

Syntax Analysis

Outline

- Syntactic stage of NLP
- Applications of Syntax analysis
- Ambiguity in the language
- Part-of-speech tagging
 - Penn treebank Tagset
 - why tagging is hard?
 - Rule based tagging
 - Stochastic POS tagging
 - Transformation-based Tagging
- ISSUES...Multiple tag and multiple words
- Unknown words
- Parsing and approaches

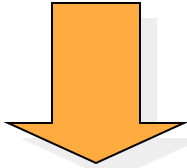
Why NL Understanding is hard?

- Natural language is extremely rich in form and structure, and **very ambiguous**.
 - How to represent meaning,
 - Which structures map to which meaning structures.
- One input can mean many different things. Ambiguity can be at different levels.
 - **Lexical (word level) ambiguity** -- different meanings of words
 - **Syntactic ambiguity** --different ways to parse the sentence
 - **Interpreting partial information** -- how to interpret pronouns
 - **Contextual information** -- context of the sentence may affect the meaning of that sentence.
- Many input can mean the same thing.
- Interaction among components of the input is not clear.

Stages of NLP

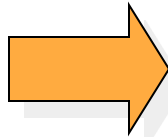
Morphological Analysis

Individual words are analyzed into their components



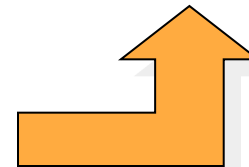
Syntactic Analysis

Linear sequences of words are transformed into structures that show how the words relate to each other



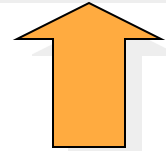
Semantic Analysis

A transformation is made from the input text to an internal representation that reflects the meaning



Pragmatic Analysis

To reinterpret what was said to what was actually meant



Discourse Analysis

Resolving references
Between sentences

The stages Involved in NLP

- **Morphology:** Concerns the way words are built up from smaller meaning bearing units.
- **Syntax:** concerns how words are put together to form correct sentences and what structural role each word has.(Ex: “the dog ate my homework”)
- **Semantics:** concerns what words mean and how these meanings combine in sentences to form sentence meanings.(plant: industrial plant/ living organism)
- **Pragmatics:** concerns how sentences are used in different situations and how it affects the interpretation of the sentence.
- **Discourse:** concerns how the immediately preceding sentences affect the interpretation of the next sentence.

Syntax Analysis

- syntax analysis or parsing is the **important phase of NLP**. The purpose of this phase is to draw exact meaning or dictionary meaning from the text.
- Syntax refers to the arrangement of words in a sentence such that they make grammatical sense.
- In NLP, syntactic analysis is used to assess how the natural language aligns with the grammatical rules.
- Computer algorithms are used to apply grammatical rules to a group of words and derive meaning from them.
- Syntactic analysis helps us understand the roles played by different words in a body of text.
- Consider “**Innocent peacefully children sleep little**” vs “**Innocent little children sleep peacefully**”

Applications of Syntax analysis

Syntactic analysis used to varying degrees in applications such as:

- **Grammar Checkers**
- **Spoken Language Understanding**
- **Question Answering systems**
- **Information Extraction**
- **Automatic Text Generation**
- **Machine Translation**

Typically, fine-grained syntactic analysis is a prerequisite for fine-grained semantic interpretation.

Ambiguity in the Language

Ex: I made her duck.

- How many **different interpretations** does this sentence have?
- What are the reasons for the ambiguity?
- The categories of knowledge of language can be thought of as ambiguity resolving components.
- How can each ambiguous piece be resolved?

Ambiguity in the Language (cont.)

- Some interpretations of : **I made her duck.**
 1. **I cooked *duck* for her.**
 2. **I cooked *duck* belonging to her.**
 3. **I created a toy duck which she owns.**
 4. **I caused her to quickly lower her head or body.**
 5. **I used magic and turned her into a *duck*.**
- duck – morphologically and syntactically ambiguous:
noun or verb.
- her – syntactically ambiguous: **dative or possessive.**
- make – semantically ambiguous: cook or create.
- make – syntactically ambiguous:
 - **Transitive – takes a direct object. => 2**
 - **Di-transitive – takes two objects. => 5**
 - **Takes a direct object and a verb. => 4**

Part Of Speech tagging (POS)

- The part of speech tagging is a process of assigning corresponding part of speech like **noun, verb, adverb, adjective, verb to each word in a sentence.**
- It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)).
- The tag in case of it is a part of speech tag, and signifies whether the word is a **noun, adjective, verb, and so on.**
- Automatic assignment of descriptors to the given tokens is called **Tagging.**
- The descriptor is called **tag.**

Book/VB that/DT flight/NN

Part-of-Speech (POS) Tagging

- Each word has a part-of-speech tag to describe its category.
- Part-of-speech tag of a word is one of major word groups (or its subgroups).
 - **open classes** -- noun, verb, adjective, adverb
 - **closed classes** -- prepositions, determiners, conjunctions, pronouns, participles
- POS Taggers try to find POS tags for the words.
- duck is a verb or noun? (morphological analyzer cannot make decision).
- A POS tagger may make that decision by looking the surrounding words.
 - Duck! (verb)
 - Duck is delicious for dinner. (noun)

Tag set for English

- Parts of speech can be divided into two broad supercategories:

Closed class types and

Open class types.

- **Closed classes** are those that have relatively fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English.
- By contrast nouns and verbs are **open classes** because new nouns and verbs are continually coined or borrowed from other languages (e.g. the new verb to fax or the borrowed noun futon).
- Open vs. Closed classes

Open: Nouns, Verbs, Adjectives, Adverbs.

**Closed: (determiners: a, an, the) (pronouns: she, he, I)
(prepositions: on, under, over, near, by)**

Tag set for English

- Open Class Words

Every known human language has nouns and verbs

Nouns: people, places, things

Classes of nouns

proper vs. common

count vs. mass

Verbs: actions and processes

Adjectives: properties, qualities

Numerals: one, two, three, third

Tag set for English

- **Closed Class Words**

Differ more from language to language than open 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, ...

Tag set for English

- Tagset

- | | |
|--------------------|-----------------------|
| 1. [N] Nouns | 6. [PP] Postpositions |
| 2. [V] Verbs | 7. [PL] Participles |
| 3. [PR] Pronouns | 8. [QT] Quantifiers |
| 4. [JJ] Adjectives | 9. [RP] Particles |
| 5. [RB] Adverbs | 10. [PU] Punctuations |

- Vary in number of tags: a dozen to over 200.
- Size of tag sets depends on **language, objectives and purpose**

Penn Treebank Tagset

- There are a small number of popular tagsets for English, many of which evolved from the **87-tag tagset used for the Brown corpus**.
- The **Penn Treebank tagset**, has been applied to the Brown corpus and a number of other corpora.

Ex: The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

- The Penn Treebank tagset was selected from the original 87-tag tagset for the Brown corpus.

Numbe	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential <i>there</i>
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNP S	Proper noun, plural
16.	PDT	Predeterminer

Number	Tag	Description
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP\$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	<i>to</i>
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP\$	Possessive wh-pronoun
36.	WRB	Wh-adverb

How do we assign POS tags to words in sentence?

- Get/V the/Det bass/N
- Time flies like an arrow.
- Time/[V,N] flies/[V,N] like/[V,Prep] an/Det arrow/N
- Time/N flies/V like/Prep an/Det arrow/N
- Fruit/N flies/N like/V a/DET banana/N
- Fruit/N flies/V like/V a/DET banana/N
- The/Det flies/N like/V a/DET banana/N
- Does/VBZ this/DT flight/NN serve/VB dinner/NN.
- Nobody/NN ever/RB takes/VBZ the/DT newspaper/NNS
she/PRP sells/VBZ

Why is Tagging Hard?

Example

1. Book/VB that/DT flight/NN
 2. Does/VBZ that/DT flight/NN serve/VB dinner/NN
 3. I was twenty-one back/RB then (adverb)
 4. A small building in the back/NN
 5. Earnings growth took a back/JJ seat.(Adjective)
- Tagging is a **type of disambiguation**
 - Book can be NN or VB
 - Can I read a book on this flight?
 - That can be a DT or complementizer(subordinating conjunction) links verb to its argument.
 - My travel agent said that there would be a meal on this flight

The Problem

- Words often have more than one word class: **this**
- **This** is a nice day = PRP
- **This** day is nice = DT
- You can go **this** far = RB

POS tag Ambiguity

- I **bank** on the **bank** on the river **bank** for my transactions .[English]

noun

verb

verb

- Mujhe **khaanna** **khaanna** hai .[Hindi]

Noun

verb

- **पुजा** देवीची **पुजा** कर. [Marathi]

Noun

verb

Ambiguity

– “**Plants**/N need light and water.”

– “Each one **plant**/V one.”

– “Flies like a flower”

- *Flies*: noun or verb?

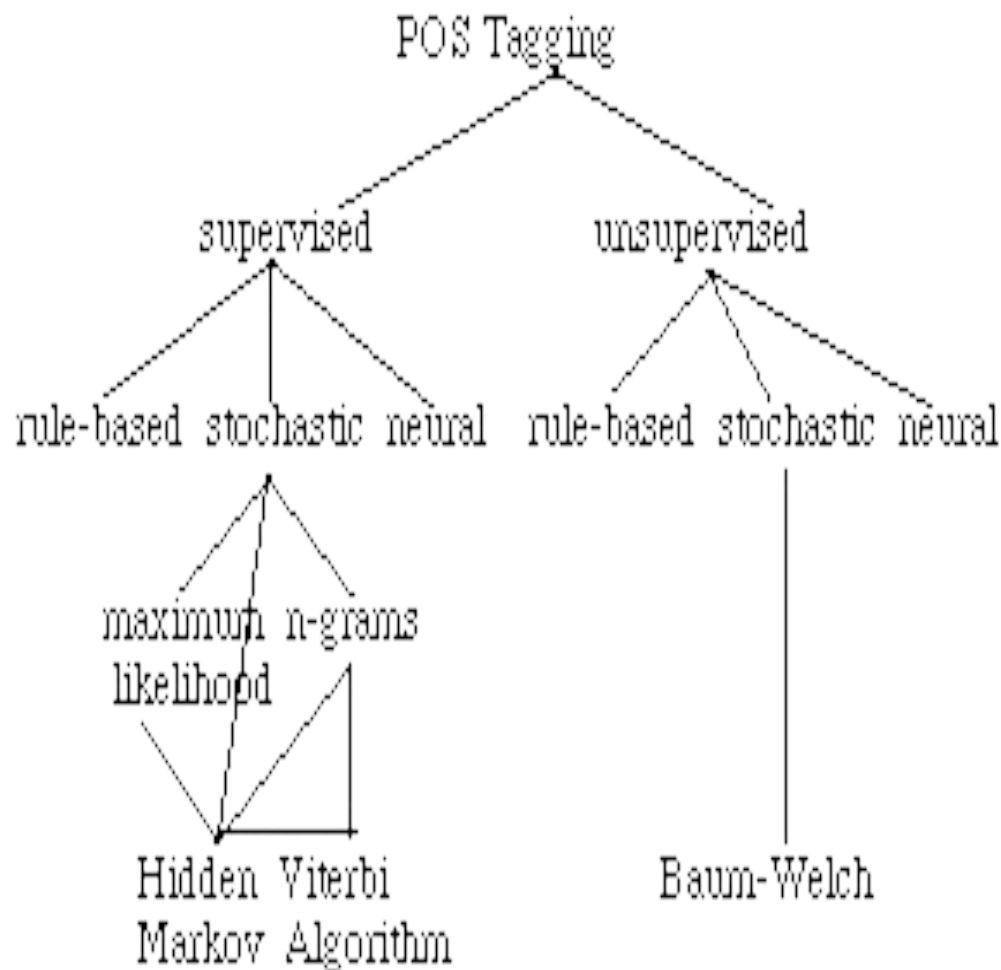
- *like*: preposition, adverb, conjunction, noun, or verb?

- *a*: article, noun, or preposition?

- *flower*: noun or verb?

Classification of POS tagging approaches

- Rule based POS tagging
- Stochastic POS tagging
- Transformation based Tagging



Rule based POS tagging

- One of the **oldest techniques** of tagging
- Rule-based taggers use **dictionary or lexicon** for getting possible tags for tagging each word.
- If the word has more than one possible tag, then rule-based taggers **use hand-written rules** to identify the correct tag.
Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words.
- For example, suppose if the preceding word of a word is article then word must be a noun.

Rule based POS tagging

- Basic Idea:

- Assign all possible tags to words
- Remove tags according to set of rules of type:

if word+1 is an adj, adv, or quantifier and the following is a sentence boundary

And

word-1 is not a verb like “consider”

then

eliminate non-adv else eliminate adv.

- Typically more than 1000 hand-written rules, but may be machine-learned.

Rule based POS tagging : Stage 1

First Stage: In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.

FOR each word

Get all possible parts of speech using a morphological analysis algorithm

Example

		NN				
		RB				
	VBN		JJ		VB	
PRP	VBD		TO	VB		DT NN
She		promised	to	back	the	bill

Rule based POS tagging : Stage 2

In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

Apply rules to remove possibilities

Example Rule:

IF VBD is an option and VBN|VBD follows “<start>PRP”
THEN Eliminate VBN

		NN				
		RB				
	VBN		JJ		VB	
PRP	VBD		TO	VB		DT NN
She	promised		to	back		the bill

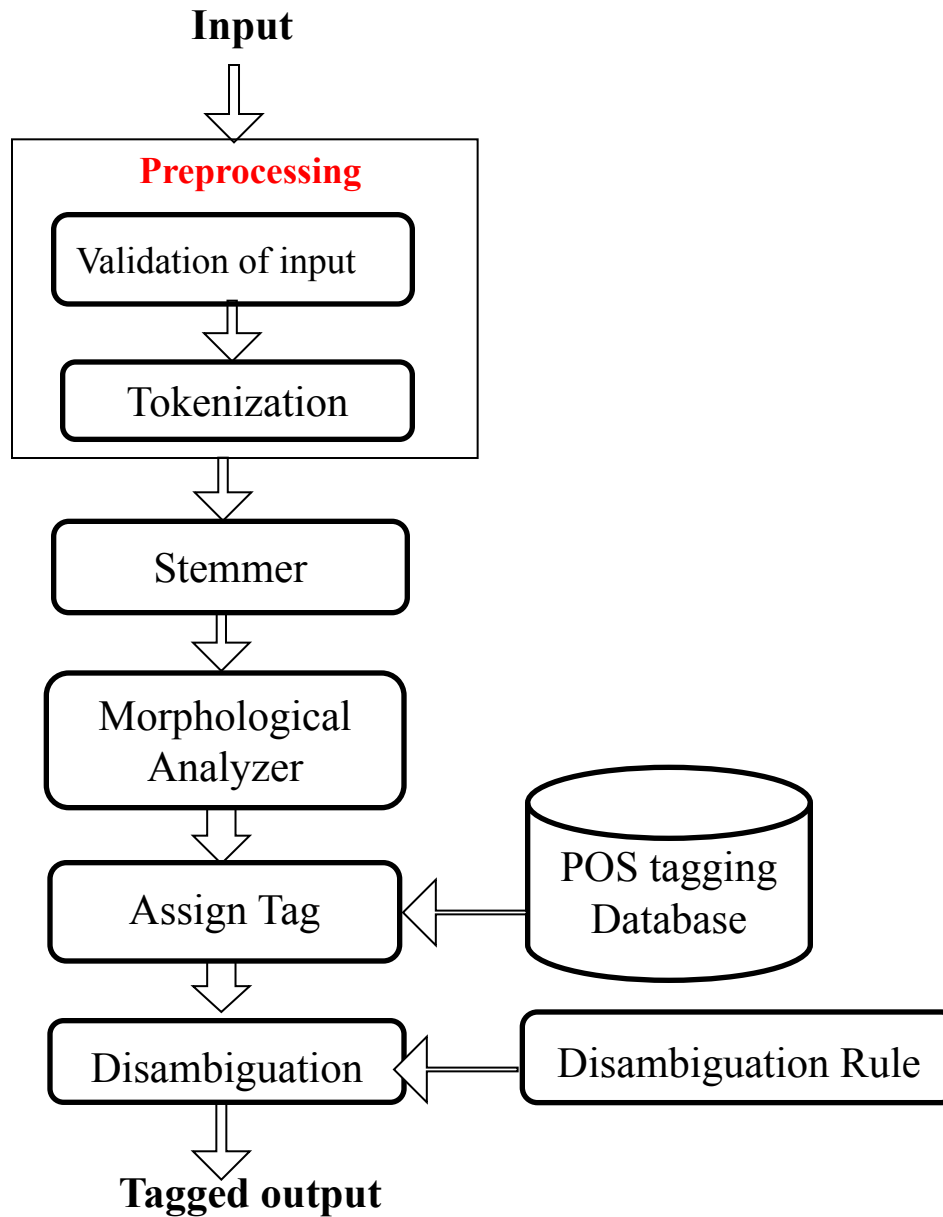
Properties of Rule-Based POS Tagging

Rule-based POS taggers possess the following properties –

- These taggers are knowledge-driven taggers.
- The rules in Rule-based POS tagging are built **manually**.
- The information is coded in the form of rules.
- We have some limited number of rules approximately around 1000.
- Smoothing and language modeling is defined explicitly in rule-based taggers.

Rule based PoS tagger for Marathi Language

- **Rule based POS tagger**, where each word will be assigned with corresponding tag and if word is assigned with multiple tags then such ambiguity will be removed by Word Sense Disambiguation (WSD) rules.
- **Input:** Accepts a sentence or document in MARATHI .
- **Output: PoS tagged Document .**



VALIDATION OF INPUT

Input: Input document

Output: Devanagari script

Algorithm

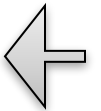
1. Use the character set as UTF-8
2. Scan the input document.
3. Compare each character from scanned input document with UTF-8.
4. If character is present in the UTF-8, then it is valid to Devanagari script otherwise not.
5. Ignore all the invalid Devanagari script characters.
6. Repeat step 3 till all characters from input script document get verified.
7. Store all the valid Devanagari character, words in file to process further.

Example:

Consider the input sentence as:

Input: मराठी भाषा हे महाराष्ट्राचे वैभव आहे. It is given in history of Marathi Language

Output: मराठी भाषा हे महाराष्ट्राचे वैभव आहे.



TOKENIZATION

- This tokenization task is possible by searching spaces between the words.
- The root words used for information retrieval applications

Input: Marathi language sentence/ document

Output: list of tokens

Consider the input sentence as:

मराठी भाषा हे महाराष्ट्राचे वैभव आहे.

- The tokenization for the given sentence is:
 - मराठी
 - भाषा
 - हे
 - महाराष्ट्राचे
 - वैभव
 - आहे
 - .

STEMMER

- ❑ Removes all possible suffixes
- ❑ Reduces the word to its stem.
- ❑ Uses suffix list to remove suffixes from words.

❑ **Problem of Stemming:** Stem contains inflections which cannot be removed using simple stemming operation.

Examples of Suffixes:

च्या, न्या, ज्या ज्या ख्या च चा ची चे ही नी ने ला, कडे, कडून, देखील, वर, खाली, पर्यंत, मागे मागेच, समोर, सुद्धा, साठी, मध्ये मध्येच मध्येही मध्येपण मधेसुद्धा, विरुद्ध

Input: List of words

Output: Stem of words

Example:

Input: मराठी भाषा हे महाराष्ट्राचे वैभव आहे.

Output: मराठी भाषा हे महाराष्ट्रा वैभव आहे.

MORPHOLOGICAL ANALYZER

- To recognize the inner structure of the word and to produce Root words for a given input document .
- Stemmed words are analyzed to check whether they are inflected or not.

Input: i) Stemmed token (with inflection)

ii) List of inflection stripping rule :

डा, डी, डे, डो, डौ, ड्या □ ड

ता, ति, ती, तु, तू, ते, तै, तो, तौ, तं, तः, त्या □ त

ना, नि, नी, नु, नू, ने, नै, नो, नौ, नं, नां, नः □ न

Output: Root words

Example :

In the word “महाराष्ट्राचे ” the stem form is “महाराष्ट्रा ” (maharashtra) whereas the root form is “महाराष्ट्र”.

Input: मराठी भाषा हे महाराष्ट्रा वैभव आहे.

Output: मराठी भाषा हे महाराष्ट्र वैभव आहे.

TAG GENERATOR

□ a well-chosen tagset is also important to represents parts of speech

Universal tagset for all ILs

- | | |
|---------------------|------------------------|
| 1. [N] Noun | 7. [PP] Postpositions |
| 2. [V] Verbs | 8. [DM] Demonstratives |
| 3. [PR] Pronoun | 9. [QT] Quantifier |
| 4. [JJ] Adjectives | 10. [RP] Particle |
| 5. [RB] Adverb | 11. [PU] Punctuation |
| 6. [PL] Participles | 12. [RD] Residual |

TAGSET for Marathi

No	Name	Tag	Description	Example
1	NOUN	NN	Common Nouns	मुलगा, साखर, मंडळी, चांगुलपणा
		NNP	Proper Nouns	मोहन, राम, सुरेश
		ABN	Abstract noun	गर्व, कौशल्य, क्रोध, चपळाई, गोडवा
2	PRONOUN	PPN	Personal pronoun	मी, आम्ही, तुम्ही
		PPS	Possessive pronoun	माझा, माझी, तुझा, तुझी, त्याचा
		PDM	Demonstrative pronoun	तो, ती, ते, हा, ही
		PRF	Reflexive pronoun	आपण, आम्ही, तुम्ही, तुम्हाला
		PRC	Reciprocal pronoun	एकमेकांचा, एकमेकाला, आपल्याला
3	ADJECTIVE	JJ	Modifier of Noun	उत्साही, श्रेष्ठ, बळवान
4	VERB	VM	Verb Main	बसणे, दिसणे, लिहिणे, पडला, झोपला
		VAUX	Verb Auxiliary	नाही, नको, करणे, हवे, नये

5	ADVERB	RB	(Modifier of Verb)	आता, काल, कधी, नेहमी, लवकर, हळूहळू
6	CONJUNCTION	CC	Coordinating and Subordinating	आणि, पण, जर, तर
7	POSTPOSITION	PSP	Postposition	आणि, वर, कडे, जवळ
8	INTERJECTION	INJ	Interjection	आहा, छान, अगो, हाय
9	NUMERAL	NUM	Number	१, २, ३, ४
		NUMCD	Cardinal Numeral	एक, दोन, तीन
		NUMO	Ordinal Numeral	पहिला, दुसरा, तिसरा
10	RESIDUAL	RDS	Symbol residual	\$, &, *, (,)
		RD_PUNC	Punctuation	? , ; : !

11	REDUPLICATION	RDP	Reduplications	जवळ-जवळ
12	NEGATIVE	NEG	Negative	नाही,नको
13	DETERMINER	QF	Quantifiers	किती,पुष्कळ,खूप,भरपूर,बरेच
14	QUESTION WORDS	WQ	Question Words	काय, कधी, कु ठे
15	INTENSIFIER	INTF	Intensifier	खूप, फार,बराच,अतिशय
16	PARTICLES	RP	Particles	तर,ओहो
17	PHRASE	PHR	Phrase	नमस्कार,अभिनंदन,खेद आहे
18	ECHO	ECH	Echo Word	जेवणबिवण,डोकेबिके,पडणबिडण
19	QUATATIVE	UT	Quatative word	म्हणजे

Input: i) Root word
 ii) Corpus for POS tagging

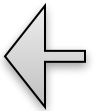
Output: Tagged word

Algorithm:

- 1) Take root word from morphological analyzer
- 2) Store all root words into array WORD
- 4) Select each word one by one from array WORD and Search and compare selected word from corpus for pos tagging.
- 5) If word is found one or more times, then store associated tag or tags of word into array TAGS
- 6) Else display “the word is not found” add this new word with corresponding tag into lexicon with linguistic rules for new word.
- 7) If one tag is stored in array TAGS, then display word with associated tag as an output.
- 8) Else apply rule to select most appropriate tag for word.
- 9) Display word with associated tag as an output.
10. End.

Example

Input	मराठी	भाषा	हे	महाराष्ट्राचे	वैभव	आहे	.
Output	मराठी	भाषा	हे	महाराष्ट्र	वैभव	आहे	.
Tag	NNP	NN	PDM	NNP	NNP/JJ	VAUX	RD_PUNC



DISAMBIGUATION

- The "correct" and the "incorrect" assignment of a tag in particular contexts.
- Disambiguation is also based on contextual information or word/tag sequences.
- The ambiguity which is identified in the POS tagging module is resolved using the **grammar rules**.

WSD Rules for Marathi


Sr. No.	Ambiguity Between	Previous word	Next word	Possible Tag
1	Adjective and Adverb	Noun	Noun or Adjective	Adjective
			Verb	Adverb
2	Noun and Adjective	Not Noun	Noun	Adjective
		Noun	Verb	
3	Noun and Verb	Noun	Verb	Adverb
4	Auxiliary Verb and Noun		Vaux main	Verb
5	Verb and Auxiliary Verb	Vaux main	Verb	Adverb
6	Conjunction Noun Postposition	Noun, Pronoun,	NULL	Postposition
		Verb		Conjunction
		< Tag not matched>	< Tag not matched>	Noun (Default)
7	Noun, Adjective, Verb	NULL	Noun, Adjective, Verb	Noun
			< Tag not matched>	Verb (Default)

Example

Input	मराठी	भाषा	हे	महाराष्ट्राचे	वैभव	आहे	.
Output	मराठी	भाषा	हे	महाराष्ट्र	वैभव	आहे	.
Tag	NNP	NN	PDM	NNP	JJ	VAUX	RD_PUNC

Ambiguity removed by using **WSD rule**

Final OUTCOME of MATRATHI POS tagger

 मराठी शब्द जाती

Marathi POS Tagger

*** मराठी शब्द जाती ***

INPUT

रामायण हे वाल्मीकी ऋषी रच एक महाकाव्य आहे . भारत रामायण खूप प्रसिद्ध आहे . रामायण हिंदू धर्मीय एक पवित्र ग्रंथ आहे . श्रीराम हे विष्णू रूप हो . भारत गुरु स्थान खूप श्रेष्ठ आहे . प्रत्येक आयुष्य गुरु ज्ञान खूप उपयोग ठर .

OUTPUT

रामायण::NNP हे::PDM वाल्मीकी::NNP ऋषी::NN रच::VM एक::NUMCD महाकाव्य::NN आहे::VAUX ...RD_PUNC
भारत::NNP रामायण::NNP खूप::INTF प्रसिद्ध::JJ आहे::VAUX ...RD_PUNC रामायण::NNP हिंदू::NN धर्मीय::NN एक::
NUMCD पवित्र::JJ ग्रंथ::NN आहे::VAUX ...RD_PUNC श्रीराम::NNP हे::PDM विष्णू::NNP रूप::NN हो::INJ ...
RD_PUNC भारत::NNP गुरु::NN स्थान::NN खूप::INTF श्रेष्ठ::JJ आहे::VAUX ...RD_PUNC प्रत्येक::QF आयुष्य::NN गुरु::
NN ज्ञान::NN खूप::QF उपयोग::NN ठर::VM ...RD_PUNC

Clear Screen

Brows File

Validation

Tokenization

Stemmer

Morphology

POS Tag

Disambiguation

ENGTWOL Rule-Based Tagger

- A Two-stage architecture
- Use lexicon FST (dictionary) to tag each word with all possible POS.
- Apply hand-written rules to eliminate tags.
- The rules eliminate tags that are inconsistent with the context, and should reduce the list of POS tags to a single POS per word.

Simple lexical entries in the ENGTWOL lexicon

Word	POS	Additional POS features
smaller	ADJ	COMPARATIVE
entire	ADJ	ABSOLUTE ATTRIBUTIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog's	N	GENITIVE SG
furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	IMPERATIVE VFIN
show	V	PRESENT -SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

ENGTWOL Rule-Based Tagger example.. a simple rule-based tagger based on the constraint grammar architecture

Pavlov	had shown that salivation ...	An example for
Pavlov	PAVLOV N NOM SG PROPER	The ENGTWOL tagger
had	HAVE V PAST VFIN SVO (preterit)	
	HAVE PCP2 SVO (past participle)	
shown	SHOW PCP2 SVOO SVO SV	
that	ADV	
	PRON DEM SG	A set of 1,100 constraints
	DET CENTRAL DEM SG	can be applied to the input
	CS (complementizer)	sentence
salivation	N NOM SG	

Example:

It isn't **that** odd!
ADV A

I consider **that** odd.
Compliment NUM

Second Stage: Apply constraints.
Constraints used in negative way.
Example: Adverbial “that” rule

Given input: “that”

If

(+1 A/ADV/QUANT)

(+2 SENT-LIM)

(NOT -1 SVOC/A)

Then eliminate non-ADV tags

Else eliminate ADV

Stochastic Tagging

- The use of probabilities in tags is quite old. Stochastic taggers use probabilistic and statistical information to assign tags to words.
 - The model that includes frequency or probability (statistics) can be called **stochastic**.
 - These taggers might use ‘**tag sequence probabilities**’, ‘**word frequency measurements**’ or a combination of both.
 - The tag encountered most frequently in the training set is the one assigned to an ambiguous instance of that word (word frequency measurements).
- Simple Method: Choose most frequent tag in training text for each word!

Working of tagger

- Collect a large annotated corpus of text and divide it into training and testing sets.
- Train a statistical model on the training data, using techniques such as maximum likelihood estimation or hidden Markov models.
- Use the trained model to predict the POS tags of the words in the testing data.
- Evaluate the performance of the model by comparing the predicted tags to the true tags in the testing data and calculating metrics such as precision and recall.
- Fine-tune the model and repeat the process until the desired level of accuracy is achieved.
- Use the trained model to perform POS tagging on new, unseen text.

Stochastic Tagging

- The best tag for a given word is determined by the probability that it occurs with the n previous tags (tag sequence probabilities)
- Resolves the ambiguity by computing the probability of a given word (or the tag).
- The problem with this approach is that it can come up with sequences of tags for sentences that are not acceptable according to the grammar rules of a language.

Stochastic Tagging

stochastic tagger applies the following approaches for POS tagging –

1. Word Frequency Approach

- In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag.
- We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word.
- The main issue with this approach is that it may yield inadmissible sequence of tags.

2. Tag Sequence Probabilities

- Here the tagger calculates the probability of a given sequence of tags occurring.
- It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

Stochastic Tagging

- **Unigram** approach assigns each word to its most common tag and consider one word at a time. $P(t_i/w_i) = \text{freq}(w_i/t_i) / \text{freq}(w_i)$
- Here Probability of tag given word is computed by frequency count of word given tag divided by frequency count of that particular word
- **Bigram** approach is based on preceding tag i.e. it take two tags: the preceding tag and current tag into account. $P(t_i/w_i) = P(w_i/t_i) \cdot P(t_i/t_{i-1})$
- Here $P(w_i/t_i)$ is the probability of current word given current tag and $P(t_i/t_{i-1})$ is the probability of a current tag given the previous tag
- **Tigram** is based on previous two tags. $P(t_i/w_i) = P(w_i/t_i) \cdot P(t_i/t_{i-2}, t_{i-1})$, Where t_i denotes tag sequence and w_i denote word sequence.
- $P(w_i/t_i)$ is the probability of current word given current tag.
- Here, $P(t_i|t_{i-2}, t_{i-1})$ is the probability of a current tag given the previous two tags.

Properties of Stochastic POST Tagging

Stochastic POS taggers possess the following properties –

- This POS tagging is based on the **probability of tag occurring.**
- It requires **training corpus...vast stored information**
- There would be no probability for the words that **do not exist in the corpus.**
- It uses different testing corpus (other than training corpus).
- It is the simplest POS tagging because it chooses **most frequent tags** associated with a word in training corpus

Stochastic POST Tagging :HMM Tagger

Intuition: Pick the most likely tag based on context

Maximize the formula using a HMM : $P(\text{word}|\text{tag}) \times P(\text{tag}|\text{previous } n \text{ tags})$

Observe: $W = w_1, w_2, \dots, w_n$

Hidden: $T = t_1, t_2, \dots, t_n$

Goal: Find POS tags that generate a sequence of words, i.e., look for most probable sequence of tags T underlying the observed words W

We cannot determine the exact sequence of tags that generated and calculate using $t = \text{argmax } P(w, t)$ and it is based on the Markovian assumption that the current tag depends only on the previous n tags.

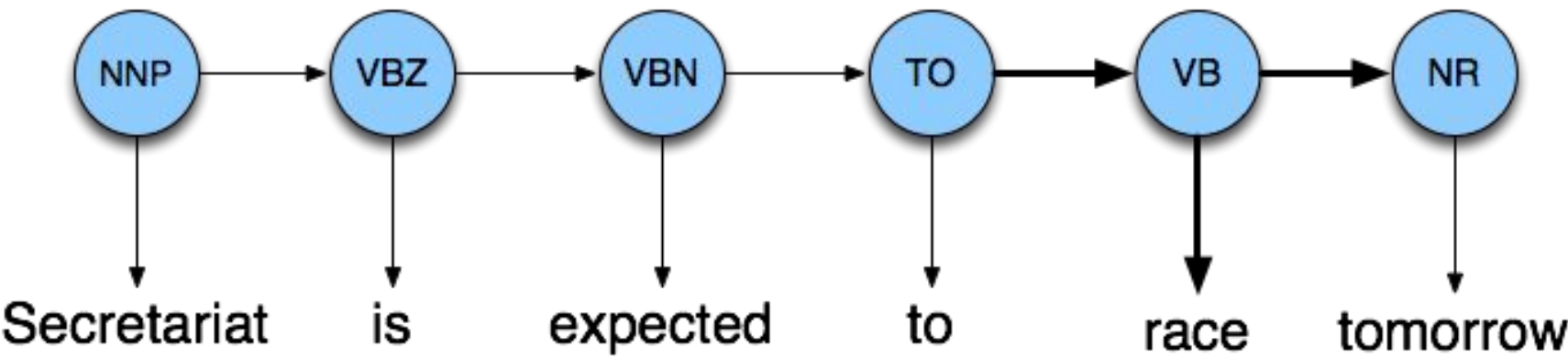
Use transition probability(i.e. forward tag and backward tags).

$$P(t_i/w_i) = P(t_i/t_{i-1}) \cdot P(t_{i+1}/t_i) \cdot P(w_i/t_i)$$

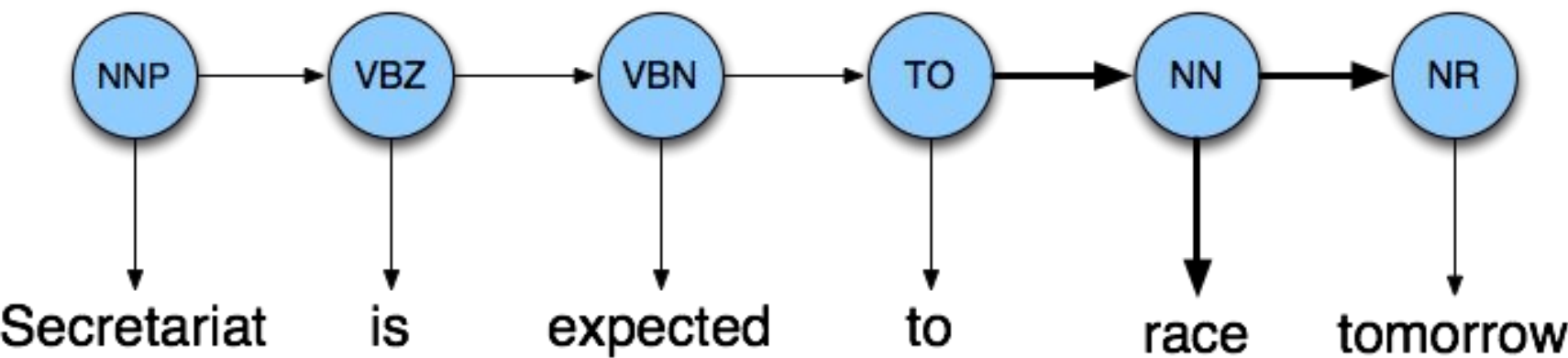
$P(t_i/t_{i-1})$ is the probability of current tag given previous tag

Disambiguity race

(a)



(b)



Secretariat/NNP is/VBZ expected/VBN **to/TO race/VB** tomorrow/NN

Example

- Secretariat/NNP is/VBZ expected/VBN **to/TO race/VB** tomorrow/NN
 - to/TO race/???
- People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN **the/DT race/NN** for/IN outer/JJ space/NN
 - the/DT race/???

tag sequence probability

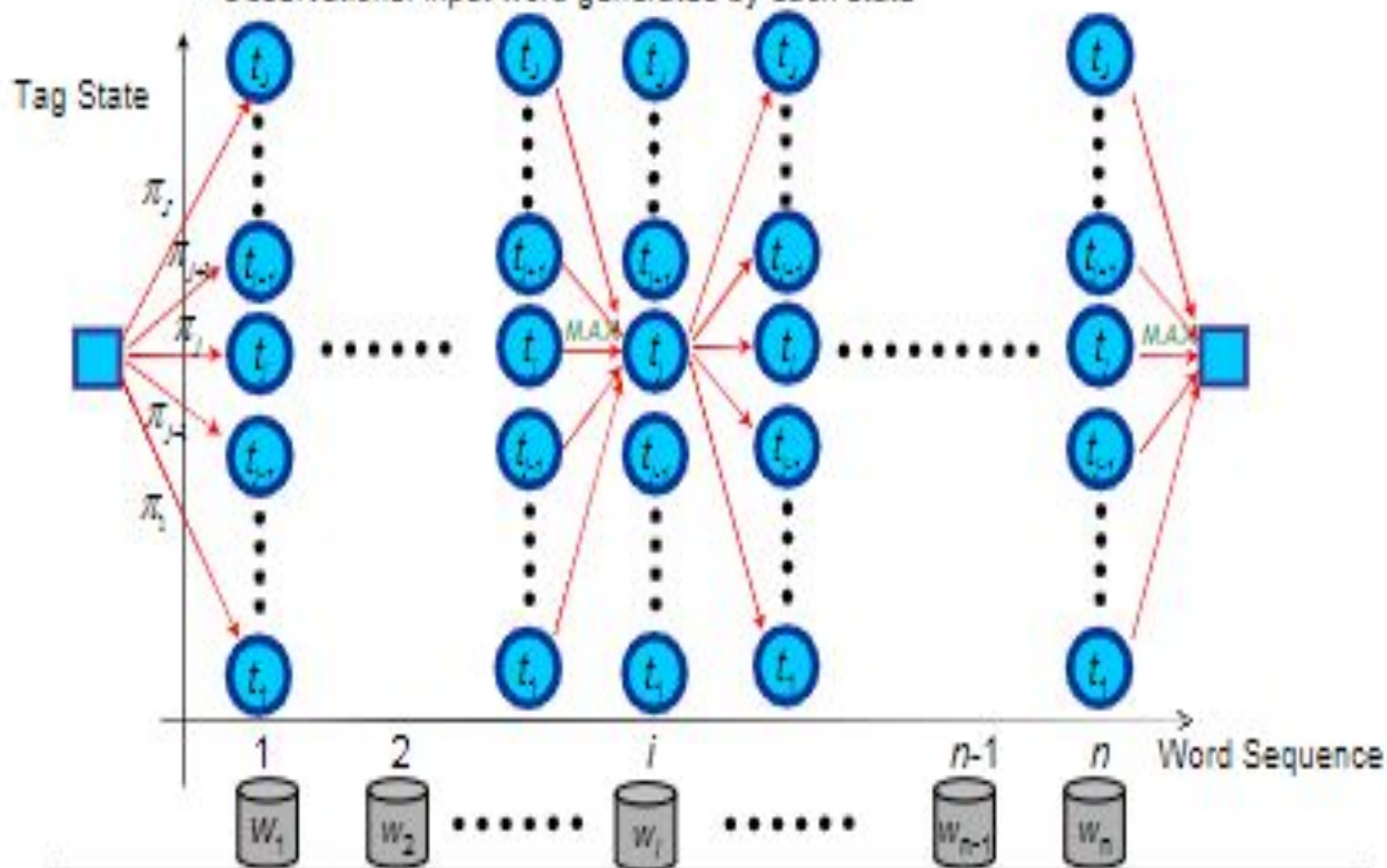
- For each word w_i , $t_i = \operatorname{argmax}_t P(t|t_{i-1})P(w_i|t)$
 - $\max(P(VB|TO) P(\text{race}|VB),$
 $P(NN|TO) P(\text{race}|NN))$

word/lexical likelihood

- From the Brown corpus
 - $P(NN|TO) = .021$ $P(\text{race}|NN) = .00041$
 - $P(VB|TO) = .34$ $P(\text{race}|VB) = .00003$
- So
 - $P(NN|TO) P(\text{race}|NN) = .021 \times .00041 = .000007$
 - $P(VB|TO) P(\text{race}|VB) = .34 \times .00003 = .00001$

- The Viterbi algorithm for the bigram-HMM tagger

- States: distinct tags
- Observations: input word generated by each state



- The Viterbi algorithm for the bigram-HMM tagger

$$1. \text{ Initialization } \delta_1(j) = \pi_j P(w_1 | t_j), \quad 1 \leq j \leq J, \quad \pi_j = P(t_j)$$

$$2. \text{ Induction } \delta_i(j) = \left[\max_k \delta_{i-1}(k) P(t_j | t_k) \right] P(w_i | t_j), \quad 2 \leq i \leq n, \quad 1 \leq j \leq J$$

$$\psi_i(j) = \operatorname{argmax}_{1 \leq k \leq J} [\delta_{i-1}(k) P(t_j | t_k)]$$

$$3. \text{ Termination } X_n = \operatorname{argmax}_{1 \leq j \leq J} \delta_n(j)$$

for $i := n-1$ to 1 step -1 do

$$X_i = \psi_i(X_{i+1})$$

end

Two assumptions of HMM Taggers

decoding

For any model, such as an HMM, that contains hidden variables, the task of determining the hidden variables sequence corresponding to the sequence of observations is called **decoding**. More formally,

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 $Q = q_1 q_2 q_3 \dots q_T$.

For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence t_1^n that is most probable given the observation sequence of n words w_1^n :

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) \quad (8.13)$$

The way we'll do this in the HMM is to use Bayes' rule to instead compute:

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} \quad (8.14)$$

Furthermore, we simplify Eq. 8.14 by dropping the denominator $P(w_1^n)$:

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n | t_1^n) P(t_1^n) \quad (8.15)$$

HMM taggers make two further simplifying assumptions. The first is that the probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i | t_i) \quad (8.16)$$

The second assumption, the **bigram** assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence;

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1}) \quad (8.17)$$

Viterbi Algorithm or A* Decoder

Resembles dynamic Programming minimum distance algorithm

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

create a path probability matrix viterbi[ $N, T$ ]
for each state  $s$  from 1 to  $N$  do                                ; initialization step
     $viterbi[s, 1] \leftarrow \pi_s * b_s(o_1)$ 
     $backpointer[s, 1] \leftarrow 0$ 
for each time step  $t$  from 2 to  $T$  do                            ; recursion step
    for each state  $s$  from 1 to  $N$  do
         $viterbi[s, t] \leftarrow \max_{s'=1}^N viterbi[s', t-1] * a_{s', s} * b_s(o_t)$ 
         $backpointer[s, t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s', t-1] * a_{s', s} * b_s(o_t)$ 

 $bestpathprob \leftarrow \max_{s=1}^N viterbi[s, T]$                         ; termination step
 $bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T]$             ; termination step
 $bestpath \leftarrow$  the path starting at state  $bestpathpointer$ , that follows  $backpointer[]$  to states back in time
return bestpath, bestpathprob
```

Figure 8.5 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A, B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

The Viterbi algorithm first sets up a probability matrix or lattice, with one column for each observation o_t and one row for each state in the state graph. Each column thus has a cell for each state q_i in the single combined automaton

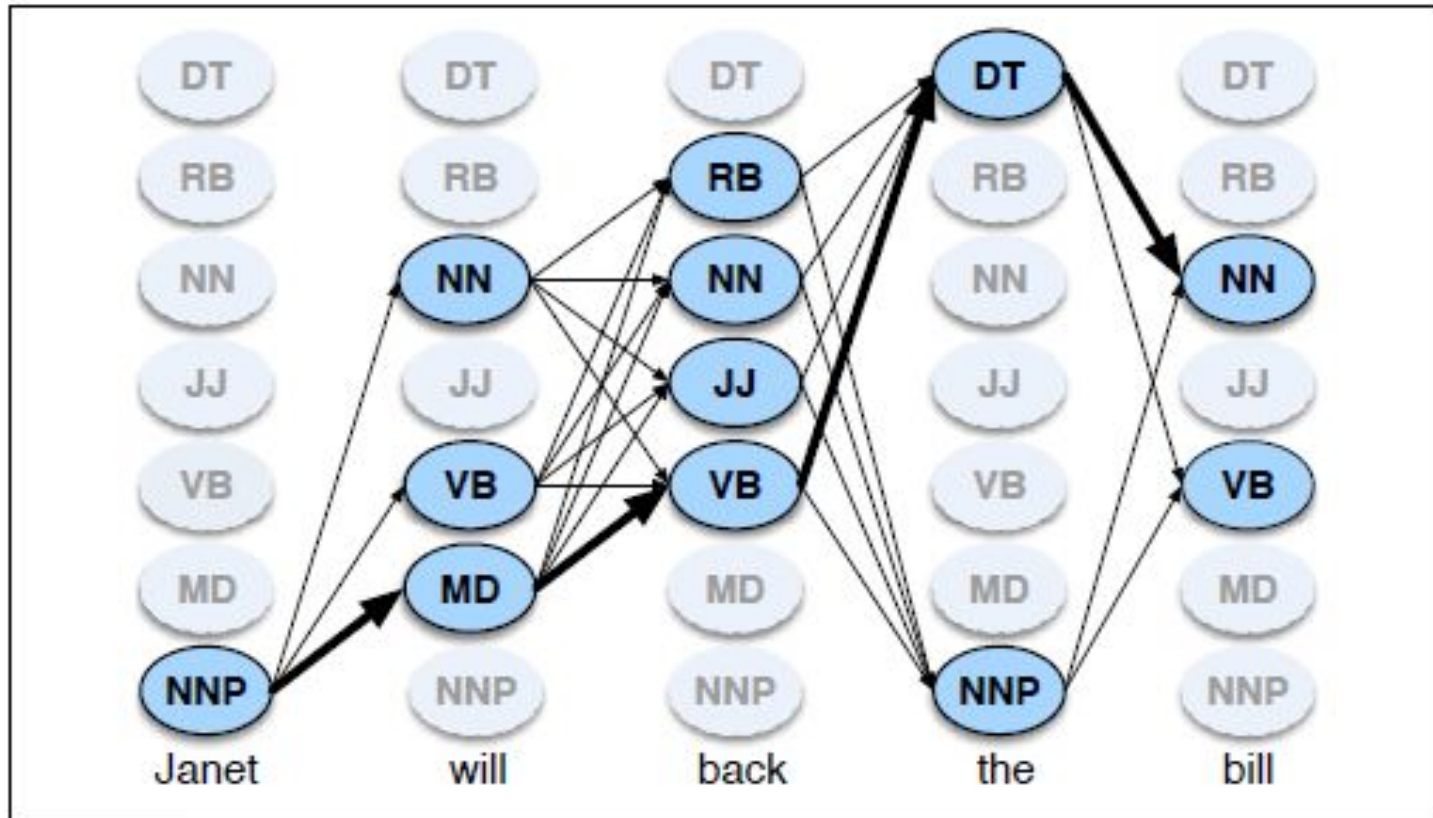


Figure 8.6 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

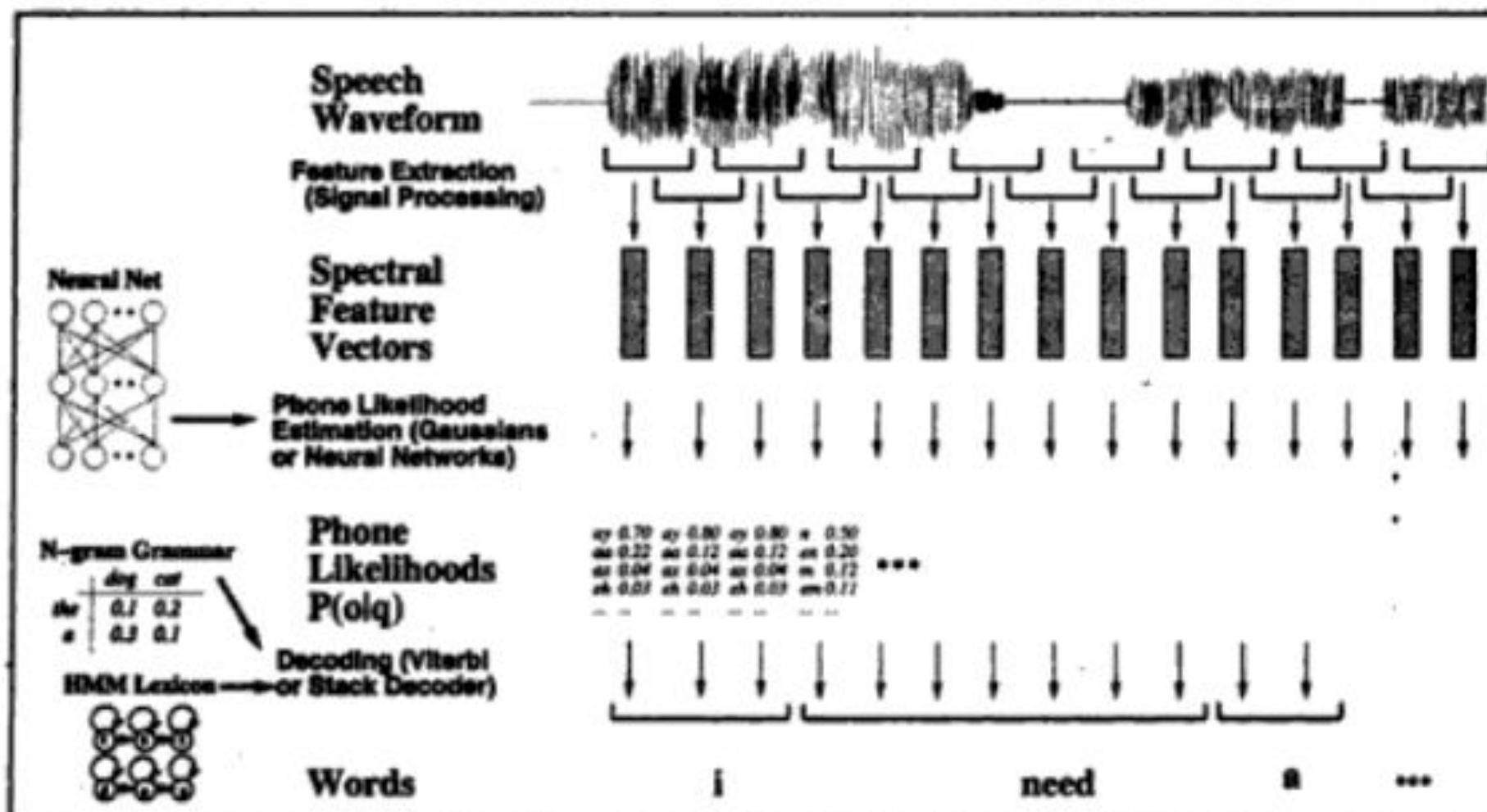
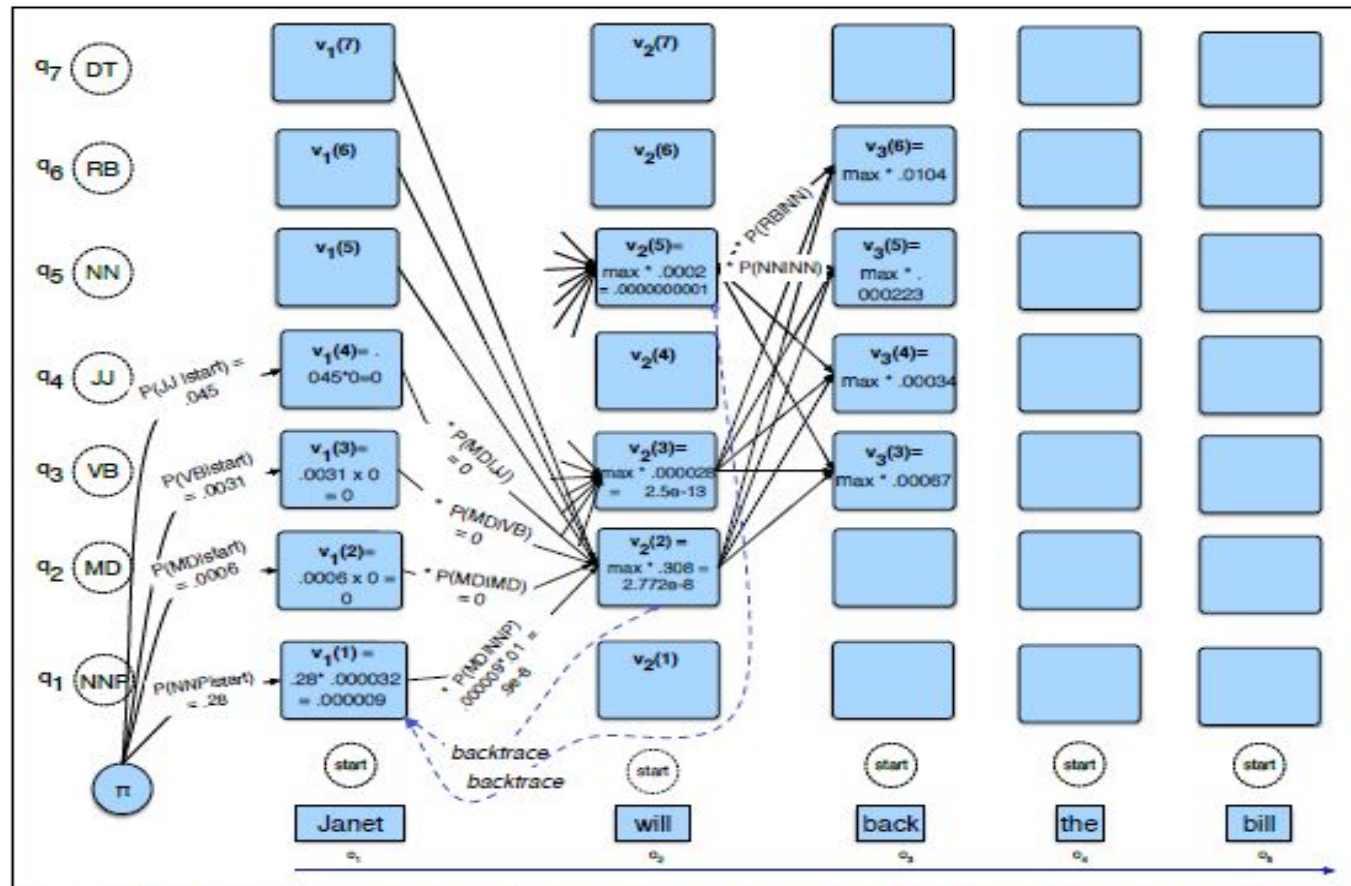


Figure 7.2 Schematic architecture for a (simplified) speech recognizer.

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2})$$

Extending the HMM Algorithm to Trigrams



also sets up probability matrix or lattice with one column per observation and one row for each state in the state graph.

Figure 8.9 The first few entries in the individual state columns for the Viterbi algorithm. Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path. We have only filled out columns 1 and 2; to avoid clutter most cells with value 0 are left empty. The rest is left as an exercise for the reader. After the cells are filled in, backtracing from the *end* state, we should be able to reconstruct the correct state sequence NNP MD VB DT NN.

Janet/NNP will/MD back/VB the/DT bill/NN

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.7 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.8 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

Unseen Trigram

but weaker estimators. For example, if we've never seen the tag sequence PRP VB TO, and so can't compute $P(\text{TO}|\text{PRP},\text{VB})$ from this frequency, we still could rely on the bigram probability $P(\text{TO}|\text{VB})$, or even the unigram probability $P(\text{TO})$. The maximum likelihood estimation of each of these probabilities can be computed from a corpus with the following counts:

$$\text{Trigrams } \hat{P}(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} \quad (8.26)$$

$$\text{Bigrams } \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \quad (8.27)$$

$$\text{Unigrams } \hat{P}(t_i) = \frac{C(t_i)}{N} \quad (8.28)$$

The standard way to combine these three estimators to estimate the trigram probability $P(t_i|t_{i-1},t_{i-2})$ is via linear interpolation. We estimate the probability $P(t_i|t_{i-1},t_{i-2})$ by a weighted sum of the unigram, bigram, and trigram probabilities:

$$P(t_i|t_{i-1},t_{i-2}) = \lambda_3 \hat{P}(t_i|t_{i-1},t_{i-2}) + \lambda_2 \hat{P}(t_i|t_{i-1}) + \lambda_1 \hat{P}(t_i) \quad (8.29)$$

We require $\lambda_1 + \lambda_2 + \lambda_3 = 1$, ensuring that the resulting P is a probability distribution. The λ s are set by **deleted interpolation** (Jelinek and Mercer, 1980): we successively delete each trigram from the training corpus and choose the λ s so as to maximize the likelihood of the rest of the corpus. The deletion helps to set the λ s

Beam Search

Beam Search is a heuristic search algorithm that explores a graph by expanding the most promising nodes (in terms of a scoring function) up to a fixed number, known as the beam width. Unlike exhaustive search methods, Beam Search prunes the less likely paths at each step, retaining only the top k paths (where k is the beam width). This makes it a trade-off between efficiency and optimality.

When to Use Beam Search

- Long Sequences: When the sequence length is very long, making exact inference computationally challenging.
- Large State Spaces: When the state space (in HMMs) or label space (in CRFs) is large.
- Resource Constraints: When there are limitations on computational resources (e.g., time or memory).

Beam search Decoding

In beam search, instead of keeping the entire column of states at each time point t , we just keep **the best few hypothesis** at that point. At time t this requires computing the Viterbi score for each of the N cells, sorting the scores, and keeping only the best-scoring states. The rest are pruned out and not continued forward to time $t + 1$.

```
function DELETED-INTERPOLATION(corpus) returns  $\lambda_1, \lambda_2, \lambda_3$ 
     $\lambda_1, \lambda_2, \lambda_3 \leftarrow 0$ 
    foreach trigram  $t_1, t_2, t_3$  with  $C(t_1, t_2, t_3) > 0$ 
        depending on the maximum of the following three values
            case  $\frac{C(t_1, t_2, t_3) - 1}{C(t_1, t_2) - 1}$ : increment  $\lambda_3$  by  $C(t_1, t_2, t_3)$ 
            case  $\frac{C(t_2, t_3) - 1}{C(t_2) - 1}$ : increment  $\lambda_2$  by  $C(t_1, t_2, t_3)$ 
            case  $\frac{C(t_3) - 1}{N - 1}$ : increment  $\lambda_1$  by  $C(t_1, t_2, t_3)$ 
        end
    end
    normalize  $\lambda_1, \lambda_2, \lambda_3$ 
    return  $\lambda_1, \lambda_2, \lambda_3$ 
```

In HMMs, the most common approach for sequence prediction is the Viterbi algorithm, which guarantees finding the most likely sequence of states given the observations. However, for very large state spaces or long sequences, Viterbi can be computationally expensive.

Figure 8.10 The deleted interpolation algorithm for setting the weights for combining unigram, bigram, and trigram tag probabilities. If the denominator is 0 for any case, we define the result of that case to be 0. N is the number of tokens in the corpus. After Brants (2000).

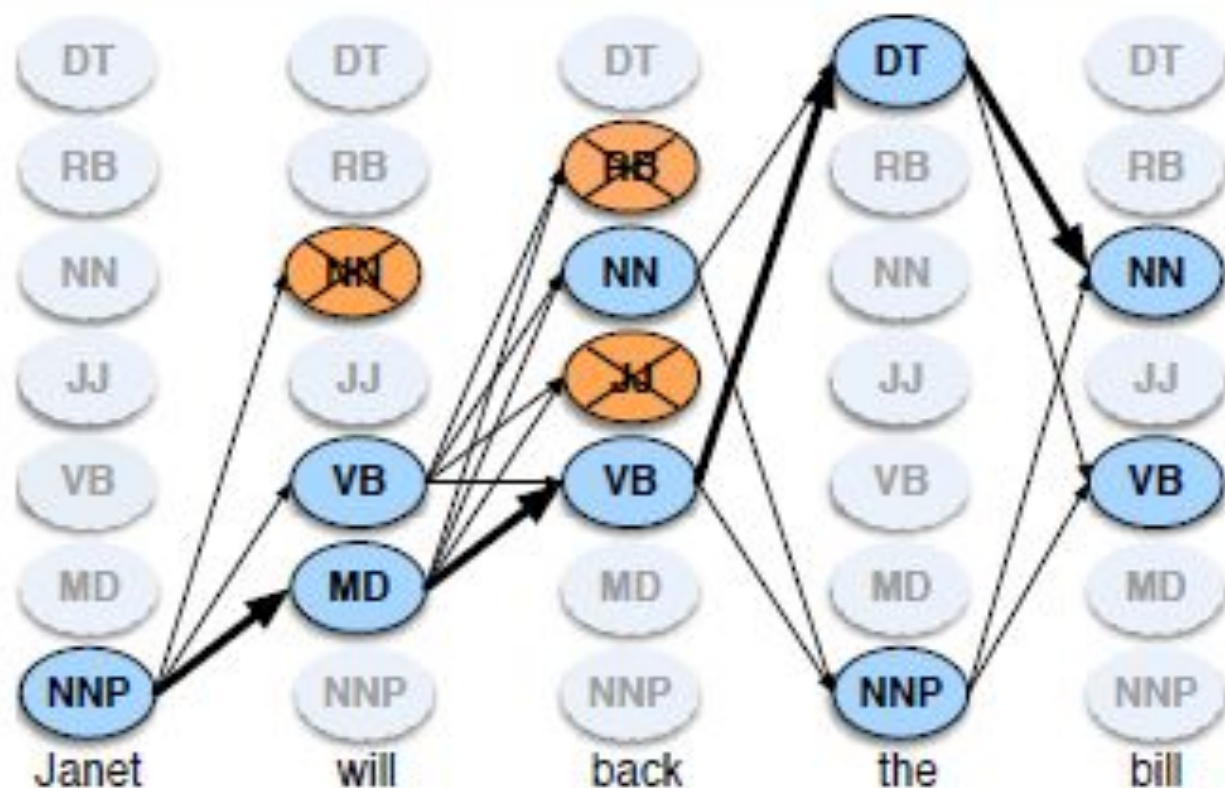


Figure 8.11 A beam search version of Fig. 8.6, showing a beam width of 2. At each time t , all (non-zero) states are computed, but then they are sorted and only the best 2 states are propagated forward and the rest are pruned, shown in orange.

First tag	Second tag						
	AT	BEZ	IN	NN	VB	PERIOD	
AT	0	0	0	48636	0	19	$P(t_i t_{1..i-1}) = \frac{c(t_{1..i-1}, t_i)}{\sum_j c(t_{1..i-1}, t_j)}$
BEZ	1973	0	426	187	0	38	
IN	43322	0	1325	17314	0	185	
NN	1067	3720	42470	11773	614	21392	
VB	6072	42	4758	1476	129	1522	
PERIOD	8016	75	4656	1329	954	0	

Table 10.3 Idealized counts of some tag transitions in the Brown Corpus. For example, NN occurs 48636 times after AT.

	AT	BEZ	IN	NN	VB	PERIOD	$P(w_i t_i) = \frac{c(w_i, t_i)}{\sum_j c(w_j, t_i)}$
<i>hear</i>	0	0	0	10	43	0	
<i>is</i>	0	10065	0	0	0	0	
<i>move</i>	0	0	0	36	133	0	
<i>on</i>	0	0	5484	0	0	0	
<i>president</i>	0	0	0	382	0	0	
<i>progress</i>	0	0	0	108	4	0	
<i>the</i>	688116	0	0	0	0	0	
<i>-</i>	0	0	0	0	0	48809	

Table 10.4 Idealized counts for the tags that some words occur with in the Brown Corpus. For example, 36 occurrences of *move* are with the tag NN.

- Probability smoothing of $P(t_i|t_{1..i-1})$ and $P(w_i|t_i)$ is necessary

Example: "He saw the dog."

Step 1: Define Transition and Emission Probabilities

- Transition Probabilities (example values):

$P(\text{VB} \mid \text{PRP}) = 0.5$ (Probability of a verb following a pronoun)

$P(\text{DT} \mid \text{VB}) = 0.4$ (Probability of a determiner following a verb)

$P(\text{NN} \mid \text{DT}) = 0.7$ (Probability of a noun following a determiner)

Emission Probabilities (example values):

$P(\text{He} \mid \text{PRP}) = 0.9$ (Probability of "He" being a pronoun)

$P(\text{saw} \mid \text{VB}) = 0.8$ (Probability of "saw" being a verb)

$P(\text{the} \mid \text{DT}) = 0.9$ (Probability of "the" being a determiner)

$P(\text{dog} \mid \text{NN}) = 0.6$ (Probability of "dog" being a noun)

Example: "He saw the dog."

Step 2: Start Probability: The sentence starts with a pronoun (PRP), so $P(\text{PRP}) = 1.0$

Step 3: Compute the Most Likely Tag Sequence

Using the Viterbi algorithm, we calculate the most likely sequence of tags. For

"He": Since it's the first word, the start probability is $P(\text{PRP}) * P(\text{He} | \text{PRP}) = 1.0 * 0.9 = 0.9$. So, the best tag for "He" is PRP.

- "saw": Considering the previous tag PRP, the most likely tag is VB.
- $P(\text{VB} | \text{PRP}) * P(\text{saw} | \text{VB}) = 0.5 * 0.8 = 0.4$

Considering the previous tag VB, the most likely tag is DT.

$P(\text{DT} | \text{VB}) * P(\text{the} | \text{DT}) = 0.4 * 0.9 = 0.36$

For "dog": Considering the previous tag DT, the most likely tag is NN.

$P(\text{NN} | \text{DT}) * P(\text{dog} | \text{NN}) = 0.7 * 0.6 = 0.42$

Step 4: Output the Best Tag Sequence

- Combining these, the most likely sequence of tags for "He saw the dog" is:
- "He/PRP saw/VB the/DT dog/NN"

Sentence: "He saw the dog." POS Tags: PRP (pronoun), VB (verb), DT

(determiner), NN (noun) Most Likely Tag Sequence: "He/PRP saw/VB the/DT dog/NN"

HMM example “she sells seashells at the seashore”

S1: Word starts with "s" (like "she", "sells", "seashells", "seashore")

S2: Word starts with any other letter

Observations:

O1: The word "she"

O2: The word "sells"

O3: The word "seashells"

O4: The word "by"

O5: The word "the"

O6: The word "seashore"

Transition Probabilities:

- $P(S1 \rightarrow S1) = 0.8$ (If the previous word starts with 's', the next word likely starts with 's')
- $P(S1 \rightarrow S2) = 0.2$ (Less likely to switch from an 's' starting word to a non-'s' word)
- $P(S2 \rightarrow S1) = 0.5$ (Switching from a non-'s' word to an 's' word)
- $P(S2 \rightarrow S2) = 0.5$ (Staying with non-'s' words)

Emission Probabilities:

- $P(O1|S1) = 0.25$ (e.g., "she" given that it's in the 's' state)
- $P(O2|S1) = 0.25$ (e.g., "sells" given that it's in the 's' state)
- $P(O3|S1) = 0.25$ (e.g., "seashells" given that it's in the 's' state)
- $P(O4|S2) = 0.5$ (e.g., "by" given that it's in the non-'s' state)
- $P(O5|S2) = 0.5$ (e.g., "the" given that it's in the non-'s' state)
- $P(O6|S1) = 0.25$ (e.g., "seashore" given that it's in the 's' state)

Initial Probabilities:

- $P(S1) = 0.6$ (The phrase is more likely to start with an 's' word)
- $P(S2) = 0.4$



Viterbi algo

Step 1: Initialization

For the first observation $O_1 = \text{"she"}$:

- $V_1(S1) = P(S1) \times P(O1|S1) = 0.6 \times 0.25 = 0.15$
- $V_1(S2) = P(S2) \times P(O1|S2) = 0.4 \times 0 = 0$ (Assuming "she" has no chance in the non-'s' state)

Step 2: Recursion

For the second observation $O_2 = \text{"sells"}$:

- $V_2(S1) = \max[V_1(S1) \times P(S1 \rightarrow S1) \times P(O2|S1), V_1(S2) \times P(S2 \rightarrow S1) \times P(O2|S1)] = \max[0.15 \times 0.8 \times 0.25, 0 \times 0.5 \times 0.25] = 0.03$
- $V_2(S2) = \max[V_1(S1) \times P(S1 \rightarrow S2) \times P(O2|S2), V_1(S2) \times P(S2 \rightarrow S2) \times P(O2|S2)] = \max[0.15 \times 0.2 \times 0, 0 \times 0.5 \times 0] = 0$

For the third observation $O_3 = \text{"seashells"}$:

- $V_3(S1) = \max[V_2(S1) \times P(S1 \rightarrow S1) \times P(O3|S1), V_2(S2) \times P(S2 \rightarrow S1) \times P(O3|S1)] = \max[0.03 \times 0.8 \times 0.25, 0 \times 0.5 \times 0.25] = 0.006$
- $V_3(S2) = \max[V_2(S1) \times P(S1 \rightarrow S2) \times P(O3|S2), V_2(S2) \times P(S2 \rightarrow S2) \times P(O3|S2)] = \max[0.03 \times 0.2 \times 0, 0 \times 0.5 \times 0] = 0$

For the fourth observation $O_4 = \text{"by"}$:

- $V_4(S1) = \max[V_3(S1) \times P(S1 \rightarrow S1) \times P(O4|S1), V_3(S2) \times P(S2 \rightarrow S1) \times P(O4|S1)] = \max[0.006 \times 0.8 \times 0, 0 \times 0.5 \times 0] = 0$
- $V_4(S2) = \max[V_3(S1) \times P(S1 \rightarrow S2) \times P(O4|S2), V_3(S2) \times P(S2 \rightarrow S2) \times P(O4|S2)] = \max[0.006 \times 0.2 \times 0.5, 0 \times 0.5 \times 0.5] = 0.0006$

For the fifth observation $O_5 = \text{"the"}$:

- $V_5(S1) = \max[V_4(S1) \times P(S1 \rightarrow S1) \times P(O5|S1), V_4(S2) \times P(S2 \rightarrow S1) \times P(O5|S1)] = \max[0 \times 0.8 \times 0, 0.0006 \times 0.5 \times 0] = 0$
- $V_5(S2) = \max[V_4(S1) \times P(S1 \rightarrow S2) \times P(O5|S2), V_4(S2) \times P(S2 \rightarrow S2) \times P(O5|S2)] = \max[0 \times 0.2 \times 0.5, 0.0006 \times 0.5 \times 0.5] = 0.00015$

For the sixth observation $O_6 = \text{"seashore"}$:

- $V_6(S1) = \max[V_5(S1) \times P(S1 \rightarrow S1) \times P(O6|S1), V_5(S2) \times P(S2 \rightarrow S1) \times P(O6|S1)] = \max[0 \times 0.8 \times 0.25, 0.00015 \times 0.5 \times 0.25] = 0.00001875$
- $V_6(S2) = \max[V_5(S1) \times P(S1 \rightarrow S2) \times P(O6|S2), V_5(S2) \times P(S2 \rightarrow S2) \times P(O6|S2)] = \max[0 \times 0.2 \times 0, 0.00015 \times 0.5 \times 0] = 0$

Step 3: Termination

The most likely final state based on the calculations:

- $V_6(S1) = 0.00001875$
- $V_6(S2) = 0$

Since $V_6(S1)$ is greater, the student ends in the "s" starting word state.

Most Likely State Sequence:

The most likely sequence of states for the given observation sequence:

- S1 (she) → S1 (sells) → S1 (seashells) → S2 (by) → S2 (the) → S1 (seashore)

Transformation based POS tagger(Hybrid)

Transformation-Based Tagging

- Transformation based tagging is also called **Brill tagging**. It is a rule-based algorithm for automatic tagging of POS to the given text.
- TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

Combination of Rule-based and stochastic tagging

- Like rule-based because rules are used to specify tags in a certain environment
- Like stochastic approach because machine learning is used—with tagged corpus as input

- Input: tagged corpus

dictionary (with most frequent tags)

Usually constructed from the tagged corpus

Working of Transformation Based Learning(TBL)

steps to understand the working of TBL –

- **Start with the solution** – The TBL usually starts with some solution to the problem and works in cycles.
- **Most beneficial transformation chosen** – In each cycle, TBL will choose the most beneficial transformation.
- **Apply to the problem** – The transformation chosen in the last step will be applied to the problem.

The algorithm will stop when the selected transformation in step 2 will not add either more value or there are no more transformations to be selected.

- Possible templates (abstracted transformations)

The preceding (following) word is tagged **z**.
 The word two before (after) is tagged **z**.
 One of the two preceding (following) words is tagged **z**.
 One of the three preceding (following) words is tagged **z**.
 The preceding word is tagged **z** and the following word is tagged **w**.
 The preceding (following) word is tagged **z** and the word
 two before (after) is tagged **w**.

Brill's templates.
 Each begins with
 "Change tag a to tag
 b when"

Schema	t_{i-3}	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}	t_{i+3}
1				*			
2				*			
3				*			
4				*			
5				*			
6				*			
7				*			
8				*			
9				*			

Table 10.7 Triggering environments in Brill's transformation-based tagger. Examples: Line 5 refers to the triggering environment "Tag t^j occurs in one of the three previous positions"; Line 9 refers to the triggering environment "Tag t^j occurs two positions earlier and tag t^k occurs in the following position."

- Learned transformations

Verb, 3sg, past tense

Modal verbs (should, can,...)

Change tags			Condition	Example
#	From	To		
1	NN	VB	Previous tag is TO	to/TO race/NN → VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not reply/NN → VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

Rules learned by
Brill's original tagger

Verb, past participle

Verb, 3sg, Present

Table 10.7 Triggering environments in Brill's transformation-based tagger. Examples: Line 5 refers to the triggering environment "Tag t^j occurs in one of the three previous positions"; Line 9 refers to the triggering environment "Tag t^j occurs two positions earlier and tag t^k occurs in the following position."

Source tag	Target tag	Triggering environment	
NN	VB	previous tag is TO	} Constraints for tags
VBP	VB	one of the previous three tags is MD	
JJR	RBR	next tag is JJ	
VBP	VB	one of the previous two words is n't	} Constraints for words

Table 10.8 Examples of some transformations learned in transformation-based tagging.

Verb to Noun

- Condition: Change a word tagged as a verb (VB) to a noun (NN) if it is the last word in the sentence. In the sentence "I need to study," the word "study" might be initially tagged as a verb (VB). The rule keeps it as a verb unless it's at the end of a sentence, where the context might suggest it should be a noun.
- Rule: VB -> NN if next_word is END

Noun to Verb

- Condition: Change a word tagged as a noun (NN) to a verb (VB) if it follows an auxiliary verb (AUX). In the sentence "She can fish," the word "fish" might initially be tagged as a noun (NN). The rule changes it to a verb (VB) because it follows the auxiliary verb "can."
- Rule: NN -> VB if previous_tag is AUX

Algorithm

1. Step 1: Label every word with most likely tag (from dictionary)
2. Step 2: Check every possible transformation & select one which most improves tagging
3. Step 3: Re-tag corpus applying the rules
4. Repeat 2-3 until some criterion is reached, e.g., X% correct with respect to training corpus
5. RESULT: Sequence of transformation rules

Transformation rules make changes to tags

“Change NN to VB when previous tag is TO”

... is/VBZ expected/VBN to/TO race/NN tomorrow/NN

becomes

... is/VBZ expected/VBN to/TO race/VB tomorrow/NN

Example 2

Given a sentence "The can is on the table."

- Initial Tags: "The/DT can/NN is/VBZ on/IN the/DT table/NN"
- After Rule Application: The tagger might apply a rule that changes "can" from a noun (NN) to a verb (VB) if it is in the context "The can..." resulting in "The/DT can/VB is/VBZ on/IN the/DT table/NN"

Advantages of Transformation-based Learning (TBL)

- We learn small set of simple rules and these rules are enough for tagging.
- Development as well as debugging is very easy in TBL because the learned rules are easy to understand.
- Complexity in tagging is reduced because in TBL there is interlacing of machine learned and human-generated rules.
- Transformation-based tagger is **much faster than** Markov-model tagger.

Disadvantages of Transformation-based Learning (TBL)

- Transformation-based learning (TBL) does not provide tag probabilities.
- In TBL, the training **time is very long especially on large corpora.**

Issues/ Challenges in POS tagging

- Multiple tags and multiple words

Adj vs past tense vs past participle(JJ/VBD/VBN)

- Unknown words

- Foreign words

- Ambiguities

- Ungrammatical input

- Tokenization

The_DT first_JJ time_NN he_PRP was_VBD shot_VBN in_IN the_DT hand_NN as_IN he_PRP chased_VBD the_DT robbers_NNS outside_RB ...							
first	time	shot	in	hand	as	chased	outside
JJ	NN	NN	IN	NN	IN	JJ	IN
RB	VB	VBD	RB	VB	RB	VBD	JJ
		VBN	RP			VBN	NN
							RB

Example showing POS ambiguity. Source: Màrquez et al. 2000, table 1. 

common words have multiple meanings and therefore multiple POS

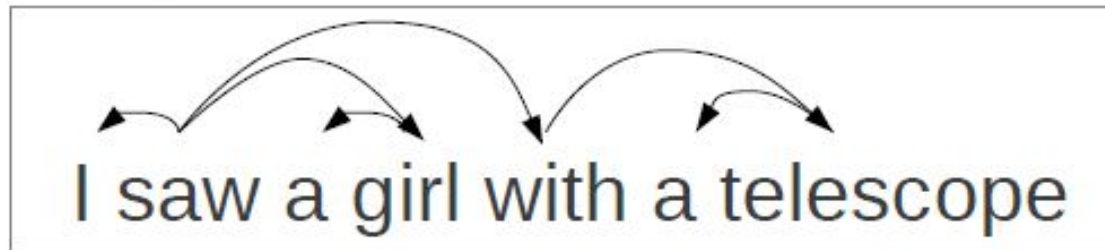
Parsing

Syntactic Processing

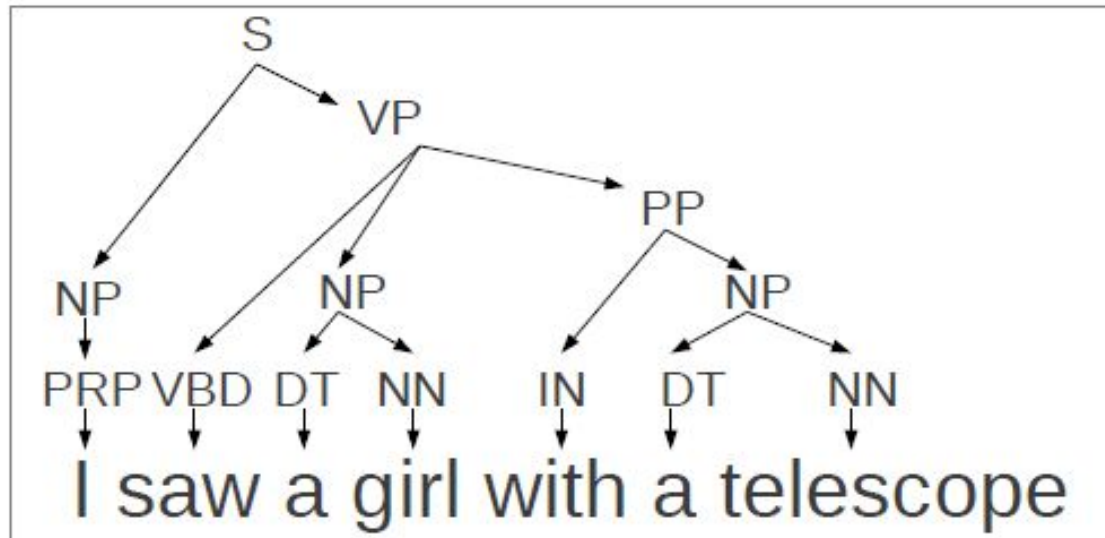
- **Parsing** -- converting a flat input sentence into a hierarchical structure that corresponds to the units of meaning in the sentence.
- There are different parsing formalisms and algorithms.
- Most formalisms have two main components:
 - **grammar** -- a declarative representation describing the syntactic structure of sentences in the language.
 - **parser** -- an algorithm that analyzes the input and outputs its structural representation (its parse) consistent with the grammar specification.
- **CFGs are** in the center of many of the parsing mechanisms. But they are complemented by some additional features that make the formalism more suitable to handle natural languages.

Two Types of Parsing

- **Dependency:** focuses on relations between words

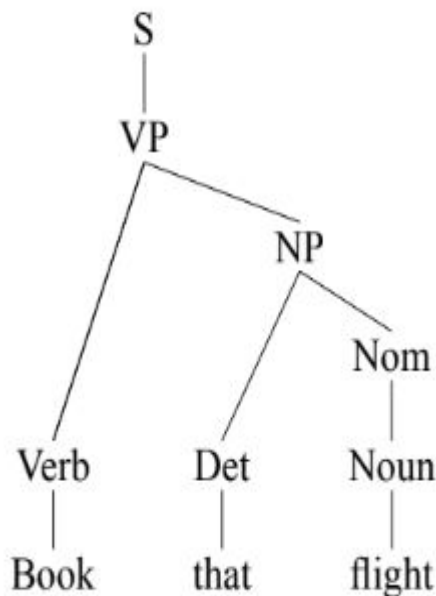


- **Phrase structure:** focuses on identifying phrases and their recursive structure



Parsing With Context Free Grammar

- In syntactic parsing, the parser can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence. The search space of possible parse trees is defined by the grammar.
- **Consider the sentence : Book that flight**



$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Noun Nominal$

$NP \rightarrow Proper-Noun$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$Det \rightarrow that \mid this \mid a$

$Noun \rightarrow book \mid flight \mid meal \mid money$

$Verb \rightarrow book \mid include \mid prefer$

$Aux \rightarrow does$

$Prep \rightarrow from \mid to \mid on$

$Proper-Noun \rightarrow Houston \mid TWA$

$Nominal \rightarrow Nominal PP$

Top-Down VS. Bottom-Up

Top-down

- Only searches for trees that can be answers
- But suggests trees that are not consistent with the words
- Guarantees that tree starts with S as root
- Does not guarantee that tree will match input words

Bottom-up

- Only forms trees consistent with the words
- Suggest trees that make no sense globally
- Guarantees that tree matches input words
- Does not guarantee that parse tree will lead to S as a root

Context Free Grammars

- **A context-free grammar (CFG)** is a list of rules that define the set of all well-formed sentences in a language.
- Each rule has a left-hand side, which identifies a syntactic category, and a right-hand side, which defines its alternative component parts, reading from left to right.
- **Constituency, grammatical relations, and subcategorization and dependencies**
- **Constituency** is that groups of words may behave as a single unit or phrase, called a constituent.
- For example we will see that a group of words called a **noun phrase** often acts as a unit; noun phrases include single words like she or Michael and phrases like the house, Russian Hill

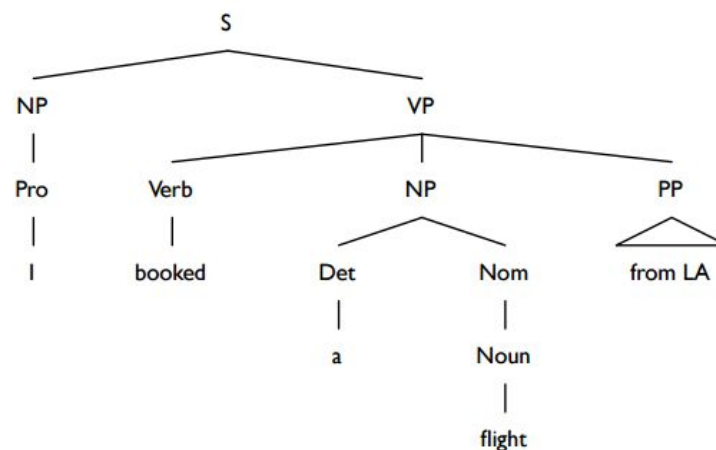
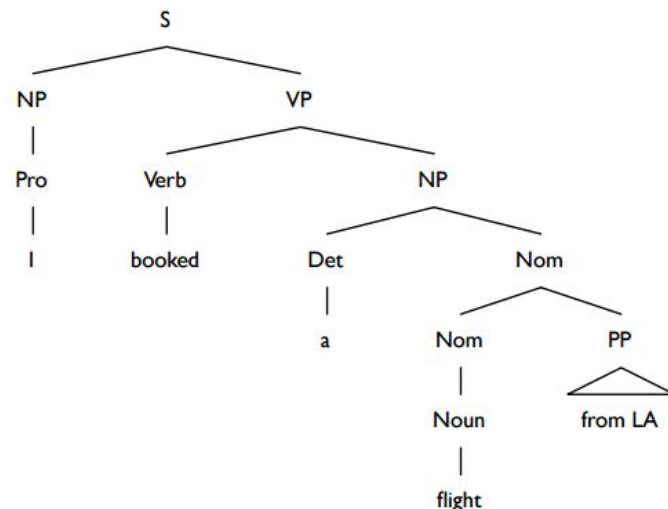
Context Free Grammars

- **Grammatical relations** are a formalization of ideas from traditional grammar about SUBJECTS and OBJECTS.
- In the sentence: She ate a mammoth breakfast. The noun phrase She is the SUBJECT and a mammoth breakfast is the OBJECT.
- **Subcategorization and dependency relations** refer to certain kinds of relations between words and phrases.
- For example the verb want can be followed by an infinitive, as in I want to fly to Detroit, or a noun phrase, as in I want a flight to Detroit. But the verb find cannot be followed by an infinitive (*I found to fly to Dallas)

Parsing

- Parsing is the automatic analysis of a sentence with respect to its syntactic structure
- Given a CFG, this means deriving a phrase structure tree assigned to the sentence by the grammar. With ambiguous grammars, each sentence may have many valid parse trees •
- Should we retrieve all of them or just one?
- If the latter, how do we know which one?

I booked a flight from LA.



Ambiguity - an input may have exponentially many parses.

Context Free Grammars

<sentence> --> <noun phrase> <verb phrase>

<noun phrase> --> <adjective> <noun phrase> | <adjective>

<singular noun>

<verb phrase> --> <singular verb> <adverb>

<adjective> --> a | the | little

<singular noun> --> boy

<singular verb> --> ran

<adverb> --> quickly

Context Free Grammars

s → np vp

np → det n

vp → tv np

→ iv

det → the

→ a

→ an

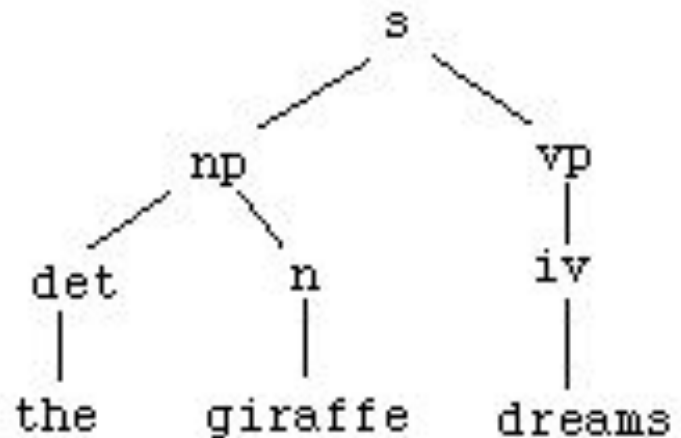
n → giraffe

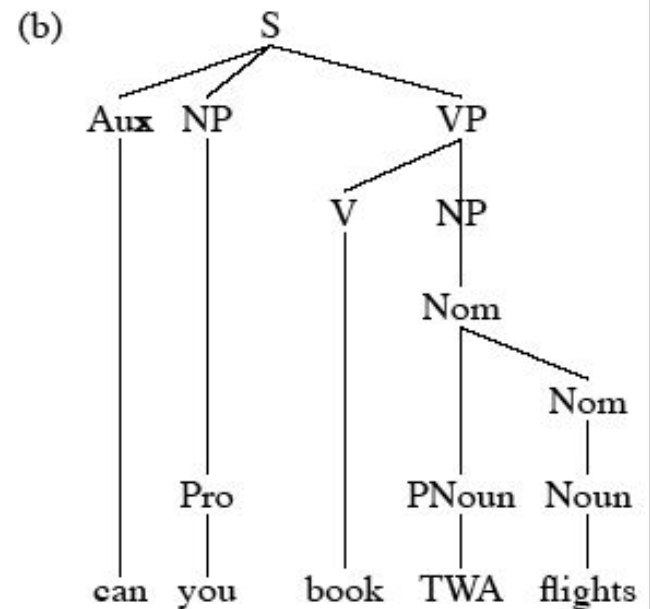
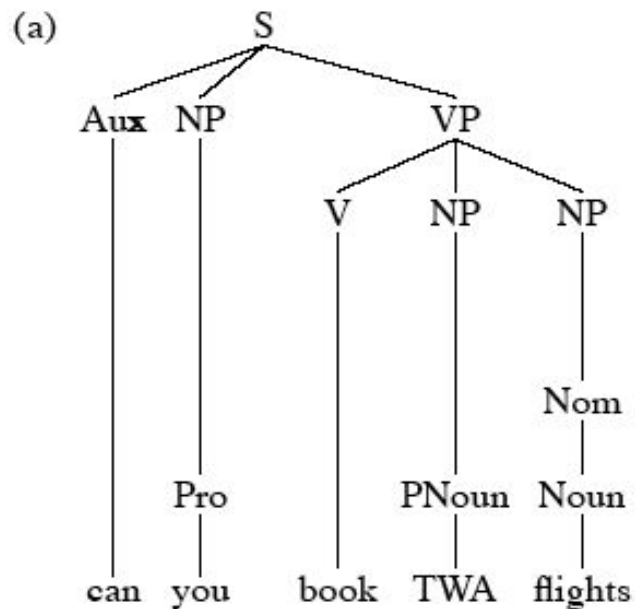
→ apple

iv → dreams

tv → eats

→ dreams





Rules			P	Rules			P
S	→	Aux NP VP	.15	S	→	Aux NP VP	.15
NP	→	Pro	.40	NP	→	Pro	.40
VP	→	V NP NP	.05	VP	→	V NP	.40
NP	→	Nom	.05	NP	→	Nom	.05
NP	→	PNoun	.35	Nom	→	PNoun Nom	.05
Nom	→	Noun	.75	Nom	→	Noun	.75
Aux	→	Can	.40	Aux	→	Can	.40
NP	→	Pro	.40	NP	→	Pro	.40
Pro	→	you	.40	Pro	→	you	.40
Verb	→	book	.30	Verb	→	book	.30
PNoun	→	TWA	.40	Pnoun	→	TWA	.40
Noun	→	flights	.50	Noun	→	flights	.50

Potential Problems in Context Free Grammars

- **Agreement**
- **Subcategorization**
- **Movement**

- **Number Agreement**

In English, some determiners agree in number with the head noun:

This dog

Those dogs

*Those dog

*This dogs

And verbs agree in number with their subjects:

What flights leave in the morning?

*What flight leave in the morning?

● Number Agreement

Expand our grammar with multiple sets of rules?

$$\text{NP}_{\text{sg}} \rightarrow \text{Det}_{\text{sg}} \text{N}_{\text{sg}}$$
$$\text{NP}_{\text{pl}} \rightarrow \text{Det}_{\text{pl}} \text{N}_{\text{pl}}$$
$$\text{S}_{\text{sg}} \rightarrow \text{NP}_{\text{sg}} \text{VP}_{\text{sg}}$$
$$\text{S}_{\text{pl}} \rightarrow \text{NP}_{\text{pl}} \text{VP}_{\text{pl}}$$
$$\text{VP}_{\text{sg}} \rightarrow \text{V}_{\text{sg}} (\text{NP}) (\text{NP}) (\text{PP})$$
$$\text{VP}_{\text{pl}} \rightarrow \text{V}_{\text{pl}} (\text{NP}) (\text{NP}) (\text{PP})$$

worse when we add person and even worse in languages with richer agreement (e.g., three genders).

lose generalizations about nouns and verbs — can't say property P is true of all words of category V.

● Subcategorization

More examples:

- ▶ *find* is subcategorized for an NP (can take an NP complement)
- ▶ *want* is subcategorized for an NP or an infinitival VP
- ▶ *bet* is subcategorized for NP NP S

A listing of the possible sequences of complements is called the **subcategorization frame** for the verb.

As with agreement, the obvious CFG solution yields rule explosion:

$$VP \rightarrow V_{\text{intr}}$$
$$VP \rightarrow V_{\text{tr}} \text{ NP}$$
$$VP \rightarrow V_{\text{ditr}} \text{ NP NP}$$

- **Subcategorization Frames**

Frame	Verb	Example
—	<i>eat, sleep</i>	<i>I want to eat</i>
NP	<i>prefer, find, leave,</i>	<i>Find [NP the flight from Pittsburgh to Boston]</i>
NP NP	<i>show, give</i>	<i>Show [NP me] [NP airlines with flights from Pittsburgh]</i>
NP PP	<i>help, load,</i>	<i>Can you help [NP me] [PP with a flight]</i>
VP _{inf}	<i>prefer, want, need</i>	<i>I would prefer [VP_{inf} to go by United airlines]</i>
S	<i>mean</i>	<i>Does this mean [S AA has a hub in Boston]?</i>

- Should these be correct?
 - John sneezed the book
 - I prefer United has a flight
 - Give with a flight
- The various rules for VPs *overgenerate*.
 - They permit the presence of strings containing verbs and arguments that don't go together
 - For example $VP \rightarrow V NP$ therefore
Sneezed the book is a VP since “sneeze” is a verb and “the book” is valid NP
- Now *overgeneration* is a problem for a generative approach
 - The grammar should represent *all and only* the strings in a language
- From a practical point of view... Not so clear that there's a problem

● Movement

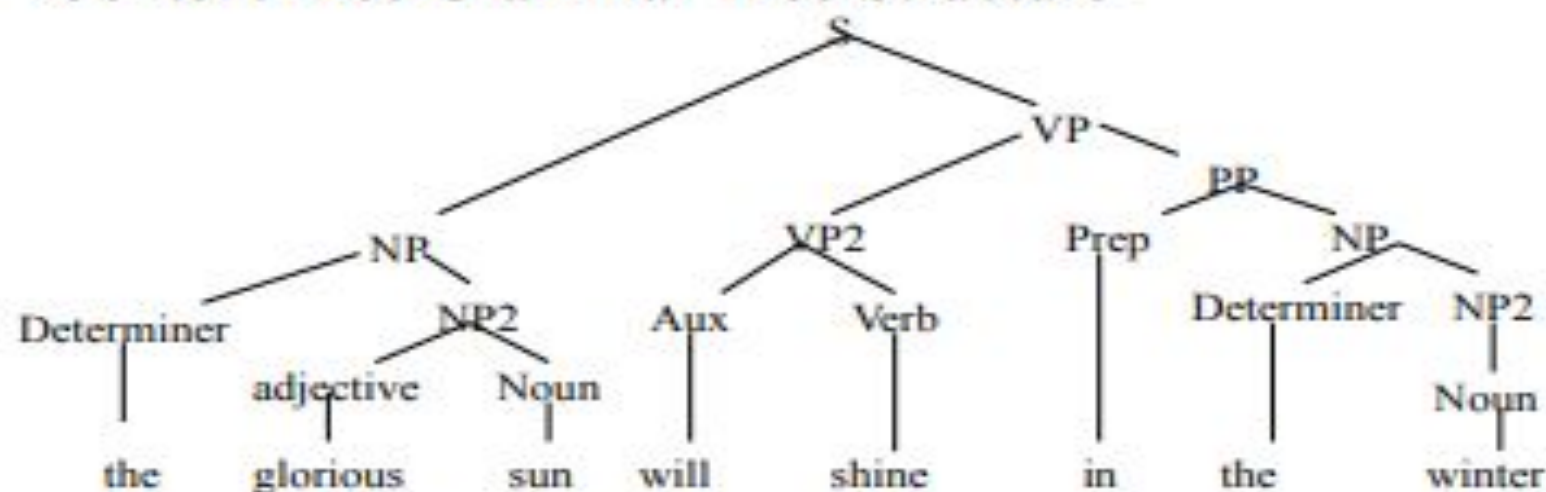
- ▶ **I gave __to the driver.*
- ▶ *I gave some money to the driver.*
- ▶ *\$5 [I gave __to the driver], (and \$1 I gave to the porter).*
- ▶ *He asked how much [I gave __to the driver].*
- ▶ *I forgot about the money which [I gave __to the driver].*
- ▶ *How much did you think [I gave __to the driver]?*
- ▶ *How much did you think he claimed [I gave __to the driver]?*
- ▶ *How much did you think he claimed that I said [I gave __to the driver]?*
- ▶ ...

Dependency Grammars

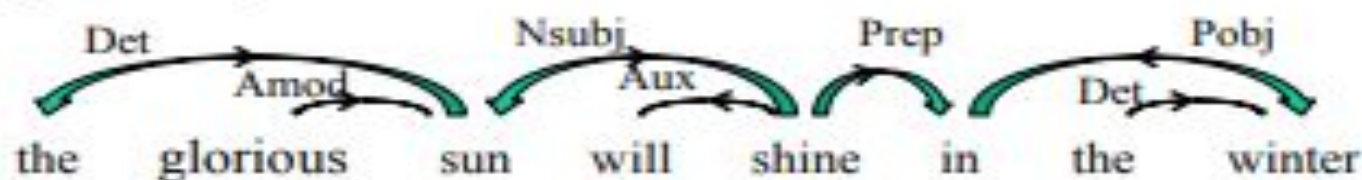
- Dependency grammars offer a different way to represent syntactic structure
 - CGS represent constituents in a parse tree that can derive the words of a sentence
 - Dependency grammars represent syntactic dependency relations between words that show the syntactic structure
 - Typed dependency grammars label those relations as to what the syntactic structure is
- Syntactic structure is the set of relations between a word (aka the head word) and its dependents.

Examples

- Context Free Grammar Tree Structure



- Dependency Relation Structure



Types of Parser

- Given a string of 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 parsers – they never explore illegal parses (e.g. which can't form an S) -- but waste time on trees that can never match the input. May reparse the same constituent repeatedly.
 - Bottom-Up parsers – they never explore trees inconsistent with input -- but waste time exploring illegal parses (with no S root)

Deterministic parsing (e.g., LL(1)) aims to address a limited amount of **local ambiguity** – the problem of not being able to decide uniquely which grammar rule to use next in a left-to-right analysis of the input string.

By re-structuring the grammar, the parser can make a unique decision, based on a limited amount of **look-ahead**.

Recursive Descent parsing also demands grammar restructuring, in order to eliminate left-recursive rules that can get it into a hopeless loop.

Simple CFG for ATIS English

Grammar

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Pronoun$

$NP \rightarrow Proper-Noun$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Nominal Noun$

$Nominal \rightarrow Nominal PP$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$VP \rightarrow VP PP$

$PP \rightarrow Prep NP$

Lexicon

$Det \rightarrow the \mid a \mid that \mid this$

$Noun \rightarrow book \mid flight \mid meal \mid money$

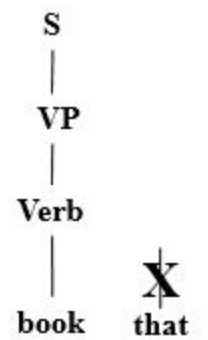
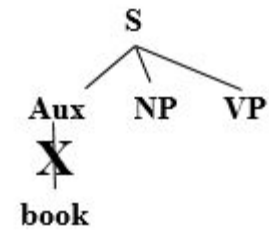
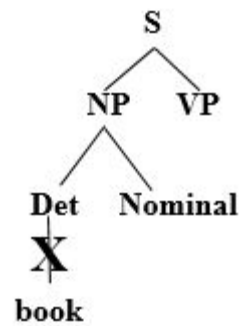
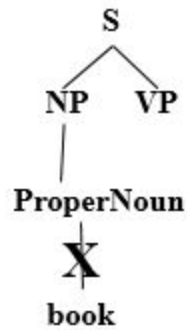
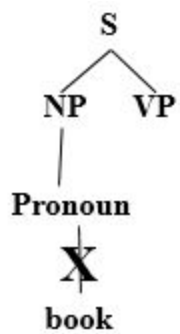
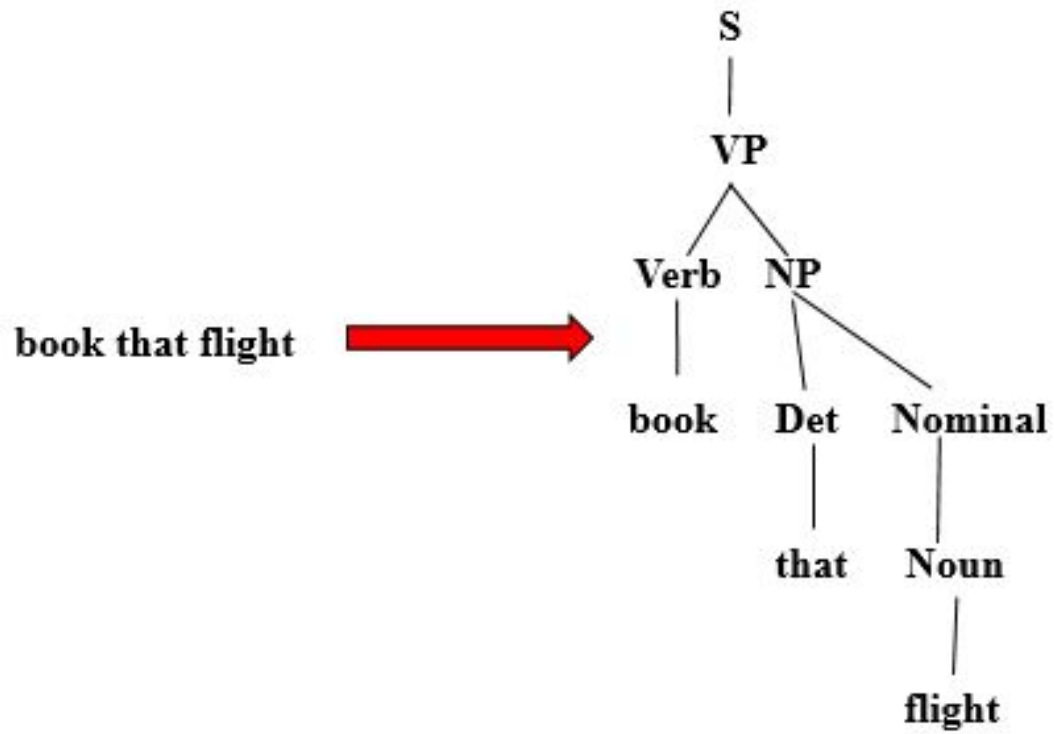
$Verb \rightarrow book \mid include \mid prefer$

$Pronoun \rightarrow I \mid he \mid she \mid me$

$Proper-Noun \rightarrow Houston \mid NWA$

$Aux \rightarrow does$

$Prep \rightarrow from \mid to \mid on \mid near \mid through$



Recursive Descent Parser

A recursive descent parser (top-down) will do badly if there are many different rules for the same LHS. Hopeless for rewriting parts of speech (preterminals) with words (terminals)

```
grammar = CFG.fromstring( """
S -> NP VP
PP -> P NP
NP -> 'the' N | N PP | 'the' N PP
VP -> V NP | V PP | V NP PP
N -> 'cat' | 'dog' | 'rug'
V -> 'chased'
P -> 'on'
""")
```

```
rd = RecursiveDescentParser(grammar)
```

the cat chased the dog on the rug

(S (NP the (N cat))

(VP (V chased) (NP the (N dog) (PP (P on) (NP
the (N rug))))))

(S (NP the (N cat))

(VP (V chased) (NP the (N dog)) (PP (P on) (NP
the (N rug))))))

CYK algorithm

earliest recognition and parsing algorithms

- For a parse spanning substring $[i,j]$, there exists some k such there are parses spanning $[i,k]$ and $[k,j]$ We can construct parses for whole sentence by building up from these stored partial parses
- So, To have a rule $A \rightarrow BC$ in $[i,j]$, We must have B in $[i,k]$ and C in $[k,j]$, for some $i < k < j$

```
function CKY-PARSE(words, grammar) returns table
```

```
  for  $j \leftarrow$  from 1 to LENGTH(words) do
```

```
     $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$ 
```

```
    for  $i \leftarrow$  from  $j-2$  downto 0 do
```

```
      for  $k \leftarrow i+1$  to  $j-1$  do
```

```
         $table[i, j] \leftarrow table[i, j] \cup$ 
```

```
           $\{A \mid A \rightarrow BC \in grammar,$ 
```

```
             $B \in table[i, k],$ 
```

```
             $C \in table[k, j]\}$ 
```

Consider the grammar G given by:

$S \rightarrow \varepsilon \mid AB \mid XB$

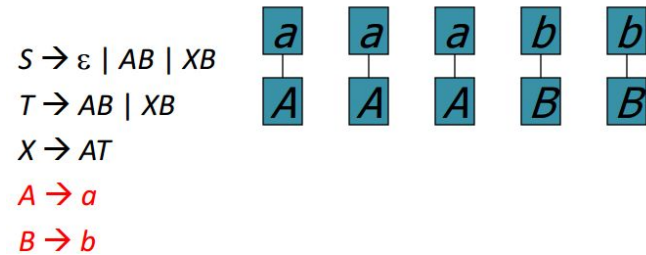
$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

1) Write variables for all length 1 substrings



1. Is $w = aaabb$ in $L(G)$?

2. Is $w = aaabbb$ in $L(G)$?

5) Write variables for all length 5 substrings.

$S \rightarrow \varepsilon \mid AB \mid XB$

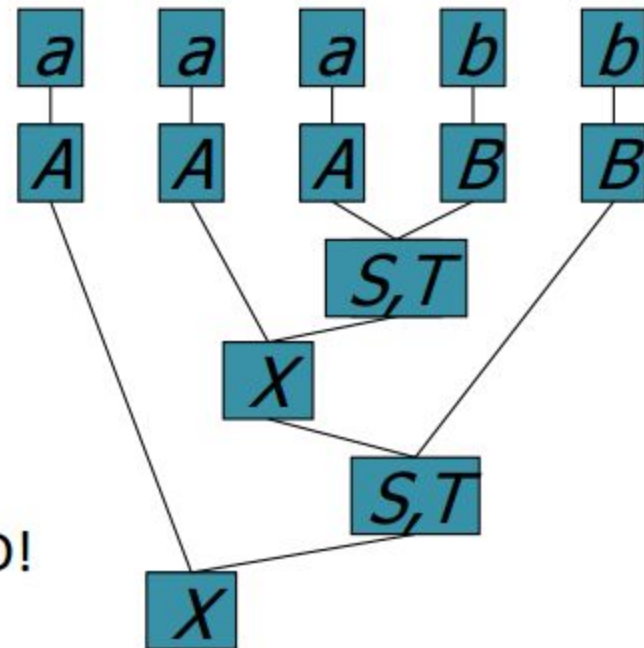
$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

$aaabb$ REJECTED!



CYK Parser

- Given an input String S of length n , Build table ($n*n$)
- Indices correspond to inter-word positions

0 Book 1 the 2 flight 3 through 4 Houston 5

Cells $[i,j]$ contain sets of non-terminals of ALL constituents spanning i,j

- $[j-1,j]$ contains pre-terminals
- If $[0,n]$ contains Start, the input is recognized

	Book	the	flight	through	Houston
	j=1	2	3	4	5
i=0					
1					
2					
3					
4					

Cell $[i,j]$ contains all constituents (non-terminals) covering words $i+1$ through j

CKY Parser

	Book	the	flight	through	Houston
	j= 1	2	3	4	5
i= 0					
1					
2					
3					
4					

Cell[i,j]
contains all
constituents
(non-terminals)
covering words
 $i+1$ through j

ATIS English Grammar Conversion

Original Grammar

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Pronoun$

$NP \rightarrow Proper-Noun$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Nominal Noun$

$Nominal \rightarrow Nominal PP$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$VP \rightarrow VP PP$

$PP \rightarrow Prep NP$

Chomsky Normal Form

$S \rightarrow NP VP$

$S \rightarrow X1 VP$

$X1 \rightarrow Aux NP$

$S \rightarrow book \mid include \mid prefer$

$S \rightarrow Verb NP$

$S \rightarrow VP PP$

$NP \rightarrow I \mid he \mid she \mid me$

$NP \rightarrow Houston \mid NWA$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow book \mid flight \mid meal \mid money$

$Nominal \rightarrow Nominal Noun$

$Nominal \rightarrow Nominal PP$

$VP \rightarrow book \mid include \mid prefer$

$VP \rightarrow Verb NP$

$VP \rightarrow VP PP$

$PP \rightarrow Prep NP$

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	through	Houston
NN, VB, Nominal, VP, S [0,1]				
	Det [1,2]			

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]			
	Det [1,2]			
		NN, Nominal [2,3]		

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]		
	Det [1,2]	NP [1,3]		
		NN, Nominal [2,3]		

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]		
	Det [1,2]	NP [1,3]		
		NN, Nominal [2,3]		
			Prep [3,4]	

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	
	Det [1,2]	NP [1,3]	[1,4]	
		NN, Nominal [2,3]	[2,4]	
			Prep [3,4]	

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	
	Det [1,2]	NP [1,3]	[1,4]	
		NN, Nominal [2,3]	[2,4]	
			Prep [3,4]	PP [3,5]
				NNP, NP [4,5]

0 Book 1 the 2 flight 3 through 4 Houston 5

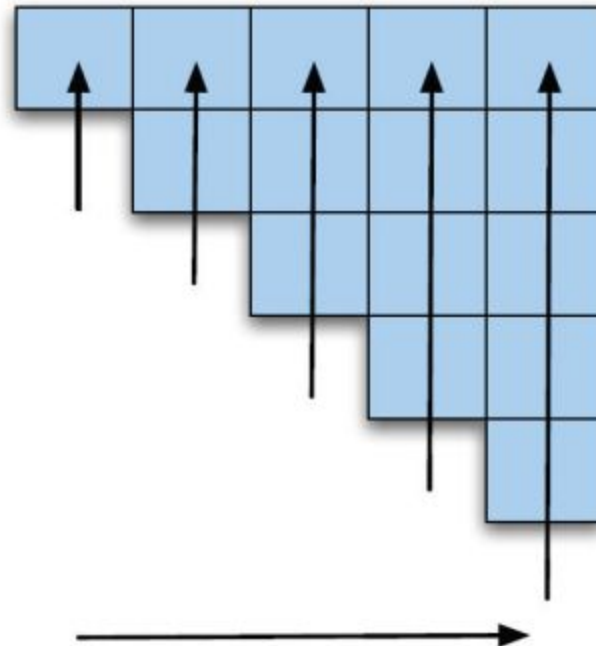
Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	S, VP, X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		NN, Nominal [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NNP, NP [4,5]

From Recognition to Parsing

- Limitations of current recognition algorithm:
 - Only stores non-terminals in cell
 - Not rules or cells corresponding to RHS
 - Stores SETS of non-terminals
 - Can't store multiple rules with same LHS
- Parsing solution:
 - All repeated versions of non-terminals
 - Pair each non-terminal with pointers to cells
 - Backpointers

- Book the flight through Houston

<i>Book the flight through Houston</i>				
S, VP, Verb Nominal, Noun [0,1]		S, VP, X2 [0,3]		S, VP [0,5]
	Det [1,2]	NP [1,3]		NP [1,5]
		Nominal, Noun [2,3]		Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]



Book the flight through Houston

S, VP, Verb, Nominal, Noun	None			
	Det	NP		
		Nominal, Noun		

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	VP		
	Det	NP		
		Nominal, Noun		

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	S VP	None	
	Det	NP	None	
		Nominal, Noun	None	
			Prep	
				NP ProperNoun

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	S VP	None	VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	S VP	None	VP S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	S VP	None	S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

Book the flight through Houston

S, VP, Verb, Nominal, Noun	None	S VP	None	S VP S VP
	Det	NP	None	NP
		Nominal, Noun	None	Nominal
			Prep	PP
				NP ProperNoun

a 1	pilot 2	likes 3	flying 4	planes 5

$S \rightarrow NP VP$
 $VP \rightarrow VBG NNS$
 $VP \rightarrow VBZ VP$
 $VP \rightarrow VBZ NP$
 $NP \rightarrow DT NN$
 $NP \rightarrow JJ NNS$
 $DT \rightarrow a$
 $NN \rightarrow pilot$
 $VBZ \rightarrow likes$
 $VBG \rightarrow flying$
 $JJ \rightarrow flying$
 $NNS \rightarrow planes$

S -> NP VP

S -> VP

NP -> NP PP

NP -> PropNoun

VP -> Verb

VP -> Verb NP

VP -> VP PP

PP -> Prep NP

PropNoun -> DALLAS

PropNoun -> ALICE

PropNoun -> BOB

PropNoun -> AUSTIN

Verb -> ADORE

Verb -> SEE

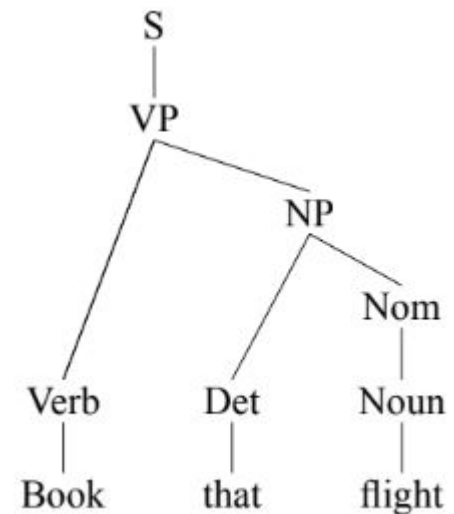
Prep -> IN

Prep -> WITH

“SEE BOB IN AUSTIN”

Top-Down Parsing

- Top-down parsing is goal-directed.
- A top-down parser starts with a list of constituents to be built.
- It rewrites the goals in the goal list by matching one against the LHS of the grammar rules, and expanding it with the RHS, attempting to match the sentence to be derived.
- If a goal can be rewritten in several ways, then there is a choice of which rule to apply (search problem).
- Can use depth-first or breadth-first search and goal ordering.
- Consider the sentence : Book that flight



Bottom-up parsing

- The initial goal list of a bottom-up parser is the string to be parsed.
- If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule.
- Parsing is finished when the goal list contains just the start symbol.
- If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)
- Can use depth-first or breadth-first search, and goal ordering. The standard presentation is as shift-reduce parsing.

Simple CFG for ATIS English

Grammar

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Pronoun$

$NP \rightarrow Proper-Noun$

$NP \rightarrow Det Nominal$

$Nominal \rightarrow Noun$

$Nominal \rightarrow Nominal Noun$

$Nominal \rightarrow Nominal PP$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$VP \rightarrow VP PP$

$PP \rightarrow Prep NP$

Lexicon

$Det \rightarrow the \mid a \mid that \mid this$

$Noun \rightarrow book \mid flight \mid meal \mid money$

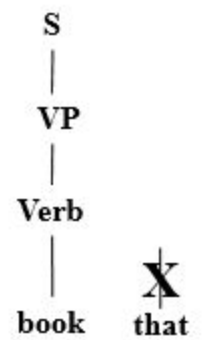
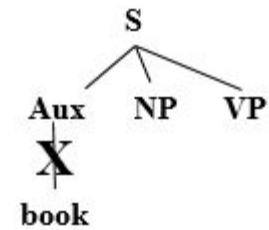
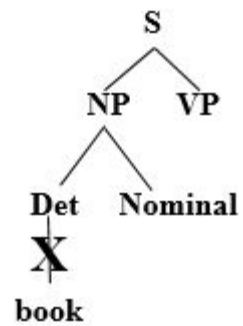
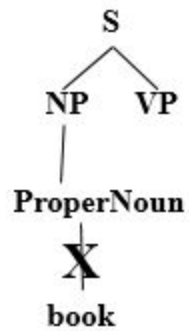
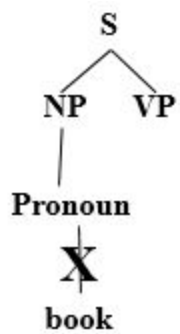
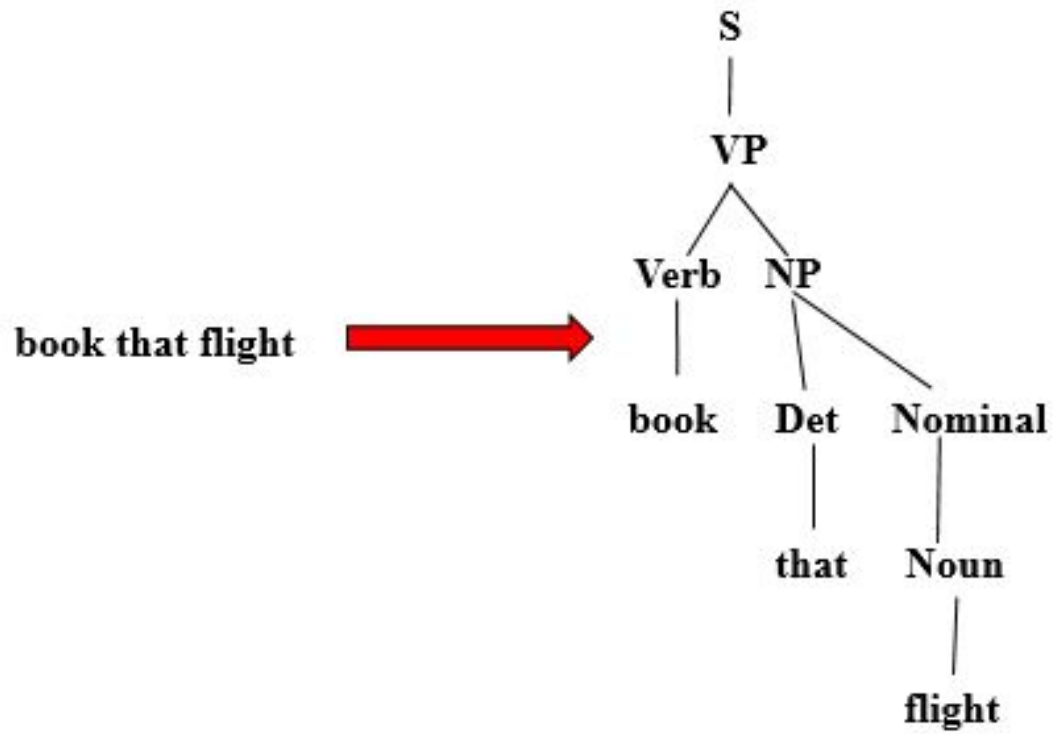
$Verb \rightarrow book \mid include \mid prefer$

$Pronoun \rightarrow I \mid he \mid she \mid me$

$Proper-Noun \rightarrow Houston \mid NWA$

$Aux \rightarrow does$

$Prep \rightarrow from \mid to \mid on \mid near \mid through$



Shift-reduce parsing

Stack	Input remaining	Action
()	Book that flight	shift
(Book)	that flight	reduce, Verb \rightarrow book, (Choice #1 of 2)
(Verb)	that flight	shift
(Verb that)	flight	reduce, Det \rightarrow that
(Verb Det)	flight	shift
(Verb Det flight)		reduce, Noun \rightarrow flight
(Verb Det Noun)		reduce, NOM \rightarrow Noun
(Verb Det NOM)		reduce, NP \rightarrow Det NOM
(Verb NP)		reduce, VP \rightarrow Verb NP
(Verb)		reduce, S \rightarrow V
(S)		SUCCESS!

CFG Summary

- ▶ CFGs capture hierarchical structure of constituents in natural language.
- ▶ More powerful than REs, and can express recursive structure.
- ▶ Hard to get a variety of linguistic generalizations in 'vanilla' CFGs, though this can be mitigated with use of features (not covered here).
- ▶ Building a CFG for a reasonably large set of English constructions is a lot of work!

Garden path sentences

- This leads the reader toward a seemingly familiar meaning that is actually not the one intended.
- It is a special type of sentence that creates a momentarily ambiguous interpretation because it contains a word or phrase that can be interpreted in multiple ways, causing the reader to begin to believe that a phrase will mean one thing when in reality it means something else.

Eg 1: The horse raced past the barn fell.

The sentence is hard to parse because raced can be interpreted as a finite verb or as a passive participle. The reader initially interprets raced as the main verb in the simple past, but when the reader encounters fell, they are forced to re-analyze the sentence, concluding that raced is being used as a passive participle and horse is the direct object of the subordinate clause.

eg 2: They painted the wall with cracks.

The horse raced past the barn fell

S -> NP VP

NP -> Det N | Det N RelClause

VP -> V NP | V PP | V NP RelClause | V

PP -> P NP

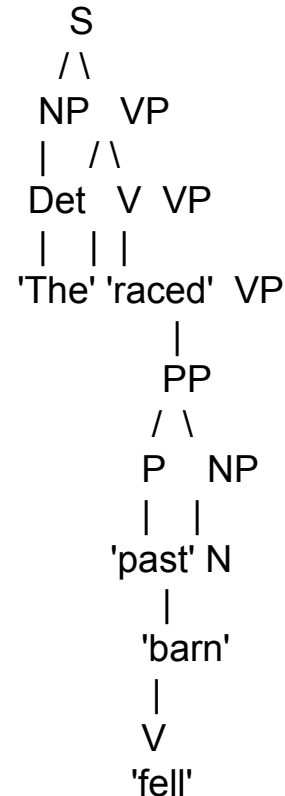
RelClause -> NP VP | VP

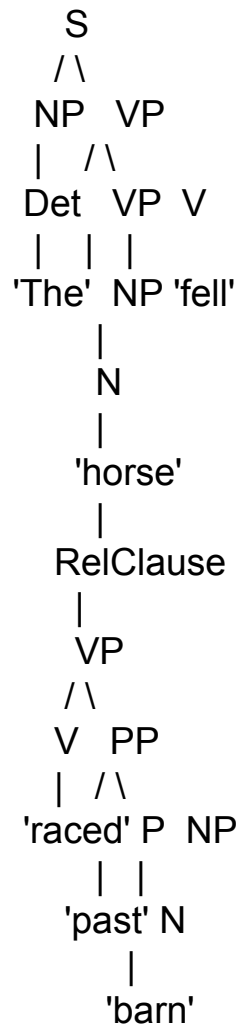
Det -> 'The'

N -> 'horse' | 'barn'

V -> 'raced' | 'fell' | 'past'

P -> 'past'





In this interpretation, "raced past the barn" acts as a reduced relative clause modifying "horse," and "fell" is the main verb.

Newspaper headline

- The newspaper headline "Tornado Touches Down in Cemetery—Hundreds Dead"

Sequence labeling

Sequence labelling

- In natural language processing, it is a common task to extract words or phrases of particular types from a given sentence or paragraph.

For example, when performing analysis of a corpus of news articles, we may want to know which countries are mentioned in the articles, and how many articles are related to each of these countries.

- This is actually a special case of sequence labelling in NLP (others include POS tagging and Chunking), in which the goal is to assign a label to each member in the sequence.
- **input = ["Paris", "is", "the", "capital", "of", "France"]**
- **output = ["I", "I", "I", "I", "I", "C"]**

where **I** means that the token of that position is an irrelevant word, and **C** means that the token of that position is a word that form part of a country's name.

Methods of Sequence Labelling

- A simple, though sometimes quite useful, approach is to prepare a dictionary of country names, and look for these names in each of the sentences in the corpus.
- **Hidden Markov Model (HMM)**
- **Maximum Entropy**
- **Conditional Random Field (CRF)**

Conditional Random Fields (CRFs)

- CRFs are discriminative models, meaning they directly model the conditional probability of the label sequence given the observed sequence.
- CRFs focus on maximizing $P(Y|X)$, where X is the sequence of observations and Y is the sequence of labels.
- They do not model the distribution of the observations X but instead focus on how the labels Y are related to the observations.

CRF

A linear chain CRF confers to a labeler in which tag assignment(for present word, denoted as y_i) depends only on the tag of just one previous word(denoted by y_{i-1}).

CRF:

$$p_{\theta}(y|x) = \frac{\exp(\sum_j w_j F_j(x, y))}{\sum_{y'} \exp(\sum_j w_j F_j(x, y'))}$$

, where $F_j(x, y) = \sum_{i=1}^L f_j(y_{i-1}, y_i, x, i)$

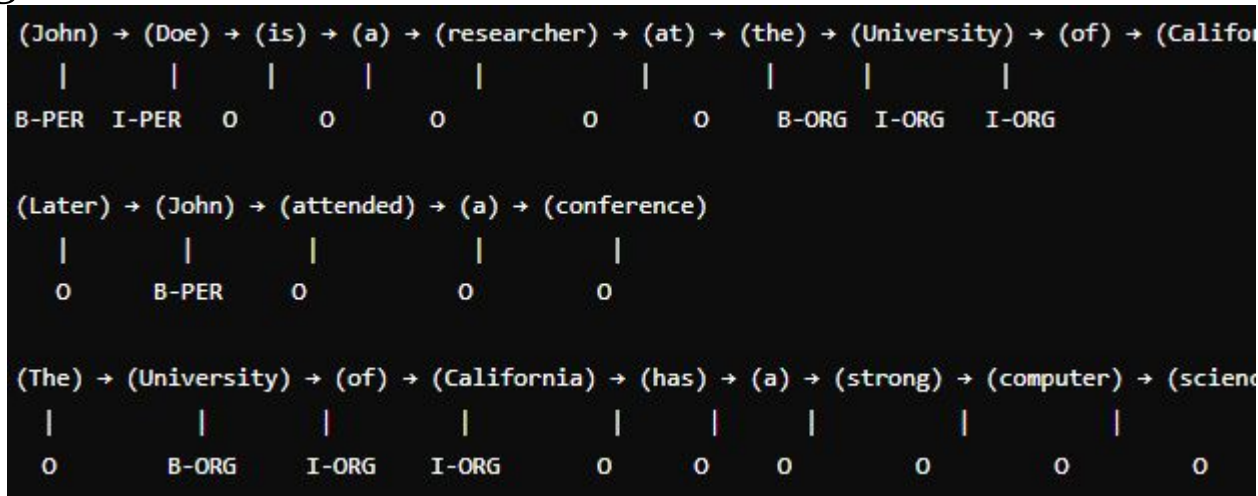
Here,

$p_{\theta}(y|x)$ refers to the probability of calculating a Label sequence(y) given a word sequence(x).

- In a standard CRF, the model would look like a linear chain, capturing dependencies between adjacent labels.

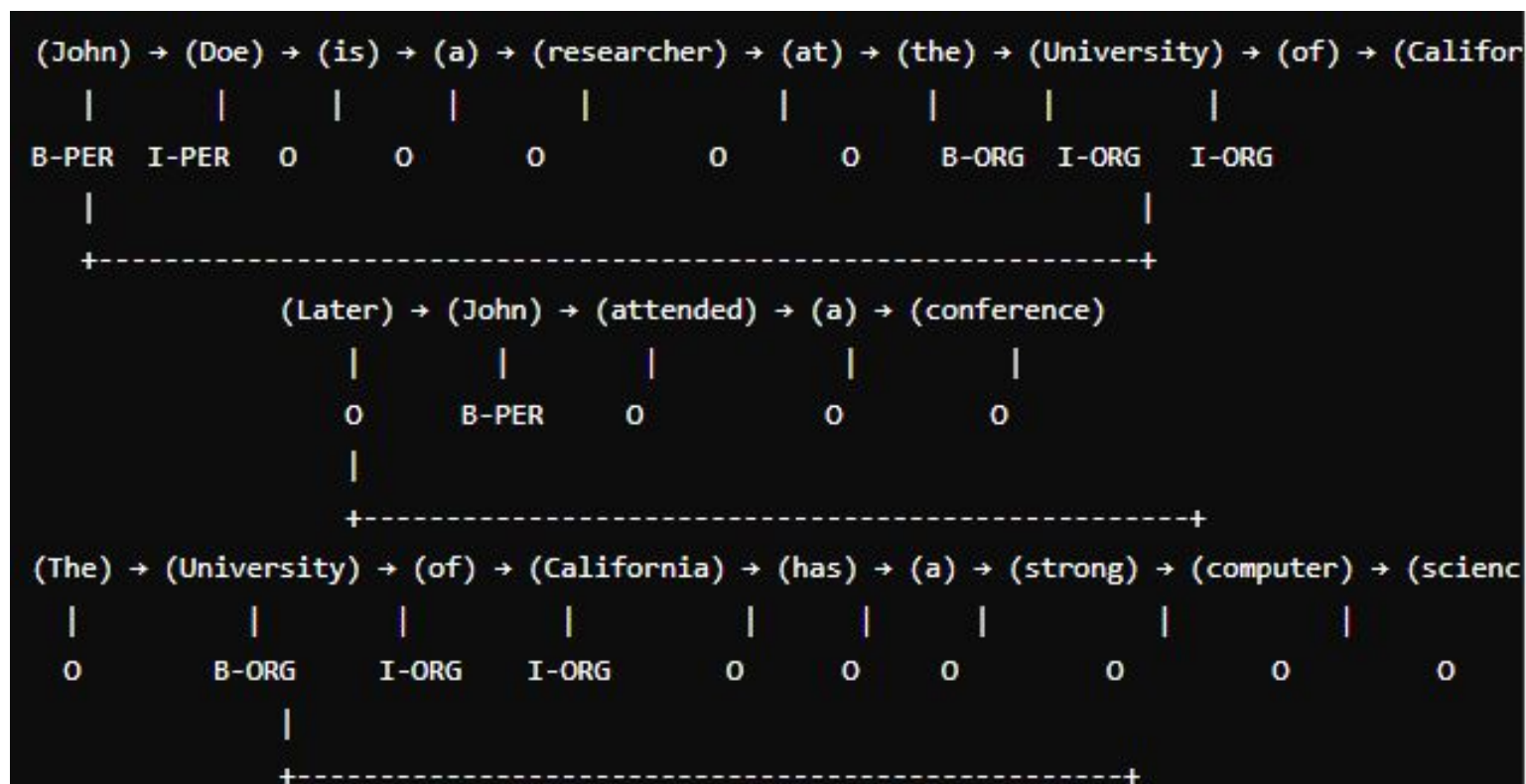
NER:

- "John Doe is a researcher at the University of California."
- "Later, John attended a conference."
- "The University of California has a strong computer science program."



Skip-chain CRF

- additional edges are introduced to capture long-range dependencies, such as connecting the two mentions of "John" and the two mentions of "University of California."



Applications of CRF

- named-entity recognition , feature induction for NER, shallow parsing , identifying protein names in biology abstracts, segmenting addresses in Web pages, information integration , finding semantic roles in text

named entity recognition dataset CoNLL 2003

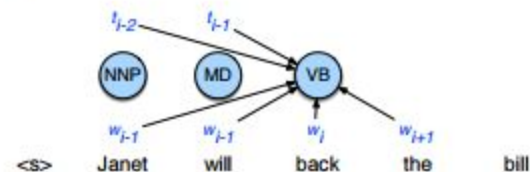
(<https://paperswithcode.com/dataset/conll-2003>)data consists of eight files covering two languages: English and German

Maxent $P(\text{tag}|\text{word})$

- Can do surprisingly well just looking at a word by itself:
 - Word the: **the** DT
 - Prefixes unfathomable: **un-** JJ
 - Suffixes Importantly: **-ly** RB
 - Capitalization Meridian: **CAP** NNP
 - Word shapes 35-year: **d-x** JJ
- Then build a classifier to predict tag
 - Maxent $P(\text{tag}|\text{word})$: 93.7% overall / 82.6% unknown

MEMMs

- Maximum Entropy Markov Model
- A sequence version of the maximum entropy classifier.



Slide adapted from Dan Jurafsky

Slide adapted from Dan Jurafsky

- w_i contains a particular prefix (from all prefixes of length ≤ 4)
- w_i contains a particular suffix (from all suffixes of length ≤ 4)
- w_i contains a number
- w_i contains an upper-case letter
- w_i contains a hyphen
- w_i is all upper case
- w_i 's word shape
- w_i 's short word shape
- w_i is upper case and has a digit and a dash (like *CFC-12*)
- w_i is upper case and followed within 3 words by Co., Inc., etc.

MEMM Decoding

- Simplest algorithm
 - Greedy: at each step in sequence, select tag that maximizes $P(\text{tag} \mid \text{nearby words, nearby tags})$
- In practice
 - Viterbi algorithm
 - Beam search

POS Tagging Accuracies

- Rough accuracies:
 - Baseline: most freq tag: ~90%
 - Trigram HMM: ~95%
 - Maxent $P(t|w)$: 93.7%
 - MEMM tagger: 96.9%
 - Bidirectional MEMM: 97.2%
 - Upper bound: ~98% (human agreement)

Slide adapted from Dan Jurafsky



Arbitrary-Oriented Scene Text Detection via Rotation Proposals.....

International Conference on Document Analysis and Recognition (ICDAR): This is a prominent global conference series dedicated to the research and development of document analysis and recognition technologies

A Hybrid Approach to Detect and Localize Texts in Natural Scene Images

Exploring the Use of Conditional Random Field Models and HMMs for
Historical Handwritten Document Recognition

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Det NOM$

$NOM \rightarrow Noun$

$NOM \rightarrow Noun NOM$

$VP \rightarrow Verb$

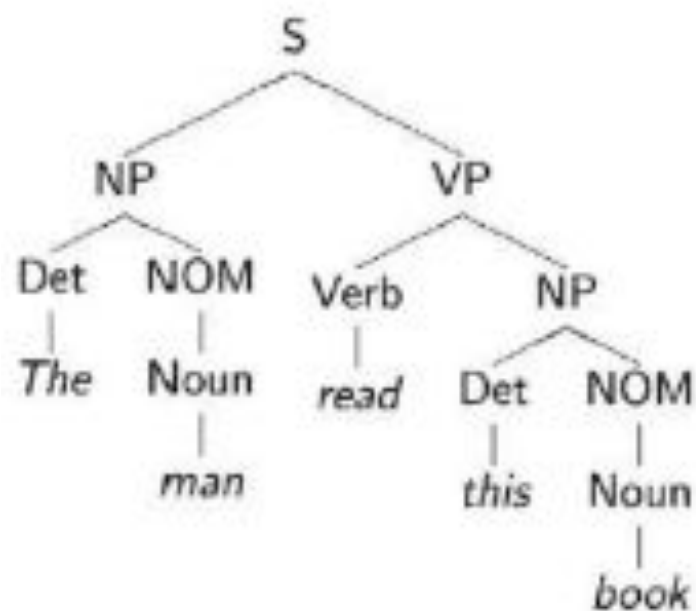
$VP \rightarrow Verb NP$

$Det \rightarrow that \mid this \mid a \mid the$

$Noun \rightarrow book \mid flight \mid meal \mid man$

$Verb \rightarrow book \mid include \mid read$

$Aux \rightarrow does$



Noun Phrase

NP -> Pronoun

- I came, you saw it, they conquered

• NP -> Proper-Noun

- **New Jersey** is west of **New York City**

- **Lee Bollinger** is the president of **Columbia**

• NP -> Det Noun

- The president

• NP -> Nominal

• Nominal -> Noun Noun

- A morning flight to Denver

What other types of nominals do you find in English? Give examples

Example sentence

- " ^ *The tortoise beat the hare in the race.* "

*Guided
by frequency*

*Guided by
Language
Knowledge*

*Guided by
world
Knowledge*

N-gram (n=3)	CFG	Probabilistic CFG	Dependency Grammar	Prob. DG
^ the tortoise $5 \cdot 10^{-3}$	S-> NP VP	S->NP VP 1.0	Semantic Roles <i>agt, obj, sen,</i> <i>etc.</i> Semantic Rules are always between "Heads"	Semantic Roles with probabilities
the tortoise beat $3 \cdot 10^{-2}$	NP->DT N	NP->DT N 0.5		
tortoise beat the $7 \cdot 10^{-5}$	VP->V NP PP	VP->V NP PP 0.4		
beat the hare $5 \cdot 10^{-6}$	PP-> P NP	PP-> P NP 1.0		

Thank you

Links

- <https://medium.com/data-science-in-your-pocket/named-entity-recognition-ner-using-conditional-random-fields-in-nlp-3660df22e95c>
- “Part of Speech (POS) Tagging, Viterbi Algorithm, Solved Problem, Natural Language Processing” <https://www.youtube.com/watch?v=OBemI2BapE0>