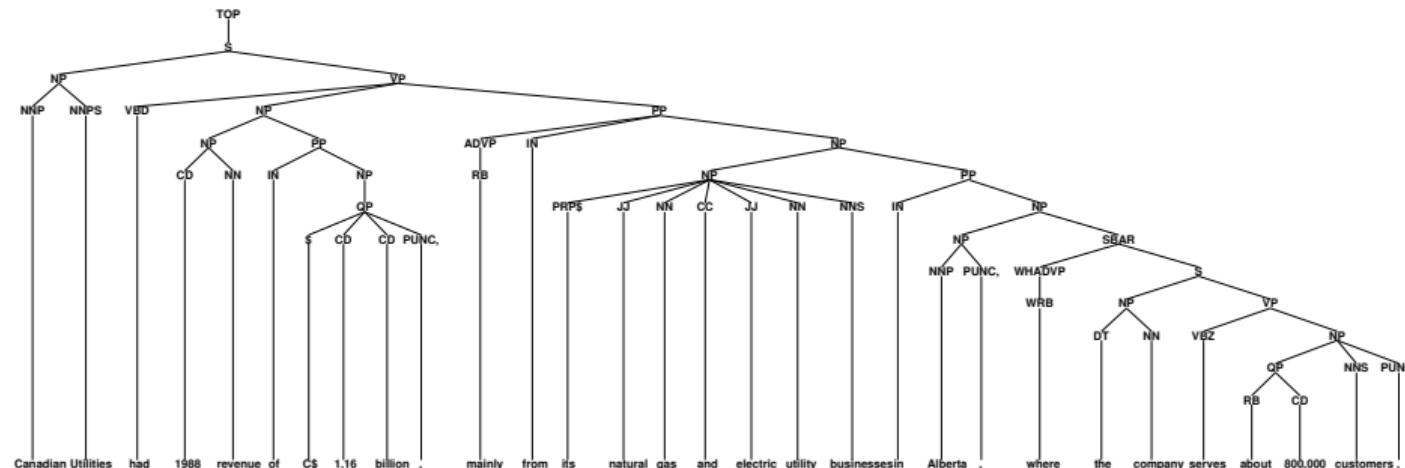
Syntactic Formalisms

- ▶ Work in formal syntax goes back to Chomsky's PhD thesis in the 1950s
- ▶ Examples of current formalisms: minimalism, lexical functional grammar (LFG), head-driven phrase-structure grammar (HPSG), tree adjoining grammars (TAG), categorial grammars

Data for Parsing Experiments

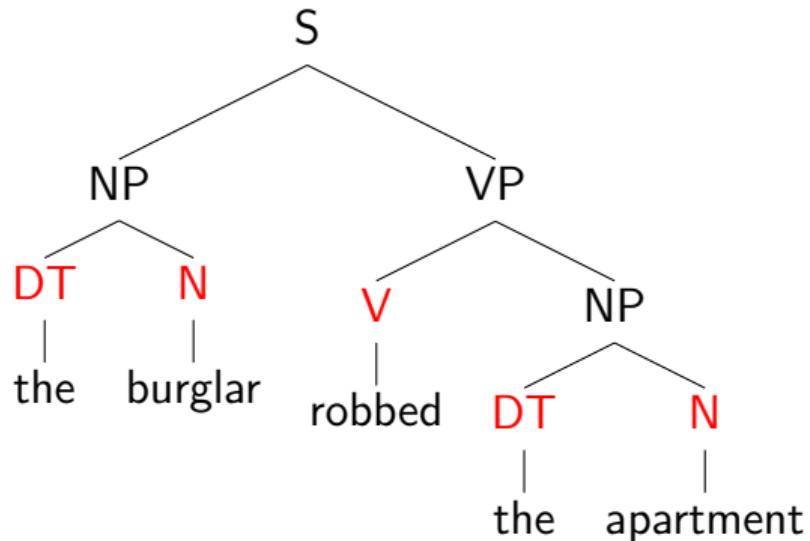
- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



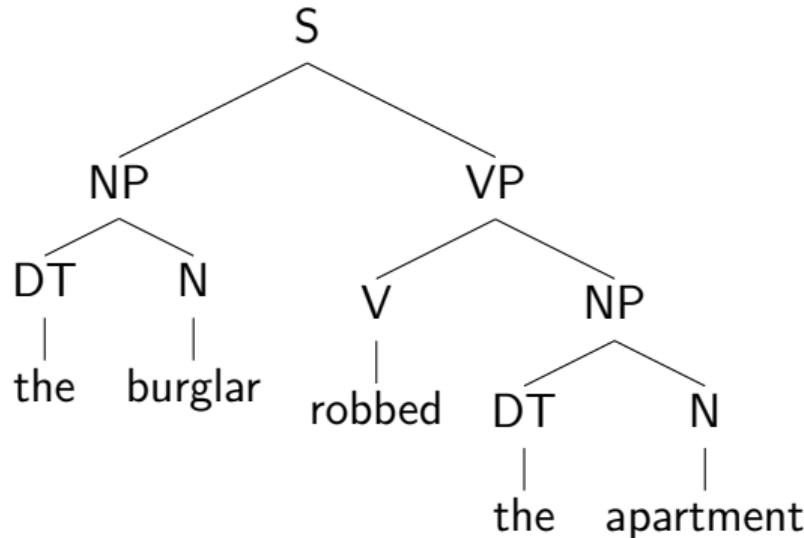
The Information Conveyed by Parse Trees

- (1) Part of speech for each word
(N = noun, V = verb, DT = determiner)



The Information Conveyed by Parse Trees (continued)

(2) Phrases



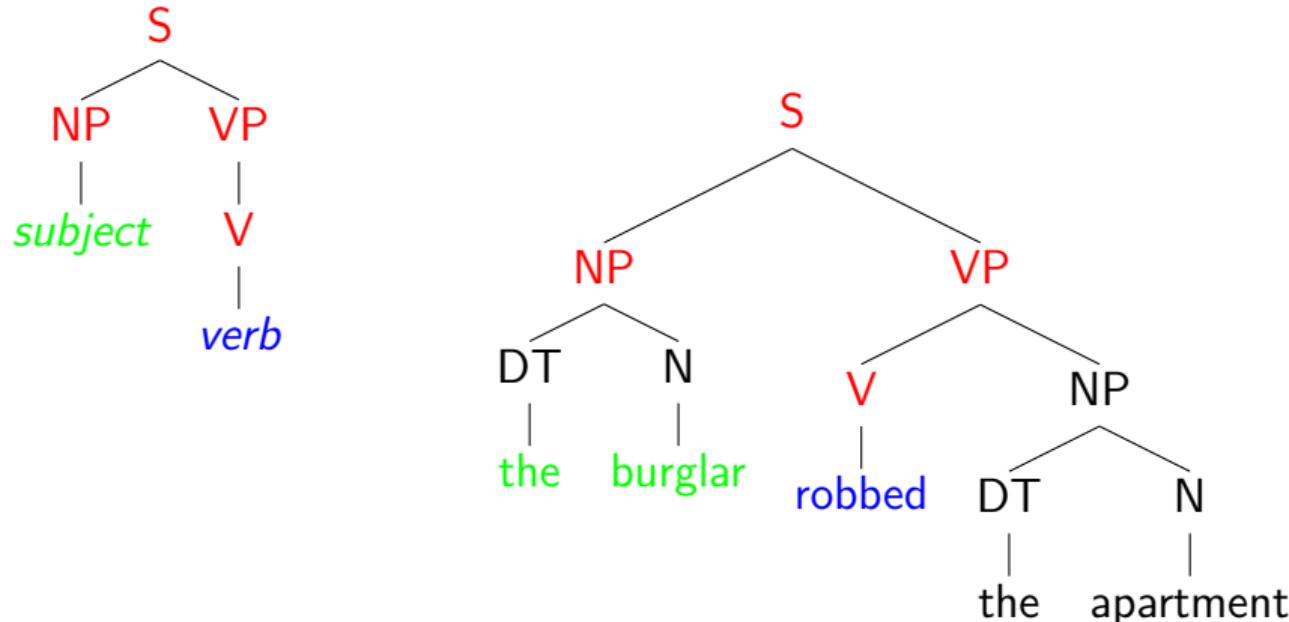
Noun Phrases (NP): "the burglar", "the apartment"

Verb Phrases (VP): "robbed the apartment"

Sentences (S): "the burglar robbed the apartment"

The Information Conveyed by Parse Trees (continued)

(3) Useful Relationships



⇒ “the burglar” is the subject of “robbed”

An Example Application: Machine Translation

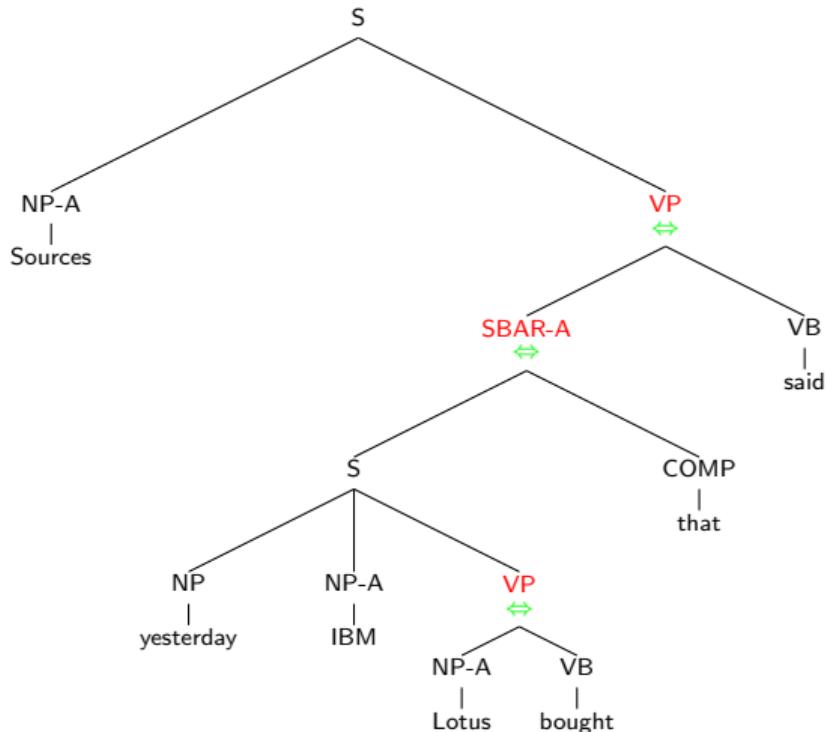
- ▶ English word order is *subject – verb – object*
- ▶ Japanese word order is *subject – object – verb*

English: IBM bought Lotus

Japanese: *IBM Lotus bought*

English: Sources said that IBM bought Lotus yesterday

Japanese: *Sources yesterday IBM Lotus bought that said*



Overview

- ▶ An introduction to the parsing problem
- ▶ Context free grammars
- ▶ A brief(!) sketch of the syntax of English
- ▶ Examples of ambiguous structures

Context-Free Grammars

Hopcroft and Ullman, 1979

A context free grammar $G = (N, \Sigma, R, S)$ where:

- ▶ N is a set of non-terminal symbols
- ▶ Σ is a set of terminal symbols
- ▶ R is a set of rules of the form $X \rightarrow Y_1Y_2\dots Y_n$
for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
- ▶ $S \in N$ is a distinguished start symbol

A Context-Free Grammar for English

$$N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$$

$$S = S$$

$$\Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\}$$

$R =$

$S \rightarrow NP \quad VP$
$VP \rightarrow Vi$
$VP \rightarrow Vt \quad NP$
$VP \rightarrow VP \quad PP$
$NP \rightarrow DT \quad NN$
$NP \rightarrow NP \quad PP$
$PP \rightarrow IN \quad NP$

$Vi \rightarrow \text{sleeps}$
$Vt \rightarrow \text{saw}$
$NN \rightarrow \text{man}$
$NN \rightarrow \text{woman}$
$NN \rightarrow \text{telescope}$
$DT \rightarrow \text{the}$
$IN \rightarrow \text{with}$
$IN \rightarrow \text{in}$

Note: S=sentence, VP=verb phrase, NP=noun phrase,
PP=prepositional phrase, DT=determiner, Vi=intransitive verb,
Vt=transitive verb, NN=noun, IN=preposition

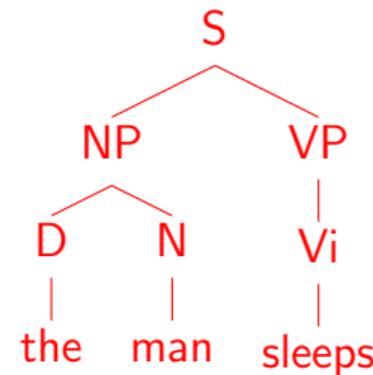
Left-Most Derivations

A left-most derivation is a sequence of strings $s_1 \dots s_n$, where

- ▶ $s_1 = S$, the start symbol
- ▶ $s_n \in \Sigma^*$, i.e. s_n is made up of terminal symbols only
- ▶ Each s_i for $i = 2 \dots n$ is derived from s_{i-1} by picking the left-most non-terminal X in s_{i-1} and replacing it by some β where $X \rightarrow \beta$ is a rule in R

For example: [S], [NP VP], [D N VP], [the N VP], [the man VP],
[the man Vi], [the man sleeps]

Representation of a derivation as a tree:



An Example

DERIVATION

S

RULES USED

An Example

DERIVATION

S

NP VP

RULES USED

$S \rightarrow NP\ VP$

An Example

DERIVATION

S

NP VP

DT N VP

RULES USED

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N$

An Example

DERIVATION

S

NP VP

DT N VP

the N VP

RULES USED

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N$

$DT \rightarrow \text{the}$

An Example

DERIVATION

S

NP VP

DT N VP

the N VP

the dog VP

RULES USED

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N$

$DT \rightarrow \text{the}$

$N \rightarrow \text{dog}$

An Example

DERIVATION

S

NP VP

DT N VP

the N VP

the dog VP

the dog VB

RULES USED

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N$

$DT \rightarrow \text{the}$

$N \rightarrow \text{dog}$

$VP \rightarrow VB$

An Example

DERIVATION

S

NP VP

DT N VP

the N VP

the dog VP

the dog VB

the dog laughs

RULES USED

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N$

$DT \rightarrow \text{the}$

$N \rightarrow \text{dog}$

$VP \rightarrow VB$

$VB \rightarrow \text{laughs}$

An Example

DERIVATION

S

NP VP

DT N VP

the N VP

the dog VP

the dog VB

the dog laughs

RULES USED

$S \rightarrow NP\ VP$

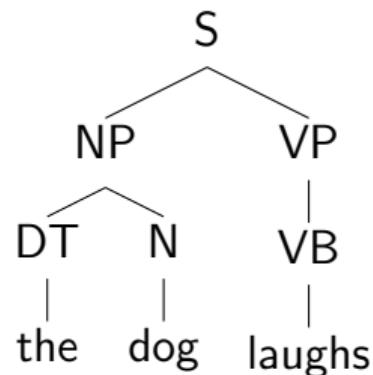
$NP \rightarrow DT\ N$

$DT \rightarrow \text{the}$

$N \rightarrow \text{dog}$

$VP \rightarrow VB$

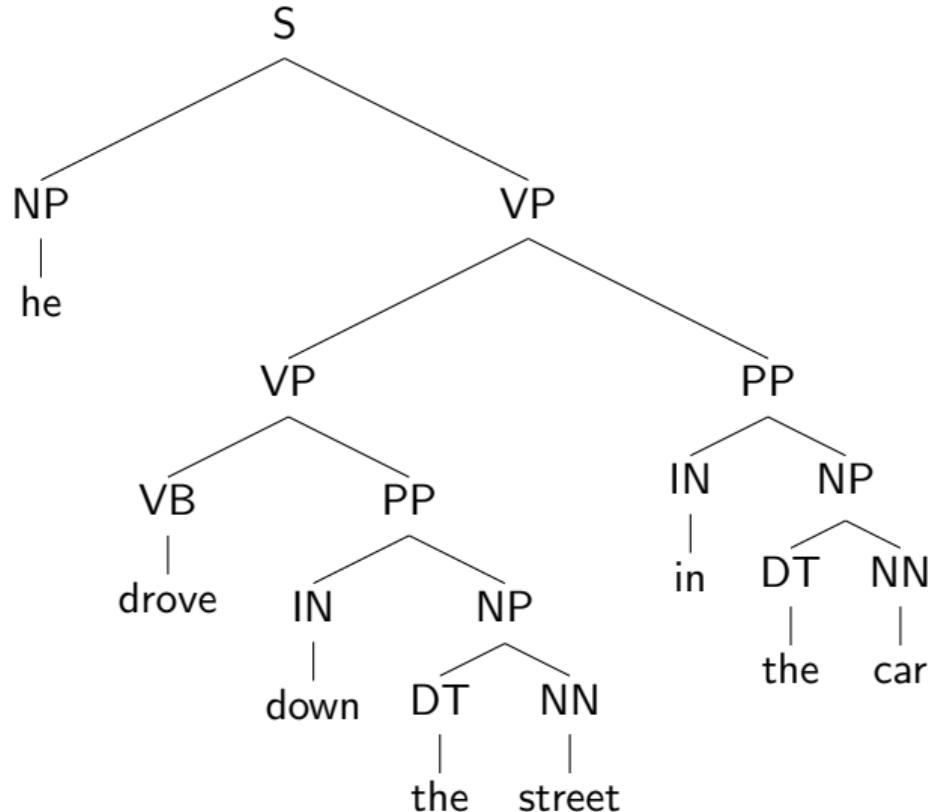
$VB \rightarrow \text{laughs}$



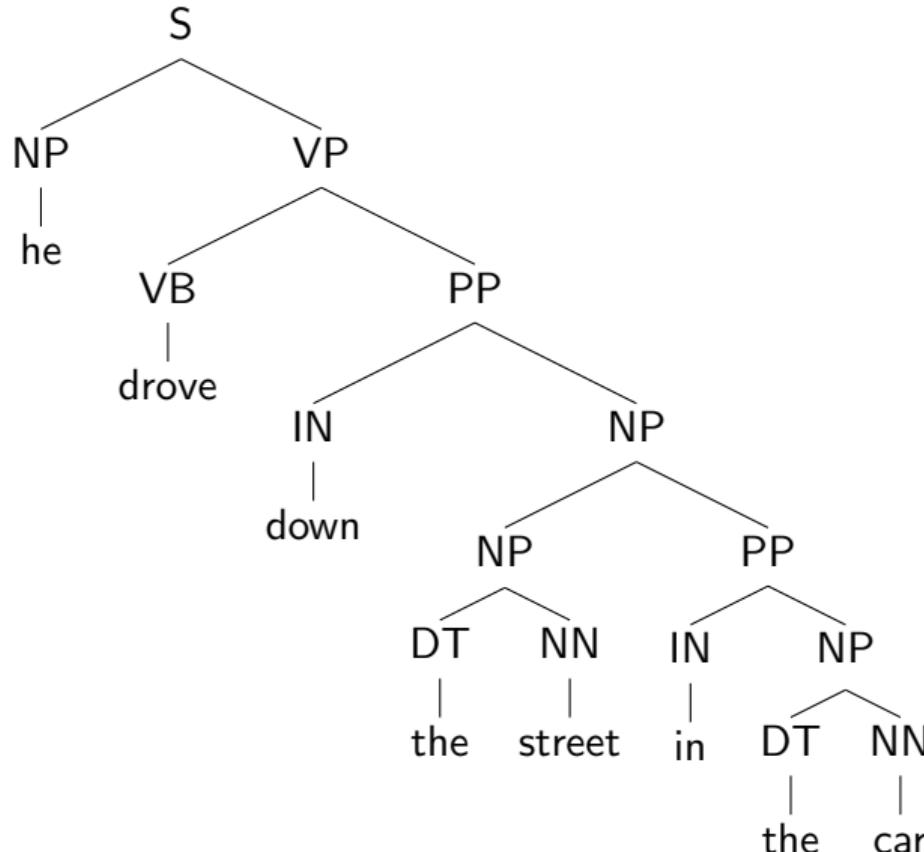
Properties of CFGs

- ▶ A CFG defines a set of possible derivations
- ▶ A string $s \in \Sigma^*$ is in the *language* defined by the CFG if there is at least one derivation that yields s
- ▶ Each string in the language generated by the CFG may have more than one derivation (“ambiguity”)

An Example of Ambiguity



An Example of Ambiguity (continued)



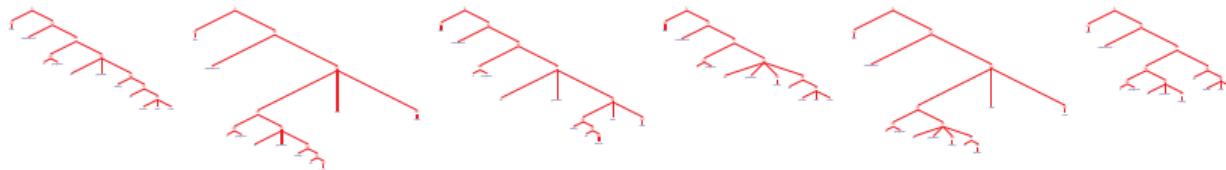
The Problem with Parsing: Ambiguity

INPUT:

She announced a program to promote safety in trucks and vans



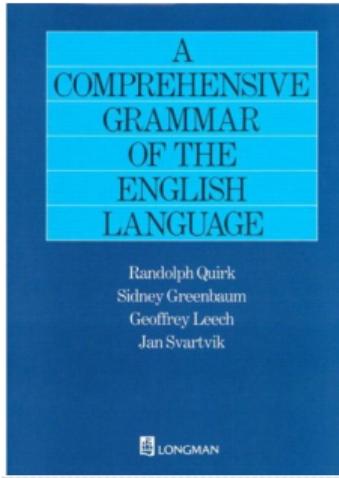
POSSIBLE OUTPUTS:



And there are more...

Overview

- ▶ An introduction to the parsing problem
- ▶ Context free grammars
- ▶ A brief(!) sketch of the syntax of English
- ▶ Examples of ambiguous structures



Product Details (from Amazon)

Hardcover: 1779 pages

Publisher: Longman; 2nd Revised edition

Language: English

ISBN-10: 0582517346

ISBN-13: 978-0582517349

Product Dimensions: 8.4 x 2.4 x 10 inches

Shipping Weight: 4.6 pounds

A Brief Overview of English Syntax

Parts of Speech (tags from the Brown corpus):

- ▶ Nouns
 - NN = singular noun e.g., man, dog, park
 - NNS = plural noun e.g., telescopes, houses, buildings
 - NNP = proper noun e.g., Smith, Gates, IBM
- ▶ Determiners
 - DT = determiner e.g., the, a, some, every
- ▶ Adjectives
 - JJ = adjective e.g., red, green, large, idealistic

A Fragment of a Noun Phrase Grammar

\bar{N}	\Rightarrow	NN
\bar{N}	\Rightarrow	NN \bar{N}
\bar{N}	\Rightarrow	JJ \bar{N}
\bar{N}	\Rightarrow	\bar{N} \bar{N}
NP	\Rightarrow	DT \bar{N}

NN	\Rightarrow	box
NN	\Rightarrow	car
NN	\Rightarrow	mechanic
NN	\Rightarrow	pigeon
DT	\Rightarrow	the
DT	\Rightarrow	a

JJ	\Rightarrow	fast
JJ	\Rightarrow	metal
JJ	\Rightarrow	idealistic
JJ	\Rightarrow	clay

Prepositions, and Prepositional Phrases

- ▶ Prepositions

IN = preposition e.g., of, in, out, beside, as

An Extended Grammar

\bar{N}	\Rightarrow	NN		NN	\Rightarrow	box		JJ	\Rightarrow	fast
\bar{N}	\Rightarrow	NN	\bar{N}	NN	\Rightarrow	car		JJ	\Rightarrow	metal
\bar{N}	\Rightarrow	JJ	\bar{N}	NN	\Rightarrow	mechanic		JJ	\Rightarrow	idealistic
\bar{N}	\Rightarrow	\bar{N}	\bar{N}	NN	\Rightarrow	pigeon		JJ	\Rightarrow	clay
NP	\Rightarrow	DT	\bar{N}	NN				IN	\Rightarrow	in
PP	\Rightarrow	IN	NP	DT	\Rightarrow	the		IN	\Rightarrow	under
\bar{N}	\Rightarrow	\bar{N}	PP	DT	\Rightarrow	a		IN	\Rightarrow	of
								IN	\Rightarrow	on
								IN	\Rightarrow	with
								IN	\Rightarrow	as

Generates:

in a box, under the box, the fast car mechanic under the pigeon in
the box, ...

An Extended Grammar

\bar{N}	\Rightarrow	NN	
\bar{N}	\Rightarrow	NN	\bar{N}
\bar{N}	\Rightarrow	JJ	\bar{N}
\bar{N}	\Rightarrow	\bar{N}	\bar{N}
NP	\Rightarrow	DT	\bar{N}
PP	\Rightarrow	IN	NP
\bar{N}	\Rightarrow	\bar{N}	PP

Verbs, Verb Phrases, and Sentences

- ▶ Basic Verb Types

Vi = Intransitive verb e.g., sleeps, walks, laughs

Vt = Transitive verb e.g., sees, saw, likes

Vd = Ditransitive verb e.g., gave

- ▶ Basic VP Rules

VP → Vi

VP → Vt NP

VP → Vd NP NP

- ▶ Basic S Rule

S → NP VP

Examples of VP:

sleeps, walks, likes the mechanic, gave the mechanic the fast car

Examples of S:

the man sleeps, the dog walks, the dog gave the mechanic the fast car

PPs Modifying Verb Phrases

A new rule: VP → VP PP

New examples of VP:

sleeps in the car, walks like the mechanic, gave the mechanic the fast car on Tuesday, . . .

Complementizers, and SBARs

- ▶ Complementizers
COMP = complementizer e.g., that
- ▶ SBAR
SBAR → COMP S

Examples:

that the man sleeps, that the mechanic saw the dog ...

More Verbs

- ▶ New Verb Types

- V[5] e.g., said, reported

- V[6] e.g., told, informed

- V[7] e.g., bet

- ▶ New VP Rules

- VP → V[5] SBAR

- VP → V[6] NP SBAR

- VP → V[7] NP NP SBAR

Examples of New VPs:

said that the man sleeps

told the dog that the mechanic likes the pigeon

bet the pigeon \$50 that the mechanic owns a fast car

Coordination

- ▶ A New Part-of-Speech:
CC = Coordinator e.g., and, or, but

- ▶ New Rules

NP	→	NP	CC	NP
Ṅ	→	Ṅ	CC	Ṅ
VP	→	VP	CC	VP
S	→	S	CC	S
SBAR	→	SBAR	CC	SBAR

We've Only Scratched the Surface...

- ▶ Agreement

The dogs laugh *vs.* The dog laughs

- ▶ Wh-movement

The dog that the cat liked ...

- ▶ Active vs. passive

The dog saw the cat *vs.*

The cat was seen by the dog

- ▶ If you're interested in reading more:

Syntactic Theory: A Formal Introduction, 2nd Edition. Ivan A. Sag, Thomas Wasow, and Emily M. Bender.

Overview

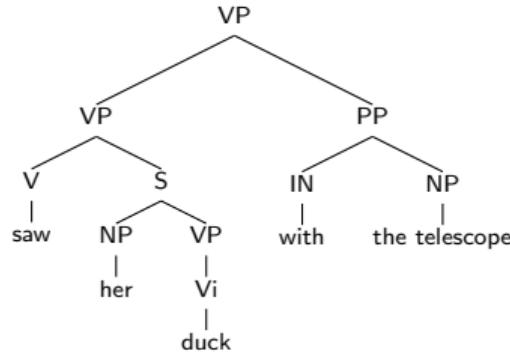
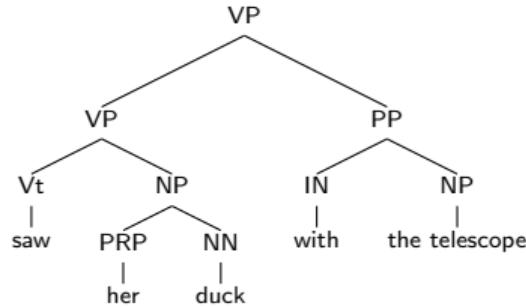
- ▶ An introduction to the parsing problem
- ▶ Context free grammars
- ▶ A brief(!) sketch of the syntax of English
- ▶ Examples of ambiguous structures

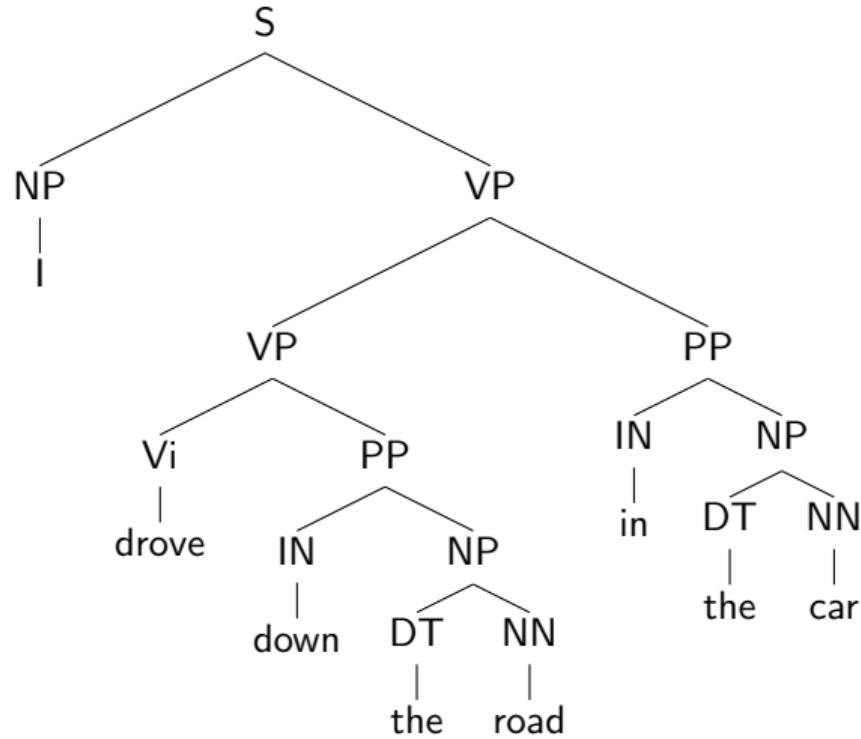
Sources of Ambiguity

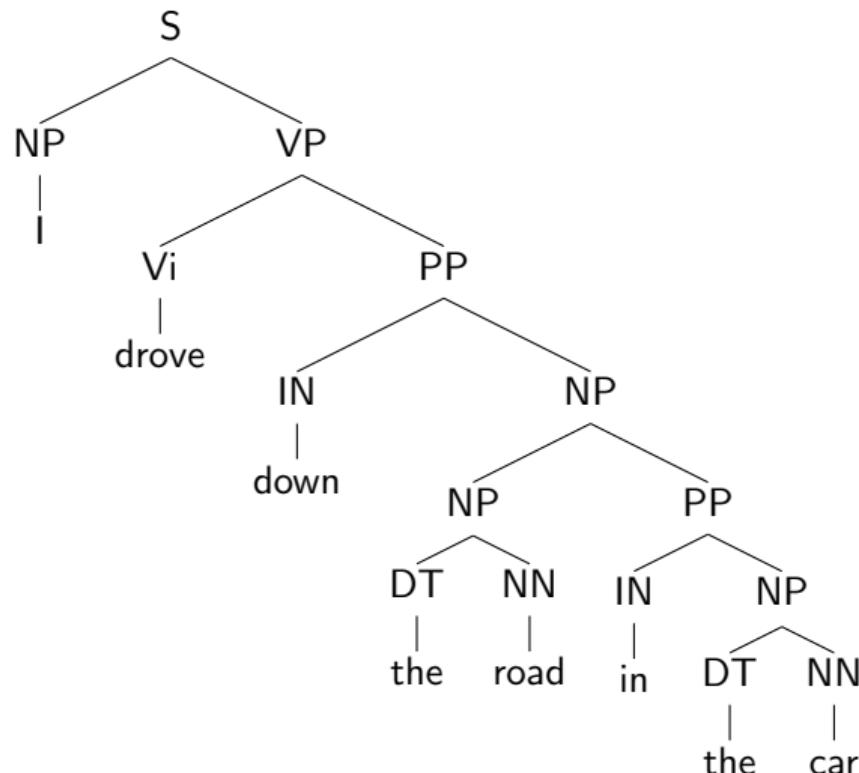
- ▶ Part-of-Speech ambiguity

NN → duck

Vi → duck



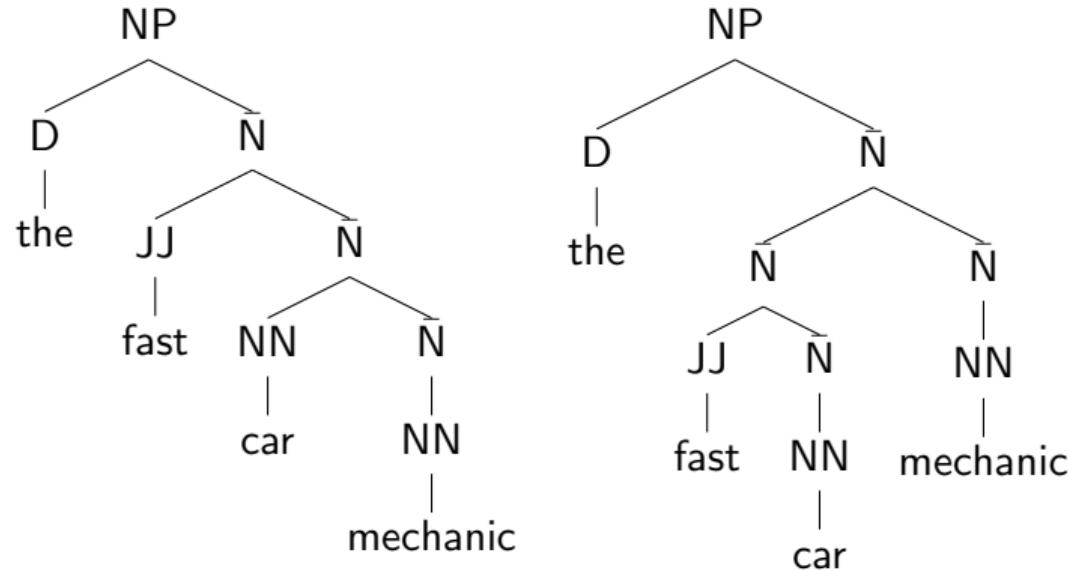




Two analyses for: John was believed to have been shot by Bill

Sources of Ambiguity: Noun Premodifiers

- ▶ Noun premodifiers:

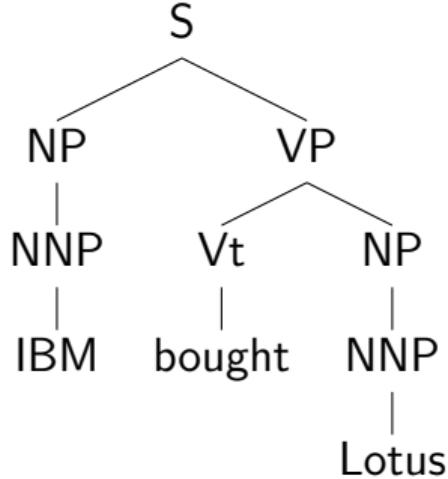


Weaknesses of Probabilistic Context-Free Grammars

Michael Collins, Columbia University

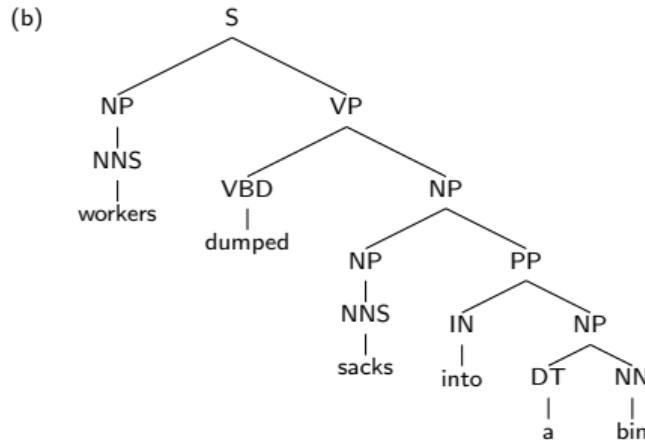
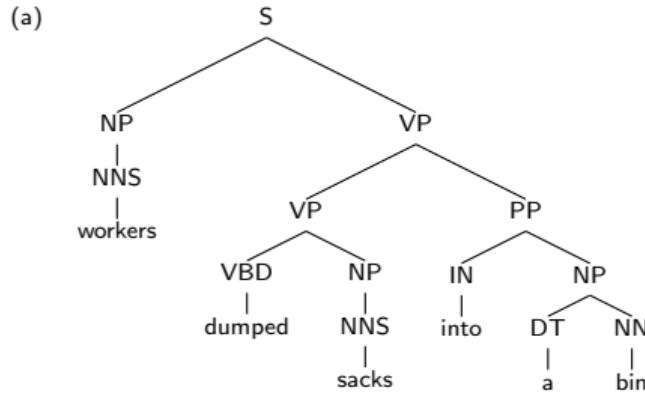
Weaknesses of PCFGs

- ▶ Lack of sensitivity to lexical information
- ▶ Lack of sensitivity to structural frequencies



$$\begin{aligned}
 p(t) = & q(S \rightarrow NP\ VP) & \times q(NNP \rightarrow IBM) \\
 & \times q(VP \rightarrow V\ NP) & \times q(Vt \rightarrow bought) \\
 & \times q(NP \rightarrow NNP) & \times q(NNP \rightarrow Lotus) \\
 & \times q(NP \rightarrow NNP)
 \end{aligned}$$

Another Case of PP Attachment Ambiguity



Rules
$S \rightarrow NP\ VP$
$NP \rightarrow NNS$
$VP \rightarrow VP\ PP$
$VP \rightarrow VBD\ NP$
$NP \rightarrow NNS$
$PP \rightarrow IN\ NP$
$NP \rightarrow DT\ NN$
$NNS \rightarrow workers$
$VBD \rightarrow dumped$
$NNS \rightarrow sacks$
$IN \rightarrow into$
$DT \rightarrow a$
$NN \rightarrow bin$

(a)

Rules
$S \rightarrow NP\ VP$
$NP \rightarrow NNS$
$NP \rightarrow NP\ PP$
$VP \rightarrow VBD\ NP$
$NP \rightarrow NNS$
$PP \rightarrow IN\ NP$
$NP \rightarrow DT\ NN$
$NNS \rightarrow workers$
$VBD \rightarrow dumped$
$NNS \rightarrow sacks$
$IN \rightarrow into$
$DT \rightarrow a$
$NN \rightarrow bin$

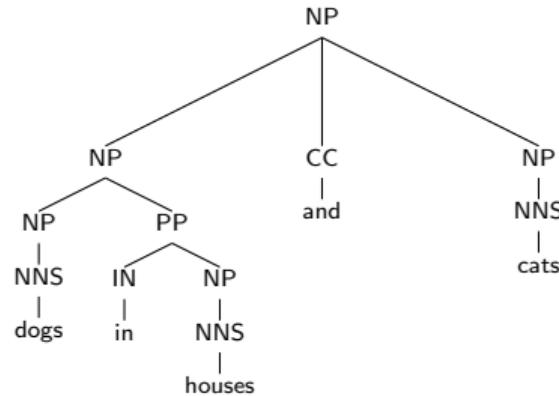
(b)

If $q(NP \rightarrow NP\ PP) > q(VP \rightarrow VP\ PP)$ then (b) is more probable, else (a) is more probable.

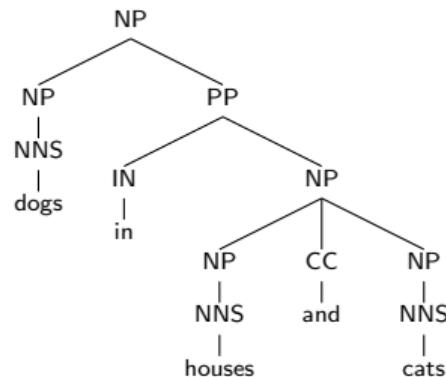
Attachment decision is completely independent of the words

A Case of Coordination Ambiguity

(a)



(b)



(a)

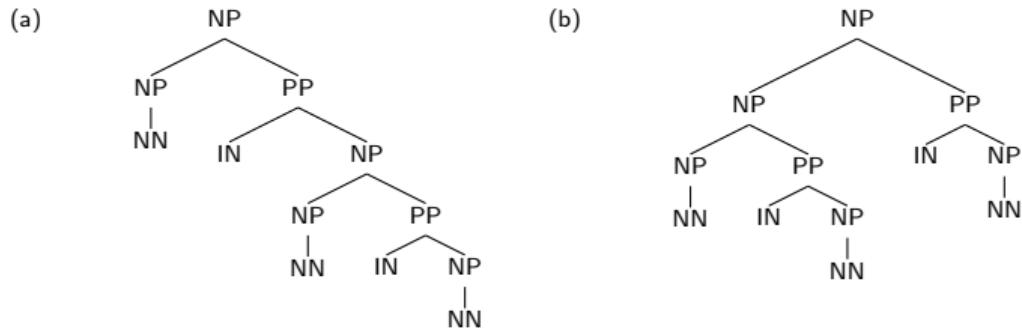
Rules
$NP \rightarrow NP\ CC\ NP$
$NP \rightarrow NP\ PP$
$NP \rightarrow NNS$
$PP \rightarrow IN\ NP$
$NP \rightarrow NNS$
$NP \rightarrow NNS$
$NNS \rightarrow \text{dogs}$
$IN \rightarrow \text{in}$
$NNS \rightarrow \text{houses}$
$CC \rightarrow \text{and}$
$NNS \rightarrow \text{cats}$

(b)

Rules
$NP \rightarrow NP\ CC\ NP$
$NP \rightarrow NP\ PP$
$NP \rightarrow NNS$
$PP \rightarrow IN\ NP$
$NP \rightarrow NNS$
$NP \rightarrow NNS$
$NNS \rightarrow \text{dogs}$
$IN \rightarrow \text{in}$
$NNS \rightarrow \text{houses}$
$CC \rightarrow \text{and}$
$NNS \rightarrow \text{cats}$

Here the two parses have identical rules, and therefore have identical probability under any assignment of PCFG rule probabilities

Structural Preferences: Close Attachment



- ▶ Example: [president of a company in Africa](#)
- ▶ Both parses have the same rules, therefore receive same probability under a PCFG
- ▶ “Close attachment” (structure (a)) is twice as likely in Wall Street Journal text.

Structural Preferences: Close Attachment

Previous example: John was believed to have been shot by Bill

Here the low attachment analysis (Bill does the *shooting*) contains same rules as the high attachment analysis (Bill does the *believing*), so the two analyses receive same probability.

Probabilistic Context-Free Grammars

Michael Collins, Columbia University

Overview

- ▶ Probabilistic Context-Free Grammars (PCFGs)
- ▶ The CKY Algorithm for parsing with PCFGs

A Probabilistic Context-Free Grammar (PCFG)

S	\Rightarrow	NP VP	1.0
VP	\Rightarrow	Vi	0.4
VP	\Rightarrow	Vt NP	0.4
VP	\Rightarrow	VP PP	0.2
NP	\Rightarrow	DT NN	0.3
NP	\Rightarrow	NP PP	0.7
PP	\Rightarrow	P NP	1.0

Vi	\Rightarrow	sleeps	1.0
Vt	\Rightarrow	saw	1.0
NN	\Rightarrow	man	0.7
NN	\Rightarrow	woman	0.2
NN	\Rightarrow	telescope	0.1
DT	\Rightarrow	the	1.0
IN	\Rightarrow	with	0.5
IN	\Rightarrow	in	0.5

- ▶ Probability of a tree t with rules

$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots, \alpha_n \rightarrow \beta_n$$

is $p(t) = \prod_{i=1}^n q(\alpha_i \rightarrow \beta_i)$ where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

DERIVATION

RULES USED

PROBABILITY

S

DERIVATION

S

NP VP

RULES USED

$S \rightarrow NP\ VP$

PROBABILITY

1.0

DERIVATION	RULES USED	PROBABILITY
S	$S \rightarrow NP\ VP$	1.0
NP VP	$NP \rightarrow DT\ NN$	0.3
DT NN VP		

DERIVATION	RULES USED	PROBABILITY
S	$S \rightarrow NP\ VP$	1.0
NP VP	$NP \rightarrow DT\ NN$	0.3
DT NN VP	$DT \rightarrow \text{the}$	1.0
the NN VP		

DERIVATION	RULES USED	PROBABILITY
S	$S \rightarrow NP\ VP$	1.0
NP VP	$NP \rightarrow DT\ NN$	0.3
DT NN VP	$DT \rightarrow \text{the}$	1.0
the NN VP	$NN \rightarrow \text{dog}$	0.1
the dog VP		

DERIVATION	RULES USED	PROBABILITY
S	$S \rightarrow NP\ VP$	1.0
NP VP	$NP \rightarrow DT\ NN$	0.3
DT NN VP	$DT \rightarrow \text{the}$	1.0
the NN VP	$NN \rightarrow \text{dog}$	0.1
the dog VP	$VP \rightarrow Vi$	0.4
the dog Vi		

DERIVATION	RULES USED	PROBABILITY
S	$S \rightarrow NP\ VP$	1.0
NP VP	$NP \rightarrow DT\ NN$	0.3
DT NN VP	$DT \rightarrow \text{the}$	1.0
the NN VP	$NN \rightarrow \text{dog}$	0.1
the dog VP	$VP \rightarrow Vi$	0.4
the dog Vi	$Vi \rightarrow \text{laughs}$	0.5
the dog laughs		

Properties of PCFGs

- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG

Properties of PCFGs

- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- ▶ Say we have a sentence s , set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability.*

Properties of PCFGs

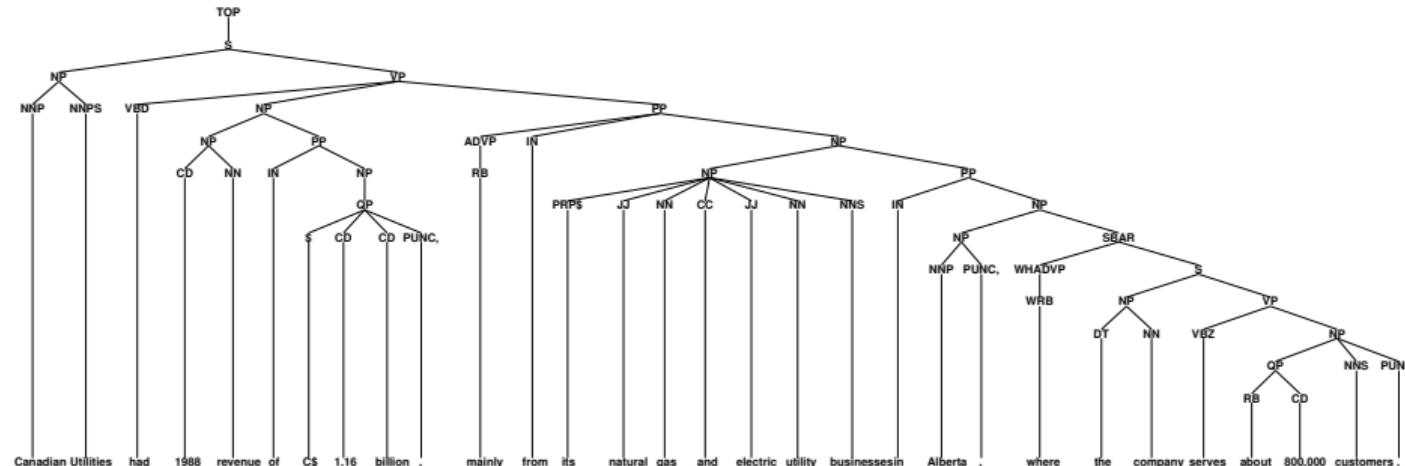
- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- ▶ Say we have a sentence s , set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability.*
- ▶ The most likely parse tree for a sentence s is

$$\arg \max_{t \in \mathcal{T}(s)} p(t)$$

Data for Parsing Experiments: Treebanks

- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



Deriving a PCFG from a Treebank

- ▶ Given a set of example trees (a treebank), the underlying CFG can simply be **all rules seen in the corpus**
- ▶ Maximum Likelihood estimates:

$$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

where the counts are taken from a training set of example trees.

- ▶ **If the training data is generated by a PCFG**, then as the training data size goes to infinity, the maximum-likelihood PCFG will converge to the same distribution as the “true” PCFG.

PCFGs

Booth and Thompson (1973) showed that a CFG with rule probabilities correctly defines a distribution over the set of derivations provided that:

1. The rule probabilities define conditional distributions over the different ways of rewriting each non-terminal.
2. A technical condition on the rule probabilities ensuring that the probability of the derivation terminating in a finite number of steps is 1. (This condition is not really a practical concern.)

Parsing with a PCFG

- ▶ Given a PCFG and a sentence s , define $\mathcal{T}(s)$ to be the set of trees with s as the yield.
- ▶ Given a PCFG and a sentence s , how do we find

$$\arg \max_{t \in \mathcal{T}(s)} p(t)$$

Chomsky Normal Form

A context free grammar $G = (N, \Sigma, R, S)$ in Chomsky Normal Form is as follows

- ▶ N is a set of non-terminal symbols
- ▶ Σ is a set of terminal symbols
- ▶ R is a set of rules which take one of two forms:
 - ▶ $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
 - ▶ $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- ▶ $S \in N$ is a distinguished start symbol

A Dynamic Programming Algorithm

- ▶ Given a PCFG and a sentence s , how do we find

$$\max_{t \in \mathcal{T}(s)} p(t)$$

- ▶ Notation:

n = number of words in the sentence

w_i = i 'th word in the sentence

N = the set of non-terminals in the grammar

S = the start symbol in the grammar

- ▶ Define a dynamic programming table

$\pi[i, j, X] =$ maximum probability of a constituent with non-terminal X
spanning words $i \dots j$ inclusive

- ▶ Our goal is to calculate $\max_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$

An Example

the dog saw the man with the telescope

A Dynamic Programming Algorithm

- ▶ Base case definition: for all $i = 1 \dots n$, for $X \in N$

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

(note: define $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in the grammar)

- ▶ Recursive definition: for all $i = 1 \dots n$, $j = (i + 1) \dots n$, $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

An Example

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

the dog saw the man with the telescope

The Full Dynamic Programming Algorithm

Input: a sentence $s = x_1 \dots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.

Initialization:

For all $i \in \{1 \dots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- ▶ For $l = 1 \dots (n - 1)$
 - ▶ For $i = 1 \dots (n - l)$
 - ▶ Set $j = i + l$
 - ▶ For all $X \in N$, calculate

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

and

$$bp(i, j, X) = \arg \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

A Dynamic Programming Algorithm for the Sum

- Given a PCFG and a sentence s , how do we find

$$\sum_{t \in \mathcal{T}(s)} p(t)$$

- Notation:

n = number of words in the sentence

w_i = i 'th word in the sentence

N = the set of non-terminals in the grammar

S = the start symbol in the grammar

- Define a dynamic programming table

$\pi[i, j, X] =$ sum of probabilities for constituent with non-terminal X
spanning words $i \dots j$ inclusive

- Our goal is to calculate $\sum_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$

Summary

- ▶ PCFGs augments CFGs by including a probability for each rule in the grammar.
- ▶ The probability for a parse tree is the product of probabilities for the rules in the tree
- ▶ To build a PCFG-parsed parser:
 1. Learn a PCFG from a treebank
 2. Given a test data sentence, use the CKY algorithm to compute the highest probability tree for the sentence under the PCFG

Lexicalized Probabilistic Context-Free Grammars

Michael Collins, Columbia University

Overview

- ▶ Lexicalization of a treebank
- ▶ Lexicalized probabilistic context-free grammars
- ▶ Parameter estimation in lexicalized probabilistic context-free grammars
- ▶ Accuracy of lexicalized probabilistic context-free grammars

Heads in Context-Free Rules

Add annotations specifying the “head” of each rule:

S	\Rightarrow	NP	VP
VP	\Rightarrow	Vi	
VP	\Rightarrow	Vt	NP
VP	\Rightarrow	VP	PP
NP	\Rightarrow	DT	NN
NP	\Rightarrow	NP	PP
PP	\Rightarrow	IN	NP

Vi	\Rightarrow	sleeps
Vt	\Rightarrow	saw
NN	\Rightarrow	man
NN	\Rightarrow	woman
NN	\Rightarrow	telescope
DT	\Rightarrow	the
IN	\Rightarrow	with
IN	\Rightarrow	in

More about Heads

- ▶ Each context-free rule has one “special” child that is the head of the rule. e.g.,

S \Rightarrow NP VP (VP is the head)

VP \Rightarrow Vt NP (Vt is the head)

NP \Rightarrow DT NN NN (NN is the head)

- ▶ A core idea in syntax
(e.g., see X-bar Theory, Head-Driven Phrase Structure Grammar)

- ▶ Some intuitions:
 - ▶ The central sub-constituent of each rule.
 - ▶ The semantic predicate in each rule.

Rules which Recover Heads: An Example for NPs

If the rule contains NN, NNS, or NNP:

Choose the rightmost NN, NNS, or NNP

Else If the rule contains an NP: Choose the leftmost NP

Else If the rule contains a JJ: Choose the rightmost JJ

Else If the rule contains a CD: Choose the rightmost CD

Else Choose the rightmost child

e.g.,

NP	\Rightarrow	DT	NNP	NN
NP	\Rightarrow	DT	NN	NNP
NP	\Rightarrow	NP	PP	
NP	\Rightarrow	DT	JJ	
NP	\Rightarrow	DT		

Rules which Recover Heads: An Example for VPs

If the rule contains V_i or V_t : Choose the leftmost V_i or V_t

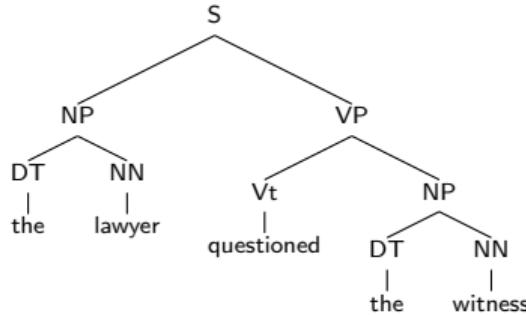
Else If the rule contains an VP: Choose the leftmost VP

Else Choose the leftmost child

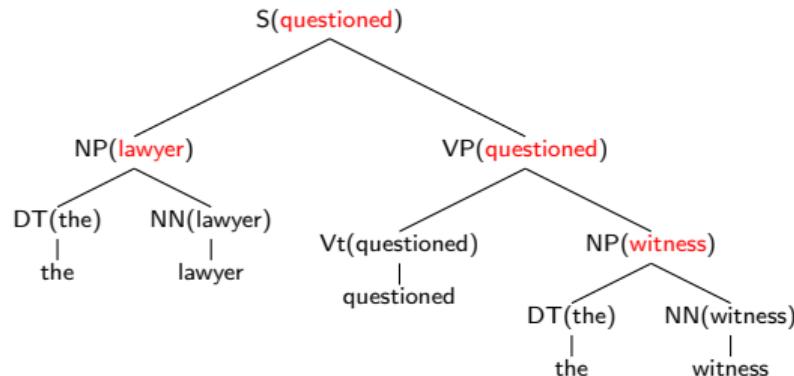
e.g.,

$$\begin{array}{lll} \text{VP} & \Rightarrow & \text{Vt} \quad \text{NP} \\ \text{VP} & \Rightarrow & \text{VP} \quad \text{PP} \end{array}$$

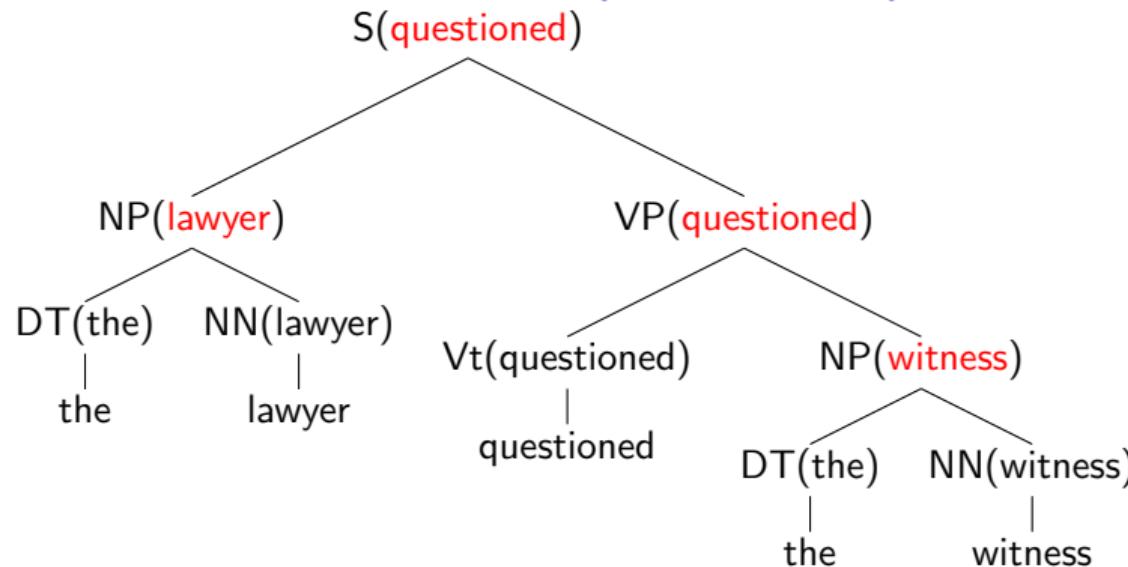
Adding Headwords to Trees



↓



Adding Headwords to Trees (Continued)



- ▶ A constituent receives its **headword** from its **head child**.

S \Rightarrow NP VP

(S receives headword from VP)

VP \Rightarrow Vt NP

(VP receives headword from Vt)

NP \Rightarrow DT NN

(NP receives headword from NN)

Overview

- ▶ Lexicalization of a treebank
- ▶ Lexicalized probabilistic context-free grammars
- ▶ Parameter estimation in lexicalized probabilistic context-free grammars
- ▶ Accuracy of lexicalized probabilistic context-free grammars

Chomsky Normal Form

A context free grammar $G = (N, \Sigma, R, S)$ in Chomsky Normal Form is as follows

- ▶ N is a set of non-terminal symbols
- ▶ Σ is a set of terminal symbols
- ▶ R is a set of rules which take one of two forms:
 - ▶ $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
 - ▶ $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- ▶ $S \in N$ is a distinguished start symbol

We can find the highest scoring parse under a PCFG in this form, in $O(n^3|N|^3)$ time where n is the length of the string being parsed.

Lexicalized Context-Free Grammars in Chomsky Normal Form

- ▶ N is a set of non-terminal symbols
- ▶ Σ is a set of terminal symbols
- ▶ R is a set of rules which take one of three forms:
 - ▶ $X(h) \rightarrow_1 Y_1(h) Y_2(w)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
 - ▶ $X(h) \rightarrow_2 Y_1(w) Y_2(h)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
 - ▶ $X(h) \rightarrow h$ for $X \in N$, and $h \in \Sigma$
- ▶ $S \in N$ is a distinguished start symbol

An Example

$S(saw)$	\rightarrow_2	$NP(man)$	$VP(saw)$
$VP(saw)$	\rightarrow_1	$Vt(saw)$	$NP(dog)$
$NP(man)$	\rightarrow_2	$DT(the)$	$NN(man)$
$NP(dog)$	\rightarrow_2	$DT(the)$	$NN(dog)$
$Vt(saw)$	\rightarrow	saw	
$DT(the)$	\rightarrow	the	
$NN(man)$	\rightarrow	man	
$NN(dog)$	\rightarrow	dog	

Parameters in a Lexicalized PCFG

- ▶ An example parameter in a PCFG:

$$q(S \rightarrow NP\ VP)$$

- ▶ An example parameter in a Lexicalized PCFG:

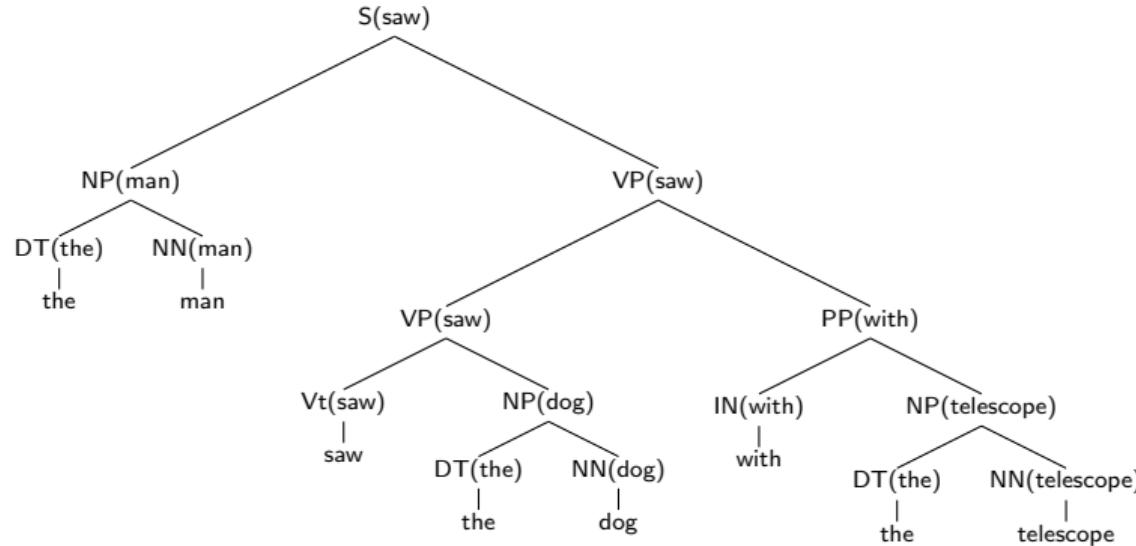
$$q(S(saw) \rightarrow_2 NP(man)\ VP(saw))$$

Parsing with Lexicalized CFGs

- ▶ The new form of grammar looks just like a Chomsky normal form CFG, but with potentially $O(|\Sigma|^2 \times |N|^3)$ possible rules.
- ▶ Naively, parsing an n word sentence using the dynamic programming algorithm will take $O(n^3|\Sigma|^2|N|^3)$ time. **But $|\Sigma|$ can be huge!!**
- ▶ Crucial observation: at most $O(n^2 \times |N|^3)$ rules can be applicable to a given sentence w_1, w_2, \dots, w_n of length n .
This is because any rules which contain a lexical item that is not one of $w_1 \dots w_n$, can be safely discarded.
- ▶ The result: we can parse in $O(n^5|N|^3)$ time.

Overview

- ▶ Lexicalization of a treebank
- ▶ Lexicalized probabilistic context-free grammars
- ▶ Parameter estimation in lexicalized probabilistic context-free grammars
- ▶ Accuracy of lexicalized probabilistic context-free grammars



$$\begin{aligned}
 p(t) = & q(S(\text{saw}) \rightarrow_2 \text{NP}(\text{man}) \text{ VP}(\text{saw})) \\
 & \times q(\text{NP}(\text{man}) \rightarrow_2 \text{DT}(\text{the}) \text{ NN}(\text{man})) \\
 & \times q(\text{VP}(\text{saw}) \rightarrow_1 \text{VP}(\text{saw}) \text{ PP}(\text{with})) \\
 & \times q(\text{VP}(\text{saw}) \rightarrow_1 \text{Vt}(\text{saw}) \text{ NP}(\text{dog})) \\
 & \times q(\text{PP}(\text{with}) \rightarrow_1 \text{IN}(\text{with}) \text{ NP}(\text{telescope})) \\
 & \times \dots
 \end{aligned}$$

A Model from Charniak (1997)

- ▶ An example parameter in a Lexicalized PCFG:

$$q(S(\text{saw}) \rightarrow_2 NP(\text{man}) VP(\text{saw}))$$

- ▶ First step: decompose this parameter into a product of two parameters

$$\begin{aligned} & q(S(\text{saw}) \rightarrow_2 NP(\text{man}) VP(\text{saw})) \\ = & q(S \rightarrow_2 NP VP | S, \text{saw}) \times q(\text{man} | S \rightarrow_2 NP VP, \text{saw}) \end{aligned}$$

A Model from Charniak (1997) (Continued)

$$\begin{aligned} & q(S(\text{saw}) \rightarrow_2 NP(\text{man}) VP(\text{saw})) \\ = & q(S \rightarrow_2 NP VP | S, \text{saw}) \times q(\text{man} | S \rightarrow_2 NP VP, \text{saw}) \end{aligned}$$

- ▶ Second step: use smoothed estimation for the two parameter estimates

$$\begin{aligned} & q(S \rightarrow_2 NP VP | S, \text{saw}) \\ = & \lambda_1 \times q_{ML}(S \rightarrow_2 NP VP | S, \text{saw}) + \lambda_2 \times q_{ML}(S \rightarrow_2 NP VP | S) \end{aligned}$$

A Model from Charniak (1997) (Continued)

$$\begin{aligned} & q(S(\text{saw}) \rightarrow_2 NP(\text{man}) VP(\text{saw})) \\ = & q(S \rightarrow_2 NP VP | S, \text{saw}) \times q(\text{man} | S \rightarrow_2 NP VP, \text{saw}) \end{aligned}$$

- ▶ Second step: use smoothed estimation for the two parameter estimates

$$\begin{aligned} & q(S \rightarrow_2 NP VP | S, \text{saw}) \\ = & \lambda_1 \times q_{ML}(S \rightarrow_2 NP VP | S, \text{saw}) + \lambda_2 \times q_{ML}(S \rightarrow_2 NP VP | S) \\ \\ & q(\text{man} | S \rightarrow_2 NP VP, \text{saw}) \\ = & \lambda_3 \times q_{ML}(\text{man} | S \rightarrow_2 NP VP, \text{saw}) + \lambda_4 \times q_{ML}(\text{man} | S \rightarrow_2 NP VP) \\ & + \lambda_5 \times q_{ML}(\text{man} | NP) \end{aligned}$$

Other Important Details

- ▶ Need to deal with rules with more than two children, e.g.,
 $\text{VP}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP}(\text{him}) \text{ PP}(\text{on}) \text{ SBAR}(\text{that})$

Other Important Details

- ▶ Need to deal with rules with more than two children, e.g.,
 $\text{VP}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP}(\text{him}) \text{ PP}(\text{on}) \text{ SBAR}(\text{that})$
- ▶ Need to incorporate parts of speech (useful in smoothing)
 $\text{VP-V}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP-PRP}(\text{him}) \text{ PP-IN}(\text{on}) \text{ SBAR-COMP}(\text{that})$

Other Important Details

- ▶ Need to deal with rules with more than two children, e.g.,
 $\text{VP}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP}(\text{him}) \text{ PP}(\text{on}) \text{ SBAR}(\text{that})$
- ▶ Need to incorporate parts of speech (useful in smoothing)
 $\text{VP-V}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP-PRP}(\text{him}) \text{ PP-IN}(\text{on}) \text{ SBAR-COMP}(\text{that})$
- ▶ Need to encode preferences for close attachment
John was believed to have been shot by Bill

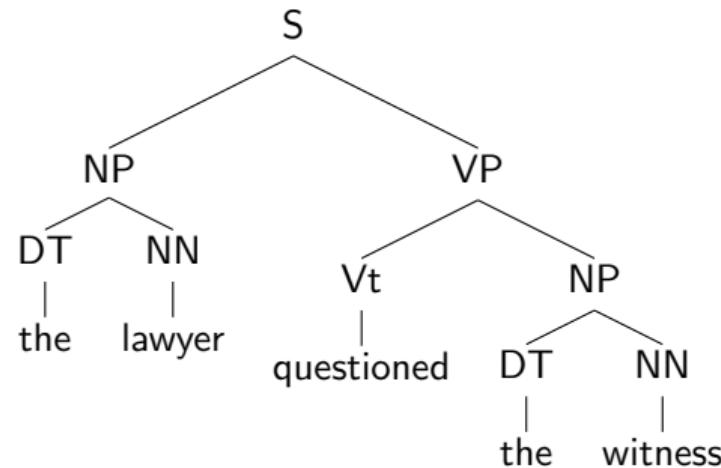
Other Important Details

- ▶ Need to deal with rules with more than two children, e.g.,
 $\text{VP}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP}(\text{him}) \text{ PP}(\text{on}) \text{ SBAR}(\text{that})$
- ▶ Need to incorporate parts of speech (useful in smoothing)
 $\text{VP-V}(\text{told}) \rightarrow \text{V}(\text{told}) \text{ NP-PRP}(\text{him}) \text{ PP-IN}(\text{on}) \text{ SBAR-COMP}(\text{that})$
- ▶ Need to encode preferences for close attachment
John was believed to have been shot by Bill
- ▶ Further reading:
Michael Collins. 2003. Head-Driven Statistical Models for Natural Language Parsing. In Computational Linguistics.

Overview

- ▶ Lexicalization of a treebank
- ▶ Lexicalized probabilistic context-free grammars
- ▶ Parameter estimation in lexicalized probabilistic context-free grammars
- ▶ Accuracy of lexicalized probabilistic context-free grammars

Evaluation: Representing Trees as Constituents



Label	Start Point	End Point
NP	1	2
NP	4	5
VP	3	5
S	1	5

Precision and Recall

Label	Start Point	End Point
NP	1	2
NP	4	5
NP	4	8
PP	6	8
NP	7	8
VP	3	8
S	1	8

Label	Start Point	End Point
NP	1	2
NP	4	5
PP	6	8
NP	7	8
VP	3	8
S	1	8

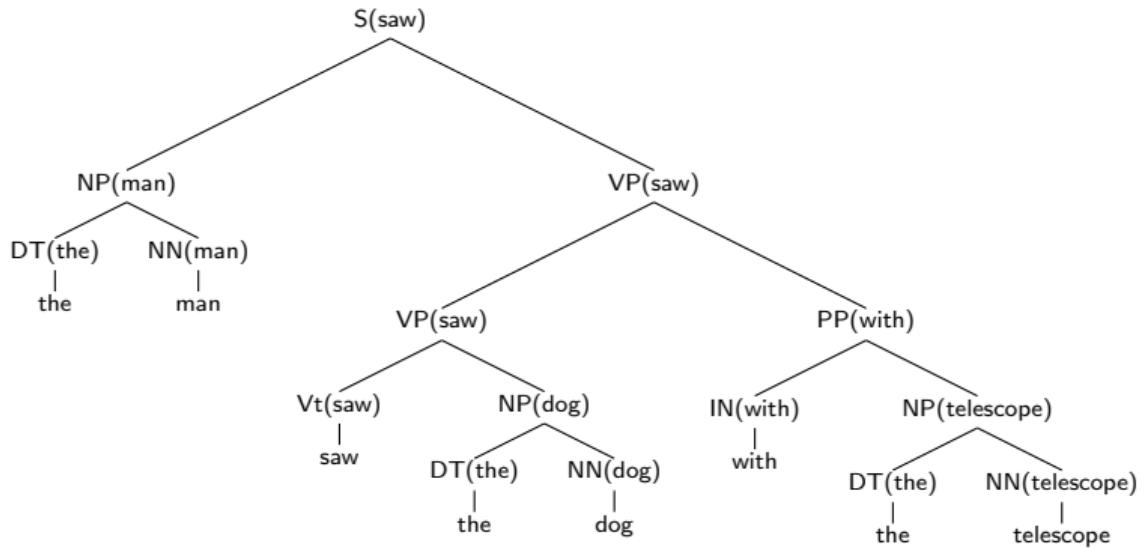
- ▶ $G = \text{number of constituents in gold standard} = 7$
- ▶ $P = \text{number in parse output} = 6$
- ▶ $C = \text{number correct} = 6$

$$\text{Recall} = 100\% \times \frac{C}{G} = 100\% \times \frac{6}{7}$$

$$\text{Precision} = 100\% \times \frac{C}{P} = 100\% \times \frac{6}{6}$$

Results

- ▶ Training data: 40,000 sentences from the Penn Wall Street Journal treebank. Testing: around 2,400 sentences from the Penn Wall Street Journal treebank.
- ▶ Results for a PCFG: 70.6% Recall, 74.8% Precision
- ▶ Magerman (1994): 84.0% Recall, 84.3% Precision
- ▶ Results for a lexicalized PCFG: 88.1% recall, 88.3% precision (from Collins (1997, 2003))
- ▶ More recent results: 90.7% Recall/91.4% Precision (Carreras et al., 2008); 91.7% Recall, 92.0% Precision (Petrov 2010); 91.2% Recall, 91.8% Precision (Charniak and Johnson, 2005)



$\langle \text{ROOT}_0,$	$\text{saw}_3,$	$\text{ROOT} \rangle$
$\langle \text{saw}_3,$	$\text{man}_2,$	$S \rightarrow_2 \text{NP VP} \rangle$
$\langle \text{man}_2,$	$\text{the}_1,$	$\text{NP} \rightarrow_2 \text{DT NN} \rangle$
$\langle \text{saw}_3,$	$\text{with}_6,$	$\text{VP} \rightarrow_1 \text{VP PP} \rangle$
$\langle \text{saw}_3,$	$\text{dog}_5,$	$\text{VP} \rightarrow_1 \text{Vt NP} \rangle$
$\langle \text{dog}_5,$	$\text{the}_4,$	$\text{NP} \rightarrow_2 \text{DT NN} \rangle$
$\langle \text{with}_6,$	$\text{telescope}_8,$	$\text{PP} \rightarrow_1 \text{IN NP} \rangle$
$\langle \text{telescope}_8,$	$\text{the}_7,$	$\text{NP} \rightarrow_2 \text{DT NN} \rangle$

Dependency Accuracies

- ▶ All parses for a sentence with n words have n dependencies
Report a single figure, dependency accuracy
- ▶ Results from Collins, 2003: 88.3% dependency accuracy
- ▶ Can calculate precision/recall on particular dependency **types**
e.g., look at all subject/verb dependencies ⇒
all dependencies with label S →₂ NP VP

Recall =

$$\frac{\text{number of subject/verb dependencies correct}}{\text{number of subject/verb dependencies in gold standard}}$$

Precision =

$$\frac{\text{number of subject/verb dependencies correct}}{\text{number of subject/verb dependencies in parser's output}}$$

Strengths and Weaknesses of Modern Parsers

(Numbers taken from Collins (2003))

- ▶ Subject-verb pairs: over 95% recall and precision
- ▶ Object-verb pairs: over 92% recall and precision
- ▶ Other arguments to verbs: \approx 93% recall and precision
- ▶ Non-recursive NP boundaries: \approx 93% recall and precision
- ▶ PP attachments: \approx 82% recall and precision
- ▶ Coordination ambiguities: \approx 61% recall and precision

Summary

- ▶ Key weakness of PCFGs: lack of sensitivity to lexical information
- ▶ Lexicalized PCFGs:
 - ▶ Lexicalize a treebank using head rules
 - ▶ Estimate the parameters of a lexicalized PCFG using smoothed estimation
- ▶ Accuracy of lexicalized PCFGs: around 88% in recovering constituents or dependencies

Tagging Problems, and Hidden Markov Models

Michael Collins, Columbia University

Overview

- ▶ The Tagging Problem
- ▶ Generative models, and the noisy-channel model, for supervised learning
- ▶ Hidden Markov Model (HMM) taggers
 - ▶ Basic definitions
 - ▶ Parameter estimation
 - ▶ The Viterbi algorithm

Part-of-Speech Tagging

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street,
as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V
forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N
Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

N = Noun

V = Verb

P = Preposition

Adv = Adverb

Adj = Adjective

...

Named Entity Recognition

INPUT: Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT: Profits soared at [Company Boeing Co.], easily topping forecasts on [Location Wall Street], as their CEO [Person Alan Mulally] announced first quarter results.

Named Entity Extraction as Tagging

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street,
as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA
topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA
their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA
quarter/NA results/NA ./NA

NA = No entity

SC = Start Company

CC = Continue Company

SL = Start Location

CL = Continue Location

...

Our Goal

Training set:

- 1 Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./.
 - 2 Mr./NNP Vinken/NNP is/VBZ chairman/NN of/IN Elsevier/NNP N.V./NNP ,/, the/DT Dutch/NNP publishing/VBG group/NN ./.
 - 3 Rudolph/NNP Agnew/NNP ,/, 55/CD years/NNS old/JJ and/CC chairman/NN of/IN Consolidated/NNP Gold/NNP Fields/NNP PLC/NNP ,/, was/VBD named/VBN a/DT nonexecutive/JJ director/NN of/IN this/DT British/JJ industrial/JJ conglomerate/NN ./.
- ...
- 38,219** It/PRP is/VBZ also/RB pulling/VBG 20/CD people/NNS out/IN of/IN Puerto/NNP Rico/NNP ,/, who/WP were/VBD helping/VBG Hurricane/NNP Hugo/NNP victims/NNS ,/, and/CC sending/VBG them/PRP to/TO San/NNP Francisco/NNP instead/RB ./.

- ▶ From the training set, induce a function/algorithm that maps new sentences to their tag sequences.

Two Types of Constraints

Influential/JJ members/NNS of/IN the/DT House/NNP Ways/NNP and/CC
Means/NNP Committee/NNP introduced/VBD legislation/NN that/WDT
would/MD restrict/VB how/WRB the/DT new/JJ savings-and-loan/NN
bailout/NN agency/NN can/MD raise/VB capital/NN ./.

- ▶ “Local”: e.g., *can* is more likely to be a modal verb **MD** rather than a noun **NN**
- ▶ “Contextual”: e.g., a noun is much more likely than a verb to follow a determiner
- ▶ Sometimes these preferences are in conflict:
The trash can is in the garage

Overview

- ▶ The Tagging Problem
- ▶ Generative models, and the noisy-channel model, for supervised learning
- ▶ Hidden Markov Model (HMM) taggers
 - ▶ Basic definitions
 - ▶ Parameter estimation
 - ▶ The Viterbi algorithm

Supervised Learning Problems

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Each $x^{(i)}$ is an input, each $y^{(i)}$ is a label.
- ▶ Task is to learn a function f mapping inputs x to labels $f(x)$

Supervised Learning Problems

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Each $x^{(i)}$ is an input, each $y^{(i)}$ is a label.
- ▶ Task is to learn a function f mapping inputs x to labels $f(x)$
- ▶ Conditional models:
 - ▶ Learn a distribution $p(y|x)$ from training examples
 - ▶ For any test input x , define $f(x) = \arg \max_y p(y|x)$

Generative Models

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.

Generative Models

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.
- ▶ Generative models:
 - ▶ Learn a distribution $p(x, y)$ from training examples
 - ▶ Often we have $p(x, y) = p(y)p(x|y)$

Generative Models

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.
- ▶ Generative models:
 - ▶ Learn a distribution $p(x, y)$ from training examples
 - ▶ Often we have $p(x, y) = p(y)p(x|y)$
- ▶ Note: we then have

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

where $p(x) = \sum_y p(y)p(x|y)$

Decoding with Generative Models

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.

Decoding with Generative Models

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.
- ▶ Generative models:
 - ▶ Learn a distribution $p(x, y)$ from training examples
 - ▶ Often we have $p(x, y) = p(y)p(x|y)$

Decoding with Generative Models

- ▶ We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.
- ▶ Generative models:
 - ▶ Learn a distribution $p(x, y)$ from training examples
 - ▶ Often we have $p(x, y) = p(y)p(x|y)$
- ▶ Output from the model:

$$\begin{aligned}f(x) &= \arg \max_y p(y|x) \\&= \arg \max_y \frac{p(y)p(x|y)}{p(x)} \\&= \arg \max_y p(y)p(x|y)\end{aligned}$$

Overview

- ▶ The Tagging Problem
- ▶ Generative models, and the noisy-channel model, for supervised learning
- ▶ Hidden Markov Model (HMM) taggers
 - ▶ Basic definitions
 - ▶ Parameter estimation
 - ▶ The Viterbi algorithm

Hidden Markov Models

- ▶ We have an input sentence $x = x_1, x_2, \dots, x_n$
(x_i is the i 'th word in the sentence)
- ▶ We have a tag sequence $y = y_1, y_2, \dots, y_n$
(y_i is the i 'th tag in the sentence)
- ▶ We'll use an HMM to define

$$p(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n)$$

for any sentence $x_1 \dots x_n$ and tag sequence $y_1 \dots y_n$ of the same length.

- ▶ Then the most likely tag sequence for x is

$$\arg \max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1, y_2, \dots, y_n)$$

Trigram Hidden Markov Models (Trigram HMMs)

For any sentence $x_1 \dots x_n$ where $x_i \in \mathcal{V}$ for $i = 1 \dots n$, and any tag sequence $y_1 \dots y_{n+1}$ where $y_i \in \mathcal{S}$ for $i = 1 \dots n$, and $y_{n+1} = \text{STOP}$, the joint probability of the sentence and tag sequence is

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^n e(x_i | y_i)$$

where we have assumed that $x_0 = x_{-1} = *$.

Parameters of the model:

- ▶ $q(s|u, v)$ for any $s \in \mathcal{S} \cup \{\text{STOP}\}$, $u, v \in \mathcal{S} \cup \{*\}$
- ▶ $e(x|s)$ for any $s \in \mathcal{S}$, $x \in \mathcal{V}$

An Example

If we have $n = 3$, $x_1 \dots x_3$ equal to the sentence *the dog laughs*, and $y_1 \dots y_4$ equal to the tag sequence D N V STOP, then

$$\begin{aligned} & p(x_1 \dots x_n, y_1 \dots y_{n+1}) \\ &= q(D|*, *) \times q(N|*, D) \times q(V|D, N) \times q(STOP|N, V) \\ & \quad \times e(the|D) \times e(dog|N) \times e(laughs|V) \end{aligned}$$

- ▶ STOP is a special tag that terminates the sequence
- ▶ We take $y_0 = y_{-1} = *$, where * is a special “padding” symbol

Why the Name?

$$p(x_1 \dots x_n, y_1 \dots y_n) = \underbrace{q(\text{STOP} | y_{n-1}, y_n) \prod_{j=1}^n q(y_j | y_{j-2}, y_{j-1})}_{\text{Markov Chain}} \times \underbrace{\prod_{j=1}^n e(x_j | y_j)}_{x_j \text{'s are observed}}$$

Overview

- ▶ The Tagging Problem
- ▶ Generative models, and the noisy-channel model, for supervised learning
- ▶ Hidden Markov Model (HMM) taggers
 - ▶ Basic definitions
 - ▶ Parameter estimation
 - ▶ The Viterbi algorithm

Smoothed Estimation

$$\begin{aligned} q(Vt \mid DT, JJ) &= \lambda_1 \times \frac{\text{Count}(Dt, JJ, Vt)}{\text{Count}(Dt, JJ)} \\ &\quad + \lambda_2 \times \frac{\text{Count}(JJ, Vt)}{\text{Count}(JJ)} \\ &\quad + \lambda_3 \times \frac{\text{Count}(Vt)}{\text{Count}()} \end{aligned}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1, \quad \text{and for all } i, \lambda_i \geq 0$$

$$e(\text{base} \mid Vt) = \frac{\text{Count}(Vt, \text{base})}{\text{Count}(Vt)}$$

Dealing with Low-Frequency Words: An Example

Profits soared at Boeing Co. , easily topping forecasts on Wall Street , as their CEO Alan Mulally announced first quarter results .

Dealing with Low-Frequency Words

A common method is as follows:

- ▶ **Step 1:** Split vocabulary into two sets

Frequent words = words occurring ≥ 5 times in training

Low frequency words = all other words

- ▶ **Step 2:** Map low frequency words into a small, finite set, depending on prefixes, suffixes etc.

Dealing with Low-Frequency Words: An Example

[Bikel et. al 1999] (**named-entity recognition**)

Word class	Example	Intuition
twoDigitNum	90	Two digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount, percentage
othernum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	first word of sentence	no useful capitalization information
initCap	Sally	Capitalized word
lowercase	can	Uncapitalized word
other	,	Punctuation marks, all other words

Dealing with Low-Frequency Words: An Example

Profits/NA soared/NA at/NA Boeing/SC Co./CC,/NA easily/NA
topping/NA forecasts/NA on/NA Wall/SL Street/CL,/NA as/NA their/NA
CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA
results/NA ./NA



firstword/NA soared/NA at/NA initCap/SC Co./CC,/NA easily/NA
lowercase/NA forecasts/NA on/NA initCap/SL Street/CL,/NA as/NA
their/NA CEO/NA Alan/SP initCap/CP announced/NA first/NA
quarter/NA results/NA ./NA

NA = No entity

SC = Start Company

CC = Continue Company

SL = Start Location

CL = Continue Location

...

Overview

- ▶ The Tagging Problem
- ▶ Generative models, and the noisy-channel model, for supervised learning
- ▶ Hidden Markov Model (HMM) taggers
 - ▶ Basic definitions
 - ▶ Parameter estimation
 - ▶ The Viterbi algorithm

The Viterbi Algorithm

Problem: for an input $x_1 \dots x_n$, find

$$\arg \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

where the arg max is taken over all sequences $y_1 \dots y_{n+1}$ such that $y_i \in \mathcal{S}$ for $i = 1 \dots n$, and $y_{n+1} = \text{STOP}$.

We assume that p again takes the form

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^n e(x_i | y_i)$$

Recall that we have assumed in this definition that $y_0 = y_{-1} = *$, and $y_{n+1} = \text{STOP}$.

Brute Force Search is Hopelessly Inefficient

Problem: for an input $x_1 \dots x_n$, find

$$\arg \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

where the arg max is taken over all sequences $y_1 \dots y_{n+1}$ such that $y_i \in \mathcal{S}$ for $i = 1 \dots n$, and $y_{n+1} = \text{STOP}$.

The Viterbi Algorithm

- ▶ Define n to be the length of the sentence
- ▶ Define S_k for $k = -1 \dots n$ to be the set of possible tags at position k :

$$S_{-1} = S_0 = \{*\}$$

$$S_k = S \quad \text{for } k \in \{1 \dots n\}$$

- ▶ Define

$$r(y_{-1}, y_0, y_1, \dots, y_k) = \prod_{i=1}^k q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^k e(x_i | y_i)$$

- ▶ Define a dynamic programming table

$\pi(k, u, v) = \text{maximum probability of a tag sequence ending in tags } u, v \text{ at position } k$

that is,

$$\pi(k, u, v) = \max_{(y_{-1}, y_0, y_1, \dots, y_k) : y_{k-1}=u, y_k=v} r(y_{-1}, y_0, y_1 \dots y_k)$$

An Example

$\pi(k, u, v)$ = maximum probability of a tag sequence
ending in tags u, v at position k

The man saw the dog with the telescope

A Recursive Definition

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition:

For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

Justification for the Recursive Definition

For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

The man saw the dog with the telescope

The Viterbi Algorithm

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u, v)$ and $e(x|s)$.

Initialization: Set $\pi(0, *, *) = 1$

Definition: $\mathcal{S}_{-1} = \mathcal{S}_0 = \{*\}$, $\mathcal{S}_k = \mathcal{S}$ for $k \in \{1 \dots n\}$

Algorithm:

- ▶ For $k = 1 \dots n$,
 - ▶ For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

- ▶ **Return** $\max_{u \in \mathcal{S}_{n-1}, v \in \mathcal{S}_n} (\pi(n, u, v) \times q(\text{STOP}|u, v))$

The Viterbi Algorithm with Backpointers

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u, v)$ and $e(x|s)$.

Initialization: Set $\pi(0, *, *) = 1$

Definition: $\mathcal{S}_{-1} = \mathcal{S}_0 = \{*\}$, $\mathcal{S}_k = \mathcal{S}$ for $k \in \{1 \dots n\}$

Algorithm:

- ▶ For $k = 1 \dots n$,

- ▶ For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

$$bp(k, u, v) = \arg \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

- ▶ Set $(y_{n-1}, y_n) = \arg \max_{(u,v)} (\pi(n, u, v) \times q(\text{STOP}|u, v))$
- ▶ For $k = (n-2) \dots 1$, $y_k = bp(k+2, y_{k+1}, y_{k+2})$
- ▶ **Return** the tag sequence $y_1 \dots y_n$

The Viterbi Algorithm: Running Time

- ▶ $O(n|\mathcal{S}|^3)$ time to calculate $q(s|u, v) \times e(x_k|s)$ for all k, s, u, v .
- ▶ $n|\mathcal{S}|^2$ entries in π to be filled in.
- ▶ $O(|\mathcal{S}|)$ time to fill in one entry
- ▶ ⇒ $O(n|\mathcal{S}|^3)$ time in total

Pros and Cons

- ▶ Hidden markov model taggers are very simple to train (just need to compile counts from the training corpus)
- ▶ Perform relatively well (over 90% performance on named entity recognition)
- ▶ Main difficulty is modeling

$$e(\text{word} \mid \text{tag})$$

can be very difficult if “words” are complex

The Brown et al. Word Clustering Algorithm

Michael Collins, Columbia University

The Brown Clustering Algorithm

- ▶ Input: a (large) corpus of words
- ▶ Output 1: a partition of words into *word clusters*
- ▶ Output 2 (generalization of 1): a hierarchical word clustering

Example Clusters (from Brown et al, 1992)

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
June March July April January December October November September August
people guys folks fellows CEOs chaps doubters commies unfortunates blokes
down backwards ashore sideways southward northward overboard aloft downwards adrift
water gas coal liquid acid sand carbon steam shale iron
great big vast sudden mere sheer gigantic lifelong scant colossal
man woman boy girl lawyer doctor guy farmer teacher citizen
American Indian European Japanese German African Catholic Israeli Italian Arab
pressure temperature permeability density porosity stress velocity viscosity gravity tension
mother wife father son husband brother daughter sister boss uncle
machine device controller processor CPU printer spindle subsystem compiler plotter
John George James Bob Robert Paul William Jim David Mike
anyone someone anybody somebody
feet miles pounds degrees inches barrels tons acres meters bytes
director chief professor commissioner commander treasurer founder superintendent dean cus-
todian

A Sample Hierarchy (from Miller et al., NAACL 2004)

lawyer	1000001101000
newspaperman	100000110100100
stewardess	100000110100101
toxicologist	10000011010011
slang	1000001101010
babysitter	100000110101100
conspirator	1000001101011010
womanizer	1000001101011011
mailman	10000011010111
salesman	100000110110000
bookkeeper	1000001101100010
troubleshooter	10000011011000110
bouncer	10000011011000111
technician	1000001101100100
janitor	1000001101100101
saleswoman	1000001101100110
...	
Nike	1011011100100101011100
Maytag	10110111001001010111010
Generali	10110111001001010111011
Gap	1011011100100101011110
Harley-Davidson	10110111001001010111110
Enfield	101101110010010101111110
genus	101101110010010101111111
Microsoft	10110111001001011000
Ventrifex	101101110010010110010
Tractebel	1011011100100101100110
Synopsys	1011011100100101100111
WordPerfect	1011011100100101101000
...	
John	1011100100000000000
Consuelo	101110010000000001
Jeffrey	101110010000000010
Kenneth	10111001000000001100
Phillip	101110010000000011010
WILLIAM	101110010000000011011
Timothy	10111001000000001110
Terrence	101110010000000011110

The Intuition

- ▶ Similar words appear in similar contexts
- ▶ More precisely: similar words have similar distributions of words to their immediate left and right

The Formulation

- ▶ \mathcal{V} is the set of all words seen in the corpus w_1, w_2, \dots, w_n
- ▶ Say $C : \mathcal{V} \rightarrow \{1, 2, \dots, k\}$ is a *partition* of the vocabulary into k classes
- ▶ The model:

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^n e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1}))$$

(note: $C(w_0)$ is a special start state)

An Example

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^n e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1}))$$

An Example

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^n e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1}))$$

$$C(\text{the}) = 1, \quad C(\text{dog}) = C(\text{cat}) = 2, \quad C(\text{saw}) = 3$$

An Example

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^n e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1}))$$

$$C(\text{the}) = 1, \quad C(\text{dog}) = C(\text{cat}) = 2, \quad C(\text{saw}) = 3$$

$$e(\text{the}|1) = 1, \quad e(\text{cat}|2) = e(\text{dog}|2) = 0.5, \quad e(\text{saw}|3) = 1$$

An Example

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^n e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1}))$$

$$C(\text{the}) = 1, \quad C(\text{dog}) = C(\text{cat}) = 2, \quad C(\text{saw}) = 3$$

$$e(\text{the}|1) = 1, \quad e(\text{cat}|2) = e(\text{dog}|2) = 0.5, \quad e(\text{saw}|3) = 1$$

$$q(1|0) = 0.2, \quad q(2|1) = 0.4, \quad q(3|2) = 0.3, \quad q(1|3) = 0.6$$

An Example

$$p(w_1, w_2, \dots, w_n) = \prod_{i=1}^n e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1}))$$

$$C(\text{the}) = 1, \quad C(\text{dog}) = C(\text{cat}) = 2, \quad C(\text{saw}) = 3$$

$$e(\text{the}|1) = 1, \quad e(\text{cat}|2) = e(\text{dog}|2) = 0.5, \quad e(\text{saw}|3) = 1$$

$$q(1|0) = 0.2, \quad q(2|1) = 0.4, \quad q(3|2) = 0.3, \quad q(1|3) = 0.6$$

$$p(\text{the dog saw the cat}) =$$

The Brown Clustering Model

A Brown clustering model consists of:

- ▶ A vocabulary \mathcal{V}
- ▶ A function $C : \mathcal{V} \rightarrow \{1, 2, \dots, k\}$ defining a *partition* of the vocabulary into k classes
- ▶ A parameter $e(v|c)$ for every $v \in \mathcal{V}, c \in \{1 \dots k\}$
- ▶ A parameter $q(c'|c)$ for every $c', c \in \{1 \dots k\}$

Measuring the Quality of C

- ▶ How do we measure the quality of a partition C ?

$$\begin{aligned}\text{Quality}(C) &= \sum_{i=1}^n \log e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1})) \\ &= \sum_{c=1}^k \sum_{c'=1}^k p(c, c') \log \frac{p(c, c')}{p(c)p(c')} + G\end{aligned}$$

where G is a constant

- ▶ Here

$$p(c, c') = \frac{n(c, c')}{\sum_{c,c'} n(c, c')} \quad p(c) = \frac{n(c)}{\sum_c n(c)}$$

where $n(c)$ is the number of times class c occurs in the corpus, $n(c, c')$ is the number of times c' is seen following c , under the function C

A First Algorithm

- ▶ We start with $|\mathcal{V}|$ clusters: each word gets its own cluster
- ▶ Our aim is to find k final clusters
- ▶ We run $|\mathcal{V}| - k$ merge steps:
 - ▶ At each merge step we pick two clusters c_i and c_j , and merge them into a single cluster
 - ▶ We greedily pick merges such that

$$\text{Quality}(C)$$

for the clustering C after the merge step is maximized at each stage

- ▶ Cost? Naive = $O(|\mathcal{V}|^5)$. Improved algorithm gives $O(|\mathcal{V}|^3)$: still too slow for realistic values of $|\mathcal{V}|$

A Second Algorithm

- ▶ Parameter of the approach is m (e.g., $m = 1000$)
- ▶ Take the top m most frequent words, put each into its own cluster, c_1, c_2, \dots, c_m
- ▶ For $i = (m + 1) \dots |\mathcal{V}|$
 - ▶ Create a new cluster, c_{m+1} , for the i 'th most frequent word. We now have $m + 1$ clusters
 - ▶ Choose two clusters from $c_1 \dots c_{m+1}$ to be merged: pick the merge that gives a maximum value for $\text{Quality}(C)$. We're now back to m clusters
- ▶ Carry out $(m - 1)$ final merges, to create a full hierarchy

Running time: $O(|\mathcal{V}|m^2 + n)$ where n is corpus length

Name Tagging with Word Clusters and Discriminative Training

Scott Miller, Jethran Guinness, Alex Zamanian

BBN Technologies

10 Moulton Street

Cambridge, MA 02138

szmiller@bbn.com

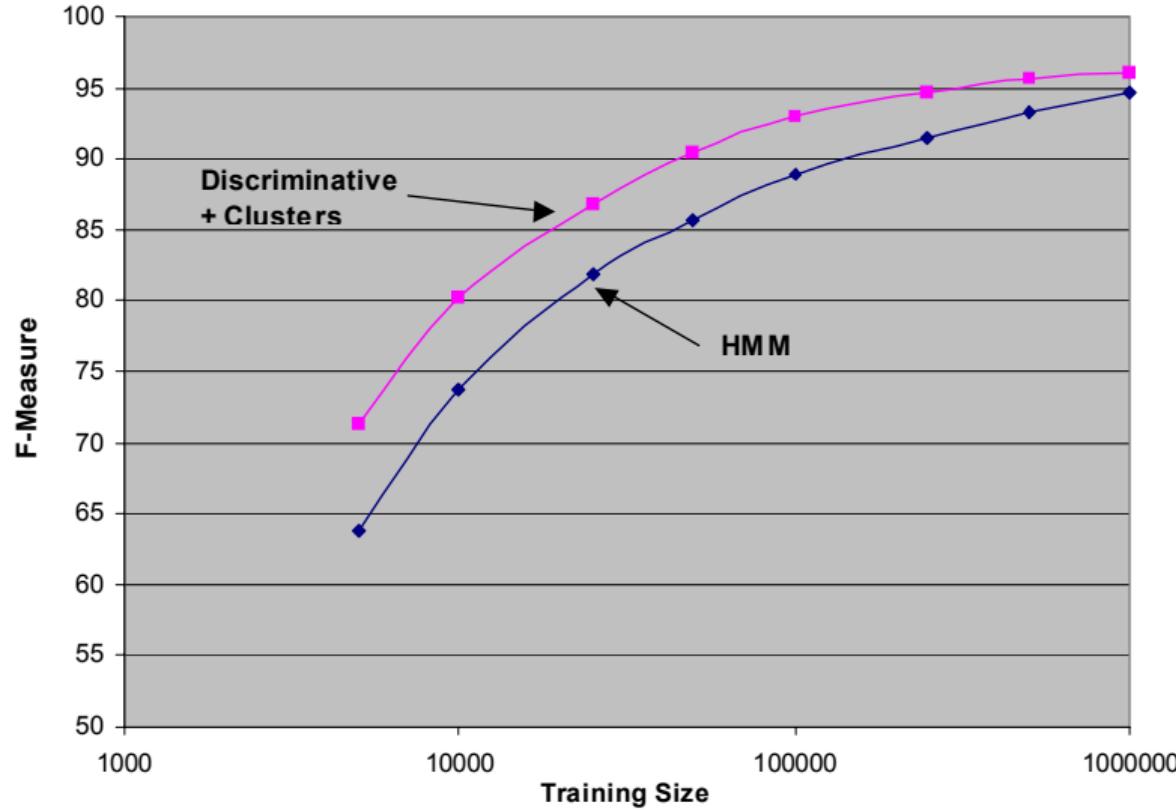
At a recent meeting, we presented name-tagging technology to a potential user. The technology had performed well in formal evaluations, had been applied successfully by several research groups, and required only annotated training examples to configure for new name classes. Nevertheless, it did not meet the user's needs.

To achieve reasonable performance, the HMM-based technology we presented required roughly 150,000 words of annotated examples, and over a million words to achieve peak accuracy. Given a typical annotation rate of 5,000 words per hour, we estimated that setting up a name finder for a new problem would take four person days of annotation work – a period we considered reasonable. However, this user's problems were too dynamic for that much setup time. To be useful, the system would have to be trainable in minutes or hours, not days or weeks.

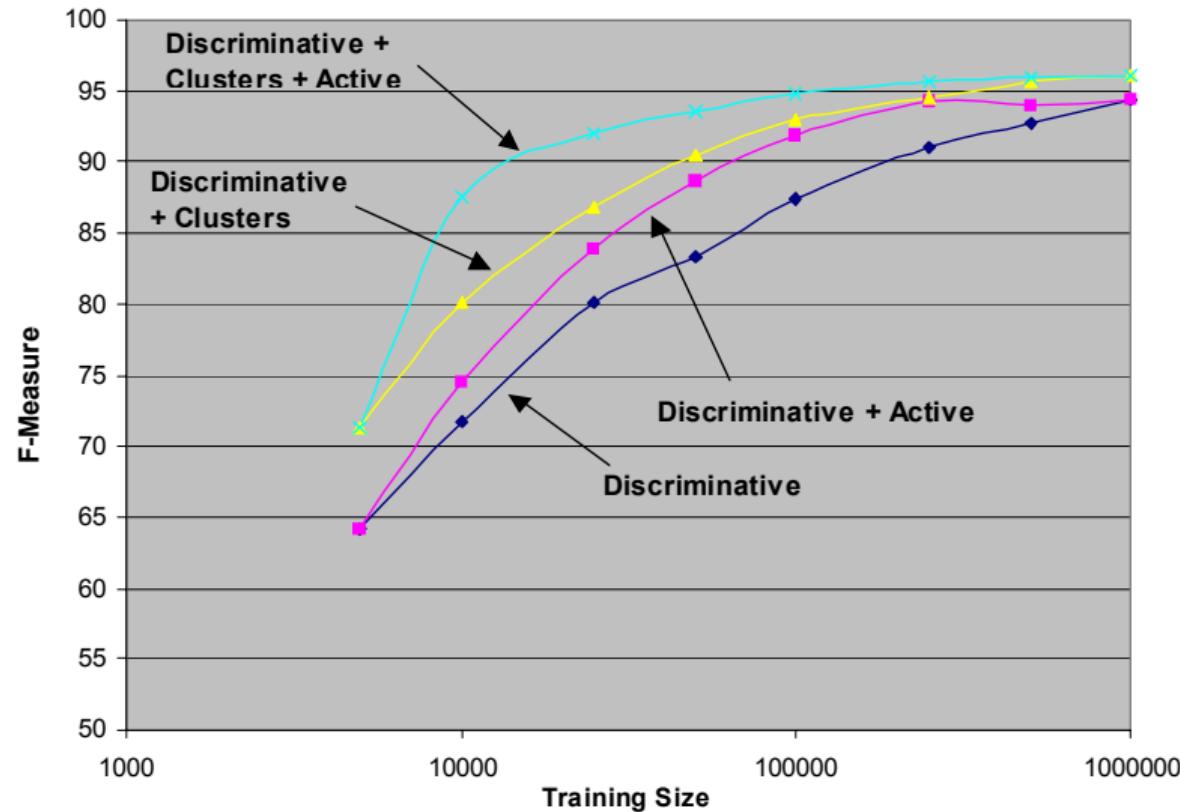
Miller et al, NAACL 2004

1. Tag + PrevTag
2. Tag + CurWord
3. Tag + CapAndNumFeatureOfCurWord
4. ReducedTag + CurWord
 //collapse start and continue tags
5. Tag + PrevWord
6. Tag + NextWord
7. Tag + DownCaseCurWord
8. Tag + Pref8ofCurrWord
9. Tag + Pref12ofCurrWord
10. Tag + Pref16ofCurrWord
11. Tag + Pref20ofCurrWord
12. Tag + Pref8ofPrevWord
13. Tag + Pref12ofPrevWord
14. Tag + Pref16ofPrevWord
15. Tag + Pref20ofPrevWord
16. Tag + Pref8ofNextWord
17. Tag + Pref12ofNextWord
18. Tag + Pref16ofNextWord
19. Tag + Pref20ofNextWord

Miller et al, NAACL 2004



Miller et al, NAACL 2004



Quizzes

[Help Center](#)

› Quizzes

✓ Quiz 1: covers material from weeks 1 and 2

[Help Center](#)[Attempt Quiz](#)**Hard Deadline** Mon 18 Mar 2013 8:59 PM PDT**Deadline** If you submit any time after the hard deadline, you will not receive credit.**Effective Score** 6.83 / 7.00*Explanation: $6.83 = 6.83 \text{ (Score for attempt 2)} * 100\% \text{ (No penalties)}$*

Each time that you attempt it, we'll record a score based on your performance and any penalties due to late submissions. Your effective score will be the highest score of all the **allowed** attempts made before the hard deadline.

of Attempts 2 / 100**Last Attempted** Sun 17 Mar 2013 10:38 AM PDT**Last Score** 6.83 / 7.00**Attempted Score**[Show Previous Attempts](#)

✓ Quiz 2: covers material from weeks 3 and 4

[Help Center](#)[Attempt Quiz](#)**Hard Deadline** Mon 1 Apr 2013 8:59 PM PDT**Deadline** If you submit any time after the hard deadline, you will not receive credit.**Effective Score** 5.67 / 6.00*Explanation: $5.67 = 5.67 \text{ (Score for attempt 2)} * 100\% \text{ (No penalties)}$*

Each time that you attempt it, we'll record a score based on your performance and any penalties due to late submissions. Your effective score will be the highest score of all the **allowed** attempts made before the hard deadline.

of 2 / 100

Attempts

Last Wed 27 Mar 2013 4:57 AM PDT

Attempted

Last 5.67 / 6.00

Attempted

Score

Show Previous Attempts

Feedback — Quiz 1: covers material from weeks 1 and 2

You submitted this quiz on **Sun 17 Mar 2013 10:38 AM PDT**. You got a score of **6.83** out of **7.00**. You can [attempt again](#), if you'd like.

[Help Center](#)

This is an open note quiz: you can use the slides from the class, and the notes at <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html> as a resource.

Question 1

Say we'd like to derive the Viterbi algorithm for a **bigram** HMM tagger. The model takes the form $p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-1}) \prod_{i=1}^n e(x_i | y_i)$. Which of the following statements is true?

Your Answer

Score

Explanation

- We can implement the Viterbi algorithm in exactly the same way as before, but with the following modification to the recursive definition:

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|u) \times e(x_k|v))$$

✓ 0.33

- We can use a dynamic programming algorithm with entries $\pi(k, u)$, and definitions $\pi(0, *) = 1$ and
- $$\pi(k, v) = \max_{u \in \mathcal{S}_{k-1}} (\pi(k-2, u) \times q(v|u) \times e(x_k|v))$$

✓ 0.33

- We can use a dynamic programming algorithm with entries $\pi(k, u)$, and definitions $\pi(0, *) = 1$ and
- $$\pi(k, v) = \max_{u \in \mathcal{S}_{k-1}} (\pi(k-1, u) \times q(v|u) \times e(x_k|v))$$

✓ 0.33

Total

1.00 /

1.00

Question 2

Say we define a backed-off model $q_{\text{BO}}(w_i | w_{i-1})$ exactly as we defined it in lecture, and we define the discounted counts as $\text{Count}^*(w_{i-1}, w_i) = \text{Count}(w_{i-1}, w_i) - 1.5$. Which of the following statements is true?

Your Answer	Score	Explanation
<input checked="" type="checkbox"/> There may be some bigrams u, v such that $q_{\text{BO}}(v u) < 0$	✓ 0.50	
<input type="checkbox"/> There may be some words u such that $\sum_{v \in \mathcal{V} \cup \{\text{STOP}\}} q_{\text{BO}}(v u) \neq 1$.	✓ 0.50	
Total	1.00 / 1.00	

Question 3

Consider the following two bigram language models (recall that a bigram language model defines $p(x_1 \dots x_n) = \prod_{i=1}^n q(x_i | x_{i-1})$).

Language Model 1

$$\mathcal{V} = \{\text{the}, \text{dog}\}$$

$$q(\text{the}|*) = q(\text{dog}|\text{the}) = q(\text{STOP}|\text{dog}) = 1$$

All other q parameters are equal to 0.

Language Model 2

$$\mathcal{V} = \{\text{the}, \text{a}, \text{dog}\}$$

$$q(\text{the}|*) = q(\text{a}|*) = 0.5$$

$$q(\text{dog}|\text{a}) = q(\text{dog}|\text{the}) = q(\text{STOP}|\text{dog}) = 1$$

All other q parameters are equal to 0.

Now assume that we have a test sentence consisting of a single sentence,

- the dog STOP

Which language model gives **lower** perplexity on this test corpus?

Your Answer	Score	Explanation
<input checked="" type="radio"/> Language Model 1	✓ 1.00	
<input type="radio"/> Language Model 2		
Total	1.00 / 1.00	

Question 4

We are now going to derive a version of the Viterbi algorithm that takes as input an integer n , and finds

$$\max_{y_1 \dots y_{n+1}, x_1 \dots x_n} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

for a trigram tagger, as defined in lecture. Hence the input to the algorithm is an integer n , and the output from the algorithm is the highest scoring **pair** of sequences $x_1 \dots x_n, y_1 \dots y_{n+1}$ under the model.

Which of the following recursive definitions gives a correct algorithm for this problem?

Your Answer

Score Explanation

- $\pi(0, *, *) = 1$, and
 $\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u))$

- $\pi(0, *, *) = 1$, and
 $\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times m(v)),$
where $m(v) = \max_{x \in \mathcal{V}} e(x|v)$

- None of the above.

Total

1.00 /

1.00

Question 5

We'd like to define a language model with $\mathcal{V} = \{\text{the, a, dog}\}$, and

$$p(x_1 \dots x_n) = \gamma \times 0.5^n$$

for any $x_1 \dots x_n$, such that $x_i \in \mathcal{V}$ for $i = 1 \dots (n-1)$, and $x_n = \text{STOP}$, where γ is some expression.

What should our definition of γ be?

(Hint: recall that $\sum_{n=1}^{\infty} 0.5^n = 1$)

Your Answer	Score	Explanation
<input type="radio"/> $\gamma = \frac{1}{3^n}$		
<input type="radio"/> $\gamma = 3^n$		
<input type="radio"/> $\gamma = 3^{n-1}$		
<input type="radio"/> $\gamma = 1$		
<input checked="" type="radio"/> $\gamma = \frac{1}{3^{n-1}}$	✓	1.00
Total	1.00 / 1.00	

Question 6

Say we train a trigram HMM tagger on a training set with the following two sentences:

- the dog saw the cat, D N V D N
- the cat saw the saw, D N V D N

Assume that we estimate the parameters of the HMM with maximum-likelihood estimation (no smoothing).

Now assume that we have the sentence

$x_1 \dots x_n = \text{the cat saw the saw}$

what is the value for

$\max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$ in this case? (Please give your answer up to 3 decimal places.)

You entered:

0.031

Your Answer

Score

Explanation

0.031



1.00

Total

1.00 / 1.00

Question 7

Assume we have a bigram language model with

$$\mathcal{V} = \{\text{the}, \text{a}\}$$

$q(\text{a}|*) = 0.6$, $q(\text{the}|*) = 0.4$, $q(\text{a}|\text{a}) = 0.9$, $q(\text{STOP}|\text{a}) = 0.1$, $q(\text{the}|\text{the}) = 0.8$, $q(\text{STOP}|\text{the}) = 0.2$, all other parameter values equal to 0.

Now say we'd like to define a bigram HMM model which defines the same distribution over sentences as the language model. By this we mean the following. The bigram HMM defines a distribution over sentences $x_1 \dots x_n$ paired with tag sequences $y_1 \dots y_{n+1}$ as follows:

$$p'(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q'(y_i | y_{i-1}) \prod_{i=1}^n e'(x_i | y_i)$$

(Note we use the notation p' , q' and e' to distinguish this from the distribution p and parameters q in the language model.)

The bigram HMM defines the same distribution over sentences as the language model if for any sentence $x_1 \dots x_n$,

$$p(x_1 \dots x_n) = \sum_{y_1 \dots y_{n+1}} p'(x_1 \dots x_n, y_1 \dots y_{n+1})$$

where p and p' are the distributions under the language model and the bigram HMM respectively.

Our HMM will have a set of tags $\mathcal{S} = \{1, 2\}$, and a vocabulary $\mathcal{V} = \{\text{the}, \text{a}\}$. We define $q'(1|*) = 0.6$.

In this question you should choose the parameters of the HMM so that it gives the same distribution over sentences as the language model given above. What should be the values for $q'(2|*)$, $q'(1|1)$, $q'(2|1)$, $q'(\text{STOP}|1)$, $e'(\text{the}|1)$, $e'(\text{the}|2)$?

Write your answers in order in the box below, separated by spaces. For example, you could write

0.2 0.3 1 0 0.4 0.5

You entered:

0.4 0.9 0.1 0.1 0 1

Your Answer	Score	Explanation
-------------	-------	-------------

0.4	✓	0.17
-----	---	------

0.9	✓	0.17
-----	---	------

0.1	✗	0.00
-----	---	------

0.1	✓	0.17
-----	---	------

0	✓	0.17
---	---	------

1	✓	0.17
---	---	------

Total	0.83 / 1.00
-------	-------------

Feedback — Quiz 2: covers material from weeks 3 and 4

You submitted this quiz on **Wed 27 Mar 2013 4:57 AM PDT**. You got a score of **5.67** out of **6.00**. You can [attempt again](#), if you'd like.

[Help Center](#)

Question 1

Say we have a context-free grammar with start symbol S, and the following rules:

- $S \rightarrow NP\ VP$
- $VP \rightarrow Vt\ NP$
- $Vt \rightarrow \text{saw}$
- $NP \rightarrow \text{John}$
- $NP \rightarrow DT\ NN$
- $DT \rightarrow \text{the}$
- $NN \rightarrow \text{dog}$
- $NN \rightarrow \text{cat}$
- $NN \rightarrow \text{house}$
- $NN \rightarrow \text{mouse}$
- $NP \rightarrow NP\ CC\ NP$
- $CC \rightarrow \text{and}$
- $PP \rightarrow IN\ NP$
- $NP \rightarrow NP\ PP$
- $IN \rightarrow \text{with}$
- $IN \rightarrow \text{in}$

How many parse trees do each of the following sentences have under this grammar?

1. John saw the cat and the dog
2. John saw the cat and the dog with the mouse
3. John saw the cat with the dog and the mouse

Write your answer as 3 numbers separated by spaces. For example if you think sentence 1 has 2 parses, sentence 2 has 5 parses, and sentence 3 has 3 parses, you would write

2 5 3

You entered:

1 3 2

Your Answer	Score	Explanation
1	✓ 0.33	
3	✗ 0.00	

2



0.33

Total

0.67 / 1.00

Question 2

Say we have a PCFG with start symbol S, and the following rules with associated probabilities:

- $q(S \rightarrow NP VP) = 1.0$
- $q(VP \rightarrow Vt NP) = 1.0$
- $q(Vt \rightarrow saw) = 1.0$
- $q(NP \rightarrow John) = 0.25$
- $q(NP \rightarrow DT NN) = 0.25$
- $q(NP \rightarrow NP CC NP) = 0.3$
- $q(NP \rightarrow NP PP) = 0.2$
- $q(DT \rightarrow the) = 1.0$
- $q(NN \rightarrow dog) = 0.25$
- $q(NN \rightarrow cat) = 0.25$
- $q(NN \rightarrow house) = 0.25$
- $q(NN \rightarrow mouse) = 0.25$
- $q(CC \rightarrow and) = 1.0$
- $q(PP \rightarrow IN NP) = 1.0$
- $q(IN \rightarrow with) = 0.5$
- $q(IN \rightarrow in) = 0.5$

Now assume we have the following sentence:

- John saw the cat and the dog with the mouse

Which of these statements is true?

Your Answer	Score	Explanation
<input checked="" type="radio"/> All parse trees for the sentence have the same probability under the PCFG	1.00	
<input type="radio"/> At least two parse trees for the sentence have different probabilities under the PCFG		
Total	1.00 / 1.00	

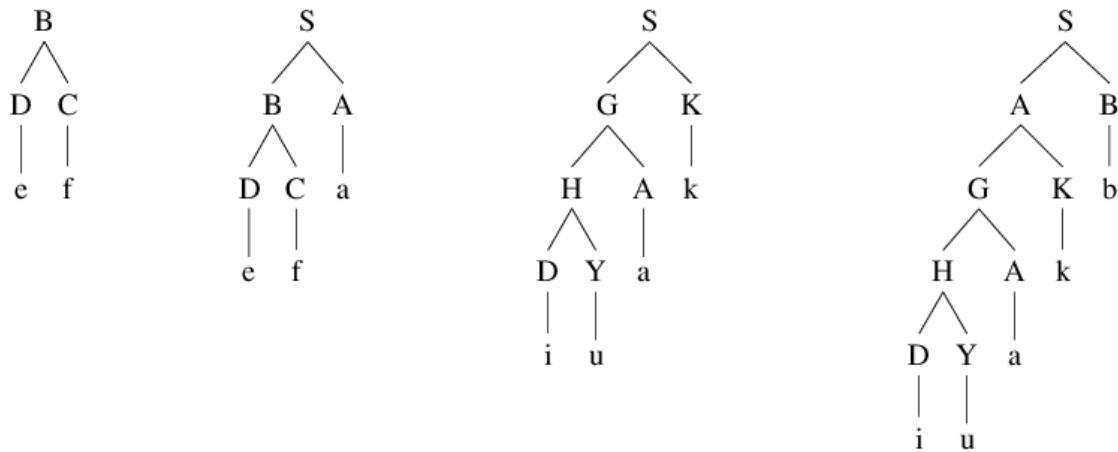
Question 3

Consider the CKY algorithm for parsing with PCFGs. The usual recursive definition in this

algorithm is as follows:

$$\pi(i, j, X) = \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

Now assume we'd like to modify the CKY parsing algorithm to that it returns the maximum probability for any *left-branching* tree for an input sentence. Here are some example left-branching trees:



It can be seen that in left-branching trees, whenever a rule of the form $X \rightarrow Y Z$ is seen in the tree, then the non-terminal Z must directly dominate a terminal symbol.

Which of the following recursive definitions is correct, assuming that our goal is to find the highest probability left-branching tree?

Your Answer

Score Explanation



$$\pi(i, j, X) = \max_{X \rightarrow Y Z \in R} (q(X \rightarrow Y Z) \times \pi(i, j - 1, Y) \times \pi(j, j, Z))$$



0.25



$$\pi(i, j, X) = \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{(i+1) \dots (j-1)\}}} (q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$



0.25



$$\pi(i, j, X) = \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$



0.25



✓ 0.25

$$\pi(i, j, X) = \max_{X \rightarrow Y} \sum_{Z \in R} (q(X \rightarrow Y | Z) \times \pi(i, i, Y) \times \pi(i + 1, j, Z))$$

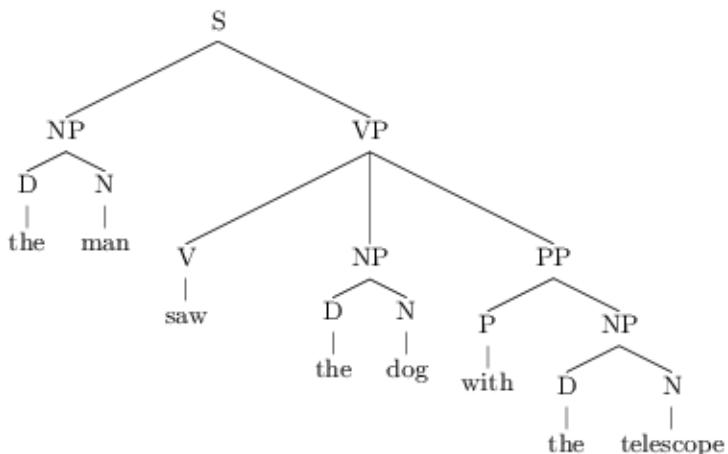
Total

1.00 /

1.00

Question 4

Consider the following parse tree:



Now assume that we add head-words to the non-terminals in the parse tree. We do this by specifying the following rules for finding the heads of context-free rules (note that these rules don't necessarily make sense from a linguistic standpoint):

- For the rule $S \rightarrow NP\ VP$, the VP is the head of the rule.
- For the rule $NP \rightarrow D\ N$, the N is the head of the rule.
- For the rule $PP \rightarrow P\ NP$, the NP is the head of the rule.
- For the rule $VP \rightarrow V\ NP\ PP$, the NP is the head of the rule.
- As is usual with head-finding rules, for any rule of the form $X \rightarrow w$ where X is a non-terminal, and w is a word, we take w to be the head of the rule (and X then has w as its head-word).

What are the head words for the following constituents?

- a) The NP "the man"
- b) The PP "with the telescope"
- c) The VP "saw the dog with the telescope"
- d) The S "the man saw the dog with the telescope"

Write the answer as four words separated by spaces, for example

the with saw the

You entered:

man telescope dog dog

Your Answer	Score	Explanation
man	✓ 0.25	
telescope	✓ 0.25	
dog	✓ 0.25	
dog	✓ 0.25	
Total	1.00 / 1.00	

Question 5

Say we have a PCFG with start symbol S, and rules and probabilities as follows:

$$q(S \rightarrow a) = 0.3$$

$$q(S \rightarrow a S) = 0.7$$

For any sentence $x = x_1 \dots x_n$, define $\mathcal{T}(x)$ to be the set of parse trees for x under the above PCFG. For any sentence x , define the probability of the sentence under the PCFG to be

$$p(x) = \sum_{t \in \mathcal{T}(x)} p(t)$$

where $p(t)$ is the probability of the tree under the PCFG.

Now assume we'd like to define a bigram language model with the same distribution over sentences as the PCFG. What should be the parameter values for $q(a|*)$, $q(a|a)$, and $q(\text{STOP}|a)$ so that the bigram language model gives the same distribution over sentences as the PCFG?

(For this question assume that the PCFG does not need to generate STOP symbols: for example the sentence "a a a" in the PCFG translates to the sentence "a a a STOP" in the bigram language model.)

Write your answer as a sequence of three numbers, for example

0.1 0.2 0.1

You entered:

1.0 0.7 0.3

Your Answer	Score	Explanation
1.0	✓ 0.33	
0.7	✓ 0.33	
0.3	✓ 0.33	
Total	1.00 / 1.00	

Question 6

Say we have a PCFG with the following rules and probabilities:

- $q(S \rightarrow NP VP) = 1.0$
- $q(VP \rightarrow Vt NP) = 0.2$
- $q(VP \rightarrow VP PP) = 0.8$
- $q(NP \rightarrow NNP) = 0.8$
- $q(NP \rightarrow NP PP) = 0.2$

- $q(\text{ NNP} \rightarrow \text{ John }) = 0.2$
- $q(\text{ NNP} \rightarrow \text{ Mary }) = 0.3$
- $q(\text{ NNP} \rightarrow \text{ Sally }) = 0.5$
- $q(\text{ PP} \rightarrow \text{ IN NP }) = 1.0$
- $q(\text{ IN} \rightarrow \text{ with }) = 1.0$
- $q(\text{ Vt} \rightarrow \text{ saw}) = 1.0$

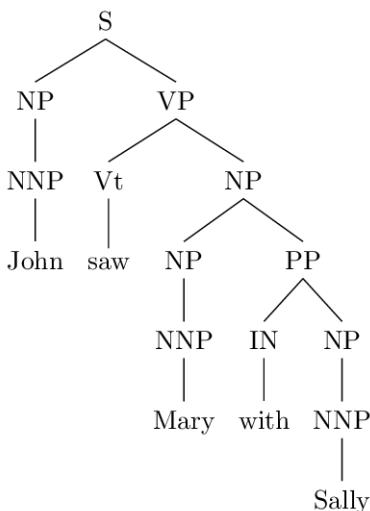
Now say we use the CKY algorithm to find the highest probability parse tree under this grammar for the sentence

- John saw Mary with Sally

We use t_{parser} to refer to the output of the CKY algorithm on this sentence.

(Note: assume here that we use a variant of the CKY algorithm that can return the highest probability parse under this grammar - don't worry that this grammar is not in Chomsky normal form, assume that we can handle grammars of this form!)

The gold-standard (human-annotated) parse tree for this sentence is



What is the precision and recall of t_{parser} (give your answers to 3 decimal places)?

Write your answer as a sequence of numbers: for example "0.3 0.8" would mean that your precision is 0.3, your recall is 0.8.

Here each non-terminal in the tree, excluding parts of speech, gives a "constituent" that is used in the definitions of precision and recall. For example, the gold-standard tree shown above has 7 constituents labeled S, NP, VP, NP, NP, PP, NP respectively (we exclude the parts of speech NNP, IN, and Vt).

You entered:

0.857 0.857

Your Answer	Score	Explanation
0.857	✓ 0.50	
0.857	✓ 0.50	
Total	1.00 / 1.00	

Assignments

[Help Center](#)**Submission Login** sandipan.dey@gmail.com**Submission Password** Y852NdTyaG [Generate New Password](#)

❖ Programming Assignments

❖ 1. Hidden Markov Models

[Help Center](#)[View Instructions](#)**Due Date**

Mon 1 Apr 2013 8:59 PM PDT

[Apply late days](#)

If you submit after the due date (but before the hard deadline), your submission score will be penalized by 3% for each day after the due date.

Hard Deadline

Mon 8 Apr 2013 8:59 PM PDT

If you submit any time after the hard deadline, you will not receive credit.

Part	Name	Last Submission	Score	Feedback	
1 / 3	Unigram Tagger	Tue 19 Mar 2013 12:59 PM PDT	20.00 / 20	View	Submit
2 / 3	Trigram Tagger	Fri 22 Mar 2013 12:52 PM PDT	30.00 / 30	View	Submit
3 / 3	Extended Tagger	Fri 22 Mar 2013 1:43 PM PDT	10.00 / 10	View	Submit
Total Score		60 / 60			

❖ 2. Parsing

[Help Center](#)[View Instructions](#)**Due Date**

Mon 15 Apr 2013 9:12 AM PDT

[Apply late days](#)

If you submit after the due date (but before the hard deadline), your submission score will be penalized by 3% for each day after the due date.

Hard Deadline	Mon 22 Apr 2013 9:00 AM PDT If you submit any time after the hard deadline, you will not receive credit.
----------------------	---

Part	Name	Last Submission	Score	Feedback	
1 / 3	Parser Setup	Sun 7 Apr 2013 5:14 AM PDT	20.00 / 20	View	Submit
2 / 3	CKY Decoder	Tue 9 Apr 2013 12:24 AM PDT	40.00 / 40	View	Submit
3 / 3	Markovization	Tue 9 Apr 2013 12:50 AM PDT	1.00 / 1	View	Submit
Total Score		61 / 61			

3. Translation Alignment

[Help Center](#)
[View Instructions](#)

Due Date	Wed 1 May 2013 9:00 AM PDT If you submit after the due date (but before the hard deadline), your submission score will be penalized by 3% for each day after the due date.
-----------------	---

Hard Deadline	Wed 8 May 2013 9:00 AM PDT If you submit any time after the hard deadline, you will not receive credit.
----------------------	--

Part	Name	Last Submission	Score	Feedback	
1 / 3	IBM Model 1	Wed 17 Apr 2013 8:28 AM PDT	25.00 / 25	View	Submit
2 / 3	IBM Model 2	Wed 17 Apr 2013 9:27 AM PDT	25.00 / 25	View	Submit
3 / 3	Growing Alignments	-	- / 1	View	Submit
Total Score		50 / 51			

4. Global Linear Models

[Help Center](#)
[View Instructions](#)

Due Date	Mon 13 May 2013 9:00 AM PDT If you submit after the due date (but before the hard deadline), your
-----------------	--

submission score will be penalized by 3% for each day after the due date.

Hard Deadline

Mon 20 May 2013 9:00 AM PDT

If you submit any time after the hard deadline, you will not receive credit.

Part	Name	Last Submission	Score	Feedback	
1 / 3	Tagging Decoder	Wed 8 May 2013 2:52 AM PDT	1.00 / 1	<input type="button" value="View"/>	<input type="button" value="Submit"/>
2 / 3	Perceptron	Thu 9 May 2013 1:15 AM PDT	0.00 / 1	<input type="button" value="View"/>	<input type="button" value="Submit"/>
3 / 3	Features	-	- / 1	<input type="button" value="View"/>	<input type="button" value="Submit"/>
Total Score		1 / 3			

NLP Programming Assignment 1: Hidden Markov Models

In this assignment, you will build a trigram hidden Markov model to identify gene names in biological text. Under this model the joint probability of a sentence x_1, x_2, \dots, x_n and a tag sequence y_1, y_2, \dots, y_n is defined as

$$p(x_1 \dots x_n, y_1 \dots y_n) = \\ q(y_1|*, *) \cdot q(y_2|*, y_1) \cdot q(\text{STOP}|y_{n-1}, y_n) \cdot \prod_{i=3}^n q(y_i|y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^n e(x_i|y_i)$$

where $*$ is a padding symbol that indicates the beginning of a sentence and `STOP` is a special HMM state indicating the end of a sentence. Your task will be to implement this probabilistic model and a decoder for finding the most likely tag sequence for new sentences.

The files for the assignment are located in the archive `h1-p.zip`. We provide a labeled training data set `gene.train`, a labeled and unlabeled version of the development set, `gene.key` and `gene.dev`, and an unlabeled test set `gene.test`. The labeled files take the format of one word per line with word and tag separated by space and a single blank line separates sentences, e.g.

```
Comparison O
with O
alkaline I-GENE
phosphatases I-GENE
and O
5 I-GENE
- I-GENE
nucleotidase I-GENE
```

```
Pharmacologic O
aspects O
of O
neonatal O
hyperbilirubinemia O
. O
:
```

The unlabeled files contain only the words of each sentence and will be used to evaluate the performance of your model.

The task consists of identifying gene names within biological text. In this dataset there is one type of entity: gene (GENE). The dataset is adapted from the BioCreAtIvE II shared task (http://biocreative.sourceforge.net/biocreative_2.html).

To help out with the assignment we have provided several utility scripts. Our code is written in Python, but you are free to use any language for your own implementation. Our scripts can be called at the command-line to pre-process the data and to check results.

Collecting Counts

The script `count_freqs.py` handles aggregating counts over the data. It takes a training file as input and produces trigram, bigram and emission counts. To see its behavior, run the script on the training data and pipe the output into a file

```
python count_freqs.py gene.train > gene.counts
```

Each line in the output contains the count for one event. There are two types of counts:

- Lines where the second token is `WORDTAG` contain emission counts $Count(y \rightsquigarrow x)$, for example

```
13 WORDTAG I-GENE consensus
```

indicates that `consensus` was tagged 13 times as `I-GENE` in the the training data.

- Lines where the second token is n -GRAM (where n is 1, 2 or 3) contain unigram counts $Count(y)$, bigram counts $Count(y_{n-1}, y_n)$, or trigram counts $Count(y_{n-2}, y_{n-1}, y_n)$. For example

```
16624 2-GRAM I-GENE O
```

indicates that there were 16624 instances of an `O` tag following an `I-GENE` tag and

```
9622 3-GRAM I-GENE I-GENE O
```

indicates that in 9622 cases the bigram `I-GENE I-GENE` was followed by an `O` tag.

Evaluation

The script `eval_gene_tagger.py` provides a way to check the output of a tagger. It takes the correct result and a user result as input and gives a detailed description of accuracy.

```
> python eval_gene_tagger.py gene.key gene_dev.p1.out
```

```
Found 2669 GENEs. Expected 642 GENEs; Correct: 424.
```

	precision	recall	F1-Score
GENE:	0.158861	0.660436	0.256116

Results for gene identification are given in terms of precision, recall, and F1-Score. Let \mathcal{A} be the set of instances that our tagger marked as GENE, and \mathcal{B} be the set of instances that are correctly GENE entities. Precision is defined as $|\mathcal{A} \cap \mathcal{B}| / |\mathcal{A}|$ whereas recall is defined as $|\mathcal{A} \cap \mathcal{B}| / |\mathcal{B}|$. F1-score represents the harmonic mean of these two values.

Part 1 (20 points)

- Using the counts produced by `count_freqs.py`, write a function that computes emission parameters

$$e(x|y) = \frac{\text{Count}(y \rightsquigarrow x)}{\text{Count}(y)}$$

- We need to predict emission probabilities for words in the test data that do not occur in the training data. One simple approach is to map infrequent words in the training data to a common class and to treat unseen words as members of this class. Replace infrequent **words** ($\text{Count}(x) < 5$) in the original training data file with a common symbol `_RARE_`. Then re-run `count_freqs.py` to produce new counts.
- As a baseline, implement a simple gene tagger that always produces the tag $y^* = \arg \max_y e(x|y)$ for each word x . Make sure your tagger uses the `_RARE_` word probabilities for rare and unseen words.

Your tagger should read in the counts file and the file `gene.dev` (which is `gene.key` without the tags) and produce output in the same format as the training file. For instance

```
Nations I-ORG
```

Write your output to a file called `gene.dev.p1.out` and locally evaluate by running

```
python eval-gene-tagger.py gene.key gene-dev.p1.out
```

The expected result should match the result above. When you are ready to submit, run your model on `gene.test` and write the output to `gene-test.p1.out`. Run `python submit.py` to submit.

Part 2 (30 Points)

- Using the counts produced by `count_freqs.py`, write a function that computes parameters

$$q(y_i|y_{i-2}, y_{i-1}) = \frac{\text{Count}(y_{i-2}, y_{i-1}, y_i)}{\text{Count}(y_{i-2}, y_{i-1})}$$

for a given trigram $y_{i-2} y_{i-1} y_i$. Make sure your function works for the boundary cases $q(y_1|*, *)$, $q(y_2|*, y_1)$ and $q(\text{STOP}|y_{n-1}, y_n)$.

- Using the maximum likelihood estimates for transitions and emissions, implement the Viterbi algorithm to compute

$$\arg \max_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n).$$

Be sure to replace infrequent words ($\text{Count}(x) < 5$) in the original training data file and in the decoding algorithm with a common symbol `_RARE_`. Your tagger should have the same basic functionality as the baseline tagger.

Run the Viterbi tagger on the development set. The model should have a total F1-Score of 0.40. When you are ready to submit, evaluate your model on `gene.test` and write the output to `gene-test.p2.out`. Run `python submit.py` to submit.

Part 3 (10 Points)

In lecture we discussed how HMM taggers can be improved by grouping words into informative word classes rather than just into a single class of rare words. For this part you should implement four rare word classes

Numeric The word is rare and contains at least one numeric characters.

All Capitals The word is rare and consists entirely of capitalized letters.

Last Capital The word is rare, not all capitals, and ends with a capital letter.

Rare The word is rare and does not fit in the other classes.

You can implement these by replacing words in the original training data and generating new counts by running `count_freqs.py`. Be sure to also replace words while testing.

The expected total development F1-Score is 0.42. When you are ready to submit, evaluate your model on `gene.test` and write the output to `gene-test.p3.out`. Run `python submit.py` to submit.

Natural Language Processing, Problem Set 2

In this programming assignment, you will train a probabilistic context-free grammar (PCFG) for syntactic parsing. In class we defined a PCFG as a tuple

$$G = (N, \Sigma, S, R, q)$$

where N is the set of non-terminals, Σ is the vocabulary, S is a root symbol, R is a set of rules, and q is the rule parameters. We will assume that the grammar is in Chomsky normal form (CNF). Recall from class that this means all rules in R will be either *binary rules* $X \rightarrow Y_1 Y_2$ where $X, Y_1, Y_2 \in N$ or *unary rules* $X \rightarrow Z$ where $X \in N$ and $Z \in \Sigma$.

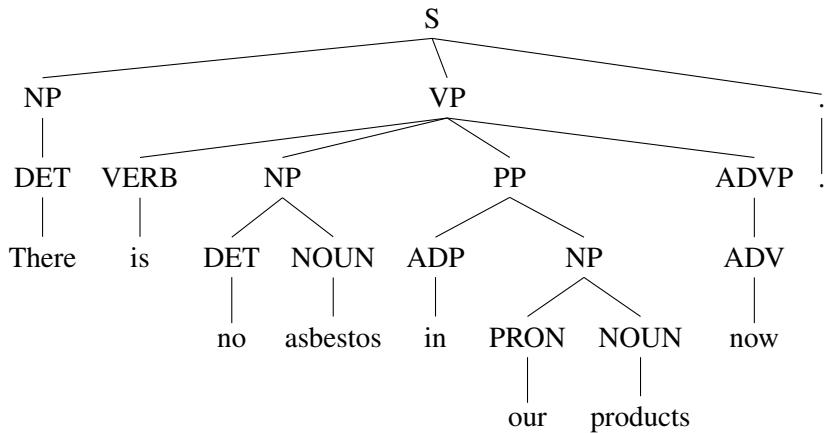
You will estimate the parameters of the grammar from a corpus of annotated questions from the QuestionBank [Judge et al., 2006]. Before diving into the details of the corpus format, we will discuss the rule structure of the grammar. You will not have to implement this section but it will be important to know when building your parser.

Rule Structure

Consider an example parse tree from the Wall Street Journal corpus (on which QuestionBank is modeled)

There/DET is/VERB no/DET asbestos/NOUN in/ADP our/PRON products/NOUN now/ADV ./.

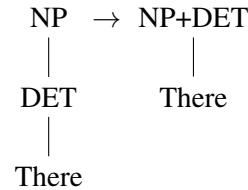
The correct parse tree for the sentence (as annotated by linguists) is as follows



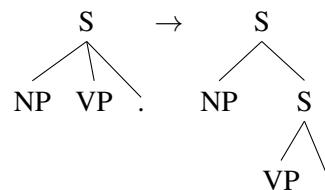
Note that nodes at the fringe of the tree are words, the nodes one level up are part-of-speech tags, and the internal nodes are syntactic tags. For the definition of the syntactic tags used in this corpus, see the work of Taylor et al. [2003]. The part-of-speech tags should be self-explanatory, for further reference see the work of Petrov et al. [2011].

Unfortunately the annotated parse tree is not in Chomsky normal form. For instance, the rule $S \rightarrow NP VP .$ has three children and the rule $NP \rightarrow DET$ has a single non-terminal child. We will need to work around this problem by applying a transformation to the grammar.

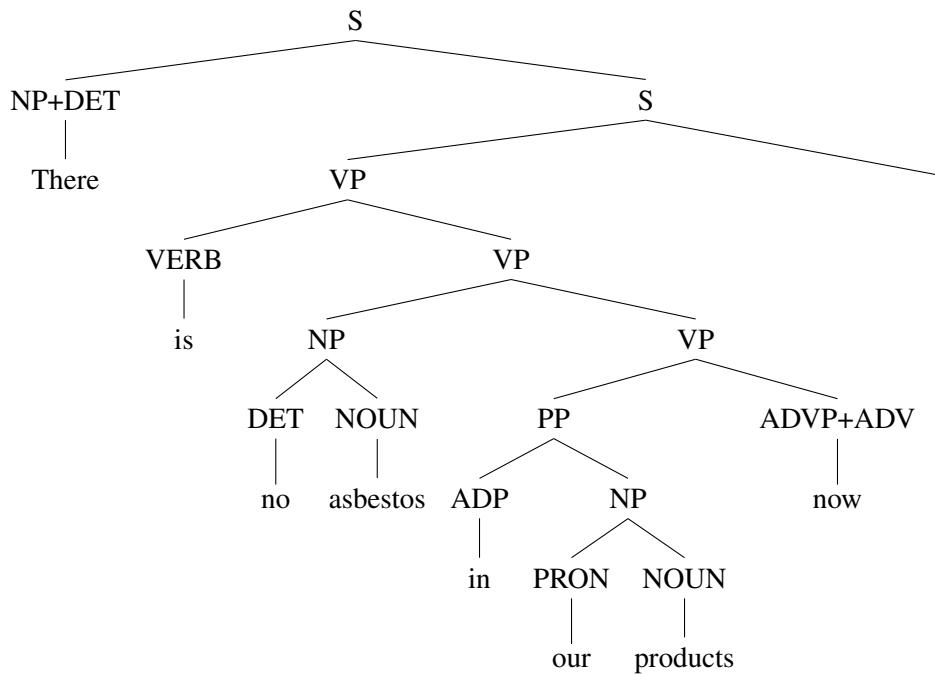
We will apply two transformations to the tree. The first is to collapse illegal unary rules of the form $X \rightarrow Y$ where $X, Y \in N$ into new non-terminals $X + Y$, for example



The second is to split n-ary rules $X \rightarrow Y_1 Y_2 Y_3$ where $X, Y_1, Y_2, Y_3 \in N$ into right branching binary rules of the form $X \rightarrow Y_1 X$ and $X \rightarrow Y_2 Y_3$ for example



With these transformation to the grammar, the correct parse tree for this sentence is



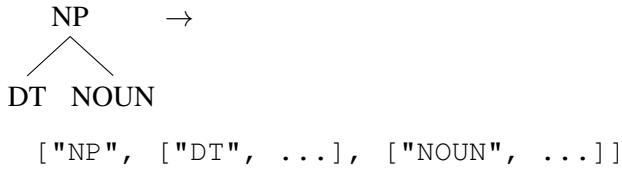
Converting the grammar to CNF will greatly simplify the algorithm for decoding. All the parses we provide for the assignment will be in the transformed format, you will not need to implement this transformation, but you should understand how it works.

Corpus Format

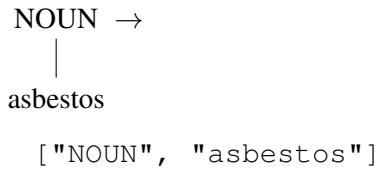
We provide a training data set with parses `parse_train.dat`, a development set of sentences `parse_dev.dat` along with the correct parses `parse_dev.key`, and a test set of sentences `parse_test.dat`. All the data comes from the QuestionBank corpus.

In the training set, each line of the file consists of a single parse tree in CNF. The trees are represented as nested multi-dimensional arrays encoded JSON. JSON is a general data encoding standard similar in spirit to XML. We use JSON because it is well-supported in pretty much every programming language (see <http://www.json.org/>).

Binary rules are represented as arrays of length 3.



Unary rules are represented as arrays of length 2.

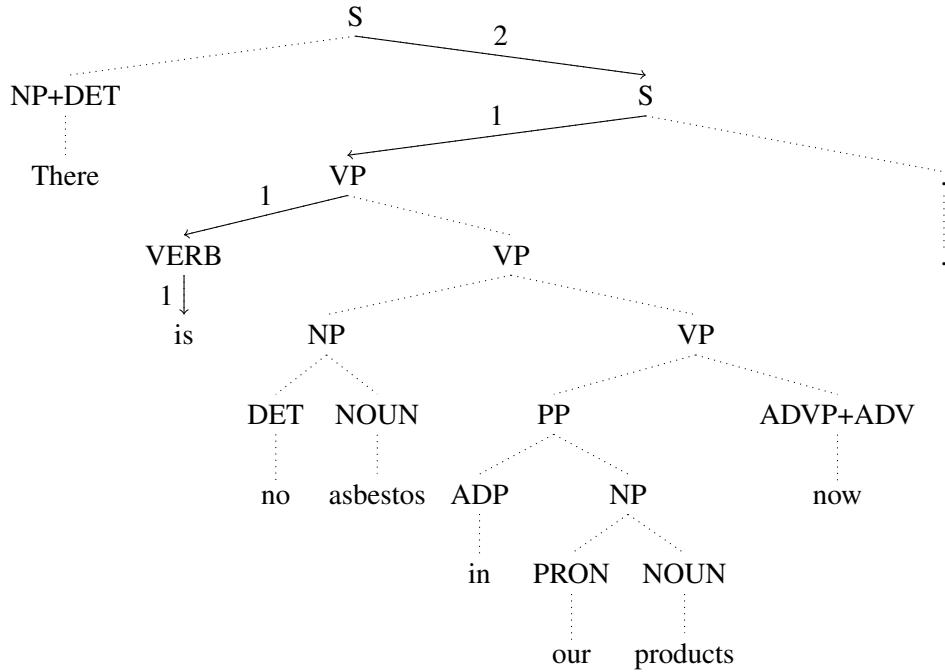


For example, the file `tree.example` has the tree given above

```
["S", ["NP", ["DET", "There"]], ["S", ["VP", ["VERB", "is"], ["VP", [
  NP, ["DET", "no"], ["NOUN", "asbestos"]]], ["VP", ["PP", ["ADP", "in
  "], ["NP", ["PRON", "our"], ["NOUN", "products"]]], ["ADVP", ["ADV",
  "now"]]]], [".", "."]]]
```

The tree is represented as a recursive, multi-dimensional JSON array. If the array is length 3 it is a binary rule, if it is length 2 it is a unary rule as shown above.

As an example, assume we want to access the node for “is” by walking down the tree as in the following diagram.



In Python, we can read and access this node using the following code

```
import json
tree = json.loads(open("tree.example").readline())
print tree[2][1][1][1]
```

The second line reads the tree into an array, and the third line walks down the tree to the node for “is”.

Other languages have similar libraries for reading in these arrays. Consult the forum to find a similar parser and sample code for your favorite language.

Tree Counts

In order to estimate the parameters of the model you will need the counts of the rules used in the corpus. The script `count_cfg_freq.py` reads in a training file and produces these counts. Run the script on the training data and pipe the output into some file:

```
python count_cfg_freq.py parse_train.dat > cfg.counts
```

Each line in the output contains the count for one event. There are three types of counts:

- Lines where the second token is `NONTERMINAL` contain counts of non-terminals $Count(X)$

```
17 NONTERMINAL NP
```

indicates that the non-terminal NP was used 17 times.

- Lines where the second token is `BINARYRULE` contain emission counts $Count(X \rightarrow Y_1 Y_2)$, for example

918 BINARYRULE NP DET NOUN

indicates that binary rule $\text{NP} \rightarrow \text{DET NOUN}$ was used 918 times in the training data.

- Lines where the second token is UNARYRULE contain unigram counts $\text{Count}(X \rightarrow Y)$.

8 UNARYRULE NP+NOUN place

indicates that the rule $\text{NP+NOUN} \rightarrow \text{place}$ was seen 8 times in the training data.

Part 1 (20 points)

- As in the tagging model, we need to predict emission probabilities for words in the test data that do not occur in the training data. Recall our approach is to map infrequent words in the training data to a common class and to treat unseen words as members of this class. Replace infrequent words ($\text{Count}(x) < 5$) in the original training data file with a common symbol `_RARE_`. Note that this is more difficult than the previous assignment because you will need to read the training file, find the words at the fringe of the tree, and rewrite the tree out in the correct format. We provide a script `python pretty_print_tree.py parse_dev.key` which you can use to view your trees in a more readable format. The script will also throw an error if the output is not in the correct format. Re-run `count_cfg_freq.py` to produce new counts.

Write the new counts out to a file `parse_train.counts.out` and submit with `submit.py`.

Part 2 (40 points)

- Using the counts produced by `count_cfg_freq.py`, write a function that computes rule parameters

$$q(X \rightarrow Y_1 Y_2) = \frac{\text{Count}(X \rightarrow Y_1 Y_2)}{\text{Count}(X)}$$

$$q(X \rightarrow w) = \frac{\text{Count}(X \rightarrow w)}{\text{Count}(X)}$$

- Using the maximum-likelihood estimates for the rule parameters, implement the CKY algorithm to compute

$$\arg \max_{t \in \mathcal{T}_G(S)} p(t).$$

Note: Since we are parsing questions, the root symbol $S = \text{SBARQ}$. This means you should return the best parse with SBARQ as the non-terminal spanning $[1, n]$.

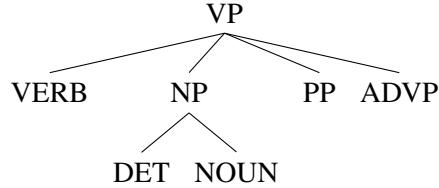
Your parser should read in the counts file (with rare words) and the file `parse_dev.dat` (which is just the sentence from `parse_dev.key` without the parse) and produce output in the following format: the tree for one sentence per line represented in the JSON tree format specified above. Each line should correspond to a parse for a sentence from `parse_dev.dat`.

View your output using `pretty_print_parse.py` and check your performance with `eval_parser.py` by running `python eval_parser.py parse_dev.key parse_dev.out`. The evaluator takes two files where each line is a matching parse in JSON format and outputs an evaluation description.

The expected development F1-Score is posted on the assignment page. When you are satisfied with development performance, run your decoder on `parse-test.dat` to produce `parse-test.p2.out`. Submit your output with `submit.py`.

Optional Part 3 (1 point)

Consider the following subtree of the original parse.



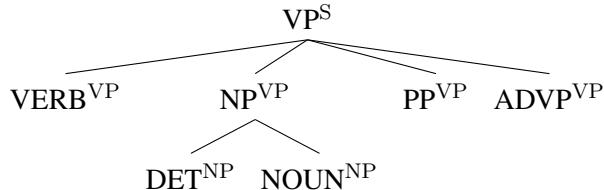
The tree structure makes the implicit independence assumption that the rule $NP \rightarrow DT\ NOUN$ is generated independently of the parent non-terminal VP, i.e. the probability of generating this subtree in context is

$$p(VP \rightarrow VERB\ NP\ PP\ ADVP \mid VP) \times p(NP \rightarrow DET\ NOUN \mid NP)$$

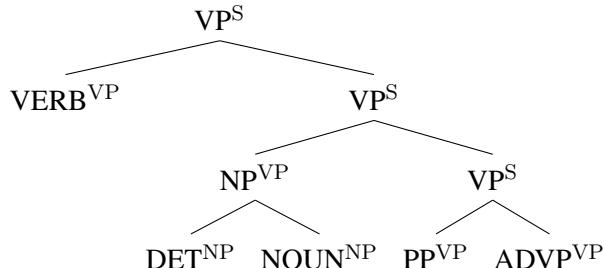
However, often this independence assumption is too strong. It can be informative for the lower rule to condition on the parent non-terminal as well, i.e. model generating this subtree as

$$p(VP \rightarrow VERB\ NP\ PP\ ADVP \mid VP) \times p(NP \rightarrow DET\ NOUN \mid NP, \text{parent}=VP)$$

This is a similar idea to using a trigram language model, except with parent non-terminals instead of words. We can implement this model without changing our parsing algorithm by applying a tree transformation known as *vertical markovization*. We simply augment each non-terminal with the symbol of its parent.



We apply vertical markovization before binarization; the new non-terminals are augmented with their original parent. After applying both transformations the final tree will look like



There is a downside of vertical markovization. The transformation greatly increases the number of rules in the grammar potentially causing data sparsity issues when estimating the probability model q . In practice

though, using one level of vertical markovization (conditioning on the parent), has been shown to increase accuracy.

- The file `parse_train_vert.dat` contains the original training sentence with vertical markovization applied to the parse trees. Train with the new file and reparse the development set. Run `python eval_parser.py parse_dev.key parse_dev.out` to evaluate performance. The expected development F1-Score is posted on the assignment page.

Note: This problem looks straightforward, but the difficulty is that we now have a much larger set of non-terminals N . It is possible that a simple implementation of CKY which worked for the last question will be too slow for this problem. If you find that your code is too slow, you will need to add optimizations to speed things up, while still finding the best parse tree. Specifically, think about how to avoid processing elements in π that already have zero probability and therefore cannot possibly lead to the highest-scoring parse tree.

The challenge is to improve the basic inefficiencies of the implementation in order to handle the extra non-terminals.

When you are satisfied with development performance, run your decoder on `parse_test.dat` to produce `parse_test.p3.out`. Submit your output with `submit.py`.

References

- John Judge, Aoife Cahill, and Josef Van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics, 2006.
- Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.
- Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The penn treebank: An overview. *Treebanks*, pages 1–22, 2003.

Natural Language Processing, Problem Set 3

This programming problem focuses on machine translation. The goal is to implement two translation models, IBM model 1 and IBM model 2, and apply these models to predict English/Spanish word alignments. Both models estimate the conditional probability of a foreign sentence $f_1 \dots f_m$ and an alignment $a_1 \dots a_m$ given a particular English sentence $e_1 \dots e_l$ and length m . The simpler model, IBM model 1, defines the conditional probability with a single set of parameters t as

$$p(f_1 \dots f_m, a_1 \dots a_m | e_1 \dots e_l, m) = \frac{1}{(l+1)^m} \prod_{i=1}^m t(f_i | e_{a_i})$$

The richer model, IBM model 2, defines the same conditional probability with two sets of parameters t and q

$$p(f_1 \dots f_m, a_1 \dots a_m | e_1 \dots e_l, m) = \prod_{i=1}^m q(a_i | i, l, m) t(f_i | e_{a_i})$$

Training Data

The main task of this assignment is to estimate these parameters from data. We estimate the parameters from a parallel corpus of aligned sentences. Each entry in the corpus contains two versions of the same sentence, one in English and one in Spanish. The parallel corpus we are using is a subset of the Europarl corpus [Koehn, 2005] accessed from <http://www.statmt.org/europarl/> which contains aligned sentences from the proceedings of the European Parliament.

The training corpus is split into two files, *corpus.en* and *corpus.es*, which contain English and Spanish sentences respectively. The i -th sentence in the English file is a translation of the i -th sentence in the Spanish file. The files are in UTF-8, contain one sentence per line, and words are separated by a space.

```
> head -n 3 corpus.en
resumption of the session
i declare resumed the session of the european parliament adjourned on ...
Although , as you will have seen , the dreaded ' millennium bug ' failed ...

> head -n 3 corpus.es
reanudación del período de sesiones
declaro reanudado el período de sesiones del parlamento europeo ...
como todos han podido comprobar , el gran " efecto del año 2000 " no se ...
```

IBM Model 1

The full algorithm for IBM model 1 is given on page 21 in the notes on machine translation <http://www.cs.columbia.edu/~mcollins/ibm12.pdf>. We recommend closely reading those notes for a full description of the problem. The core portion of the algorithm is summarized here.

Recall that IBM model 1 only has word translation parameters $t(f|e)$, which can be interpreted as the conditional probability of generating a foreign word f from an English word e (or from NULL).

We can estimate $t(f|e)$ using the EM algorithm, which iterates through the parallel corpus repeatedly. For the k -th sentence pair and each index i in the foreign sentence $f^{(k)}$ and j in the English sentence $e^{(k)}$, we define

$$\delta(k, i, j) = \frac{t(f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} t(f_i^{(k)} | e_j^{(k)})}$$

where l_k is the length of the English sentence. The delta function is used to update the *expected counts* for the current iteration:

$$\begin{aligned} c(e_j^{(k)}, f_i^{(k)}) &\leftarrow c(e_j^{(k)}, f_i^{(k)}) + \delta(k, i, j) \\ c(e_j^{(k)}) &\leftarrow c(e_j^{(k)}) + \delta(k, i, j) \end{aligned}$$

After each iteration through the parallel corpus, we revise our estimate for t parameters:

$$t(f|e) = \frac{c(e, f)}{c(e)}$$

for all possible foreign words f and English words e (and NULL).

IBM Model 2

The complete algorithm for estimating IBM model 2 parameters is given on page 13 of the notes. Be sure to understand the details of this algorithm before beginning to code the assignment.

We note here that IBM model 2 extends the implementation of the EM algorithm for IBM model 1. The main additional step is adapting the delta function to include $q(j|i, l, m)$ parameters

$$\delta(k, i, j) = \frac{q(j|i, l_k, m_k) t(f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} q(j|i, l_k, m_k) t(f_i^{(k)} | e_j^{(k)})}.$$

and computing expected counts $c(j|i, l, m)$ and $c(i, l, m)$.

After each iteration through the corpus we re-estimate the $t(f_i, e_j)$ parameters as before and add new updates for the q parameters:

$$q(j|i, l, m) = \frac{c(j|i, l, m)}{c(i, l, m)}.$$

Evaluation

We provide aligned development and test data, `dev.en` / `dev.es` and `test.en` / `test.es` in order to evaluate alignment accuracy. The format of these files is identical to the corpus training files, one sentence per line with aligned sentences.

We also provide a file, `dev.key`, which contains the manually annotated gold alignments for the development sentences. The key is adapted from Lambert et al. [2005] and accessed at http://gps-tsc.upc.es/veu/LR/epps_enwp_alignref.php3. The file contains lines of the form

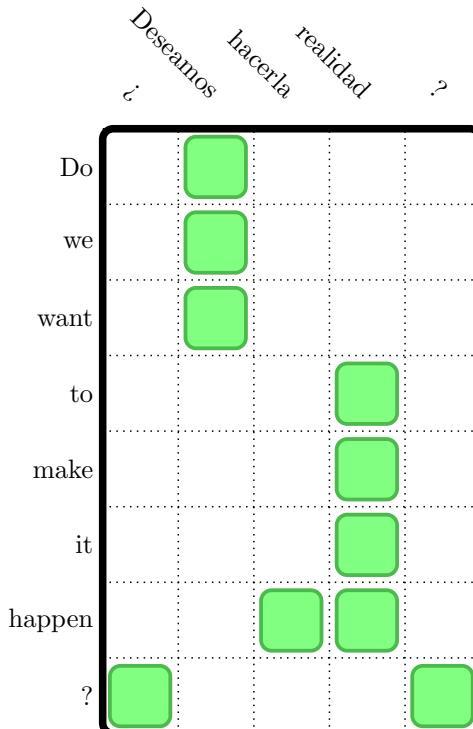
```
...
6 7 3
6 7 4
6 8 5
...
```

Each line specifies a word alignment of the form `SentenceIndex EnglishIndex ForeignIndex`. The first line of the example specifies that in the 6th pair of sentences the 7th English word is aligned to the 3rd Spanish word. Note that the gold alignments do not include NULL word alignments, all three indices start from 1, and unlike previous assignments there are no blank lines between sentences.

As an example, consider the 6th sentence pair in the development set.

- $e = \text{Do we want to make it happen ?}$
- $f = \zeta \text{ Deseamos hacerla realidad ?}$

The gold word alignment, 6 7 3, indicates that the English word “happen” is aligned with the Spanish word “hacerla”. The full alignment of the sentence pair is



For all of the questions in this assignment you should output predicted alignments in this format. We have included a script `eval_alignment.py` to evaluate the accuracy of these alignments. Assuming that you have written alignments to a file `dev.out`, running `python eval_alignment.py dev.key dev.out` will give the F-Score of the predicted alignments compared to the gold alignments.

Note: The alignments given in the key file often map single English words to multiple foreign words, whereas our models do not. For this reason it is impossible to get perfect recall or F-Score on this data set. For questions 1 and 2 we ignore this issue. Question 3 discusses a method to fix this problem.

Question 1 (25 points) - IBM Model 1

The first problem is to estimate the parameters of IBM model 1 using `corpus.en` and `corpus.es` as input. Before implementing please read the following notes carefully.

- Your implementation should only store t parameters for possible pairs of foreign and English words, i.e. words that occur together in some parallel translation, and the special English word `NULL`. We recommend implementing t as a collection of sparse maps for each English word e where the keys of the map are the words f that are occur with e and the values are the corresponding parameters $t(f|e)$. Using a non-sparse implementation will likely result in memory issues.
- In the initialization step, set $t(f|e)$ to be the uniform distribution over all foreign words that could be aligned to e in the corpus. More specifically

$$t(f|e) = \frac{1}{n(e)},$$

where $n(e)$ is the number of different words that occur in any translation of a sentence containing e . Note that the special English word `NULL` can be aligned to any foreign word in the corpus.

- Starting from the initial $t(f|e)$ parameters, run 5 iterations of the EM algorithm for IBM model 1 (this may take a while). When your model is completed, save your t parameters to a file, as you will need them for the next problem.
- Finally, use your model to find alignments for the development sentence pairs `dev.en` / `dev.es`. For each sentence, align each foreign word f_i to the English word with the highest $t(f|e)$ score, i.e.

$$a_i = \arg \max_{j \in \{0 \dots l\}} t(f_i|e_j).$$

Write your alignments to a file in the format of `dev.key` described above. Check the accuracy with `eval_alignments.py`.

When you are satisfied with development performance, run your model on `test.en` and `test.es` to produce `alignment-test.p1.out`. Submit your output with `submit.py`.

Question 2 (25 pts) - IBM Model 2

We now extend the alignment model to IBM model 2 by adding alignment parameters $q(j|i, l, m)$.

- Initialize the q parameters to the uniform distribution over all j for each i, l , and m , i.e.

$$q(j|i, l, m) = \frac{1}{l + 1}$$

You only need to store parameters for pairs of sentence lengths l and m that occur in the corpus.

- To initialize the $t(f|e)$ parameters, use the last set of parameters (after 5 iterations) produced by your implementation of IBM model 1.
- Run 5 iterations of EM for IBM model 2.
- As before, use the model to compute alignments for the development sentence pairs in the corpus. For each foreign word f_i , the best alignment is

$$a_i = \arg \max_{j \in \{0 \dots l\}} q(j|i, l, m) t(f_i|e_j).$$

Write your alignments to a file in the format of `dev.key` described above. Check the accuracy with `eval_alignments.py`.

When you are satisfied with development performance, run your model on `test.en` and `test.es` to produce `alignment-test.p2.out`. Submit your output with `submit.py`.

Optional Question 3 (1 point) - Growing Alignments

(This question is an optional extension of the first two problems.)

As we noted above, the gold alignments allow English words to be aligned with multiple Spanish words. This problem explores a method to get around this issue and generate more complete alignments. This process is a crucial first step in order to extract a lexicon for phrase-based translation.

The recipe we use is described on slide 11 of the lectures on phrase-based translation. It consists of the following steps

1. Estimate IBM model 2 for $p(f|e)$ (exactly as in question 2).
2. Estimate IBM model 2 for $p(e|f)$ (as in question 2 **except** with English as the foreign language).
3. Calculate the best alignments with $p(f|e)$ and $p(e|f)$.
4. Calculate the intersection and union of these alignments.
5. Starting from the intersection, apply a heuristic to grow the alignment.

A good example heuristic is sketched out on slide 12.

- Only explore alignment in the union of the $p(f|e)$ and $p(e|f)$ alignments.
- Add one alignment point at a time.
- Only add alignment points which align a word that currently has no alignment.

- At first, restrict ourselves to alignment points that are “neighbors” (adjacent or diagonal) of current alignment points. Later, consider other alignment points.

For this problem, you should estimate both $p(f|e)$ and $p(e|f)$ and use these models to generate new alignments. We recommend starting with the heuristic from the lecture and begin with the intersection alignment while growing towards the union alignment. However, feel free to try out different alignment heuristics to improve the score of the model. The auto-grader for this problem will not have an upper-bound on accuracy.

When you are satisfied with development performance, run your model on `test.en` and `test.es` to produce `alignment-test.p3.out`. Submit your output with `submit.py`.

References

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, 2005.

Patrik Lambert, Adrià De Gispert, Rafael Banchs, and José B Mariño. Guidelines for word alignment evaluation and manual alignment. *Language Resources and Evaluation*, 39(4):267–285, 2005.

Natural Language Processing: Problem Set 4

In this programming assignment, you will train a global linear model for named-entity recognition using the perceptron algorithm. In the decoding problem for an input instance $x \in \mathcal{X}$, a global linear model (GLM) uses the following three components: (1) a function that generates possible output structures $\text{GEN}(x)$, e.g. x is a sentence w_1, \dots, w_n and $y \in \text{GEN}(x)$ is a possible tagging t_1, \dots, t_n , (2) a feature function $f(x, y)$, and (3) a weight vector $v \in R^d$. The decoding problem is defined as

$$y^* = \arg \max_{y \in \text{GEN}(x)} v \cdot f(x, y)$$

We saw in lecture that if the feature function f factors into local features then decoding is tractable even when $\text{GEN}(x)$ is large. For tagging, we partition a tagging into a sequence of histories, h_1, \dots, h_n and decompose f into a sum of local feature functions over each history and output pair, i.e. $\sum_{i=1}^n g(h_i, t_i)$. Our focus will be on trigram tagging, so we define a history as

$$h = \langle t_{-2}, t_{-1}, x, i \rangle$$

The full decoding problem can be described in terms of local features over history/tag pairs as

$$y^* = v \cdot \arg \max_{y \in \text{GEN}(x)} f(x, y) = \arg \max_{t_1, \dots, t_n \in \text{GEN}(x)} \sum_{i=1}^n v \cdot g(\langle t_{i-2}, t_{i-1}, i, x \rangle, t)$$

In this assignment you will experiment with different feature functions and implement the perceptron training algorithm for learning the vector v .

Data

The data and evaluation for this assignment is identical to PA 1.

Recall that we provide a labeled training data set `gene.train`, a labeled and unlabeled version of the development set, `gene.key` and `gene.dev`, and an unlabeled test set `gene.test`. The labeled files take the format of one word per line with each word and tag separated by a space, and each sentence separated by a blank line. The unlabeled files contain only the words of each sentence and will be used to evaluate the performance of your model.

The task consists of identifying gene names within a biological text. In this dataset there is one type of entity: gene (GENE). The set of tags for this assignment is $\mathcal{T} = \{\text{O}, \text{I-GENE}\}$. The dataset is adapted from the BioCreAtIVe II shared task (http://biocreative.sourceforge.net/biocreative_2.html).

Histories

For the first part of the assignment, we assume the weight vector v is given and fixed. The goal is to solve the GLM decoding problem with this vector. Consider the task of tagging a typical sentence from PA 1

```

of
lipase
activity
.
```

The first crucial step is to understand the set of possible histories for the sentence. For instance, for $i = 3$ the possible histories and tag pairs consist of

- $(\langle \text{I-GENE}, \text{I-GENE}, x, 3 \rangle, \text{I-GENE})$
- $(\langle \text{O}, \text{I-GENE}, x, 3 \rangle, \text{I-GENE})$
- $(\langle \text{I-GENE}, \text{O}, x, 3 \rangle, \text{I-GENE})$
- $(\langle \text{O}, \text{O}, x, 3 \rangle, \text{I-GENE})$
- $(\langle \text{I-GENE}, \text{I-GENE}, x, 3 \rangle, \text{O})$
- $(\langle \text{O}, \text{I-GENE}, x, 3 \rangle, \text{O})$
- $(\langle \text{I-GENE}, \text{O}, x, 3 \rangle, \text{O})$
- $(\langle \text{O}, \text{O}, x, 3 \rangle, \text{O})$

Each of these history tag pairs will map to a different set of local features and a different score.

Feature Vectors

Once we have a history we can compute its local features. Much of the art of global linear models is selecting good features to describe the data. Feature functions typically look like

$$g_1(\langle t_{-2}, t_{-1}, x, i \rangle, t) = \begin{cases} 1 & \text{if } t_{-2} = \text{O}, t_{-1} = \text{O}, t = \text{I-GENE} \\ 0 & \text{otherwise} \end{cases}$$

which is a local binary feature indicating that the previous two tags are O and the current tag is I-GENE. A common practice is to give features more informative names. For instance, let us call this feature “TRIGRAM:O:O:I-GENE”. We will also include features “TRIGRAM:s:u:v” for all $s, u, v \in \mathcal{T}$, so this naming helps organize the feature vector.

Once we have the set of features, we need to be able to compute the feature vector $g(h, t)$ for a history/tag pair h, t as well as the inner product $v \cdot g(h, t)$.

- We recommend implementing the feature vector (the output of $g(h, t)$) as a map from strings to counts. Since most features are 0 for a given history, i.e. the local feature vector is sparse, this is more efficient than representing the full vector. For instance if the history/tag pair is $(h, t) = (\langle \text{O}, \text{O}, x, 3 \rangle, \text{I-GENE})$, then $g(h, t)$ would contain the mapping “TRIGRAM:O:O:I-GENE” $\rightarrow 1$.

For this assignment the weight vector v can also be implemented as a map. For instance, in Python the weight vector v might be a dictionary mapping features to weights, e.g.

```
print v["TRIGRAM:O:O:I-GENE"]
```

The crucial inner product $v \cdot g(h, t)$ in the decoding problem can then be implemented as a product between these two maps. In Python, this might look like

```
sum( (v[k] * val for k, val in g.iteritems()) )
```

For more details about this sparse representation, see the notes on log-linear models.

Decoding

So far we have only discussed local histories and features. To find the best global tagging for a sentence, we solve the following decoding problem:

$$y^* = \arg \max_{y \in \text{GEN}(x)} v \cdot f(x, y) = \arg \max_{t_1, \dots, t_n \in \text{GEN}(x)} \sum_{i=1}^n v \cdot g(\langle t_{i-2}, t_{i-1}, i, x \rangle, t)$$

Luckily this global maximization can be solved using a variation of the Viterbi algorithm. We assume \mathcal{S} is defined the same as with HMMs. The main difference in the algorithm is the scoring function used

```
procedure VITERBIGLM( $v, g, x$ )  $\triangleright v$  is the weight vector,  $g$  is the feature function,  $x$  is the sentence
 $\pi(0, *, *) = 0$ 
for  $k = 1 \dots n$  do
    for  $u \in \mathcal{S}_{k-1}, s \in \mathcal{S}_k$  do
         $\pi(k, u, s) = \max_{t \in \mathcal{S}_{k-2}} \pi(k-1, t, u) + v \cdot g(\langle t, u, x, k \rangle, s)$ 
         $bp(k, u, s) = \arg \max_{t \in \mathcal{S}_{k-2}} \pi(k-1, t, u) + v \cdot g(\langle t, u, x, k \rangle, s)$ 
     $(t_{n-1}, t_n) = \arg \max_{u \in \mathcal{S}_{n-1}, s \in \mathcal{S}_n} bp(n, u, s) + v \cdot g(\langle u, s, x, n+1 \rangle, \text{STOP})$ 
    for  $k = (n-2) \dots 1$  do
         $t_k = bp(k+2, t_{k+1}, t_{k+2})$ 
    return  $t_1, \dots, t_n$ 
```

Note that the GLM Viterbi algorithm requires computing the local score $v \cdot g(\langle t, u, x, k \rangle, s)$. We recommend abstracting this part out of the function and computing it as described in the previous section. The rest of the code is very similar to HMM Viterbi. You should be able to repurpose the code from PA 1 for this part of the assignment.

Part 1 (1 points)

In this problem you will experiment with building a trigram tagging decoder with a pre-trained model.

- Consider a very simple model with two types of feature functions for all tags s, u, v and words r

$$g_{\text{TRIGRAM:s:u:v}}(\langle t_{-2}, t_{-1}, x, i \rangle, t) = \begin{cases} 1 & \text{if } t_{-2} = s \text{ and } t_{-1} = u \text{ and } t = v \\ 0 & \text{otherwise} \end{cases}$$

$$g_{\text{TGAG:u:r}}(\langle t_{-2}, t_{-1}, (w_1 \dots w_n), i \rangle, t) = \begin{cases} 1 & \text{if } w_i = r \text{ and } t = u \\ 0 & \text{otherwise} \end{cases}$$

Consider the example sentence (“Characteristics of ...”) as x , we extract the following feature vector (as a map from features to counts) from an example history/tag pair

$$g(\langle O, O, x, 3 \rangle, I - GENE) = \{ "TRIGRAM:O:O:I-GENE" \rightarrow 1, "TAG:I-GENE:lipase" \rightarrow 1 \}$$

For this problem we provide a pre-trained weight vector v for these features (initialized $v = 0$ and trained for $k = 5$ iterations). The file `tag.model` contains the vector. All lines in the file consist of two columns `FEATURE`, `WEIGHT` indicating $v(FEATURE) = WEIGHT$. More specifically.

- Lines of the form `TAG:lipase:I-GENE 0.23` mean the feature `TAG:lipase:I-GENE` representing lipase tagged with `I-GENE` has weight 0.23.
- Lines of the form `TRIGRAM:O:O:I-GENE 0.45` mean the feature `TRIGRAM:O:O:I-GENE` representing the trigram `O, O, I-GENE` has weight 0.45.
- The goal of this problem is to decode with this model. First read `tag.model` into a map from feature strings to weights. Next for each sentence in development data run the Viterbi algorithm as described above.

Write your output to a file called `gene_dev.p1.out` and locally evaluate by running

```
python eval_gene_tagger.py gene.key gene_dev.p1.out
```

When you are ready to submit, run your model on `gene.test` and write the output to `gene_test.p1.out`. Run `python submit.py` to submit.

Perceptron Training

The next two parts of the assignment focus on estimating the weight vector v using the perceptron algorithm. Note that unlike HMMs we cannot simply read off the parameters from the training data. We will instead repeatedly solve the decoding problem on the training data to estimate better parameters.

Training the tagger requires a corpus of tagged examples $(x^{(i)}, y^{(i)})$ for $i \in \{1, \dots, M\}$. The file `gene.train` contains these training sentences.

Consider the tagged version of the training data above.

```
Characteristics O
of O
lipase I-GENE
activity O
. O
```

We can see that $(\langle O, O, x, 3 \rangle, I-GENE)$ is the correct history/tag pair for position $i = 3$.

The perceptron algorithm is defined in detail in the Week 10 class slides. The algorithm looks a bit complicated, but once you understand it, the code should be very simple.

We start with $v = 0$, then for each sentence/tag pair (x, y) in the training data in order we run the following update

1. Compute the best tagging, $z = \arg \max_{t_1, \dots, t_n \in \text{GEN}(x)} \sum_{i=1}^n v \cdot g(\langle t_{i-2}, t_{i-1}, i, x \rangle, t)$
2. Compute the best tagging feature vector, $f(x, z)$
3. Compute the gold tagging feature vector, $f(x, y)$
4. Update the weights, $v \leftarrow v + f(x, y) - f(x, z)$

The first step should make use of Part 1. The second and third steps should use your helper function from above. The final step is vector arithmetic.

Part 2 (1 point)

A major reason that the model from Part 1 performs so poorly is that it does nothing to handle rare words. We could introduce `_RARE_` tokens or word classes; however, a major benefit of GLMs is that we can instead introduce features that target specific properties of the words.

A good example of this type of feature is word suffixes. We saw in PA1 that suffixes can be a useful indicator as a word class, so we expect they will be informative features. The new features are, for all suffixes u , tag v , and lengths $j \in \{1, 2, 3\}$

$$g_{\text{SUFF}:u;j:v}(\langle t_{-2}, t_{-1}, (w_1 \dots w_n), i \rangle, t) = \begin{cases} 1 & \text{if } u = \text{suffix}(w_i, j) \text{ and } t = v \\ 0 & \text{otherwise} \end{cases}$$

where $\text{suffix}(w, j)$ returns the last j letters of w .

- For this question, you should first implement the perceptron algorithm to estimate a weight vector v for the new set of features. Start with $v = 0$ and run $K = 5$ iterations. Follow the steps in the Perceptron Training section. When your code is finished write the final model out to `suffix_tagger.model`.
- After training is complete, run your decoder from Part 1 (with the suffix features) and then run on the development set.

When you are ready to submit, run your model on `gene.test` and write the output to `gene_test.p2.out`. Run `python submit.py` to submit.

Part 3 (1 Point)

As mentioned above, the benefit of this formulation is the ability to add arbitrary features conditioned on the input sentence x . In the last question we experimented with adding suffix features from i -th word of the sentence. Many other features have been shown to be useful for tagging.

- For this question you should run tagging experiments with custom features. There are many ways to come up with new features. One strategy is to look at the errors your tagger made in Part 2 and try out features that target specific issues. We also encourage you to look up features in the NLP literature or in open-source taggers. Don't be discouraged if adding seemingly useful features sometimes decreases the accuracy of your tagger; feature selection can be a difficult problem.

When you are ready to submit, run your model on `gene.test` and write the output to `gene_test.p3.out`. Run `python submit.py` to submit.