



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 1  
Date – 29<sup>th</sup> August 2020  
Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Session Content

---

- Objective of course
  - Evaluation Plan
  - What is Natural Language Processing?
  - Application areas of Natural Language Processing
  - What will we learn in this course?
  - Introduction to Natural Language Processing
-

# Objective of course

---

- To learn the fundamental concepts and techniques of natural language processing (NLP)
- To learn computational properties of natural languages and the commonly used algorithms for processing linguistic information
- To apply NLP techniques in state of art applications
- To learn implementation of NLP algorithms and techniques

# Books

---

## Text books and Reference book(s)

T1	Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition]
T2	Natural language understanding[2nd edition] by James Allen
R1	Handbook of Natural Language Processing, Second Edition— NitinIndurkhy, Fred J. Damerau, Fred J. Damerau
R2	Natural Language Processing with Python by Steven Bird, Ewan Klein, Edward Lopper

# Evaluation Plan

Name	Type	Duration	Weight
Assignment-I	Take Home		15%
Quiz	Online		5%
Mid-Semester Exam (after 7 sessions)	Closed Book	1.5 Hrs	30%
Comprehensive Exam	Open Book	2.5 Hrs	50%

# What is Natural Language Processing?

---

- Natural Language Processing
  - Process information contained in natural language text.
  - Also known as Computational Linguistics (CL), Human Language Technology (HLT), Natural Language Engineering (NLE)

# What is it..

---

- Analyze, understand and generate human languages just like humans do.
  - Applying computational techniques to language domain..
  - To explain linguistic theories, to use the theories to build systems that can be of social use..
  - Started off as a branch of Artificial Intelligence..
  - Borrows from Linguistics, Psycholinguistics, Cognitive Science & Statistics.
  - Make computers learn our language rather than we learn theirs.
-

# Why Study NLP?

---

- A hallmark of human intelligence.
  - Text is the largest repository of human knowledge and is growing quickly.
    - emails, news articles, web pages, IM, scientific articles, insurance claims, customer complaint letters, transcripts of phone calls, technical documents, government documents, patent portfolios, court decisions, contracts, .....
    - Are we reading any faster than before?
-

# Why are language technologies needed?

---

- Many companies make a lot of money if they could use computer programmes that understood text or speech.
    - answering the phone, and replying to a question
    - understanding the text on a Web page to decide who it might be of interest to
    - translating a daily newspaper from Japanese to English
    - understanding text in journals / books and building an expert systems based on that understanding
-

# Dreams or reality??

---

- Will my computer talk to me like another human ??
  - Will the search engine get me exactly what I am looking for??
  - Can my PC read the whole newspaper and tell me the important news only..??
  - Can my palmtop translate what that Japanese lady is telling me.. ??
  - Ahhh.. Can my PC do my NLP assignments ??
  - Do you know how our brain processes language ??
-

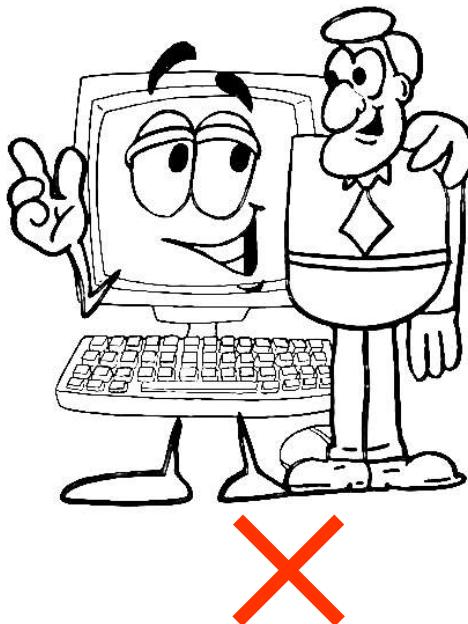
# The Dream

- It'd be great if machines could
  - Process our email (usefully)
  - Translate languages accurately
  - Help us manage, summarize, and aggregate information
  - Talk to us / listen to us
- But they can't:
  - Language is complex, ambiguous, flexible, and subtle
  - Good solutions need linguistics and machine learning knowledge



# The mystery

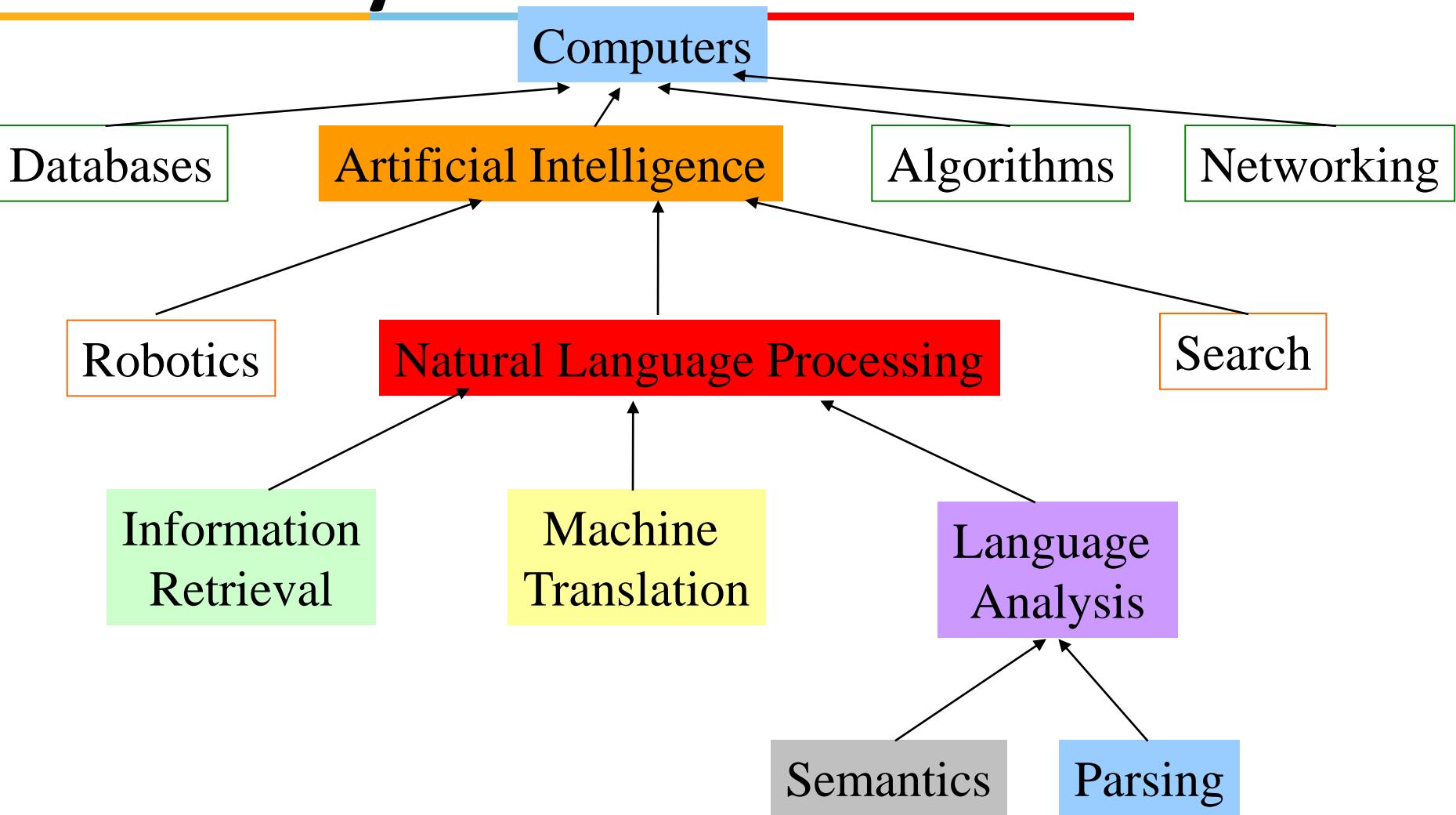
- What's now impossible for computers (and any other species) to do is effortless for humans



# Dreams??



# Where does it fit in the CS taxonomy?



# Brief history of NLP

---

- 1966: Eliza
- 1988: Latent Semantic Analysis patent
- January 2011: IBM Watson beats Jeopardy! champions
- October 2011: Apple Siri launches in beta
- April 2014: Microsoft Cortana demoed
- November 2014: Amazon Alexa
- May 2016: Google Assistant

## 2020 – Conversational Agents

# What We will learn in this Course

---

- Introduction to Natural Language Understanding
  - N-gram Language Models
  - Hidden Markov Models
  - Part-of-Speech Tagging
  - Grammars and Parsing
  - Statistical Constituency Parsing
  - Word sense and word net
  - Dependency Parsing
  - Statistical Machine translation
  - Semantic web ontology
  - Question Answering
  - Dialogue Systems and Chatbots
  - Sentiment analysis
  - Implementation using NLTK
-

# Introduction to Natural Language Processing

---

- The Study of Language.
- Applications of Natural Language Understanding.
- Evaluating Language Understanding Systems.
- The Different Levels of Language Analysis.
- Representations and Understanding.
- The Organization of Natural Language Processing Systems.

---

Dave: Open the pod bay doors, HAL.

HAL: I am sorry, Dave. I am afraid I can't do that.

Dave: What's the problem.

HAL: I think you know what the problem is just as well as I do.

Dave: I don't know what you're talking about.

HAL: I know that you and Frank were planning to disconnect me, and I'm afraid that's something I cannot allow to happen.

General speech and language understanding and generation capabilities

Politeness: emotional intelligence

Self-awareness: a model of self, including goals and plans

Belief ascription: modeling others; reasoning about their goals and plans

---

---

Hal: I can tell from the tone of your voice, Dave, that you're upset.  
Why don't you take a stress pill and get some rest.

[Dave has just drawn another sketch of Dr. Hunter].

HAL: Can you hold it a bit closer?

[Dave does so].

HAL: That's Dr. Hunter, isn't it?

Dave: Yes.

**Recognition of emotion from speech**

**Vision capability including visual recognition of emotions and faces**

**Also: situational ambiguity**

---

To attain the levels of performance we attribute to HAL, we need to be able to define, model, acquire and manipulate

- Knowledge of the world and of agents in it,
- Text meaning,
- Intention

# NLP Applications

---

- Question answering
    - Who is the first Taiwanese president?
  - Text Categorization/Routing
    - e.g., customer e-mails.
  - Text Mining
    - Find everything that can be done with NLP
  - Machine (Assisted) Translation
  - Language Teaching/Learning
    - Usage checking
  - Spelling correction
    - Is that just dictionary lookup?
-

# Application areas

---

- Text-to-Speech & Speech recognition
  - Healthcare
  - Natural Language Dialogue Interfaces to Databases
  - Information Retrieval
  - Information Extraction
  - Document Classification
  - Document Image Analysis
  - Automatic Summarization
  - Text Proofreading – Spelling & Grammar
  - Machine Translation
  - Story understanding systems
  - Plagiarism detection
  - Can u think of anything else ??
-

## Some commercial tools

- [IBM Watson](#) | A pioneer AI platform for businesses
- [Google Cloud NLP API](#) | Google technology applied to NLP
- [Amazon Comprehend](#) | An AWS service to get insights from text

## Open Source Tools

- [Stanford Core NLP](#) is a popular Java library built and maintained by Stanford University.
- **SpaCy** - One of the newest open-source Natural Language Processing with Python libraries
- [Gensim](#) is a highly specialized Python library that largely deals with topic modeling tasks using algorithms like Latent Dirichlet Allocation (LDA)
- [Natural Language Toolkit \(NLTK\)](#) is the most popular Python library

- The Natural Language Toolkit (NLTK) is a platform used for building programs for text analysis.

Open Anaconda terminal, run

**pip install nltk.**

Anaconda and Jupiter are best and popular data science tools

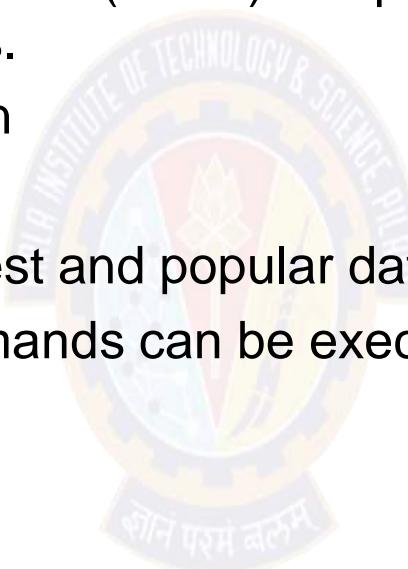
In Jupyter, the console commands can be executed by the ‘!’ sign before the command within the cell.

**! pip install nltk**

[NLTK book](#)

[NLTK discussion forum](#)

<https://www.nltk.org/install.html>



# Why is NLP Big Deal

---

- L = Words + rules + exceptions..
- Ambiguity at all levels..
- We speak different languages..
- And language is a cultural entity..
- So they are not equivalent..
- Highly systematic but also complex..
- Keeps changing.. New words, New rules and New exceptions..
- Source : Electronic texts / Printed texts / Acoustic Speech Signal..  
they are noisy..
- Language looks obvious to us.. But it is a Big Deal ☺ !



# Types of Ambiguities

---

## I. Structural Ambiguities

- Namrata thinks she understands me.
- She thinks Namrata understands me.
- Visiting relatives can be nuisance. (two meanings)

## II. Grammatical Ambiguities

- I (feminine or masculine) go.
- Can- Noun = container, Can – Modal(auxiliary verb),  
Can-verb = to can means to pack etc

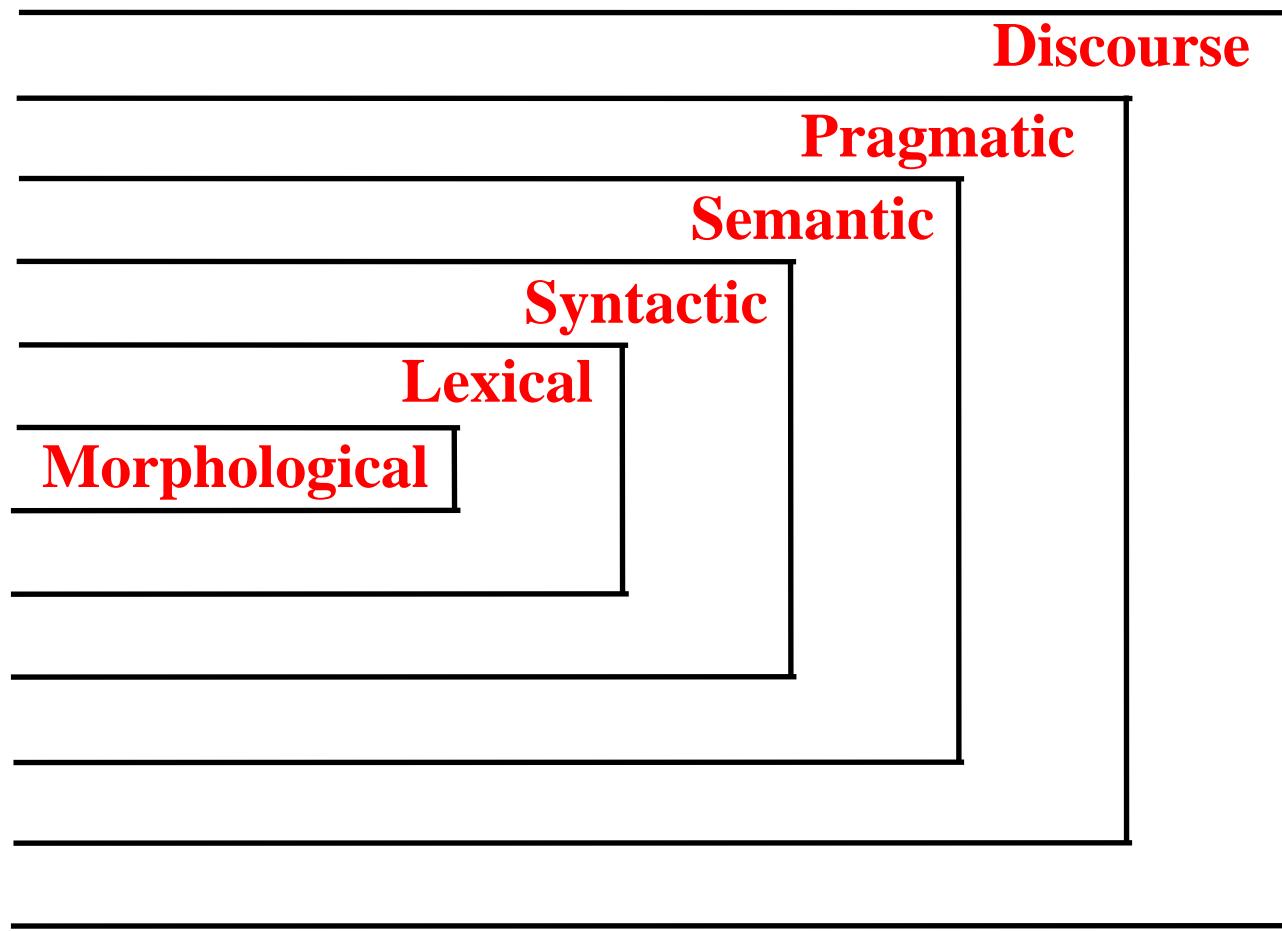
## III. Lexical Ambiguities:

Polysemy Ex: "understand" (I get it)

- Homonymy Ex: Bank= river, financial bank

# Different Levels of Language Analysis

---



# Levels of language understanding

---

**Morphological Knowledge** : Concerns how words are constructed from basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (Ex: “*friendly*” is derived from the meaning of noun “*friend*” and suffix “-ly”, which transforms noun into adjective)

**Lexical Knowledge** : Concerns with listing of words and categorizing them Ex: friendful or beautyship is incorrect lexically. But friendship and beautiful is correct

**Syntactic Knowledge** : Concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subpart of other phrases Ex: “Large have green ideas nose” is lexically correct but syntactically incorrect.

---

# Levels of language understanding

---

**Semantic Knowledge :**Concerns what words mean and how these meanings – combine in sentences to form sentence meanings. This is the study of context-independent meaning. Ex: “Green ideas have large noses” is syntactically correct but semantically incorrect.

**Pragmatic Knowledge :**Concerns how sentences are used in different situations how use affects the interpretation of sentence “She cuts banana with a pen” is semantically correct but pragmatically incorrect as it has no useful meaning.

**Discourse Knowledge :**Concerns how the immediately preceding sentences affect the interpretation of next sentence  
Ex: Chetana is a PhD student at IIT bombay. She is also teacher at Cummins College of Engineering for Women.

# Why is NLP difficult?



Where is Black Panther playing in Mountain View?

Black Panther is playing at the Century 16 Theater.

When is it playing there?

It's playing at 2pm, 5pm, and 8pm.



OK. I'd like 1 adult and 2 children for the first show.

How much would that cost?

Need domain knowledge, discourse knowledge, world knowledge

# Context Free Grammar

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART\ ADJ\ N$
- (3)  $NP \rightarrow ART\ N$
- (4)  $NP \rightarrow ADJ\ N$
- (5)  $VP \rightarrow AUX\ VP$
- (6)  $VP \rightarrow V\ NP$

# Representations and Understanding

Allen 1995: Natural Language Understanding - Introduction

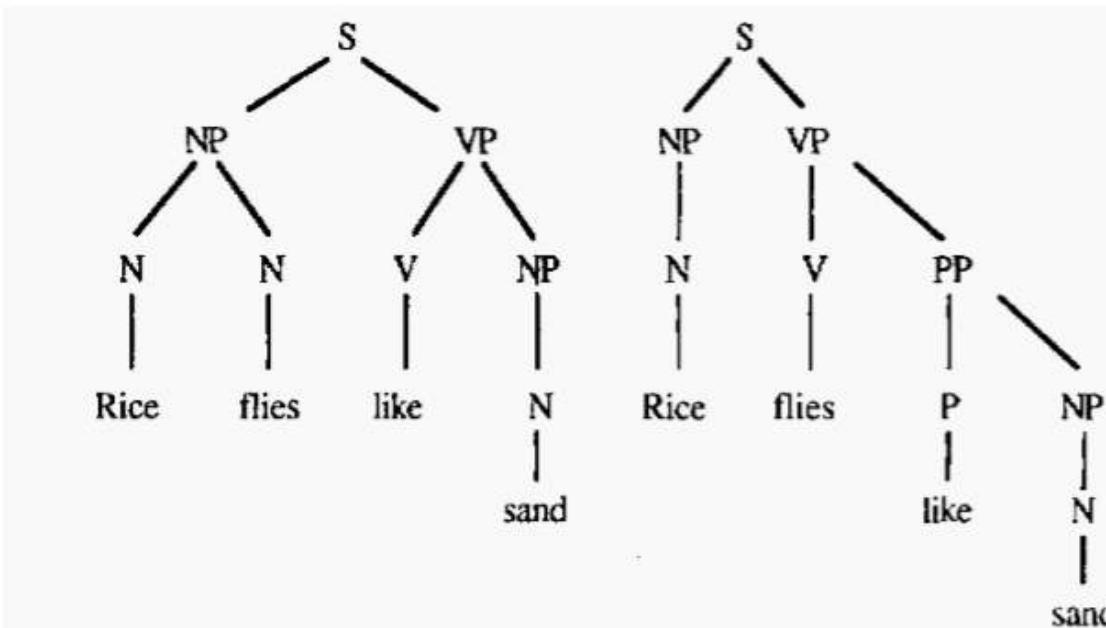


Figure 1.4 Two structural representations of *Rice flies like sand*.

## CFG Rules

$S \rightarrow NP\ VP$   
 $NP \rightarrow N\ N$   
 $NP \rightarrow N$   
 $VP \rightarrow V\ NP$   
 $VP \rightarrow V\ PP$   
 $PP \rightarrow P\ NP$

## Lexicon

Rice : N  
 Flies : N, V  
 Like : V, P  
 Sand : N

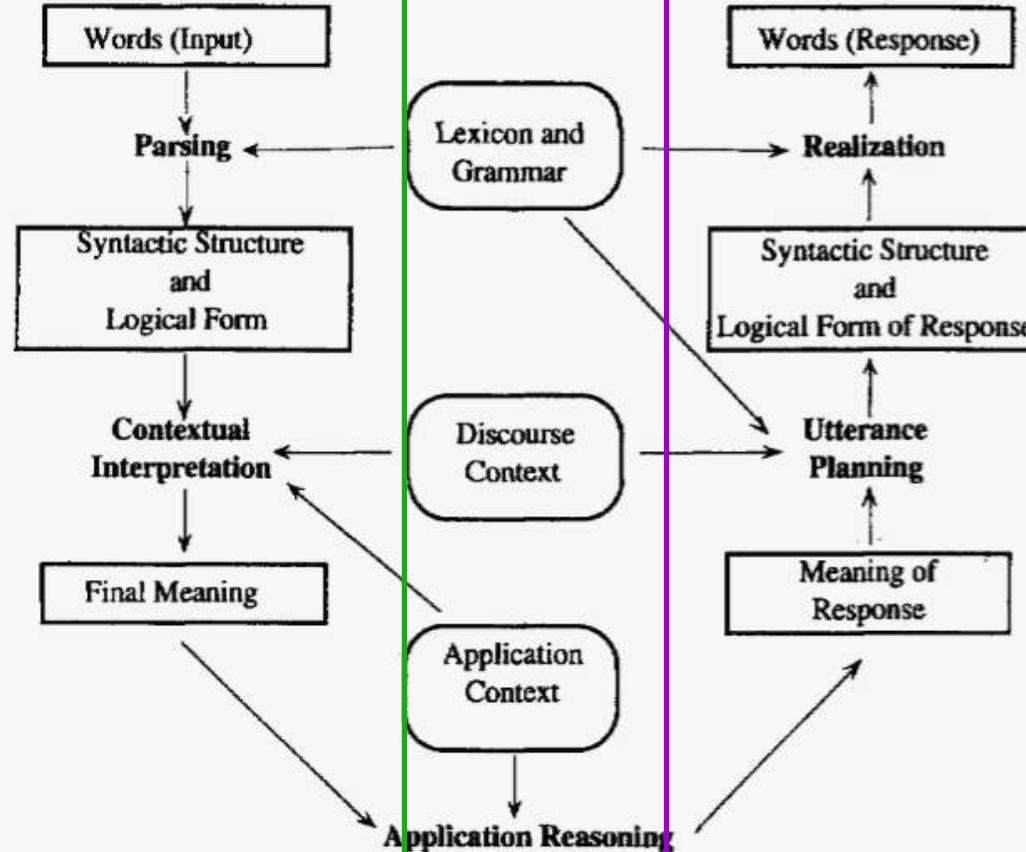
Figure 1.4 Two structural representations of "Rice flies like sand".

# The Organization of Natural Language Processing Systems

Natural Language Understanding

Natural Language Generation

Allen 1995: Natural Language Understanding - Introduction



# Evaluating Language Understanding Systems

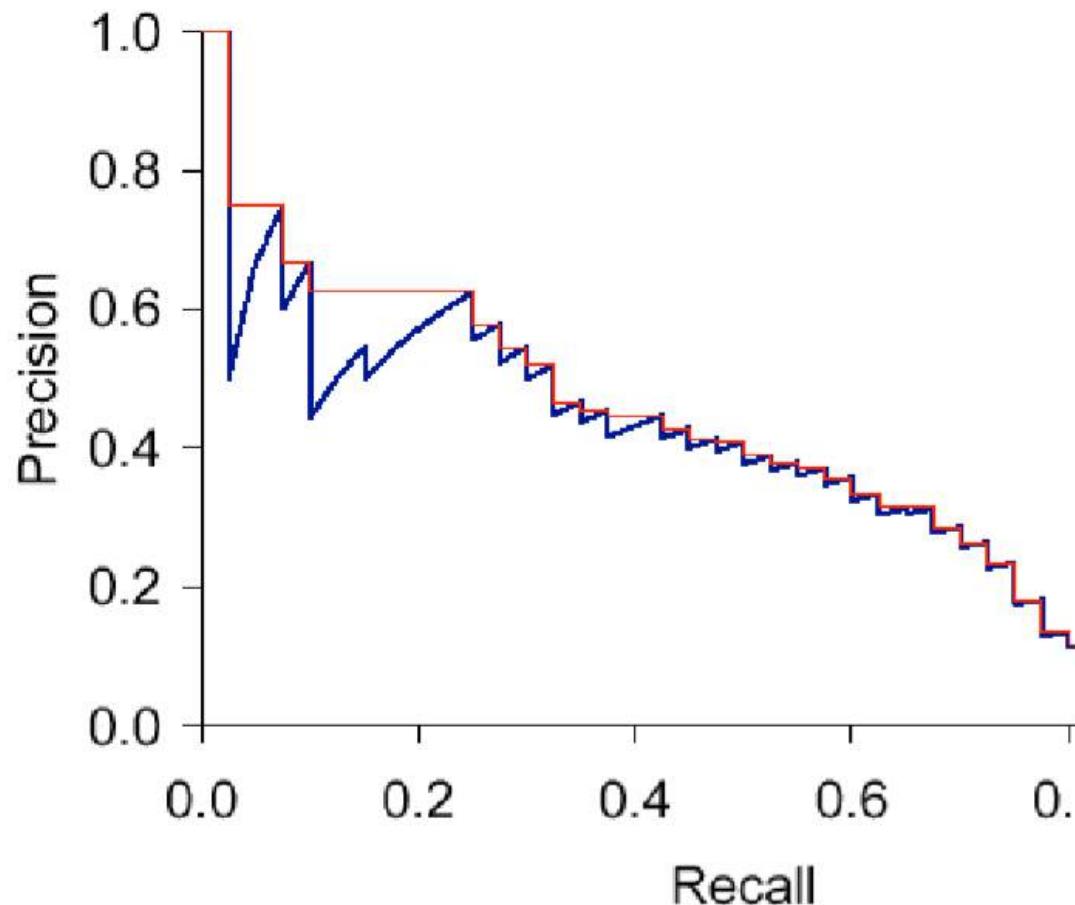
---

- What metrics to use?
  - How to deal with complex outputs like translations?
  - Are the human judgments measuring something real? reliable?
  - Is the sample of texts sufficiently representative?
  - How reliable or certain are the results?
-

# Contingency Table

		<i>gold standard labels</i>		
		gold positive	gold negative	
<i>system output labels</i>	system positive	<b>true positive</b>	<b>false positive</b>	<b>precision</b> = $\frac{tp}{tp+fp}$
	system negative	<b>false negative</b>	<b>true negative</b>	
		<b>recall</b> = $\frac{tp}{tp+fn}$		<b>accuracy</b> = $\frac{tp+tn}{tp+fp+tn+fn}$

# Tradeoff between Precision and Recall



# Some other evaluation measures

---

- **Manual (the best!?):**
  - SSER (subjective sentence error rate)
  - Correct/Incorrect
  - **Adequacy and Fluency** (5 or 7 point scales)
  - Error categorization
  - **Comparative ranking of translations**
- Testing in an application that uses MT as one sub-component
  - E.g., question answering from foreign language documents
    - May not test many aspects of the translation (e.g., cross-lingual IR)



# NLTK Demo

# Pre-processing text data



Cleaning (or pre-processing) the data typically consists of a number of steps:

- **Remove punctuation**
- **Tokenization**
- **Remove stop words**
- Lemmatize/Stem

# Good References

<https://emerj.com/partner-content/nlp-current-applications-and-future-possibilities/>

<https://venturebeat.com/2019/04/05/why-nlp-will-be-big-in-2019/>

<https://www.nltk.org/book/>



Thank you for your time!!



# Natural Language Processing DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 2  
Date – 5<sup>th</sup> September 2020  
Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin and many others who made their course materials freely available online.

# Session Content

---

- Language Models
- N-gram language models
- Evaluation of Language models
- Smoothing
- Interpolation and Back off

# Language modelling



A model that computes either of these:

Probability of a sentence (  $P(W)$  ) or

Probability of an upcoming word (  $P(w_n | w_1, w_2, \dots, w_{n-1})$  )  
is called a **language model**.

Simply we can say that

A language model learns to predict the probability of a sequence of words.

# Example



# Language Models



## 1. Machine Translation:

Machine translation system is used to translate one language to another language

$P(\text{high winds tonight}) > P(\text{large winds tonight})$

## 2.Spell correction



$P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$

### 3.Speech Recognition

---

**Speech recognition** is the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format.

Example:

$P(\text{I saw a van}) >> P(\text{eyes awe of an})$



# How to build a language model



- Recall the definition of conditional probabilities

$$p(B|A) = P(A,B)/P(A) \quad \text{Rewriting: } P(A,B) = P(A)P(B|A)$$

- More variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- The **Chain Rule** in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\dots P(x_n|x_{n-1}, x_{n-2}, \dots, x_1)$$

# The Chain Rule applied to compute joint probability of words in sentence

---

$$P(w_1 w_2 \dots w_n) = \prod_{k=1}^n P(w_k | w_1^{k-1})$$

For ex:

$P(\text{"its water is so transparent"}) =$

$P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water})$

$\times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so})$

# Markov Assumption

- Simplifying assumption:

*limit history of fixed number of words N1*



Andrei Markov

$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{that})$

or

$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{transparent that})$

# Markov Assumption

---

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-k} \dots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-k} \dots w_{i-1})$$

# N-gram Language models

---

- N gram is a sequence of tokens(words)
- Unigram language model(N=1 )

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Example:

$P(\text{I want to eat healthy food}) \approx P(\text{I})P(\text{want})P(\text{to})P(\text{eat})P(\text{healthy})P(\text{food})$

# Bigram model

N=2

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

Example:

$P(\text{I want to eat healthy food}) \approx P(\text{I} | \text{<start>}) P(\text{want} | \text{I}) P(\text{to} | \text{want})$   
 $P(\text{eat} | \text{to}) P(\text{healthy} | \text{eat}) P(\text{food} | \text{ healthy})$   
 $P(\text{<end>} | \text{food})$

# N-gram models

---

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
  - because language has **long-distance dependencies**:  
“The computer(s) which I had just put into the machine room  
on the fifth floor is (are) crashing.”

# Estimating bigram probabilities

---

- The Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

# An example

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{/s} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

More examples:

## Berkeley Restaurant Project sentences

---

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

# Raw bigram counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

# Raw bigram probabilities

- Normalize by unigrams:

- Result:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

# Bigram estimates of sentence probabilities

---

$$\begin{aligned} P(< \text{s} > \text{ I want english food } < / \text{s} >) &= \\ P(\text{I} | < \text{s} >) & \\ \times P(\text{want} | \text{I}) & \\ \times P(\text{english} | \text{want}) & \\ \times P(\text{food} | \text{english}) & \\ \times P(< / \text{s} > | \text{food}) & \\ = .000031 & \end{aligned}$$

# What kinds of knowledge?

---

- $P(\text{english} | \text{want}) = .0011$
  - $P(\text{chinese} | \text{want}) = .0065$
  - $P(\text{to} | \text{want}) = .66$
  - $P(\text{eat} | \text{to}) = .28$
  - $P(\text{food} | \text{to}) = 0$
  - $P(\text{want} | \text{spend}) = 0$
  - $P(\text{i} | \langle s \rangle) = .25$
-

# Practical Issues

---

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

$$\log(p_1 \cdot p_2 \cdot p_3 \cdot p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

# Google N-Gram Release

AUG

3

## All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

# Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

<http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html>

<https://pypi.org/project/google-ngram-downloader/>

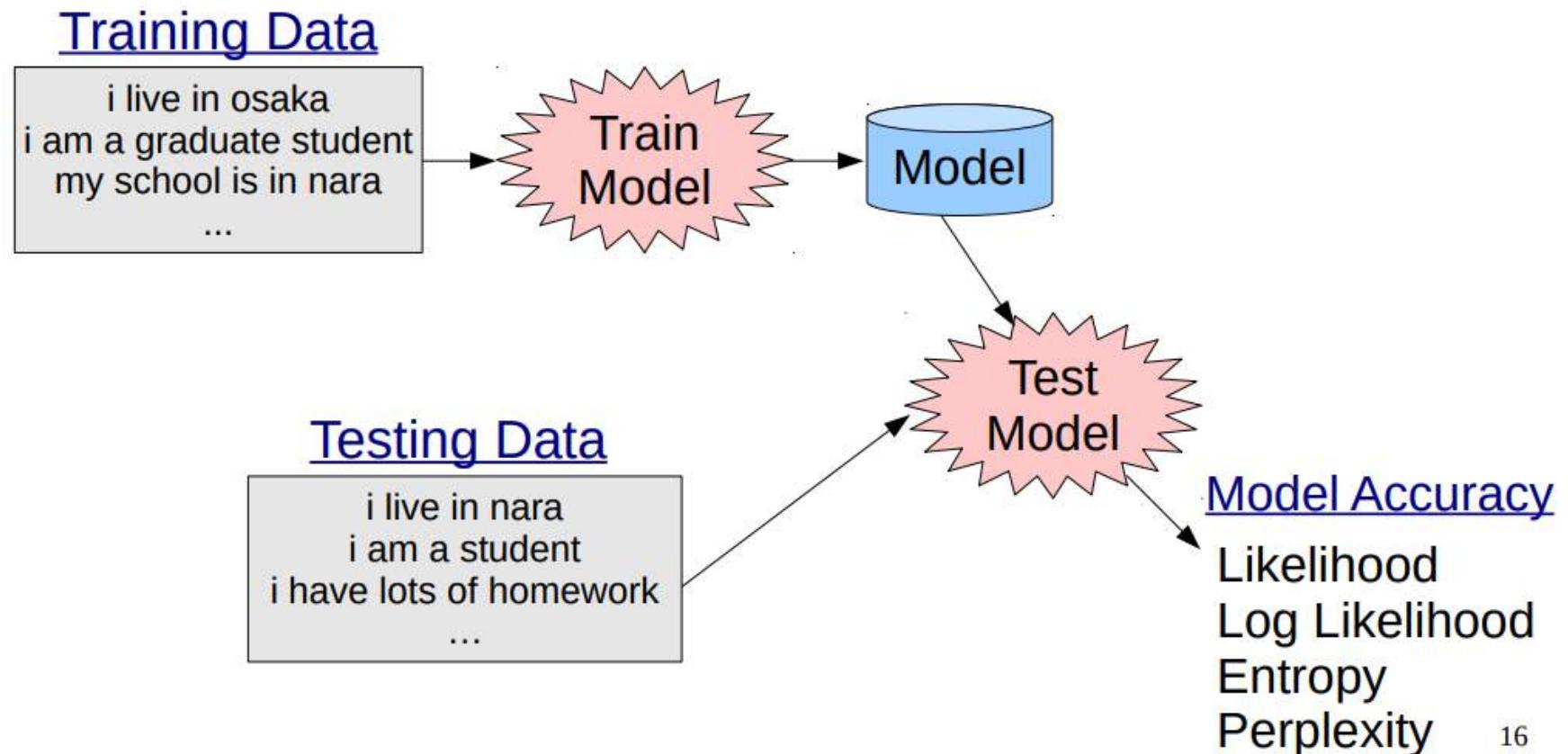
# Evaluation: How good is our model?

---



- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

# Experimental setup



# Extrinsic evaluation of N-gram models

---



- Best evaluation for comparing models A and B
    - Put each model in a task
      - spelling corrector, speech recognizer, MT system
    - Run the task, get an accuracy for A and for B
      - How many misspelled words corrected properly
      - How many words translated correctly
    - Compare accuracy for A and B
-

# Difficulty of extrinsic (in-vivo) evaluation of N-gram models

---

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
  - But is helpful to think about.

# Intuition of Perplexity



- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and \_\_\_\_\_

The 33<sup>rd</sup> President of the US was \_\_\_\_\_

I saw a \_\_\_\_\_

{ mushrooms 0.1  
pepperoni 0.1  
anchovies 0.01  
....  
fried rice 0.0001  
....  
and 1e-100

- Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

# Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest  $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

**Minimizing perplexity is the same as maximizing probability**

# Perplexity as branching factor

---

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign  $P=1/10$  to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

# Lower perplexity = better model

---

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

# The perils of overfitting

---

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
      - But occur in the test set

# Zeros

---

- Training set:
  - ... denied the allegations
  - ... denied the reports
  - ... denied the claims
  - ... denied the request
- Test set
  - ... denied the offer
  - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

# Zero probability bigrams

---

- Bigrams with zero probability
    - mean that we will assign 0 probability to the test set!
  - And hence we cannot compute perplexity (can't divide by 0)!
-

# Laplace Smoothing (Add 1 smoothing)

---

- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:
- Add-1 estimate:

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

# Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is  $400/1,000,000 = .0004$
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

# Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

# Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	<b>0.00025</b>	0.0025	<b>0.00025</b>	<b>0.00025</b>	<b>0.00025</b>	0.00075
want	0.0013	<b>0.00042</b>	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	<b>0.00026</b>	0.0013	0.18	0.00078	<b>0.00026</b>	0.0018	0.055
eat	<b>0.00046</b>	0.00046	0.0014	<b>0.00046</b>	0.0078	0.0014	0.02	<b>0.00046</b>
chinese	0.0012	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	0.052	0.0012	0.00062
food	0.0063	<b>0.00039</b>	0.0063	<b>0.00039</b>	0.00079	0.002	<b>0.00039</b>	0.00039
lunch	0.0017	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	0.0011	<b>0.00056</b>	<b>0.00056</b>
spend	0.0012	<b>0.00058</b>	0.0012	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>

# Reconstituted counts



$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Add-1 estimation is a blunt instrument

---



- So add-1 isn't used for N-grams:
    - We'll see better methods
  - But add-1 is used to smooth other NLP models
    - For text classification
    - In domains where the number of zeros isn't so huge.
-

# Backoff and Interpolation

---

- Sometimes it helps to use **less** context
    - Condition on less context for contexts you haven't learned much about
  - **Backoff:**
    - use trigram if you have good evidence,
    - otherwise bigram, otherwise unigram
  - **Interpolation:**
    - mix unigram, bigram, trigram
  - Interpolation works better
-

# Linear Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2 P(w_n | w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}\qquad\qquad\qquad \sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2(w_{n-2}^{n-1}) P(w_n | w_{n-1}) \\ &\quad + \lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

# How to set the lambdas?

- Use a **held-out** corpus

Training Data

Held-Out  
Data

Test  
Data

- Choose  $\lambda$ s to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for  $\lambda$ s that give largest probability to held-out set:

$$\log P(w_1 \dots w_n | M(/_1 \dots /_k)) = \sum_i \log P_{M(/_1 \dots /_k)}(w_i | w_{i-1})$$

# Unknown words: Open versus closed vocabulary tasks

---

- If we know all the words in advanced
  - Vocabulary  $V$  is fixed
  - Closed vocabulary task
- Often we don't know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon  $L$  of size  $V$
    - At text normalization phase, any training word not in  $L$  changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

# Huge web-scale n-grams

---

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
    - Remove singletons of higher-order n-grams
  - Entropy-based pruning
- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)

# Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants *et al.* 2007)
- No discounting, just use relative frequencies

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

# References

---

<https://www.coursera.org/learn/language-processing/home/welcome>

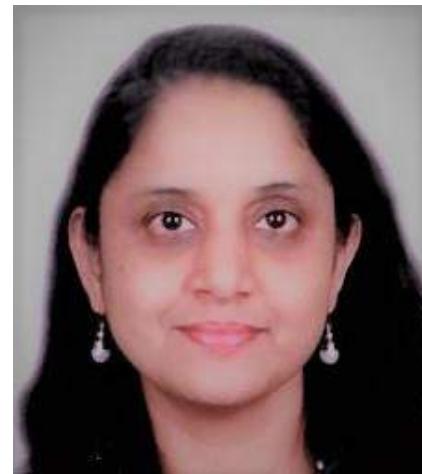
<https://books.google.com/ngrams>

<http://www.speech.sri.com/projects/srilm/>

[https://www.youtube.com/watch?time\\_continue=283&v=Saq1QagC8KY&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=283&v=Saq1QagC8KY&feature=emb_logo)

<https://www.youtube.com/watch?v=paCMAZ-lKq8>

<https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html>



Thank you for your time!!



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 3-Part-of-Speech Tagging**  
**Date – 12<sup>th</sup> September 2020**  
**Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin and many others who made their course materials freely available online.

# Session Content

---

- (Mostly) English Word Classes
  - The Penn Treebank Part-of-Speech Tag set
  - Part-of-Speech Tagging
  - Markov Chains
  - Hidden Markov Model
  - HMM Part-of-Speech Tagging
  - Part-of-Speech Tagging for Morphological Rich Languages
-

# Recap: Ambiguities in language

## I. Structural Ambiguities

- Namrata thinks she understands me.
- She thinks Namrata understands me.
- Visiting relatives can be nuisance. (two meanings)

## II. Grammatical Ambiguities

- I (feminine or masculine) go.
- Can- Noun = container, Can – Modal(auxiliary verb),  
Can-verb = to can means to pack etc

## III. Lexical Ambiguities:

Polysemy Ex: "understand" (I get it)

- Homonymy Ex: Bank= river, financial bank

Don't worry! There is no problem  
with your eyes or computer.

# Let's try

ଓ/DT এৰেৱা/NN ০ ১০/VBZ কোৱা/VBG ও/DT  
কোৱা/NN  .

ଓ/DT এৰে ৪/NN ০ ১০/VBZ ৯ ১ ৫ ৫ ০ ৫/VBG  .

ଓ/DT কুৰি ৫/NN ০ ১০/VBZ ১০ ০ ৫/VBG ০ ৫/VBG  .

ଓ/DT একা ৭ ৭ ৫/JJ কুৰি ০ ৯/NN

What is the POS tag sequence of the following sentence?

ଓ একা ৭ ৭ ৫ কুৰি ৩ ও ১০ ১০ ০ ৫/VBG ০ ৫/VBG 

# Let's try

- ഓ/DT ഓ⑥എ/NN ①⑩/VBZ കുഞ്ഞു ⑩①⑤എ/VBG ഓ/DT കുങ്ഞു①/NN
    1. a/DT dog/NN is/VBZ chasing/VBG a/DT cat/NN ./.  
2. ഓ/DT ഫുഡ് ④/NN ①⑩/VBZ ⑨①⑤⑤①⑤എ/VBG പേജ്/.  
3. ഓ/DT ബുഡ് ⑤/NN ①⑩/VBZ ⑩①⑤എ ①⑤എ/VBG പേജ്/.  
4. ഓ/DT ഫുഡ് ⑦ ⑦ ⑤/JJ ബുഡ് ① ⑨എ/NN  
5. ഓ ഫുഡ് ⑦ ⑦ ⑤ കുങ്ഞു① ③കുഞ്ഞു⑩ ⑩①⑤എ ①⑤എ പേജ്  
6. a happy cat was singing .

# POS Tagging

- The process of assigning a part-of-speech or lexical class marker to each word in a sentence (and all sentences in a collection).

**Input:** the lead paint is unsafe

**Output:** the/Det lead/N paint/N is/V unsafe/Adj

# Why is POS Tagging Useful?

- First step of a vast number of practical tasks
- Helps in stemming/lemmatization
- Parsing
  - Need to know if a word is an N or V before you can parse
  - Parsers can build trees directly on the POS tags instead of maintaining a lexicon
- Information Extraction
  - Finding names, relations, etc.
- Machine Translation
- Selecting words of specific Parts of Speech (e.g. nouns) in pre-processing documents (for IR etc.)

# Parts of Speech

- 8 (ish) traditional parts of speech
  - Noun, verb, adjective, preposition, adverb, article, interjection, pronoun, conjunction, etc
  - Called: parts-of-speech, lexical categories, word classes, morphological classes, lexical tags...
  - Lots of debate within linguistics about the number, nature, and universality of these
    - We'll completely ignore this debate.

# POS examples

- N noun *chair, bandwidth, pacing*
- V verb *study, debate, munch*
- ADJ adjective *purple, tall, ridiculous*
- ADV adverb *unfortunately, slowly*
- P preposition *of, by, to*
- PRO pronoun *I, me, mine*
- DET determiner *the, a, that, those*

# POS Tagging

- The process of assigning a part-of-speech or lexical class marker to each word in a collection.

<u>WORD</u>	<u>tag</u>
<b>the</b>	<b>DET</b>
<b>koala</b>	<b>N</b>
<b>put</b>	<b>V</b>
<b>the</b>	<b>DET</b>
<b>keys</b>	<b>N</b>
<b>on</b>	<b>P</b>
<b>the</b>	<b>DET</b>
<b>table</b>	<b>N</b>

# Open and Closed Classes

- Closed class: a small fixed membership
  - Prepositions: of, in, by, ...
  - Auxiliaries: may, can, will had, been, ...
  - Pronouns: I, you, she, mine, his, them, ...
  - Usually **function words** (short common words which play a role in grammar)
- Open class: new ones can be created all the time
  - English has 4: Nouns, Verbs, Adjectives, Adverbs
  - Many languages have these 4, but not all!

# Open Class Words

- Nouns
  - Proper nouns (Boulder, Granby, Eli Manning)
    - English capitalizes these.
  - Common nouns (the rest).
  - Count nouns and mass nouns
    - Count: have plurals, get counted: goat/goats, one goat, two goats
    - Mass: don't get counted (snow, salt, communism) (\*two snows)
- Adverbs: tend to modify things
  - **Unfortunately**, John walked home **extremely slowly yesterday**
  - Directional/locative adverbs (here, home, downhill)
  - Degree adverbs (extremely, very, somewhat)
  - Manner adverbs (slowly, slinkily, delicately)
- Verbs
  - In English, have morphological affixes (eat/eats/eaten)

# Closed Class Words

Examples:

- prepositions: *on, under, over, ...*
- particles: *up, down, on, off, ...*
- determiners: *a, an, the, ...*
- pronouns: *she, who, I, ..*
- conjunctions: *and, but, or, ...*
- auxiliary verbs: *can, may should, ...*
- numerals: *one, two, three, third, ...*

# Prepositions from CELEX

of	540,085	through	14,964	worth	1,563	pace	12
in	331,235	after	13,670	toward	1,390	nigh	9
for	142,421	between	13,275	plus	750	re	4
to	125,691	under	9,525	till	686	mid	3
with	124,965	per	6,515	amongst	525	o'er	2
on	109,129	among	5,090	via	351	but	0
at	100,169	within	5,030	amid	222	ere	0
by	77,794	towards	4,700	underneath	164	less	0
from	74,843	above	3,056	versus	113	midst	0
about	38,428	near	2,026	amidst	67	o'	0
than	20,210	off	1,695	sans	20	thru	0
over	18,071	past	1,575	circa	14	vice	0

# English Particles

aboard	aside	besides	forward(s)	opposite	through
about	astray	between	home	out	throughout
above	away	beyond	in	outside	together
across	back	by	inside	over	under
ahead	before	close	instead	overhead	underneath
alongside	behind	down	near	past	up
apart	below	east, etc.	off	round	within
around	beneath	eastward(s),etc.	on	since	without

# Conjunctions

and	514,946	yet	5,040	considering	174	forasmuch as	0
that	134,773	since	4,843	lest	131	however	0
but	96,889	where	3,952	albeit	104	immediately	0
or	76,563	nor	3,078	providing	96	in as far as	0
as	54,608	once	2,826	whereupon	85	in so far as	0
if	53,917	unless	2,205	seeing	63	inasmuch as	0
when	37,975	why	1,333	directly	26	insomuch as	0
because	23,626	now	1,290	ere	12	insomuch that	0
so	12,933	neither	1,120	notwithstanding	3	like	0
before	10,720	whenever	913	according as	0	neither nor	0
though	10,329	whereas	867	as if	0	now that	0
than	9,511	except	864	as long as	0	only	0
while	8,144	till	686	as though	0	provided that	0
after	7,042	provided	594	both and	0	providing that	0
whether	5,978	whilst	351	but that	0	seeing as	0
for	5,935	suppose	281	but then	0	seeing as how	0
although	5,424	cos	188	but then again	0	seeing that	0
until	5,072	supposing	185	either or	0	without	0

# POS Tagging

## Choosing a Tagset

- There are so many parts of speech, potential distinctions we can draw
- To do POS tagging, we need to choose a standard set of tags to work with
- Could pick very coarse tagsets
  - N, V, Adj, Adv.
- More commonly used set is finer grained, the “Penn TreeBank tagset”, 45 tags
  - PRP\$, WRB, WP\$, VBG
- Even more fine-grained tagsets exist

# Penn TreeBank POS Tagset

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	+,%,&
CD	cardinal number	<i>one, two, three</i>	TO	“to”	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential ‘there’	<i>there</i>	VB	verb, base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb, past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb, gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VBN	verb, past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb, non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb, 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WP\$	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, singular	<i>IBM</i>	\$	dollar sign	\$
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	#
PDT	predeterminer	<i>all, both</i>	“	left quote	‘ or “
POS	possessive ending	<i>'s</i>	”	right quote	’ or ”
PRP	personal pronoun	<i>I, you, he</i>	(	left parenthesis	[, (, {, <
PRP\$	possessive pronoun	<i>your, one's</i>	)	right parenthesis	], ), }, >
RB	adverb	<i>quickly, never</i>	,	comma	,
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	. ! ?
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	: ; ... --
RP	particle	<i>up, off</i>			

# Using the Penn Tagset

- The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
- Prepositions and subordinating conjunctions marked IN (“although/IN I/PRP..”)
- Except the preposition “to” is just marked “TO”.

# POS Tagging

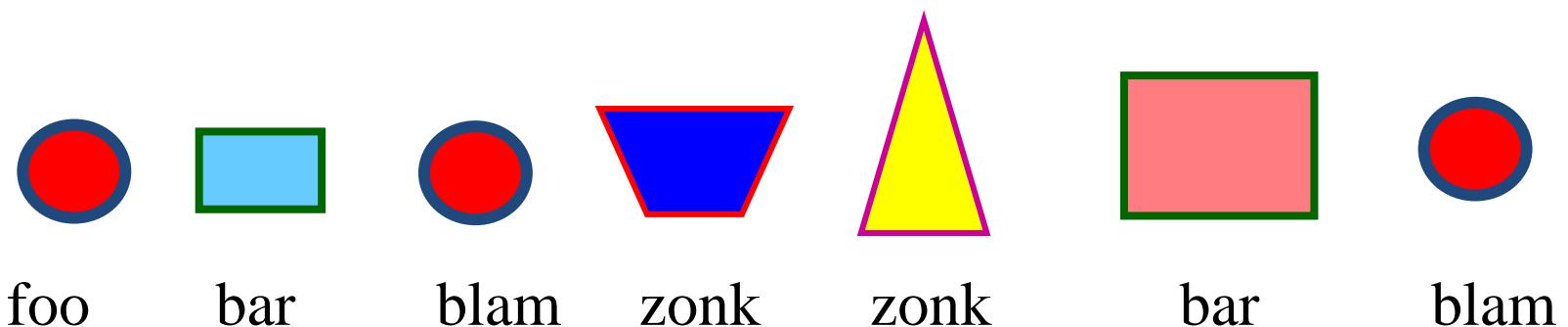
- Words often have more than one POS:  
*back*
  - The **back** door = JJ
  - On my **back** = NN
  - Win the voters **back** = RB
  - Promised to **back** the bill = VB
- The POS tagging problem is to determine the POS tag for a particular instance of a word.

# POS Tagging as Sequence Classification

- We are given a sentence (an “observation” or “sequence of observations”)
  - *Secretariat is expected to race tomorrow*
- What is the best sequence of tags that corresponds to this sequence of observations?
- Probabilistic view
  - Consider all possible sequences of tags
  - Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of  $n$  words  $w_1 \dots w_n$ .

# Sequence Labeling Problem

- Many NLP problems can be viewed as sequence labeling.
- Each token in a sequence is assigned a label.
- Labels of tokens are dependent on the labels of other tokens in the sequence, particularly their neighbors (not i.i.d).



# Information Extraction

- Identify phrases in language that refer to specific types of entities and relations in text.
- Named entity recognition is task of identifying names of people, places, organizations, etc. in text.  
**people    organizations    places**
  - Michael Dell is the CEO of Dell Computer Corporation and lives in Austin Texas.
- Extract pieces of information relevant to a specific application, e.g. used car ads:  
**make    model    year    mileage    price**
  - For sale, 2002 Toyota Prius, 20,000 mi, \$15K or best offer. Available starting July 30, 2006.

# Semantic Role Labeling

- For each clause, determine the semantic role played by each noun phrase that is an argument to the verb.

agent patient source destination  
instrument

- John drove Mary from Austin to Dallas in his Toyota Prius.
- The hammer broke the window.
- Also referred to a “case role analysis,” “thematic analysis,” and “shallow semantic parsing”

# Bioinformatics

- Sequence labeling also valuable in labeling genetic sequences in genome analysis.  
**extron    intron**  
– AGCTAACGTTCGATACGGATTACAGCCT

# Problems with Sequence Labeling as Classification

- Not easy to integrate information from category of tokens on both sides.
- Difficult to propagate uncertainty between decisions and “collectively” determine the most likely joint assignment of categories to all of the tokens in a sequence.

# Probabilistic Sequence Models

- Probabilistic sequence models allow integrating uncertainty over multiple, interdependent classifications and collectively determine the most likely global assignment.
- standard model
  - Hidden Markov Model (HMM)

# Hidden Markov Models

- It is a **sequence model**.
- Assigns a label or class to each unit in a sequence, thus mapping a **sequence of observations** to a **sequence of labels**.
- Probabilistic sequence model: given a sequence of units (e.g. words, letters, morphemes, sentences), compute a probability distribution over possible sequences of labels and choose the best label sequence.
- This is a kind of *generative* model.

# Markov Model / Markov Chain

- A finite state machine with probabilistic state transitions.
- Makes Markov assumption that next state only depends on the current state and independent of previous history.

# Hidden Markov Models

- *Generative* model.
  - There is a **hidden** underlying generator of observable events
  - The hidden generator can be modeled as a network of states and transitions
  - We want to infer the underlying state sequence given the observed event sequence

# Hidden Markov Models (Formal)

- States  $Q = q_1, q_2 \dots q_N$ ;
- Observations  $O = o_1, o_2 \dots o_N$ ;
  - Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots v_V\}$
- Transition probabilities
  - Transition probability matrix  $A = \{a_{ij}\}$ 
$$a_{ij} = P(q_t = j | q_{t-1} = i) \quad 1 \leq i, j \leq N$$
- Observation likelihoods
  - Output probability matrix  $B = \{b_i(k)\}$ 
$$b_i(k) = P(X_t = o_k | q_t = i)$$
- Special initial probability vector  $\pi$ 
$$\rho_i = P(q_1 = i) \quad 1 \leq i \leq N$$

# Markov Chain: “First-order observable Markov Model”

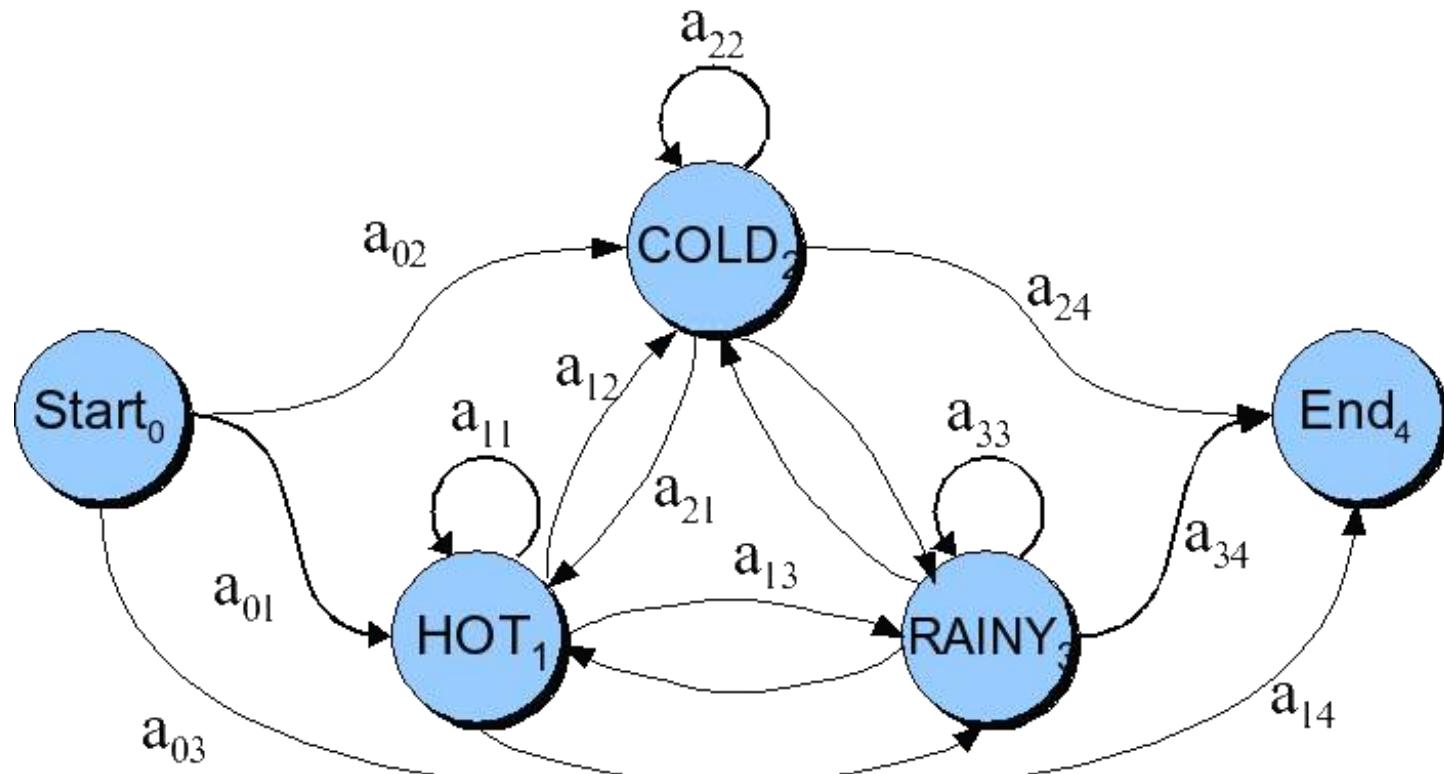
- A set of states
  - $Q = q_1, q_2 \dots q_N$ ; the state at time  $t$  is  $q_t$
- Transition probabilities:
  - a set of probabilities  $A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$ .
  - Each  $a_{ij}$  represents the probability of transitioning from state  $i$  to state  $j$
  - The set of these is the transition probability matrix  $A$
  - Special **initial** probability vector  $\pi$
- Current state only depends on previous state

$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

# How to build a second-order HMM?

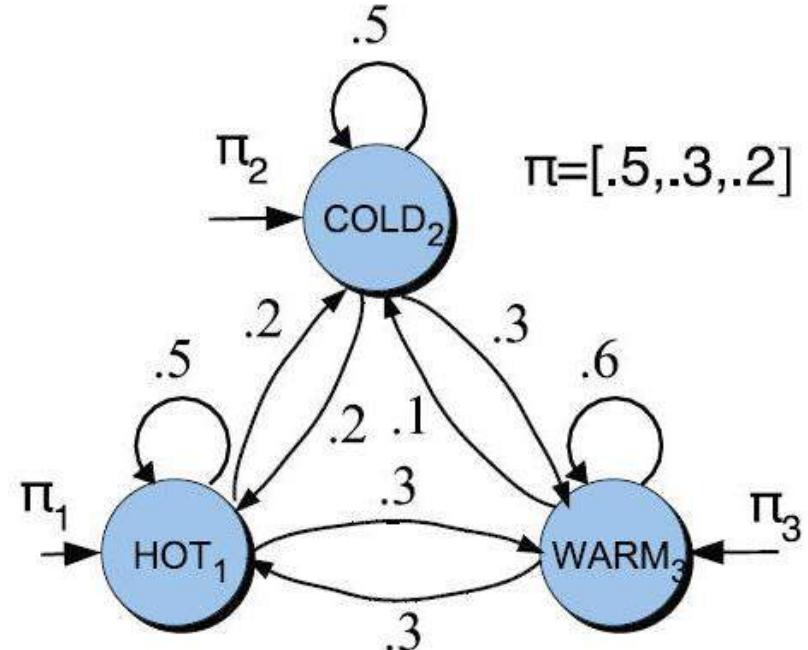
- Second-order HMM
  - Current state only depends on previous 2 states
- Example
  - Trigram model over POS tags
  - $P(\mathbf{t}) = \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})$
  - $P(\mathbf{w}, \mathbf{t}) = \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})P(w_i | t_i)$

# Markov Chain for Weather



# Markov Chain for Weather

- What is the probability of 4 consecutive warm days?
- Sequence is  
warm-warm-warm-warm
- And state sequence is  
3-3-3-3
- $P(3,3,3,3) =$   
 $- \pi_3 a_{33} a_{33} a_{33} a_{33} = 0.2 \times (0.6)^3 = 0.0432$

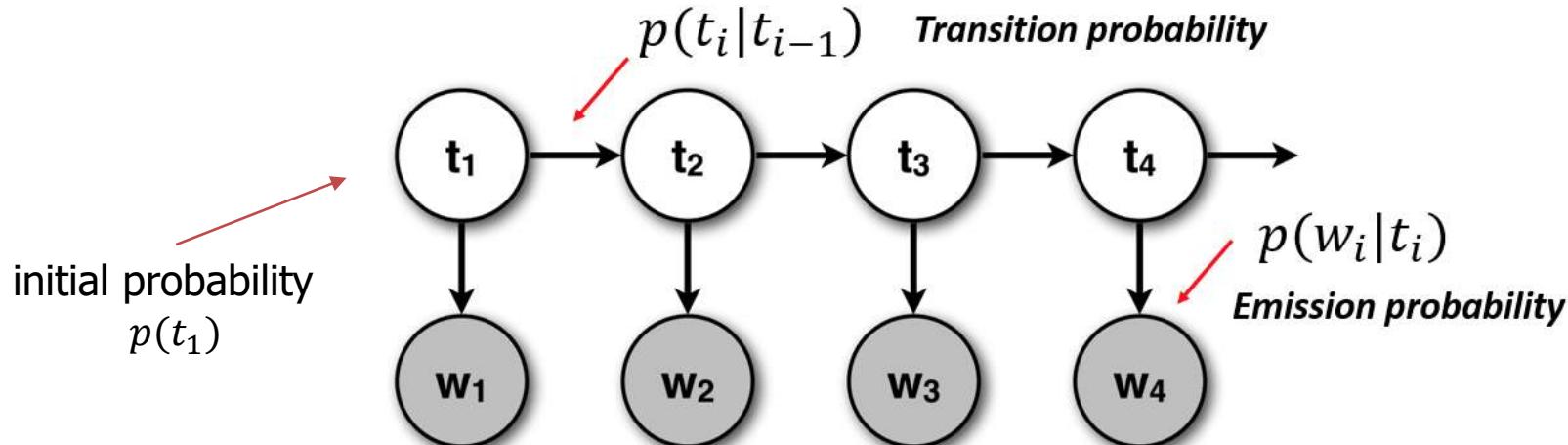


# HMM to predict the tags

- Two types of information are useful
  - Relations between words and tags
  - Relations between tags and tags
    - DT NN, DT JJ NN...

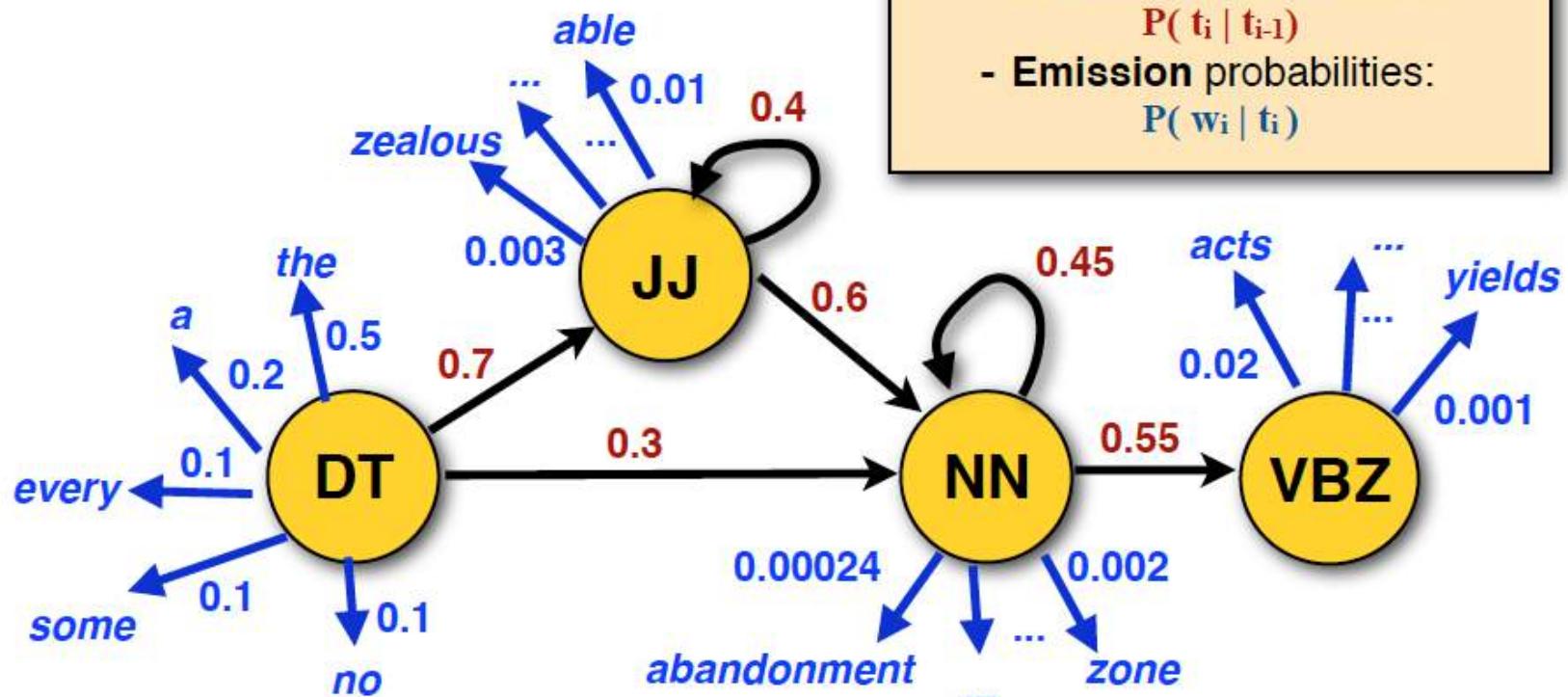
# Prediction in generative model

- Inference: What is the most likely sequence of tags for the given sequence of words  $w$



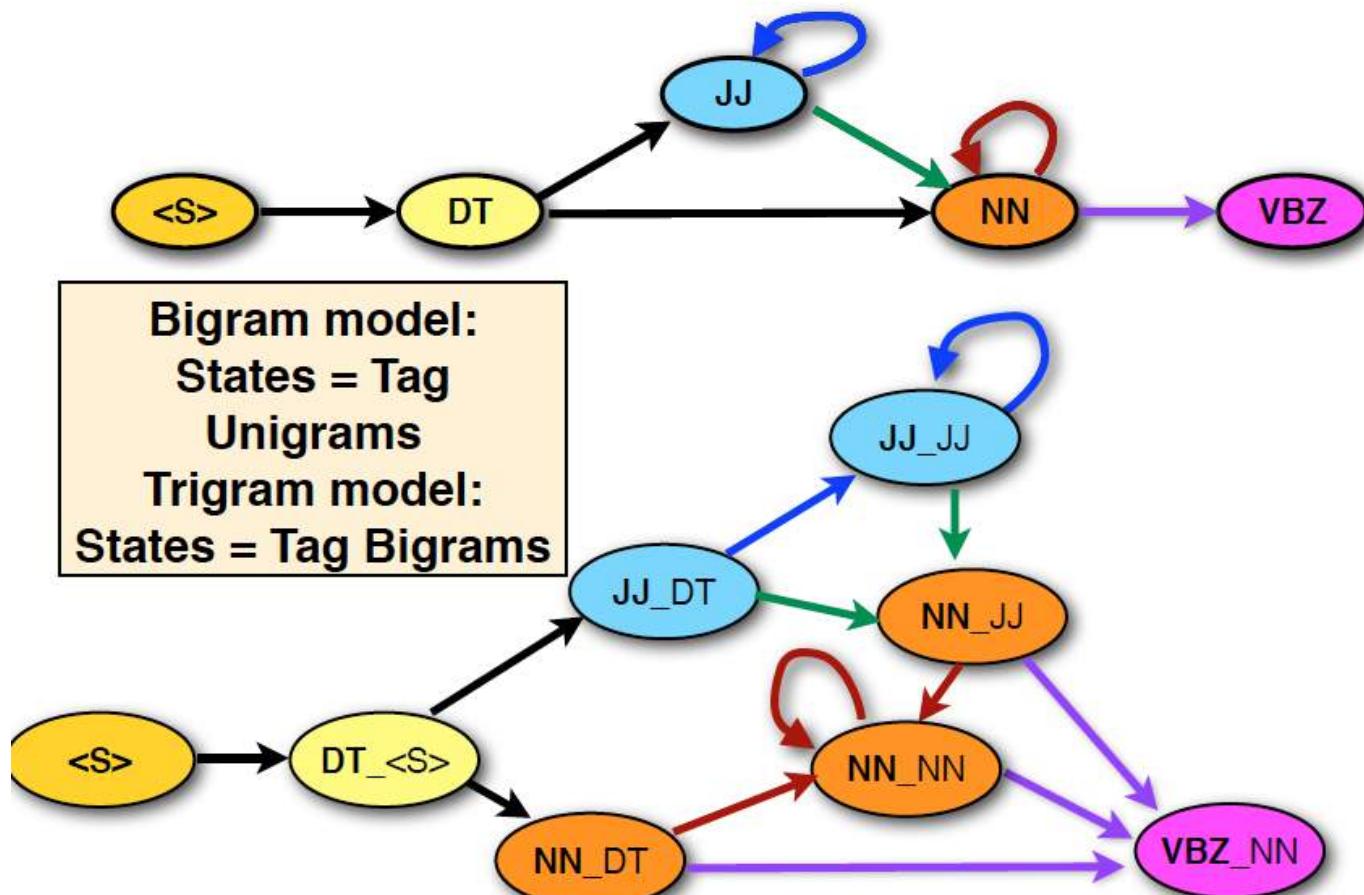
- What are the latent states that most likely generate the sequence of word  $w$

# HMMs as probabilistic FSA

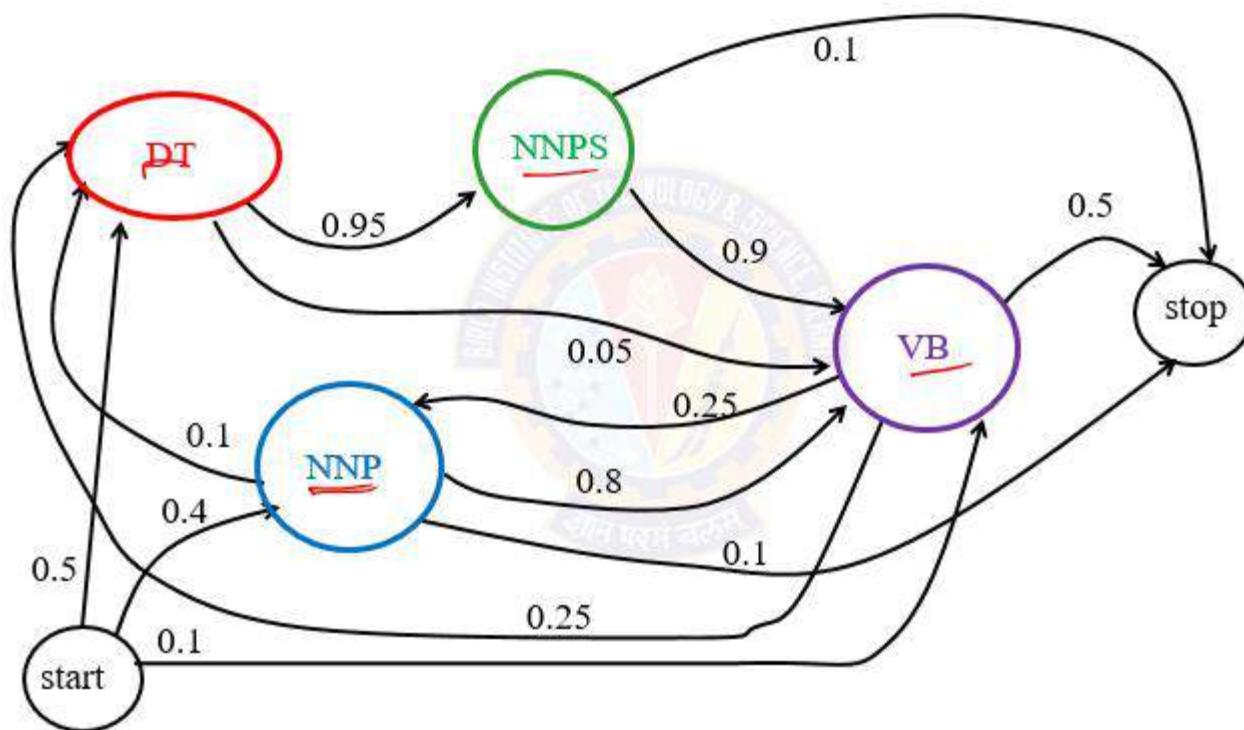


Julia Hockenmaier: Intro to NLP

# Probabilistic FSA for second-order HMM



Julia Hockenmaier: Intro to NLP



# Statistical POS Tagging

- We want, out of all sequences of n tags  $t_1 \dots t_n$  the single tag sequence such that

$P(t_1 \dots t_n | w_1 \dots w_n)$  is highest.

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

- Hat ^ means “our estimate of the best one”
- Argmax<sub>x</sub> f(x) means “the x such that f(x) is maximized”

# Statistical POS Tagging

- This equation should give us the best tag sequence

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

- But how to make it operational? How to compute this value?
- Intuition of Bayesian inference:
  - Use Bayes rule to transform this equation into a set of probabilities that are easier to compute (and give the right answer)

# Using Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{P(w_1^n | t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n | t_1^n)P(t_1^n)$$

# Likelihood and Prior



$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{\text{likelihood}}{P(w_1^n | t_1^n)} \frac{\text{prior}}{P(t_1^n)}$$

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i | t_i)$$



$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1})$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) \approx \operatorname{argmax}_{t_1^n} \prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1})$$

# Hidden Markov Models (POS Tagging)

- States  $T = t_1, t_2 \dots t_N$ ;
- Observations  $W = w_1, w_2 \dots w_N$ ;
  - Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots v_V\}$
- Transition probabilities
  - Transition probability matrix  $A = \{a_{ij}\}$   
 $a_{ij} = P(t_i = j \mid t_{i-1} = i) \quad 1 \leq i, j \leq N$
- Observation likelihoods
  - Output probability matrix  $B = \{b_i(k)\}$   
 $b_i(k) = P(w_i = v_k \mid t_i = i)$
- Special initial probability vector  $\pi$   
 $\pi_i = P(t_1 = i) \quad 1 \leq i \leq N$

# Two Kinds of Probabilities

## 1. State transition probabilities -- $p(t_i|t_{i-1})$

- State-to-state transition probabilities

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

## 2. Observation/Emission probabilities --

$$p(w_i|t_i)$$

- Probabilities of observing various values at a given state

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

# Two Kinds of Probabilities

## 1. Tag transition probabilities -- $p(t_i|t_{i-1})$

- Determiners likely to precede adjs and nouns
  - That/DT flight/NN
  - The/DT yellow/JJ hat/NN
  - So we expect  $P(NN|DT)$  and  $P(JJ|DT)$  to be high
- Compute  $P(NN|DT)$  by counting in a labeled corpus:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

# Two Kinds of Probabilities

## 2. Word likelihood/emission probabilities

$$p(w_i|t_i)$$

- VBZ (3sg Pres Verb) likely to be “is”
- Compute  $P(is|VBZ)$  by counting in a labeled corpus:

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

# Sample Probabilities

## *Bigram Probabilities*

Bigram(Ti, Tj)	Count(i, i + 1)	Prob(Tj Ti)
φ,ART	213	.71 (213/300)
φ,N	87	.29 (87/300)
φ,V	10	.03 (10/300)
ART,N	633	1
N,V	358	.32
N,N	108	.10
N,P	366	.33
V,N	134	.37
V,P	150	.42
V,ART	194	.54
P,ART	226	.62
P,N	140	.38
V,V	30	.08

## *Tag Frequencies*

Φ	ART	N	V	P
300	633	1102	358	366

# Sample Lexical Generation Probabilities

- $P(\text{an} \mid \text{ART})$  .36
- $P(\text{an} \mid \text{N})$  .001
- $P(\text{flies} \mid \text{N})$  .025
- $P(\text{flies} \mid \text{V})$  .076
- $P(\text{time} \mid \text{N})$  .063
- $P(\text{time} \mid \text{V})$  .012
- $P(\text{arrow} \mid \text{N})$  .076
- $P(\text{like} \mid \text{N})$  .012
- $P(\text{like} \mid \text{V})$  .10

# Computing the Probability of a Sentence and Tags

- Find the sequence of tags that maximizes the formula

$$\prod_{i=1}^n P(T_i | T_{i-1}) * P(w_i | T_i)$$

- $P(T_1..T_n | w_1..w_n)$ , which can be estimated as:
  - $P(T_i | T_{i-1})$  is computed by multiplying the arc values in the HMM.
  - $P(w_i | T_i)$  is computed by multiplying the lexical generation probabilities associated with each word

# Statistical POS tagging

- What is the most likely sequence of tags for the given sequence of words  $w$

$$\begin{aligned}\operatorname{argmax}_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) &= \operatorname{argmax}_{\mathbf{t}} \frac{P(\mathbf{t}, \mathbf{w})}{P(\mathbf{w})} \\ &= \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t}, \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t})P(\mathbf{w}|\mathbf{t})\end{aligned}$$

$$\begin{aligned}P(\text{ DT JJ NN } | \text{ a smart dog}) \\ &= P(\text{DD JJ NN a smart dog}) / P(\text{ a smart dog}) \\ &= P(\text{DD JJ NN}) P(\text{a smart dog} | \text{ DD JJ NN })\end{aligned}$$

# Transition Probability

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
  - $P(\mathbf{t}) = P(t_1, t_2, \dots, t_n)$   
 $= P(t_1)P(t_2 | t_1)P(t_3 | t_2, t_1) \dots P(t_n | t_1 \dots t_{n-1})$   
 $\sim P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$   
 $= \prod_{i=1}^n P(t_i | t_{i-1})$
- Markov assumption
- Bigram model over POS tags!  
(similarly, we can define a n-gram model over POS tags, usually we called high-order HMM)

# Emission Probability

- Joint probability  $P(t, w) = P(t)P(w|t)$
  - Assume words only depend on their POS-tag
  - $P(w|t) \sim P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$
- Independent assumption
- $$= \prod_{i=1}^n P(w_i | t_i)$$

i.e.,  $P(\text{a smart dog} | \text{DD JJ NN})$

$$= P(\text{a} | \text{DD}) P(\text{smart} | \text{JJ}) P(\text{dog} | \text{NN})$$

# Put them together

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
- $P(\mathbf{t}, \mathbf{w})$ 
$$= P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$$
$$P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$$
$$= \prod_{i=1}^n P(w_i | t_i)P(t_i | t_{i-1})$$

e.g.,  $P(\text{a smart dog , DD JJ NN })$   
 $= P(\text{a} | \text{DD}) P(\text{smart} | \text{JJ }) P(\text{ dog} | \text{NN })$   
 $P(\text{DD} | \text{start}) P(\text{JJ} | \text{DD}) P(\text{NN} | \text{JJ })$

# Table representation

Transition Matrix  $A$

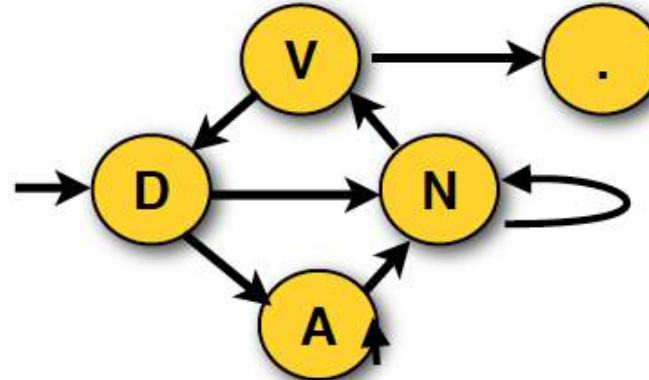
	D	N	V	A	.
D	0.8		0.2		
N	0.7	0.3			
V	0.6				0.4
A		0.8		0.2	
.					

Emission Matrix  $B$

	<i>the</i>	<i>man</i>	<i>ball</i>	<i>throws</i>	<i>sees</i>	<i>red</i>	<i>blue</i>	.
D	1.0							
N		0.7	0.3					
V				0.6	0.4			
A						0.8	0.2	
.								1

Initial state vector  $\pi$

	D	N	V	A	.
$\pi$	1.0				



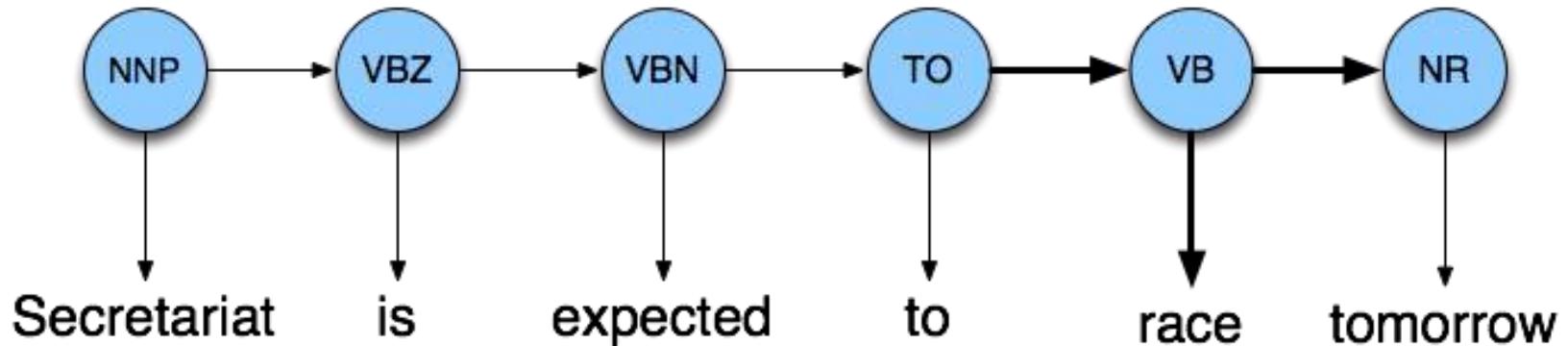
Let  $\lambda = \{A, B, \pi\}$  represents all parameters

# Example: The Verb “race”

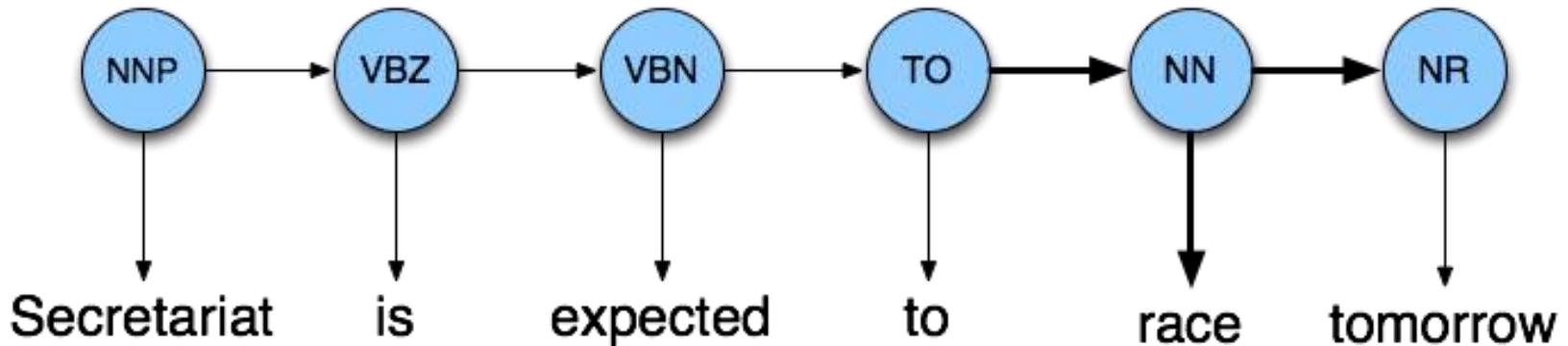
- Secretariat/**NNP** is/**VBZ** expected/**VBN** to/**TO**  
**race**/**VB** tomorrow/**NR**
- People/**NNS** continue/**VB** to/**TO** inquire/**VB**  
the/**DT** reason/**NN** for/**IN** the/**DT** **race**/**NN**  
for/**IN** outer/**JJ** space/**NN**
- How do we pick the right tag?

# Disambiguating “race”

(a)

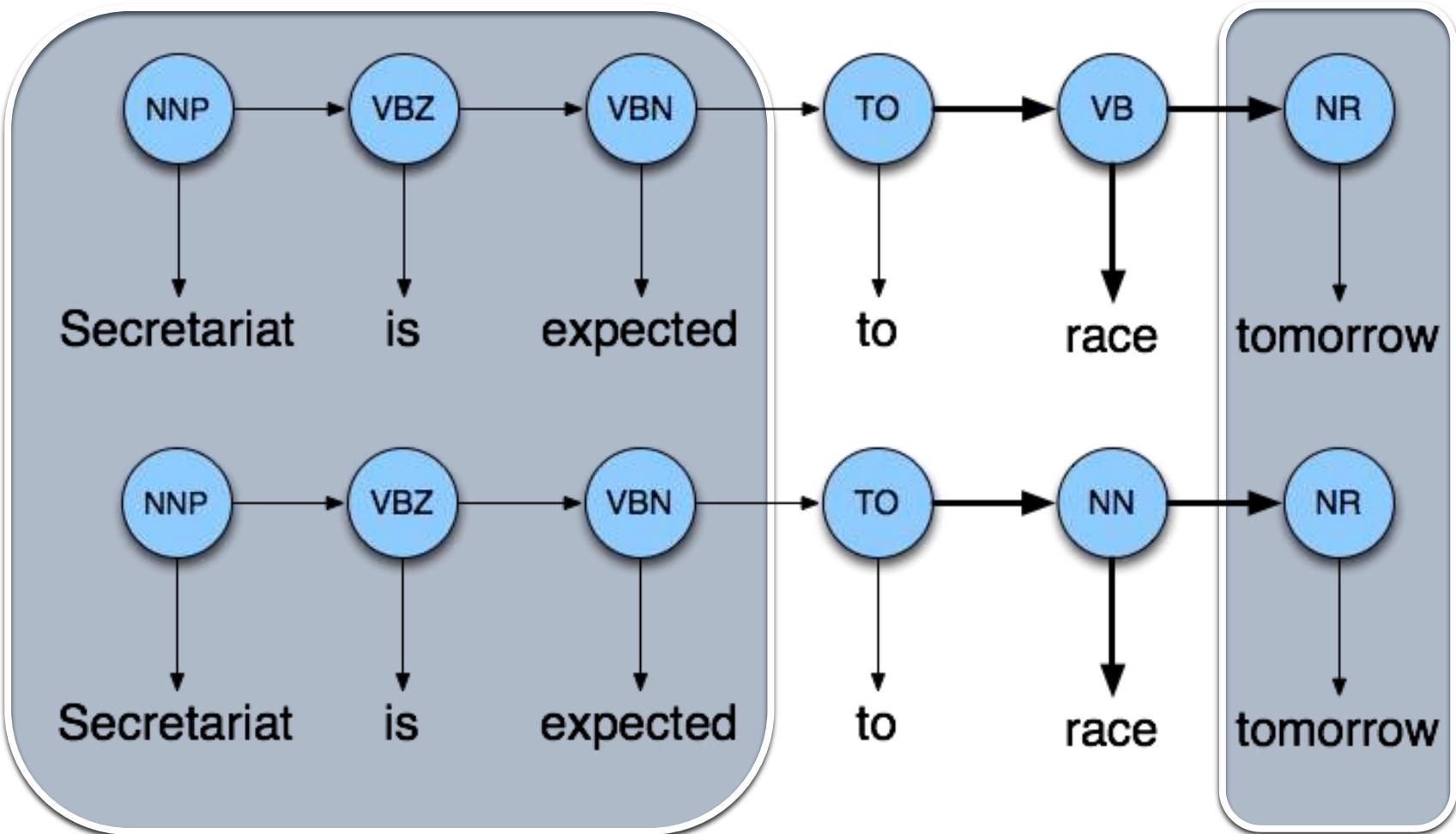


(b)



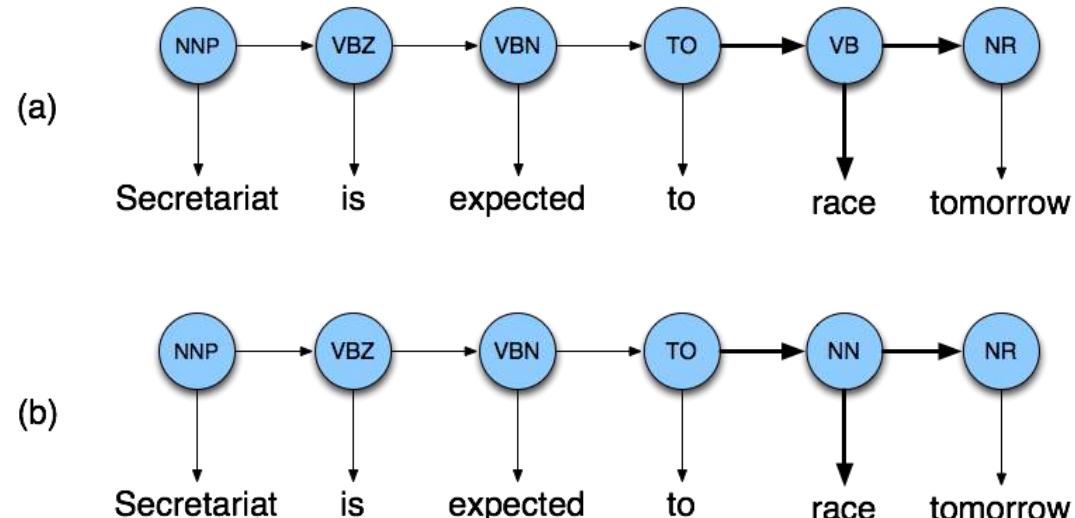
# Disambiguating “race”

(a)



# Example : NLTK POS tags

- $P(NN|TO) = .00047$
- $P(VB|TO) = .83$
- $P(race|NN) = .00057$
- $P(race|VB) = .00012$
- $P(NR|VB) = .0027$
- $P(NR|NN) = .0012$



NR-Adverbial Noun (tomorrow), TO-proposition

- $P(VB|TO)P(NR|VB)P(race|VB) = .00000027$
- $P(NN|TO)P(NR|NN)P(race|NN) = .00000000032$
- So we (correctly) choose the verb tag for “race”

<https://www.guru99.com/pos-tagging-chunking-nltk.html>

# Part-of-Speech Tagging for Morphological Rich Languages

- Augmentations to tagging algorithms become necessary when dealing with languages with rich morphology like Czech, Hungarian and Indian languages
- Highly inflectional languages also have much more information than English coded in word morphology, like case (nominative, accusative etc) or gender (masculine, feminine). Ex: I am speaking in English doesn't tell about gender but in Hindi/Marathi you can come to know gender depending on Bolti or Bolta(inflections)
- This information is important for tasks like parsing and coreference resolution, part-of-speech taggers for morphologically rich languages need to label words with case and gender information.

# Part-of-Speech Tagging for Morphological Rich Languages

---

- Using a morphological parse sequence like Noun+A3sg+Pnon+Gen as the part-of-speech tag greatly increases the number of parts of speech, and so tagsets can be 4 to 10 times larger than the 50–100 tags we have seen for English.
- With such large tagsets, each word needs to be morphologically analyzed to generate the list of possible morphological tag sequences (part-of-speech tags) for the word.
- The role of the tagger is then to disambiguate among these tags.
- This method also helps with unknown words since morphological parsers can accept unknown stems and still segment the affixes properly.

# Part-of-Speech Tagging for Morphological Rich Languages

- For non-word-space languages like Chinese, word segmentation is either applied before tagging or done jointly.
- Although Chinese words are on average very short (around 2.4 characters per unknown word compared with 7.7 for English) the problem of unknown words is still large.
- While English unknown words tend to be proper nouns in Chinese the majority of unknown words are common nouns and verbs because of extensive compounding.
- Tagging models for Chinese use similar unknown word features to English, including character prefix and suffix features

# NLTK POS Tagging Demo



There are different open source POS taggers available

- Stanford PoS Tagger python bind- Java based, but can be used in python but difficult to install
- Flair - POS tagger available for python. It is almost 98% accurate.

These two are the best ones.

- NLTK implementation to be very precise, around 97% and its quite fast.



**Thank you for your time!!**



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 1**  
**Date – 19<sup>th</sup> September 2020**  
**Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin d many others who made their course materials freely available online.

# Session Content

---

- The Hidden Markov Model
- Likelihood Computation:
  - The Forward Algorithm
- Decoding: The Viterbi Algorithm
- HMM Training:
  - The Forward-Backward Algorithm

# Hidden Markov Models

- It is a **sequence model**.
- Assigns a label or class to each unit in a sequence, thus mapping a **sequence of observations** to a **sequence of labels**.
- Probabilistic sequence model: given a sequence of units (e.g. words, letters, morphemes, sentences), compute a probability distribution over possible sequences of labels and choose the best label sequence.
- This is a kind of *generative* model.

# Hidden Markov Model (HMM)

- Oftentimes we want to know what produced the sequence – the **hidden sequence** for the **observed sequence**. For example,
  - Inferring the **words** (hidden) from **acoustic signal** (observed) in speech recognition
  - Assigning **part-of-speech tags** (hidden) to a **sentence** (sequence of words) – **POS tagging**.
  - Assigning named **entity categories** (hidden) to a **sentence** (sequence of words) – Named Entity Recognition.

# HMM Applications

- Speech Recognition including siri
- Gene Prediction
- Handwriting recognition
- Transportation forecasting
- Computational finance
- Cryptanalysis (security)

And all applications which requires sequence processing...

# Definition of HMM

- States  $Q = q_1, q_2 \dots q_N;$
- Observations  $O = o_1, o_2 \dots o_N;$ 
  - Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots v_V\}$
- Transition probabilities
  - Transition probability matrix  $A = \{a_{ij}\}$ 
$$a_{ij} = P(q_t = j | q_{t-1} = i) \quad 1 \leq i, j \leq N$$
- Observation likelihoods
  - Output probability matrix  $B = \{b_i(k)\}$ 
$$b_i(k) = P(X_t = o_k | q_t = i)$$
- Special initial probability vector  $\pi$ 
$$\rho_i = P(q_1 = i) \quad 1 \leq i \leq N$$

# Three Problems

- Given this framework there are 3 problems that we can pose to an HMM
  1. The likelihood of a observation sequence given a model and state sequence
  2. Given an observation sequence and a model, what is the most likely state sequence?
  3. Given an observation sequence, find the best model parameters for a partially specified model

# Problem 1: Observation Likelihood

- The likelihood of a observation sequence given a model and state sequence
  - Used in model development... How do I know if some change I made to the model is making things better?
  - And in classification tasks
    - Word spotting in ASR, language identification, speaker identification, author identification, etc.
      - Train one HMM model per class
      - Given an observation, pass it to each model and compute  $P(\text{seq}|\text{model})$ .

# Problem 2:

## Decoding

- Most probable state sequence given a model and an observation sequence

**Decoding:** Given as input an HMM  $\lambda = (A, B)$  and a sequence of observations  $O = o_1, o_2, \dots, o_T$ , find the most probable sequence of states  $Q = q_1 q_2 q_3 \dots q_T$ .

- Typically used in tagging problems, where the tags correspond to hidden states
  - As we'll see almost any problem can be cast as a sequence labeling problem

# Problem 3: Learning

- Infer the best model parameters, given a partial model and an observation sequence...
  - That is, fill in the A and B tables with the right numbers --  
the numbers that make the observation sequence most likely
- This is to learn the probabilities!

# Solutions

- Problem 1: Forward (learn observation sequence)
- Problem 2: Viterbi (learn state sequence)
- Problem 3: Forward-Backward (learn probabilities)
  - An instance of EM (Expectation Maximization)

## Problem 2: Decoding

- We want, out of all sequences of n tags  $t_1 \dots t_n$  the single tag sequence such that  $P(t_1 \dots t_n | w_1 \dots w_n)$  is highest.

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

- Hat ^ means “our estimate of the best one”
- Argmax<sub>x</sub> f(x) means “the x such that f(x) is maximized”

# Getting to HMMs

- This equation should give us the best tag sequence  $\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$
- But how to make it operational? How to compute this value?
- Intuition of Bayesian inference:
  - Use Bayes rule to transform this equation into a set of probabilities that are easier to compute (and give the right answer)

# Using Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Know this.

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{P(w_1^n | t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n | t_1^n)P(t_1^n)$$

# Likelihood and Prior



$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{\text{likelihood}}{P(w_1^n | t_1^n)} \frac{\text{prior}}{P(t_1^n)}$$

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i | t_i)$$



$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1})$$

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) \approx \operatorname{argmax}_{t_1^n} \prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1})$$

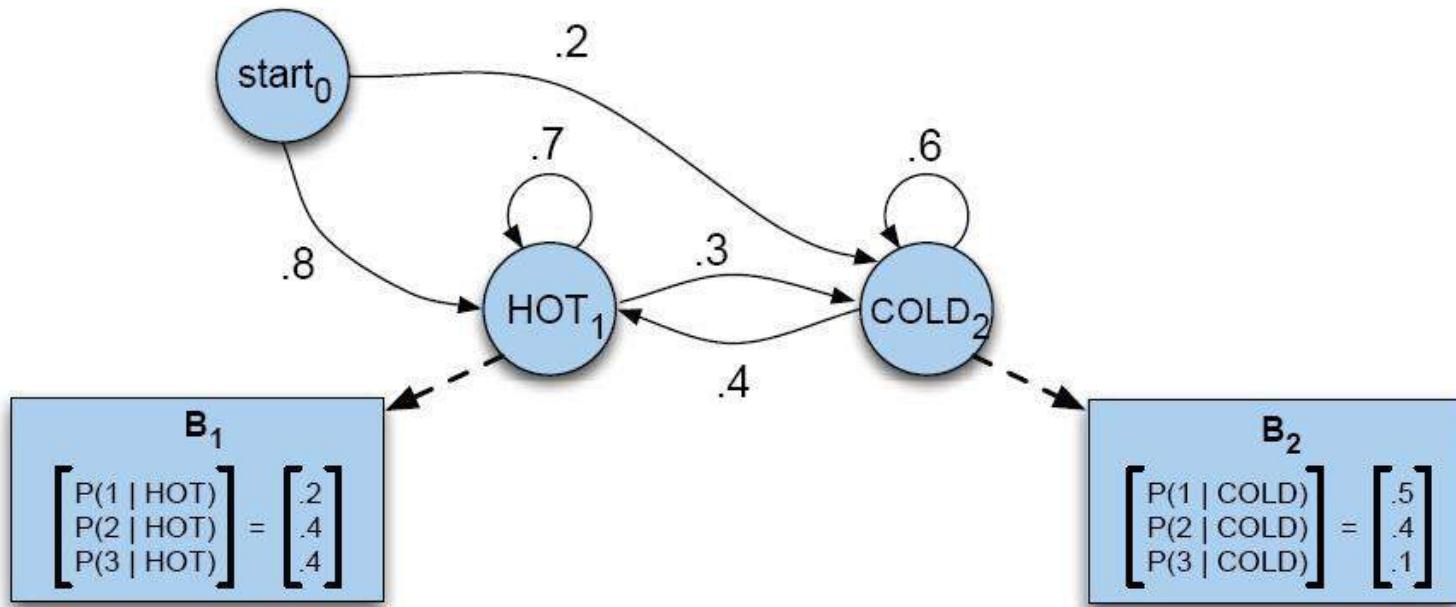
# HMMs for Ice Cream

- You are a climatologist in the year 2799 studying global warming
- You can't find any records of the weather in Baltimore for summer of 2007
- But you find Jason Eisner's diary which lists how many ice-creams Jason ate every day that summer
- Your job: figure out how hot it was each day



# Eisner Task

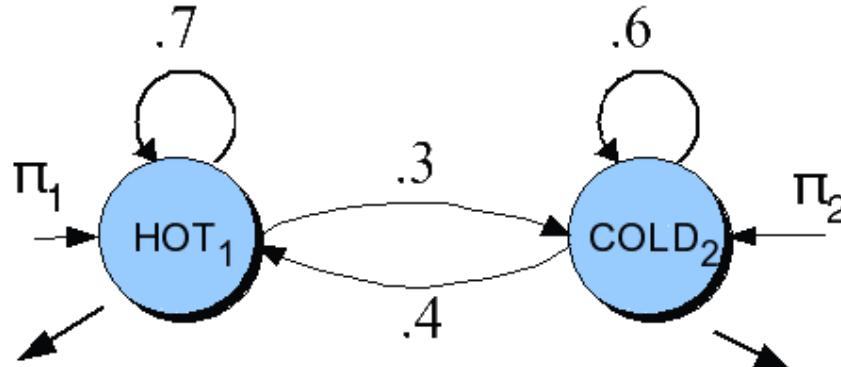
- Given
  - Ice Cream Observation Sequence: 1,2,3,2,2,2,3...
- Produce:
  - Hidden Weather Sequence:  
H,C,H,H,H,C, C...



What's the state sequence for the observed sequence  
“1 3 1”?

# HMM for Ice Cream

$$\pi = [.8, .2]$$

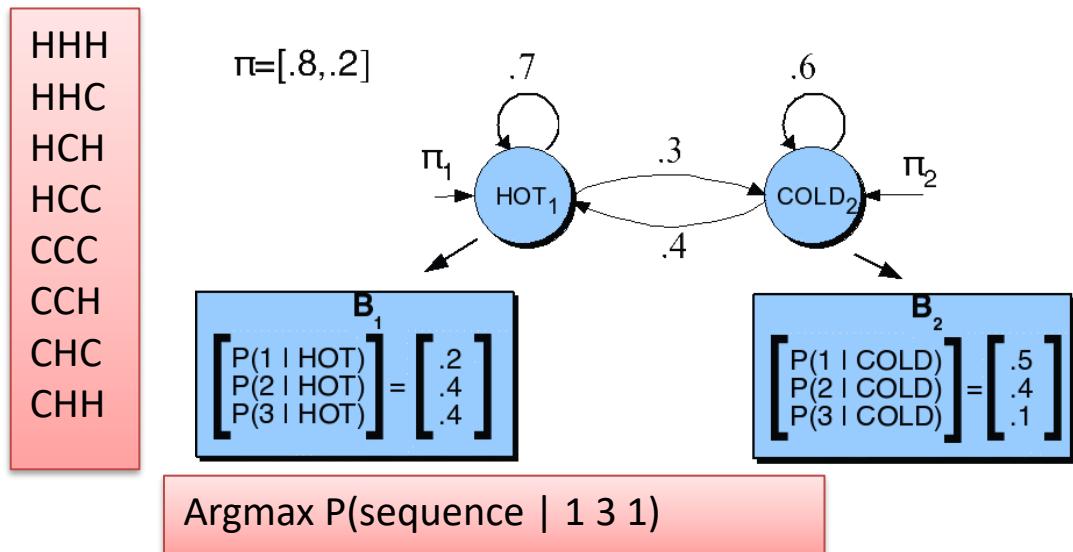


$$B_1 \left[ \begin{array}{c} P(1 \mid HOT) \\ P(2 \mid HOT) \\ P(3 \mid HOT) \end{array} \right] = \left[ \begin{array}{c} .2 \\ .4 \\ .4 \end{array} \right]$$

$$B_2 \left[ \begin{array}{c} P(1 \mid COLD) \\ P(2 \mid COLD) \\ P(3 \mid COLD) \end{array} \right] = \left[ \begin{array}{c} .5 \\ .4 \\ .1 \end{array} \right]$$

# Ice Cream HMM

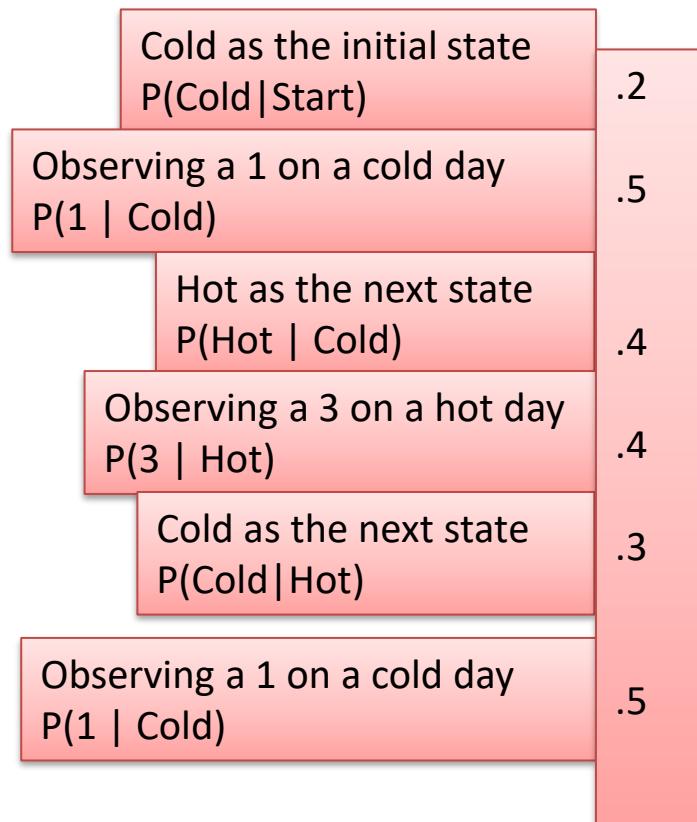
- Let's just do **131** as the sequence
  - How many underlying state (hot/cold) sequences are there?



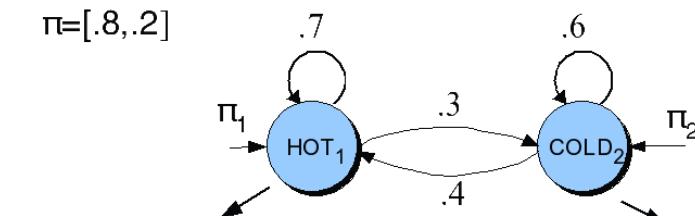
- How do you pick the right one?

# Ice Cream HMM

Let's just do 1 sequence: CHC



$$\pi = [.8, .2]$$



$$B_1 = \begin{bmatrix} P(1 | \text{HOT}) \\ P(2 | \text{HOT}) \\ P(3 | \text{HOT}) \end{bmatrix} = \begin{bmatrix} .2 \\ .4 \\ .4 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} P(1 | \text{COLD}) \\ P(2 | \text{COLD}) \\ P(3 | \text{COLD}) \end{bmatrix} = \begin{bmatrix} .5 \\ .4 \\ .1 \end{bmatrix}$$

$$\begin{aligned} P(\text{CHC}) &= 0.2 * 0.5 * 0.4 * 0.4 * 0.3 * 0.5 \\ &= 0.0024 \end{aligned}$$

# Viterbi Algorithm

- Create an array
  - Columns corresponding to observations
  - Rows corresponding to possible hidden states
- Recursively compute the probability of the most likely subsequence of states that accounts for the first  $t$  observations and ends in state  $s_j$ .

$$v_t(j) = \max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1, \dots, q_{t-1}, o_1, \dots, o_t, q_t = s_j | \lambda)$$

- Also record “backpointers” that subsequently allow backtracing the most probable state sequence.

# Viterbi Example 1: POS Tagging

Example: I want to race.

	<b>VB</b>	<b>TO</b>	<b>NN</b>	<b>PPSS</b>
<b>&lt;s&gt;</b>	.019	.0043	.041	.067
<b>VB</b>	.0038	.035	.047	.0070
<b>TO</b>	.83	0	.00047	0
<b>NN</b>	.0040	.016	.087	.0045
<b>PPSS</b>	.23	.00079	.0012	.00014

**Figure 5.15** Tag transition probabilities (the  $a$  array,  $p(t_i|t_{i-1})$ ) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus  $P(PPSS|VB)$  is .0070. The symbol  $<s>$  is the start-of-sentence symbol.

	<b>I</b>	<b>want</b>	<b>to</b>	<b>race</b>
<b>VB</b>	0	.0093	0	.00012
<b>TO</b>	0	0	.99	0
<b>NN</b>	0	.000054	0	.00057
<b>PPSS</b>	.37	0	0	0

**Figure 5.16** Observation likelihoods (the  $b$  array) computed from the 87-tag Brown corpus without smoothing.

# Viterbi Algorithm Example

---

Algorithm first creates N or four state columns.

- First column corresponds to the observation of the first word “I”,
- the second to the second word “want”,
- the third to the third word “to”, and
- the fourth to the fourth word “race”

Begin by setting the Viterbi value in each cell to the product of the transition probability (into it from the state state) and the observation probability (of the first word)

# Viterbi Algorithm Example

---

- For each state  $q_j$  at time  $t$ , the value Viterbi  $[s,t]$  is computed by taking the maximum over the extensions of all the paths that lead to the current cell, following the following equation:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

- $v_{t-1}(i)$  the **previous Viterbi path probability** from the previous time step  
 $a_{ij}$  the **transition probability** from previous state  $q_i$  to current state  $q_j$   
 $b_j(o_t)$  the **state observation likelihood** of the observation symbol  $o_t$  given the current state  $j$

$\alpha_{word}$  $q_4$ 

$$v_1(4) = .041 \times 0 = 0$$

 $q_3$ 

$$v_1(3) = .0043 \times 0 = 0$$

 $q_2$ 

$$P(\text{VBG}|\text{VB}, \text{past}) \\ .041 \times 1.0 = .041$$

$$v_1(2) = .019 \times 0 = 0$$

 $q_1$ 

$$P(\text{VBG}|\text{VB}, \text{past}) \\ .0043 \times 1.0 = .0043$$

$$v_1(1) = .087 \times .37 = .0325$$

 $q_0$  $O_1$  $O_2$  $O_3$  $O_4$  $t$

# The Viterbi Algorithm

**function** VITERBI(*observations* of len  $T$ ,*state-graph* of len  $N$ ) **returns** *best-path*

create a path probability matrix  $viterbi[N+2,T]$

**for** each state  $s$  **from** 1 **to**  $N$  **do** ; initialization step

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[s,1] \leftarrow 0$

**for** each time step  $t$  **from** 2 **to**  $T$  **do** ; recursion step

**for** each state  $s$  **from** 1 **to**  $N$  **do**

$viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$

$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$

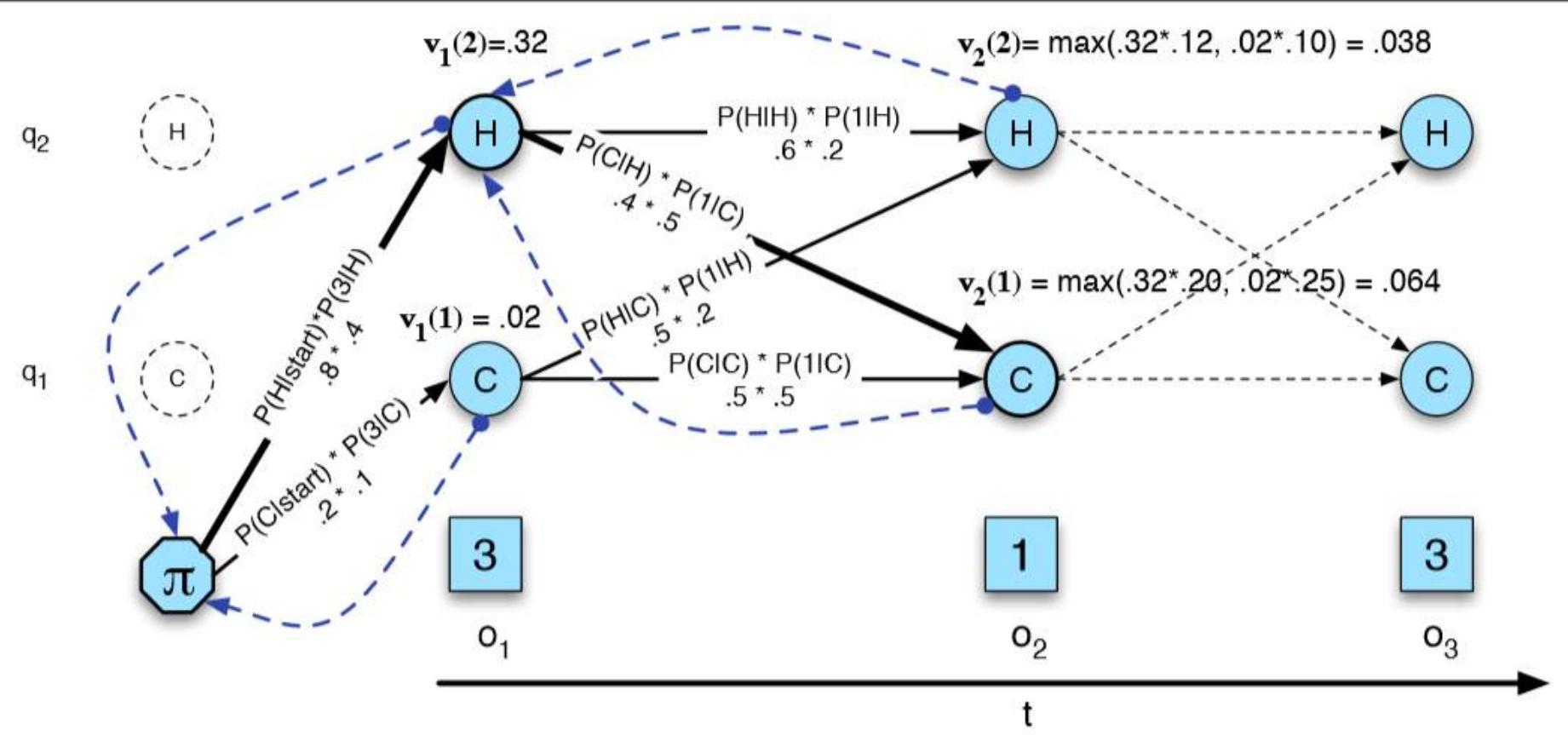
$viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step

$backpointer[q_F, T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step

**return** the backtrace path by following backpointers to states back in time from  $backpointer[q_F, T]$

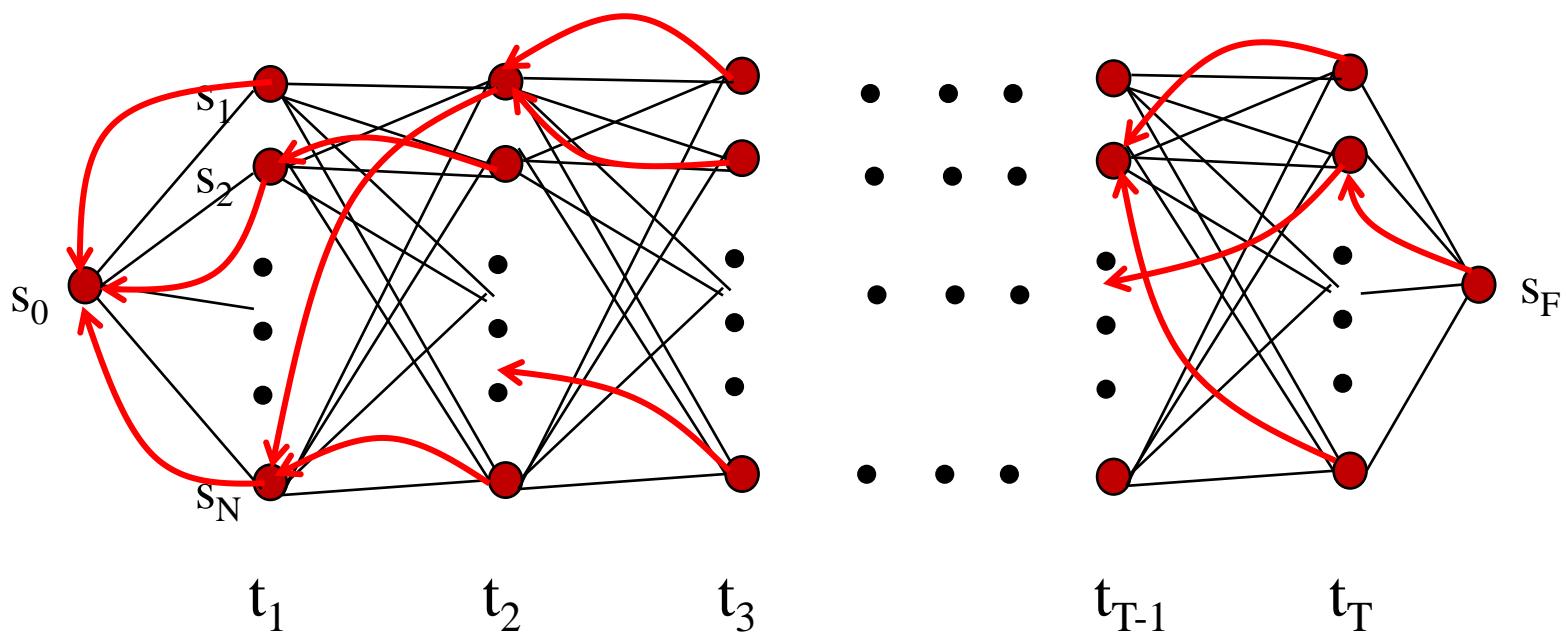


# Viterbi Example 2: Ice Cream

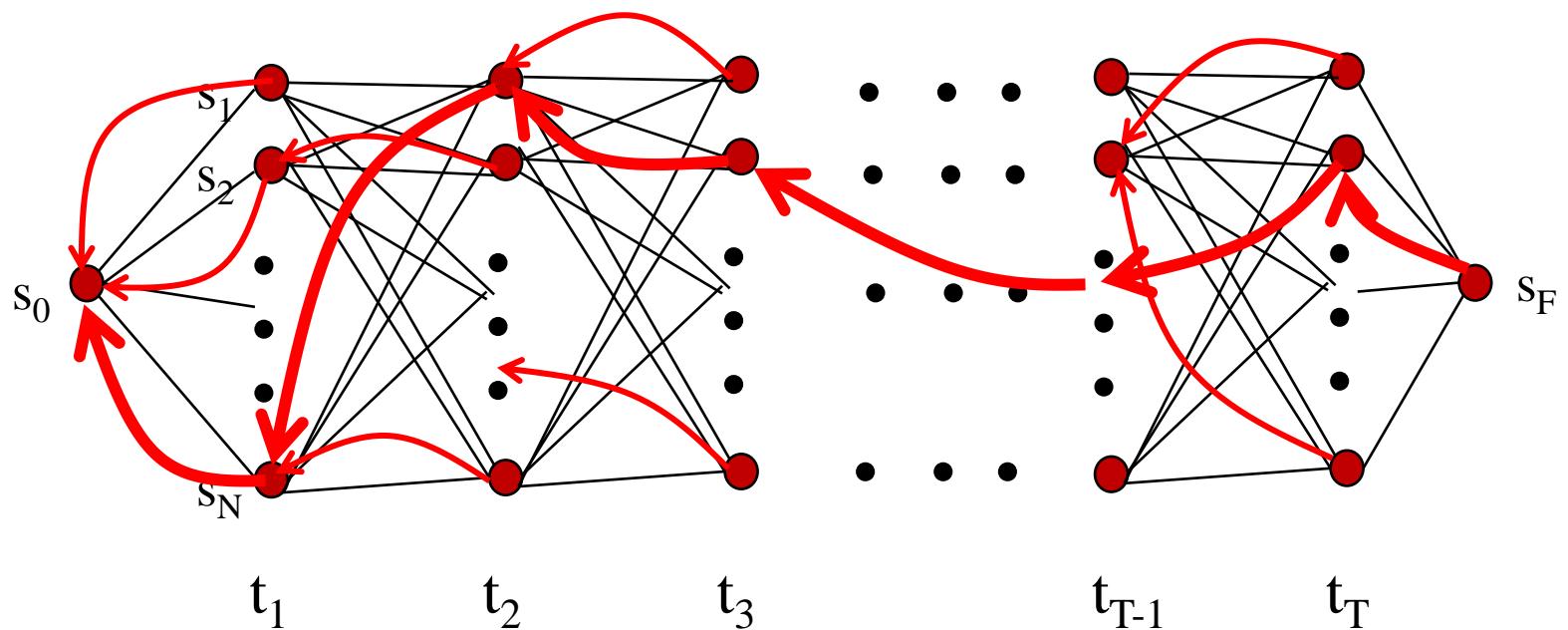


**Figure A.10** The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

# Viterbi Backpointers



# Viterbi Backtrace



Most likely Sequence:  $s_0 \ s_N \ s_1 \ s_2 \dots s_2 \ s_F$

# Forward

- Efficiently computes the probability of an observed sequence given a model
  - $P(\text{sequence}|\text{model})$
- Nearly identical to Viterbi; **replace the MAX with a SUM**

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \dots$$

# Forward

- Efficiently computes the probability of an observed sequence given a model
  - $P(\text{sequence}|\text{model})$
- Nearly identical to Viterbi; **replace the MAX with a SUM**

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \dots$$

# The Forward Algorithm

**function** FORWARD(*observations* of len  $T$ , *state-graph* of len  $N$ ) **returns** *forward-prob*

create a probability matrix  $forward[N+2,T]$

**for** each state  $s$  **from** 1 **to**  $N$  **do** ; initialization step

$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$

**for** each time step  $t$  **from** 2 **to**  $T$  **do** ; recursion step

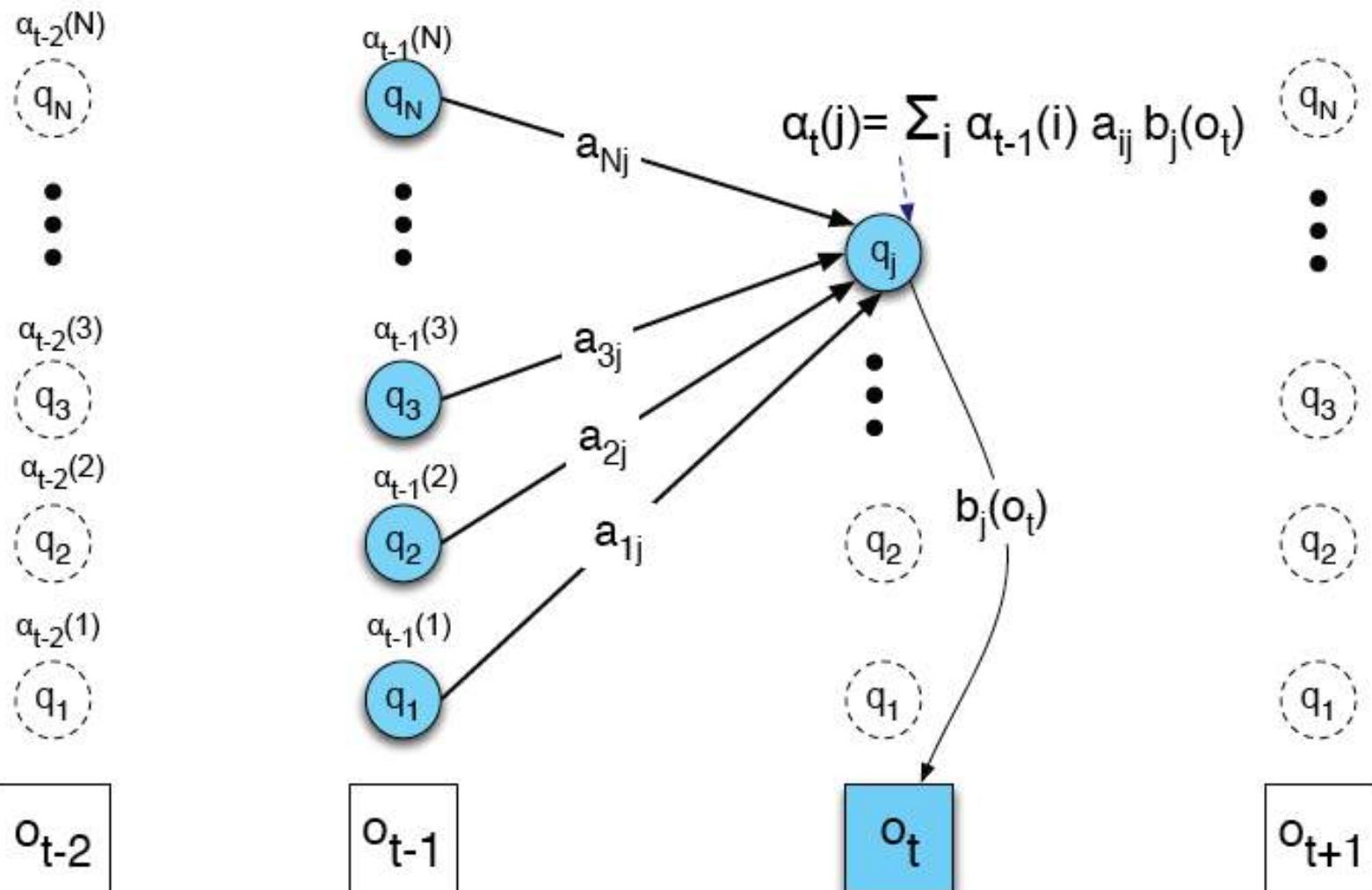
**for** each state  $s$  **from** 1 **to**  $N$  **do**

$forward[s,t] \leftarrow \sum_{s'=1}^N forward[s',t-1] * a_{s',s} * b_s(o_t)$

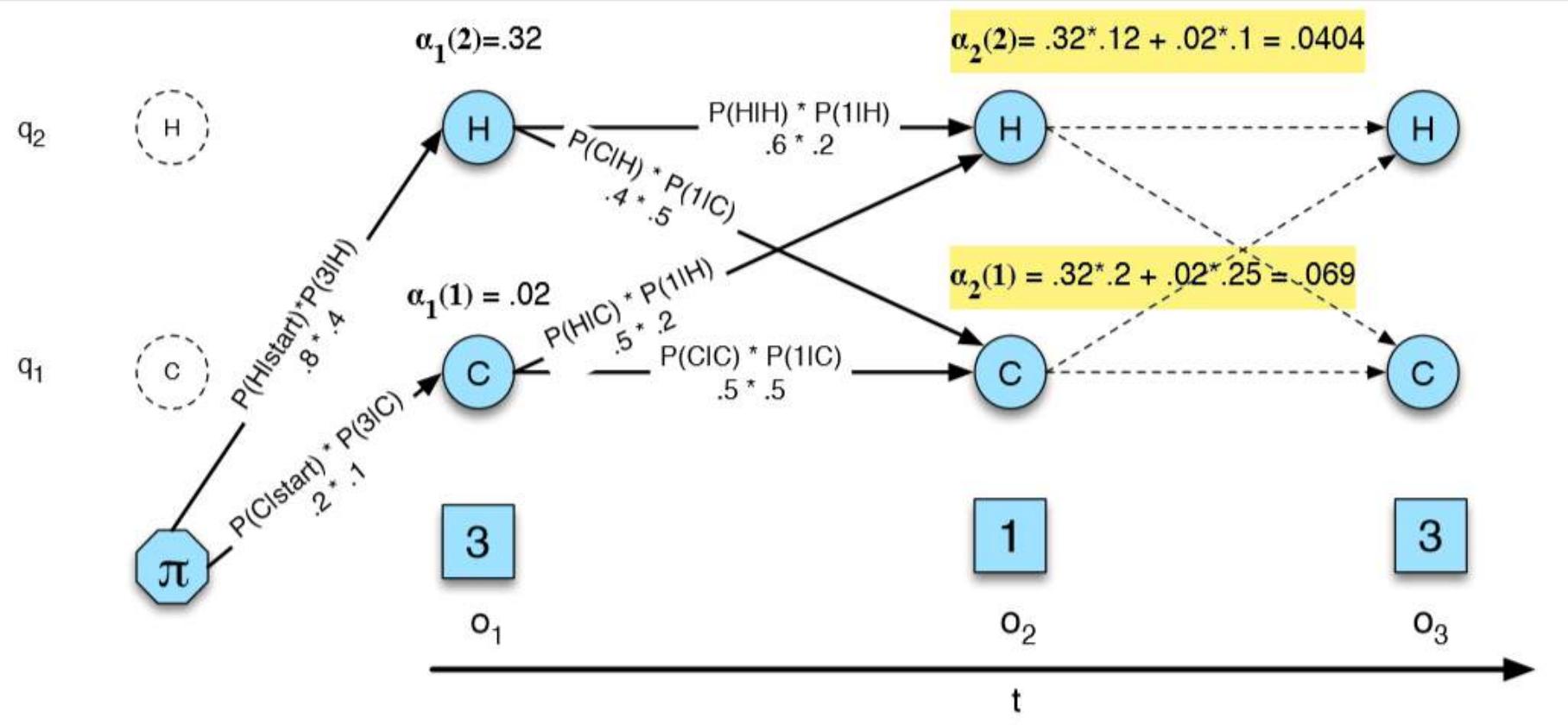
$forward[q_F, T] \leftarrow \sum_{s=1}^N forward[s, T] * a_{s,q_F}$  ; termination step

**return**  $forward[q_F, T]$

# Visualizing Forward



# Forward Algorithm: Ice Cream



**Figure A.5** The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of  $\alpha_t(j)$  for two states at two time steps. The computation in each cell follows Eq. A.12:  $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$ . The resulting probability expressed in each cell is Eq. A.11:  $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$ .

# Problem 3

- Infer the best model parameters, given a skeletal model and an observation sequence...
  - That is, fill in the A and B tables with the right numbers...
    - The numbers that make the observation sequence most likely
  - Useful for getting an HMM without having to hire annotators...

# Forward-Backward

**Learning:** Given an observation sequence  $O$  and the set of possible states in the HMM, learn the HMM parameters  $A$  and  $B$ .

- **Baum-Welch = Forward-Backward Algorithm**  
(Baum 1972)
- Is a special case of the EM or Expectation-Maximization algorithm
- The algorithm will let us learn the transition probabilities  $A = \{a_{ij}\}$  and the emission probabilities  $B = \{b_i(o_t)\}$  of the HMM

# Sketch of Baum-Welch (EM) Algorithm for Training HMMs

Assume an HMM with  $N$  states.

Randomly set its parameters  $\lambda = (A, B)$

(making sure they represent legal distributions)

Until converge (i.e.  $\lambda$  no longer changes) do:

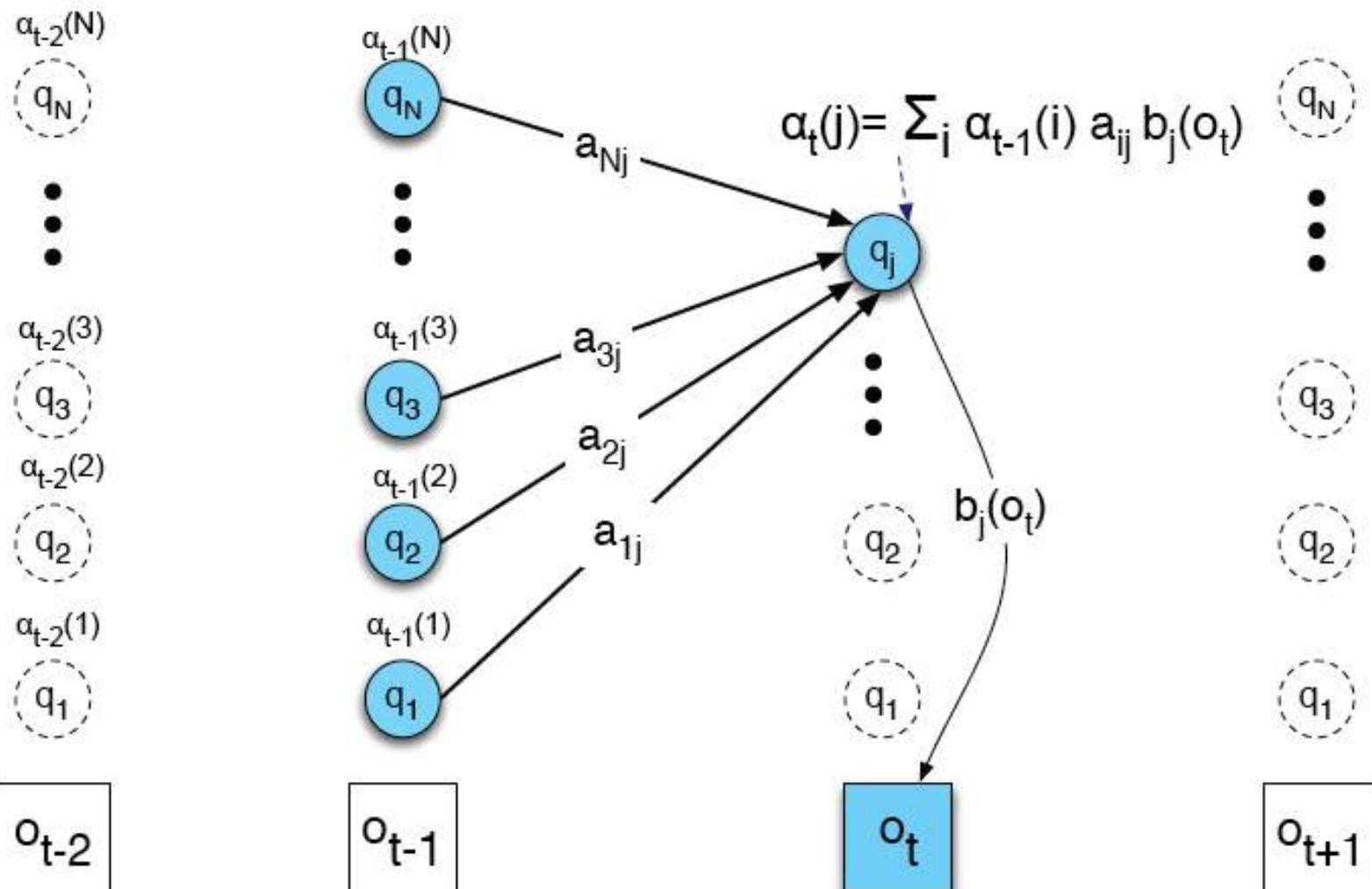
E Step: Use the forward/backward procedure to determine the probability of various possible state sequences for generating the training data

M Step: Use these probability estimates to re-estimate values for all of the parameters  $\lambda$

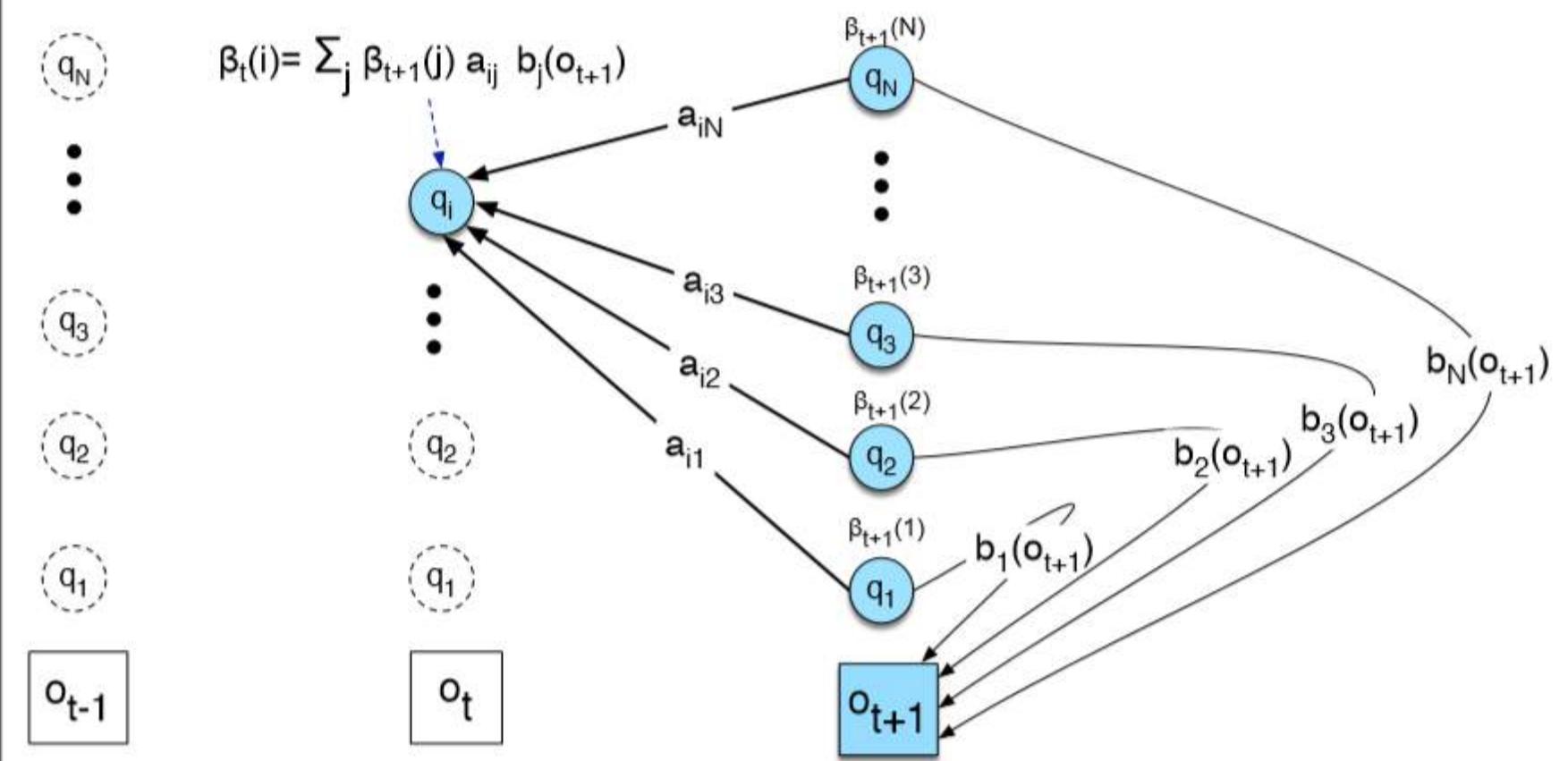
# EM Properties

- Each iteration changes the parameters in a way that is guaranteed to increase the likelihood of the data:  $P(O|\lambda)$ .
- Anytime algorithm: Can stop at any time prior to convergence to get approximate solution.
- Converges to a local maximum.

# Visualizing Forward



# Backward Probabilities



**Figure A.11** The computation of  $\beta_t(i)$  by summing all the successive values  $\beta_{t+1}(j)$  weighted by their transition probabilities  $a_{ij}$  and their observation probabilities  $b_j(o_{t+1})$ . Start and end states not shown.

# Intuition for re-estimation of $a_{ij}$

- We will estimate  $\hat{a}_{ij}$  via this intuition:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- intuition:
  - If we knew this probability for *each* time  $t$ , we could sum over all  $t$  to get expected value for  $i \rightarrow j$ .

# Intuition for re-estimation of $a_{ij}$

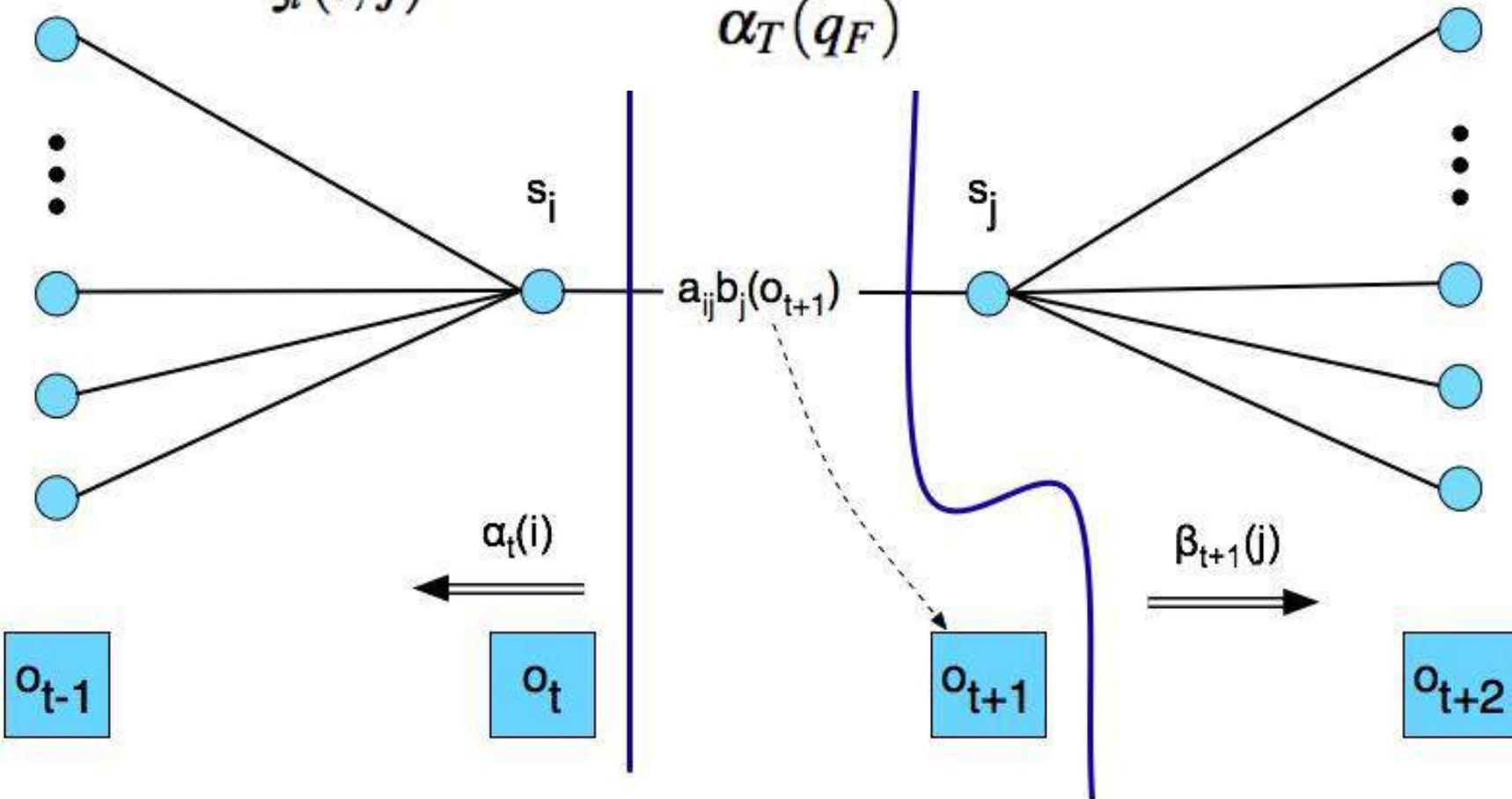
- Assume we had some estimate of the probability that a given transition  $i \rightarrow j$  was taken at a particular point in time  $t$  in the observation sequence.
- If we knew this probability for each particular time  $t$ , we could sum over all times  $t$  to estimate the total count for the transition  $i \rightarrow j$ .

More formally, let's define the probability  $\xi_t$  as the probability of being in state  $i$  at time  $t$  and state  $j$  at time  $t + 1$ , given the observation sequence and of course the model:

$$\xi_t(i, j) = P(q_t = i, q_{t+1} = j | O, \lambda) \quad (\text{A.17})$$

# Intuition for re-estimation of $a_{ij}$

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\alpha_T(q_F)}$$



# Re-estimating $a_{ij}$

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- The expected number of transitions from state  $i$  to state  $j$  is the sum over all  $t$  of  $\xi$
- The total expected number of transitions out of state  $i$  is the sum over all transitions out of state  $i$
- Final formula for reestimated  $a_{ij}$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{j=1}^N \xi_t(i, j)}$$

# The Forward-Backward Alg

**function** FORWARD-BACKWARD(*observations* of len  $T$ , *output vocabulary*  $V$ , *hidden state set*  $Q$ ) **returns**  $HMM=(A,B)$

**initialize**  $A$  and  $B$

**iterate** until convergence

## E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall t, i, \text{ and } j$$

## M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} N} \quad \hat{b}_j(v_k) = \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$$

**return**  $A, B$

# Summary: Forward-Backward Algorithm

- 1) Initialize  $\Phi = (A, B)$
- 2) Compute  $\alpha, \beta, \xi$  using observations
- 3) Estimate new  $\Phi' = (A, B)$
- 4) Replace  $\Phi$  with  $\Phi'$
- 5) If not converged go to 2

# Bidirectionality

---

- One problem with the HMM models as presented is that they are exclusively run left-to-right.
- Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions

# Bidirectionality

---

- Any sequence model can be turned into a bidirectional model by using multiple passes.
- For example, the first pass would use only part-of-speech features from already-disambiguated words on the left. In the second pass, tags for all words, including those on the right, can be used.
- Alternately, the tagger can be run twice, once left-to-right and once right-to-left.
- In Viterbi decoding, the classifier chooses the higher scoring of the two sequences (left-to-right or right-to-left).
- Modern taggers are generally run bidirectionally.

# Issues with HMM

---

- Unknown Words
  - We do not have the probabilities
    - Use smoothing
- Limited Context
  - Use trigrams/ 4 grams etc.
  - Sparsity

# Maximum Entropy Markov Model

---

- Identify heterogeneous set of features
  - tag given, the previous tag or the word given, the tag you can use, any other features which may contribute to the choice of part of speech tags.
- Turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word as a feature in the classification of the next word.
- When we apply logistic regression in this way, it's called maximum entropy Markov model or MEMM

# Maximum Entropy Markov Model

---

- Let the sequence of words be  $W = w_1^n$  and the sequence of tags  $T = t_1^n$
- In an HMM to compute the best tag sequence that maximizes  $P(T|W)$  we rely on Bayes' rule and the likelihood  $P(W|T)$ :

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T P(W|T)P(T) \\ &= \operatorname{argmax}_T \prod_i P(\text{word}_i|\text{tag}_i) \prod_i P(\text{tag}_i|\text{tag}_{i-1})\end{aligned}$$

# Maximum Entropy Markov Model

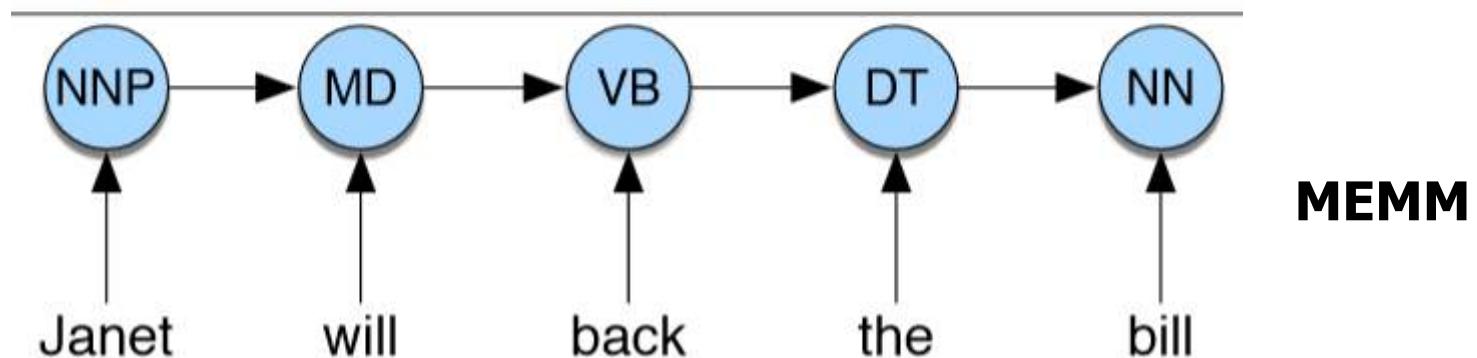
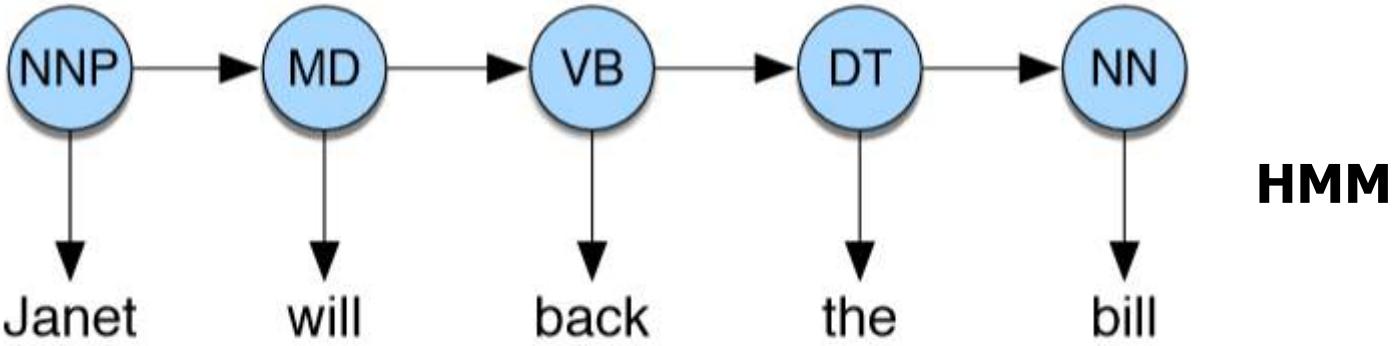
---

- In an MEMM, by contrast, we compute the posterior  $P(T|W)$  directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$

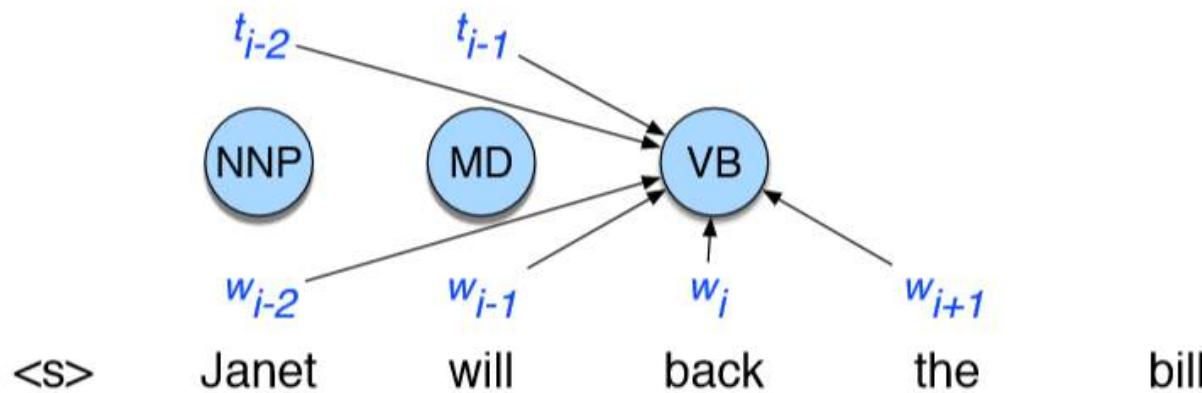
# Maximum Entropy Markov Model

- Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability  $P(t_i|w_i, t_{i-1})$  in a different way than an HMM
- HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).



# Maximum Entropy Markov Model

- Reason to use a discriminative sequence model is that it's easier to incorporate a lot of features



**Figure 8.13** An MEMM for part-of-speech tagging showing the ability to condition on more features.

# MEMM

---

- Janet/NNP will/MD back/VB the/DT bill/NN, when  $w_i$  is the word back, would generate the following features

$t_i = \text{VB}$  and  $w_{i-2} = \text{Janet}$

$t_i = \text{VB}$  and  $w_{i-1} = \text{will}$

$t_i = \text{VB}$  and  $w_i = \text{back}$

$t_i = \text{VB}$  and  $w_{i+1} = \text{the}$

$t_i = \text{VB}$  and  $w_{i+2} = \text{bill}$

$t_i = \text{VB}$  and  $t_{i-1} = \text{MD}$

$t_i = \text{VB}$  and  $t_{i-1} = \text{MD}$  and  $t_{i-2} = \text{NNP}$

$t_i = \text{VB}$  and  $w_i = \text{back}$  and  $w_{i+1} = \text{the}$

# Decoding and Training MEMMs

- The most likely sequence of tags is then computed by combining these features of the input word  $w_i$ , its neighbors within  $l$  words  $w_{i-l}^{i+l}$ , and the previous  $k$  tags  $t_{i-k}^{i-1}$  as follows (using  $\theta$  to refer to feature weights instead of  $w$  to avoid the confusion with  $w$  meaning words):

$$\begin{aligned}
 \hat{T} &= \operatorname{argmax}_T P(T|W) \\
 &= \operatorname{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
 &= \operatorname{argmax}_T \prod_i \frac{\exp \left( \sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}{\sum_{t' \in \text{tagset}} \exp \left( \sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}
 \end{aligned}$$

# How to decode to find this optimal tag sequence $\hat{T}$ ?

- Simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on.
- This is called a greedy decoding algorithm

```

function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T
    for  $i = 1$  to  $\text{length}(W)$ 
         $\hat{t}_i = \underset{t' \in T}{\text{argmax}} P(t' | w_{i-l}^{i+l}, t_{i-k}^{i-1})$ 

```

**Figure 8.14** In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

# Issue with greedy algorithm

---

- The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can't use evidence from future decisions.
- Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too great a drop in performance, and we don't use it.
- MEMM with the Viterbi algorithm just as with the HMM, Viterbi finding the sequence of part-of-speech tags that is optimal for the whole sentence

# MEMM with Viterbi algorithm

---

- Finding the sequence of part-of-speech tags that is optimal for the whole sentence. Viterbi value of time t for state j

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

- In HMM

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i) P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T$$

- In MEMM

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T$$

---

# Learning MEMM

---

- Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression.
- Given a sequence of observations, feature functions, and corresponding hidden states, we use gradient descent to train the weights to maximize the log-likelihood of the training corpus.

# References

---

- <https://www.nltk.org/>
- <https://likegeeks.com/nlp-tutorial-using-python-nltk/>
- <https://www.guru99.com/pos-tagging-chunking-nltk.html>
- <https://medium.com/greyatom/learning-pos-tagging-chunking-in-nlp-85f7f811a8cb>
- <https://nlp.stanford.edu/software/tagger.shtml>
- <https://www.forbes.com/sites/mariyayao/2020/01/22/what-are--important-ai--machine-learning-trends-for-2020/#601ce9623239>
- <https://medium.com/fintechexplained/nlp-text-processing-in-data-science-projects-f083009d78fc>



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 5-Parsing  
Date – 26<sup>th</sup> September 2020  
Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

# Session Content

(Ref: Allen James - Chapter 3)

---

- Grammars and Sentence Structure
  - What Makes a Good Grammar
  - Parsing
  - A Top-Down Parser
  - A Bottom-Up Chart Parser
  - Top-Down Chart Parsing
  - Finite State Models and Morphological Processing.
  - Grammars and Logic Programming
-

# Ambiguity is Explosive

- “I saw the man with the telescope”: 2 parses
- “I saw the man on the hill with the telescope.”: 5 parses
- “I saw the man on the hill in Texas with the telescope”: 14 parses
- “I saw the man on the hill in Texas with the telescope at noon.”: 42 parses
- “I saw the man on the hill in Texas with the telescope at noon on Monday” 132 parses

# Some funny examples

---

- Policeman to little boy: “We are looking for a thief with a bicycle.” Little boy: “Wouldn’t you be better using your eyes.”
- Why is the teacher wearing sun-glasses.  
Because the class is so bright.

# Grammars and Sentence Structure

---

- **Grammar**- formal specification of the structures allowable in the language, and
- **Parsing technique**- method of analyzing a sentence to determine its structure according to the grammar

# What Makes a Good Grammar

---

- In small grammars, such as those that describe only a few types of sentences, one structural analysis of a sentence may appear as understandable as another, and little can be said as to why one is superior to the other.
- The analysis that retains its simplicity and generality as it is extended is more desirable
- Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammars (CFGs).

# Context Free Grammar

N a set of non-terminal symbols (or variables)

$\Sigma$  a set of terminal symbols (disjoint from N)

R a set of productions or rules of the form  $A \rightarrow \beta$ , where A is a non-terminal and  $\beta$  is a string of symbols from  $(\Sigma \cup N)^*$

S, a designated non-terminal called the start symbol

# Categories of Phrases

**Noun phrase (NP):** Noun acts as the head word. They start with an article or noun.

**Verb phrase (VP):** Verb acts as the head word. They start with an verb

**Adjective phrase (ADJP):** Adjective as the head word. They start with an adjective

**Adverb phrase (ADVP):** Adverb acts as the head word. They usually start with an adjective

**Prepositional phrase (PP):** Preposition as the head word. They start with an preposition.

# Simple grammar and Parsing Example

S -> NP VP

VP -> V NP

NP -> N

NP -> Adj NP

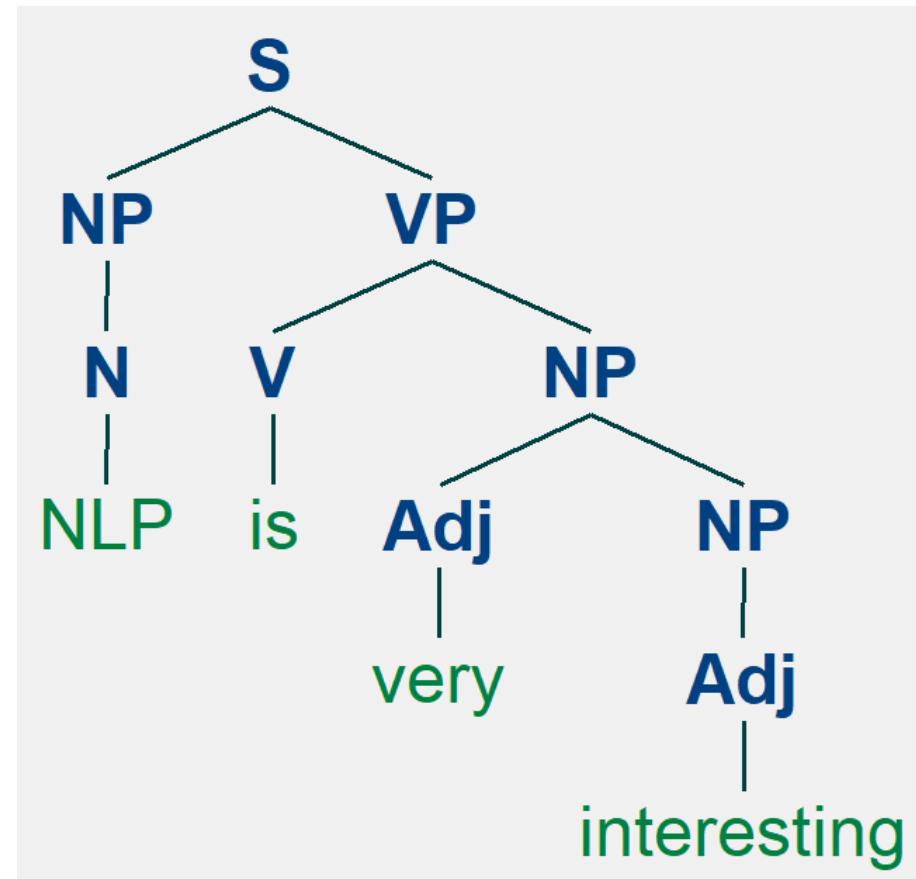
NP -> Adj

N -> NLP

V -> is

Adj -> very

Adj -> interesting

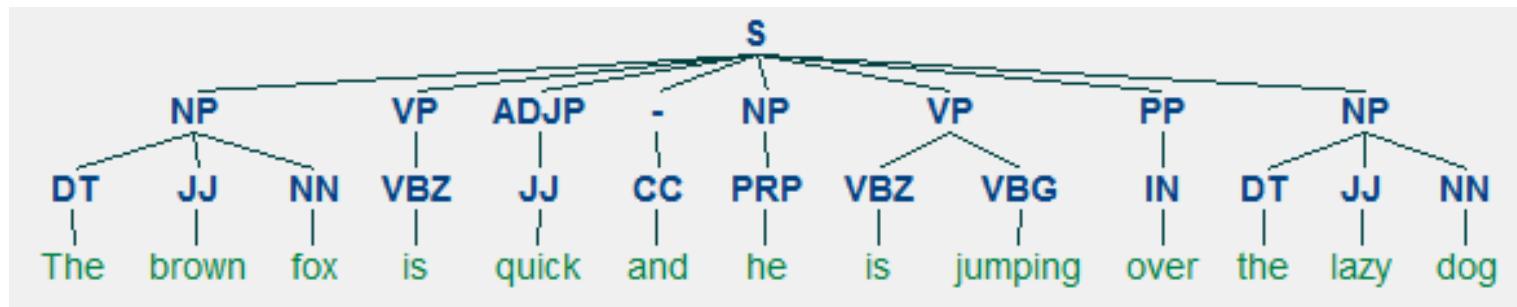


# Parsing

Analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words.

Analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases.

This includes POS tags as well as phrases from a sentence.



# Applications of Parsing

---

**Sentiment Analysis:** Compare "I like Frozen", "I do not like Frozen", and "I like frozen yogurt". The three sentences' words are very similar to each other, yet the first and the second contain inverse statements about the movie "Frozen", while the third is a statement about something else. Parsing here is crucial for understanding.

**Relation Extraction:** "Rome is the capital of Italy and the region of Lazio". While entity extraction can give us the entities here, we need parsing to see which entity is the capital of which other entities.

**Question Answering:** When answering "Who was the first man in space?" you need to parse the question and use parsed sentences to build the answer.

---

# Applications of Parsing

---

**Speech Recognition:** Parsing scores the strings with either a pass/fail or a likelihood score, to give a powerful language model for speech recognition. Similarly, parsing could help in **spell checking, optical character recognition (OCR), text prediction, handwriting detection** etc.

**Machine Translation:** Parsing allows us to choose between several possible translations. Also, it makes translation of phrases and terms easier.

**Grammar Checking:** Parsing can help in checking the grammaticality of a document.

# Parsing

Given a string of non-terminals and a CFG, determine if the string can be generated by the CFG.

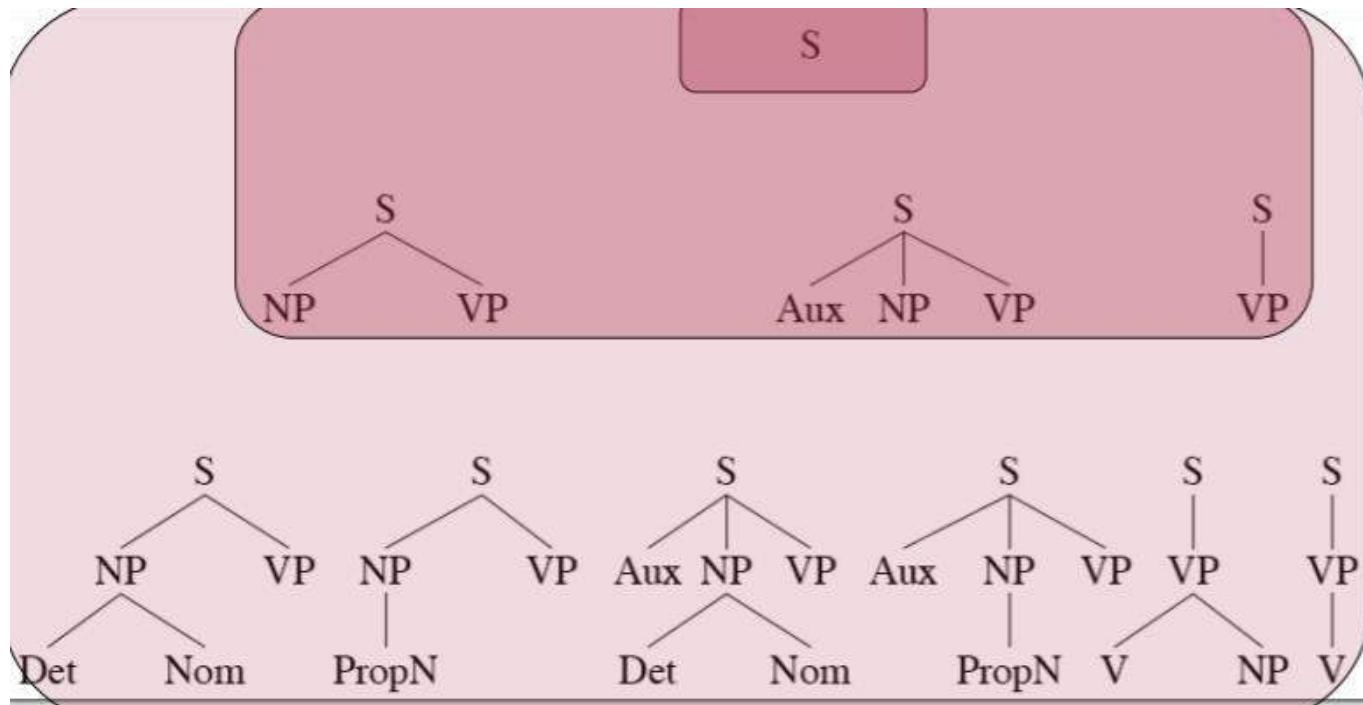
- Also return a parse tree for the string
- Also return all possible parse trees for the string

Must search space of derivations for one that derives the given string.

- Top-Down Parsing: Start searching space of derivations for the start symbol.
- Bottom-up Parsing: Start search space of reverse deviations from the terminal symbols in the string.

# Top down Parsing

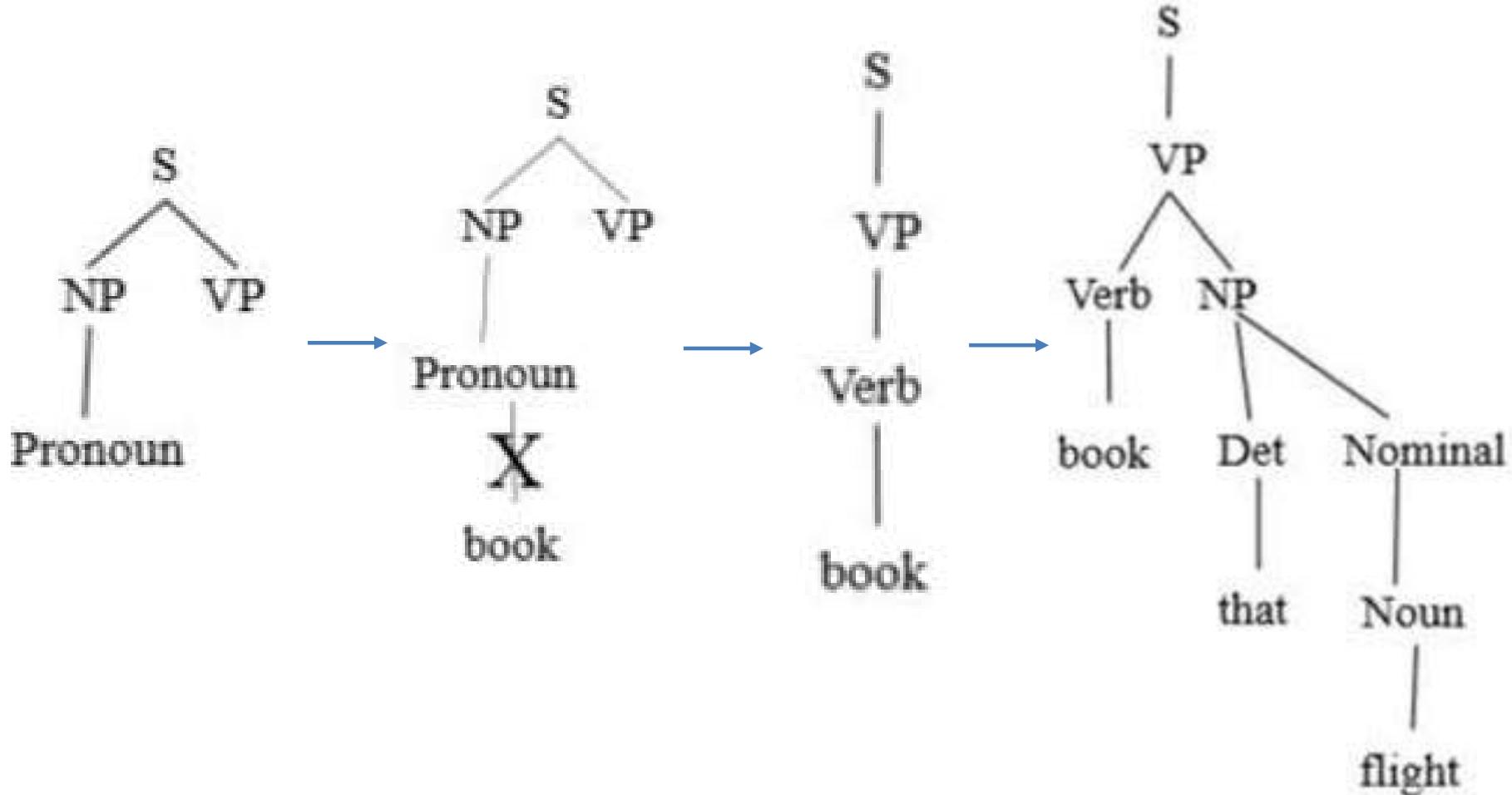
- Parse tree is generated in the top to bottom fashion from root to leaves
- Search space from the start symbol sentence S on LHS



# Top down parsing

1. **S -> NP VP**
2. **NP -> ART N**
3. **NP -> ART ADJ N**
  
4. **VP -> V**
  
5. **VP -> V NP**

# Top down parsing



# Parsing as a search procedure

---

- Parsing as a special case of a search problem as defined in AI.
  1. Select the first state from the possibilities list (and remove it from the list).
  2. Generate the new states by trying every possible option from the selected state (there may be none if we are on a bad path).
  3. Add the states generated in step 2 to the possibilities list.

# Top down parse of : “<sub>1</sub>The <sub>2</sub> old <sub>3</sub> man <sub>4</sub> smiled. <sub>5</sub>”

Step	Current State	Backup States	Comment	
1.	((S) 1)			1. S → NP VP
2.	((NP VP) 1)		S rewritten to NP VP	2. NP → ART N
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states	3. NP → ART ADJ N
4.	((N VP) 2)	((ART ADJ N VP) 1)		
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains	4. VP → V
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)		
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)		5. VP → V NP
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen	
9.	((NP) 4)	((ART ADJ N VP) 1)		
10.	((ART N) 4)	((ART ADJ N VP) 1) ((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails	
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again	
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3	
13.	((ADJ N VP) 2)			
14.	((N VP) 3)			
15.	((VP) 4)			
16.	((V) 4)	((V NP) 4)		
17.	(( ) 5)		success!	

## Lexicon

The → ART  
 Old → N, ADJ  
 Man → N, V  
 Smiled -> V

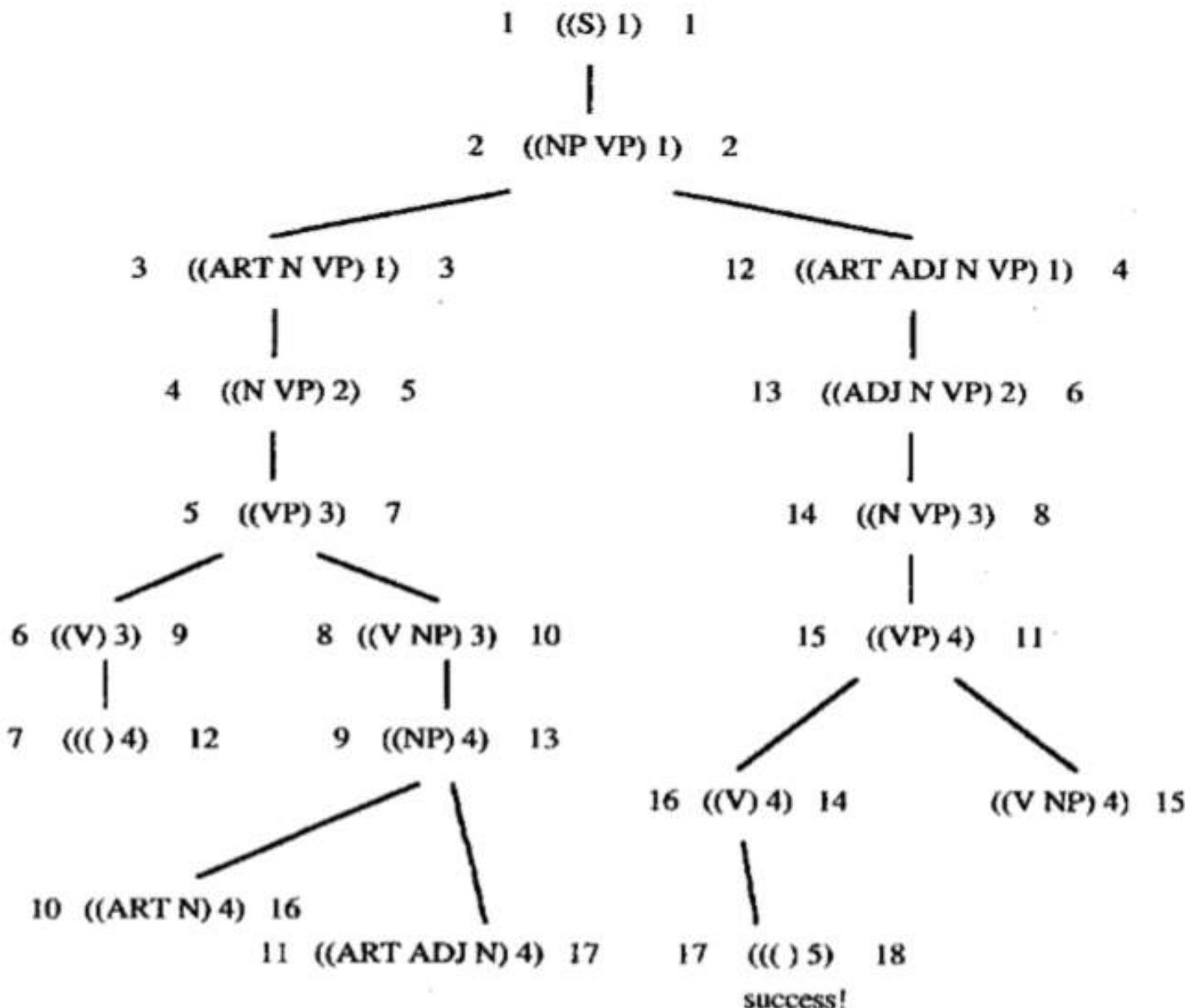


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

# Depth-first strategy vs Breadth-first strategy

---

- For a depth-first strategy, the possibilities list is a stack, yielding a last-in first-out (LIFO) strategy.
- In contrast, in a breadth-first strategy the possibilities list is manipulated as a queue, yielding a first-in first-out (FIFO) strategy
- With the depth-first strategy, one interpretation is considered and expanded until it fails; only then is the second one considered.
- With the breadth-first strategy, both interpretations are considered alternately, each being expanded one step at a time
- Many parsers built today use the depth-first strategy because it tends to minimize the number of backup states needed and thus uses less memory and requires less bookkeeping

# Bottom up parsing

---

The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules.

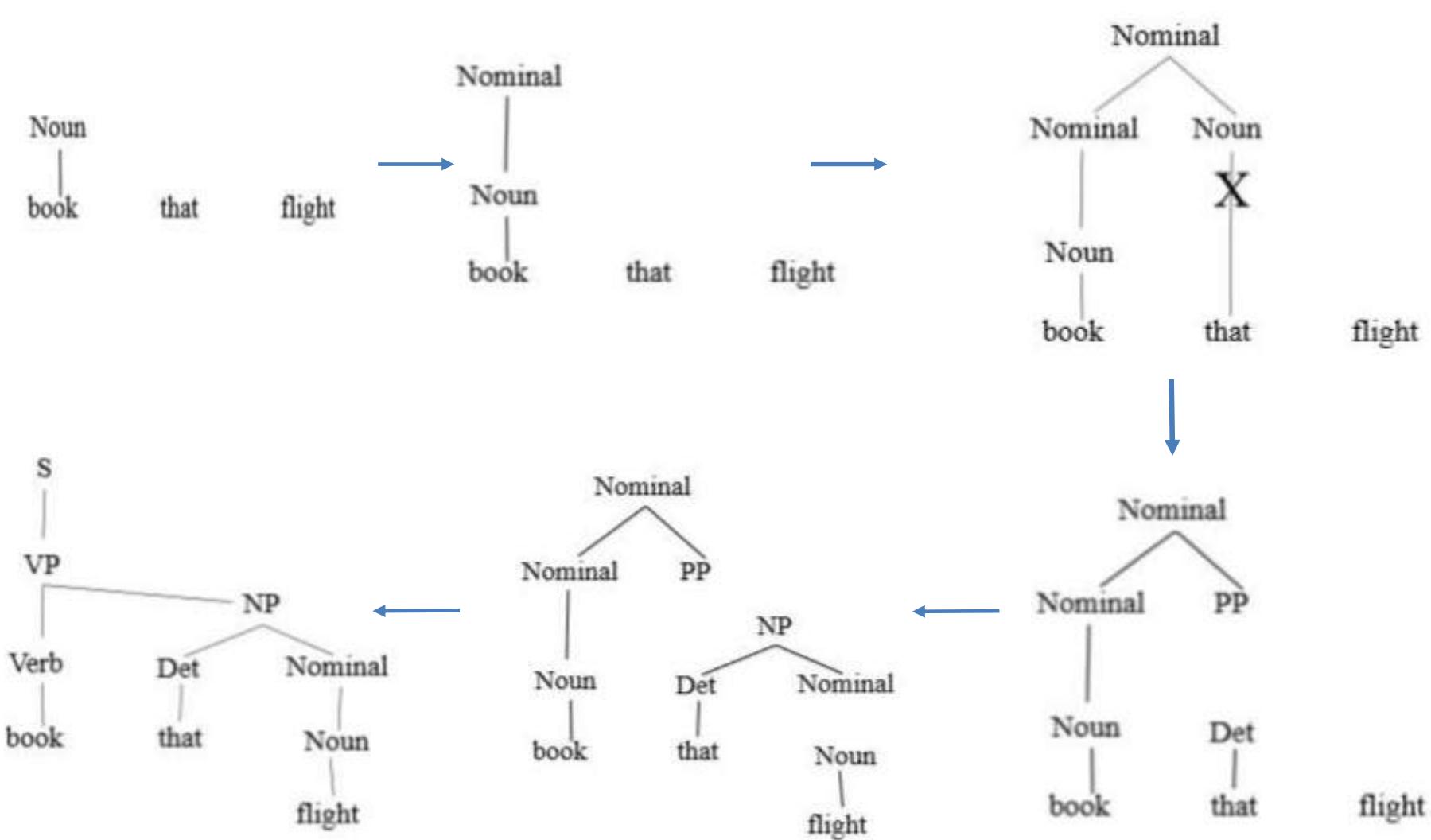
You could build a bottom-up parser simply by formulating this matching process as a search process.

The state would simply consist of a symbol list, starting with the words in the sentence.

Successor states could be generated by exploring all possible ways to

- Rewrite a word by its possible lexical categories
  - Replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol
-

# Bottom up parsing



# Top Down vs Bottom Up

---

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

# Chart parsing

---

- Parser would tend to try the same matches again and again, thus duplicating much of its work unnecessarily.
- To avoid this problem, a data structure called a chart is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

# Chart Parsing

- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to “match”
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

# Chart Parsing

- Assume you are parsing a sentence that starts with an ART.
- With this ART as the key, rules 2 and 3 are matched because they start with ART.
- To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART.
- You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record
  - 2'. NP -> ART o ADJ N
  - 3'. NP -> ART o N
- If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2 may be extended to give
  - 2''. NP -> ART ADJ o N

# Chart- active arcs

- The chart maintains the record of all the constituents derived from the sentence so far in the parse.
- Maintains the record of rules that have matched partially but are not complete. These are called the active arcs.

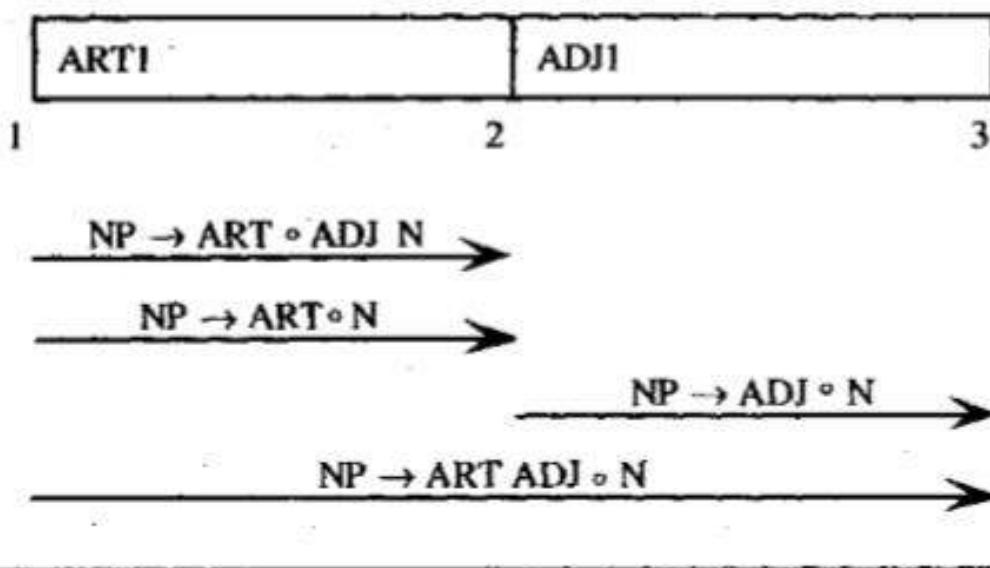


Figure 3.9 The chart after seeing an ADJ in position 2

# Chart Parsing

Extending Active Arcs with a Key:

- Each **Active Arc** has the form:  $[A \rightarrow X_1 \dots \bullet C \dots X_m](p_i, p_j)$
- A Key constituent has the form:  $C(p_i, p_j)$
- When processing the Key  $C(p_1, p_2)$ , we search the active arc list for an arc  $[A \rightarrow X_1 \dots \bullet C \dots X_m](p_0, p_1)$ , and then create a new active arc  $[A \rightarrow X_1 \dots C \bullet \dots X_m](p_0, p_2)$
- If the new active arc is a completed rule:  $[A \rightarrow X_1 \dots C \bullet](p_0, p_2)$ , then we add  $A(p_0, p_2)$  to the Agenda
- After “using” the key to extend all relevant arcs, it is entered into the Chart

# Chart Parsing

Steps in the Process:

- Input is processed left-to-right, one word at a time
1. Find all POS of word (terminal-level)
  2. Initialize Agenda with all POS of the word
  3. Pick a Key from the Agenda
  4. Add all grammar rules that start with the Key as active arcs
  5. Extend any existing active arcs with the Key
  6. Add LHS constituents of newly completed rules to the Agenda
  7. Add the Key to the Chart
  8. If Agenda not empty - goto (3), else goto (1)

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

POS of Input Words:

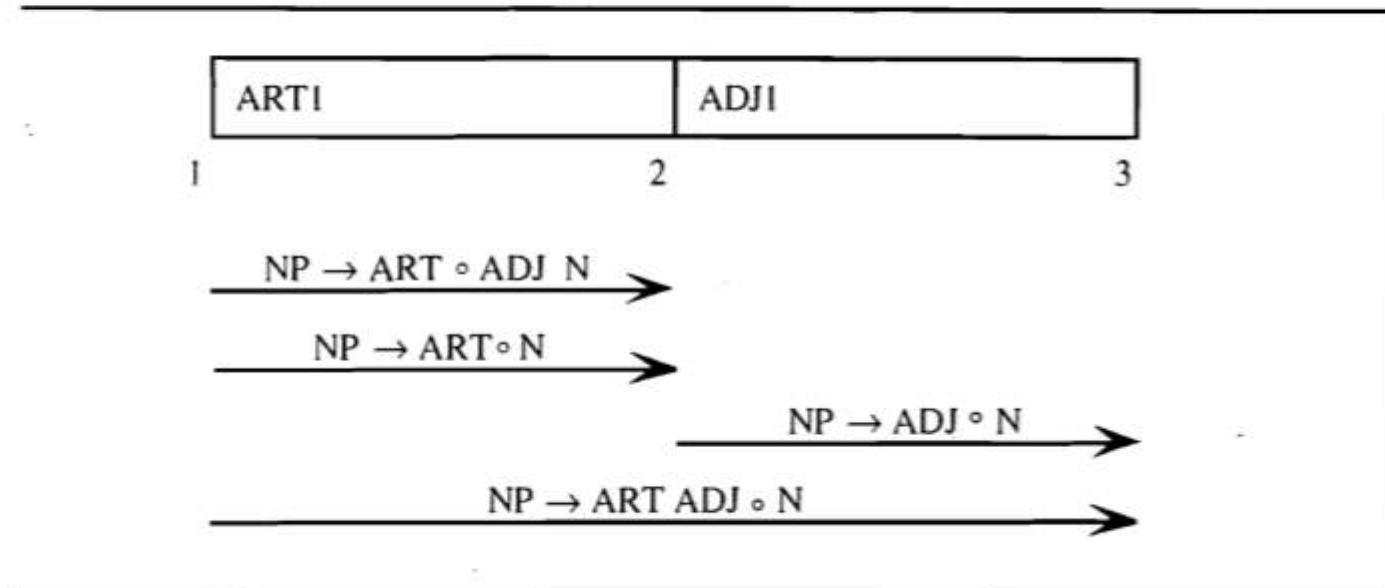
- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Chart parsing example

The input: “**x = The large can can hold the water**”

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$



**Figure 3.9** The chart after seeing an ADJ in position 2

# Chart parsing example

The input: “***x = The large can can hold the water***”

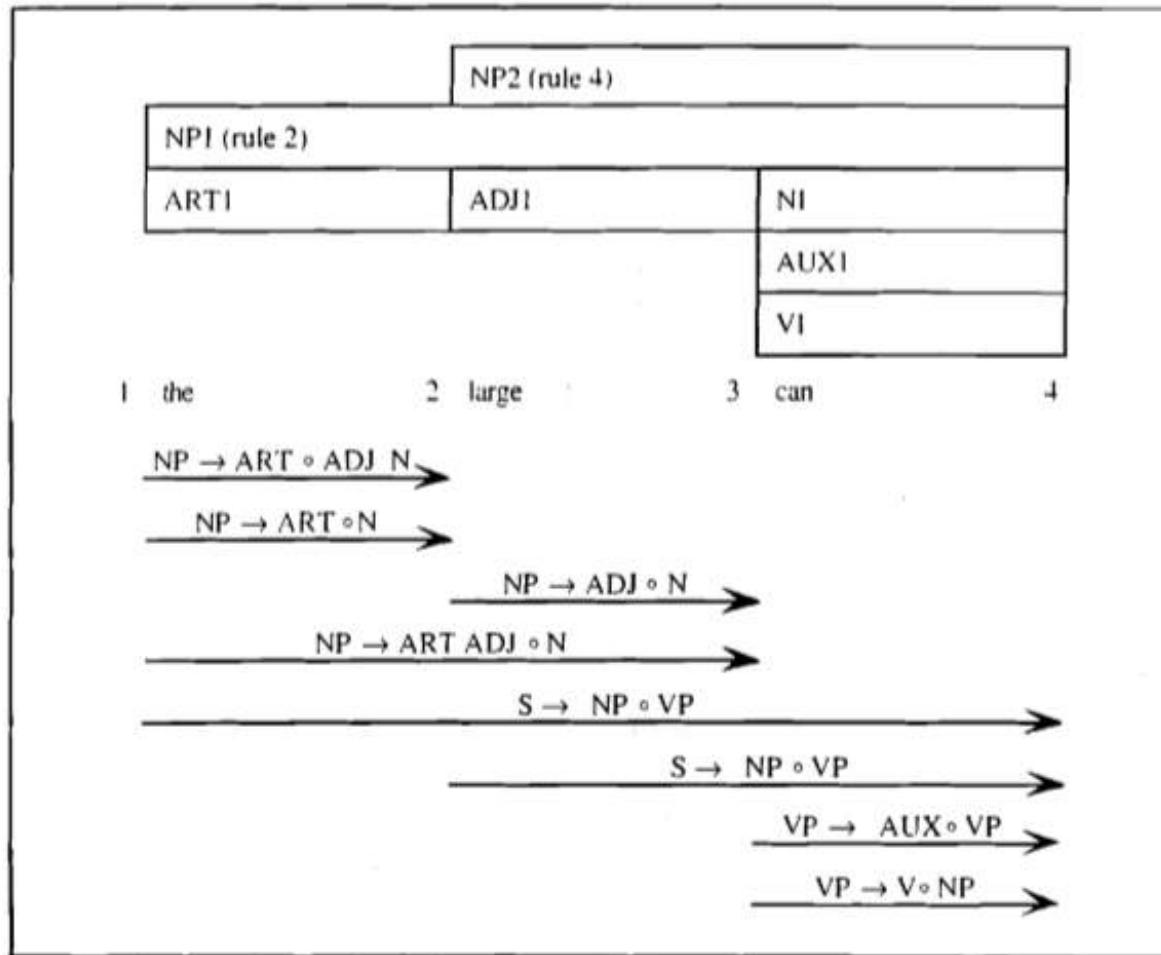


Figure 3.12 After parsing *the large can*

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

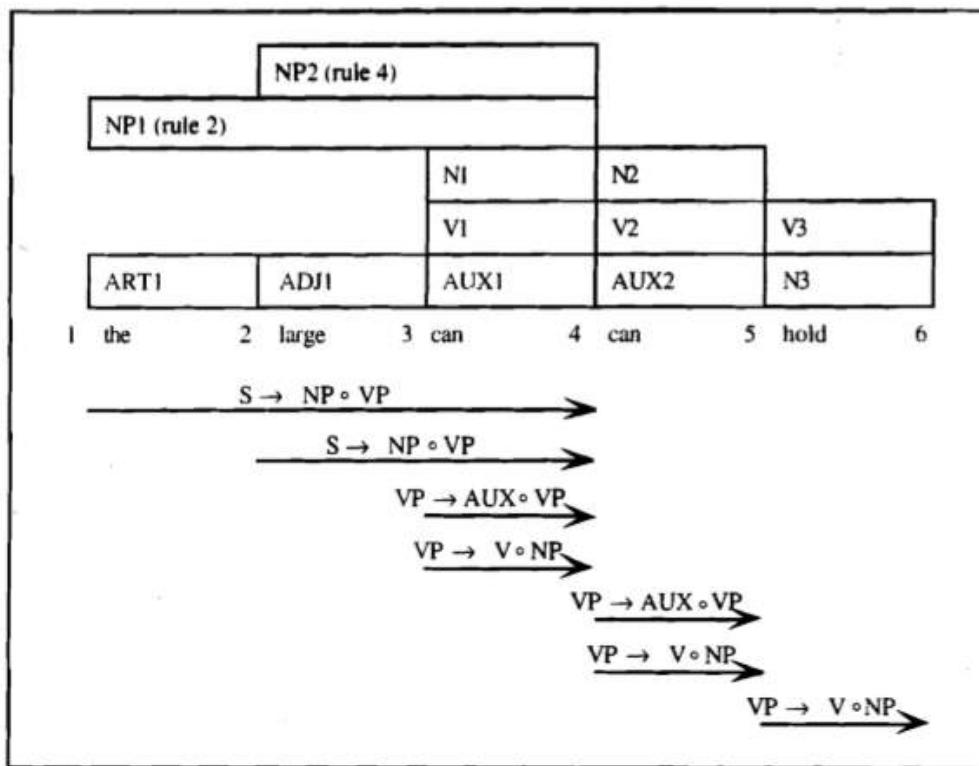
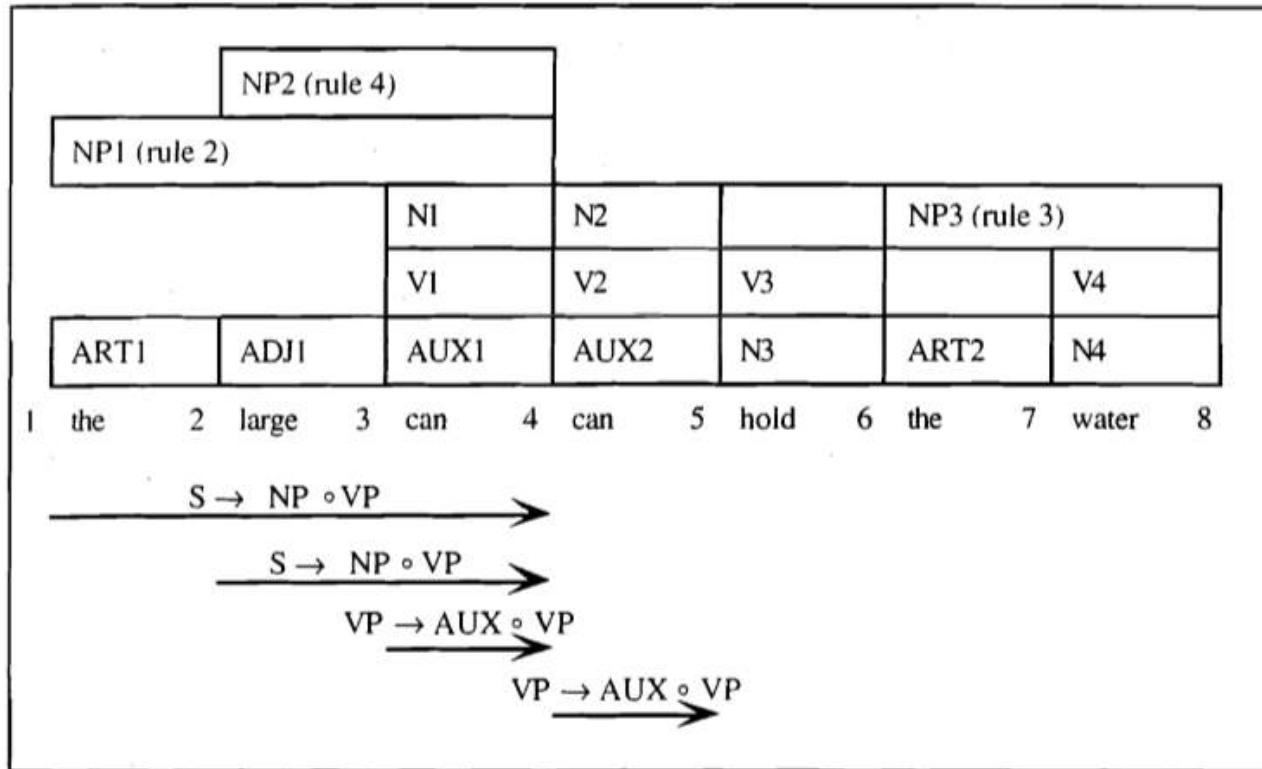


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Chart parsing example

The input: “**x = The large can can hold the water**”

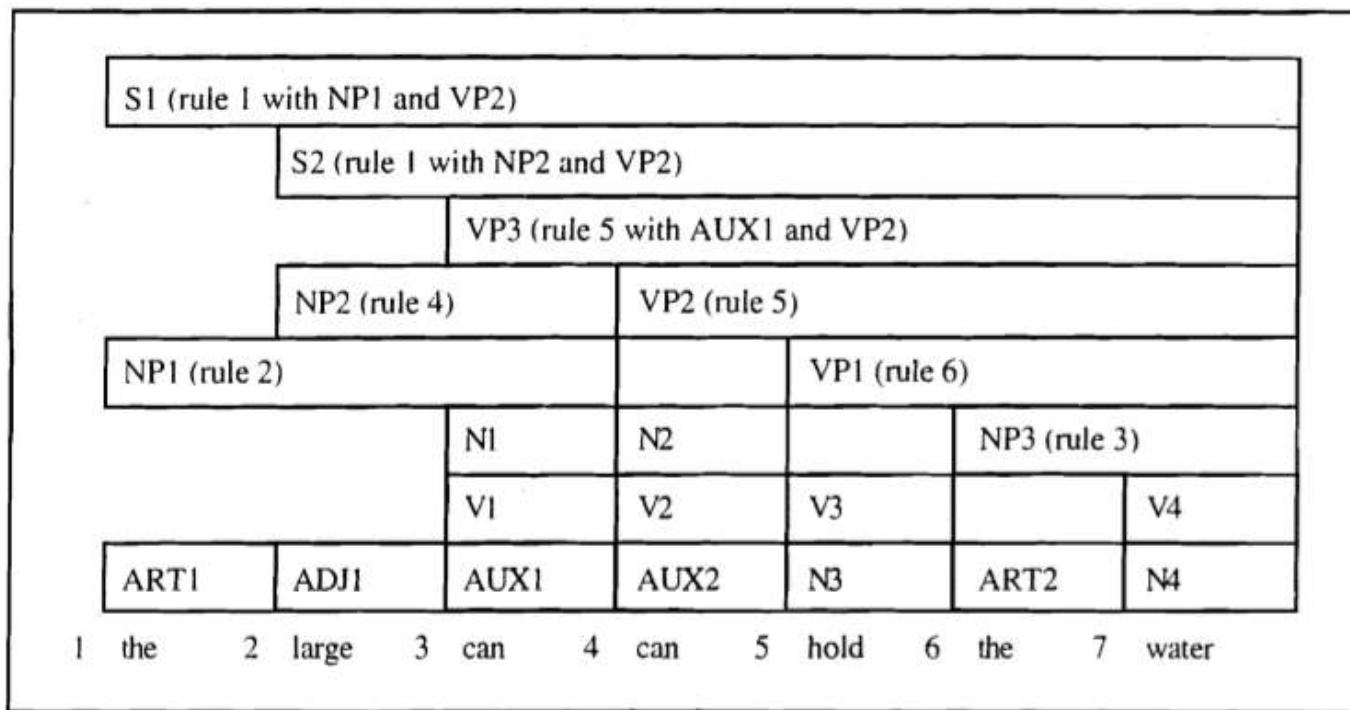


- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

Figure 3.14 The chart after all the NPs are found, omitting all but the crucial active arcs

# Final Chart

The input: “***x = The large can can hold the water***”



- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

Figure 3.15 The final chart

# Top down chart parsing

---

- Top-down methods have the advantage of being highly predictive.
- A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered.

# Top down arc induction

---

- Consider this new algorithm operating with the same grammar on “The large can can hold the water.”
- In the initialization stage, an arc labeled  $S \rightarrow o$  NP VP is added. Then, active arcs for each rule that can derive an NP are added:  $NP \rightarrow o$  ART ADJ N,  $NP \rightarrow o$  ART N,

## Top-Down Arc Introduction Algorithm

To add an arc  $S \rightarrow C_1 \dots o C_1 \dots C_n$  ending at position j, do the following:

For each rule in the grammar of form  $C_i \rightarrow X_1 \dots X_k$ , recursively add the new arc  $C_i \rightarrow o X_1 \dots X_k$  from position j to j.

# Top down chart parsing

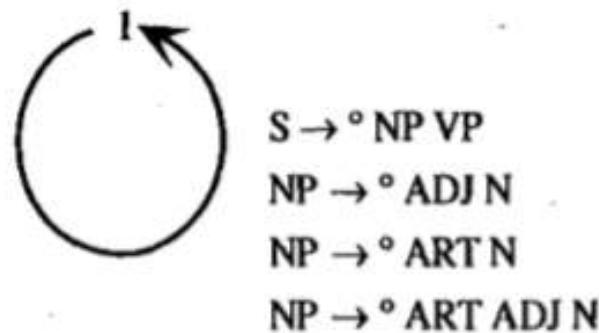
---

Initialization: For every rule in the grammar of form  $S \rightarrow X_1 \dots X_k$ , add an arc labeled  $S \rightarrow o | X_1 \dots X_k$  using the arc introduction algorithm.

Parsing: Do until there is no input left:

1. If the agenda is empty, look up the interpretations of the next word and add them to the agenda.
2. Select a constituent from the agenda (call it constituent C).
3. Using the arc extension algorithm, combine C with every active arc on the chart. Any new constituents are added to the agenda.
4. For any active arcs created in step 3, add them to the chart using the top-down arc introduction algorithm.

# Top down chart parsing example



**Figure 3.23** The initial chart

# Top down chart parsing example

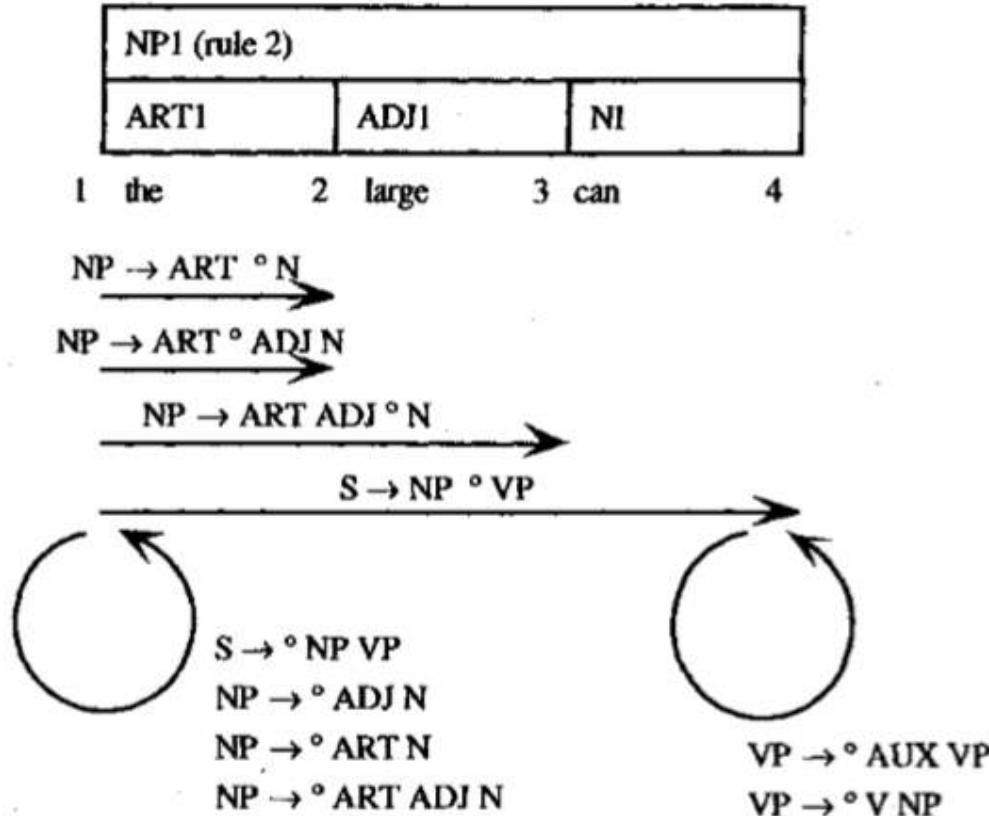


Figure 3.24 The chart after building the first NP

# Top down chart parsing example

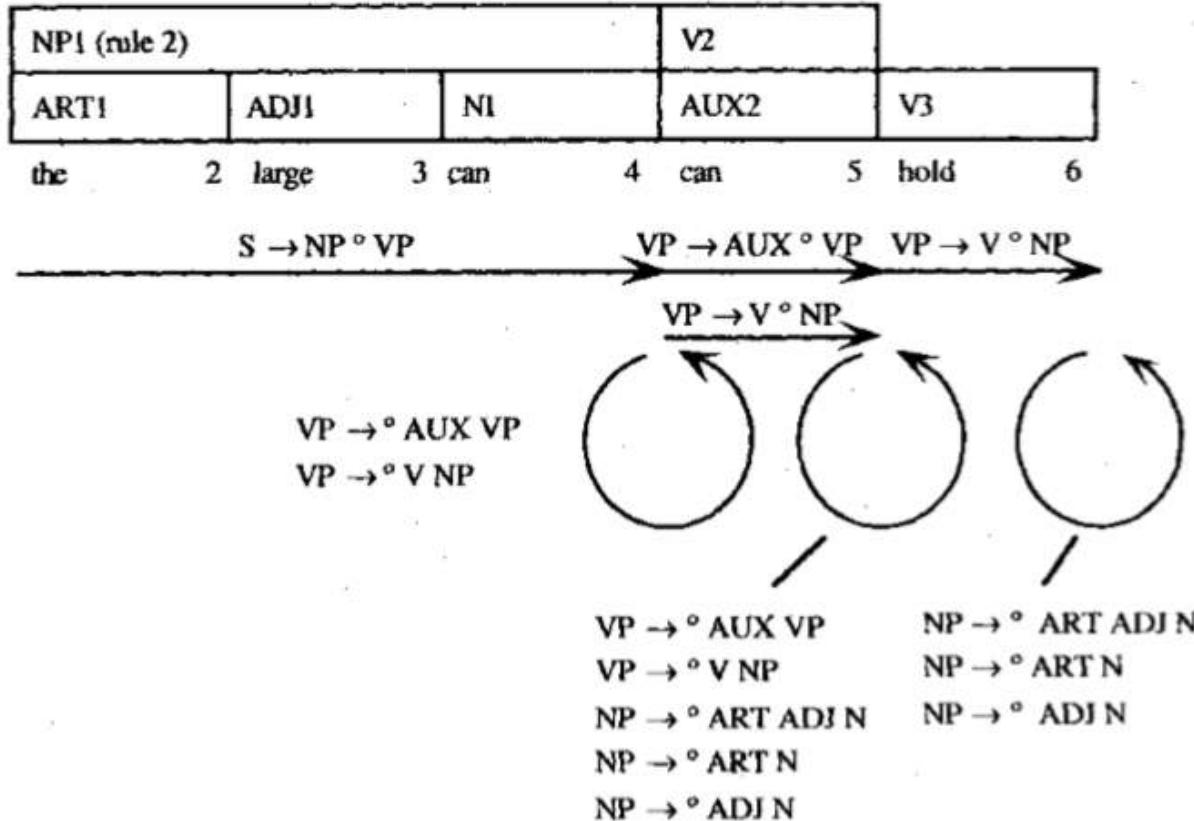


Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

# Top down chart parsing example

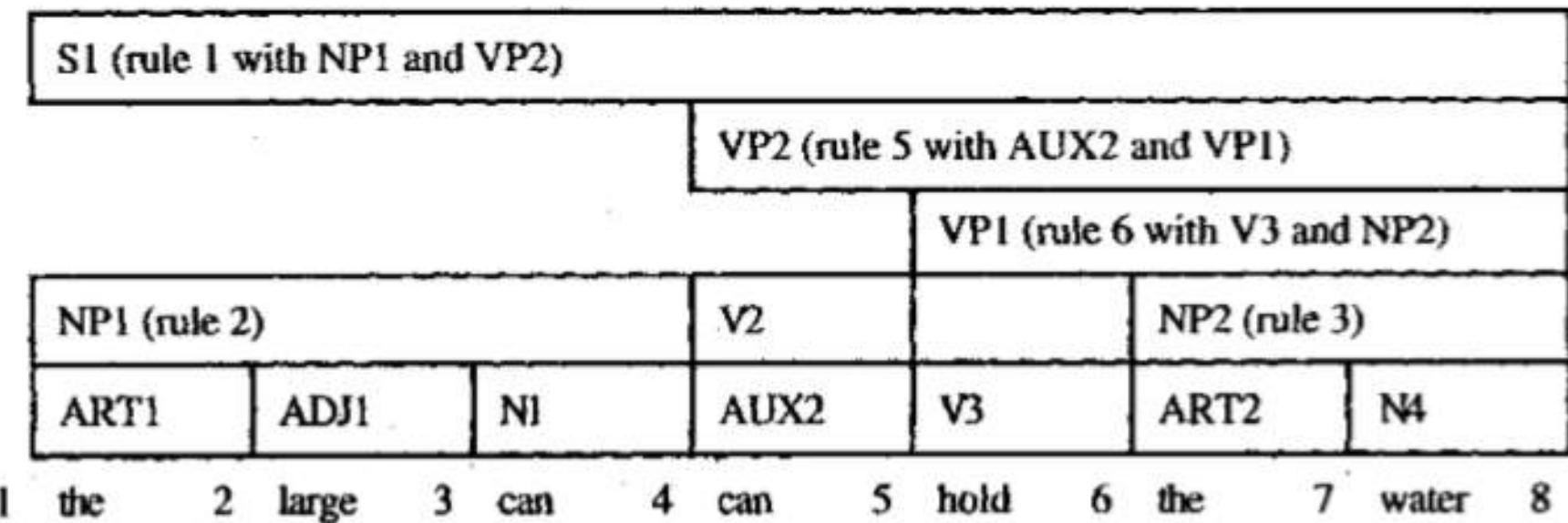


Figure 3.26 The final chart for the top-down filtering algorithm

# Chart Parsing

- When a chart parser begins parsing a text, it creates a new (empty) chart, spanning the text.
- It then incrementally adds new edges to the chart.
- A set of "chart rules" specifies the conditions under which new edges should be added to the chart.
- Once the chart reaches a stage where none of the chart rules adds any new edges, parsing is complete.

## Advantages of Chart Parsing

- No repeated computation of same sub problem
- Deals well with left-recursive grammars
- Deals well with ambiguity
- No backtracking necessary

## Stanford Parser

Please enter a sentence to be parsed:

```
I love natural language processing class
```

Language: English ▾      Sample Sentence

Parse

### Your query

*I love natural language processing class*

### Tagging

I/PRP love/VBP natural/JJ language/NN processing/NN class/NN

### Parse

```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBP love)
      (NP (JJ natural) (NN language) (NN processing) (NN class)))))
```

---

# Parsing Implementation Demo

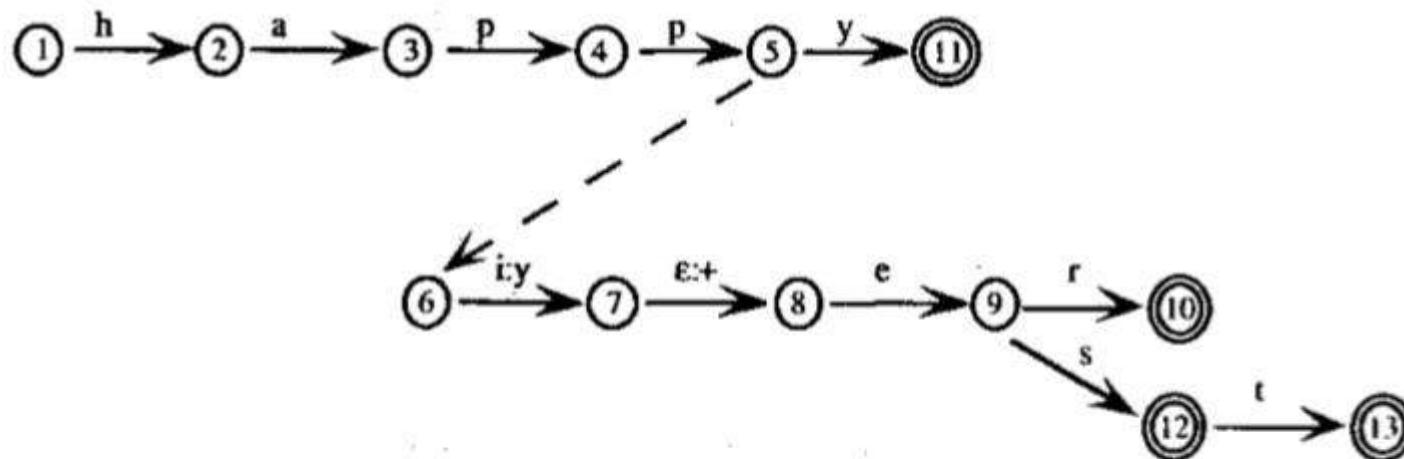
# Morphological Processing

---

- Not only are there a large number of words, but each word may combine with affixes to produce additional related words.
- One way to address this problem is to preprocess the input sentence into a sequence of morphemes.
- A word may consist of single morpheme, but often a word consists of a root form plus an affix (prefix or suffix).
- For instance, the word eaten consists of the root form eat and the suffix -en, which indicates the past participle form.
- Without any pre-processing, a lexicon would have to list all the forms of eat, including eats, eating, ate, and eaten.
- With preprocessing, there would be one morpheme eat that may combine with suffixes such as -ing, -s, and -en, and one entry for the irregular form ate. Thus the lexicon would only need to store two entries (eat and ate) rather than four.

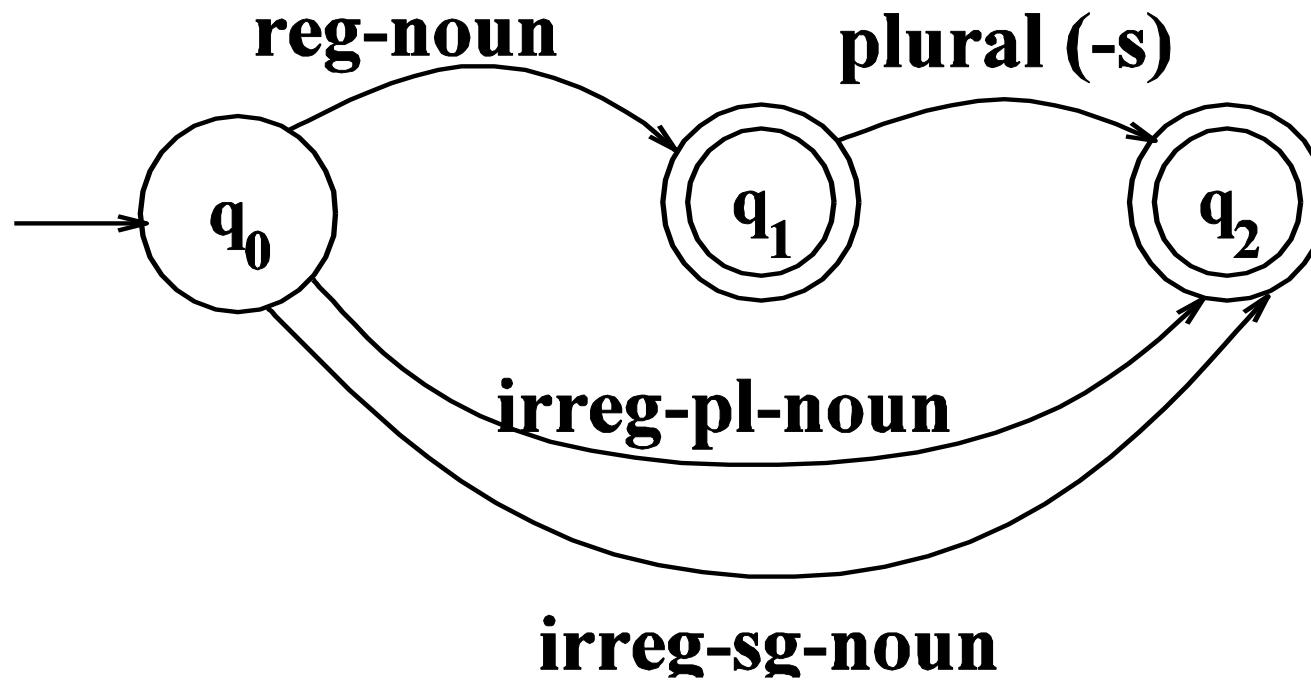
# Finite State Models and Morphological Processing example

- Word happiest breaks down into the root form happy and the suffix -est, and thus does not need a separate entry in the lexicon
- An arc in an FST is labeled with a pair of symbols.
- For example, an arc labeled i:y could only be followed if the current input is the letter i and the output is the letter y.
- FSTs can be used to concisely represent the lexicon and to transform the surface form of words into a sequence of morphemes.



**Figure 3.27** A simple FST for the forms of *happy*

# Simple Rules



# Parsing/Generation

---

No Image Usually if we find some string in the language we need to find the structure in it (**parsing**)

No Image Or we have some structure and we want to produce a surface form  
**(production/generation)**

No Image Example

No Image From “cats” to “cat +N +PL” and back

→**Morphological analysis**

# FSTs

*Lexical*



*Surface*



# Ambiguity

---

No  
Image

What's the right parse for

No  
Image

Unionizable

No  
Image

Union-ize-able

No  
Image

Un-ion-ize-able

No  
Image

Each represents a valid path through the derivational morphology machine.

# Lexicon-only Morphology

- The lexicon lists all surface level and lexical level pairs
- No rules ...
- Analysis/Generation is easy
- Very large for English
- What about
  - Arabic or
  - Turkish or
  - Chinese?

acclaim	acclaim	\$N\$
acclaim	acclaim	\$V+0\$
acclaimed	acclaim	\$V+ed\$
acclaimed	acclaim	\$V+en\$
acclaiming	acclaim	\$V+ing\$
acclaims	acclaim	\$N+s\$
acclaims	acclaim	\$V+ss\$
acclamation	acclamation	\$N\$
acclamations	acclamation	\$N+s\$
acclimate	acclimate	\$V+0\$
acclimated	acclimate	\$V+ed\$
acclimated	acclimate	\$V+en\$
acclimates	acclimate	\$V+ss\$
acclimating	acclimate	\$V+ing\$

# Stemming

---

No  
Image

Sometimes you just need to know the stem of a word

No  
Image

In fact you may not even care if you get the right stem, as long as you get a consistent string.

No  
Image

This is **stemming**... it most often shows up in IR applications

# Stemming

- Stemming is the process of reducing inflectional form of words to their root form.
  - Example words like operation, operations, operational, operating can be reduced to operati (root word)
- Stemming is crude form of Normalization in which the suffixes are removed.
- The advantage of suffix stripping is to **reduce the total number of terms** in the inverted index resulting in a **smaller size and complexity of the data** in the system.

# Stemming in IR

---

No  
Image

Run a stemmer on the documents to be indexed

No  
Image

Run a stemmer on users queries

No  
Image

Match

No  
Image

This is basically a form of hashing

No  
Image

Example: Computerization

No  
Image

ization -> -ize computerize

No  
Image

ize -> ε computer

# Porter

No  
Image

No lexicon needed

No  
Image

Basically a set of staged sets of rewrite rules  
that strip suffixes

- Doesn't guarantee that the resulting stem is  
really a stem (see first bullet)
- Lack of guarantee doesn't matter for IR

# Porter Stemmer

- A consonant in a word is a letter other than A, E, I, O or U
- Any letter not a consonant is a Vowel.
- All the words in English are of the form C(VC)<sup>m</sup>V where m is measure of any word or word part when represented in this form (VC).

Examples:

$m=0$  TR, EE, TREE, Y, BY.

$m=1$  TROUBLE, OATS, TREES

$m=2$  TROUBLES, PRIVATE, OATEN, ORRERY.

# Porter Stemmer (contd..)

---

- The rules for removing a suffix will be given in the form (condition) S1 -> S2
- This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2.

(m > 1) EMENT ->

---

# Porter Stemmer (contd..)

## Step 1a

SSES -> SS

caresses -> caress

IES -> I

ponies -> poni

SS -> SS

ties -> ti

S ->

caress -> caress

cats -> cat

- Step 1 deals with plurals and past participles.

## Step 1b

(m>0) EED -> EE

feed -> feed

(\*v\*) ED ->

agreed -> agree

plastered -> plaster

bled -> bled

(\*v\*) ING ->

motoring -> motor

sing -> sing

- The subsequent steps are much more straightforward.

## Step 1c

(\*v\*) Y -> I

happy -> happi

sky -> sky

# Porter Stemmer

## Derivational Morphology I: Multiple Suffixes

(m>0) ATIONAL	->	ATE	relational	->	relate
(m>0) TIONAL	->	TION	conditional	->	condition
			rational	->	rational
(m>0) ENCI	->	ENCE	valenci	->	valence
(m>0) ANCI	->	ANCE	hesitanci	->	hesitance
(m>0) IZER	->	IZE	digitizer	->	digitize
(m>0) ABLI	->	ABLE	conformabli	->	conformable
(m>0) ALLI	->	AL	radicalli	->	radical
(m>0) ENTLI	->	ENT	differentli	->	different
(m>0) ELI	->	E	vileli	->	vile
(m>0) OUSLI	->	OUS	analogousli	->	analogous
(m>0) IZATION	->	IZE	vietnamization	->	vietnamize
(m>0) ATION	->	ATE	predication	->	predicate
(m>0) ATOR	->	ATE	operator	->	operate
(m>0) ALISM	->	AL	feudalism	->	feudal
(m>0) IVENESS	->	IVE	decisiveness	->	decisive
(m>0) FULNESS	->	FUL	hopefulness	->	hopeful
(m>0) OUSNESS	->	OUS	callousness	->	callous
(m>0) ALITI	->	AL	formaliti	->	formal
(m>0) IVITI	->	IVE	sensitiviti	->	sensitive
(m>0) BILITI	->	BLE	sensibiliti	->	sensible

# Effect of stemming

- Suffix stripping of a vocabulary of 10,000 words

Number of words reduced in step 1: 3597

"	2:	766
"	3:	327
"	4:	2424
"	5:	1373

Number of words not reduced: 3650

- The resulting vocabulary of stems contained 6370 distinct entries.
- Thus the suffix stripping process reduced the size of the vocabulary by about one third.

# Porter Stemmer

No  
Image

## Errors of Omission

No Image	European	Europe
No Image	analysis	analyzes
No Image	matrices	matrix
No Image	noise	noisy
No Image	explain	explanation

No  
Image

## Errors of Commission

No Image	organization	organ
No Image	doing	doe
No Image	generalization	generic
No Image	numerical	numerous
No Image	university	universe

# LEMMATIZATION

- Process through which several different forms of the same word are mapped to one single form, which we can call the root form or the base form.
- In more technical terms, the root form is called a lemma.
- By reducing the number of forms a word can take, we make sure that we reduce our data space and that we don't have to check every single form of a word.
- It helps us ignore morphological variations on a single word. Lemmatization brings context to the words.
- So it goes a step further by linking words with similar meaning to one word.
- For example if a paragraph has words like cars, trains and automobile, then it will link all of them to automobile.

# Grammars and Logic Programming

---

- "A sequence of words is a legal S if it begins with a legal NP that is followed by a legal VP."
- If you number each word in a sentence by its position, you can restate this rule as: "There is an S between position p1 and p3, if there is a position p2 such that there is an NP between p1 and p2 and a VP between p2 and p3."
- In PROLOG this would be the following axiom, where variables are indicated as atoms with an initial capitalized letter:  $s(P1, P3) \rightarrow np(P1, P2), vp(P2, P3)$

# Prolog Example

---

- To set up the process, add axioms listing the words in the sentence by their position.
  - For example, the sentence John ate the apple is described by
    - word(John, 1, 2)
    - word(ate, 2, 3)
    - word(the, 3, 4)
    - word(apple, 4, 5)
  - isart(the)
  - isname(john)
  - isverb(ate)
  - isnoun(apple)
-

# Prolog based grammar

---

1. **s(P1, P3) :- np(P1, P2), vp(P2, P3)**
  
2. **np(P1, P3) :- art(P1, P2), n(P2, P3)**
  
3. **np(P1, P3) :- name(P1, P3)**
  
4. **pp(P1, P3) :- p(P1, P2), np(P2, P3)**
  
5. **vp(P1, P2) :- v(P1, P2)**
  
6. **vp(P1, P3) :- v(P1, P2), np(P2, P3)**
  
7. **vp(P1, P3) :- v(P1, P2), pp(P2, P3)**

# References

---

- <http://www.ai.mit.edu/courses/6.863/lecture7-03.pdf>
- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin
- Natural language understanding by James Allen
- <https://nlp.stanford.edu/software/lex-parser.shtml>
- <https://www.nltk.org/api/nltk.parse.html>



**Thank you for your time!!**

---

**Extra slides if time permits 😊**

---

# Vector Space Model

---

Documents are represented as vectors in term space

- Terms are usually *stems*
- Documents represented by binary or weighted vectors of terms

Queries represented the same as documents

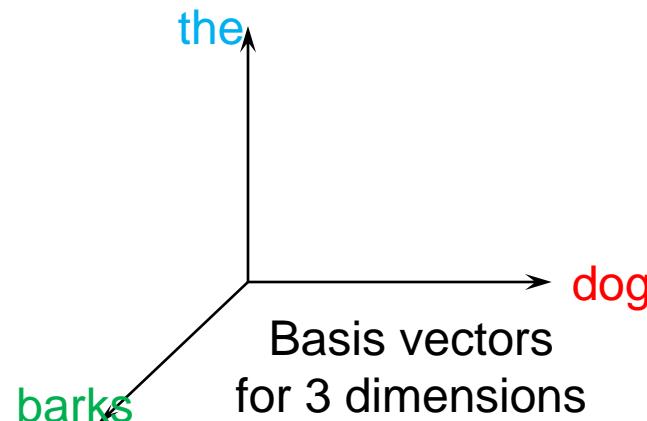
Query and Document weights are based on length and direction of their vector

A vector distance measure between the query and documents is used to rank retrieved documents

# Vector Space Representation from linear algebra perspective

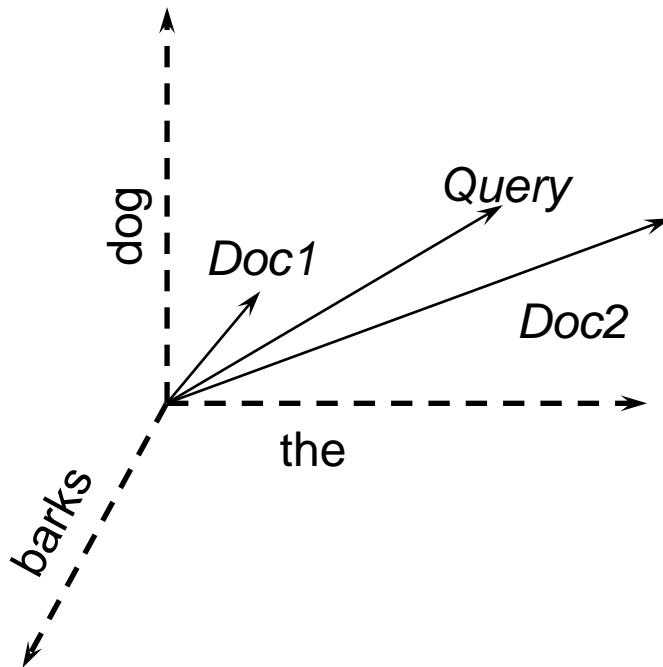


- Formally, a vector space is defined by a **set of linearly independent basis vectors**.
- Basis vectors:
  - correspond to the *dimensions* or *directions* in the vector space;
  - determine what can be described in the vector space; and
  - must be *orthogonal*, or *linearly independent*, i.e. a value along one dimension implies nothing about a value along another.



# Vector Coefficients

- How to represent the documents and queries?



Doc1: the dog barks <1 1 1>  
Doc2: the dog dog barks barks  
barks <1 2 3>

Query: the dog dog barks barks  
<1 2 2>

# Vector Space Similarity

## Sim(X,Y)

Inner product

(# nonzero dimensions)

Dice coefficient

(Length normalized  
Inner Product)

Cosine coefficient

(like Dice, but lower  
penalty with diff # features)

Jaccard coefficient

(like Dice, but penalizes  
low overlap cases)

## Binary Term Vectors

$$|X \cap Y|$$

$$\frac{2|X \cap Y|}{|X| + |Y|}$$

$$\frac{|X \cap Y|}{\sqrt{|X|} \sqrt{|Y|}}$$

$$\frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

## Weighted Term Vectors

$$\sum x_i \cdot y_i$$

$$\frac{2 \sum x_i y_i}{\sum x_i^2 + \sum y_i^2}$$

$$\frac{\sum x_i \cdot y_i}{\sqrt{\sum x_i^2} \cdot \sqrt{\sum y_i^2}}$$

$$\frac{\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2 - \sum x_i \cdot y_i}$$

# Term Frequency

Frequency of occurrence for the term in each document is included in the vector

<i>docs</i>	<i>t1</i>	<i>t2</i>	<i>t3</i>
D1	2	0	3
D2	1	0	0
D3	0	4	7
D4	3	0	0
D5	1	6	3
D6	3	5	0
D7	0	8	0
D8	0	10	0
D9	0	0	1
D10	0	3	5
D11	4	0	1

# Inverse document frequency

$df_t$  is the document frequency of  $t$ : the number of documents that contain  $t$

- df is a measure of the informativeness of  $t$

We define the idf (inverse document frequency) of  $t$  by

$$idf_t = \log N/df_t$$

We use  $\log N/df_t$  instead of  $N/df_t$  to “dampen” the effect of idf

# Inverse Document Frequency

---

IDF provides high values for rare words and low values for common words

$$\log\left(\frac{10000}{10000}\right) = 0$$

For a collection of 10000 documents (N = 10000)

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

# Assigning Weights

---

**tf\*idf measure:**

- Term frequency (tf)
- Inverse document frequency (idf)

**Goal:**

Assign a tf\*idf weight to each term in each document

# Simple tf\*idf

$$w_{ik} = tf_{ik} * \log(N/n_k)$$

$T_k$  = term  $k$  in document  $D_i$

$tf_{ik}$  = frequency of term  $T_k$  in document  $D_i$

$idf_k$  = inverse document frequency of term  $T_k$  in  $C$

$N$  = total number of documents in the collection  $C$

$n_k$  = the number of documents in  $C$  that contain  $T_k$

$$idf_k = \log\left(\frac{N}{n_k}\right)$$



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,

IIT Bombay-Monash University Australia

Associate Professor, BITS Pilani

[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



## **Session 6 -Probabilistic Context Free Grammar and Parsing**

**Date – 3<sup>rd</sup> October 2020**

**Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin and many others who made their course materials freely available online.

# Session Content



(Ref: Chapter 14 Jurafsky and Martin)

---

- CKY Parsing
- Probabilistic Context-Free Grammars
- PCFG for disambiguation
- Probabilistic CKY Parsing of PCFGs
- Ways to Learn PCFG Rule Probabilities
- Probabilistic Lexicalized CFGs
- Evaluating Parsers
- Problems with PCFGs

# Some funny examples

---

- Policeman to little boy: “We are looking for a thief with a bicycle.” Little boy: “Wouldn’t you be better using your eyes.”
- Why is the teacher wearing sun-glasses.  
Because the class is so bright.

# Ambiguity is Explosive

- “I saw the man with the telescope”: 2 parses
- “I saw the man on the hill with the telescope.”: 5 parses
- “I saw the man on the hill in Texas with the telescope”: 14 parses
- “I saw the man on the hill in Texas with the telescope at noon.”: 42 parses
- “I saw the man on the hill in Texas with the telescope at noon on Monday” 132 parses

# CKY parsing

---

Classic, bottom-up dynamic programming algorithm (Cocke-Kasami-Younger).

Requires input grammar based on Chomsky Normal Form

- A CNF grammar is a Context-Free Grammar in which:
  - Every rule LHS is a non-terminal
  - Every rule RHS consists of either a single terminal or two non-terminals.
  - Examples:
    - $A \rightarrow BC$
    - $NP \rightarrow N PP$
    - $A \rightarrow a$
    - Noun  $\rightarrow$  man
  - But not:
    - $NP \rightarrow \text{the } N$
    - $S \rightarrow VP$

# Chomsky Normal Form

- Any CFG can be re-written in CNF, without any loss of expressiveness.
  - That is, for any CFG, there is a corresponding CNF grammar which accepts exactly the same set of strings as the original CFG.

# Converting a CFG to CNF

- To convert a CFG to CNF, we need to deal with three issues:
  1. Rules that mix terminals and non-terminals on the RHS
    - E.g.  $NP \rightarrow \text{the Nominal}$
  2. Rules with a single non-terminal on the RHS (called unit productions)
    - E.g.  $NP \rightarrow \text{Nominal}$
  3. Rules which have more than two items on the RHS
    - E.g.  $NP \rightarrow \text{Det Noun PP}$

# Converting a CFG to CNF

1. Rules that mix terminals and non-terminals on the RHS
  - E.g.  $NP \rightarrow \text{the Nominal}$
  - Solution:
    - Introduce a dummy non-terminal to cover the original terminal
      - E.g.  $Det \rightarrow \text{the}$
    - Re-write the original rule:
      - $NP \rightarrow Det\ Nominal$
      - $Det \rightarrow \text{the}$

# Converting a CFG to CNF

2. Rules with a single non-terminal on the RHS (called unit productions)
  - E.g.  $NP \rightarrow Nominal$
  - Solution:
    - Find all rules that have the form  $Nominal \rightarrow \dots$ 
      - $Nominal \rightarrow Noun\ PP$
      - $Nominal \rightarrow Det\ Noun$
    - Re-write the above rule several times to eliminate the intermediate non-terminal:
      - $NP \rightarrow Noun\ PP$
      - $NP \rightarrow Det\ Noun$
  - Note that this makes our grammar “flatter”

# Converting a CFG to CNF

3. Rules which have more than two items on the RHS
  - E.g.  $NP \rightarrow Det\ Noun\ PP$
  - Solution:
    - Introduce new non-terminals to spread the sequence on the RHS over more than 1 rule.
      - $Nominal \rightarrow Noun\ PP$
      - $NP \rightarrow Det\ Nominal$

# CNF Grammar

- If we parse a sentence with a CNF grammar, we know that:
  - Every phrase-level non-terminal (above the part of speech level) will have exactly 2 daughters.
    - $\text{NP} \rightarrow \text{Det N}$
  - Every part-of-speech level non-terminal will have exactly 1 daughter, and that daughter is a terminal:
    - $\text{N} \rightarrow \text{lady}$

# Recognising strings with CKY

Example input: The flight includes a meal.

The CKY algorithm proceeds by:

1. Splitting the input into words and indexing each position.  
(0) the (1) flight (2) includes (3) a (4) meal (5)
2. Setting up a table. For a sentence of length  $n$ , we need  $(n+1)$  rows and  $(n+1)$  columns.
3. Traversing the input sentence left-to-right
4. Use the table to store constituents and their span.

# CKY example

- $S \rightarrow NP\ VP$
- $NP \rightarrow Det\ N$
- $VP \rightarrow V\ NP$
- $V \rightarrow includes$
- $Det \rightarrow the$
- $Det \rightarrow a$
- $N \rightarrow meal$
- $N \rightarrow flight$

# CKY example

Rule: Det → *the*

[0,1] for "the"



	1	2	3	4	5
0	Det				S
1					
2					
3					
4					

the            flight            includes            a            meal

# CKY example

Rule1: Det  $\rightarrow$  *the*  
Rule 2: N  $\rightarrow$  flight

[0,1] for "the"

[1,2] for "flight"

	1	2	3	4	5
0	Det				S
1		N			
2					
3					
4					

the

flight

includes

a

meal

# CKY example

Rule1: Det → *the*  
Rule 2: N → flight  
Rule 3: NP → Det N

	1	2	3	4	5
0	Det	NP			S
1		N			
2					
3					
4					

the                    flight                    includes                    a                    meal

[0,2] for "the flight"  
[0,1] for "the"      [1,2] for "flight"

# CKY: lexical step ( $j = 1$ )

- *The flight includes a meal.*

Lexical lookup

- Matches Det → the

	1	2	3	4	5
0	Det				
1					
2					
3					
4					
5					

# CKY: lexical step ( $j = 2$ )

- *The flight includes a meal.*

Lexical lookup

- Matches N → flight

	1	2	3	4	5
0	Det				
1		N			
2					
3					
4					
5					

# CKY: syntactic step ( $j = 2$ )

- *The flight includes a meal.*

Syntactic lookup:

- look backwards and see if there is any rule that will cover what we've done so far.

	1	2	3	4	5
0	Det	NP			
1		N			
2					
3					
4					
5					

# CKY: lexical step ( $j = 3$ )

- *The flight includes a meal.*

Lexical lookup

- Matches V → includes

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3					
4					
5					

# CKY: lexical step ( $j = 3$ )

- *The flight includes a meal.*

Syntactic lookup

- There are no rules in our grammar that will cover Det, NP, V

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>					
<b>4</b>					
<b>5</b>					

# CKY: lexical step ( $j = 4$ )

- *The flight includes a meal.*

Lexical lookup

- Matches Det → a

	1	2	3	4	5
0	Det	NP			
1		N			
2			V		
3				Det	
4					
5					

# CKY: lexical step ( $j = 5$ )

- *The flight includes a meal.*

Lexical lookup

- Matches N → meal

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	
<b>4</b>					N

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

- We find that we have  
 $NP \rightarrow Det\ N$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	NP
<b>4</b>					N

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

- We find that we have  
 $VP \rightarrow V \ NP$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		VP
<b>3</b>				Det	NP
<b>4</b>					N

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

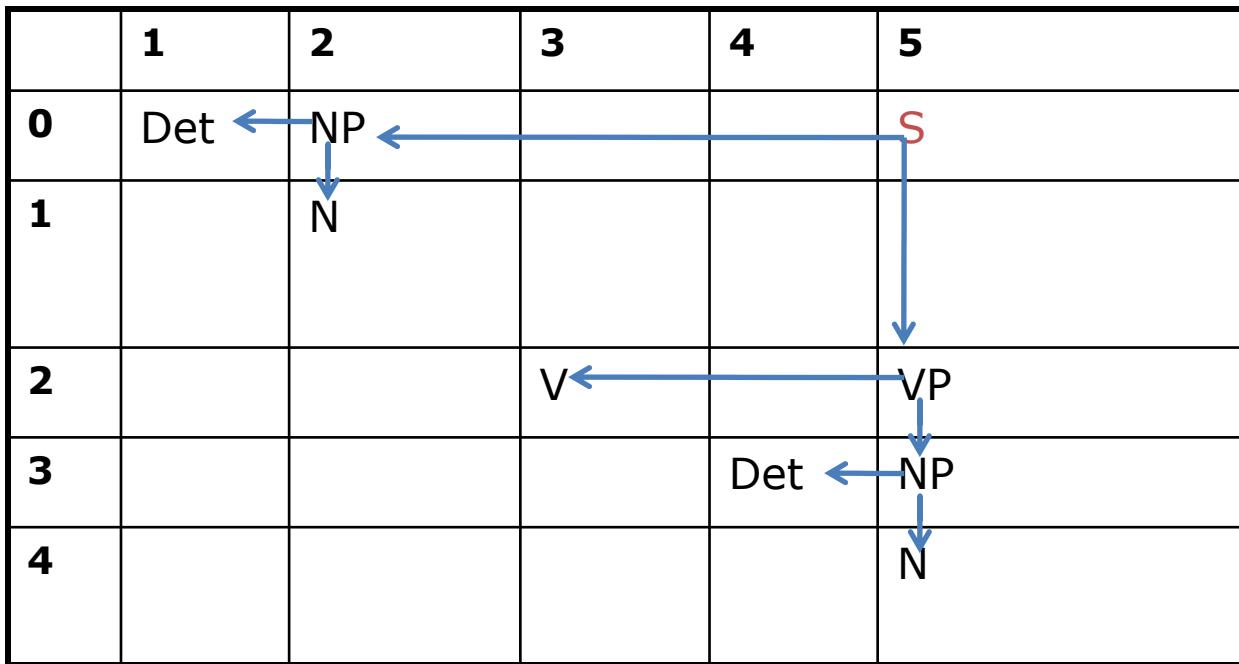
- We find that we have S  
→ NP VP

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			S
<b>1</b>		N			
<b>2</b>			V		VP
<b>3</b>				Det	NP
<b>4</b>					N

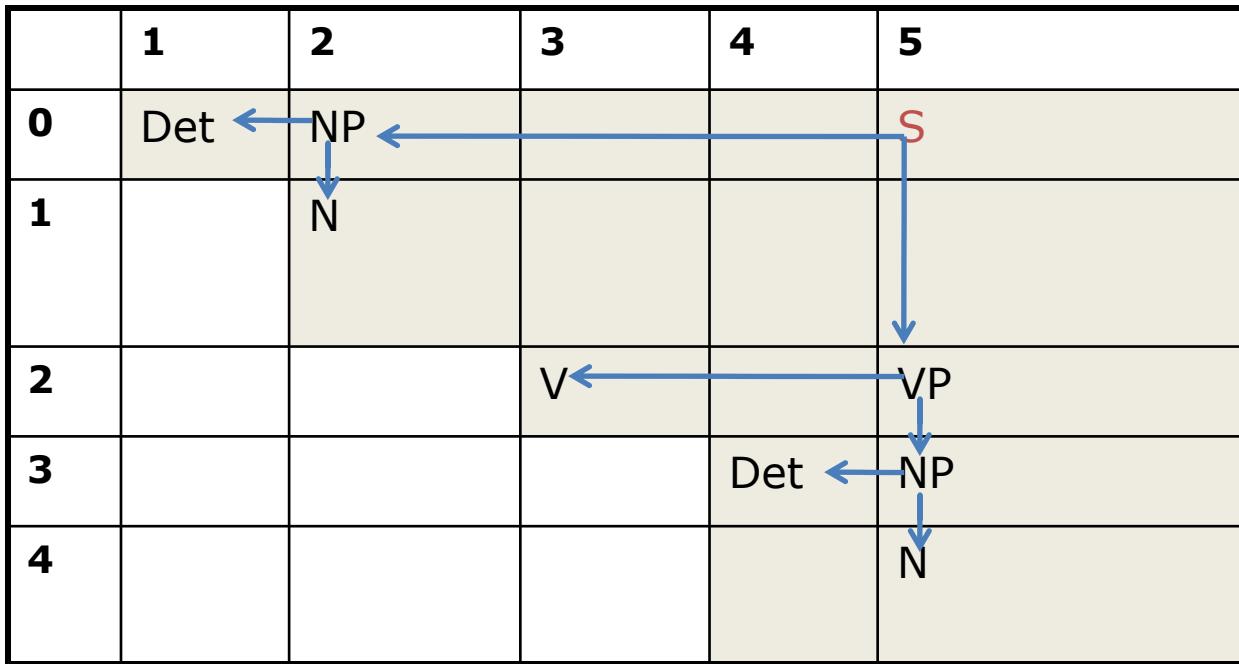
# From recognition to parsing

- The procedure so far will recognise a string as a legal sentence in English.
- But we'd like to get a parse tree back!
- Solution:
  - We can work our way back through the table and collect all the partial solutions into one parse tree.
  - Cells will need to be augmented with “backpointers”, i.e. With a pointer to the cells that the current cell covers.

# From recognition to parsing



# From recognition to parsing



NB: This algorithm always fills the top “triangle” of the table!

# What about ambiguity?

- The algorithm does not assume that there is only one parse tree for a sentence.
  - (Our simple grammar did not admit of any ambiguity, but this isn't realistic of course).
- There is nothing to stop it returning several parse trees.
- If there are multiple local solutions, then more than one non-terminal will be stored in a cell of the table.

# CFG definition (reminder)

- A CFG is a 4-tuple:  $(N, \Sigma, R, S)$ :
  - $N$  = a set of non-terminal symbols (e.g. NP, VP)
  - $\Sigma$  = a set of terminals (e.g. words)
    - $N$  and  $\Sigma$  are disjoint (no element of  $N$  is also an element of  $\Sigma$ )
  - $R$  = a set of rules of the form  $A \rightarrow \beta$  where:
    - $A$  is a non-terminal (a member of  $N$ )
    - $\beta$  is any string of terminals and non-terminals
  - $S$  = a designated start symbol (usually, “sentence”)

# Motivation

- Context-free grammars can be generalized to include probabilistic information by adding it to CFG rule
- Probabilistic Context Free Grammars (PCFGs) are the simplest and most natural probabilistic model for tree structures and the algorithms for them are closely related to those for HMMs.
- PCFG are also known as Stochastic Context-Free Grammar (SCFG)

# Formal Definition of a PCFG

- A PCFG consists of:
  - A set of terminals,  $\{w^k\}$ ,  $k= 1, \dots, V$
  - A set of nonterminals,  $N^i$ ,  $i= 1, \dots, n$
  - A designated start symbol  $N^1$
  - A set of **rules**,  $\{N^i \rightarrow \xi^j\}$ , (where  $\xi^j$  is a sequence of terminals and nonterminals)
  - A corresponding set of **probabilities on rules** such that:  $\forall i \quad \sum_j P(N^i \rightarrow \xi^j) = 1$

# Probability of a Derivation Tree and a String

- The probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..k} p(r(i))$$

where  $r(1), \dots, r(k)$  are the rules of the CFG used to generate the sentence  $w_{1m}$  of which  $T$  is a parse.

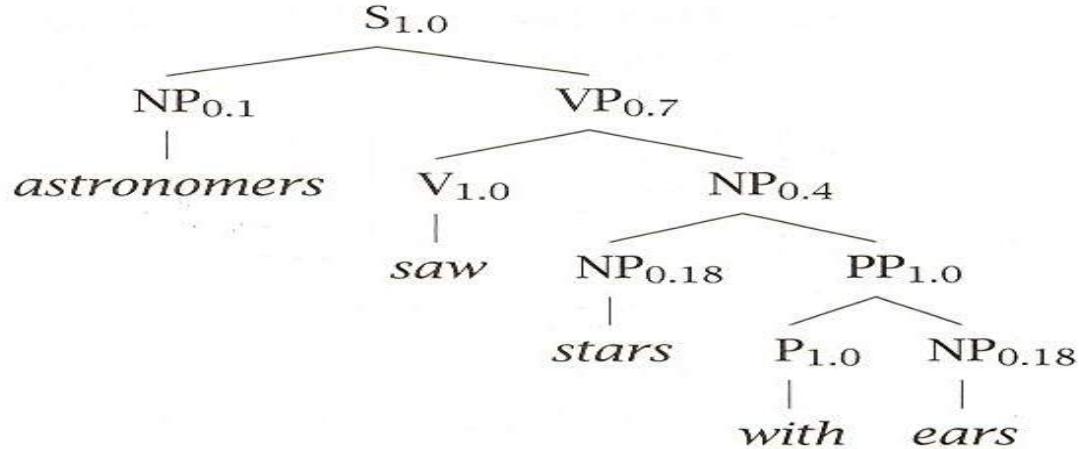
- The probability of a sentence (according to grammar  $G$ ) is given by:

$$P(w_{1m}) = \sum_t P(w_{1m}, t) = \sum_{\{t: \text{yield}(t)=w_{1m}\}} P(t)$$

where  $t$  is a parse tree of the sentence. Need dynamic programming to make this efficient!

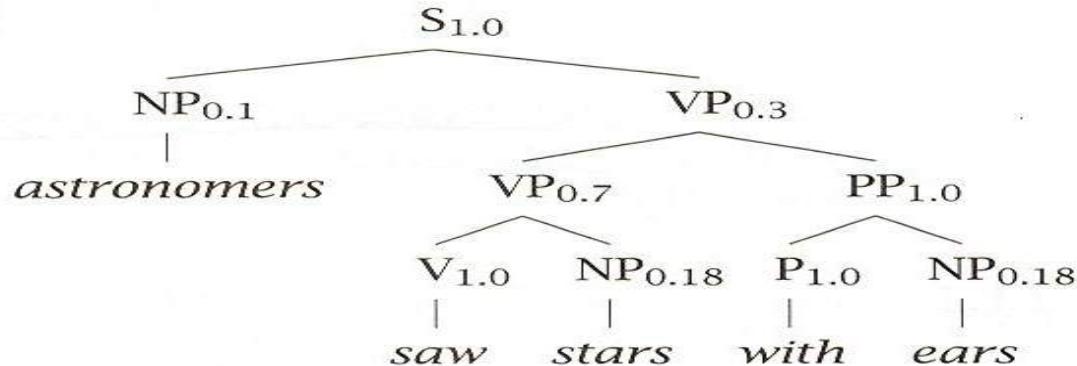
# Example: Probability of a Derivation Tree

$t_1:$



$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow astronomers$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow ears$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow saw$	0.04
$P \rightarrow with$	1.0	$NP \rightarrow stars$	0.18
$V \rightarrow saw$	1.0	$NP \rightarrow telescopes$	0.1

$t_2:$



# Some Features of PCFGs

- A PCFG gives some idea of the plausibility of different parses; however, the probabilities are based on **structural factors** and **not lexical ones**.
- PCFGs are good for grammar induction.
- PCFGs are robust.
- PCFGs give a **probabilistic language model** for English.
- The predictive power of a PCFG tends to be greater than for an HMM.
- PCFGs are not good models alone but they can be combined with a trigram model.

# 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  $T(s)$ . Then a PCFG assigns a probability  $p(t)$  to each member of  $T(s)$ . i.e., *we now have a ranking in order of probability.*
- ▶ The most likely parse tree for a sentence  $s$  is

$$\arg \max_{t \in T(s)} p(t)$$

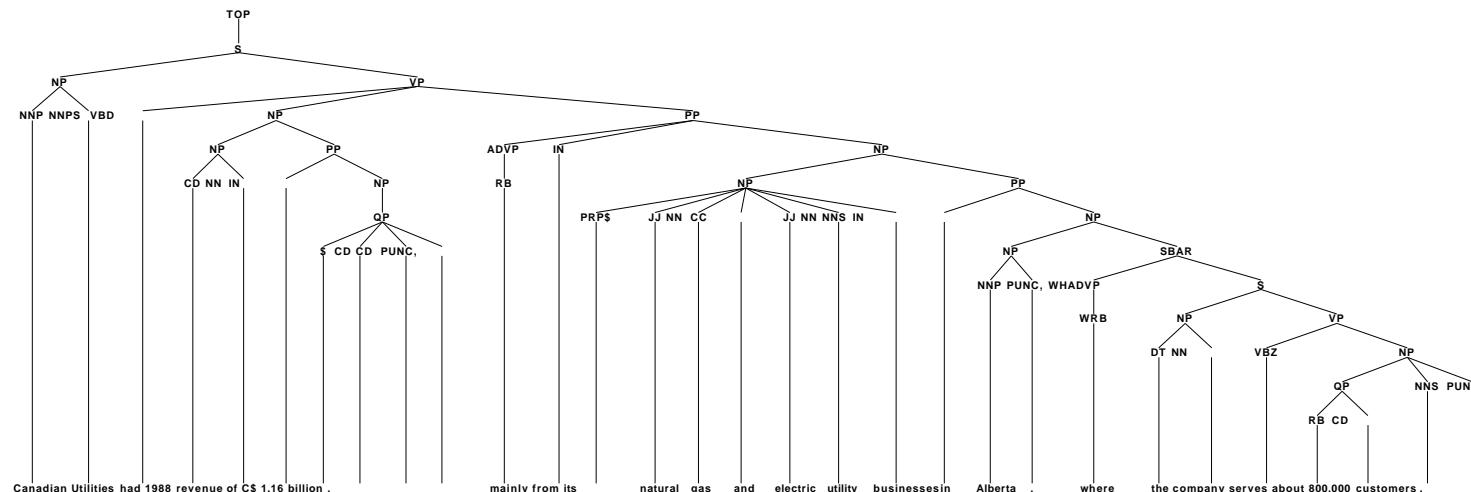
# PCFG based Grammar

- ▶ 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

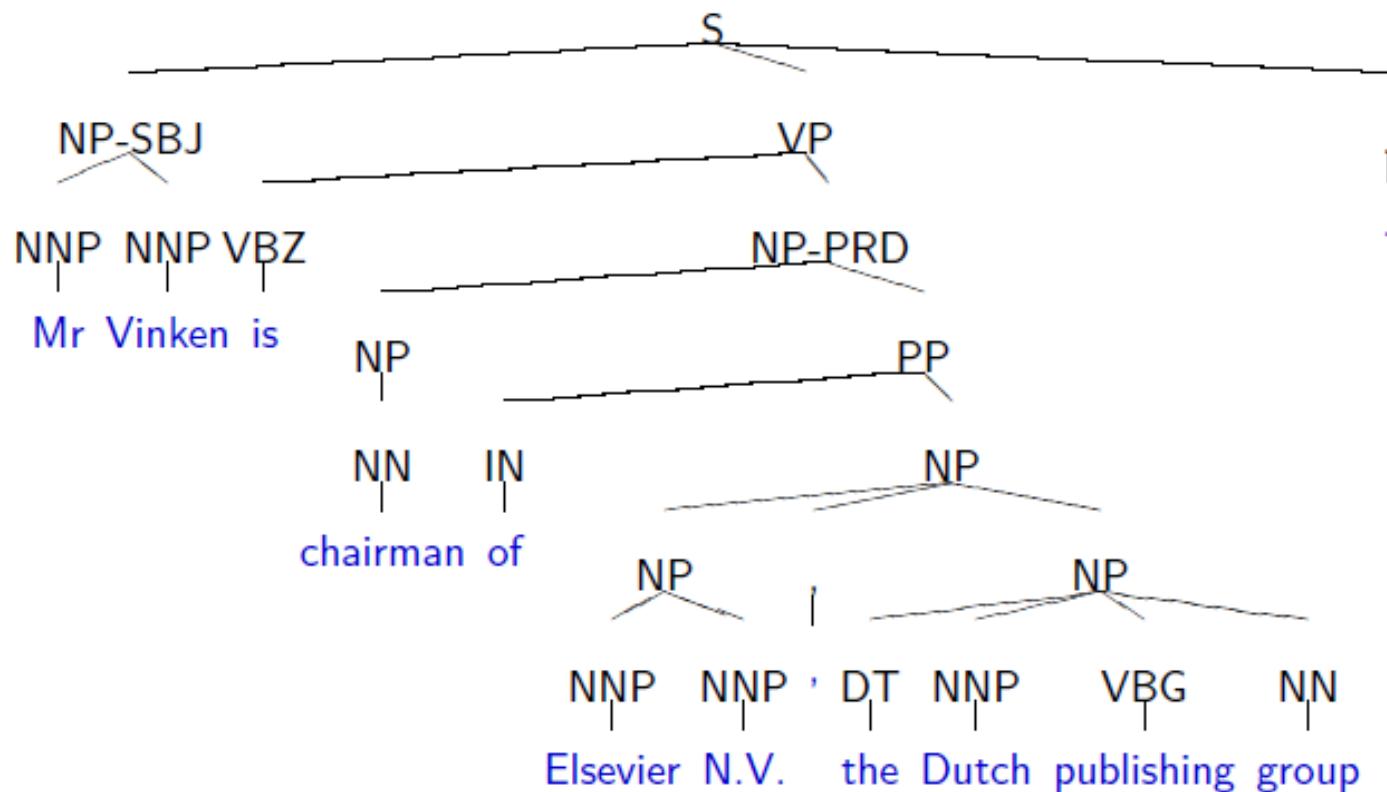
# Data for Parsing Experiments: Treebanks

- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



# Example tree



## Characteristics of PCFGs

- In a PCFG, the probability  $P(A \rightarrow \beta)$  expresses the likelihood that the non-terminal A will expand as  $\beta$ .
  - e.g. the likelihood that  $S \rightarrow NP\ VP$ 
    - (as opposed to  $S \rightarrow VP$ , or  $S \rightarrow NP\ VP\ PP$ , or... )
- can be interpreted as a conditional probability:
  - probability of the expansion, given the LHS non-terminal
  - $P(A \rightarrow \beta) = P(A \rightarrow \beta | A)$
- Therefore, for any non-terminal A, probabilities of every rule of the form  $A \rightarrow \beta$  must sum to 1
  - in this case, we say the PCFG is **consistent**

# Word/Tag Counts

	<b>N</b>	<b>V</b>	<b>ART</b>	<b>P</b>	<b>TOTAL</b>
<i>flies</i>	21	23	0	0	44
<i>fruit</i>	49	5	1	0	55
<i>like</i>	10	30	0	21	61
<i>a</i>	1	0	201	0	202
<i>the</i>	1	0	300	2	303
<i>flower</i>	53	15	0	0	68
<i>flowers</i>	42	16	0	0	58
<i>birds</i>	64	1	0	0	65
<i>others</i>	592	210	56	284	1142
<b>TOTAL</b>	<b>833</b>	<b>300</b>	<b>558</b>	<b>307</b>	<b>1998</b>

# Lexical Probability Estimates

$$P(\text{the}|\text{ART}) = 300 / 558 = 0.54$$

ReARD	.54	ReART	.36
ReesN	.05	ReN	.01
Reesy	.05	Reesey	.03
Reey	.1	Reey	.5
ReeD	.08	ReesN	.05
ReeN	.02		

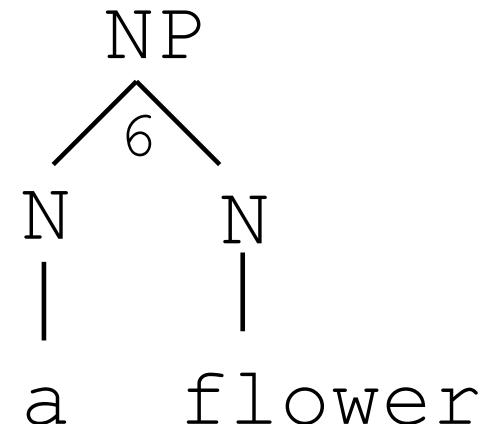
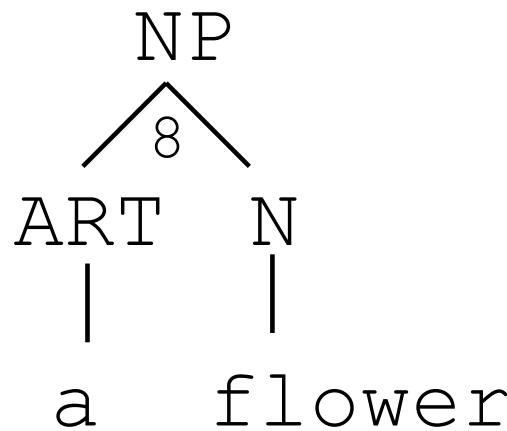
# The PCFG

- Below is a probabilistic CFG (PCFG) with probabilities derived from analyzing a parsed version of Allen's corpus.

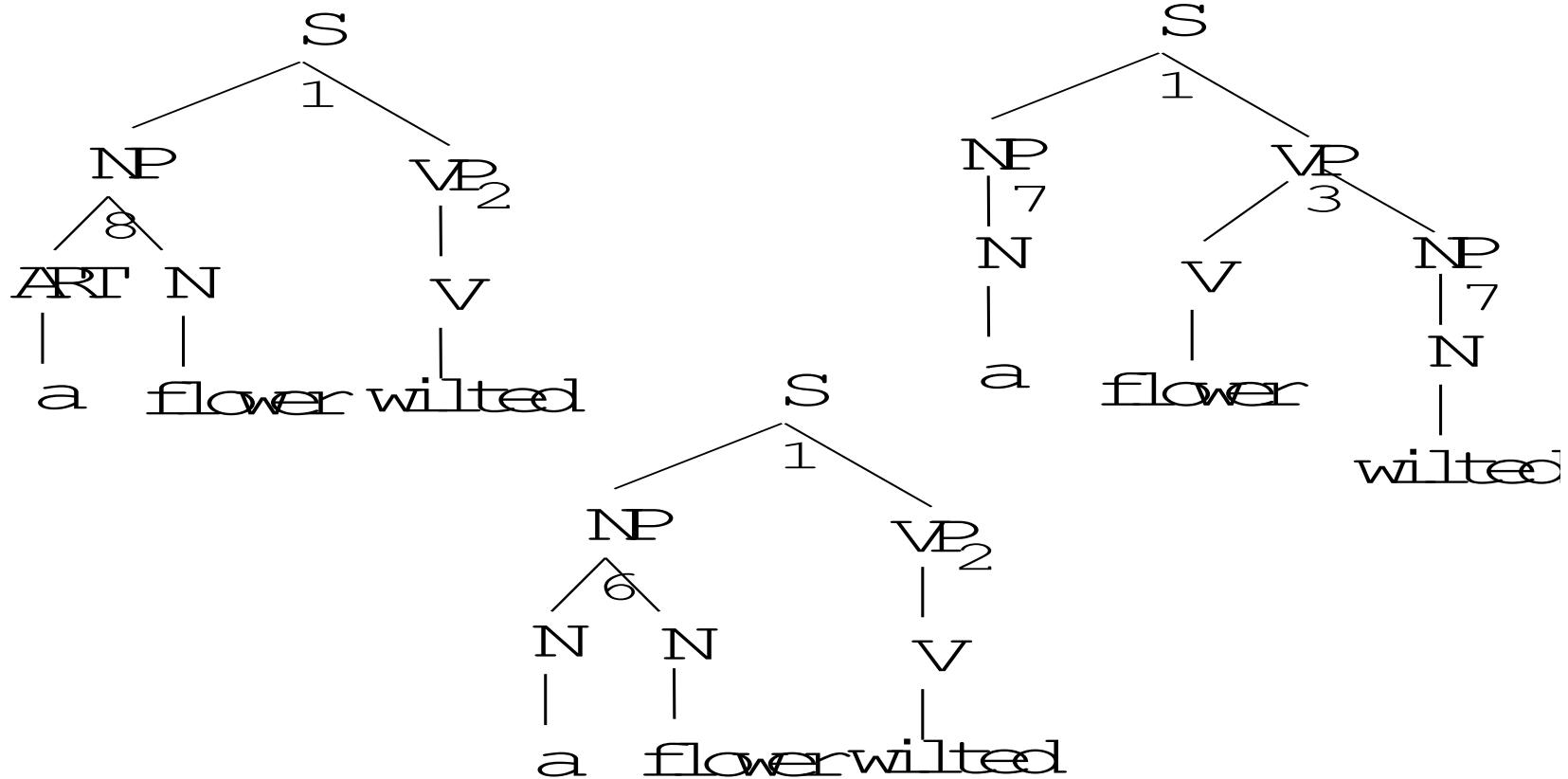
Rule	Count for IHS	Count for Rule	Prob
1. $S \rightarrow NP VP$	300	300	1
2 $VP \rightarrow V$	300	116	.386
3 $VP \rightarrow VNP$	300	118	.393
4 $VP \rightarrow VNPPP$	300	66	.22
5 $NP \rightarrow NPP$	1032	241	.23
6 $NP \rightarrow NN$	1032	92	.09
7. $NP \rightarrow N$	1032	141	.14
8 $NP \rightarrow ARTN$	1032	558	.54
9 $PP \rightarrow PNP$	307	307	1

# Parsing with a PCFG

- Using the lexical probabilities, we can derive probabilities that the constituent NP generates a sequence like *a flower*. Two rules could generate the string of words:



# Three Possible Trees for an S



# Parsing with a PCFG

- The probability of a sentence generating *A flower wilted*:

$$P(a \text{ flower } \text{ wilted} | S) = P(R_1 | S) \times P(a \text{ flower} | NP) \times \\ P(wilted | VP) + P(R_1 | S) \times P(a | NP) \times P(flower \text{ wilted} | VP)$$

- Using this approach, the probability that a given sentence will be generated by the grammar can be efficiently computed.
- It only requires some way of recording the value of each constituent between each two possible positions. The requirement can be filled by a packed chart structure.

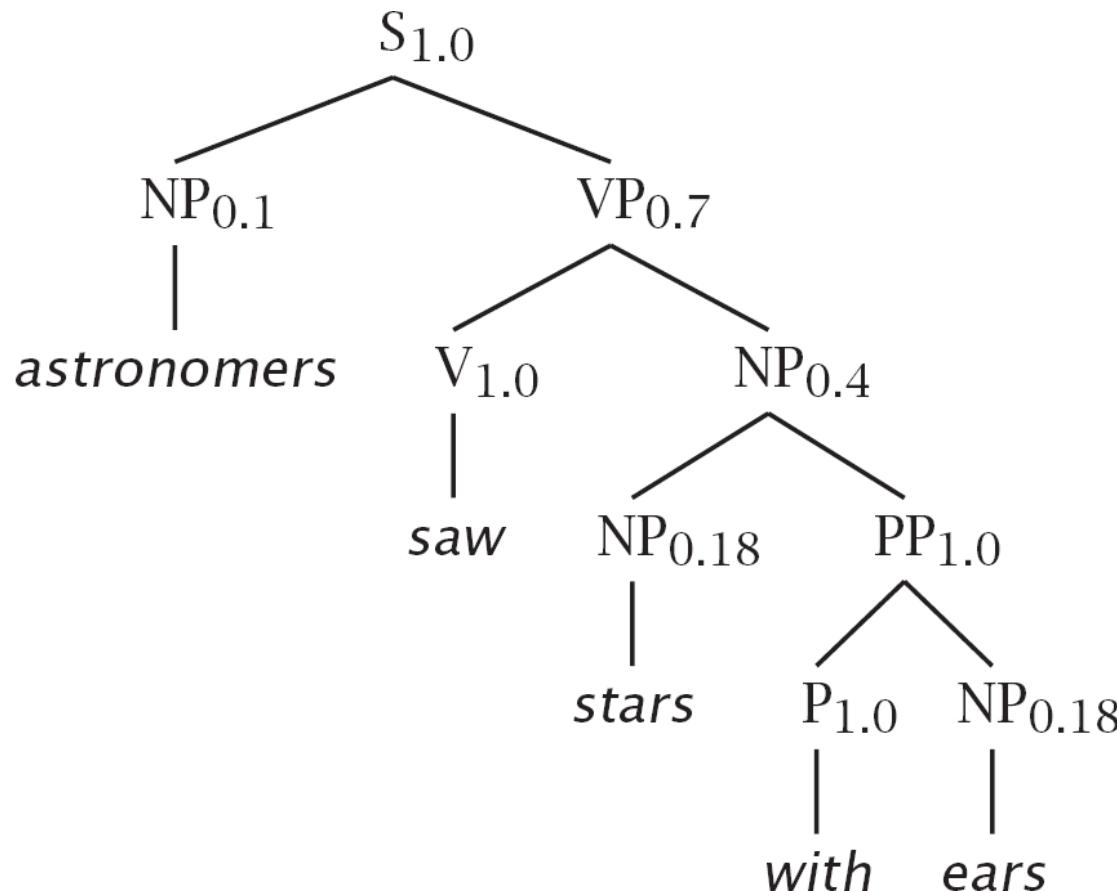
# Example

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow \text{astronomers}$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow \text{ears}$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow \text{saw}$	0.04
$P \rightarrow \text{with}$	1.0	$NP \rightarrow \text{stars}$	0.18
$V \rightarrow \text{saw}$	1.0	$NP \rightarrow \text{telescopes}$	0.1

- Terminals      with, saw, astronomers, ears, stars, telescopes
- Nonterminals    S, PP, P, NP, VP, V
- Start symbol    S

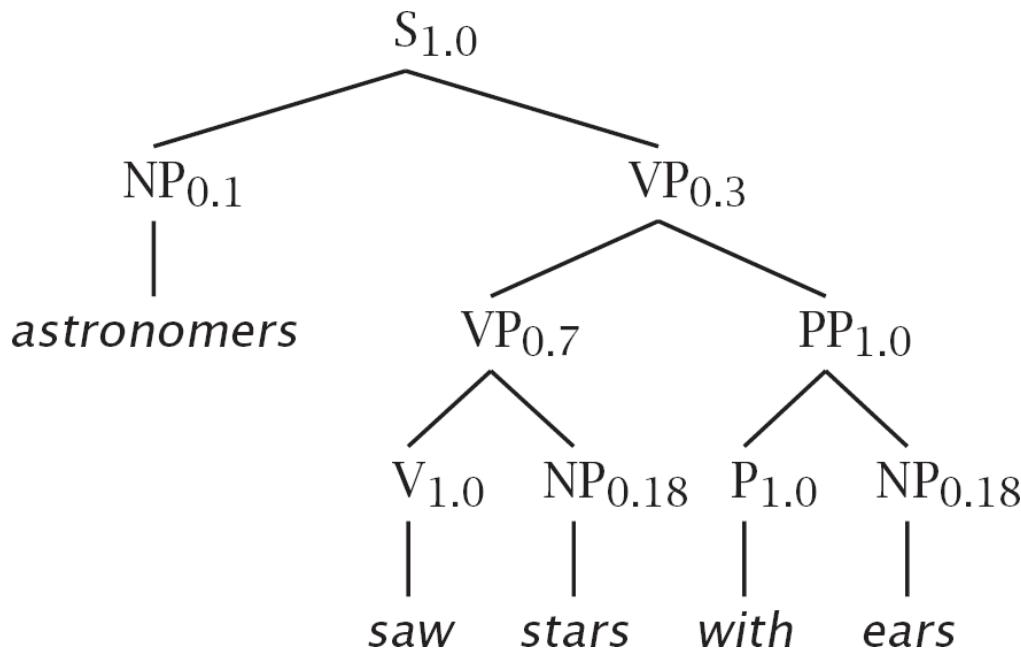
# astronomers saw stars with ears

$t_1:$



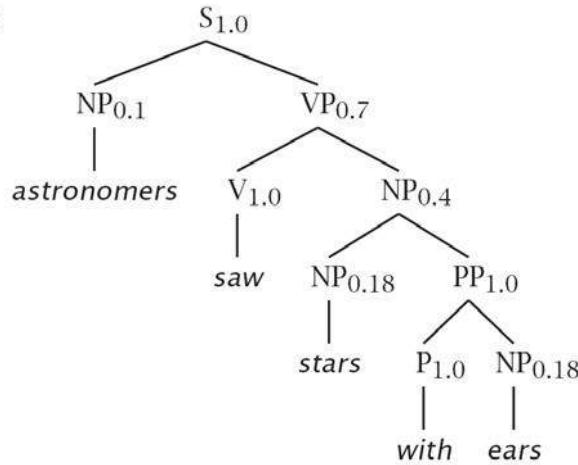
# astronomers saw stars with ears

$t_2:$



# Probabilities

$t_1:$

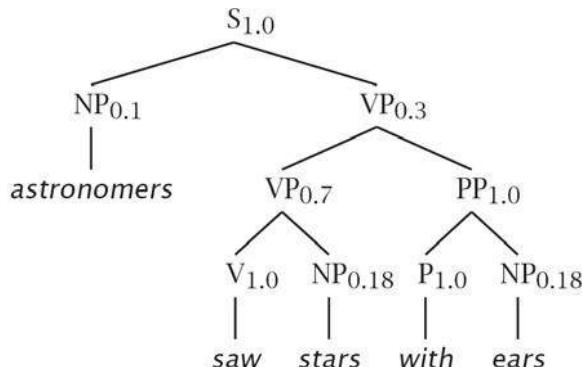


$$\begin{aligned}
 P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0009072
 \end{aligned}$$

$$\begin{aligned}
 P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0006804
 \end{aligned}$$

$$P(w_{15}) = P(t_1) + P(t_2) = 0.0015876$$

$t_2:$



# Uses of probabilities in parsing

- **Disambiguation:** given  $n$  legal parses of a string, which is the most likely?
  - e.g. PP-attachment ambiguity can be resolved this way
- **Speed:** we've defined parsing as a search problem
  - search through space of possible applicable derivations
  - search space can be pruned by focusing on the most likely sub-parses of a parse
- Parser can be used as a model to determine the probability of a sentence, given a parse
  - typical use in speech recognition, where input utterance can be “heard” as several possible sentences

# Using PCFG probabilities

- PCFG assigns a probability to every parse-tree  $t$  of a string  $W$ 
  - e.g. every possible parse (derivation) of a sentence recognised by the grammar
- Notation:
  - $G$  = a PCFG
  - $s$  = a sentence
  - $t$  = a particular tree under our grammar
    - $t$  consists of several nodes  $n$
    - each node is generated by applying some rule  $r$

# Probability of a tree vs. a sentence

- We work out the probability of a parse tree  $t$  by multiplying the probability of every rule (node) that gives rise to  $t$  (i.e. the derivation of  $t$ ).
- Note that:
  - A tree can have multiple derivations
    - (different sequences of rule applications could give rise to the same tree)
  - But the probability of the tree remains the same
    - (it's the same probabilities being multiplied)
  - We usually speak as if a tree has only one derivation, called the **canonical derivation**

# Picking the best parse in a PCFG

- A sentence will usually have several parses
  - we usually want them ranked, or only want the  $n$  best parses
  - we need to focus on  $P(t|s, G)$ 
    - probability of a parse, given our sentence and our grammar
  - definition of the best parse for  $s$ :
    - The tree for which  $P(t|s, G)$  is highest

# Probability of a sentence

- Given a probabilistic context-free grammar  $G$ , we can calculate the probability of a sentence (as opposed to a tree).
- Observe that:
  - As far as our grammar is concerned, a sentence is only a sentence if it can be recognised by the grammar (it is “legal”)
  - There can be multiple parse trees for a sentence.
    - Many trees whose **yield** is the sentence
  - The probability of the sentence is the sum of all the probabilities of the various trees that yield the sentence.

# Using CKY to parse with a PCFG

- The basic CKY algorithm remains unchanged.
- However, rather than only keeping partial solutions in our table cells (i.e. The rules that match some input), we also keep their probabilities.

# Probabilistic CKY: example PCFG

- $S \rightarrow NP\ VP \ [.80]$
- $NP \rightarrow Det\ N \ [.30]$
- $VP \rightarrow V\ NP \ [.20]$
- $V \rightarrow includes \ [.05]$
- $Det \rightarrow the \ [.4]$
- $Det \rightarrow a \ [.4]$
- $N \rightarrow meal \ [.01]$
- $N \rightarrow flight \ [.02]$

# Probabilistic CYK: initialisation

- *The flight includes a meal.*
  - $S \rightarrow NP\ VP$  [.80]
  - $NP \rightarrow Det\ N$  [.30]
  - $VP \rightarrow V\ NP$  [.20]
  - $V \rightarrow \text{includes}$  [.05]
  - $Det \rightarrow \text{the}$  [.4]
  - $Det \rightarrow a$  [.4]
  - $N \rightarrow \text{meal}$  [.01]
  - $N \rightarrow \text{flight}$  [.02]

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>					
<b>1</b>					
<b>2</b>					
<b>3</b>					
<b>4</b>					
<b>5</b>					

# Probabilistic CYK: lexical step

- *The flight includes a meal.*
- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow includes$  [.05]
- $Det \rightarrow the$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow meal$  [.01]
- $N \rightarrow flight$  [.02]

	1	2	3	4	5
0	Det (.4)				
1					
2					
3					
4					
5					

# Probabilistic CYK: lexical step

- *The flight includes a meal.*
  - $S \rightarrow NP\ VP [ .80 ]$
  - $NP \rightarrow Det\ N [ .30 ]$
  - $VP \rightarrow V\ NP [ .20 ]$
  - $V \rightarrow includes [ .05 ]$
  - $Det \rightarrow the [ .4 ]$
  - $Det \rightarrow a [ .4 ]$
  - $N \rightarrow meal [ .01 ]$
  - $N \rightarrow flight [ .02 ]$

	1	2	3	4	5
0	Det (.4)				
1		N .02			
2					
3					
4					
5					

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow includes$  [.05]
- $Det \rightarrow the$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow meal$  [.01]
- $N \rightarrow flight$  [.02]

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2					
3					
4					
5					

Note: probability of NP in [0,2]  
 $P(Det \rightarrow the) * P(N \rightarrow meal) * P(NP \rightarrow Det\ N)$

# Probabilistic CYK: lexical step

- *The flight includes a meal.*
  - $S \rightarrow NP\ VP$  [.80]
  - $NP \rightarrow Det\ N$  [.30]
  - $VP \rightarrow V\ NP$  [.20]
  - $V \rightarrow \text{includes}$  [.05]
  - $Det \rightarrow \text{the}$  [.4]
  - $Det \rightarrow a$  [.4]
  - $N \rightarrow \text{meal}$  [.01]
  - $N \rightarrow \text{flight}$  [.02]

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2			V .05		
3					
4					
5					

# Probabilistic CYK: lexical step

- *The flight includes a meal.*
- $S \rightarrow NP\ VP [ .80 ]$
- $NP \rightarrow Det\ N [ .30 ]$
- $VP \rightarrow V\ NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det (.4)	NP .0024			
<b>1</b>		N .02			
<b>2</b>			V .05		
<b>3</b>				Det .4	
<b>4</b>					
<b>5</b>					

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP$  [.80]
- $NP \rightarrow Det\ N$  [.30]
- $VP \rightarrow V\ NP$  [.20]
- $V \rightarrow \text{includes}$  [.05]
- $Det \rightarrow \text{the}$  [.4]
- $Det \rightarrow a$  [.4]
- $N \rightarrow \text{meal}$  [.01]
- $N \rightarrow \text{flight}$  [.02]

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det (.4)	NP .0024			
<b>1</b>		N .02			
<b>2</b>			V .05		
<b>3</b>				Det .4	
<b>4</b>					N .01

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP\ [.80]$
- $NP \rightarrow Det\ N\ [.30]$
- $VP \rightarrow V\ NP\ [.20]$
- $V \rightarrow includes\ [.05]$
- $Det \rightarrow the\ [.4]$
- $Det \rightarrow a\ [.4]$
- $N \rightarrow meal\ [.01]$
- $N \rightarrow flight\ [.02]$

	1	2	3	4	5
0	Det (.4)	NP .0024			
1		N .02			
2			V .05		
3				Det .4	NP .001
4					N .01

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP [ .80 ]$
- $NP \rightarrow Det\ N [ .30 ]$
- $VP \rightarrow V\ NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det (.4)	NP .0024			
<b>1</b>		N .02			
<b>2</b>			V .05		VP .00001
<b>3</b>				Det .4	NP .001
<b>4</b>					N .01

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*
- $S \rightarrow NP\ VP [ .80 ]$
- $NP \rightarrow Det\ N [ .30 ]$
- $VP \rightarrow V\ NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det (.4)	NP .0024			S .00000001 92
<b>1</b>		N .02			
<b>2</b>			V .05		VP .00001
<b>3</b>				Det .4	NP .001
<b>4</b>					N .01

# Probabilistic CYK: summary

- Cells in chart hold probabilities
- Bottom-up procedure computes probability of a parse incrementally.
- To obtain parse trees, we traverse the table “backwards” as before.
  - Cells need to be augmented with backpointers.

---

# Parsing Implementation Demo

# Problems with PCFGs

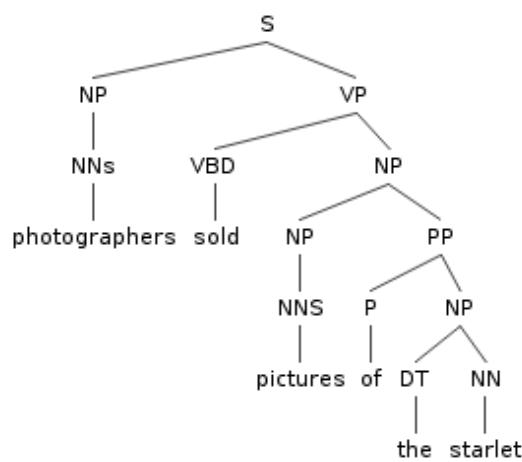
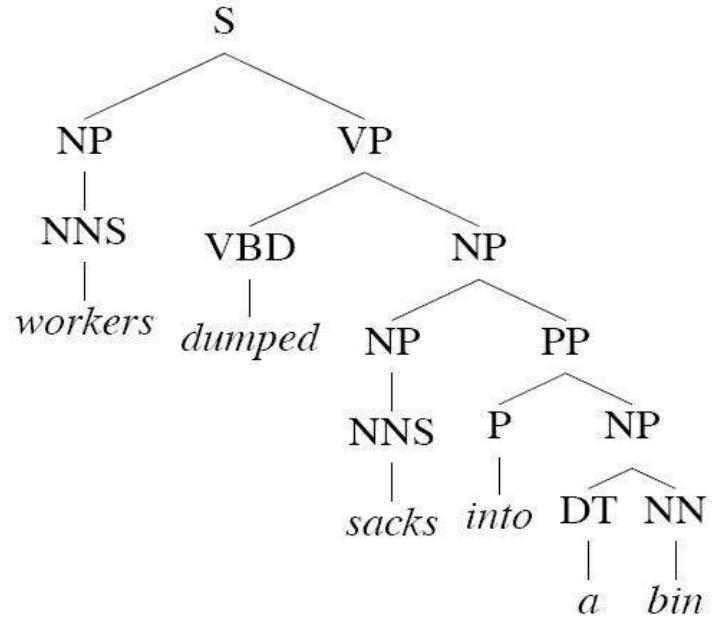
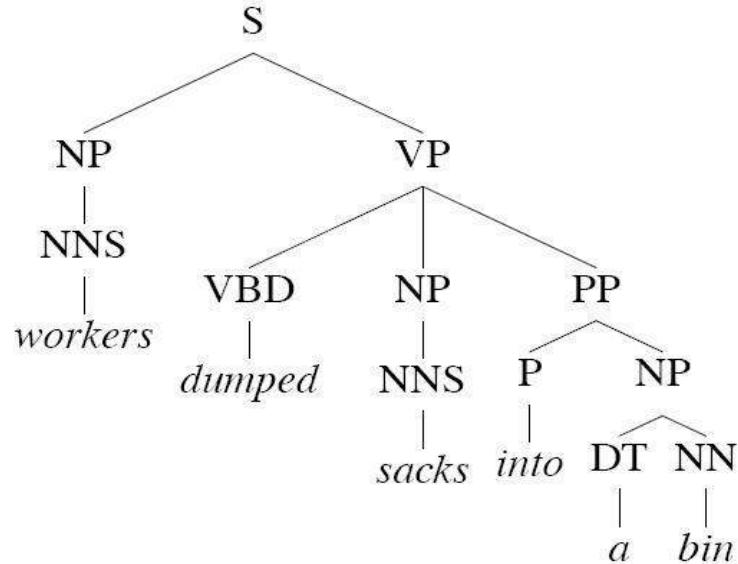
- No Context
  - (immediate prior context, speaker, ...)
- No Lexicalization
  - “VP NP NP” more likely if verb is “hand” or “tell”
  - fail to capture lexical dependencies (n-grams do)
- No Structural Context
  - How NP expands depends on position

## Flaws I: Structural independence

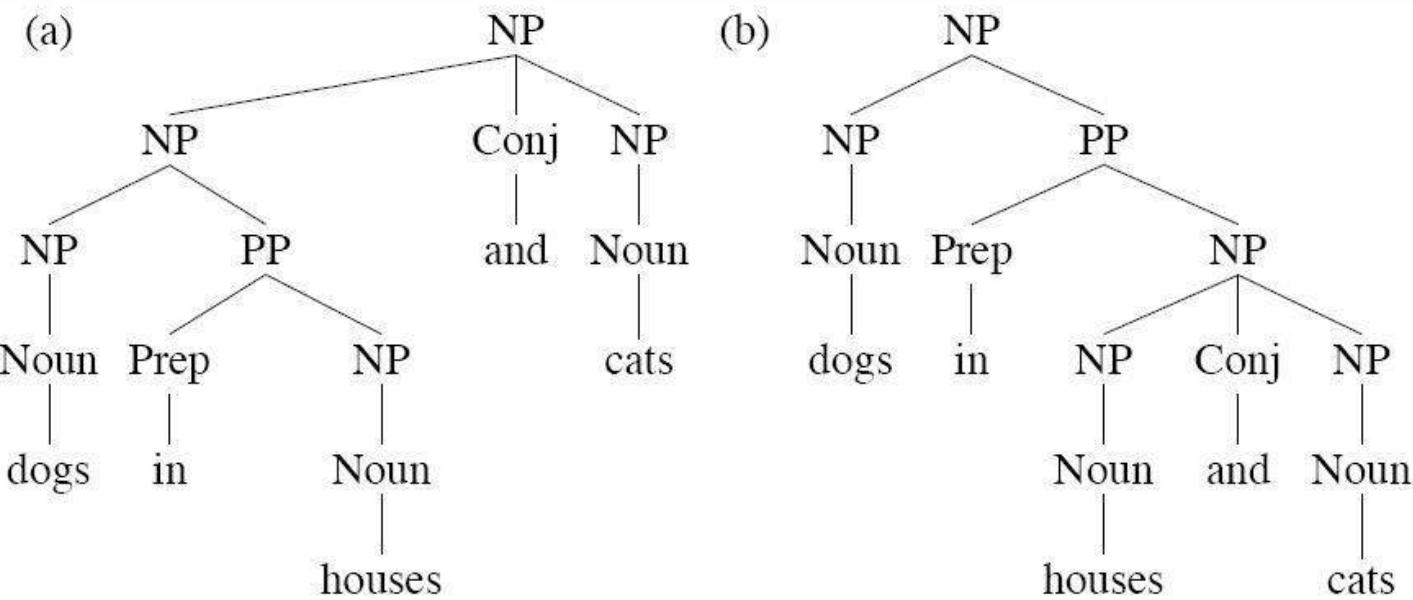
- Probability of a rule  $r$  expanding node  $n$  depends only on  $n$ .
- Independent of other non-terminals
- Example:
  - $P(NP \rightarrow Pro)$  is independent of where the NP is in the sentence
  - but we know that  $NP \rightarrow Pro$  is much more likely in subject position
  - Francis et al (1999) using the Switchboard corpus:
    - 91% of subjects are pronouns;
    - only 34% of objects are pronouns

## Flaws II: lexical independence

- vanilla PCFGs ignore lexical material
  - e.g.  $P(VP \rightarrow V NP PP)$  independent of the head of NP or PP or lexical head V
- Examples:
  - prepositional phrase attachment preferences depend on lexical items; cf:
    - Workers dumped [sacks into a bin]
    - Workers dumped [sacks] [into a bin] (preferred parse)
  - coordination ambiguity:
    - [dogs in houses] and [cats]
    - [dogs] [in houses and cats]



# Conjunction attachment

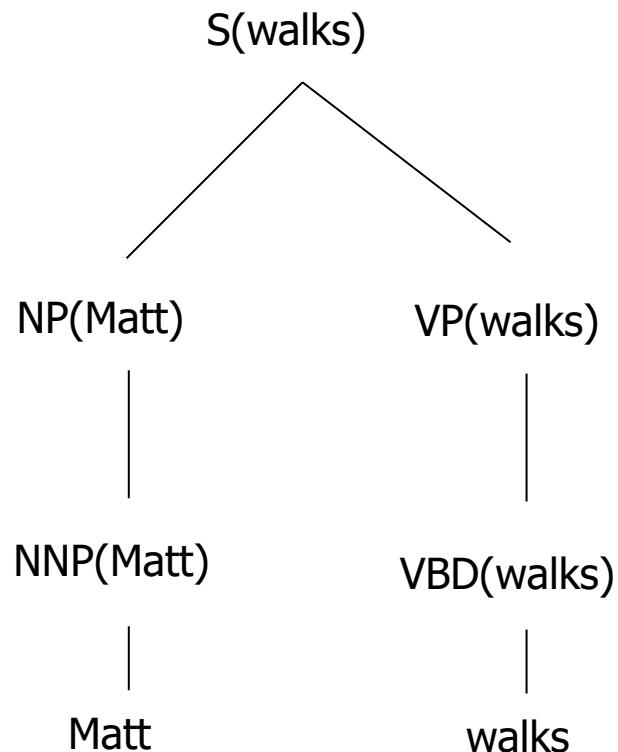


# Lexicalised PCFGs

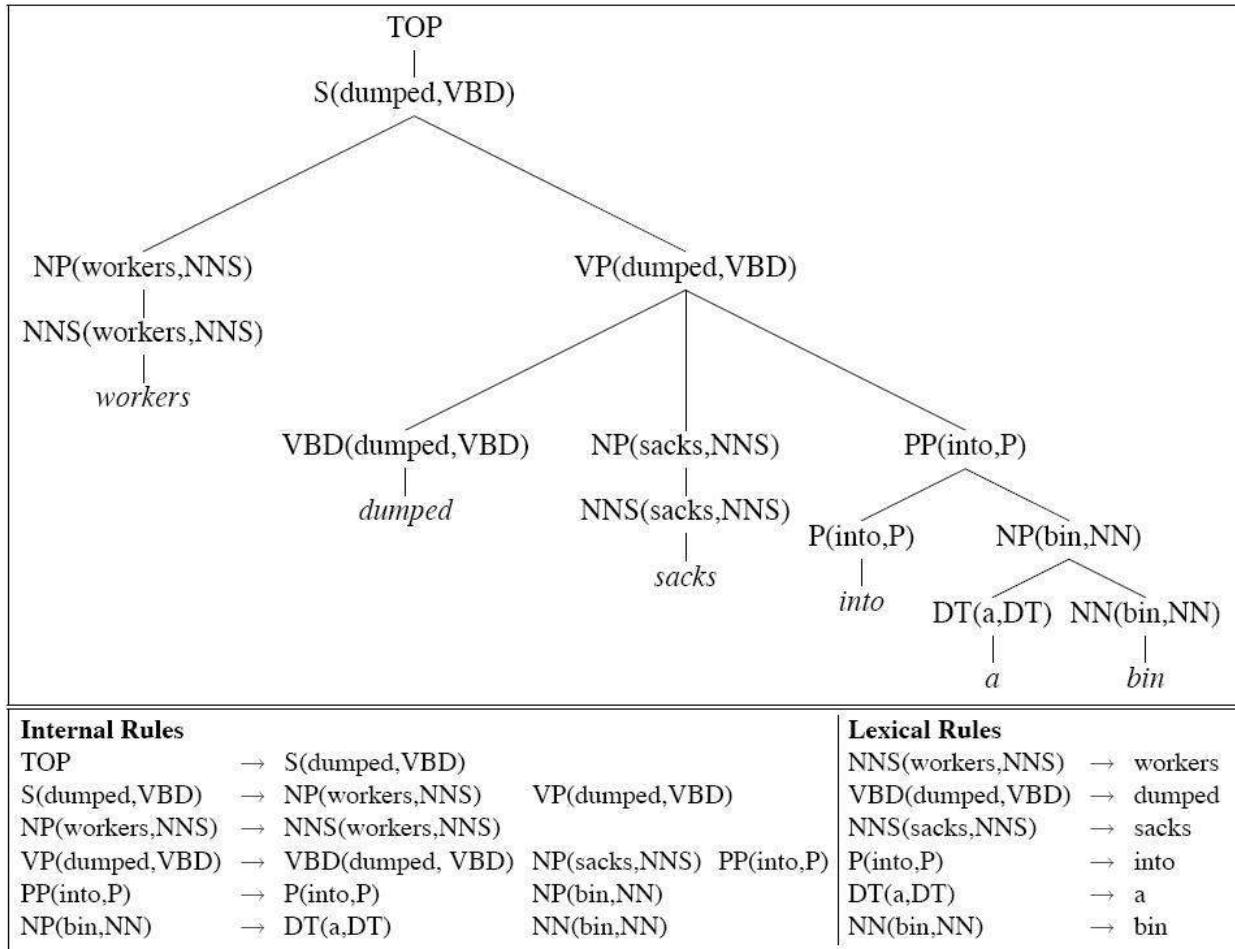
- Attempt to weaken the lexical independence assumption.
- Most common technique:
  - mark each phrasal head (N,V, etc) with the lexical material
  - this is based on the idea that the most crucial lexical dependencies are between head and dependent
  - E.g.: Charniak 1997, Collins 1999

# Lexicalised PCFGs: *Matt walks*

- Makes probabilities partly dependent on lexical content.
- $P(VP \rightarrow VBD | VP)$  becomes:  
 $P(VP \rightarrow VBD | VP, h(VP)=\text{walks})$
- NB: normally, we can't assume that all heads of a phrase of category C are equally probable.



# Lexical attachment



# Practical problems for lexicalised PCFGs

- Data sparseness: we don't necessarily see all heads of all phrasal categories often enough in the training data
- Flawed assumptions: lexical dependencies occur elsewhere, not just between head and complement
  - *I got the easier problem of the two to solve*
  - *of the two* and *to solve* are very likely because of the prehead modifier *easier*

# Evaluating Parsers

- We need a measure to evaluate parser performance against gold standard
  - ratio of fully correct sentences parses too coarse
  - ratio of correct constituents
- Does *correct* mean *precision*?

$$\text{precision} = \frac{\text{count}(\text{matching constituents})}{\text{count}(\text{predicted constituents})}$$

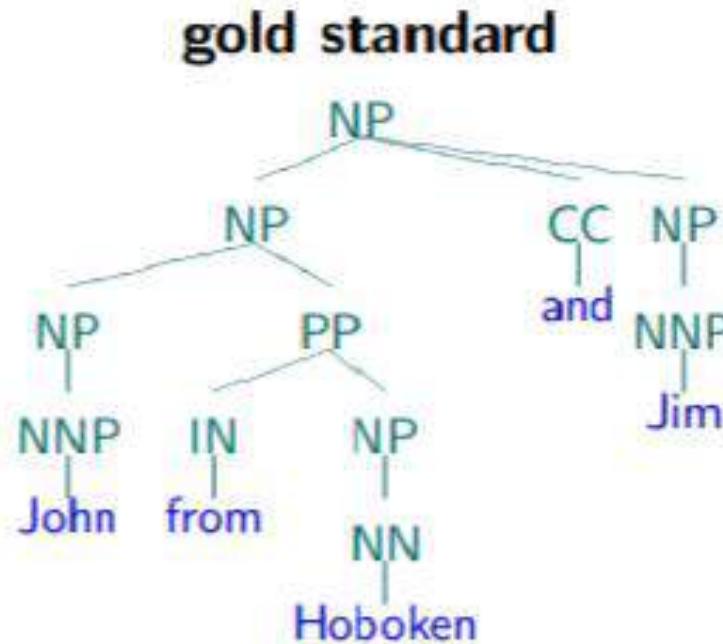
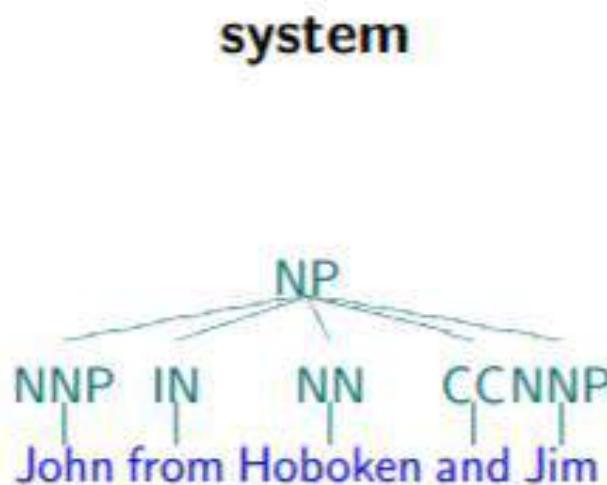
- Does *correct* mean *recall*?

$$\text{precision} = \frac{\text{count}(\text{matching constituents})}{\text{count}(\text{gold standard constituents})}$$

Credits: Philipp Koehn

# Evaluating Parsers

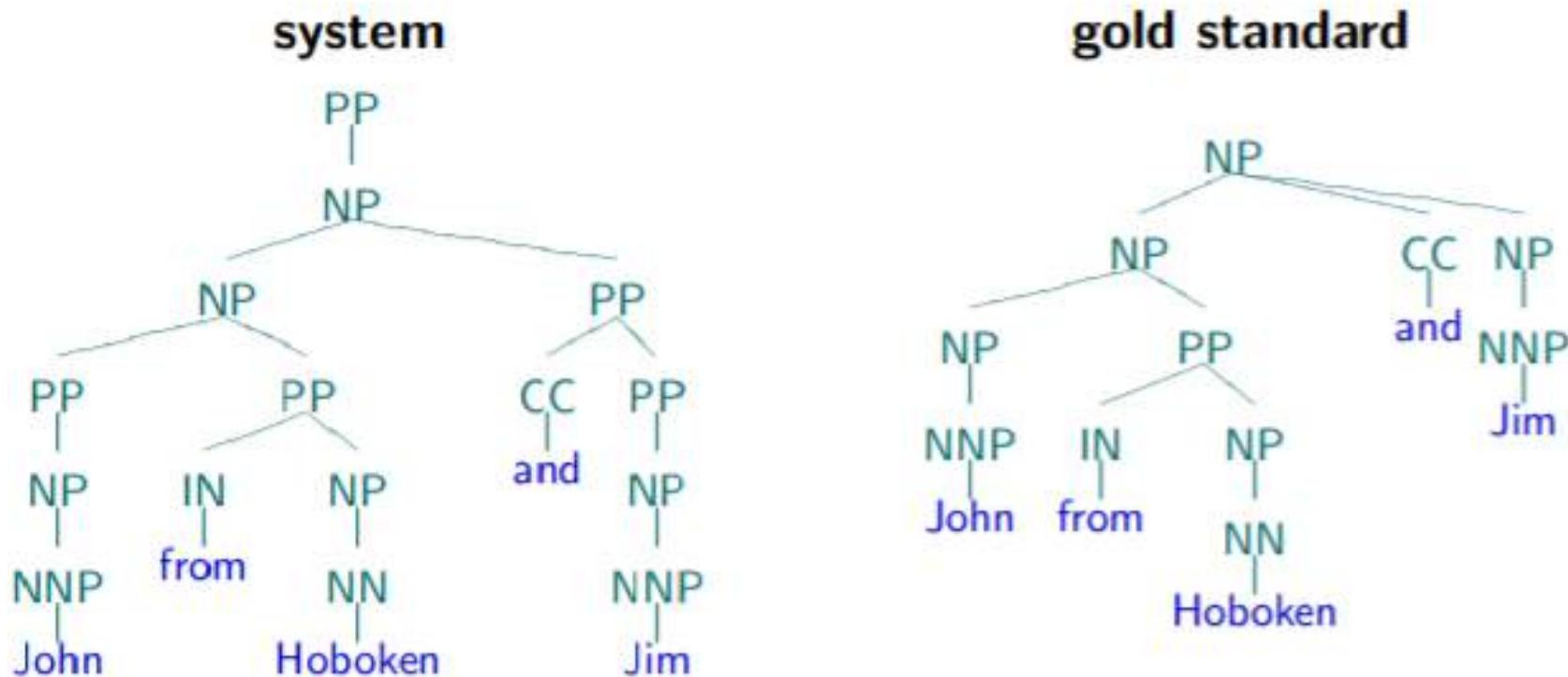
## High Precision, Low Recall



all predicted constituents match gold standard (precision 1/1)  
... but we are missing quite a few (recall 1/6)

# Evaluating Parsers

## Low Precision, High Recall



all gold standard constituents are predicted (recall 6/6)  
... but we are predicting many more (precision 6/10)

# Evaluating Parsers

- F-measure: balance of precision and recall

$$F_1 = \frac{\text{precision} \times \text{recall}}{(\text{precision} + \text{recall})/2}$$

- F-measure is used in many other NLP tasks and may be adjusted to give more emphasis to either precision or recall

Credits: Philipp Koehn

# Extra Reading

- <https://www.youtube.com/watch?v=Z6GsoBA-09k&list=PLQiyVNMPDLKnZYBTUOISI9mi9wAErFtFm&index=62>
- <https://lost-contact.mit.edu/afs/cs.pitt.edu/projects/nltk/docs/tutorial/pcfg/nochunks.html>
- <http://www.nltk.org/howto/grammar.html>
- [https://www.tutorialspoint.com/natural\\_language\\_toolkit/natural\\_language\\_toolkit\\_parsing.htm](https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_parsing.htm)
- <https://lost-contact.mit.edu/afs/cs.pitt.edu/projects/nltk/docs/tutorial/pcfg/nochunks.html>
- [http://courses.washington.edu/ling571/ling571\\_WIN2017/slides/ling571\\_class\\_6\\_pcfg\\_impr\\_flat.pdf](http://courses.washington.edu/ling571/ling571_WIN2017/slides/ling571_class_6_pcfg_impr_flat.pdf)
- <http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/lexpcfgs.pdf>



**Thank you for your time!!**



# Natural Language Processing

## DSECL ZG565



**BITS** Pilani  
Pilani Campus

Dr. Chetana Gavankar, Ph.D,

IIT Bombay-Monash University Australia

Associate Professor, BITS Pilani

[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)



**Session 7 - Dependency Parsing  
Date – 10<sup>th</sup> October 2020  
Time – 9 am to 11 am**

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Jurafsky and Prof. Martin, Prof. Pawan Goyal and many others who made their course materials freely available online.

# Session Content



(Ref: Chapter 15 Jurafsky and Martin)

---

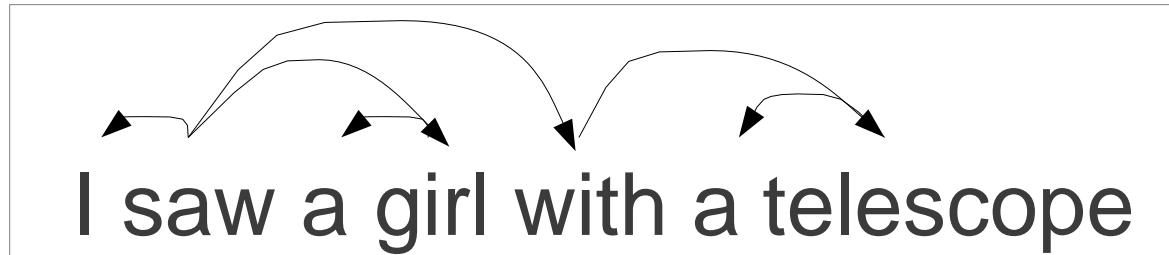
- Motivation
- Dependency structure and Dependency grammar
- Dependency Relation
- Universal Dependencies
- Method of Dependency Parsing
- Graph based dependency Parsing
- Evaluation

# Ambiguity is Explosive

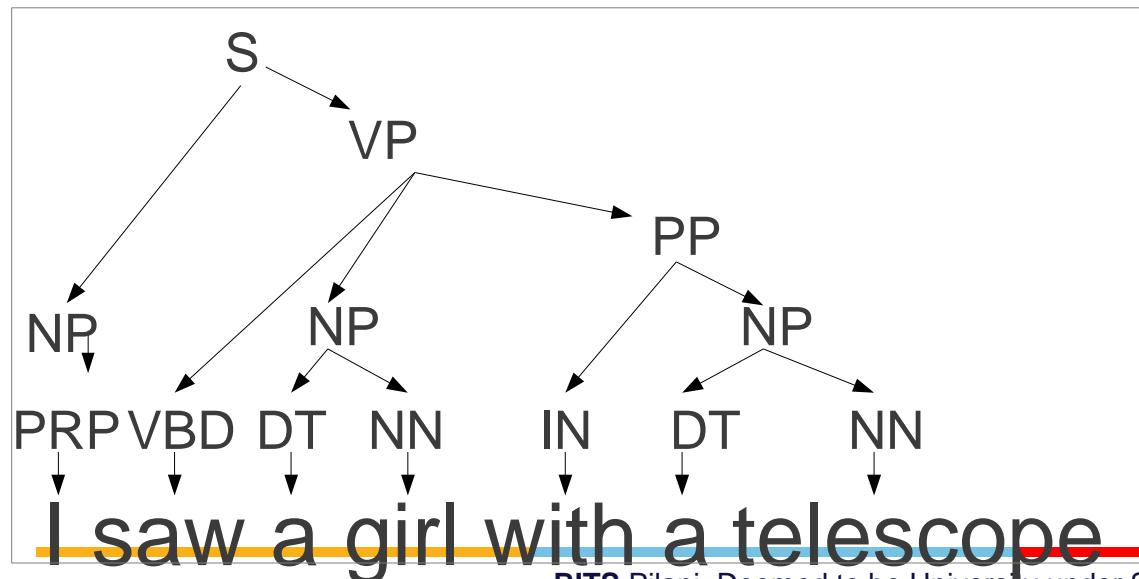
- “I saw the man with the telescope”: 2 parses
- “I saw the man on the hill with the telescope.”: 5 parses
- “I saw the man on the hill in Texas with the telescope”: 14 parses
- “I saw the man on the hill in Texas with the telescope at noon.”: 42 parses
- “I saw the man on the hill in Texas with the telescope at noon on Monday” 132 parses

# Two Types of Parsing

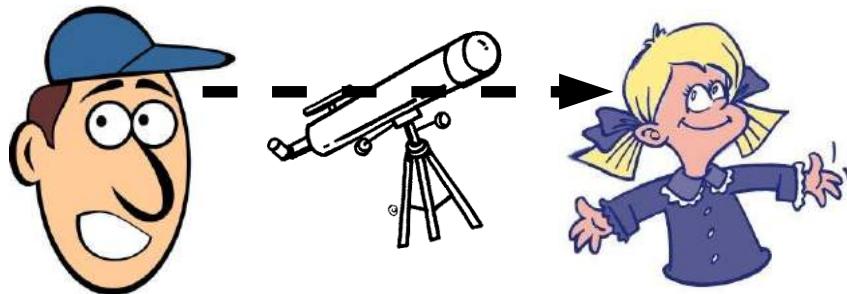
- Dependency: focuses on relations between words



- Phrase structure: focuses on identifying phrases and their recursive structure



# Dependencies Also Resolve Ambiguity



I saw a girl with a telescope

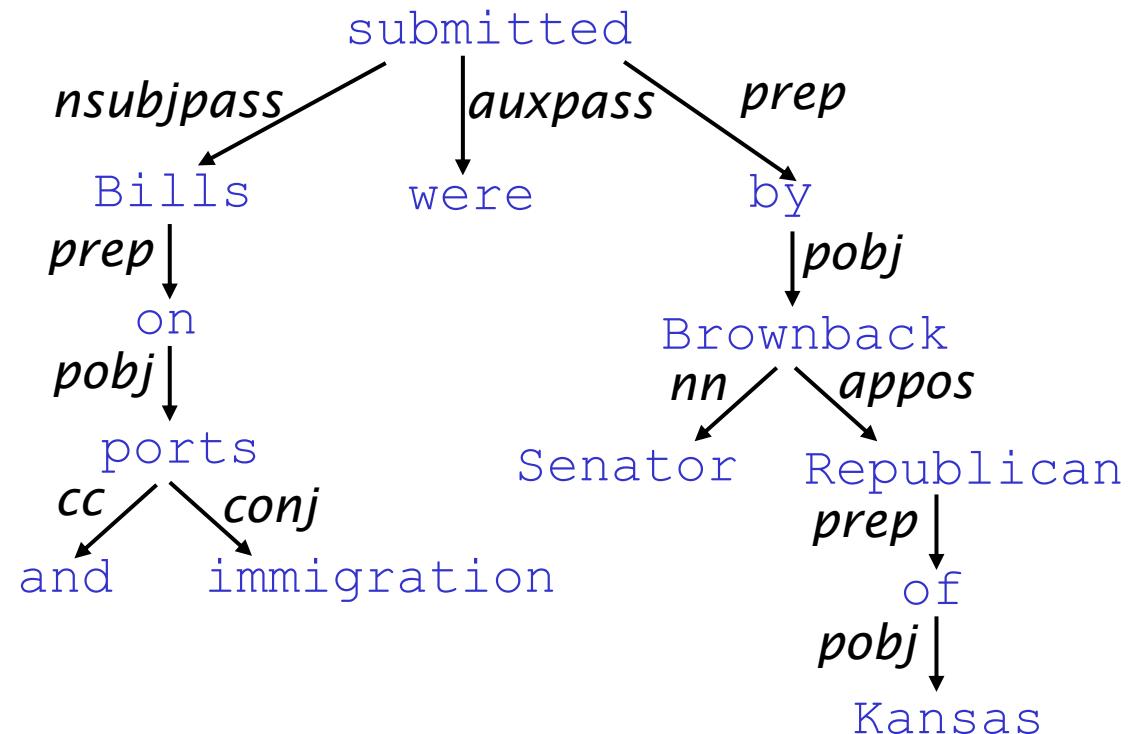


I saw a girl with a telescope

# Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies

The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc.)

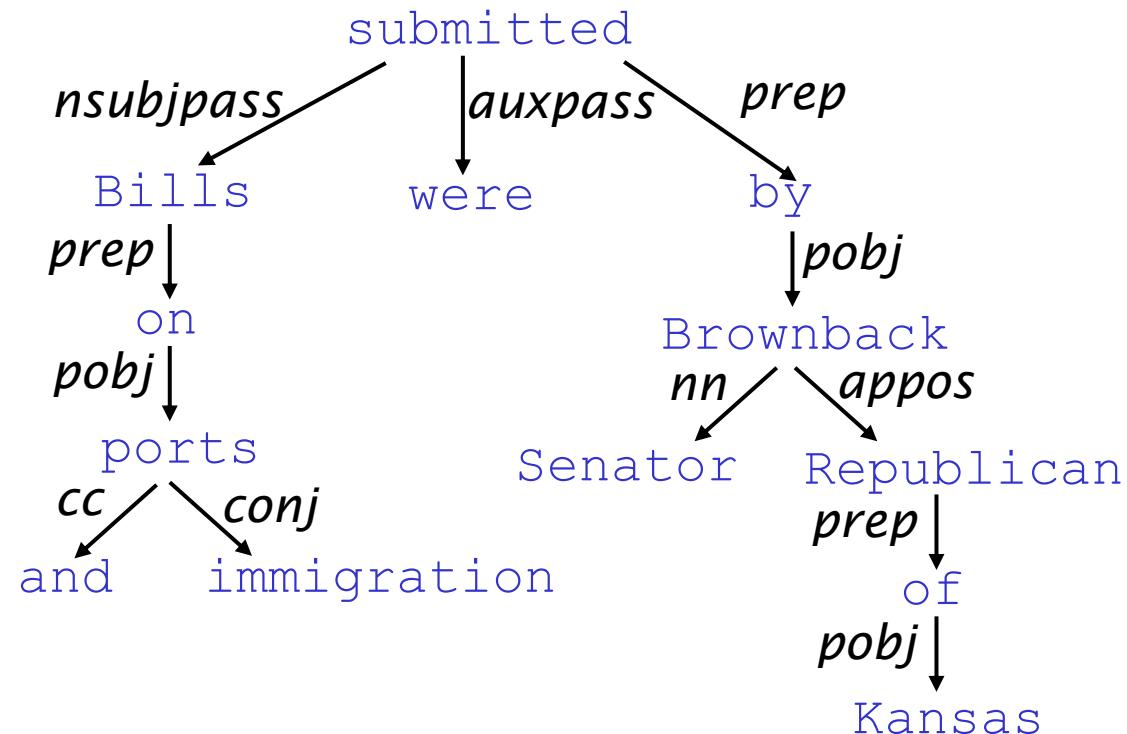


# Dependency Grammar and Dependency Structure

**Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies**

The arrow connects a **head** (governor) with a **dependent** (modifier)

Usually, dependencies form a tree  
(connected, acyclic,  
single-head)



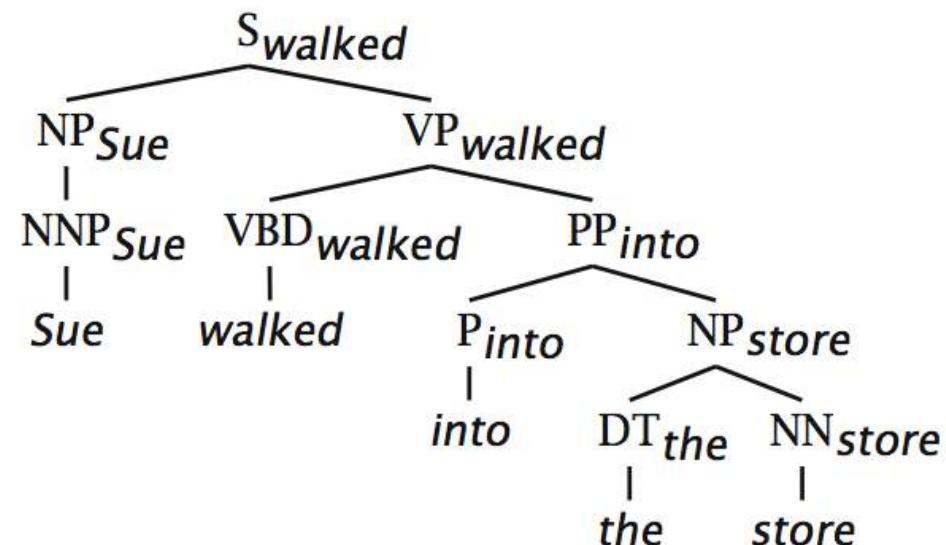
# Relation between phrase structure and dependency structure

A dependency grammar has a notion of a head. Officially, CFGs don't. But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, ...) do, via hand-written phrasal "head rules":

- The head of a Noun Phrase is a noun/number/adj/...
- The head of a Verb Phrase is a verb/modal/....

The head rules can be used to extract a dependency parse from a CFG parse

- The closure of dependencies give constituency from a dependency tree
- But the dependents of a word must be at the same level (i.e., "flat")



# Dependency graph

- A dependency structure can be defined as a directed graph  $G$ , consisting of
  - ▶ a set  $V$  of nodes,
  - ▶ a set  $A$  of arcs (edges),
- Labeled graphs:
  - ▶ Nodes in  $V$  are labeled with word forms (and annotation).
  - ▶ Arcs in  $A$  are labeled with dependency types.
- Notational convention:
  - ▶ Arc  $(w_i, d, w_j)$  links head  $w_i$  to dependent  $w_j$  with label  $d$
  - ▶  $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
  - ▶  $i \rightarrow j \equiv (i, j) \in A$
  - ▶  $i \rightarrow^* j \equiv i = j \vee \exists k : i \rightarrow k, k \rightarrow^* j$

# Formal conditions on dependency graph

---

- $G$  is connected:
  - ▶ For every node  $i$  there is a node  $j$  such that  $i \rightarrow j$  or  $j \rightarrow i$ .
- $G$  is acyclic:
  - ▶ if  $i \rightarrow j$  then not  $j \rightarrow^* i$ .
- $G$  obeys the single head constraint:
  - ▶ if  $i \rightarrow j$  then not  $k \rightarrow j$ , for any  $k \neq i$ .
- $G$  is projective:
  - ▶ if  $i \rightarrow j$  then  $j \rightarrow^* k$ , for any  $k$  such that both  $j$  and  $k$  lie on the same side of  $i$ .

# Universal dependencies

<http://universaldependencies.org/>

- Annotated treebanks in many languages
- Uniform annotation scheme across all languages:
  - Universal POS tags
  - Universal dependency relations

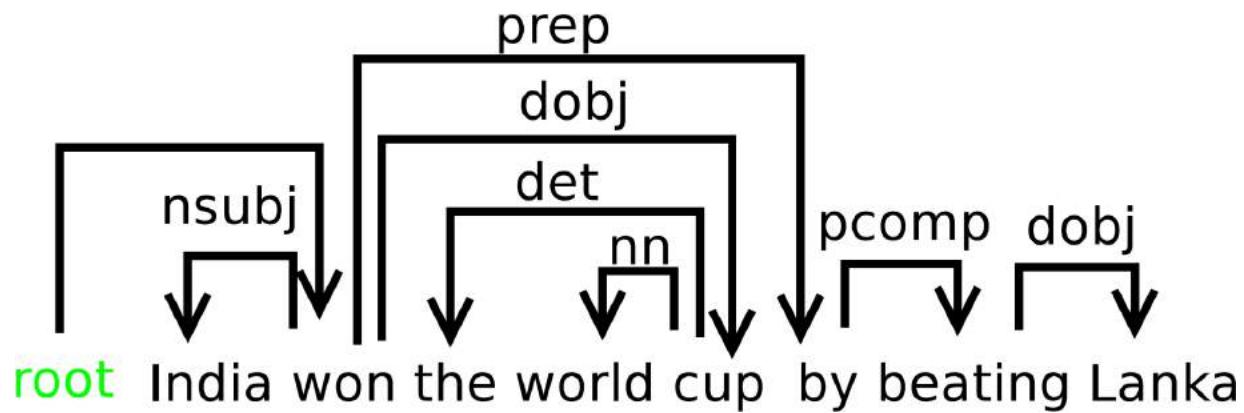
# Dependency Relations

<b>Argument Dependencies</b>	<b>Description</b>
<b>nsubj</b>	nominal subject
<b>csubj</b>	clausal subject
<b>dobj</b>	direct object
<b>iobj</b>	indirect object
<b>pobj</b>	object of preposition

<b>Modifier Dependencies</b>	<b>Description</b>
<b>tmod</b>	temporal modifier
<b>appos</b>	appositional modifier
<b>det</b>	determiner
<b>prep</b>	prepositional modifier

# Example Dependency Parse



# Methods of Dependency Parsing

## 1. Dynamic programming (like in the CKY algorithm)

You can do it similarly to lexicalized PCFG parsing: an  $O(n^5)$  algorithm Eisner (1996) gives a clever algorithm that reduces the complexity to  $O(n^3)$ , by producing parse items with heads at the ends rather than in the middle

## 2. Graph algorithms

You create a Maximum Spanning Tree for a sentence

## 3. Constraint Satisfaction

Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

## 4. Deterministic parsing

Greedy choice of attachments guided by machine learning classifiers MaltParser (Nivre et al. 2008) – discussed in the next segment

# Deterministic parsing

## *Basic idea*

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

## *Configurations*

A parser configuration is a triple  $c = (S, B, A)$ , where

- $S$  : a stack  $[ \dots, w_i ]_S$  of partially processed words,
- $B$  : a buffer  $[ w_j, \dots ]_B$  of remaining input words,
- $A$  : a set of labeled arcs  $(w_i, d, w_j)$ .

### Stack

$[\text{sent}, \text{her}, \text{a}]_S$

### Buffer

$[\text{letter}, .]_B$

### Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

# Transition based systems for Dependency parsing

A transition system for dependency parsing is a quadruple  $S = (C, T, c_s, C_t)$ , where

- $C$  is a set of configurations,
- $T$  is a set of transitions, such that  $t : C \rightarrow C$ ,
- $c_s$  is an initialization function
- $C_t \subseteq C$  is a set of terminal configurations.

A transition sequence for a sentence  $x$  is a set of configurations

$C_{0,m} = (c_0, c_1, \dots, c_m)$  such that

$c_0 = c_s(x)$ ,  $c_m \in C_t$ ,  $c_i = t(c_{i-1})$  for some  $t \in T$

**Initialization:**  $([], [w_1, \dots, w_n]_B, \{\})$

**Termination:**  $(S, [], A)$

# Arc eager parsing(Malt parser)

Left-Arc( $d$ )  $\frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots]_S, [w_j, \dots]_B, A \cup \{(w_j, d, w_i)\})} \neg \text{HEAD}(w_i)$

Right-Arc( $d$ )  $\frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots, w_i, w_j]_S, [\dots]_B, A \cup \{(w_i, d, w_j)\})}$

Reduce  $\frac{([\dots, w_i]_S, B, A)}{([\dots]_S, B, A)} \text{HEAD}(w_i)$

Shift  $\frac{([\dots]_S, [w_i, \dots]_B, A)}{([\dots, w_i]_S, [\dots]_B, A)}$

# Arc Eager Parsing example

## Transitions:

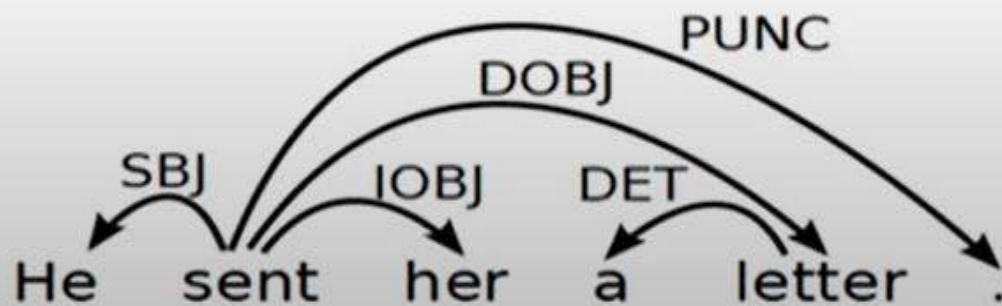
**Stack**

! ls

**Buffer**

[He, sent, her, a, letter, .]\_B

**Arcs**



# Arc Eager Parsing example

**Transitions:** SH-LA

**Stack**

[ ]s

**Buffer**

[sent, her, a, letter, .]B

**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent



# Arc Eager Parsing example

**Transitions:** SH-LA-SH

## Stack

[sent]<sub>S</sub>

## Buffer

[her, a, letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA

## Stack

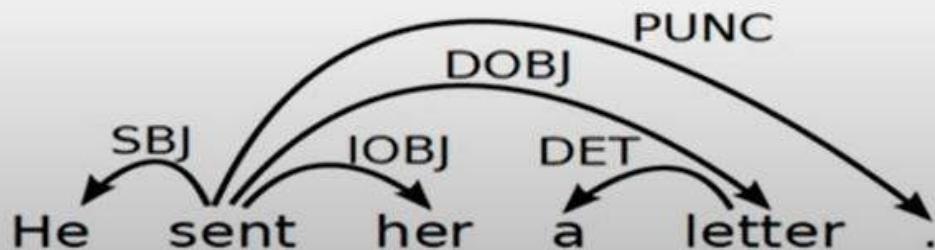
[sent, her]<sub>S</sub>

## Buffer

[a, letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH

**Stack**

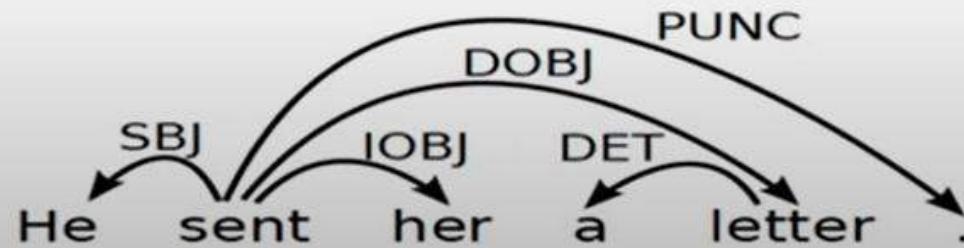
[sent, her, a]<sub>S</sub>

**Buffer**

[letter, .]<sub>B</sub>

**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH-LA

## Stack

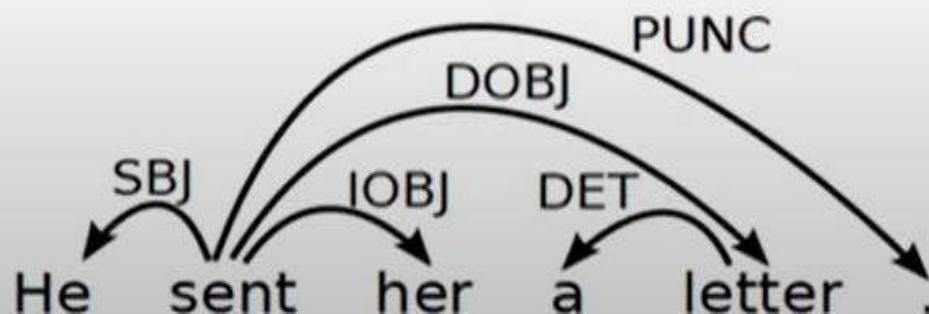
[sent, her]<sub>S</sub>

## Buffer

[letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter



# Arc Eager Parsing example

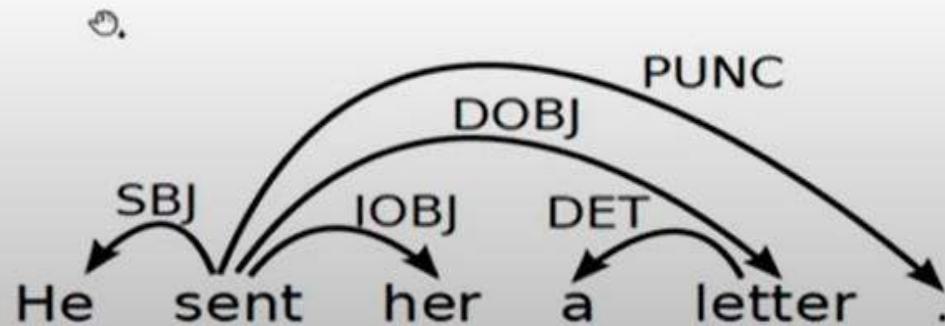
**Transitions:** SH-LA-SH-RA-SH-LA-RE

## Stack

[sent]<sub>S</sub>

## Buffer

[letter, .]<sub>B</sub>



## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter

# Arc Eager Parsing example

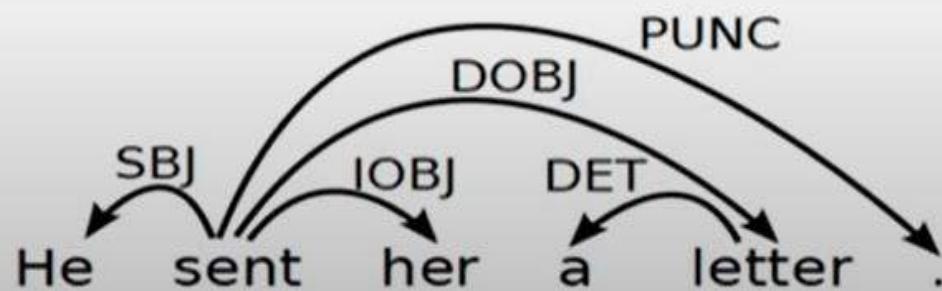
**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA

## Stack

[sent, letter]s

## Buffer

[.]<sub>B</sub>



## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter

# Arc Eager Parsing example

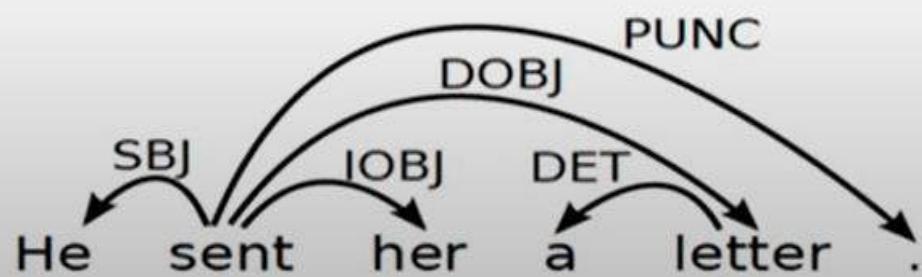
**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE

## Stack

[sent]<sub>S</sub>

## Buffer

[.]<sub>B</sub>



## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter

# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

## Stack

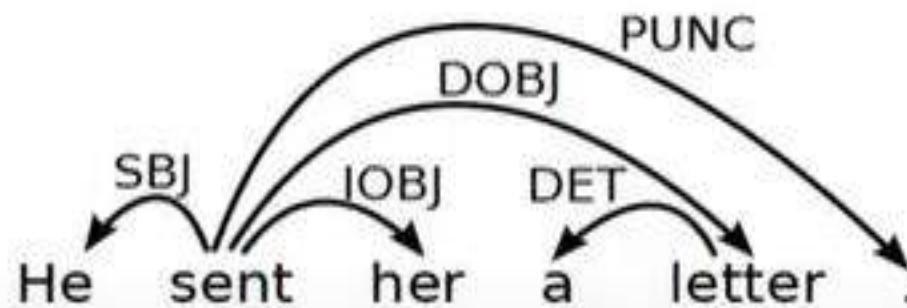
[sent, .]s

## Buffer

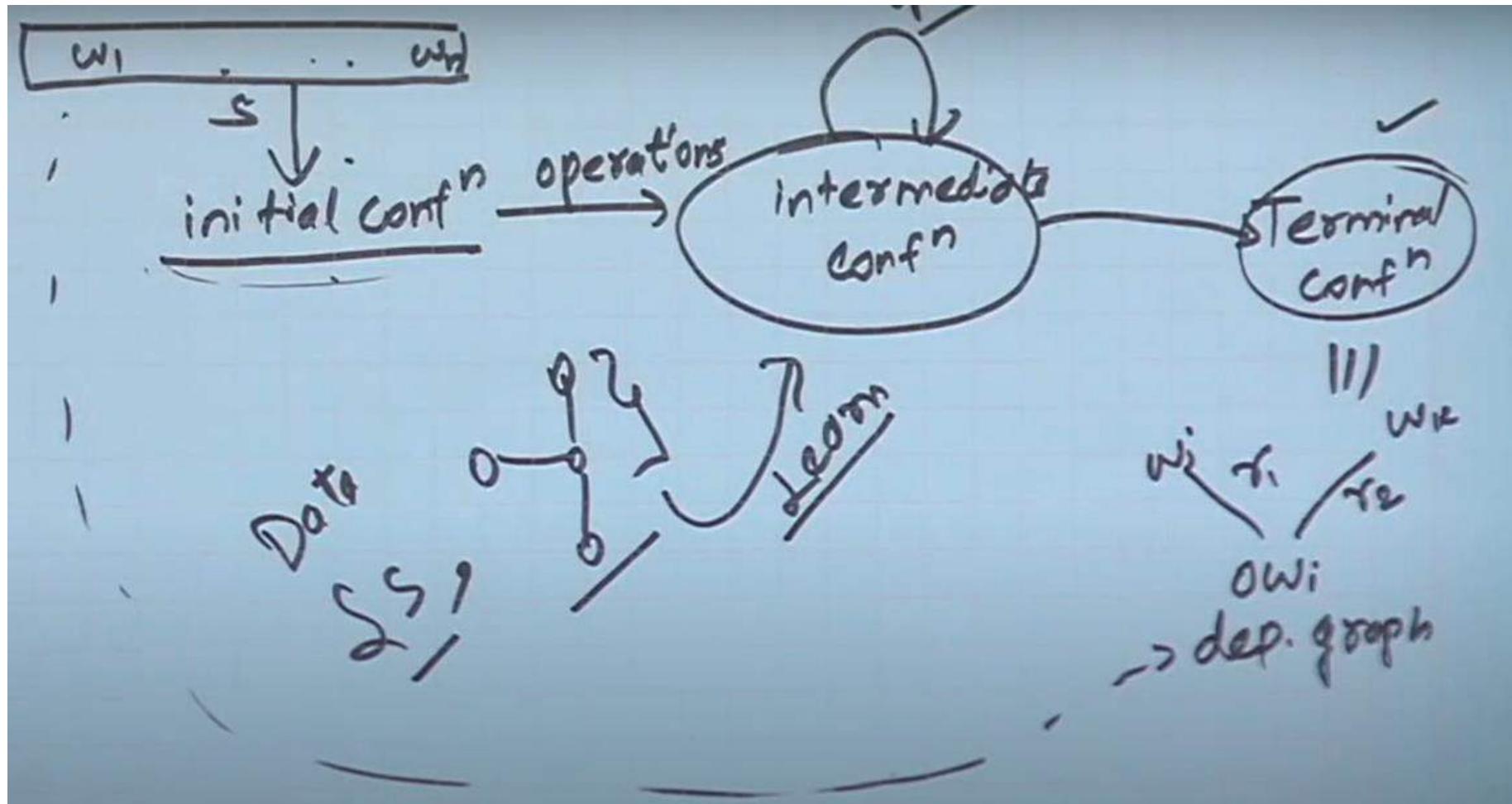
[ ]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter  
 sent  $\xrightarrow{\text{PUNC}}$



# Learning weights

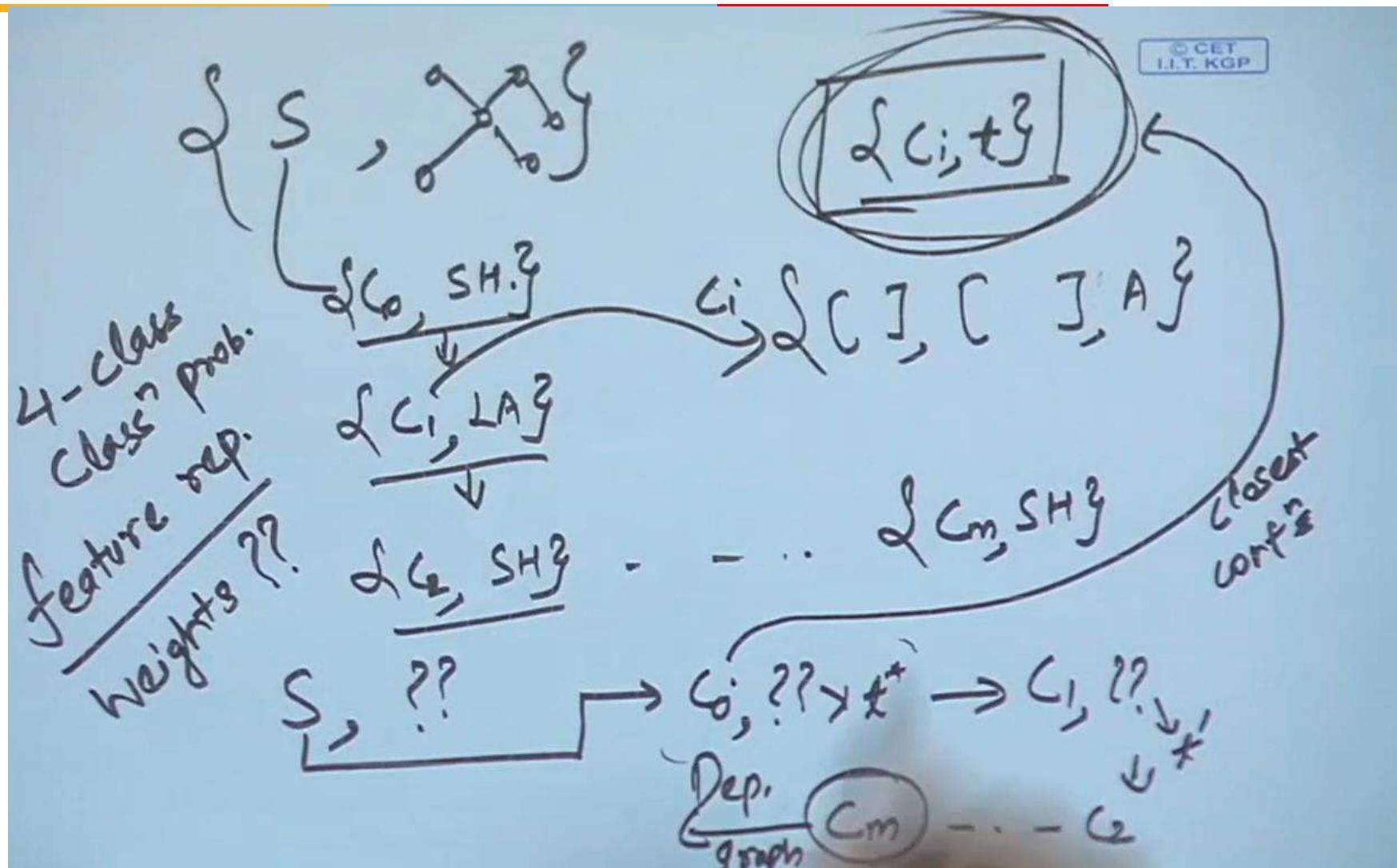


# Deriving dependency graph for a new sentence

Innovate

achieve

lead



# Classifier for learning the transmission

*Data-driven deterministic parsing:*

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

*Learning Problem*

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

*Three issues*

- How to represent configurations by feature vectors?
- How to derive training data from treebanks?
- How to learn classifiers?

# Feature Models

A feature representation  $f(c)$  of a configuration  $c$  is a vector of simple features  $f_i(c)$ .

## *Typical Features*

- Nodes:
  - Target nodes (top of  $S$ , head of  $B$ )
  - Linear context (neighbors in  $S$  and  $B$ )
  - Structural context (parents, children, siblings in  $G$ )
- Attributes:
  - Word form (and lemma)
  - Part-of-speech (and morpho-syntactic features)
  - Dependency type (if labeled)
  - Distance (between target tokens)

To guide the parser, a linear classifier can be used:

$$t^* = \arg \max_t w.f(c, t)$$

Weight vector  $w$  learned from treebank data.

*Using classifier at run-time*

PARSE( $w_1, \dots, w_n$ )

- 1       $c \leftarrow ([], [w_1, \dots, w_n]_B, \{\})$
- 2      **while**  $B_c \neq []$
- 3             $t^* \leftarrow \arg \max_t w.f(c, t)$
- 4             $c \leftarrow t^*(c)$
- 5      **return**  $T = (\{w_1, \dots, w_n\}, A_c)$

# Training Data

- Training instances have the form  $(f(c), t)$ , where
  - $f(c)$  is a feature representation of a configuration  $c$ ,
  - $t$  is the correct transition out of  $c$  (i.e.,  $o(c) = t$ ).
- Given a dependency treebank, we can sample the oracle function  $o$  as follows:
  - For each sentence  $x$  with gold standard dependency graph  $G_x$ , construct a transition sequence  $C_{0,m} = (c_0, c_1, \dots, c_m)$  such that
$$c_0 = c_s(x),$$
$$G_{c_m} = G_x$$
  - For each configuration  $c_i (i < m)$ , we construct a training instance  $(f(c_i), t_i)$ , where  $t_i(c_i) = c_{i+1}$ .

# Standard Oracle for Arc Eager parsing

$o(c, T) =$

- **Left-Arc** if  $\text{top}(S_c) \leftarrow \text{first}(B_c)$  in  $T$
- **Right-Arc** if  $\text{top}(S_c) \rightarrow \text{first}(B_c)$  in  $T$
- **Reduce** if  $\exists w < \text{top}(S_c) : w \leftrightarrow \text{first}(B_c)$  in  $T$
- **Shift** otherwise

# Online learning with an oracle

```
LEARN({ $T_1, \dots, T_N$ })
1    $w \leftarrow 0.0$ 
2   for  $i$  in  $1..K$ 
3     for  $j$  in  $1..N$ 
4        $c \leftarrow ([], [w_1, \dots, w_{n_j}]_B, \{\})$ 
5       while  $B_c \neq []$ 
6          $t^* \leftarrow \arg \max_t w.f(c, t)$ 
7          $t_o \leftarrow o(c, T_i)$ 
8         if  $t^* \neq t_o$ 
9            $w \leftarrow w + f(c, t_o) - f(c, t^*)$ 
10           $c \leftarrow t_o(c)$ 
11    return  $w$ 
```

Oracle  $o(c, T_i)$  returns the optimal transition of  $c$  and  $T_i$

# Example

Consider the sentence, 'John saw Mary'.

- Draw a dependency graph for this sentence.
- Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are 'Nouns', while the POS tag of *saw* is 'Verb'. Assume that your features correspond to the following conditions:
  - The stack is empty
  - Top of stack is Noun and Top of buffer is Verb
  - Top of stack is Verb and Top of buffer is Noun

Initialize the weights of all your features to 5.0, except that in all of the above cases, you give a weight of 5.5 to *Left-Arc*. Define your feature vector and the initial weight vector.

- Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

# Example

$F(c,t) = [(c_0, LA), (c_1, LA), (c_2, LA) | (c_0, RA), (c_1, RA), (c_2, RA) \dots]$

$W = [5.5, 5.5, 5.5 | 5.0, 5.0, 5.0 | 5.0, 5.0, 5.0 | \dots]$

So for the given conditions

$F(c, LA) = [1, 0, 0 | 0, 0, 0 | \dots]$

$F(c, RA) = [0, 0, 0 | 1, 0, 0 | \dots]$

$t^* = \text{argmax}(w^* f(c, t))$

$t^* = LA$

as per oracle /optimal transition  $t^0 = SH$

So we need to update the weights|

To update the weights

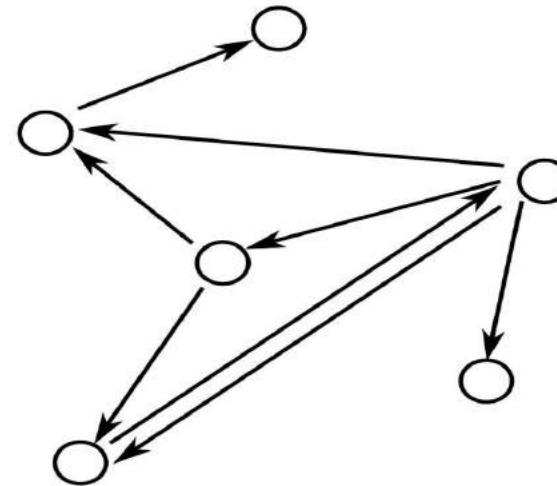
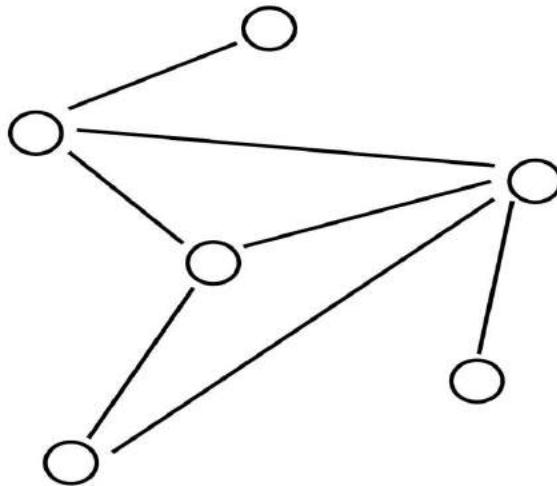
$W = W + f(c, t^0) - f(c, t^*)$

$W = [5.5, 5.0, 5.0 | 5.0, 5.0 \dots] + [0, 0, 0 | 0, 0, 0 | \dots 1, 0, 0] - [1, 0, 0 | 0, 0, 0 | \dots]$

New vector =  $[4.5, 5.5, 5.5 | 5.0 \dots 6.0, 5.0, 5.0]$

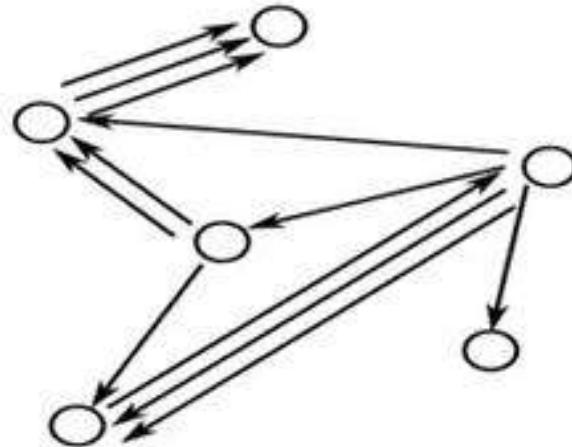
# Graph

- ▶ A graph  $G = (V, A)$  is a set of vertices  $V$  and arcs  $(i, j) \in A$ , where  $i, j \in V$
- ▶ Undirected graphs:  $(i, j) \in A \Leftrightarrow (j, i) \in A$
- ▶ **Directed graphs (digraphs):**  $(i, j) \in A \not\Leftrightarrow (j, i) \in A$



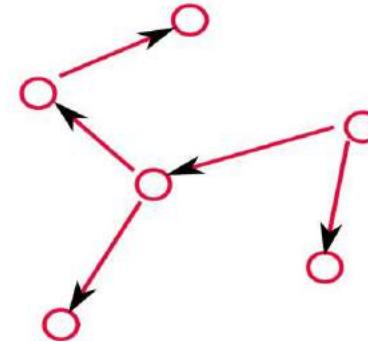
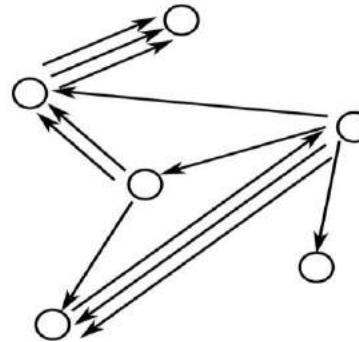
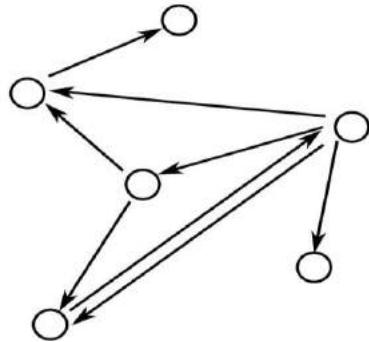
# Multi Digraph

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i, j, k) \in A$  represents the  $k^{th}$  arc from vertex  $i$  to vertex  $j$ .



# Directed Spanning Trees

- ▶ A directed spanning tree of a (multi-)digraph  $G = (V, A)$ , is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- ▶ A spanning tree of the following (multi-)digraphs



# Weighted Spanning tree

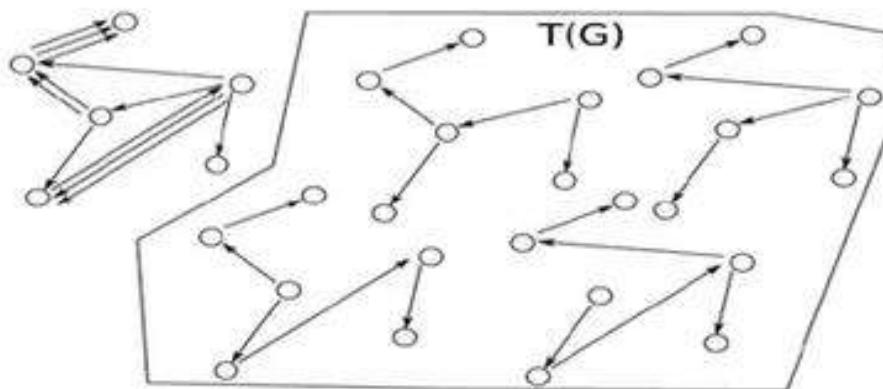
---

- Assume we have a weight function for each arc in a multi-digraph  $G = (V, A)$ .
- Define  $w_{ij}^k \geq 0$  to be the weight of  $(i, j, k) \in A$  for a multi-digraph
- Define the weight of directed spanning tree  $G'$  of graph  $G$  as

$$w(G') = \sum_{(i,j,k) \in G'} w_{ij}^k$$

# MST

Let  $T(G)$  be the set of all spanning trees for graph  $G$



The MST problem

Find the spanning tree  $G'$  of the graph  $G$  that has the highest weight

$$G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \sum_{(i,j,k) \in G'} w_{ij}^k$$

# Finding MST

## Directed Graph

For each sentence  $x$ , define the directed graph  $G_x = (V_x, E_x)$  given by

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

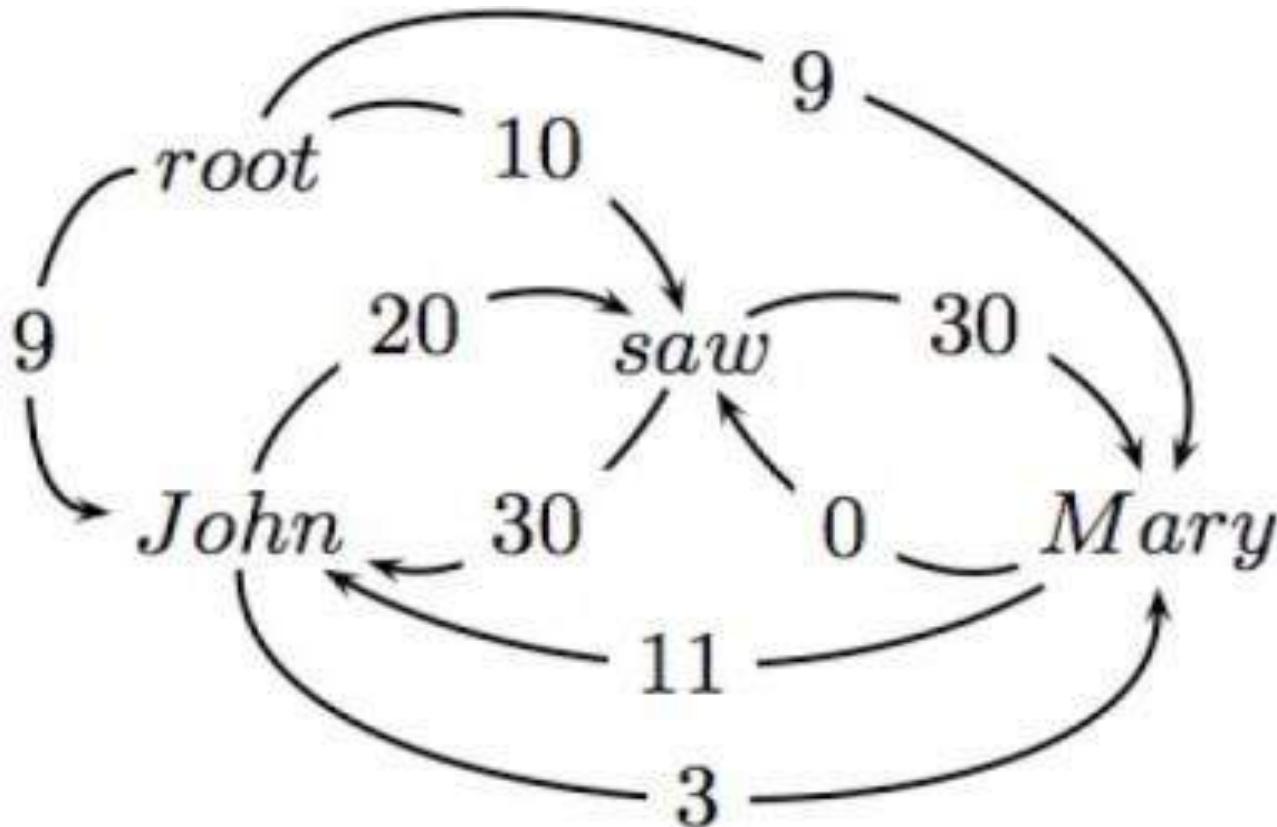
$G_x$  is a graph with

- the sentence words and the dummy root symbol as vertices and
- a directed edge between every pair of distinct words and
- a directed edge from the root symbol to every word

# Chu-Liu-Edmonds algorithm

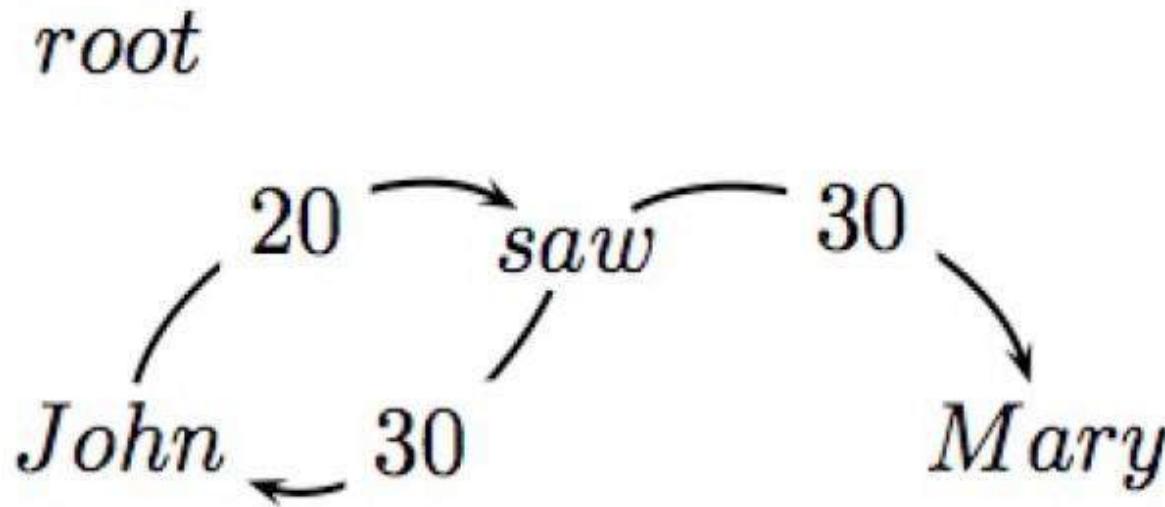
- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - Identify the cycle and contract it into a single vertex.
  - Recalculate edge weights going into and out of the cycle.

# Chu-Liu-Edmonds Example



# Chu-Liu-Edmonds Example

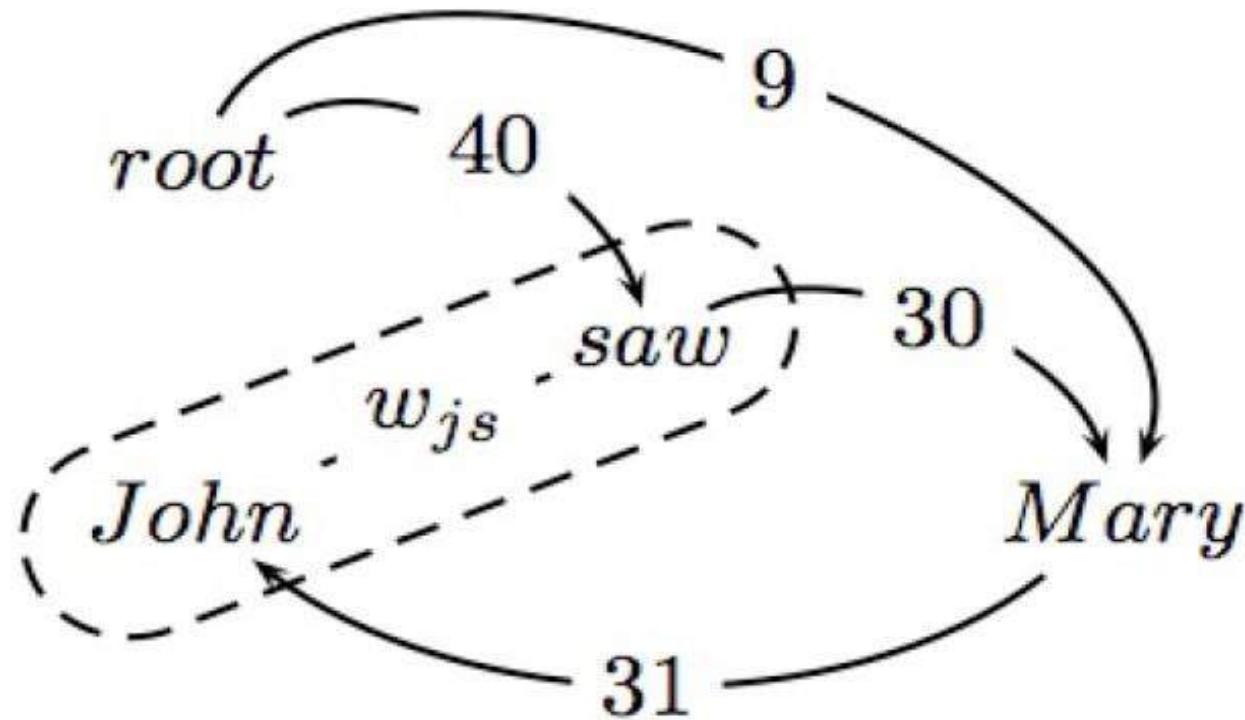
- ▶ Find highest scoring incoming arc for each vertex



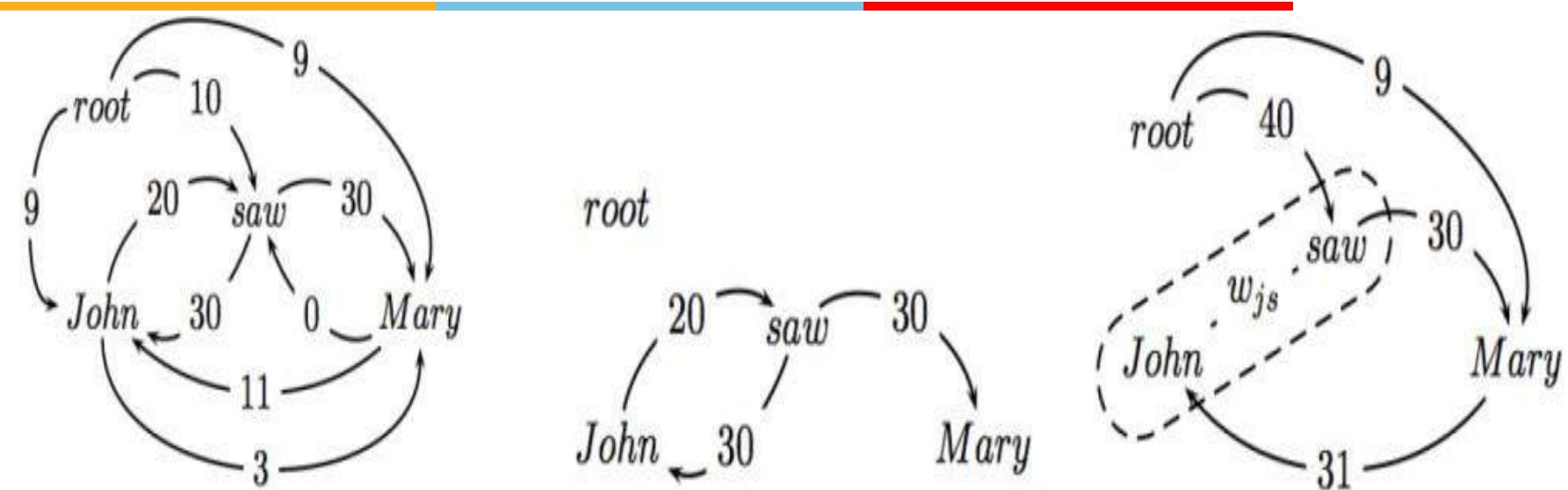
- ▶ If this is a tree, then we have found MST!!

# Chu-Liu-Edmonds Example

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle

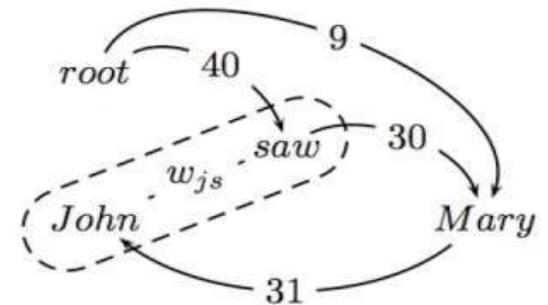
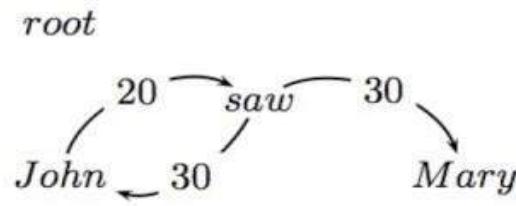
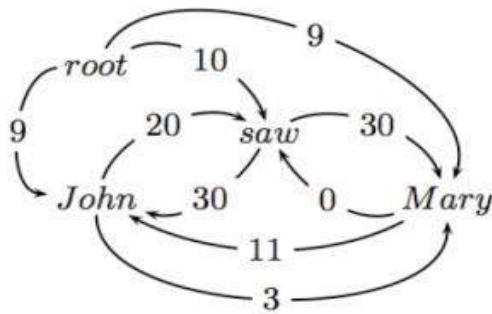


# Chu-Liu-Edmonds Example



- ▶ Outgoing arc weights
  - ▶ Equal to the max of outgoing arc over all vertexes in cycle
  - ▶ e.g., John → Mary is 3 and saw → Mary is 30

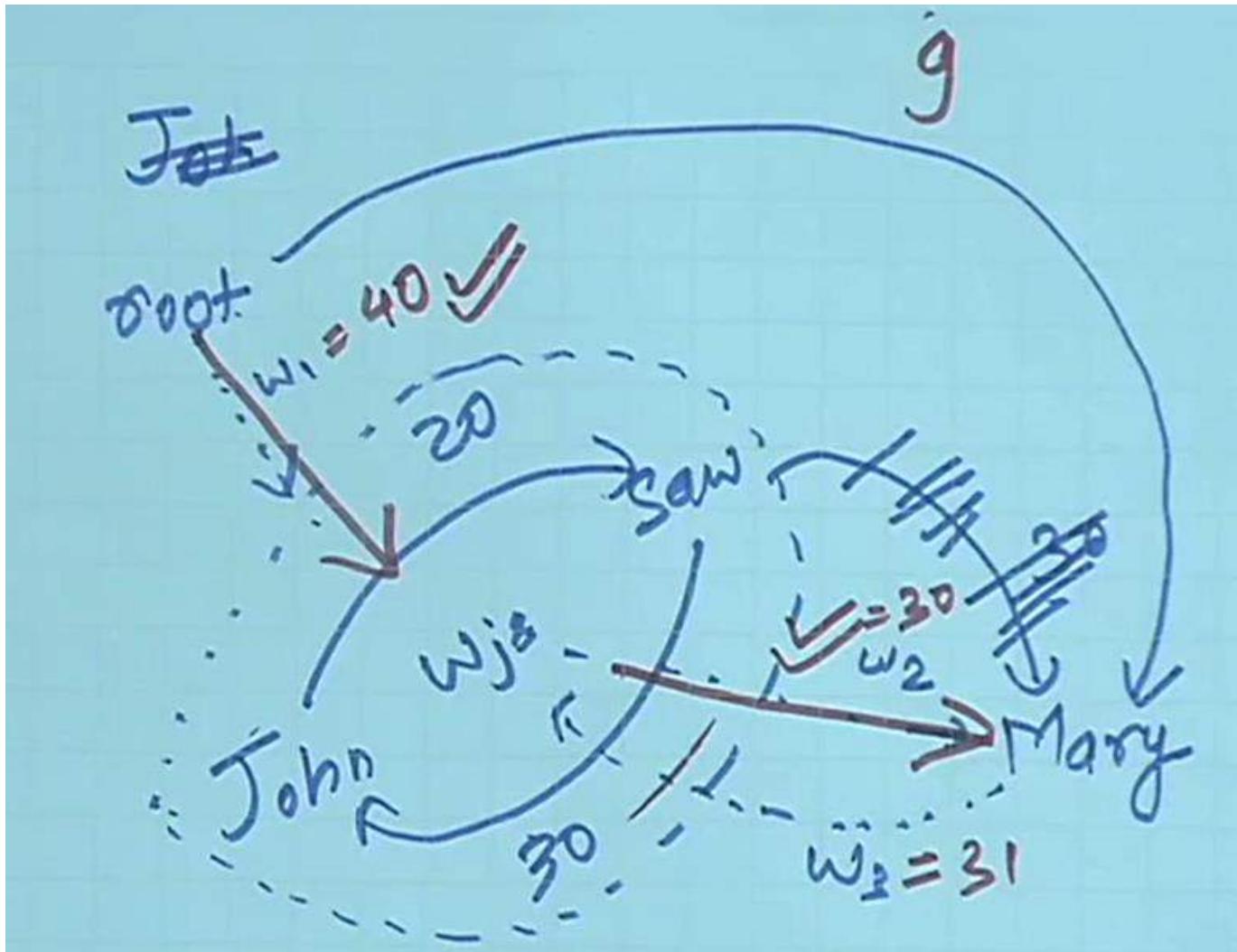
# Chu-Liu-Edmonds Example



## ► Incoming arc weights

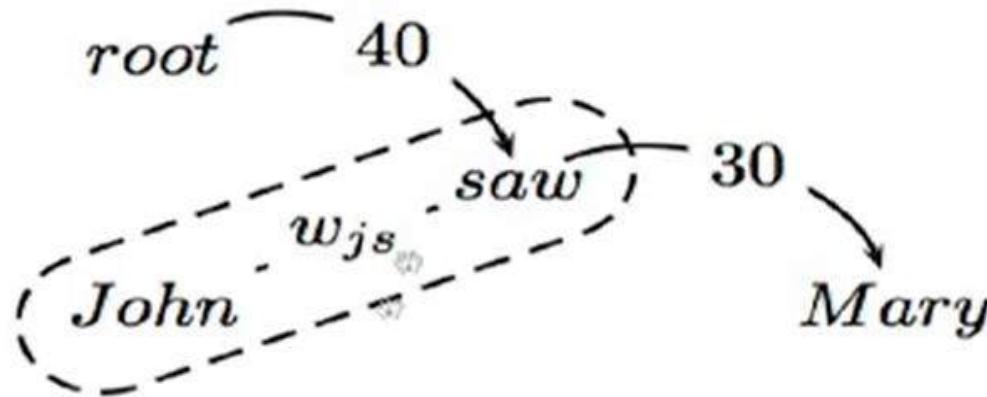
- ▶ Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- ▶  $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40 (\*\*)
- ▶  $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

# Chu-Liu-Edmonds Example



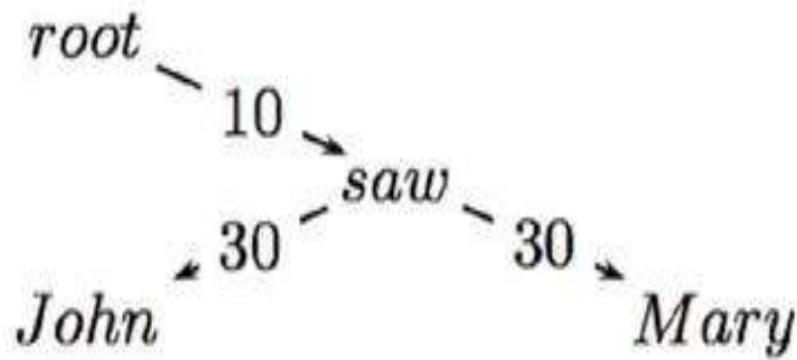
# Chu-Liu-Edmonds Example

Calling the algorithm again on the contracted graph:



- This is a tree and the MST for the contracted graph
- Go back up the recursive call and reconstruct final graph

# Chu-Liu-Edmonds Example



- The edge from  $w_{js}$  to *Mary* was from *saw*
- The edge from *root* to  $w_{js}$  represented a tree from *root* to *saw* to *John*.

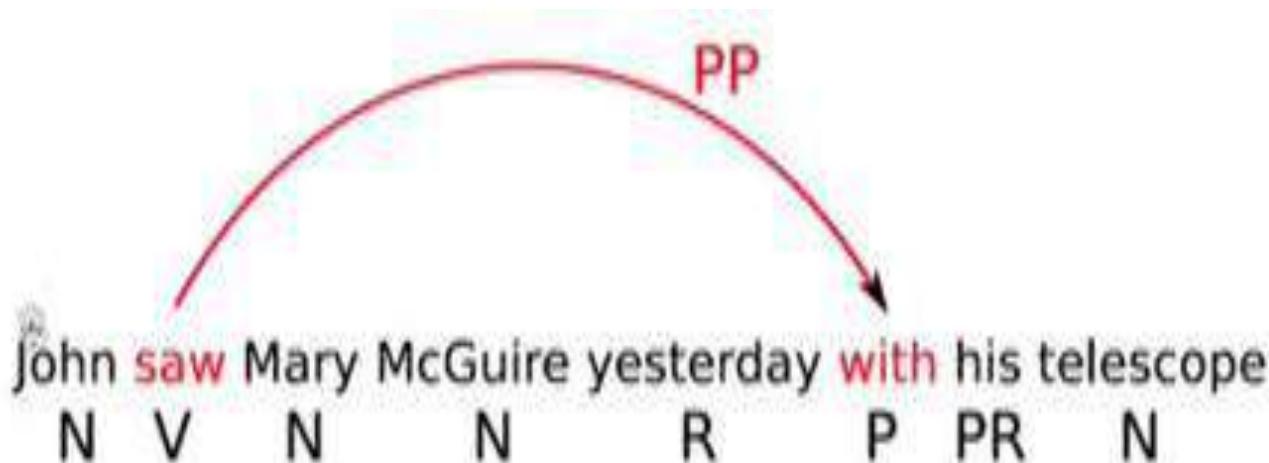
# Linear classifiers

---

$$w_{ij}^k = w.f(i,j,k)$$

- Arc weights are a linear combination of features of the arc  $f(i,j,k)$  and a corresponding weight vector  $w$
- What arc features?

# Arc features



## Features

Identities of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head = saw & dependent=with

# Arc features



## Features

Part-of-speech tags of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head-pos = Verb & dependent-pos=Preposition

# Arc features



## Features

Part-of-speech of words surrounding and between  $w_i$  and  $w_j$

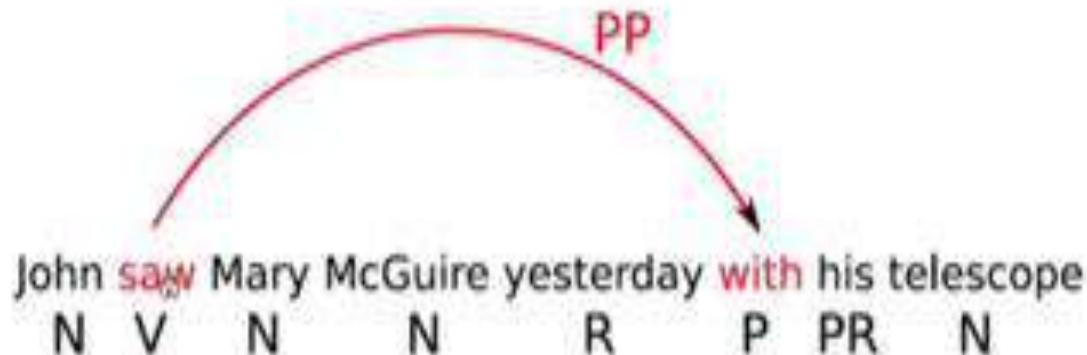
inbetween-pos = Noun

inbetween-pos = Adverb

dependent-pos-right = Pronoun

head-pos-left=Noun

# Arc features



## Features

Number of words between  $w_i$  and  $w_j$ , and their orientation

arc-distance = 3

arc-direction = right

# Learning the parameters

---

- Re-write the inference problem

$$\begin{aligned} G &= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w_{ij}^k \\ &= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) \\ &= \arg \max_{G \in T(G_x)} w \cdot f(G) \end{aligned}$$

# Inference based learning

---

Training data:  $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

1.  $w^{(0)} = 0; i = 0$
2. **for**  $n : 1..N$
3.     **for**  $t : 1..|T|$
4.         Let  $G' = argmax_{G'} w^{(i)}.f(G')$
5.         if  $G' \neq G_t$ ,
6.              $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7.          $i = i + 1$
8.     **return**  $w^i$

# EXAMPLE

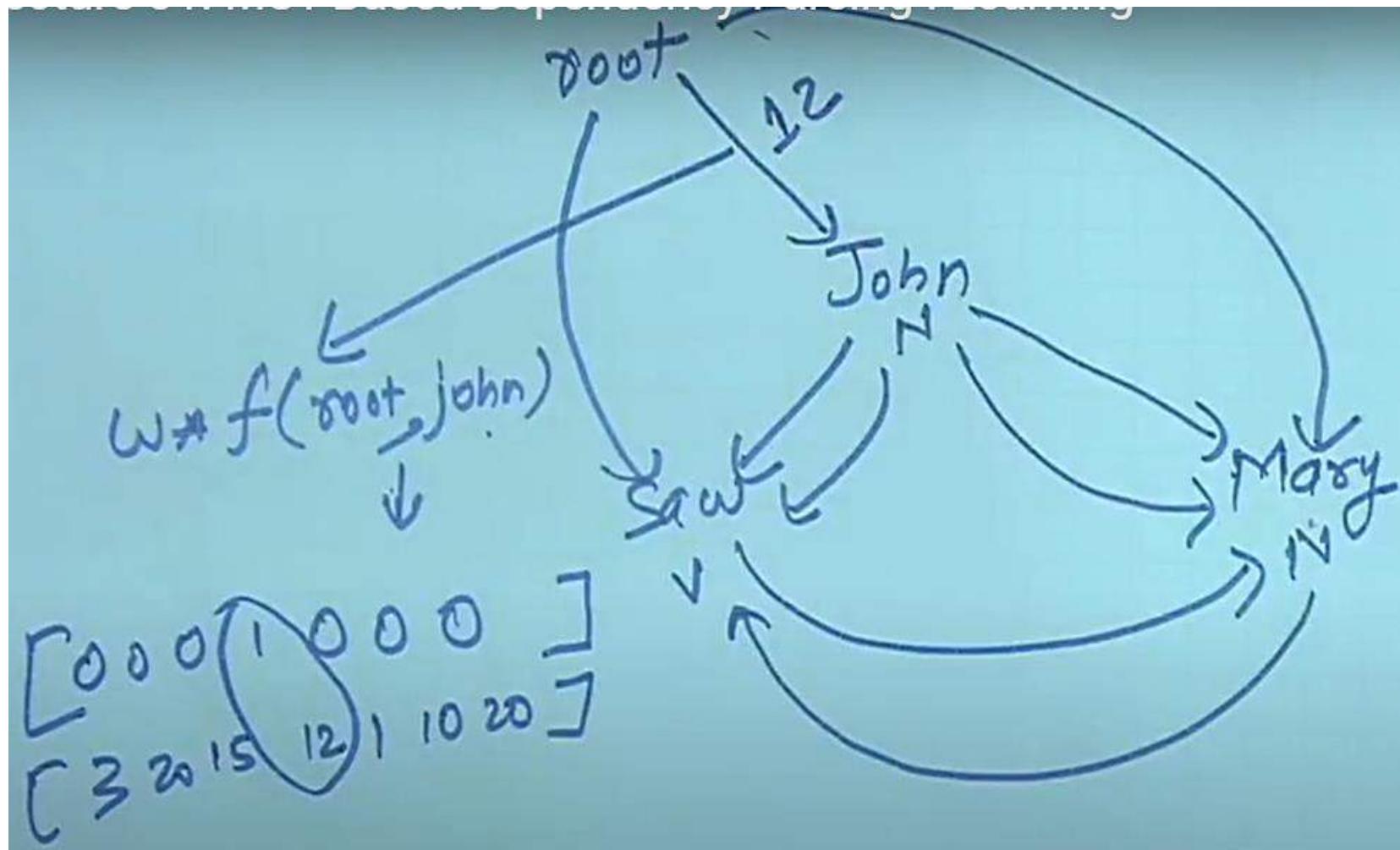
Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word  $w_i$  to  $w_j$ , your features may be simplified to depend only on words  $w_i$  and  $w_j$  and not on the relation label.

Below is the set of features

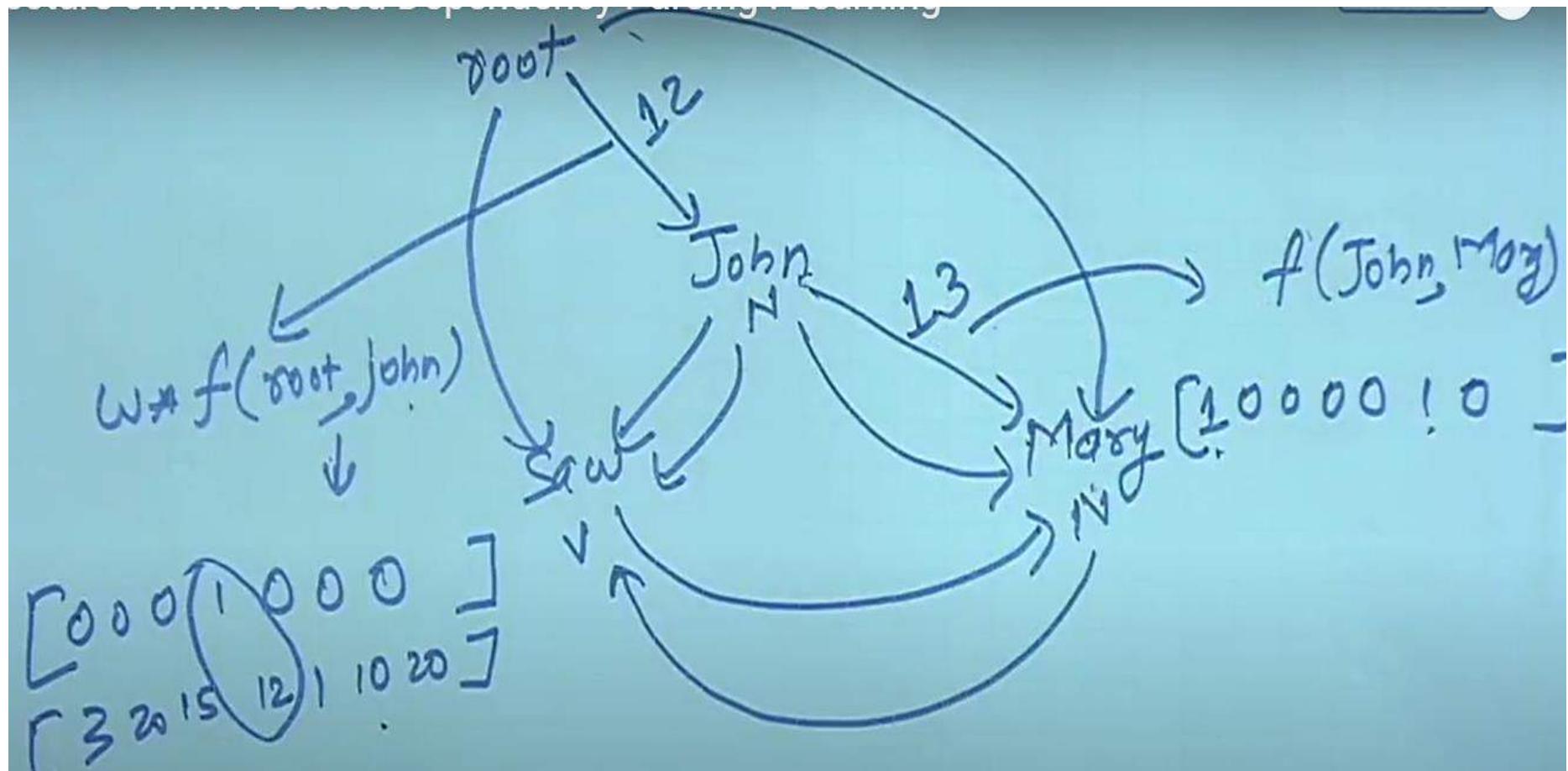
- $f_1: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_2: \text{pos}(w_i) = \text{Verb}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_3: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Verb}$
- $f_4: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_5: w_i = \text{Root}$  and  $w_j$  occurs at the end of sentence
- $f_6: w_i$  occurs before  $w_j$  in the sentence
- $f_7: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Verb}$

The feature weights before the start of the iteration are: {3, 20, 15, 12, 1, 10, 20}. Determine the weights after an iteration over this example.

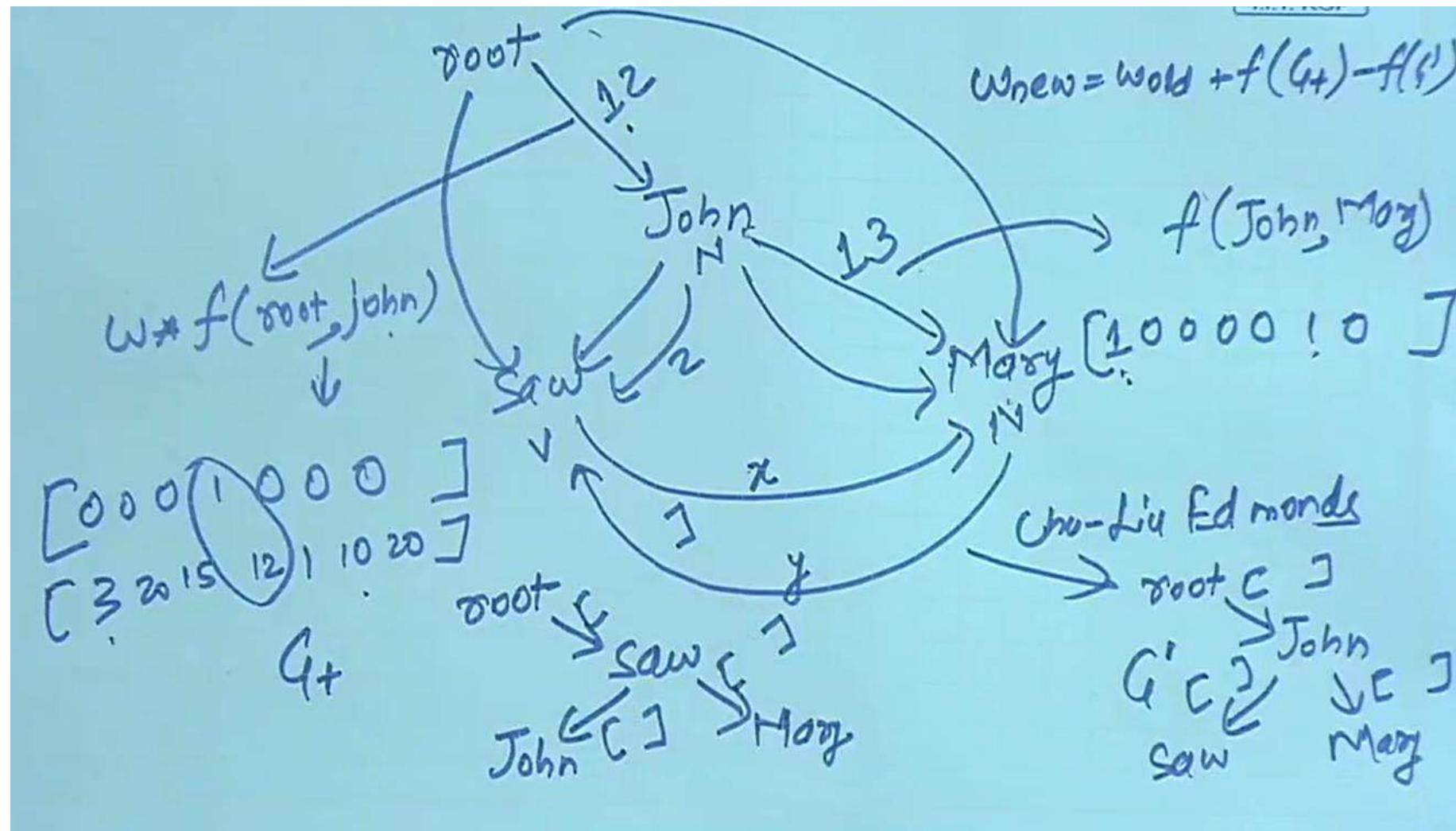
# EXAMPLE



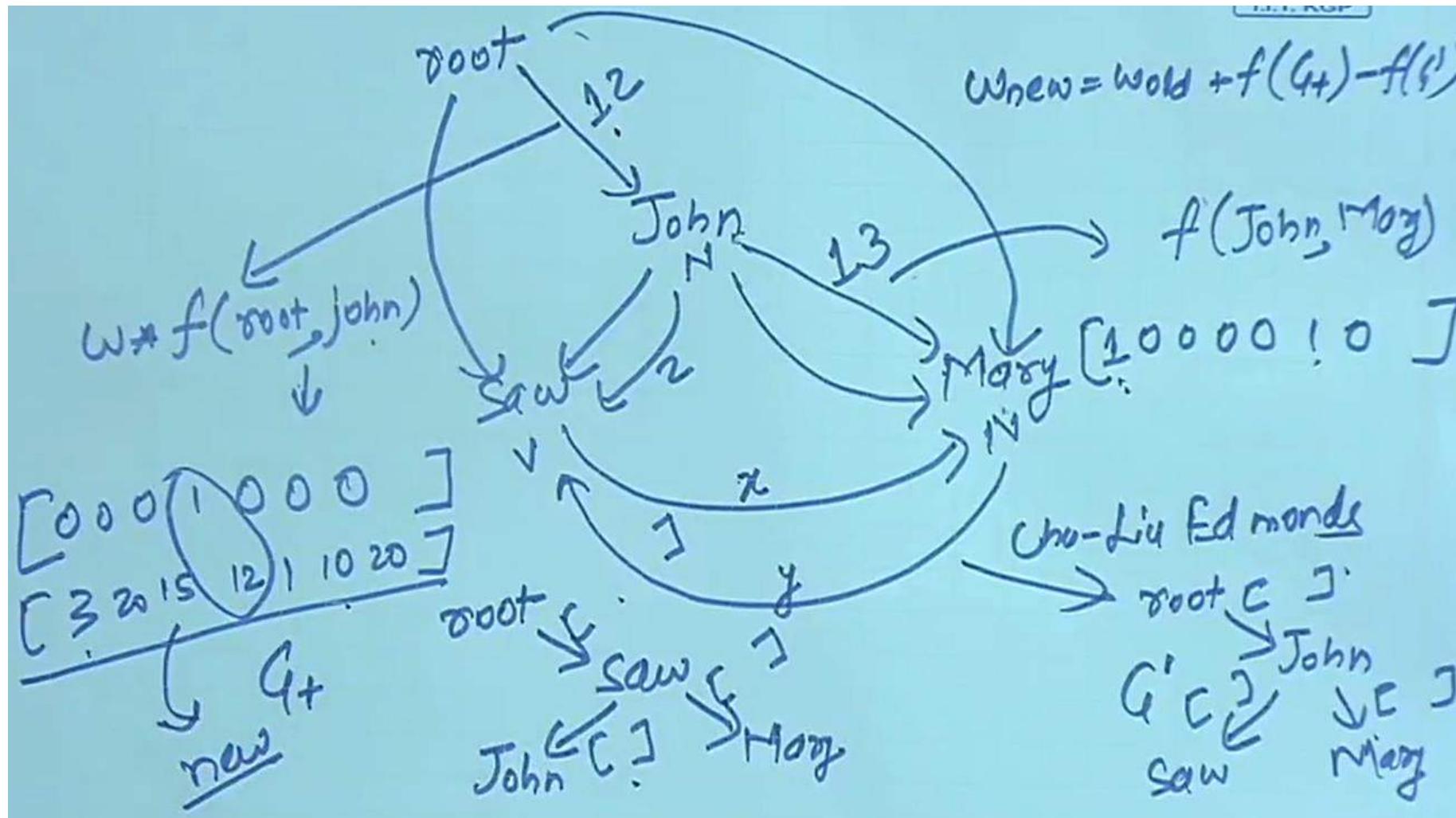
# EXAMPLE



# EXAMPLE



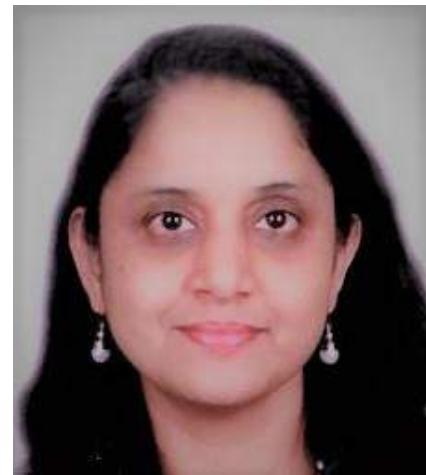
# EXAMPLE



# Extra Reading

---

- Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition].
- <https://www.youtube.com/watch?v=PVShkZgXznc>
- [https://www.youtube.com/watch?v=R1wL7sA\\_hHM&list=PLzJaFd3A7DZutMK8fFxZx\\_mhmFQqzijGE&index=28](https://www.youtube.com/watch?v=R1wL7sA_hHM&list=PLzJaFd3A7DZutMK8fFxZx_mhmFQqzijGE&index=28)
- [https://www.researchgate.net/publication/328731166\\_Weighted\\_Machine\\_Learning](https://www.researchgate.net/publication/328731166_Weighted_Machine_Learning)



**Thank you for your time!!**



**BITS** Pilani  
Pilani Campus

## **Session 8-Midsem Review**

**Date – 17<sup>th</sup> October 2020**

**Time – 9 am to 11 am**

Dr. Chetana Gavankar, Ph.D,  
IIT Bombay-Monash University Australia  
[Chetana.gavankar@pilani.bits-pilani.ac.in](mailto:Chetana.gavankar@pilani.bits-pilani.ac.in)

# Session 8: Review of session 1 to 7

---

- Session 1 Review: **Introduction**
- Session 2 Review: **Language Models**
- Session 3 Review: **POS Tagging**
- Session 4 Review: **Hidden Markov Models**
- Session 5 Review: **Parsing**
- Session 6 Review: **Statistical Constituency Parsing**
- Session 7 Review: **Dependency Parsing**

# Module 1: Introduction

---

## Application areas

- Text-to-Speech & Speech recognition
- Healthcare
- Natural Language Dialogue Interfaces to Databases
- Information Retrieval
- Information Extraction
- Document Classification
- Document Image Analysis
- Automatic Summarization
- Text Proofreading – Spelling & Grammar
- Machine Translation
- Story understanding systems
- Plagiarism detection
- Can u think of anything else ??

# Levels of language understanding

---

**Morphological Knowledge** : Concerns how words are constructed from basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (Ex: “*friendly*” is derived from the meaning of noun “*friend*” and suffix “-ly”, which transforms noun into adjective)

**Lexical Knowledge** : Concerns with listing of words and categorizing them Ex: friendful or beautyship is incorrect lexically. But friendship and beautiful is correct

**Syntactic Knowledge** : Concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subpart of other phrases Ex: “Large have green ideas nose” is lexically correct but syntactically incorrect.

# Levels of language understanding

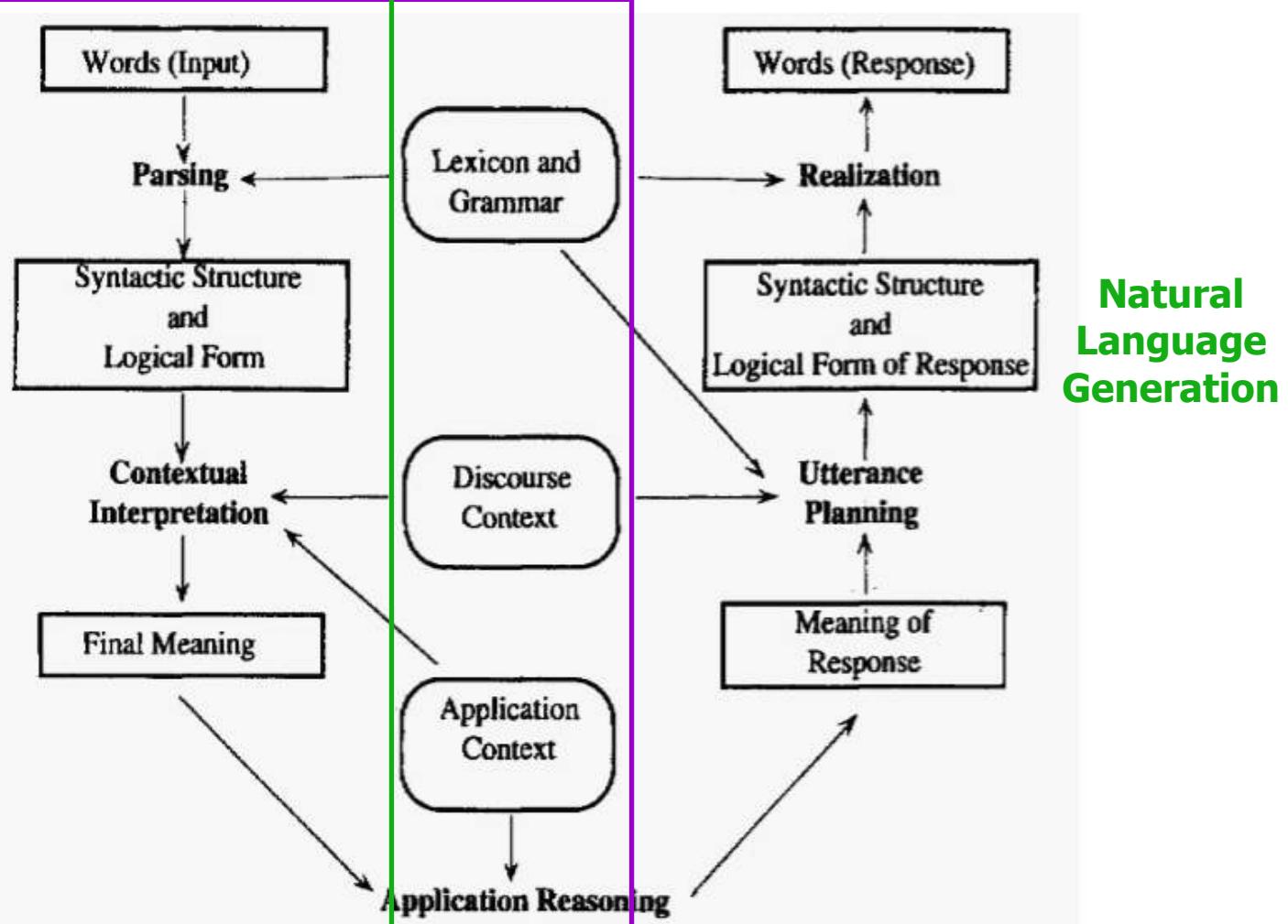
**Semantic Knowledge :**Concerns what words mean and how these meanings – combine in sentences to form sentence meanings. This is the study of context-independent meaning. Ex: “Green ideas have large noses” is syntactically correct but semantically incorrect.

**Pragmatic Knowledge :**Concerns how sentences are used in different situations how use affects the interpretation of sentence “She cuts banana with a pen” is semantically correct but pragmatically incorrect as it has no useful meaning.

**Discourse Knowledge :**Concerns how the immediately preceding sentences affect the interpretation of next sentence  
Ex: Chetana is a PhD student at IIT bombay. She is also teacher at Cummins College of Engineering for Women.

# The Organization of Natural Language Processing Systems

Natural  
Language  
Understanding



# Evaluating Language Understanding Systems

---



What metrics to use?

How to deal with complex outputs like translations?

Are the human judgments measuring something real?  
reliable?

Is the sample of texts sufficiently representative?

How reliable or certain are the results?

# Module 2: Language modelling



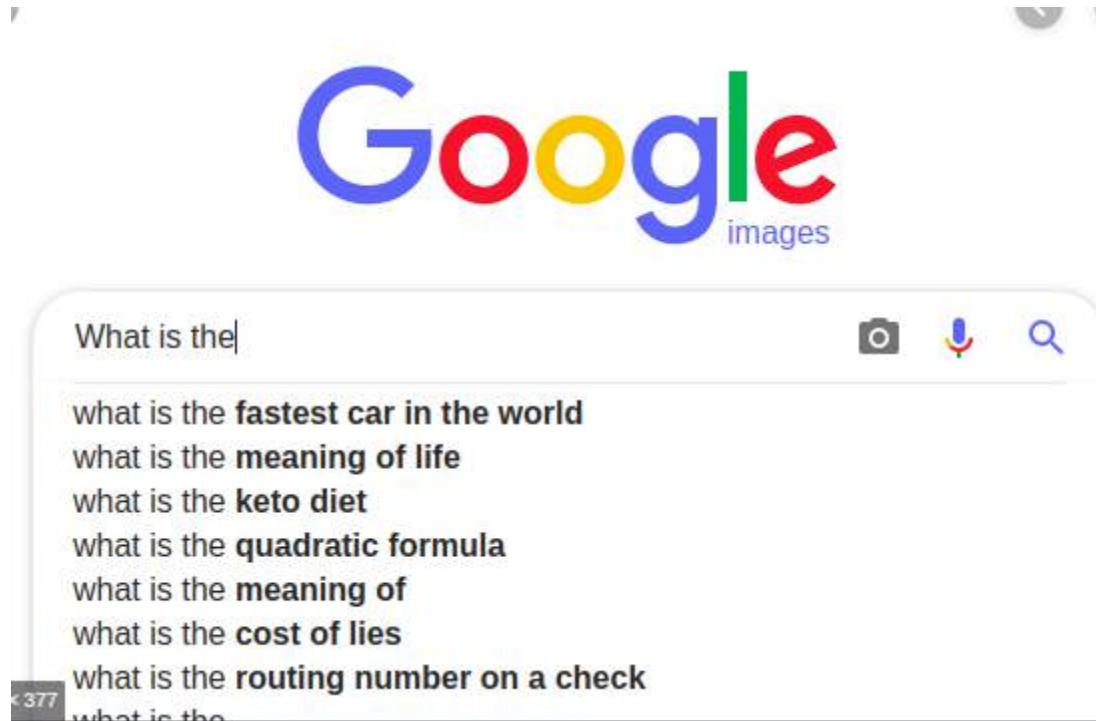
A model that computes either of these:

- Probability of a sentence (  $P(W)$  ) or
  - Probability of an upcoming word (  $P(w_n | w_1, w_2, \dots, w_{n-1})$  )
- is called a **language model**.

Simply we can say that

A language model learns to predict the probability of a sequence of words.

# Example



# How to build a language model



tion of conditional probabilities

$$p(B|A) = P(A,B)/P(A)$$

Rewriting:  $P(A,B) = P(A)P(B|A)$

More variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

The **Chain Rule** in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\dots P(x_n|x_{n-1}, x_{n-2}, \dots, x_1)$$

# The Chain Rule applied to compute joint probability of words in sentence

---

$$P(w_1 w_2 \dots \dots w_n) = \prod_{k=1}^n P(w_k | w_1^{k-1})$$

For ex:

$P(\text{"its water is so transparent"}) =$

$$P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water})$$

$$\times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so})$$

# Markov Assumption

Simplifying assumption:

*limit history of fixed number of words N1*



Andrei Markov

$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{that})$

or

$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{transparent that})$

# N-gram Language models

---

N gram is a sequence of tokens(words)

Unigram language model(N=1 )

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Example:

**P(I want to eat Chinese food)  $\approx$  P(I)P(want)**

**P(to)P(eat)P(Chinese)P(food)**

# Bigram model

---

N=2

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

Example:

$P(\text{I want to eat Chinese food}) \approx P(\text{I} | \langle \text{start} \rangle)$

$P(\text{want} | \text{I}) P(\text{to} | \text{want}) P(\text{eat} | \text{to}) P(\text{Chinese} | \text{eat})$

$P(\text{food} | \text{Chinese})$

$P(\langle \text{end} \rangle | \text{food})$

# Estimating bigram probabilities

## The Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

# An example

< s > I am Sam < /s >

< s > Sam I am < /s >

< s > I do not like green eggs and ham < /s >

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P(I | < s >) = \frac{2}{3} = .67 \quad P(Sam | < s >) = \frac{1}{3} = .33 \quad P(am | I) = \frac{2}{3} = .67$$

$$P(< /s > | Sam) = \frac{1}{2} = 0.5 \quad P(Sam | am) = \frac{1}{2} = .5 \quad P(do | I) = \frac{1}{3} = .33$$

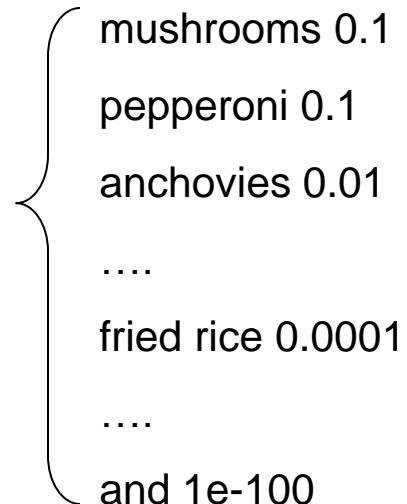
# Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and \_\_\_\_\_

The 33<sup>rd</sup> President of the US was \_\_\_\_\_

I saw a \_\_\_\_\_



- Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

# Perplexity

- The best language model is one that best predicts an unseen test set
  - Gives the highest  $P(\text{sentence})$
  - Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

For bigrams:

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

**Minimizing perplexity is the same as maximizing probability**

# Perplexity as branching factor

---

Let's suppose a sentence consisting of random digits

What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

# Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is  $400/1,000,000 = .0004$
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

# Raw bigram counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

# Raw bigram probabilities

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

$$P(i|i) = 5/2533 = 0.002$$

$$P(want|i) = 927 / 2533 = 0.33$$

$$P(to|i) = 0 / 2533 = 0$$

$$P(eat|i) = 9 / 2533 = 0.0036$$

$$P(i|want) = 2/927 = 0.0022$$

$$P(to|want) = 608 / 927 = 0.66$$

$$P(eat|to) = 686/2417 = 0.28$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

# The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
      - But occur in the test set

# Laplace Smoothing(Add 1 smoothing)

- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:
- Add-1 estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

# Berkeley Restaurant Corpus: Laplace smoothed bigram counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

V = Total number of unique words in corpus = 1446

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

# Laplace-smoothed bigrams

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

V = Total number of unique words in corpus = 1446

$$P(\text{want}|i) = 828 / 2533 + 1446 = 828 / 3979 = 0.21$$

$$P(\text{to}|i) = 1 / 3979 = 0.00025$$

$$P(\text{eat}|i) = 10 / 3979 = 0.0025$$

$$P(\text{to|want}) = 609 / 927 + 1446 = 609 / 2373 = 0.26$$

$$P(\text{eat|to}) = 687 / 2417 + 1446 = 0.18$$

$$P^*(w_n | w_{n-1}) = \frac{C(w_{n-1} w_n) + 1}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

# Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

$$c(i, \text{want}) = (828 * 2533) / (2533 + 1446) = 527$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

# Add-1 estimation is a blunt instrument

---

So add-1 isn't used for N-grams:

- We'll see better methods

But add-1 is used to smooth other NLP models

- For text classification
- In domains where the number of zeros isn't so huge.

# Backoff and Interpolation

---

Sometimes it helps to use **less** context

- Condition on less context for contexts you haven't learned much about

**Backoff:**

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

**Interpolation:**

- mix unigram, bigram, trigram

Interpolation works better

# Linear Interpolation

---

Simple interpolation

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2 P(w_n | w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}$$

Lambdas conditional on context:  $\sum_i \lambda_i = 1$

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2(w_{n-2}^{n-1}) P(w_n | w_{n-1}) \\ &\quad + \lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

---

# How to set the lambdas?

---

Use a **held-out** corpus

Training Data

Held-Out  
Data

Test  
Data

Choose  $\lambda$ s to maximize the probability of held-out data:

- Fix the N-gram probabilities (on the training data)
- Then search for  $\lambda$ s that give largest probability to held-out set:

$$\log P(w_1 \dots w_n \mid M(/_1 \dots /_k)) = \sum_i \log P_{M(/_1 \dots /_k)}(w_i \mid w_{i-1})$$

# Smoothing for Web-scale N-grams

---

“Stupid backoff” (Brants *et al.* 2007)

No discounting, just use relative frequencies

$$S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ 0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

# Session 3: POS Tagging

---

The process of assigning a part-of-speech or lexical class marker to each word in a collection.

<u>WORD</u>	<u>tag</u>
<b>the</b>	<b>DET</b>
<b>koala</b>	<b>N</b>
<b>put</b>	<b>V</b>
<b>the</b>	<b>DET</b>
<b>keys</b>	<b>N</b>
<b>on</b>	<b>P</b>
<b>the</b>	<b>DET</b>
<b>table</b>	<b>N</b>

# Open and Closed Classes

- Closed class: a small fixed membership
  - Prepositions: of, in, by, ...
  - Auxiliaries: may, can, will had, been, ...
  - Pronouns: I, you, she, mine, his, them, ...
  - Usually **function words** (short common words which play a role in grammar)
- Open class: new ones can be created all the time
  - English has 4: Nouns, Verbs, Adjectives, Adverbs
  - Many languages have these 4, but not all!

# Penn TreeBank POS Tagset

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	+,%,&
CD	cardinal number	<i>one, two, three</i>	TO	“to”	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential ‘there’	<i>there</i>	VB	verb, base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb, past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb, gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VBN	verb, past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb, non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb, 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WP\$	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, singular	<i>IBM</i>	\$	dollar sign	\$
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	#
PDT	predeterminer	<i>all, both</i>	“	left quote	‘ or “
POS	possessive ending	<i>'s</i>	”	right quote	’ or ”
PRP	personal pronoun	<i>I, you, he</i>	(	left parenthesis	[, (, {, <
PRP\$	possessive pronoun	<i>your, one's</i>	)	right parenthesis	], ), }, >
RB	adverb	<i>quickly, never</i>	,	comma	,
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	. ! ?
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	: ; ... --
RP	particle	<i>up, off</i>			

# POS Tagging as Sequence Classification

---

We are given a sentence (an “observation” or “sequence of observations”)

- *Secretariat is expected to race tomorrow*

What is the best sequence of tags that corresponds to this sequence of observations?

Probabilistic view

- Consider all possible sequences of tags
- Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of  $n$  words  $w_1 \dots w_n$ .

# Hidden Markov Models

---

It is a **sequence model**.

Assigns a label or class to each unit in a sequence, thus mapping a **sequence of observations** to a **sequence of labels**.

Probabilistic sequence model: given a sequence of units (e.g. words, letters, morphemes, sentences), compute a probability distribution over possible sequences of labels and choose the best label sequence.

This is a kind of *generative* model.

# Hidden Markov Model (HMM)

---

Oftentimes we want to know what produced the sequence

– the **hidden sequence** for the **observed sequence**.

For example,

- Inferring the **words** (hidden) from **acoustic signal** (observed) in speech recognition
- Assigning **part-of-speech tags** (hidden) to a **sentence** (sequence of words) – **POS tagging**.
- Assigning named **entity categories** (hidden) to a **sentence** (sequence of words) – Named Entity Recognition.

# Markov Chain: “First-order observable Markov Model”

- A set of states
  - $Q = q_1, q_2 \dots q_N$ ; the state at time  $t$  is  $q_t$
- Transition probabilities:
  - a set of probabilities  $A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$ .
  - Each  $a_{ij}$  represents the probability of transitioning from state  $i$  to state  $j$
  - The set of these is the transition probability matrix  $A$
  - Special **initial** probability vector  $\pi$
- Current state only depends on previous state

$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1})$$

# Markov Chain for Weather

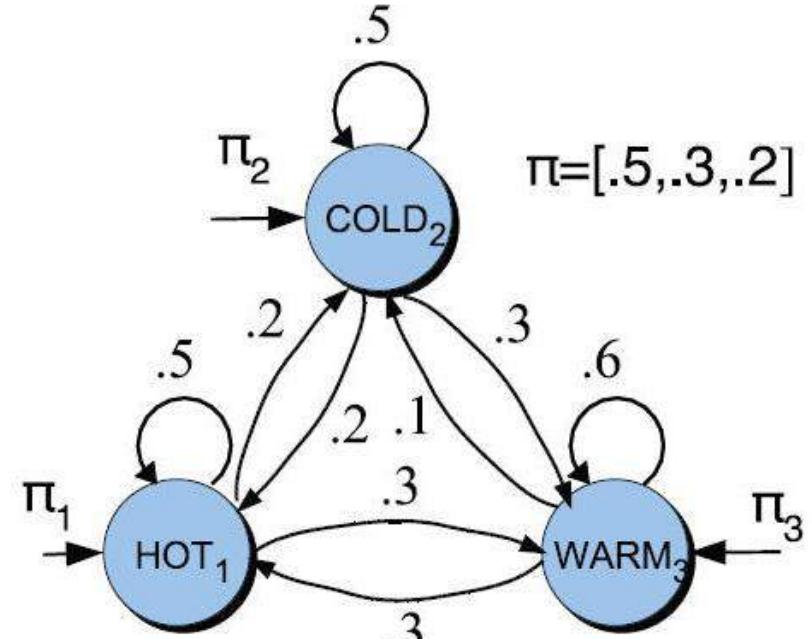
What is the probability of 4 consecutive warm days?

Sequence is  
warm-warm-warm-warm

And state sequence is  
3-3-3-3

$$P(3,3,3,3) =$$

- $\pi_3 a_{33} a_{33} a_{33} a_{33} = 0.2 \times (0.6)^3 = 0.0432$



# HMM to predict the tags

---

Two types of information are useful

- Emission Probability:
  - Relations between **words** and **tags**
- Transition Probability:
  - Relations between tags and tags
  - DT NN, DT JJ NN...

# Hidden Markov Models (POS Tagging)

States  $T = t_1, t_2 \dots t_N$ ;

Observations  $W = w_1, w_2 \dots w_N$ ;

- Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots, v_V\}$

Transition probabilities

- Transition probability matrix  $A = \{a_{ij}\}$   
$$a_{ij} = P(t_i = j \mid t_{i-1} = i) \quad 1 \leq i, j \leq N$$

Observation likelihoods

- Output probability matrix  $B = \{b_i(k)\}$   
$$b_i(k) = P(w_i = v_k \mid t_i = i)$$

Special initial probability vector  $\pi$

$$\pi_i = P(t_1 = i) \quad 1 \leq i \leq N$$

# Statistical POS Tagging

We want, out of all sequences of n tags  $t_1 \dots t_n$ , the single tag sequence such that

$P(t_1 \dots t_n | w_1 \dots w_n)$  is highest.

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

Hat ^ means “our estimate of the best one”

$\operatorname{Argmax}_x f(x)$  means “the x such that  $f(x)$  is maximized”

# Two Kinds of Probabilities

---

## 1. State transition probabilities -- $p(t_i|t_{i-1})$

- State-to-state transition probabilities

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

## 2. Observation/Emission probabilities -- $p(w_i|t_i)$

- Probabilities of observing various values at a given state

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

# Two Kinds of Probabilities

---

## 1. Tag transition probabilities -- $p(t_i|t_{i-1})$

- Determiners likely to precede adjs and nouns
  - That/DT flight/NN
  - The/DT yellow/JJ hat/NN
  - So we expect  $P(NN|DT)$  and  $P(JJ|DT)$  to be high
- Compute  $P(NN|DT)$  by counting in a labeled corpus:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

# Two Kinds of Probabilities

---

## 2. Word likelihood/emission probabilities $p(w_i|t_i)$

- VBZ (3sg Pres Verb) likely to be “is”
- Compute  $P(is|VBZ)$  by counting in a labeled corpus:

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

# Statistical POS tagging

- What is the most likely sequence of tags for the given sequence of words  $w$

$$\begin{aligned}\operatorname{argmax}_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) &= \operatorname{argmax}_{\mathbf{t}} \frac{P(\mathbf{t}, \mathbf{w})}{P(\mathbf{w})} \\ &= \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t}, \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{t}} P(\mathbf{t})P(\mathbf{w}|\mathbf{t})\end{aligned}$$

$$\begin{aligned}P(\text{ DT JJ NN } | \text{ a smart dog}) \\ &= P(\text{DD JJ NN a smart dog}) / P(\text{ a smart dog}) \\ &= P(\text{DD JJ NN}) P(\text{a smart dog} | \text{ DD JJ NN })\end{aligned}$$

# Transition Probability

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
  - $P(\mathbf{t}) = P(t_1, t_2, \dots, t_n)$   
 $= P(t_1)P(t_2 | t_1)P(t_3 | t_2, t_1) \dots P(t_n | t_1 \dots t_{n-1})$   
 $\sim P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$   
 $= \prod_{i=1}^n P(t_i | t_{i-1})$
- Markov assumption
- Bigram model over POS tags!  
(similarly, we can define a n-gram model over POS tags, usually we called high-order HMM)

# Emission Probability

- Joint probability  $P(t, w) = P(t)P(w|t)$
  - Assume words only depend on their POS-tag
  - $P(w|t) \sim P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$
- Independent assumption
- $$= \prod_{i=1}^n P(w_i | t_i)$$

i.e.,  $P(\text{a smart dog} | \text{DD JJ NN})$

$$= P(\text{a} | \text{DD}) P(\text{smart} | \text{JJ}) P(\text{dog} | \text{NN})$$

# Put them together

- Joint probability  $P(\mathbf{t}, \mathbf{w}) = P(\mathbf{t})P(\mathbf{w}|\mathbf{t})$
- $P(\mathbf{t}, \mathbf{w})$ 
$$= P(t_1)P(t_2 | t_1)P(t_3 | t_2) \dots P(t_n | t_{n-1})$$
$$P(w_1 | t_1)P(w_2 | t_2) \dots P(w_n | t_n)$$
$$= \prod_{i=1}^n P(w_i | t_i)P(t_i | t_{i-1})$$

e.g.,  $P(\text{a smart dog , DD JJ NN })$   
 $= P(\text{a} | \text{DD}) P(\text{smart} | \text{JJ }) P(\text{ dog} | \text{NN })$   
 $P(\text{DD} | \text{start}) P(\text{JJ} | \text{DD}) P(\text{NN} | \text{JJ })$

# Table representation

Transition Matrix  $A$

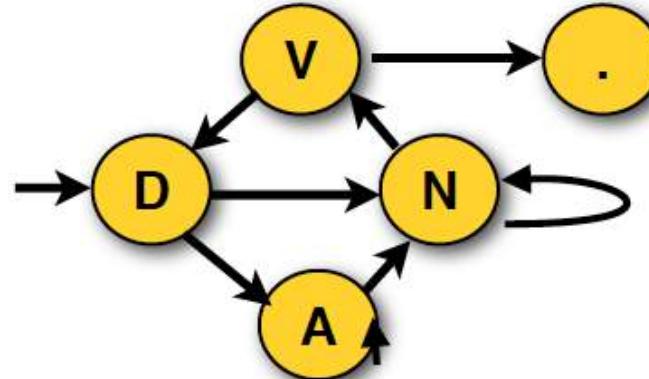
	D	N	V	A	.
D	0.8		0.2		
N	0.7	0.3			
V	0.6				0.4
A		0.8		0.2	
.					

Emission Matrix  $B$

	<i>the</i>	<i>man</i>	<i>ball</i>	<i>throws</i>	<i>sees</i>	<i>red</i>	<i>blue</i>	.
D	1.0							
N		0.7	0.3					
V				0.6	0.4			
A						0.8	0.2	
.								1

Initial state vector  $\pi$

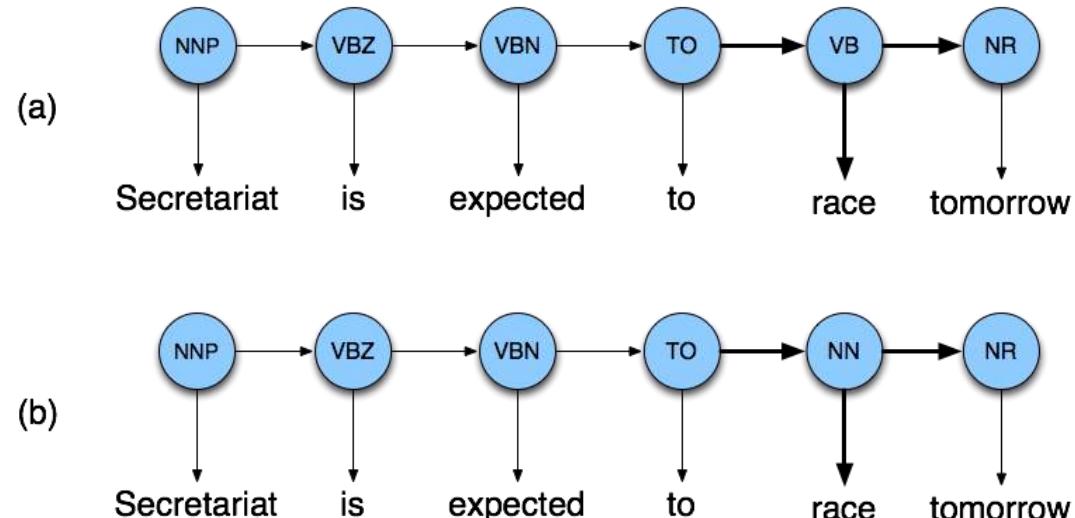
	D	N	V	A	.
$\pi$	1.0				



Let  $\lambda = \{A, B, \pi\}$  represents all parameters

# Example : NLTK POS tags

- $P(NN|TO) = .00047$
- $P(VB|TO) = .83$
- $P(race|NN) = .00057$
- $P(race|VB) = .00012$
- $P(NR|VB) = .0027$
- $P(NR|NN) = .0012$



NR-Adverbial Noun (tomorrow), TO-proposition

- $P(VB|TO)P(NR|VB)P(race|VB) = .00000027$
- $P(NN|TO)P(NR|NN)P(race|NN) = .00000000032$
- So we (correctly) choose the verb tag for “race”

<https://www.guru99.com/pos-tagging-chunking-nltk.html>

# Session 4: Hidden Markov Models

States  $Q = q_1, q_2 \dots q_N$ ;

Observations  $O = o_1, o_2 \dots o_N$ ;

- Each observation is a symbol from a vocabulary  $V = \{v_1, v_2, \dots v_V\}$

Transition probabilities

- Transition probability matrix  $A = \{a_{ij}\}$   

$$a_{ij} = P(q_t = j | q_{t-1} = i) \quad 1 \leq i, j \leq N$$

Observation likelihoods

- Output probability matrix  $B = \{b_i(k)\}$

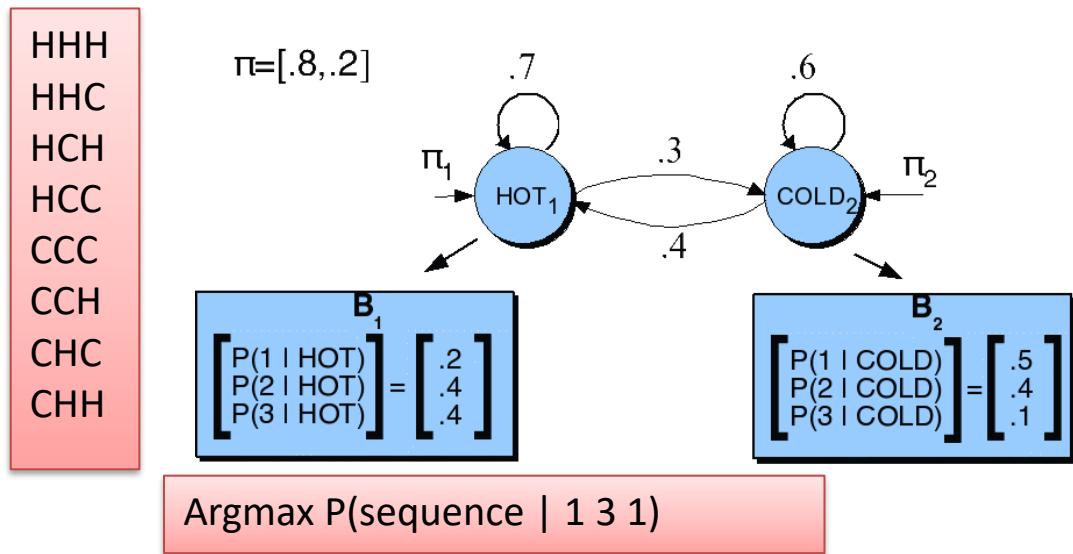
$$b_i(k) = P(X_t = o_k | q_t = i)$$

Special initial probability vector  $\pi$

$$\rho_i = P(q_1 = i) \quad 1 \leq i \leq N$$

# Ice Cream HMM

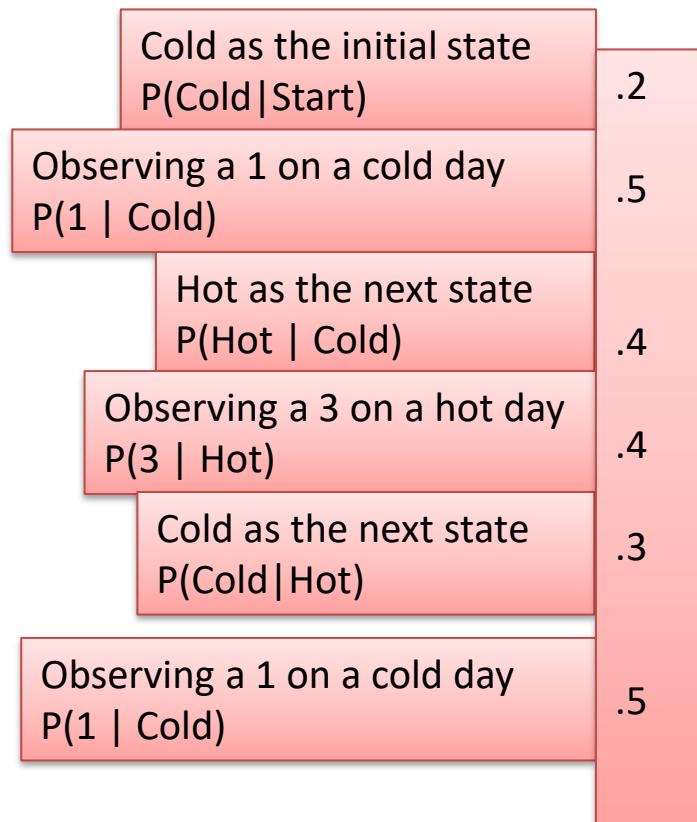
- Let's just do **131** as the sequence
  - How many underlying state (hot/cold) sequences are there?



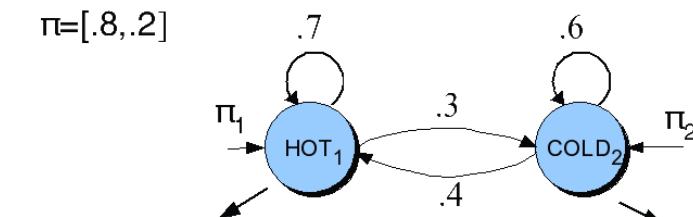
- How do you pick the right one?

# Ice Cream HMM

Let's just do 1 sequence: CHC



$$\pi = [.8, .2]$$



$$B_1 = \begin{bmatrix} P(1 | \text{HOT}) \\ P(2 | \text{HOT}) \\ P(3 | \text{HOT}) \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.4 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} P(1 | \text{COLD}) \\ P(2 | \text{COLD}) \\ P(3 | \text{COLD}) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.4 \\ 0.1 \end{bmatrix}$$

$$\begin{aligned} P(\text{CHC}) &= 0.2 * 0.5 * 0.4 * 0.4 * 0.3 * 0.5 \\ &= 0.0024 \end{aligned}$$

# Three Problems

---

Given this framework there are 3 problems that we can pose to an HMM

1. The likelihood of a observation sequence given a model and state sequence
2. Given an observation sequence and a model, what is the most likely state sequence?
3. Given an observation sequence, find the best model parameters for a partially specified model

# Problem 1: Observation Likelihood

The likelihood of a observation sequence given a model and state sequence

- Used in model development... How do I know if some change I made to the model is making things better?
- And in classification tasks
  - Word spotting in ASR, language identification, speaker identification, author identification, etc.
    - Train one HMM model per class
    - Given an observation, pass it to each model and compute  $P(\text{seq}|\text{model})$ .

# Problem 2: Decoding

---

Most probable state sequence given a model and an observation sequence

**Decoding:** Given as input an HMM  $\lambda = (A, B)$  and a sequence of observations  $O = o_1, o_2, \dots, o_T$ , find the most probable sequence of states  $Q = q_1 q_2 q_3 \dots q_T$ .

- Typically used in tagging problems, where the tags correspond to hidden states
  - As we'll see almost any problem can be cast as a sequence labeling problem

# Problem 3: Learning

---

Infer the best model parameters, given a partial model and an observation sequence...

– That is, fill in the A and B tables with the right numbers --

the numbers that make the observation sequence most likely

This is to learn the probabilities!

# Solutions

---

Problem 1: Forward (learn observation sequence)

Problem 2: Viterbi (learn state sequence)

Problem 3: Forward-Backward (learn probabilities)

- An instance of EM (Expectation Maximization)

# Viterbi Algorithm

- Create an array
  - Columns corresponding to observations
  - Rows corresponding to possible hidden states
- Recursively compute the probability of the most likely subsequence of states that accounts for the first  $t$  observations and ends in state  $s_j$ .

$$v_t(j) = \max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1, \dots, q_{t-1}, o_1, \dots, o_t, q_t = s_j | \lambda)$$

- Also record “backpointers” that subsequently allow backtracing the most probable state sequence.

# Viterbi Example 1: POS Tagging

Example: I want to race.

	<b>VB</b>	<b>TO</b>	<b>NN</b>	<b>PPSS</b>
<b>&lt;s&gt;</b>	.019	.0043	.041	.067
<b>VB</b>	.0038	.035	.047	.0070
<b>TO</b>	.83	0	.00047	0
<b>NN</b>	.0040	.016	.087	.0045
<b>PPSS</b>	.23	.00079	.0012	.00014

**Figure 5.15** Tag transition probabilities (the  $a$  array,  $p(t_i|t_{i-1})$ ) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus  $P(PPSS|VB)$  is .0070. The symbol  $<s>$  is the start-of-sentence symbol.

	<b>I</b>	<b>want</b>	<b>to</b>	<b>race</b>
<b>VB</b>	0	.0093	0	.00012
<b>TO</b>	0	0	.99	0
<b>NN</b>	0	.000054	0	.00057
<b>PPSS</b>	.37	0	0	0

**Figure 5.16** Observation likelihoods (the  $b$  array) computed from the 87-tag Brown corpus without smoothing.

# Viterbi Algorithm Example

---

Algorithm first creates N or four state columns.

- First column corresponds to the observation of the first word “I”,
- the second to the second word “want”,
- the third to the third word “to”, and
- the fourth to the fourth word “race”

Begin by setting the Viterbi value in each cell to the product of the transition probability (into it from the state state) and the observation probability (of the first word)

$\alpha_{word}$  $q_4$ 

$$v_1(4) = .041 \times 0 = 0$$

 $q_3$ 

$$v_1(3) = .0043 \times 0 = 0$$

 $q_2$ 

$$P(\text{VBG}|\text{VB}, \text{past}) \\ .041 \times 1.0 = .041$$

$$v_1(2) = .019 \times 0 = 0$$

 $q_1$ 

$$P(\text{VBG}|\text{VB}, \text{past}) \\ .0043 \times 1.0 = .0043$$

$$v_1(1) = .087 \times .37 = .025$$

 $q_0$  $O_1$  $O_2$  $O_3$  $O_4$  $t$

# Viterbi Algorithm

- For each state  $q_j$  at time  $t$ , the value Viterbi  $[s,t]$  is computed by taking the maximum over the extensions of all the paths that lead to the current cell, following the following equation:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

- $v_{t-1}(i)$  the **previous Viterbi path probability** from the previous time step  
 $a_{ij}$  the **transition probability** from previous state  $q_i$  to current state  $q_j$   
 $b_j(o_t)$  the **state observation likelihood** of the observation symbol  $o_t$  given the current state  $j$

# The Viterbi Algorithm

**function** VITERBI(*observations* of len  $T$ ,*state-graph* of len  $N$ ) **returns** *best-path*

create a path probability matrix  $viterbi[N+2,T]$

**for** each state  $s$  **from** 1 **to**  $N$  **do** ; initialization step

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[s,1] \leftarrow 0$

**for** each time step  $t$  **from** 2 **to**  $T$  **do** ; recursion step

**for** each state  $s$  **from** 1 **to**  $N$  **do**

$viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$

$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$

$viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step

$backpointer[q_F, T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step

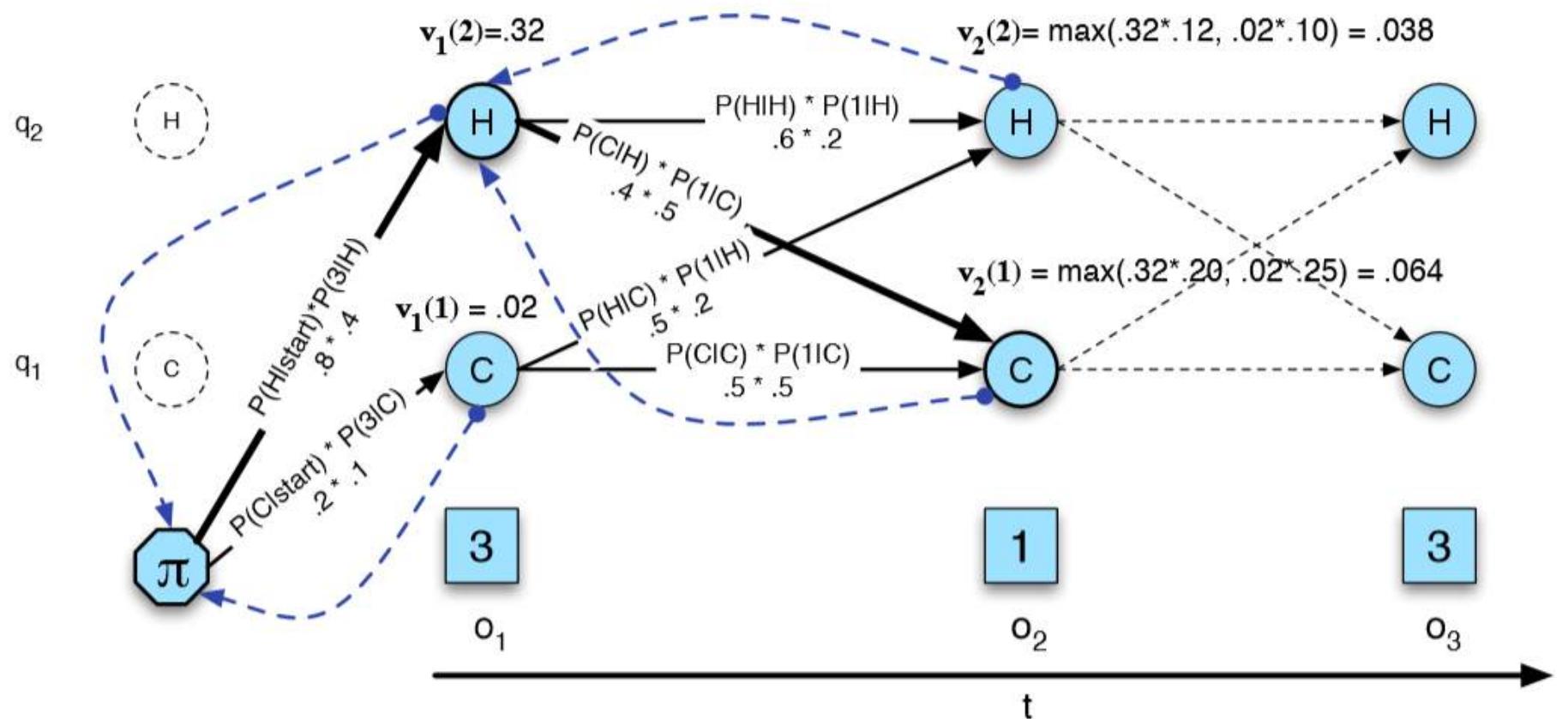
**return** the backtrace path by following backpointers to states back in time from  $backpointer[q_F, T]$

Speech and Language

Processing - Jurafsky and  
Martin

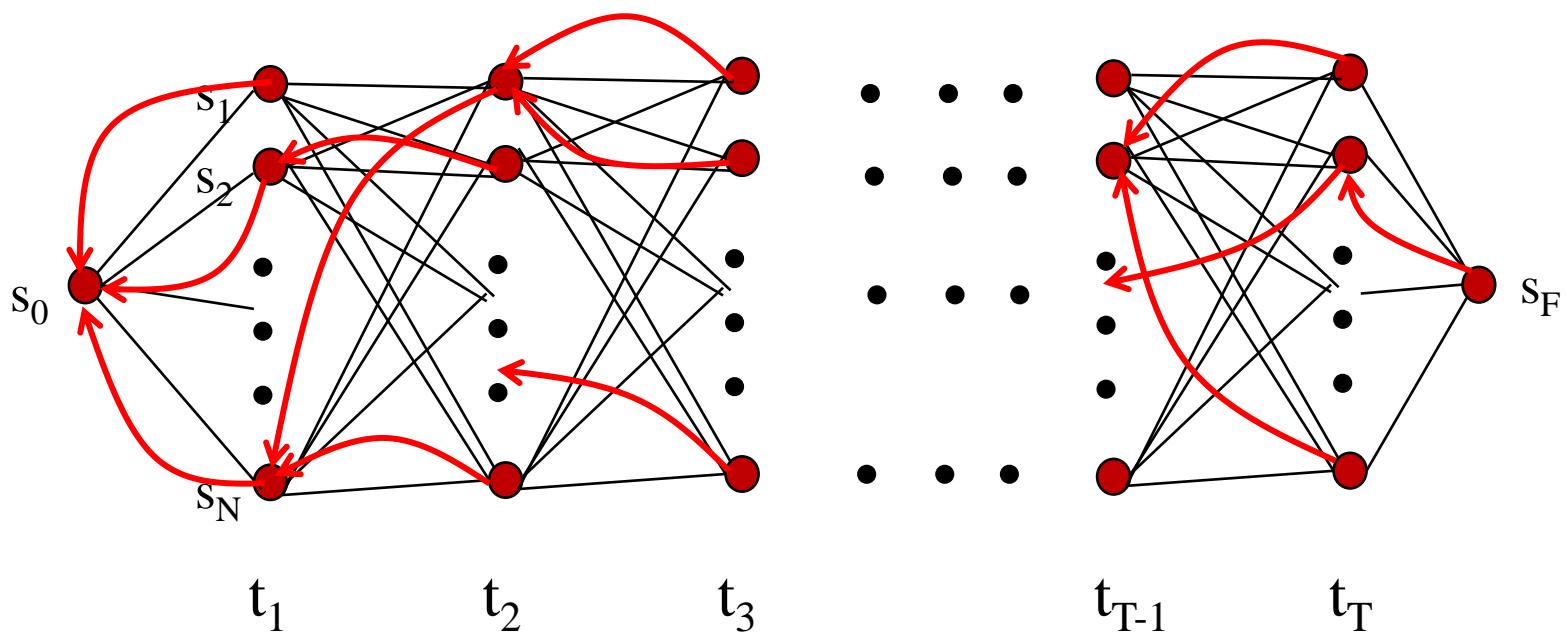


# Viterbi Example 2: Ice Cream

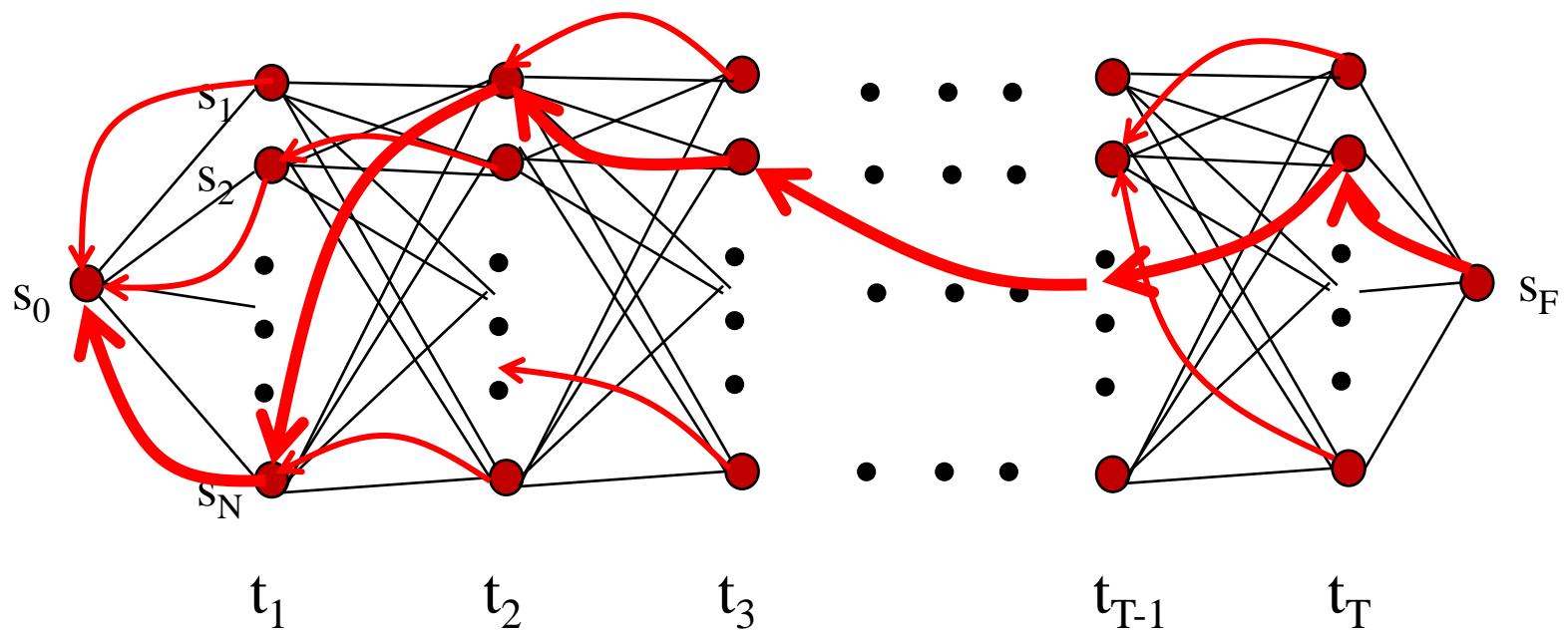


**Figure A.10** The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

# Viterbi Backpointers



# Viterbi Backtrace



Most likely Sequence:  $s_0 s_N s_1 s_2 \dots s_2 s_F$

# Forward

- Efficiently computes the probability of an observed sequence given a model
  - $P(\text{sequence}|\text{model})$
- Nearly identical to Viterbi; **replace the MAX with a SUM**

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \dots$$

# Forward

- Efficiently computes the probability of an observed sequence given a model
  - $P(\text{sequence}|\text{model})$
- Nearly identical to Viterbi; **replace the MAX with a SUM**

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \dots$$

# The Forward Algorithm

**function** FORWARD(*observations* of len  $T$ , *state-graph* of len  $N$ ) **returns** *forward-prob*

create a probability matrix  $forward[N+2,T]$

**for** each state  $s$  **from** 1 **to**  $N$  **do** ; initialization step

$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$

**for** each time step  $t$  **from** 2 **to**  $T$  **do** ; recursion step

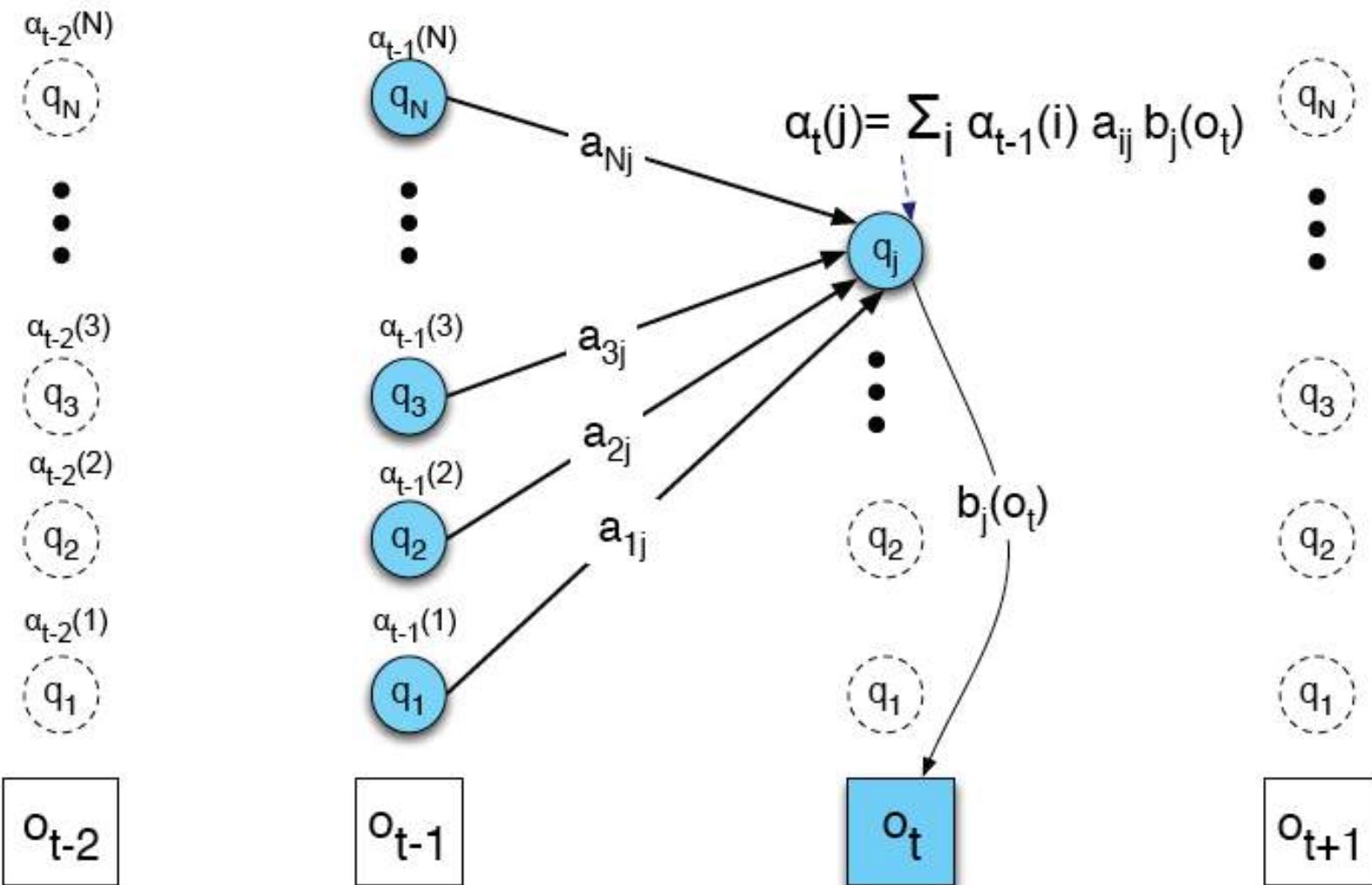
**for** each state  $s$  **from** 1 **to**  $N$  **do**

$forward[s,t] \leftarrow \sum_{s'=1}^N forward[s',t-1] * a_{s',s} * b_s(o_t)$

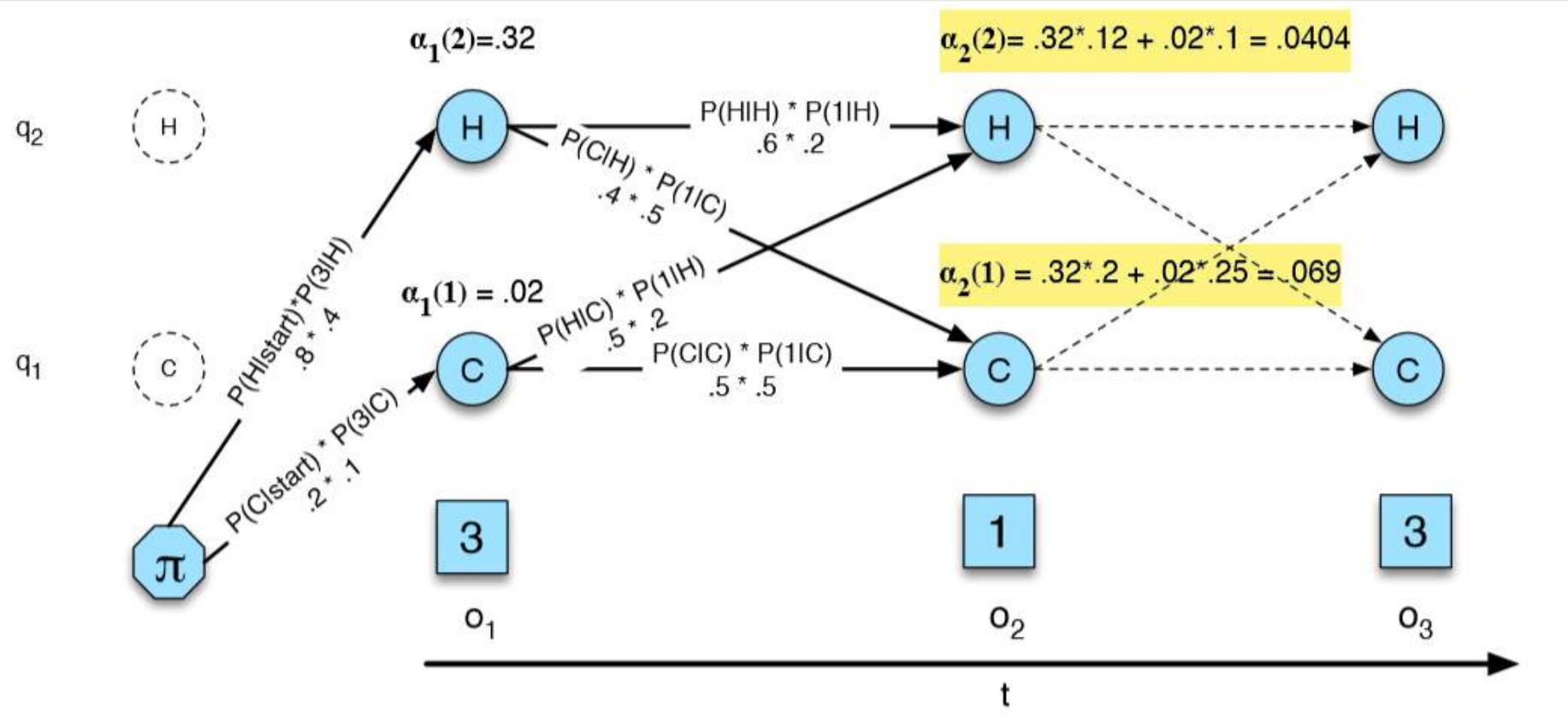
$forward[q_F, T] \leftarrow \sum_{s=1}^N forward[s, T] * a_{s,q_F}$  ; termination step

**return**  $forward[q_F, T]$

# Visualizing Forward



# Forward Algorithm: Ice Cream



**Figure A.5** The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of  $\alpha_t(j)$  for two states at two time steps. The computation in each cell follows Eq. A.12:  $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$ . The resulting probability expressed in each cell is Eq. A.11:  $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$ .

# Problem 3

- Infer the best model parameters, given a skeletal model and an observation sequence...
  - That is, fill in the A and B tables with the right numbers...
    - The numbers that make the observation sequence most likely
  - Useful for getting an HMM without having to hire annotators...

# Forward-Backward

**Learning:** Given an observation sequence  $O$  and the set of possible states in the HMM, learn the HMM parameters  $A$  and  $B$ .

- **Baum-Welch = Forward-Backward Algorithm**  
(Baum 1972)
- Is a special case of the EM or Expectation-Maximization algorithm
- The algorithm will let us learn the transition probabilities  $A = \{a_{ij}\}$  and the emission probabilities  $B = \{b_i(o_t)\}$  of the HMM

# Sketch of Baum-Welch (EM) Algorithm for Training HMMs

Assume an HMM with  $N$  states.

Randomly set its parameters  $\lambda = (A, B)$   
(making sure they represent legal distributions)

Until converge (i.e.  $\lambda$  no longer changes) do:

E Step: Use the forward/backward procedure to  
determine the probability of various possible  
state sequences for generating the training data

M Step: Use these probability estimates to  
re-estimate values for all of the parameters  $\lambda$

# Issues with HMM

---

- Unknown Words
  - We do not have the probabilities
    - Use smoothing
- Limited Context
  - Use trigrams/ 4 grams etc.
  - Sparsity

# Maximum Entropy Markov Model

---

- Identify heterogeneous set of features
  - tag given, the previous tag or the word given, the tag you can use, any other features which may contribute to the choice of part of speech tags.
- Turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word as a feature in the classification of the next word.
- When we apply logistic regression in this way, it's called maximum entropy Markov model or MEMM

# Maximum Entropy Markov Model

---

- Let the sequence of words be  $W = w_1^n$  and the sequence of tags  $T = t_1^n$
- In an HMM to compute the best tag sequence that maximizes  $P(T|W)$  we rely on Bayes' rule and the likelihood  $P(W|T)$ :

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T P(W|T)P(T) \\ &= \operatorname{argmax}_T \prod_i P(\text{word}_i|\text{tag}_i) \prod_i P(\text{tag}_i|\text{tag}_{i-1})\end{aligned}$$

# Maximum Entropy Markov Model

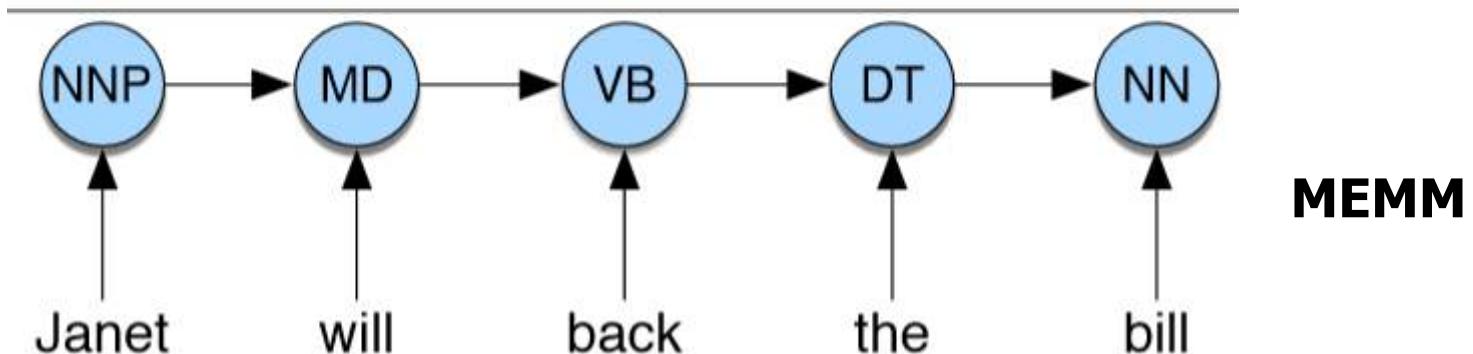
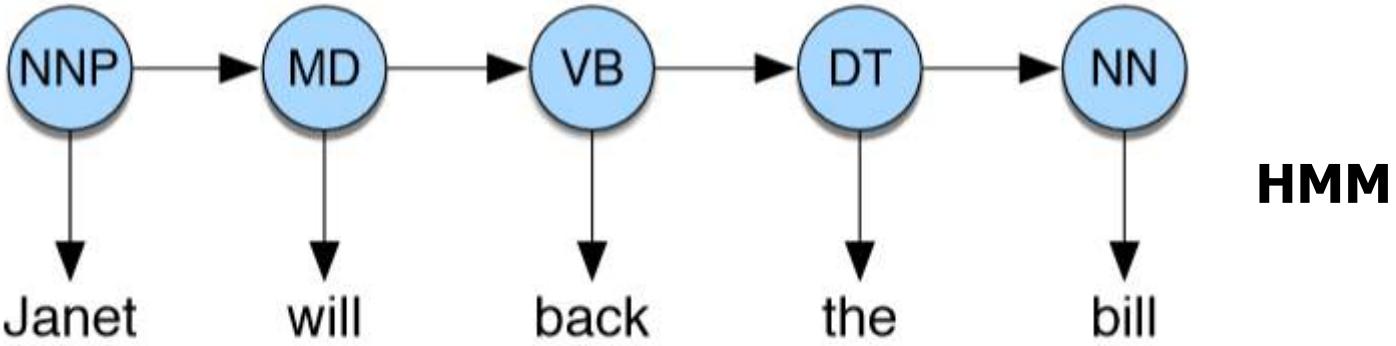
---

- In an MEMM, by contrast, we compute the posterior  $P(T|W)$  directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}$$

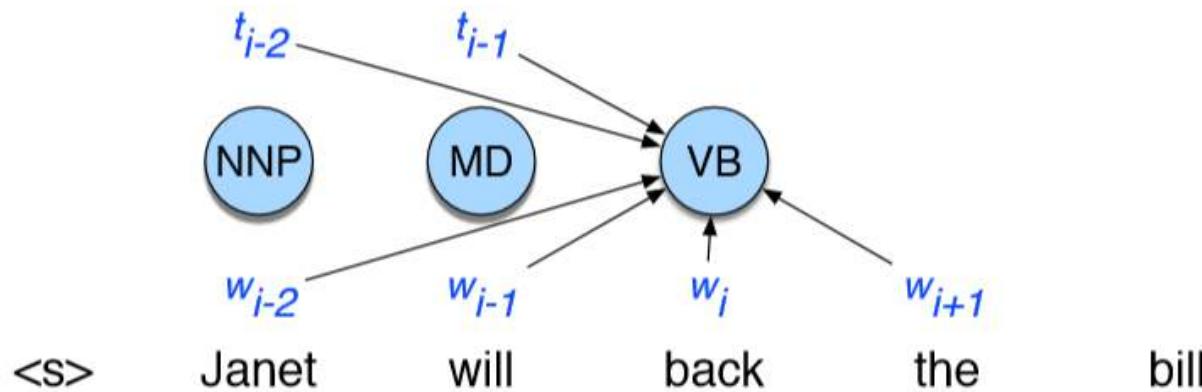
# Maximum Entropy Markov Model

- Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability  $P(t_i|w_i, t_{i-1})$  in a different way than an HMM
- HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).



# Maximum Entropy Markov Model

- Reason to use a discriminative sequence model is that it's easier to incorporate a lot of features



**Figure 8.13** An MEMM for part-of-speech tagging showing the ability to condition on more features.

# MEMM

---

- Janet/NNP will/MD back/VB the/DT bill/NN, when  $w_i$  is the word back, would generate the following features

$t_i = \text{VB}$  and  $w_{i-2} = \text{Janet}$

$t_i = \text{VB}$  and  $w_{i-1} = \text{will}$

$t_i = \text{VB}$  and  $w_i = \text{back}$

$t_i = \text{VB}$  and  $w_{i+1} = \text{the}$

$t_i = \text{VB}$  and  $w_{i+2} = \text{bill}$

$t_i = \text{VB}$  and  $t_{i-1} = \text{MD}$

$t_i = \text{VB}$  and  $t_{i-1} = \text{MD}$  and  $t_{i-2} = \text{NNP}$

$t_i = \text{VB}$  and  $w_i = \text{back}$  and  $w_{i+1} = \text{the}$

# Decoding and Training MEMMs

- The most likely sequence of tags is then computed by combining these features of the input word  $w_i$ , its neighbors within  $l$  words  $w_{i-l}^{i+l}$ , and the previous  $k$  tags  $t_{i-k}^{i-1}$  as follows (using  $\theta$  to refer to feature weights instead of  $w$  to avoid the confusion with  $w$  meaning words):

$$\begin{aligned}
 \hat{T} &= \operatorname{argmax}_T P(T|W) \\
 &= \operatorname{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
 &= \operatorname{argmax}_T \prod_i \frac{\exp \left( \sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}{\sum_{t' \in \text{tagset}} \exp \left( \sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}
 \end{aligned}$$

# How to decode to find this optimal tag sequence $\hat{T}$ ?

- Simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on.
- This is called a greedy decoding algorithm

```
function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T
    for i = 1 to length(W)
         $\hat{t}_i = \operatorname{argmax}_{t' \in T} P(t' | w_{i-l}^{i+l}, t_{i-k}^{i-1})$ 
```

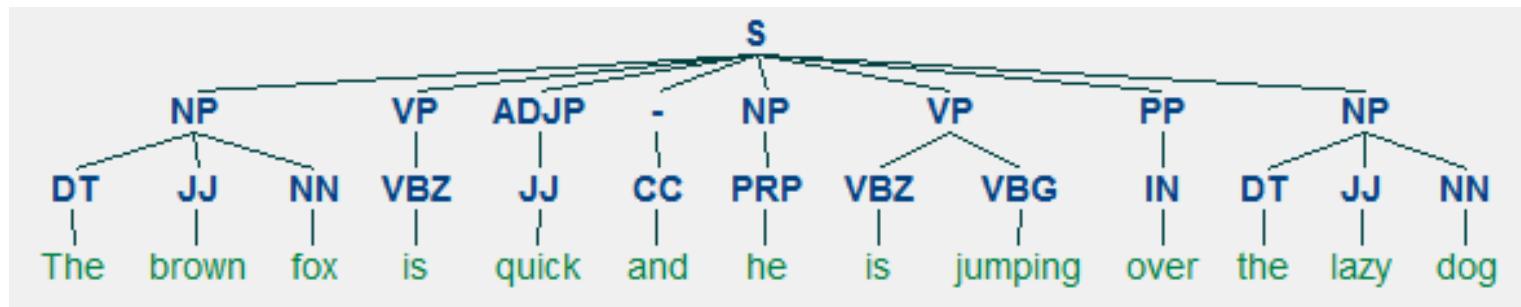
**Figure 8.14** In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

# Session 5: Parsing

Analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words.

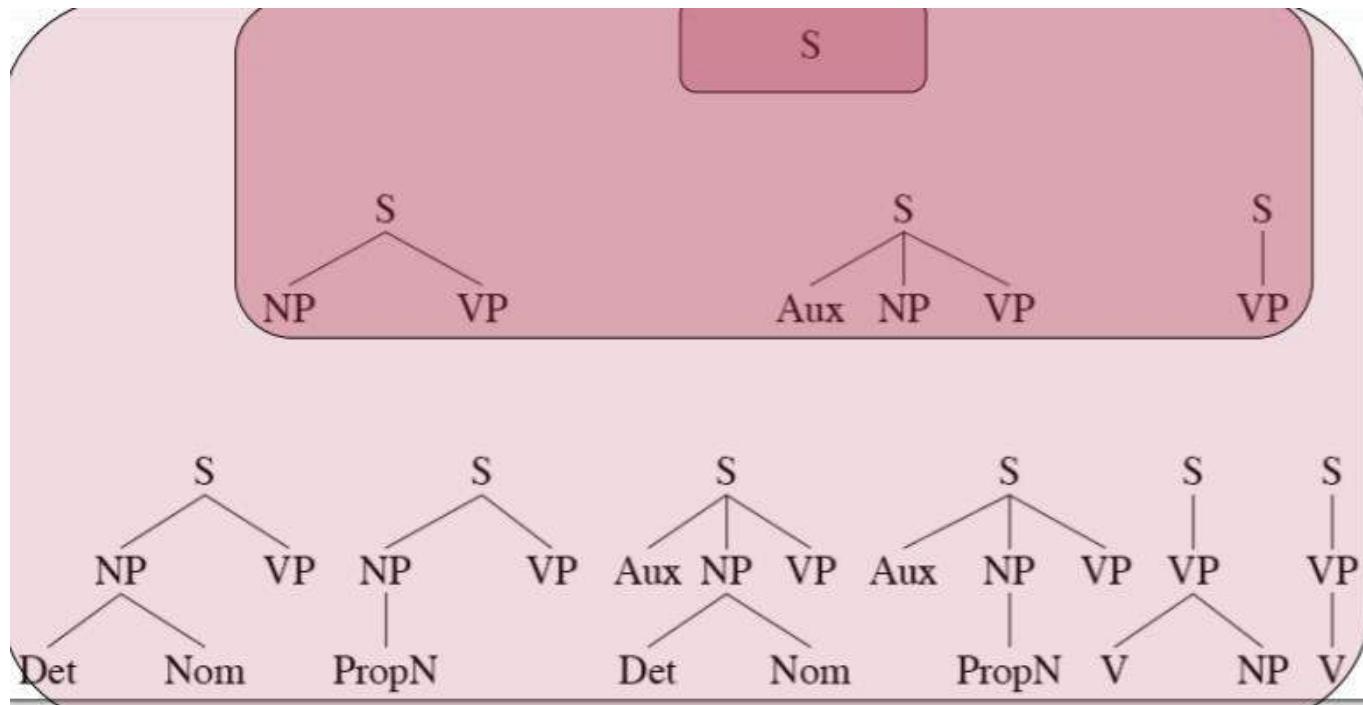
Analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases.

This includes POS tags as well as phrases from a sentence.



# Top down Parsing

- Parse tree is generated in the top to bottom fashion from root to leaves
- Search space from the start symbol sentence S on LHS



# Top down parse of : “<sub>1</sub>The <sub>2</sub> old <sub>3</sub> man <sub>4</sub> smiled. <sub>5</sub>”

Step	Current State	Backup States	Comment	
1.	((S) 1)			1. S → NP VP
2.	((NP VP) 1)		S rewritten to NP VP	2. NP → ART N
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states	3. NP → ART ADJ N
4.	((N VP) 2)	((ART ADJ N VP) 1)		
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains	4. VP → V
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)		
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)		5. VP → V NP
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen	
9.	((NP) 4)	((ART ADJ N VP) 1)		
10.	((ART N) 4)	((ART ADJ N VP) 1) ((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails	
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again	
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3	
13.	((ADJ N VP) 2)			
14.	((N VP) 3)			
15.	((VP) 4)			
16.	((V) 4)	((V NP) 4)		
17.	(( ) 5)		success!	

## Lexicon

The → ART  
 Old → N, ADJ  
 Man → N, V  
 Smiled -> V

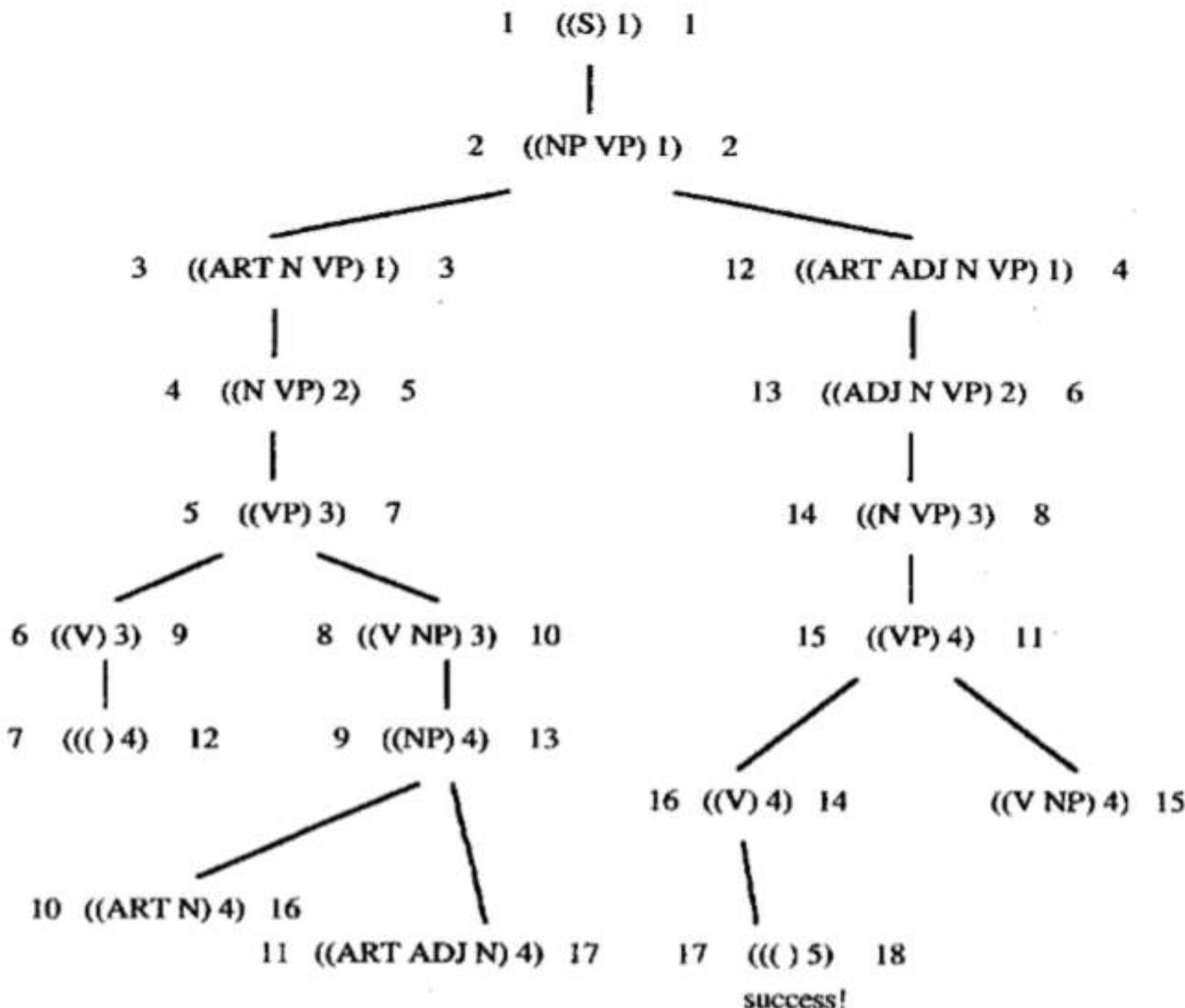


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

# Depth-first strategy vs Breadth-first strategy

---

- For a depth-first strategy, the possibilities list is a stack, yielding a last-in first-out (LIFO) strategy.
- In contrast, in a breadth-first strategy the possibilities list is manipulated as a queue, yielding a first-in first-out (FIFO) strategy
- With the depth-first strategy, one interpretation is considered and expanded until it fails; only then is the second one considered.
- With the breadth-first strategy, both interpretations are considered alternately, each being expanded one step at a time
- Many parsers built today use the depth-first strategy because it tends to minimize the number of backup states needed and thus uses less memory and requires less bookkeeping

# Bottom up parsing

---

The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules.

You could build a bottom-up parser simply by formulating this matching process as a search process.

The state would simply consist of a symbol list, starting with the words in the sentence.

Successor states could be generated by exploring all possible ways to

- Rewrite a word by its possible lexical categories
  - Replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol
-

# Chart Parsing

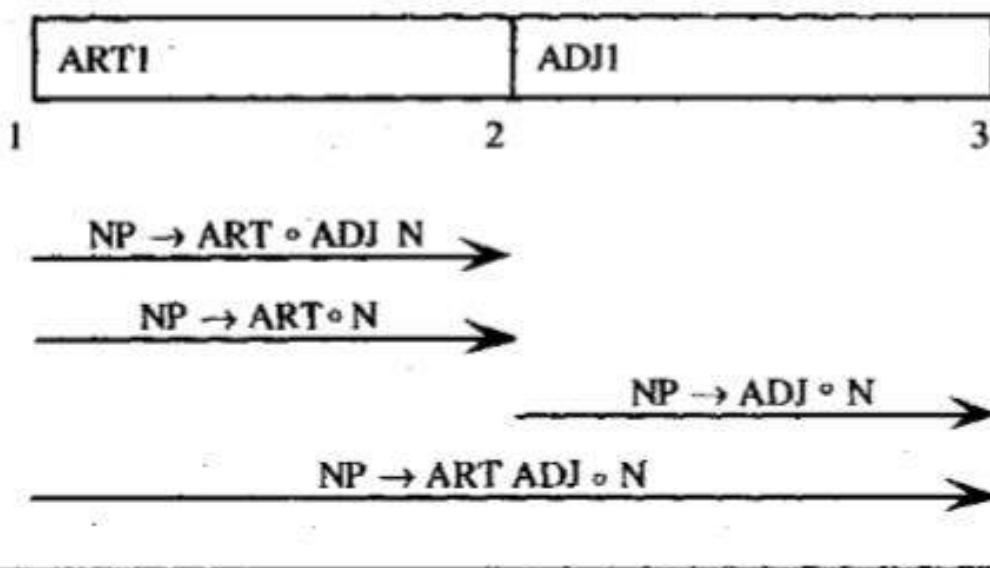
- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to “match”
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

# Chart Parsing

- Assume you are parsing a sentence that starts with an ART.
- With this ART as the key, rules 2 and 3 are matched because they start with ART.
- To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART.
- You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record
  - 2'. NP -> ART o ADJ N
  - 3'. NP -> ART o N
- If the next input key is an ADJ, then rule 4 may be started, and the modified rule 2 may be extended to give
  - 2''. NP -> ART ADJ o N

# Chart- active arcs

- The chart maintains the record of all the constituents derived from the sentence so far in the parse.
- Maintains the record of rules that have matched partially but are not complete. These are called the active arcs.



1.  $S \rightarrow NP VP$
2.  $NP \rightarrow ART N$
3.  $NP \rightarrow ART ADJ N$
  
4.  $VP \rightarrow V$
  
5.  $VP \rightarrow V NP$

Figure 3.9 The chart after seeing an ADJ in position 2

# Chart Parsing

Steps in the Process:

- Input is processed left-to-right, one word at a time
1. Find all POS of word (terminal-level)
  2. Initialize Agenda with all POS of the word
  3. Pick a Key from the Agenda
  4. Add all grammar rules that start with the Key as active arcs
  5. Extend any existing active arcs with the Key
  6. Add LHS constituents of newly completed rules to the Agenda
  7. Add the Key to the Chart
  8. If Agenda not empty - goto (3), else goto (1)

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

POS of Input Words:

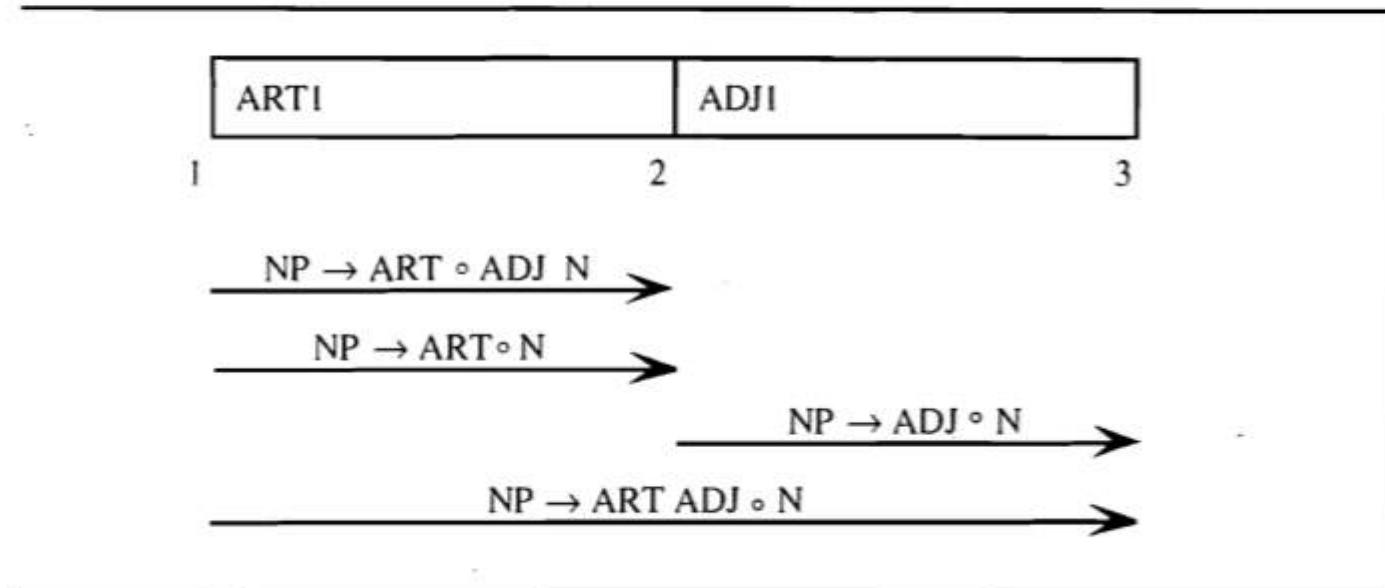
- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Chart parsing example

The input: “**x = The large can can hold the water**”

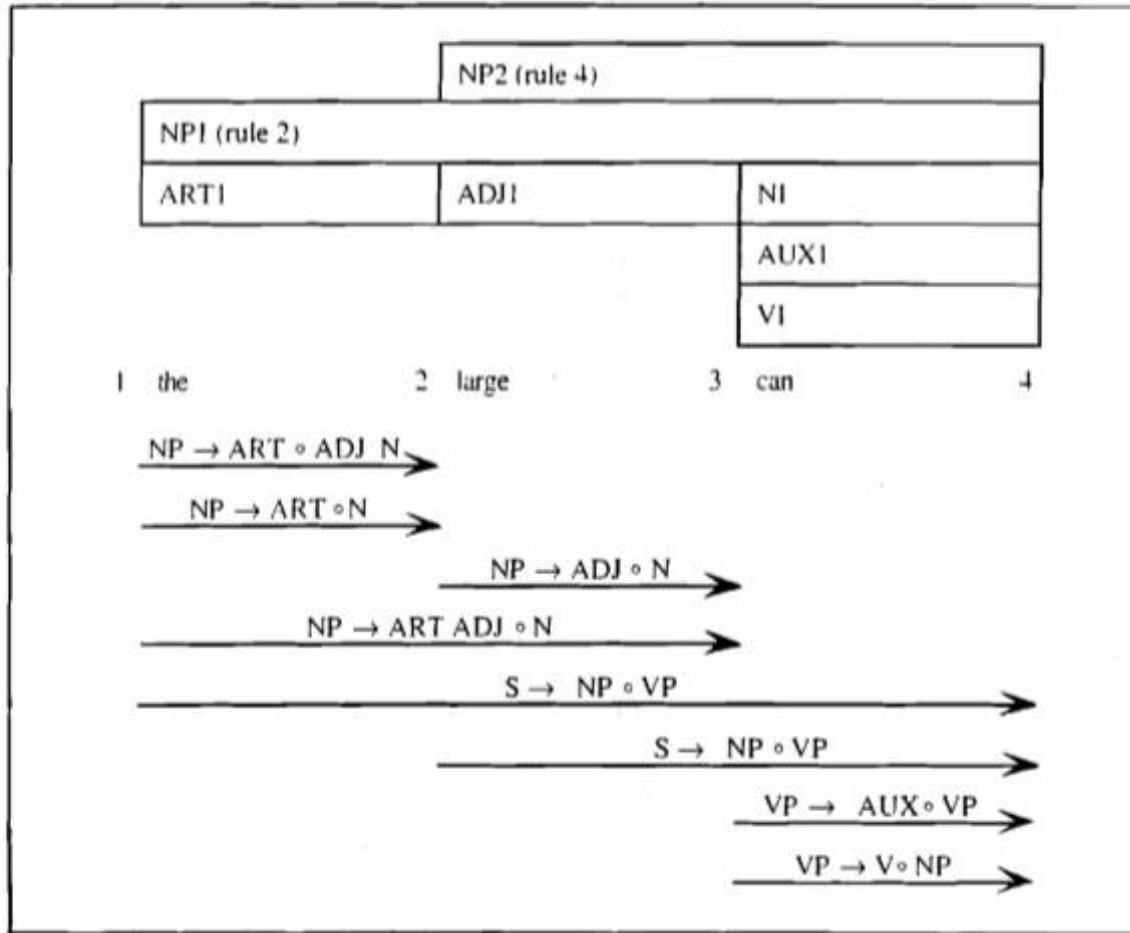
- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$



**Figure 3.9** The chart after seeing an ADJ in position 2

# Chart parsing example

The input: “**x = The large can can hold the water**”



- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Chart parsing example

The input: “ $x = \text{The large can can hold the water}$ ”

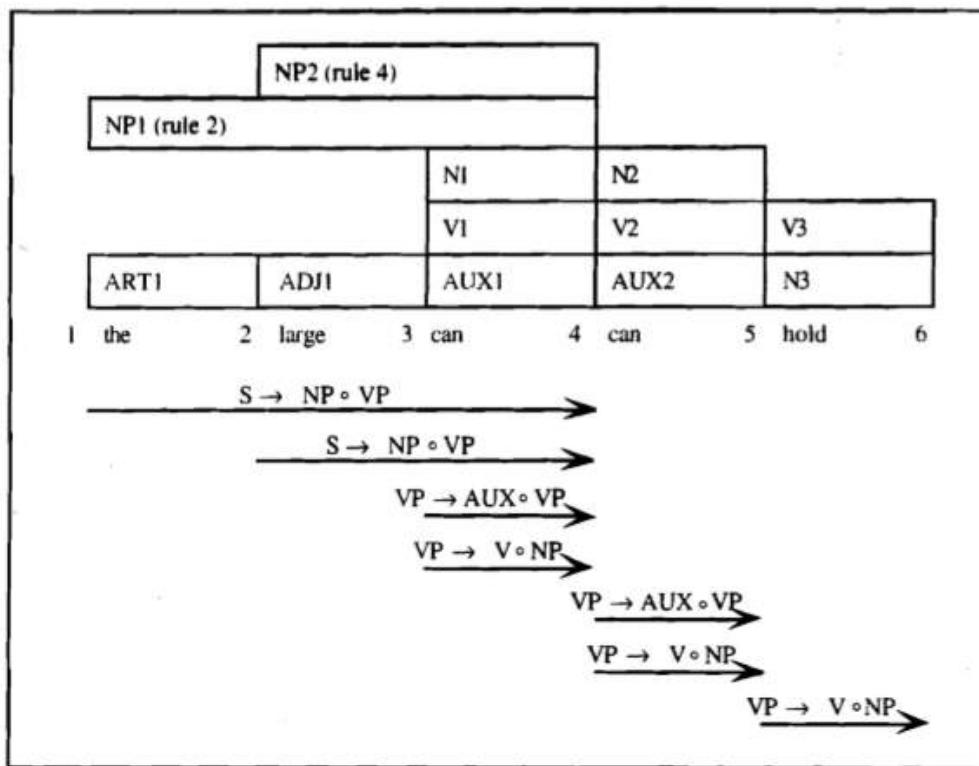
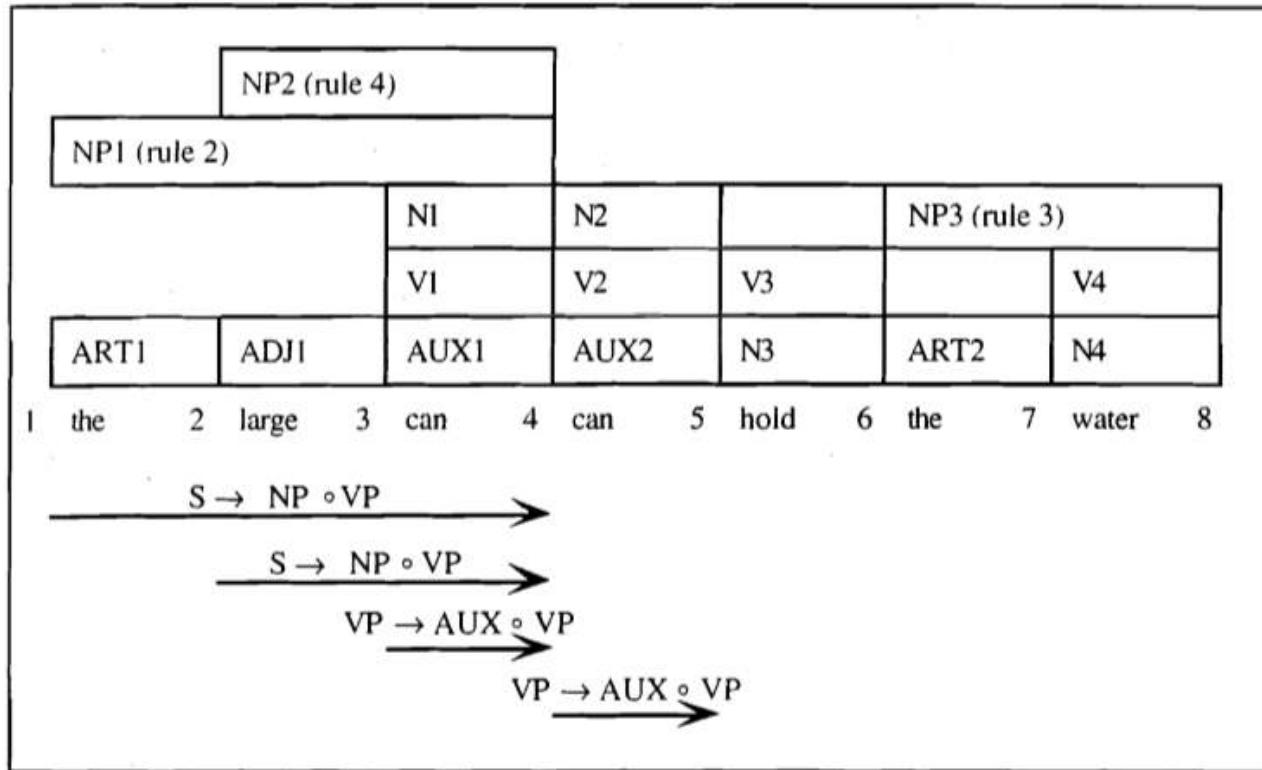


Figure 3.13 The chart after adding *hold*, omitting arcs generated for the first NP

- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

# Chart parsing example

The input: “**x = The large can can hold the water**”

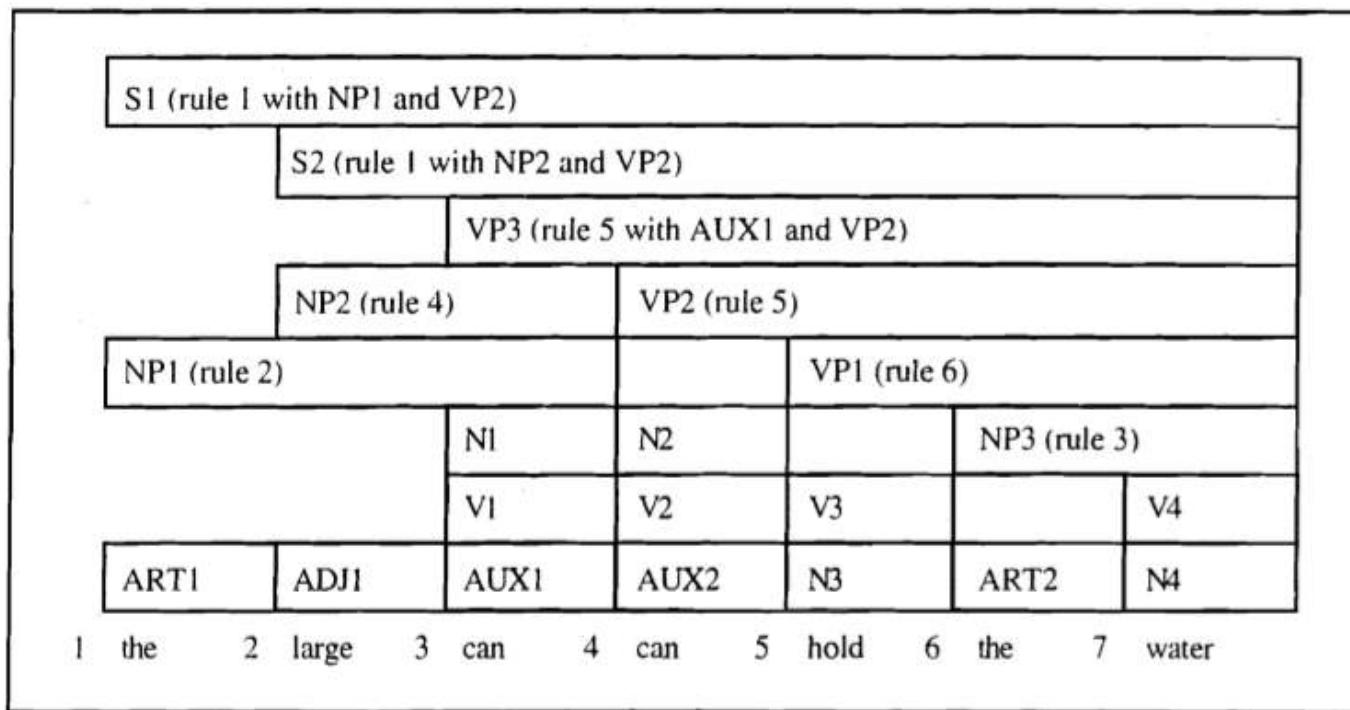


- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

**Figure 3.14** The chart after all the NPs are found, omitting all but the crucial active arcs

# Final Chart

The input: “***x = The large can can hold the water***”



- (1)  $S \rightarrow NP VP$
- (2)  $NP \rightarrow ART ADJ N$
- (3)  $NP \rightarrow ART N$
- (4)  $NP \rightarrow ADJ N$
- (5)  $VP \rightarrow AUX VP$
- (6)  $VP \rightarrow V NP$

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

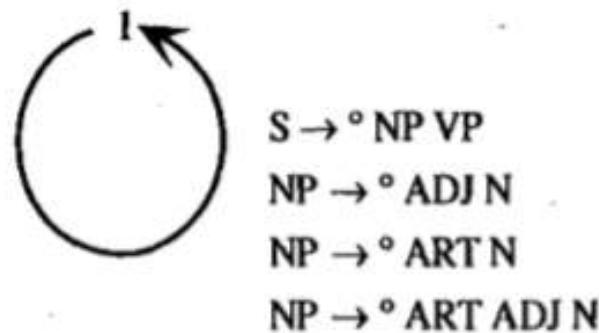
Figure 3.15 The final chart

# Top down chart parsing

---

- Top-down methods have the advantage of being highly predictive.
- A word might be ambiguous in isolation, but if some of those possible categories cannot be used in a legal sentence, then these categories may never even be considered.

# Top down chart parsing example



**Figure 3.23** The initial chart

# Top down chart parsing example

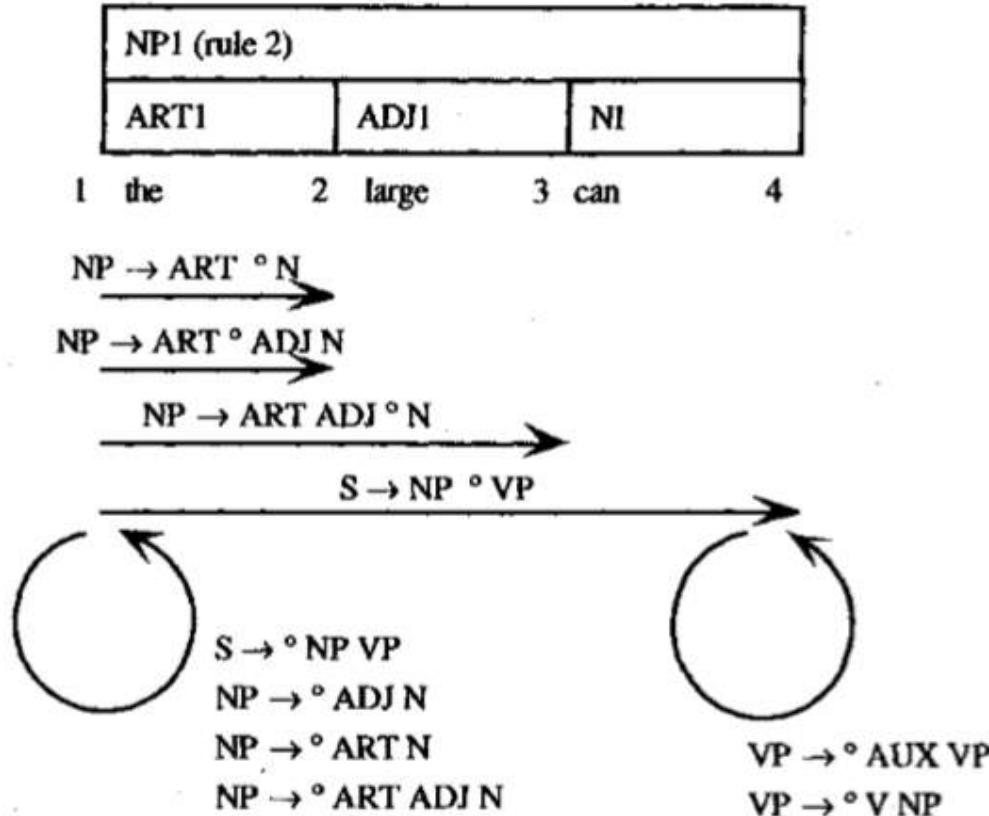


Figure 3.24 The chart after building the first NP

# Top down chart parsing example

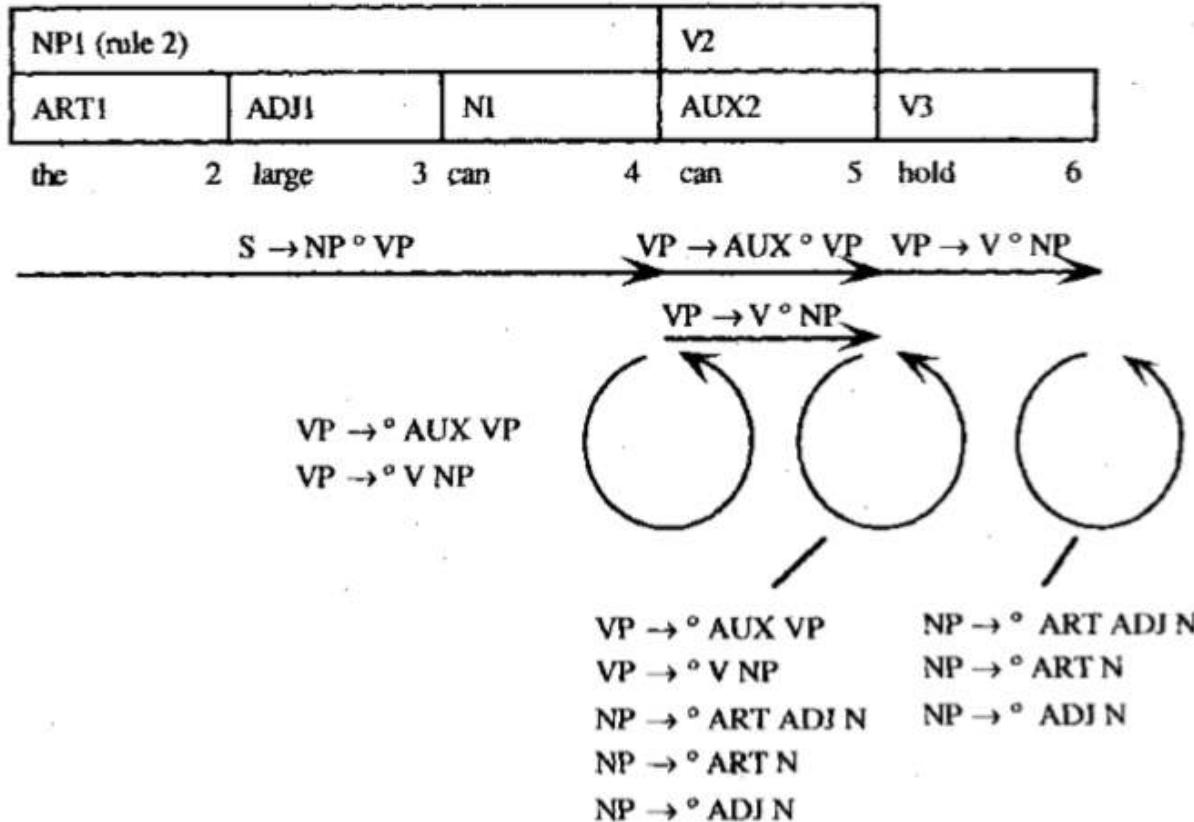


Figure 3.25 The chart after adding *hold*, omitting arcs generated for the first NP

# Top down chart parsing example

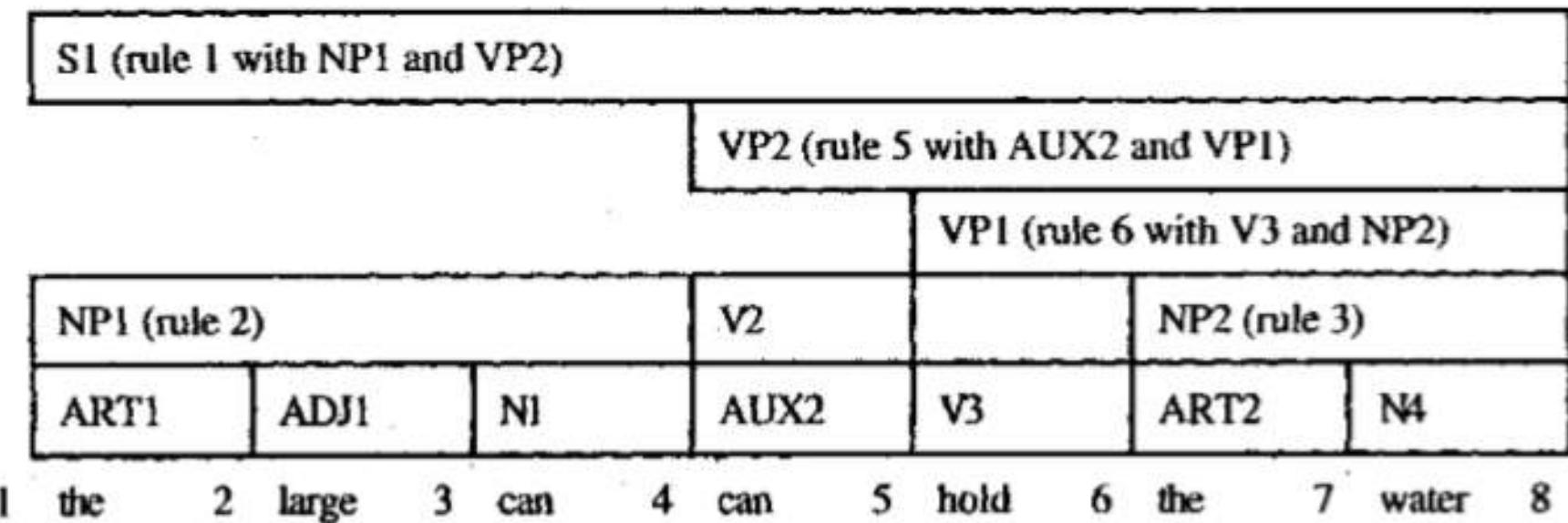


Figure 3.26 The final chart for the top-down filtering algorithm

# Chart Parsing

- When a chart parser begins parsing a text, it creates a new (empty) chart, spanning the text.
- It then incrementally adds new edges to the chart.
- A set of "chart rules" specifies the conditions under which new edges should be added to the chart.
- Once the chart reaches a stage where none of the chart rules adds any new edges, parsing is complete.

## Advantages of Chart Parsing

- No repeated computation of same sub problem
- Deals well with left-recursive grammars
- Deals well with ambiguity
- No backtracking necessary

# Session 6: Statistical Constituency Parsing

---



## CKY Parsing

Classic, bottom-up dynamic programming algorithm (Cocke-Kasami-Younger).

Requires input grammar based on Chomsky Normal Form

- A CNF grammar is a Context-Free Grammar in which:
  - Every rule LHS is a non-terminal
  - Every rule RHS consists of either a single terminal or two non-terminals.
  - Examples:
    - $A \rightarrow BC$
    - $NP \rightarrow N\ PP$
    - $A \rightarrow a$
    - Noun  $\rightarrow$  man
  - But not:
    - $NP \rightarrow \text{the}\ N$
    - $S \rightarrow VP$

# Chomsky Normal Form

---

Any CFG can be re-written in CNF, without any loss of expressiveness.

- That is, for any CFG, there is a corresponding CNF grammar which accepts exactly the same set of strings as the original CFG.

# Converting a CFG to CNF

To convert a CFG to CNF, we need to deal with three issues:

1. Rules that mix terminals and non-terminals on the RHS
  - E.g.  $NP \rightarrow \text{the Nominal}$
2. Rules with a single non-terminal on the RHS (called unit productions)
  - E.g.  $NP \rightarrow \text{Nominal}$
3. Rules which have more than two items on the RHS
  - E.g.  $NP \rightarrow \text{Det Noun PP}$

# Converting a CFG to CNF

---

## 1. Rules that mix terminals and non-terminals on the RHS

- E.g.  $NP \rightarrow \text{the Nominal}$
- Solution:
  - Introduce a dummy non-terminal to cover the original terminal
    - E.g.  $Det \rightarrow \text{the}$
  - Re-write the original rule:
    - $NP \rightarrow Det\ Nominal$
    - $Det \rightarrow \text{the}$

# Converting a CFG to CNF

## 2. Rules with a single non-terminal on the RHS (called unit productions)

- E.g.  $NP \rightarrow \text{Nominal}$
- Solution:
  - Find all rules that have the form  $\text{Nominal} \rightarrow \dots$ 
    - $\text{Nominal} \rightarrow \text{Noun PP}$
    - $\text{Nominal} \rightarrow \text{Det Noun}$
  - Re-write the above rule several times to eliminate the intermediate non-terminal:
    - $NP \rightarrow \text{Noun PP}$
    - $NP \rightarrow \text{Det Noun}$
- Note that this makes our grammar “flatter”

# Converting a CFG to CNF

## 3. Rules which have more than two items on the RHS

- E.g.  $NP \rightarrow Det\ Noun\ PP$

### Solution:

- Introduce new non-terminals to spread the sequence on the RHS over more than 1 rule.
  - $Nominal \rightarrow Noun\ PP$
  - $NP \rightarrow Det\ Nominal$

# CNF Grammar

---

If we parse a sentence with a CNF grammar, we know that:

- Every phrase-level non-terminal (above the part of speech level) will have exactly 2 daughters.
  - $\text{NP} \rightarrow \text{Det N}$
- Every part-of-speech level non-terminal will have exactly 1 daughter, and that daughter is a terminal:
  - $\text{N} \rightarrow \text{lady}$

# Recognising strings with CKY

---

Example input: The flight includes a meal.

The CKY algorithm proceeds by:

1. Splitting the input into words and indexing each position.  
(0) the (1) flight (2) includes (3) a (4) meal (5)
  
2. Setting up a table. For a sentence of length  $n$ , we need  $(n+1)$  rows and  $(n+1)$  columns.
  
3. Traversing the input sentence left-to-right
  
4. Use the table to store constituents and their span.

# CKY example

Rule1: Det  $\rightarrow$  the  
 Rule 2: N  $\rightarrow$  flight  
 Rule 3: NP  $\rightarrow$  Det N

	1	2	3	4	5
0	Det	NP			S
1		N			
2					
3					
4					

the      flight      includes      a      meal

# CKY: lexical step ( $j = 5$ )

## Lexical lookup

- Matches N → meal

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	
<b>4</b>					N

# CKY: syntactic step ( $j = 5$ )

- *The flight includes a meal.*

Syntactic lookup

- We find that we have

$\text{NP} \rightarrow \text{Det N}$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			
<b>1</b>		N			
<b>2</b>			V		
<b>3</b>				Det	NP
<b>4</b>					N

# CKY: syntactic step ( $j = 5$ )

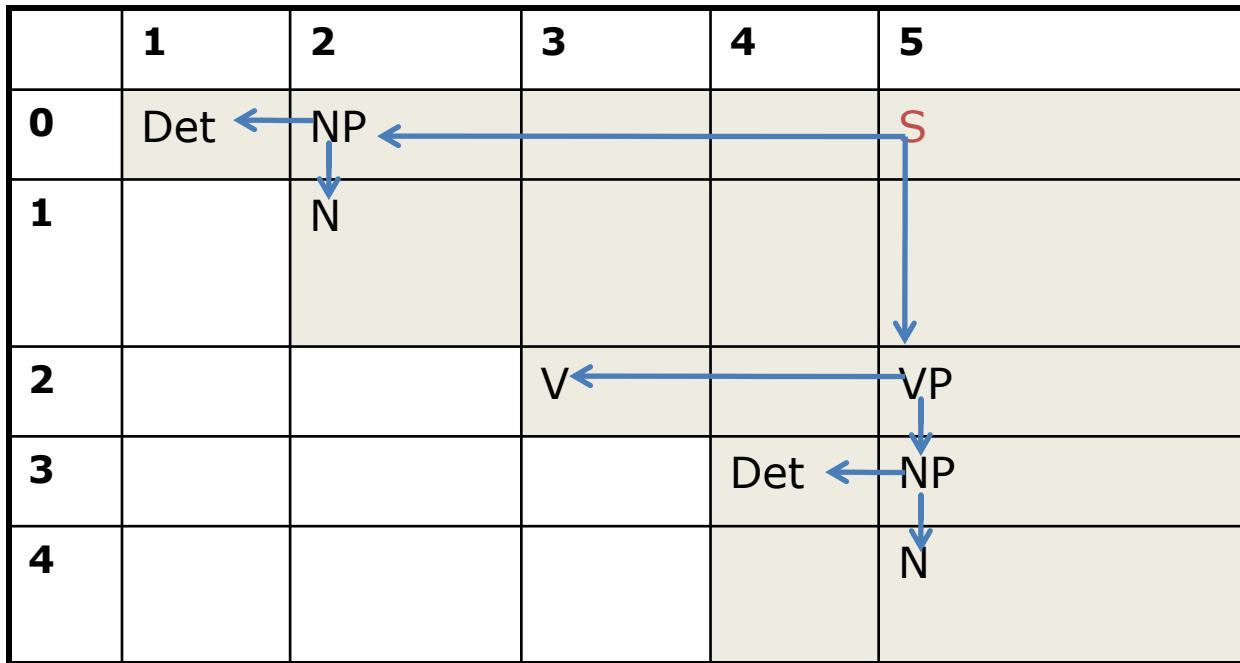
*The flight includes a meal.*

Syntactic lookup

- We find that we have S  
 $\rightarrow$  NP VP

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>0</b>	Det	NP			S
<b>1</b>		N			
<b>2</b>			V		VP
<b>3</b>				Det	NP
<b>4</b>					N

# From recognition to parsing



NB: This algorithm always fills the top “triangle” of the table!

# Motivation

---

- Context-free grammars can be generalized to include probabilistic information by adding it to CFG rule
- Probabilistic Context Free Grammars (PCFGs) are the simplest and most natural probabilistic model for tree structures and the algorithms for them are closely related to those for HMMs.
- PCFG are also known as Stochastic Context-Free Grammar (SCFG)

# Formal Definition of a PCFG

---

- A PCFG consists of:
  - A set of terminals,  $\{w^k\}$ ,  $k= 1, \dots, V$
  - A set of nonterminals,  $N^i$ ,  $i= 1, \dots, n$
  - A designated start symbol  $N^1$
  - A set of **rules**,  $\{N^i \rightarrow \xi^j\}$ , (where  $\xi^j$  is a sequence of terminals and nonterminals)
  - A corresponding set of **probabilities on rules** such that:  $\forall i \quad \sum_j P(N^i \rightarrow \xi^j) = 1$

# Probability of a Derivation Tree and a String



- The probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..k} p(r(i))$$

where  $r(1), \dots, r(k)$  are the rules of the CFG used to generate the sentence  $w_{1m}$  of which  $T$  is a parse.

- The probability of a sentence (according to grammar  $G$ ) is given by:

$$P(w_{1m}) = \sum_t P(w_{1m}, t) = \sum_{\{t: \text{yield}(t)=w_{1m}\}} P(t)$$

where  $t$  is a parse tree of the sentence. Need dynamic programming to make this efficient!

# Probability of a sentence

---

Given a probabilistic context-free grammar  $G$ , we can calculate the probability of a sentence (as opposed to a tree).

Observe that:

- As far as our grammar is concerned, a sentence is only a sentence if it can be recognised by the grammar (it is “legal”)
- There can be multiple parse trees for a sentence.
  - Many trees whose **yield** is the sentence
- The probability of the sentence is the sum of all the probabilities of the various trees that yield the sentence.

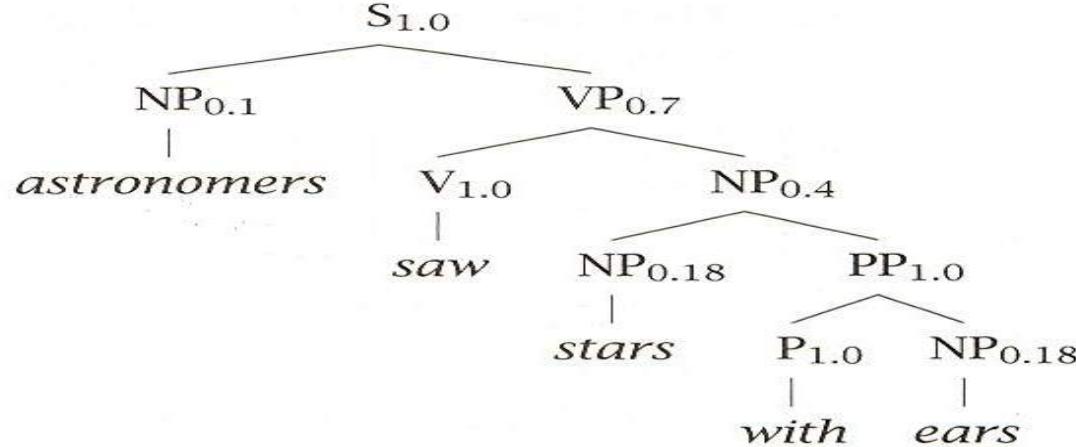
# Problems with PCFGs

---

- No Context
  - (immediate prior context, speaker, ...)
- No Lexicalization
  - “VP NP NP” more likely if verb is “hand” or “tell”
  - fail to capture lexical dependencies (n-grams do)
- No Structural Context
  - How NP expands depends on position

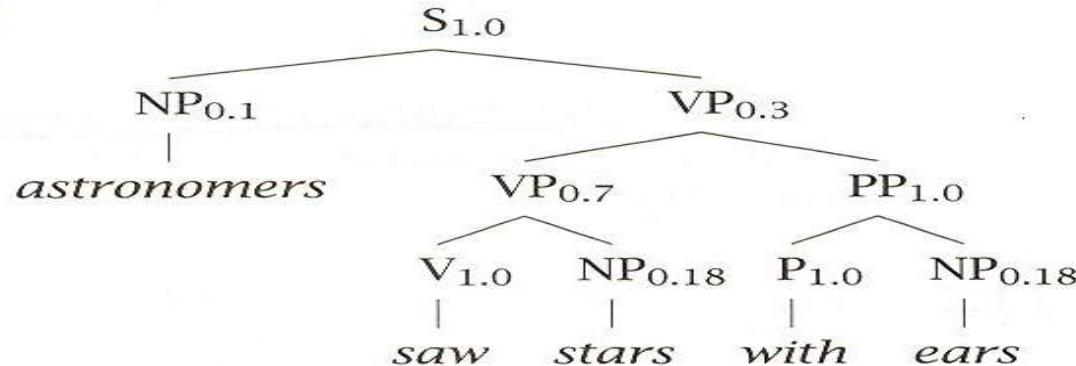
# Probability of a Derivation Tree

$t_1$ :



$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow astronomers$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow ears$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow saw$	0.04
$P \rightarrow with$	1.0	$NP \rightarrow stars$	0.18
$V \rightarrow saw$	1.0	$NP \rightarrow telescopes$	0.1

$t_2$ :



# Word/Tag Counts

	<b>N</b>	<b>V</b>	<b>ARI</b>	<b>P</b>	<b>TOTAL</b>
<i>flies</i>	21	23	0	0	44
<i>fruit</i>	49	5	1	0	55
<i>like</i>	10	30	0	21	61
<i>a</i>	1	0	201	0	202
<i>the</i>	1	0	300	2	303
<i>flower</i>	53	15	0	0	68
<i>flowers</i>	42	16	0	0	58
<i>birds</i>	64	1	0	0	65
<i>others</i>	592	210	56	284	1142
<b>TOTAL</b>	<b>833</b>	<b>300</b>	<b>558</b>	<b>307</b>	<b>1998</b>

# Lexical Probability Estimates

$$P(\text{the}|\text{ART}) = 300 / 558 = 0.54$$

ReARD	.54	ReART	.36
ReesN	.05	ReN	.01
Reesy	.05	Reesey	.03
Reey	.1	Reey	.5
Reed	.08	ReesN	.05
Reen	.02		

# The PCFG

- Below is a probabilistic CFG (PCFG) with probabilities derived from analyzing a parsed version of Allen's corpus.

<b>Rule</b>	<b>Count for IHs</b>	<b>Count for Rule</b>	<b>PROB</b>
1. $S \rightarrow NP VP$	300	300	1
2. $VP \rightarrow V$	300	116	.386
3. $VP \rightarrow VNP$	300	118	.393
4. $VP \rightarrow VNPPP$	300	66	.22
5. $NP \rightarrow NPPP$	1032	241	.23
6. $NP \rightarrow NN$	1032	92	.09
7. $NP \rightarrow N$	1032	141	.14
8. $NP \rightarrow ARTN$	1032	558	.54
9. $PP \rightarrow PNP$	307	307	1

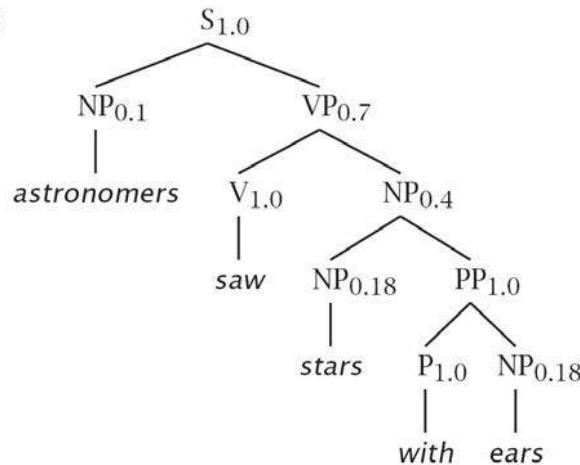
# Example

$S \rightarrow NP\ VP$	1.0	$NP \rightarrow NP\ PP$	0.4
$PP \rightarrow P\ NP$	1.0	$NP \rightarrow \text{astronomers}$	0.1
$VP \rightarrow V\ NP$	0.7	$NP \rightarrow \text{ears}$	0.18
$VP \rightarrow VP\ PP$	0.3	$NP \rightarrow \text{saw}$	0.04
$P \rightarrow \text{with}$	1.0	$NP \rightarrow \text{stars}$	0.18
$V \rightarrow \text{saw}$	1.0	$NP \rightarrow \text{telescopes}$	0.1

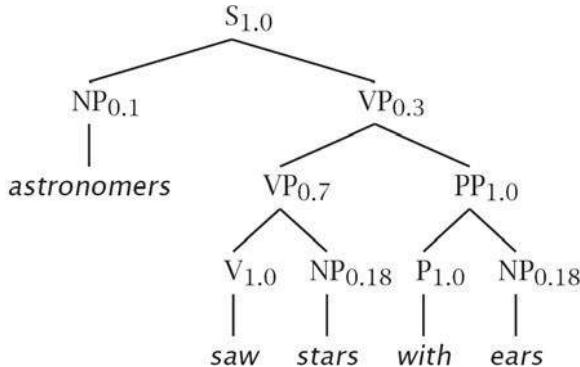
- Terminals      with, saw, astronomers, ears, stars, telescopes
- Nonterminals    S, PP, P, NP, VP, V
- Start symbol    S

# Probabilities

$t_1:$



$t_2:$



$$\begin{aligned}
 P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0009072
 \end{aligned}$$

$$\begin{aligned}
 P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \\
 &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
 &= 0.0006804
 \end{aligned}$$

$$P(w_{15}) = P(t_1) + P(t_2) = 0.0015876$$

# Using CKY to parse with a PCFG

- The basic CKY algorithm remains unchanged.
- However, rather than only keeping partial solutions in our table cells (i.e. The rules that match some input), we also keep their probabilities.

# Probabilistic CYK: syntactic step

- *The flight includes a meal.*

- $S \rightarrow NP\ VP [ .80 ]$
- $NP \rightarrow Det\ N [ .30 ]$
- $VP \rightarrow V\ NP [ .20 ]$
- $V \rightarrow includes [ .05 ]$
- $Det \rightarrow the [ .4 ]$
- $Det \rightarrow a [ .4 ]$
- $N \rightarrow meal [ .01 ]$
- $N \rightarrow flight [ .02 ]$

	1	2	3	4	5
0	Det (.4)	NP .0024			$S(0.0024 * 0.00001 * 0.80)$ .0000000192
1		N .02			
2			V .05		VP .00001
3				Det .4	NP .001
4					N .01

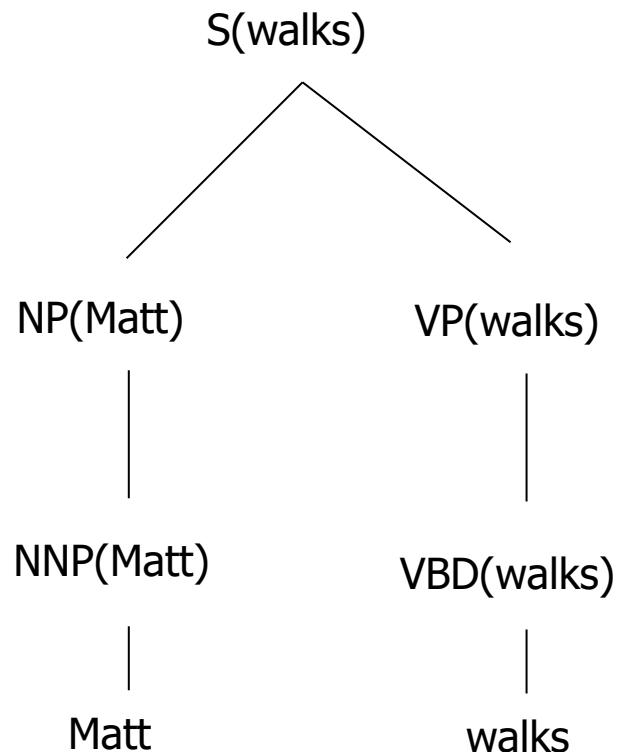
# Problems with PCFGs

---

- No Context
  - (immediate prior context, speaker, ...)
- No Lexicalization
  - “VP NP NP” more likely if verb is “hand” or “tell”
  - fail to capture lexical dependencies (n-grams do)
- No Structural Context
  - How NP expands depends on position

# Lexicalised PCFGs: *Matt walks*

- Makes probabilities partly dependent on lexical content.
- $P(VP \rightarrow VBD | VP)$  becomes:  
 $P(VP \rightarrow VBD | VP, h(VP)=\text{walks})$
- NB: normally, we can't assume that all heads of a phrase of category C are equally probable.



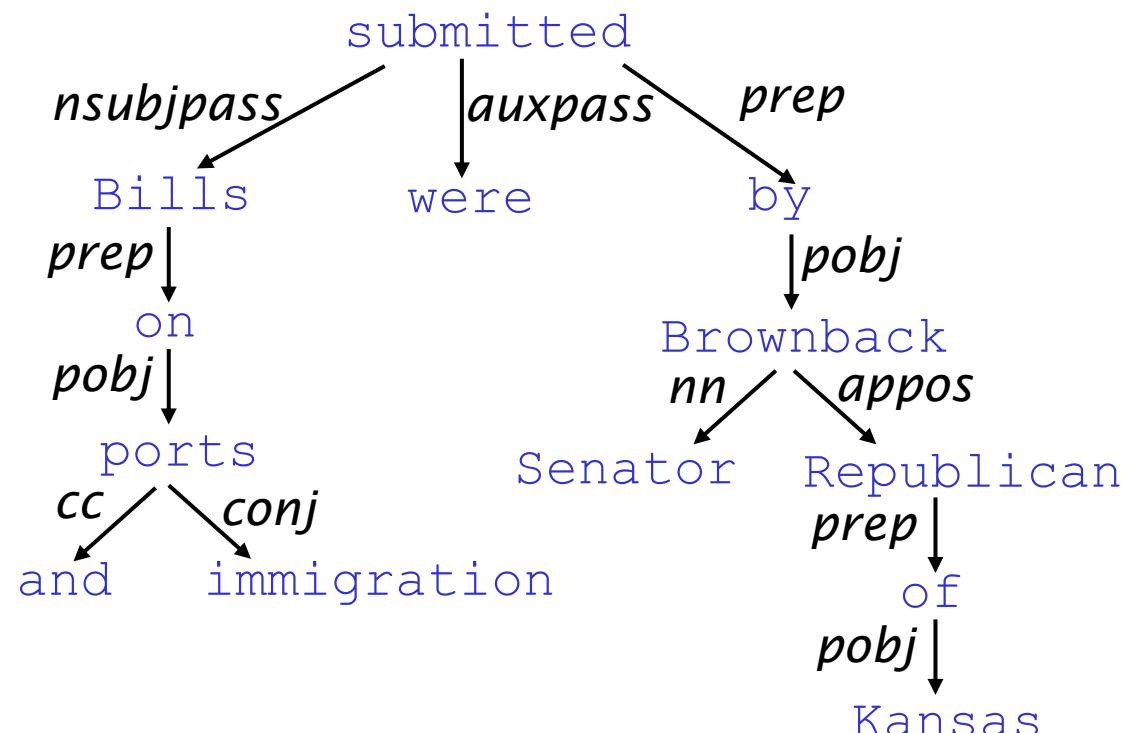
# Practical problems for lexicalised PCFGs

- Data sparseness: we don't necessarily see all heads of all phrasal categories often enough in the training data
- Flawed assumptions: lexical dependencies occur elsewhere, not just between head and complement
  - *I got the easier problem of the two to solve*
  - *of the two* and *to solve* are very likely because of the prehead modifier *easier*

# Session 7: Dependency Parsing

## Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies



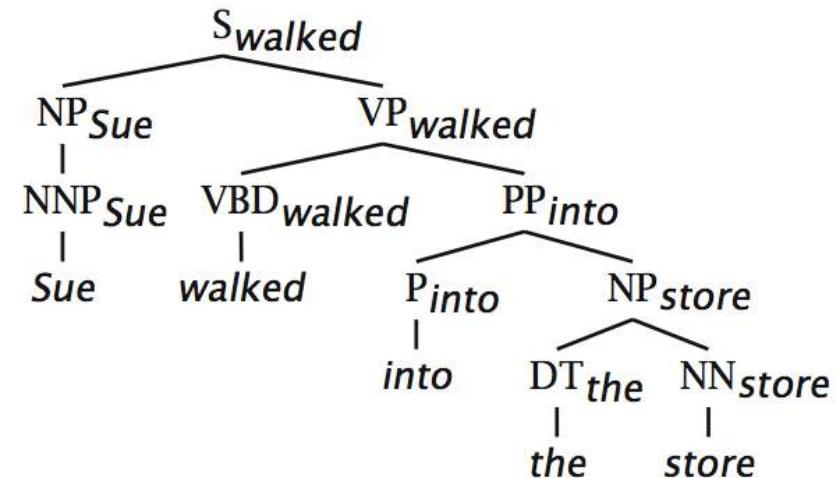
# Relation between phrase structure and dependency structure

The closure of dependencies give constituency from a dependency tree  
 But the dependents of a word must be at the same level (i.e., “flat”)

A dependency grammar has a notion of a head. Officially, CFGs don’t.  
 But modern linguistic theory and all modern statistical parsers  
 (Charniak, Collins, Stanford, ...) do, via hand-written phrasal “head  
 rules”:

- The head of a Noun Phrase is a noun/number/adj/...
- The head of a Verb Phrase is a verb/modal/....

The head rules can  
 be used to extract a  
 dependency parse  
 from a CFG parse



# Dependency graph

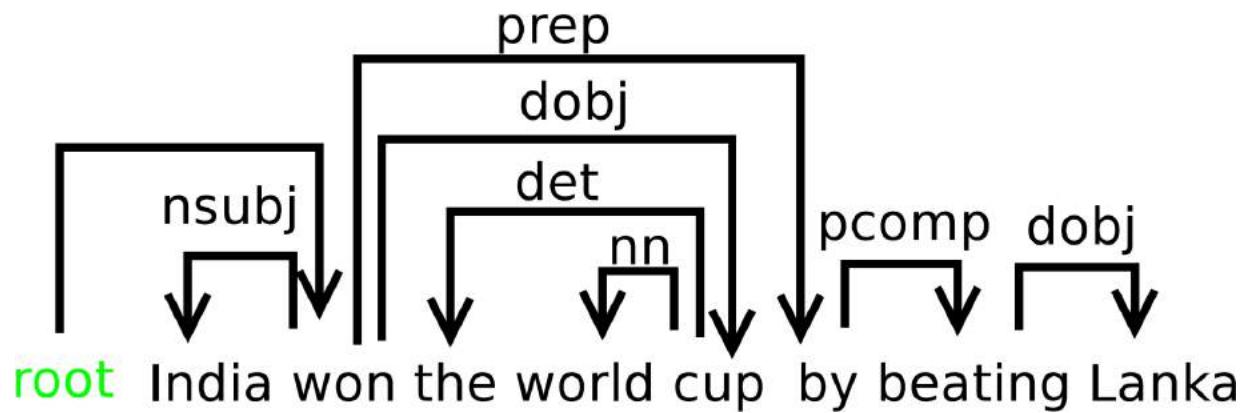
- A dependency structure can be defined as a directed graph  $G$ , consisting of
  - ▶ a set  $V$  of nodes,
  - ▶ a set  $A$  of arcs (edges),
- Labeled graphs:
  - ▶ Nodes in  $V$  are labeled with word forms (and annotation).
  - ▶ Arcs in  $A$  are labeled with dependency types.
- Notational convention:
  - ▶ Arc  $(w_i, d, w_j)$  links head  $w_i$  to dependent  $w_j$  with label  $d$
  - ▶  $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
  - ▶  $i \rightarrow j \equiv (i, j) \in A$
  - ▶  $i \rightarrow^* j \equiv i = j \vee \exists k : i \rightarrow k, k \rightarrow^* j$

# Formal conditions on dependency graph

---

- $G$  is connected:
  - For every node  $i$  there is a node  $j$  such that  $i \rightarrow j$  or  $j \rightarrow i$ .
- $G$  is acyclic:
  - if  $i \rightarrow j$  then not  $j \rightarrow^* i$ .
- $G$  obeys the single head constraint:
  - if  $i \rightarrow j$  then not  $k \rightarrow j$ , for any  $k \neq i$ .
- $G$  is projective:
  - if  $i \rightarrow j$  then  $j \rightarrow^* k$ , for any  $k$  such that both  $j$  and  $k$  lie on the same side of  $i$ .

# Example Dependency Parse



# Deterministic parsing

## *Basic idea*

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

## *Configurations*

A parser configuration is a triple  $c = (S, B, A)$ , where

- $S$  : a stack  $[ \dots, w_i ]_S$  of partially processed words,
- $B$  : a buffer  $[ w_j, \dots ]_B$  of remaining input words,
- $A$  : a set of labeled arcs  $(w_i, d, w_j)$ .

### Stack

$[\text{sent, her, a}]_S$

### Buffer

$[\text{letter, .}]_B$

### Arcts

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

# Transition based systems for Dependency parsing

A transition system for dependency parsing is a quadruple  $S = (C, T, c_s, C_t)$ , where

- $C$  is a set of configurations,
- $T$  is a set of transitions, such that  $t : C \rightarrow C$ ,
- $c_s$  is an initialization function
- $C_t \subseteq C$  is a set of terminal configurations.

A transition sequence for a sentence  $x$  is a set of configurations

$C_{0,m} = (c_0, c_1, \dots, c_m)$  such that

$c_0 = c_s(x)$ ,  $c_m \in C_t$ ,  $c_i = t(c_{i-1})$  for some  $t \in T$

**Initialization:**  $([], [w_1, \dots, w_n]_B, \{\})$

**Termination:**  $(S, [], A)$

# Arc eager parsing(Malt parser)

$$\text{Left-Arc}(d) \frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots]_S, [w_j, \dots]_B, A \cup \{(w_j, d, w_i)\})} \neg \text{HEAD}(w_i)$$

$$\text{Right-Arc}(d) \frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots, w_i, w_j]_S, [\dots]_B, A \cup \{(w_i, d, w_j)\})}$$

$$\text{Reduce} \frac{([\dots, w_i]_S, B, A)}{([\dots]_S, B, A)} \text{HEAD}(w_i)$$

$$\text{Shift} \frac{([\dots]_S, [w_i, \dots]_B, A)}{([\dots, w_i]_S, [\dots]_B, A)}$$

# Arc Eager Parsing example

## Transitions:

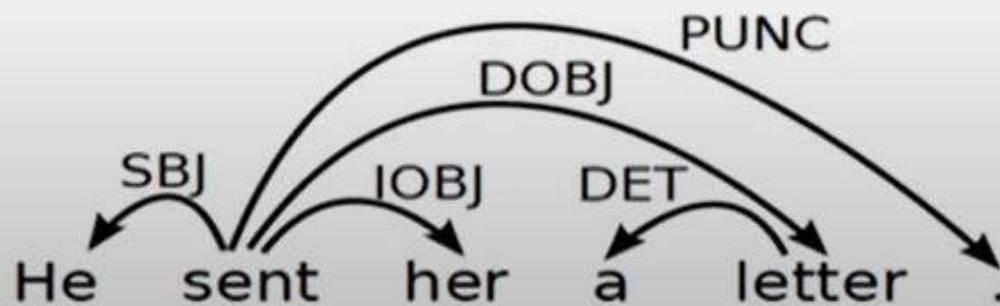
**Stack**

It ]s

**Buffer**

[He, sent, her, a, letter, .]B

**Arcs**



# Arc Eager Parsing example

**Transitions:** SH-LA

—    =

**Stack**

[ ]s

**Buffer**

[sent, her, a, letter, .]<sub>B</sub>

**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent



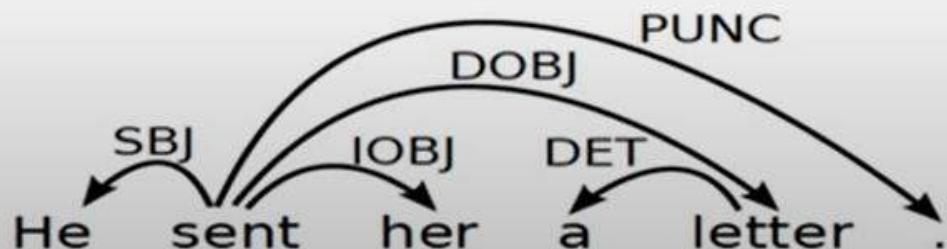
# Arc Eager Parsing example

**Transitions:** SH-LA-SH

**Stack**  
[sent]<sub>S</sub>

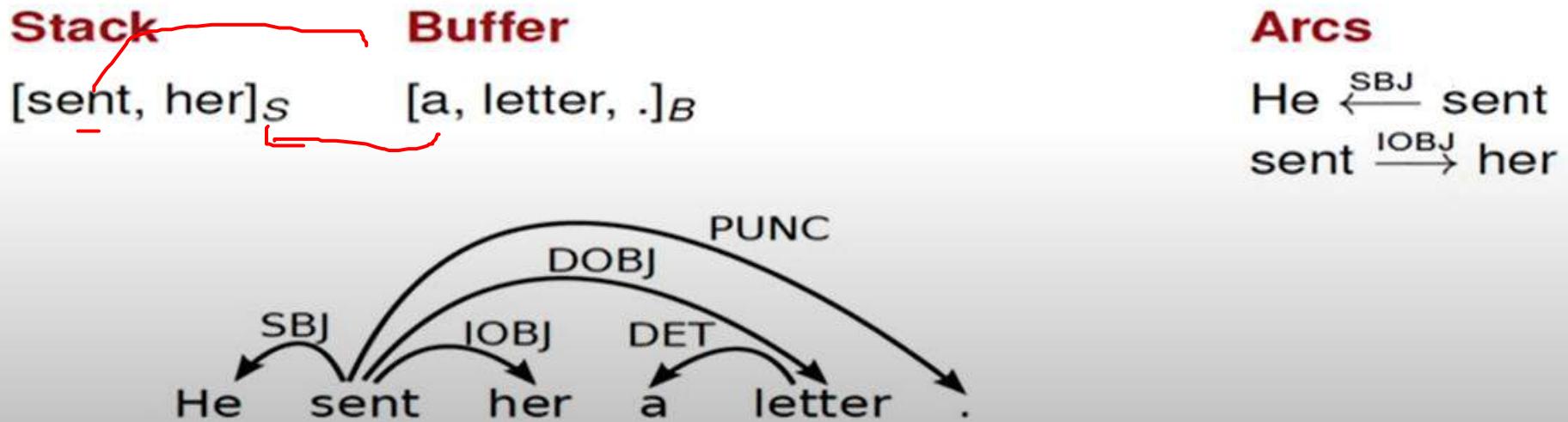
**Buffer**  
[her, a, letter, .]<sub>B</sub>

**Arcs**  
He ←<sup>SBJ</sup> sent



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH

**Stack**

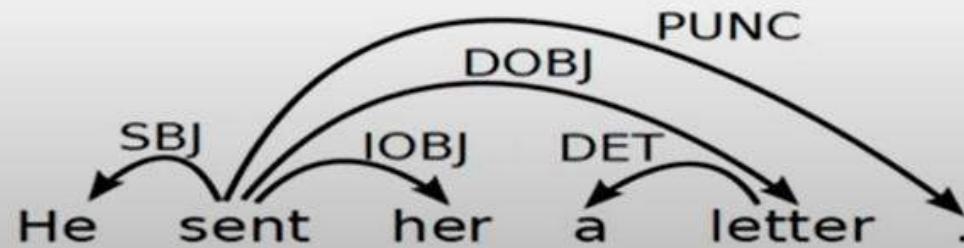
[sent, her, a]<sub>S</sub>

**Buffer**

[letter, .]<sub>B</sub>

**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her



# Arc Eager Parsing example

**Transitions:** SH-LA-SH-RA-SH-LA

## Stack

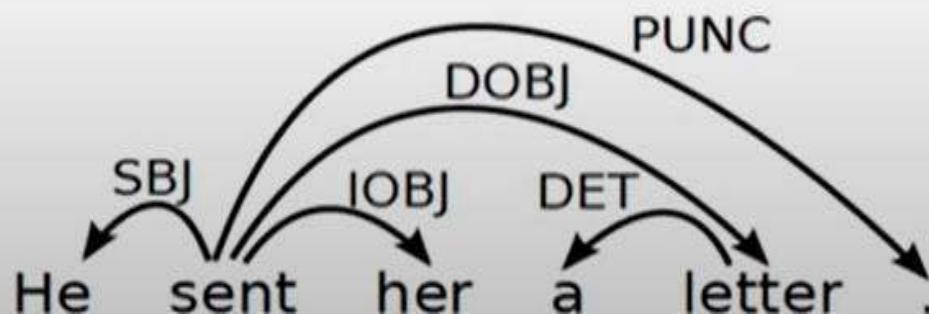
[sent, her]<sub>S</sub>

## Buffer

[letter, .]<sub>B</sub>

## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter



# Arc Eager Parsing example

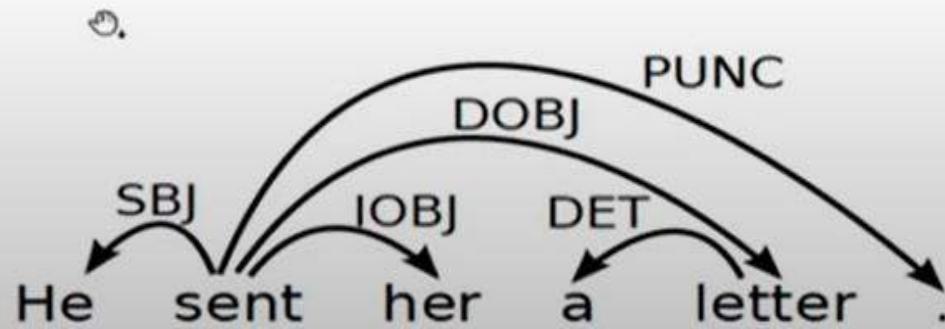
**Transitions:** SH-LA-SH-RA-SH-LA-RE

## Stack

[sent]<sub>S</sub>

## Buffer

[letter, .]<sub>B</sub>



## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter

# Arc Eager Parsing example

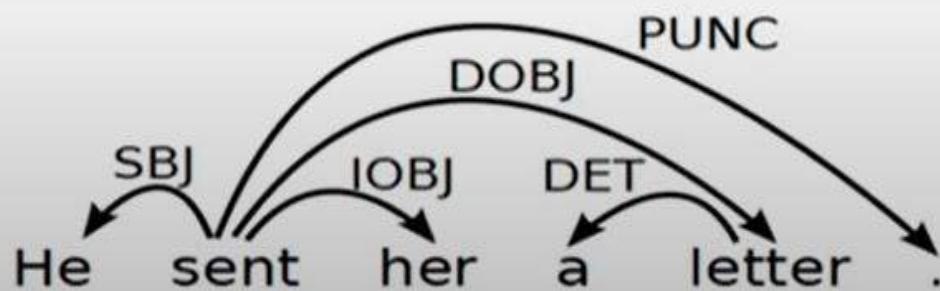
**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA

## Stack

[sent, letter]s

## Buffer

[.]<sub>B</sub>



## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter

# Arc Eager Parsing example

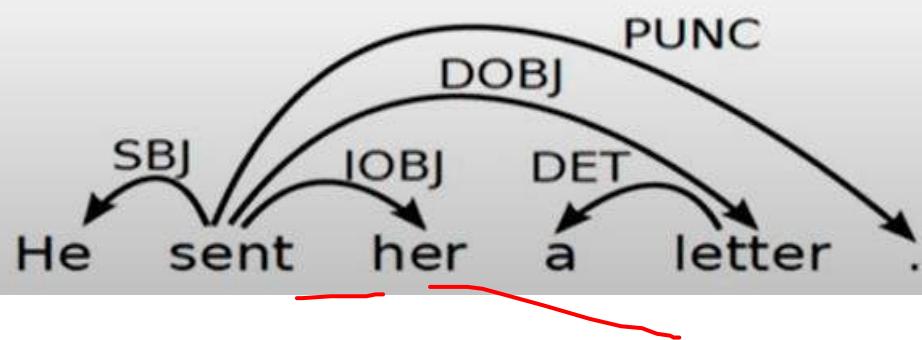
**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE

## Stack

[sent]<sub>S</sub>

## Buffer

[.]<sub>B</sub>



## Arcs

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter

# Arc Eager Parsing example

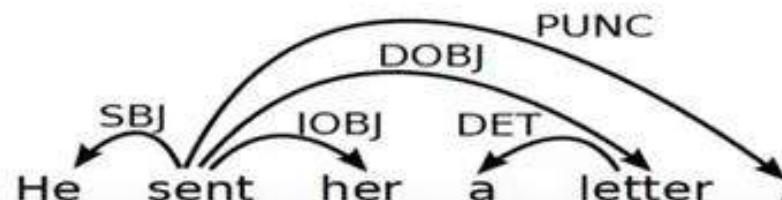
**Transitions:** SH-LA-SH-RA-SH-LA-RE-RA-RE-RA

**Stack**

[sent, .]s

**Buffer**

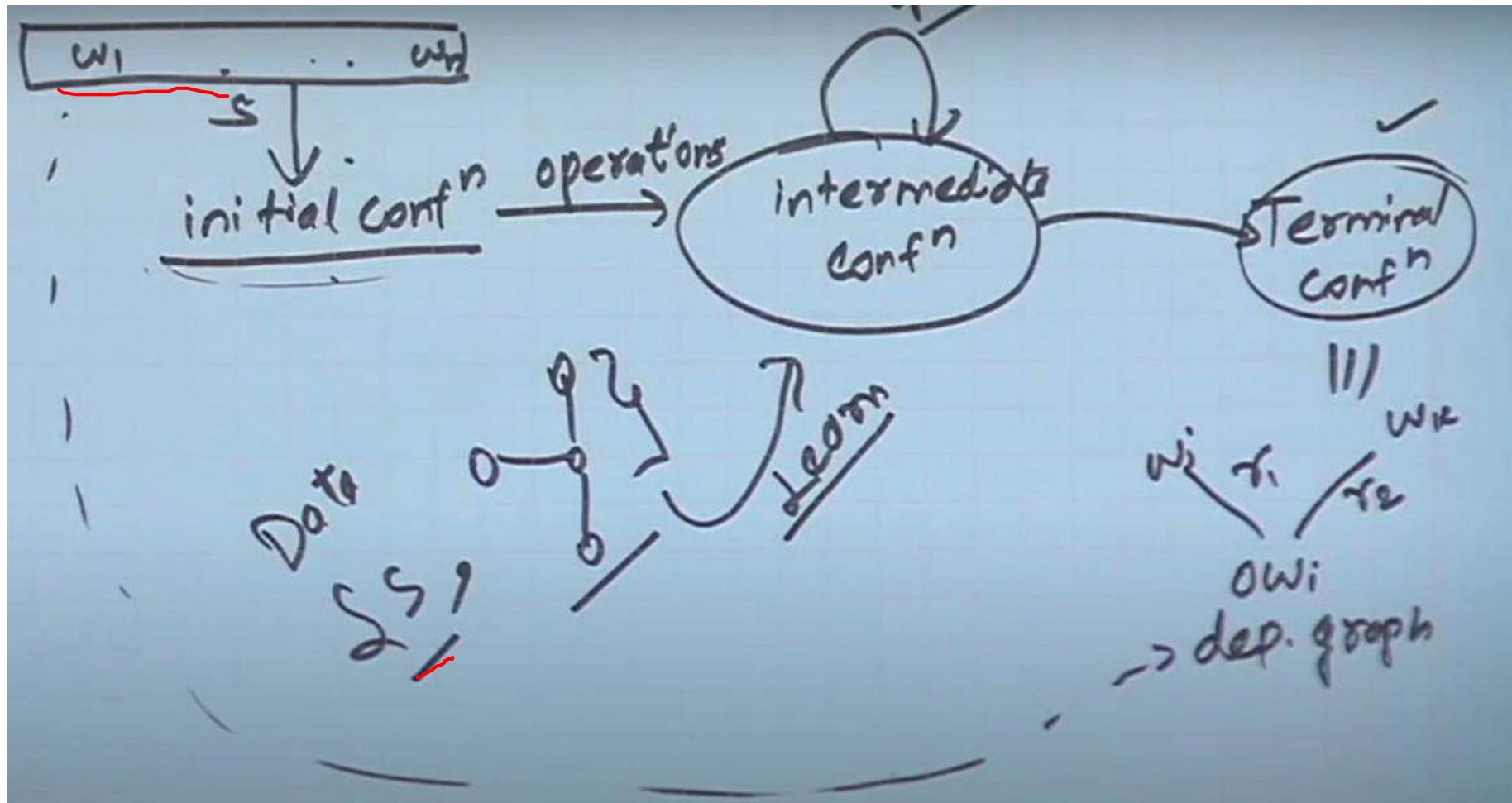
[ ]<sub>B</sub>



**Arcs**

He  $\xleftarrow{\text{SBJ}}$  sent  
 sent  $\xrightarrow{\text{IOBJ}}$  her  
 a  $\xleftarrow{\text{DET}}$  letter  
 sent  $\xrightarrow{\text{DOBJ}}$  letter  
 sent  $\xrightarrow{\text{PUNC}}$

# Learning weights

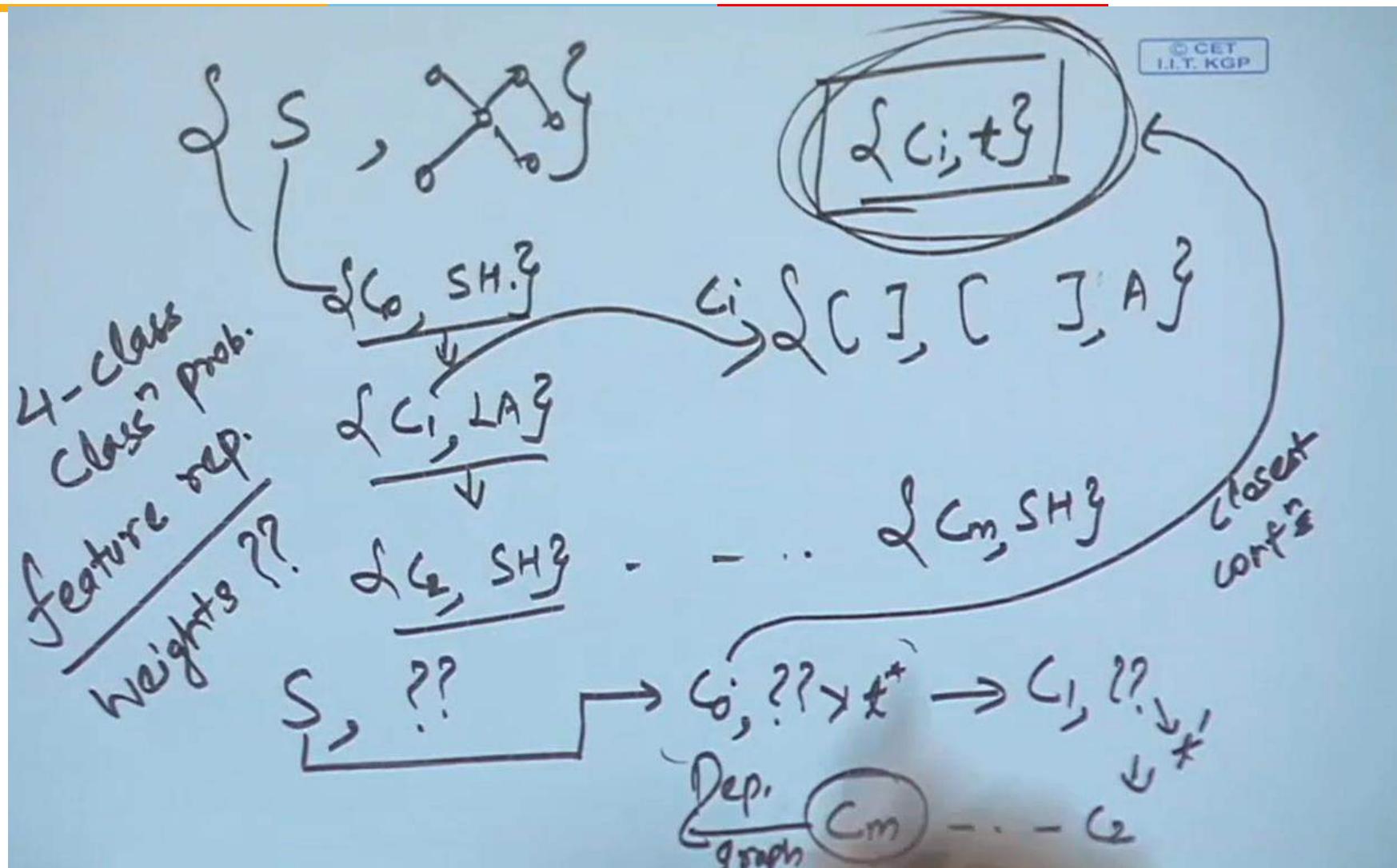


# Deriving dependency graph for a new sentence

innovate

achieve

lead



# Classifier for learning the transmission

## *Data-driven deterministic parsing:*

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

## *Learning Problem*

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

## *Three issues*

- How to represent configurations by feature vectors?
- How to derive training data from treebanks?
- How to learn classifiers?

# Feature Models

A feature representation  $f(c)$  of a configuration  $c$  is a vector of simple features  $f_i(c)$ .

## *Typical Features*

- Nodes:
  - Target nodes (top of  $S$ , head of  $B$ )
  - Linear context (neighbors in  $S$  and  $B$ )
  - Structural context (parents, children, siblings in  $G$ )
- Attributes:
  - Word form (and lemma)
  - Part-of-speech (and morpho-syntactic features)
  - Dependency type (if labeled)
  - Distance (between target tokens)

To guide the parser, a linear classifier can be used:

$$t^* = \arg \max_t w.f(c, t)$$

Weight vector  $w$  learned from treebank data.

### *Using classifier at run-time*

```
PARSE( $w_1, \dots, w_n$ )
1    $c \leftarrow ([], S, [w_1, \dots, w_n]_B, \{\})$ 
2   while  $B_c \neq []$ 
3        $t^* \leftarrow \arg \max_t w.f(c, t)$ 
4        $c \leftarrow t^*(c)$ 
5   return  $T = (\{w_1, \dots, w_n\}, A_c)$ 
```

# Training Data

---

- Training instances have the form  $(f(c), t)$ , where
  - ▶  $f(c)$  is a feature representation of a configuration  $c$ ,
  - ▶  $t$  is the correct transition out of  $c$  (i.e.,  $o(c) = t$ ).
- Given a dependency treebank, we can sample the oracle function  $o$  as follows:
  - ▶ For each sentence  $x$  with gold standard dependency graph  $G_x$ , construct a transition sequence  $C_{0,m} = (c_0, c_1, \dots, c_m)$  such that
$$c_0 = c_s(x),$$
$$G_{c_m} = G_x$$
  - ▶ For each configuration  $c_i (i < m)$ , we construct a training instance  $(f(c_i), t_i)$ , where  $t_i(c_i) = c_{i+1}$ .

# Standard Oracle for Arc Eager parsing

$o(c, T) =$

- **Left-Arc** if  $\text{top}(S_c) \leftarrow \text{first}(B_c)$  in  $T$
- **Right-Arc** if  $\text{top}(S_c) \rightarrow \text{first}(B_c)$  in  $T$
- **Reduce** if  $\exists w < \text{top}(S_c) : w \leftrightarrow \text{first}(B_c)$  in  $T$
- **Shift** otherwise

# Online learning with an oracle

```
LEARN({ $T_1, \dots, T_N$ })
1    $w \leftarrow 0.0$ 
2   for  $i$  in  $1..K$ 
3     for  $j$  in  $1..N$ 
4        $c \leftarrow ([], [w_1, \dots, w_{n_j}]_B, \{\})$ 
5       while  $B_c \neq []$ 
6          $t^* \leftarrow \arg \max_t w.f(c, t)$ 
7          $t_o \leftarrow o(c, T_i)$ 
8         if  $t^* \neq t_o$ 
9            $w \leftarrow w + f(c, t_o) - f(c, t^*)$ 
10           $c \leftarrow t_o(c)$ 
11   return  $w$ 
```

Oracle  $o(c, T_i)$  returns the optimal transition of  $c$  and  $T_i$

# Example

Consider the sentence, 'John saw Mary'.

- Draw a dependency graph for this sentence.
- Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are 'Nouns', while the POS tag of *saw* is 'Verb'. Assume that your features correspond to the following conditions:
  - ▶ The stack is empty 
  - ▶ Top of stack is Noun and Top of buffer is Verb 
  - ▶ Top of stack is Verb and Top of buffer is Noun 

Initialize the weights of all your features to 5.0, except that in all of the above cases, you give a weight of 5.5 to Left-Arc. Define your feature vector and the initial weight vector.

- Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

# Example

$F(c,t) = [(c_0, LA), (c_1, LA), (c_2, LA) | (c_0, RA), (c_1, RA), (c_2, RA) \dots]$

$W = [5.5, 5.5, 5.5 | 5.0, 5.0, 5.0 | 5.0, 5.0, 5.0 | \dots]$

So for the given conditions

$F(c, LA) = [1, 0, 0 | 0, 0, 0 | \dots]$

$F(c, RA) = [0, 0, 0 | 1, 0, 0 | \dots]$

$t^* = \text{argmax}(w^* f(c, t))$

$t^* = LA$

as per oracle /optimal transition  $t^0 = SH$

So we need to update the weights|

To update the weights

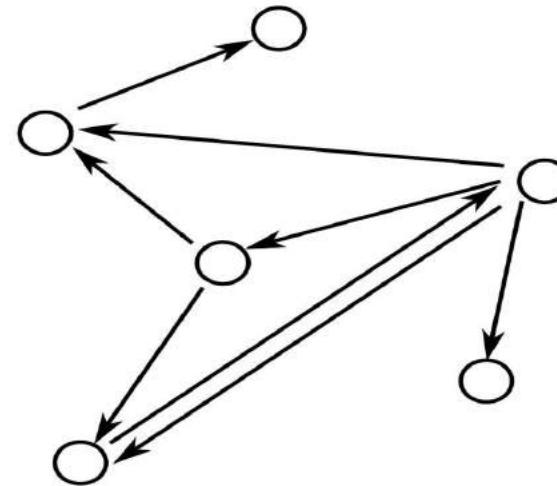
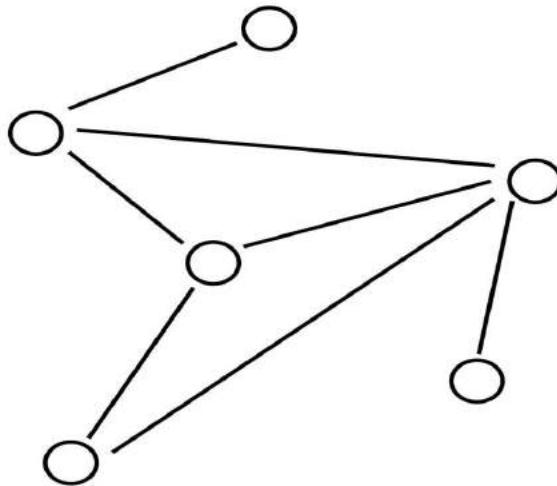
$W = W + f(c, t^0) - f(c, t^*)$

$W = [5.5, 5.5, 5.5, | 5.0, 5.0, \dots] + [0, 0, 0 | 0, 0, 0 | \dots, 1, 0, 0] - [1, 0, 0 | 0, 0, 0 | \dots]$

New vector =  $[4.5, 5.5, 5.5 | 5.0, \dots, 6.0, 5.0, 5.0]$

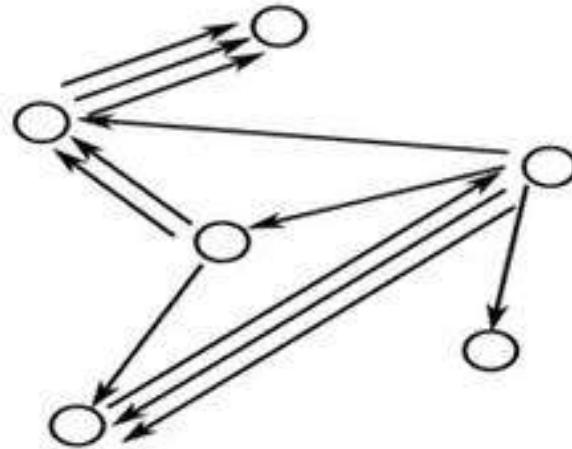
# Graph

- ▶ A graph  $G = (V, A)$  is a set of vertices  $V$  and arcs  $(i, j) \in A$ , where  $i, j \in V$
- ▶ Undirected graphs:  $(i, j) \in A \Leftrightarrow (j, i) \in A$
- ▶ **Directed graphs (digraphs):**  $(i, j) \in A \not\Leftrightarrow (j, i) \in A$



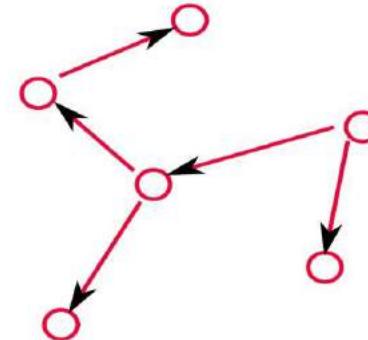
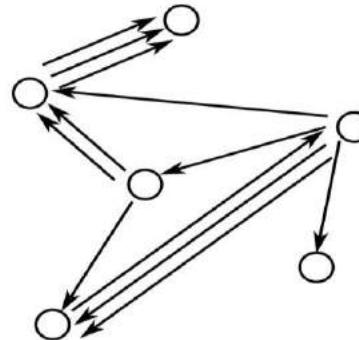
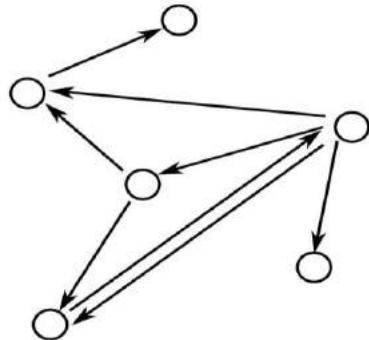
# Multi Digraph

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i, j, k) \in A$  represents the  $k^{th}$  arc from vertex  $i$  to vertex  $j$ .



# Directed Spanning Trees

- ▶ A directed spanning tree of a (multi-)digraph  $G = (V, A)$ , is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- ▶ A spanning tree of the following (multi-)digraphs



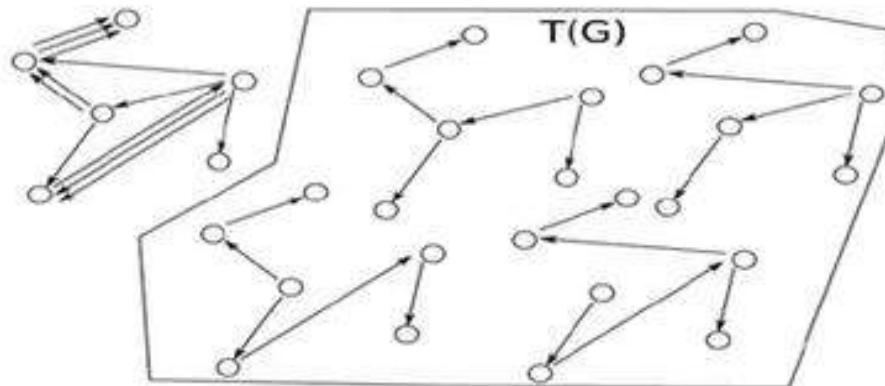
# Weighted Spanning tree

- Assume we have a weight function for each arc in a multi-digraph  $G = (V, A)$ .
- Define  $w_{ij}^k \geq 0$  to be the weight of  $(i, j, k) \in A$  for a multi-digraph
- Define the weight of directed spanning tree  $G'$  of graph  $G$  as

$$w(G') = \sum_{(i,j,k) \in G'} w_{ij}^k$$

# MST

Let  $T(G)$  be the set of all spanning trees for graph  $G$



The MST problem

Find the spanning tree  $G'$  of the graph  $G$  that has the highest weight

$$G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \sum_{(i,j,k) \in G'} w_{ij}^k$$

# Finding MST

## Directed Graph

For each sentence  $x$ , define the directed graph  $G_x = (V_x, E_x)$  given by

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

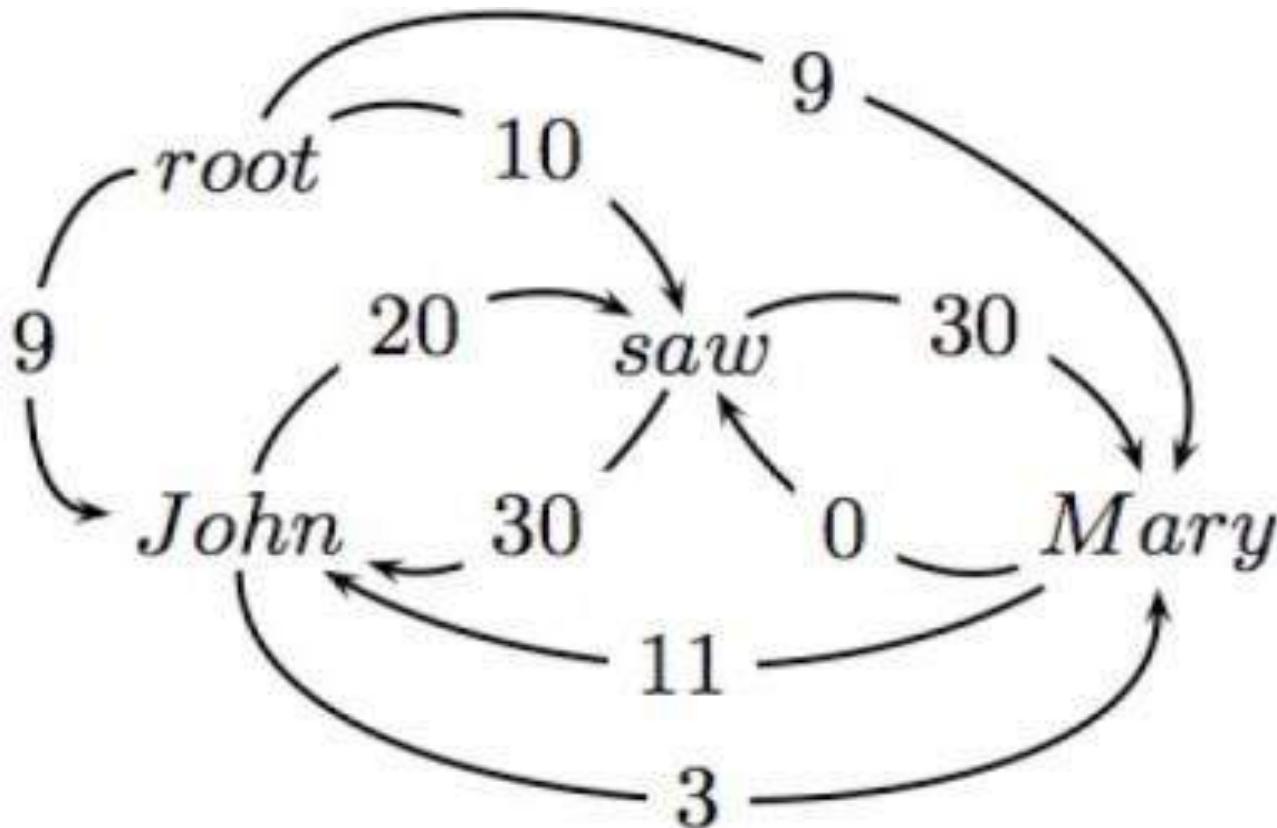
$G_x$  is a graph with

- the sentence words and the dummy root symbol as vertices and
- a directed edge between every pair of distinct words and
- a directed edge from the root symbol to every word

# Chu-Liu-Edmonds algorithm

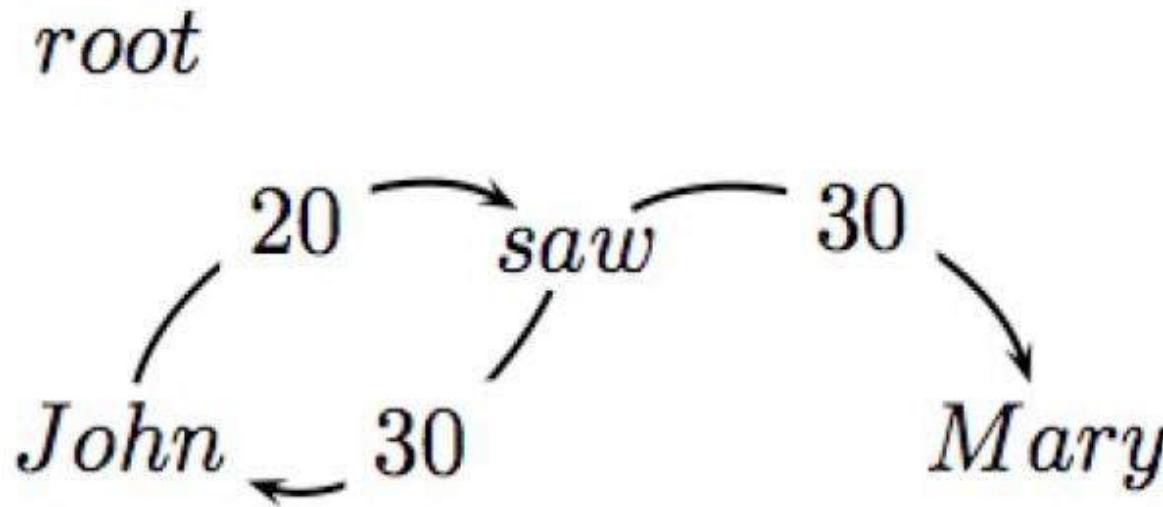
- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - Identify the cycle and contract it into a single vertex.
  - Recalculate edge weights going into and out of the cycle.

# Chu-Liu-Edmonds Example



# Chu-Liu-Edmonds Example

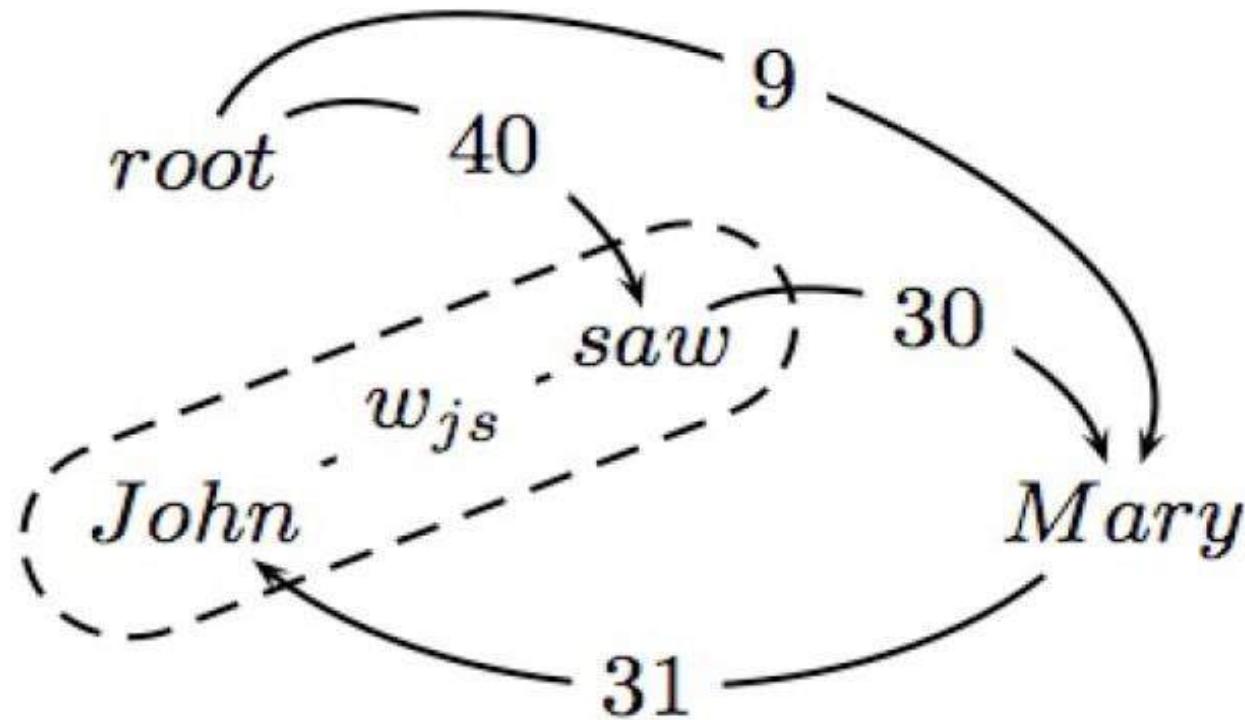
- ▶ Find highest scoring incoming arc for each vertex



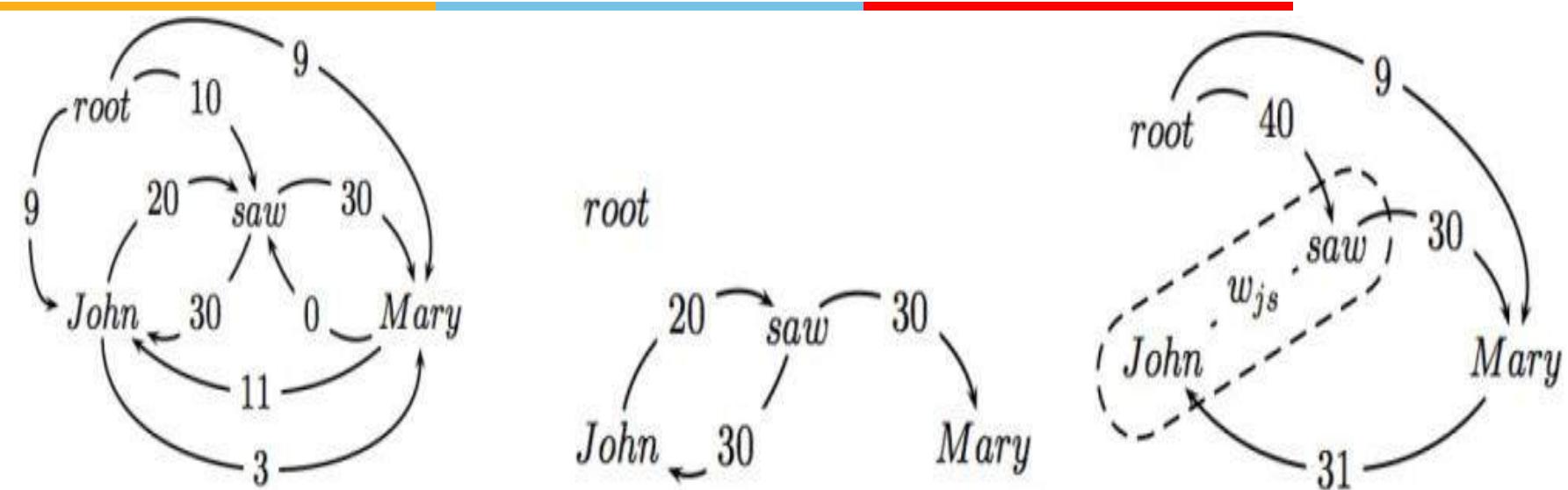
- ▶ If this is a tree, then we have found MST!!

# Chu-Liu-Edmonds Example

- ▶ If not a tree, identify cycle and contract
- ▶ Recalculate arc weights into and out-of cycle

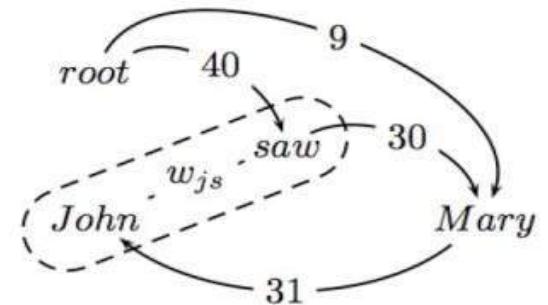
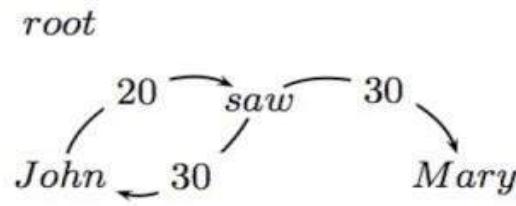
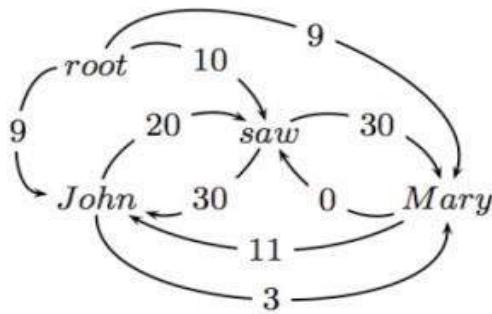


# Chu-Liu-Edmonds Example



- ▶ Outgoing arc weights
  - ▶ Equal to the max of outgoing arc over all vertexes in cycle
  - ▶ e.g., John → Mary is 3 and saw → Mary is 30

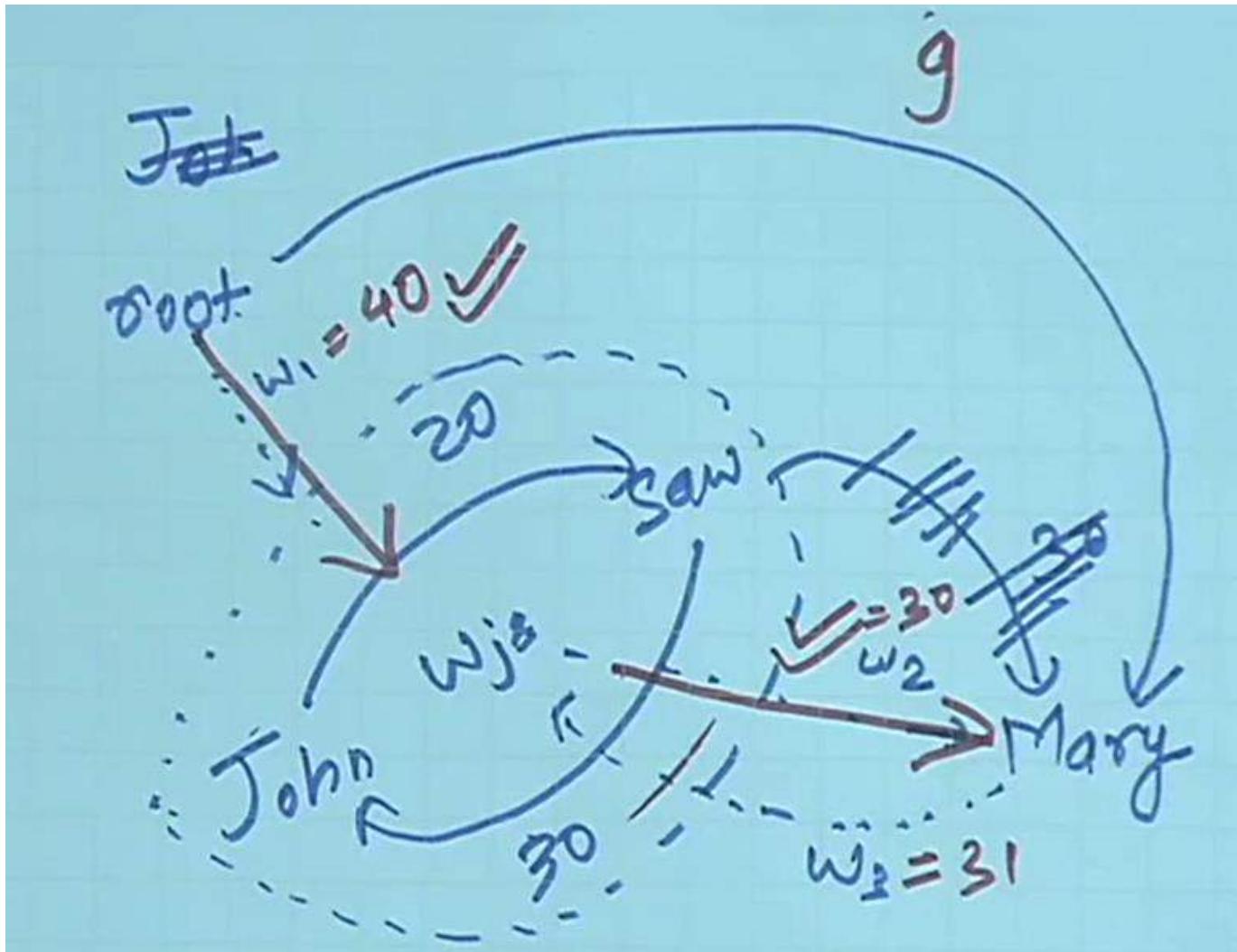
# Chu-Liu-Edmonds Example



## ► Incoming arc weights

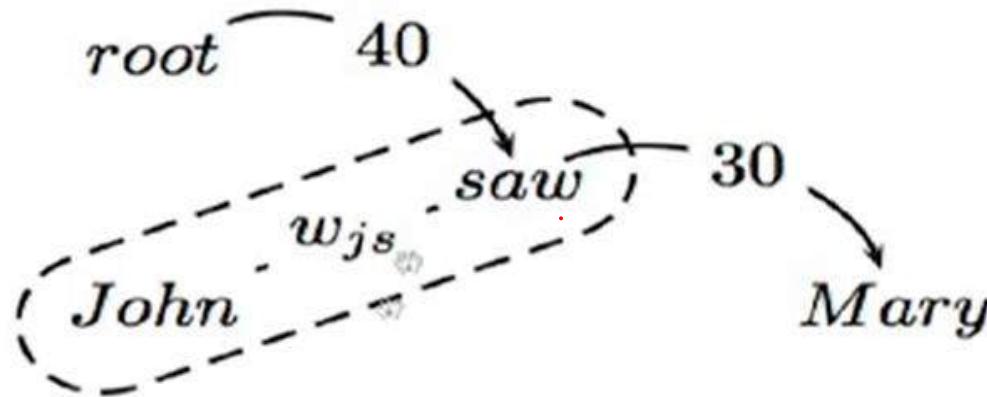
- ▶ Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
- ▶  $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40 (\*\*)
- ▶  $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

# Chu-Liu-Edmonds Example



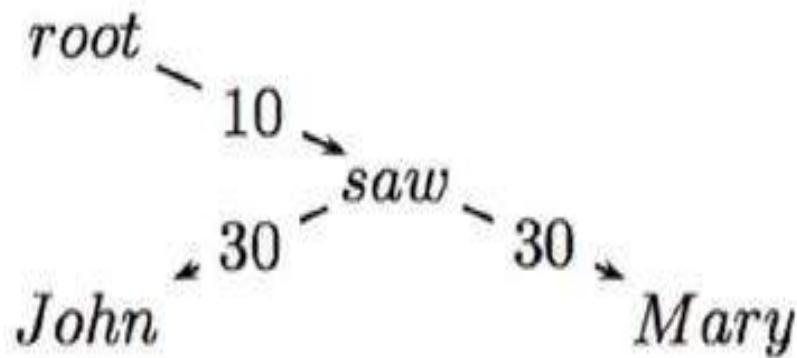
# Chu-Liu-Edmonds Example

Calling the algorithm again on the contracted graph:



- This is a tree and the MST for the contracted graph
- Go back up the recursive call and reconstruct final graph

# Chu-Liu-Edmonds Example



- The edge from  $w_{js}$  to *Mary* was from *saw*
- The edge from *root* to  $w_{js}$  represented a tree from *root* to *saw* to *John*.

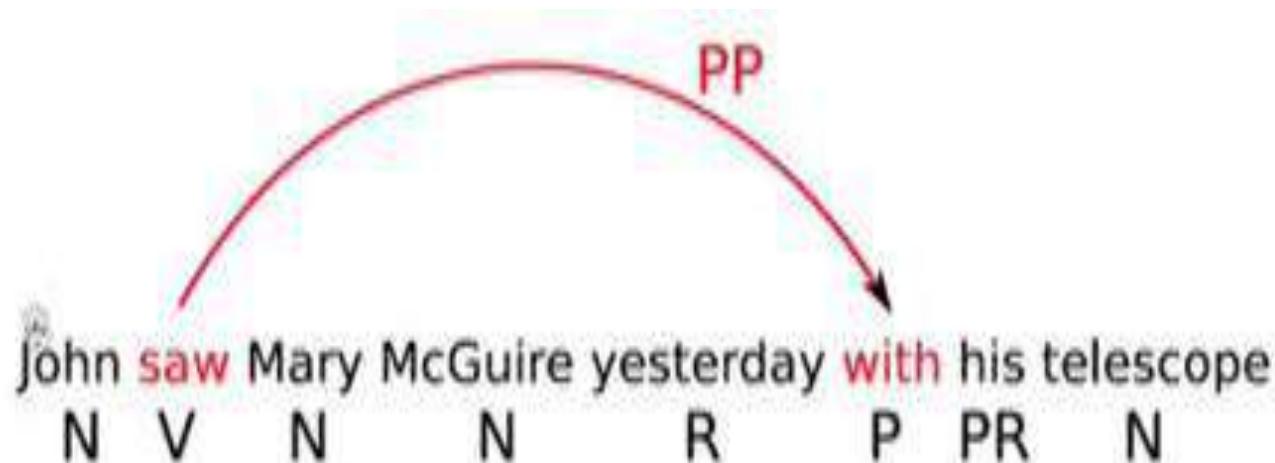
# Linear classifiers

---

$$w_{ij}^k = w.f(i,j,k)$$

- Arc weights are a linear combination of features of the arc  $f(i,j,k)$  and a corresponding weight vector  $w$
- What arc features?

# Arc features



## Features

Identities of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head = saw & dependent=with

# Arc features



## Features

Part-of-speech tags of the words  $w_i$  and  $w_j$  for a label  $l_k$   
head-pos = Verb & dependent-pos=Preposition

# Arc features



## Features

Part-of-speech of words surrounding and between  $w_i$  and  $w_j$

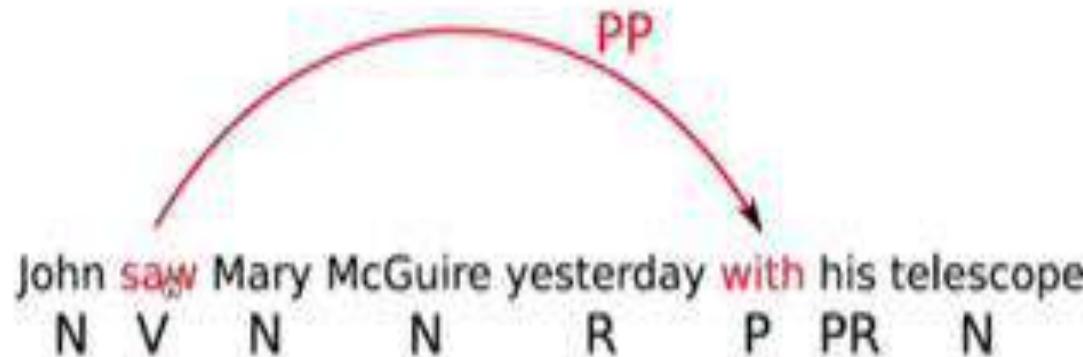
inbetween-pos = Noun

inbetween-pos = Adverb

dependent-pos-right = Pronoun

head-pos-left=Noun

# Arc features



## Features

Number of words between  $w_i$  and  $w_j$ , and their orientation

arc-distance = 3

arc-direction = right

# Learning the parameters

- Re-write the inference problem

$$G = \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w_{ij}^k$$

$$= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k)$$

$$= \arg \max_{G \in T(G_x)} w \cdot \underline{f(G)}$$

# Inference based learning

---

Training data:  $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

1.  $w^{(0)} = 0; i = 0$
2. **for**  $n : 1..N$
3.     **for**  $t : 1..|T|$
4.         Let  $G' = argmax_{G'} w^{(i)}, f(G')$
5.         if  $G' \neq G_t$
6.              $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7.          $i = i + 1$
8.     **return**  $w^i$

# EXAMPLE

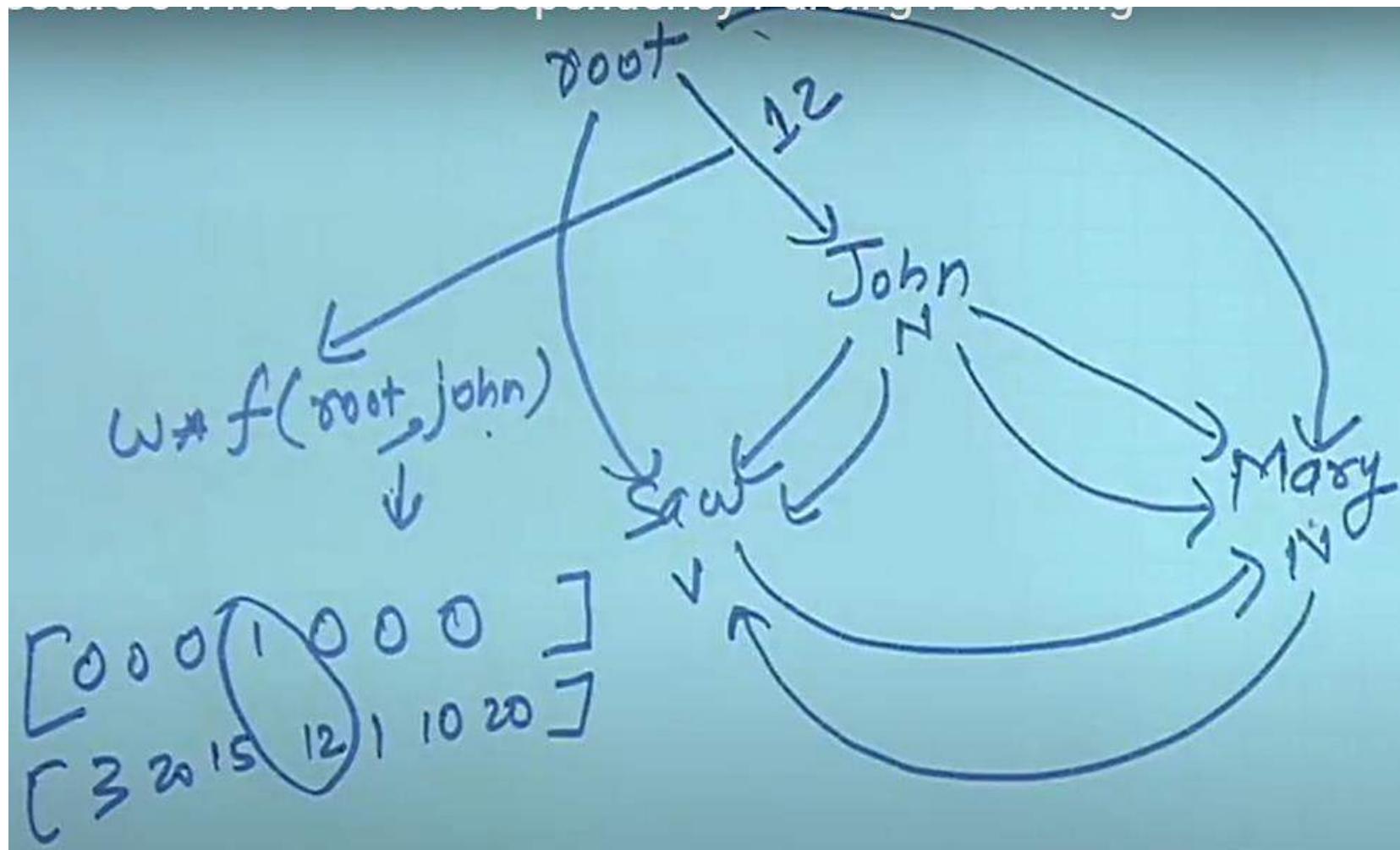
Suppose you are training MST Parser for dependency and the sentence, "John saw Mary" occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, "rel". Thus, for every arc from word  $w_i$  to  $w_j$ , your features may be simplified to depend only on words  $w_i$  and  $w_j$  and not on the relation label.

Below is the set of features

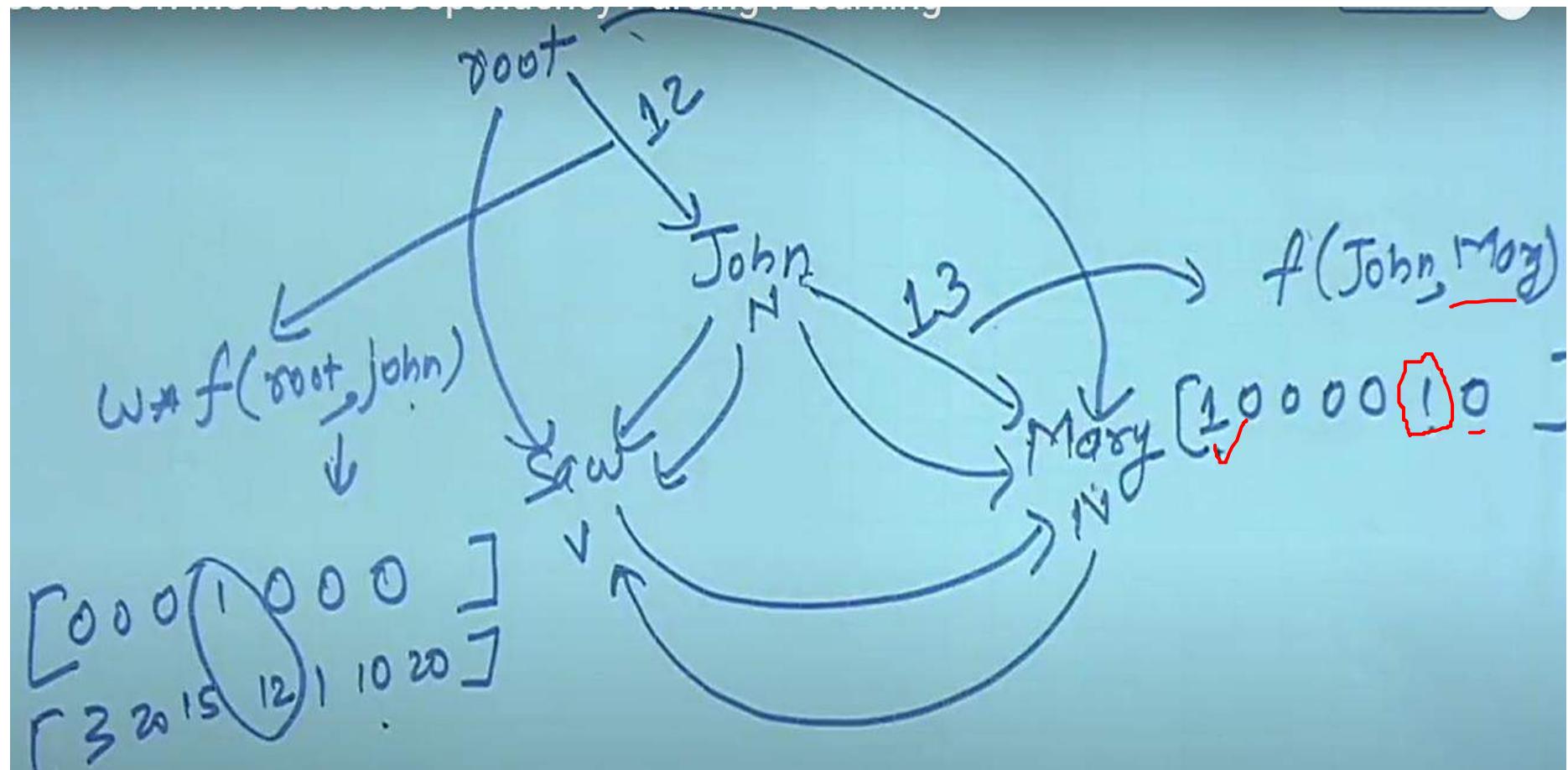
- $f_1: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_2: \text{pos}(w_i) = \text{Verb}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_3: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Verb}$
- $f_4: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_5: w_i = \text{Root}$  and  $w_j$  occurs at the end of sentence
- $f_6: w_i$  occurs before  $w_j$  in the sentence
- $f_7: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Verb}$

The feature weights before the start of the iteration are: {3, 20, 15, 12, 1, 10, 20}. Determine the weights after an iteration over this example.

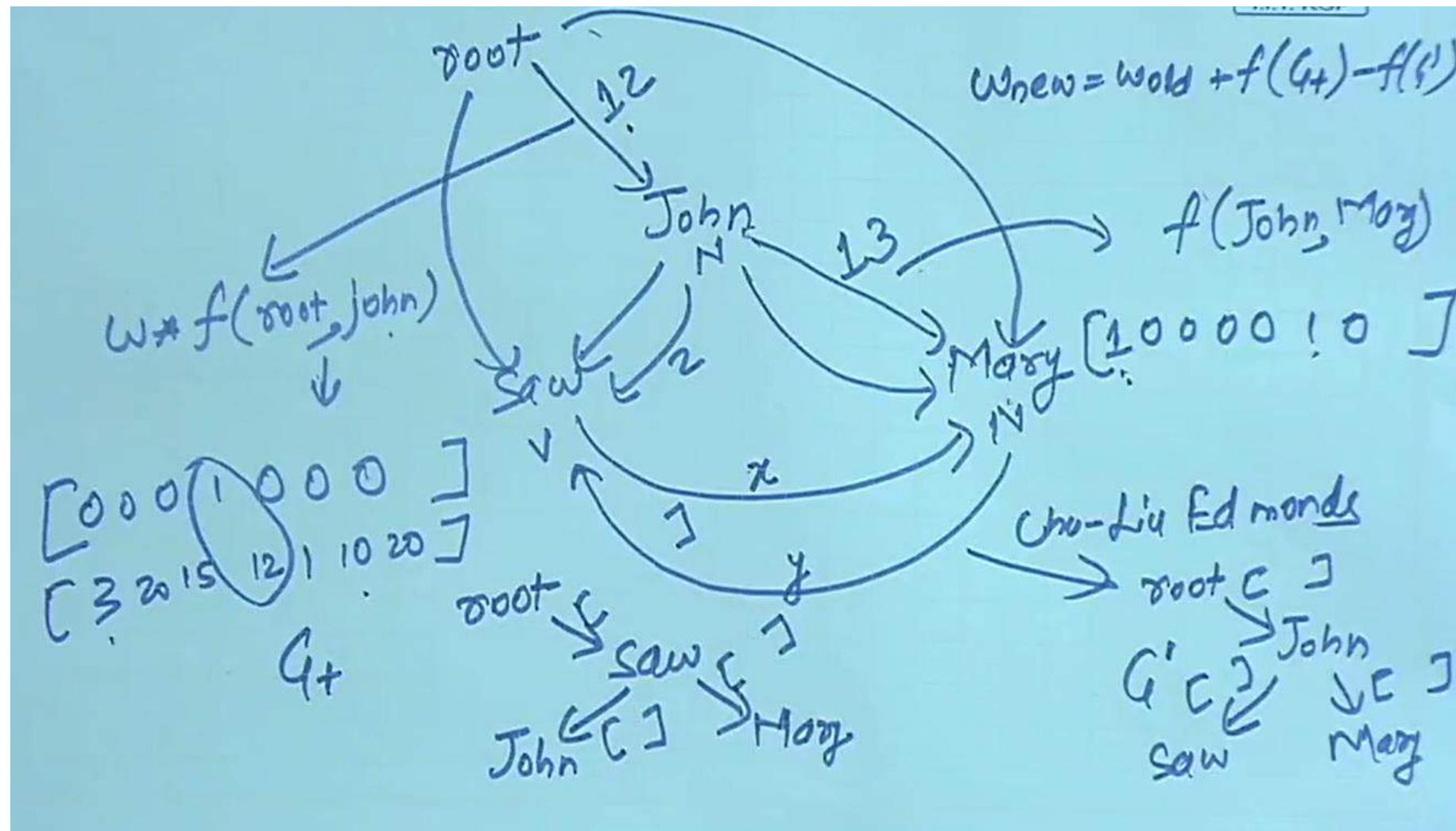
# EXAMPLE



# EXAMPLE



# EXAMPLE



# EXAMPLE

