Dr. Cengiz Günay, Emory Univ.

"SO HE GOES "LIKE I'M SURE" AND I'M LIKE... YOU KNOW... "I DON'T THINK SO"... AND THEN HE'S ALL "OH, RIGHT."

WISE ALDRICH
So Probabilities Enough for Understanding Language?

He came from out of nowhere.
He came from out of nowhere. From out of nowhere, he came.
He came from out of nowhere. From out of nowhere, he came.

- Same meaning but different ordering: non-Markovian.
- How do we understand that both sentences have similar meaning?
He came from out of nowhere.
From out of nowhere, he came.

- Same meaning but different ordering: non-Markovian.
- How do we understand that both sentences have similar meaning?
- Look at sentence structure: “from out of nowhere” and “he came”
He came from out of nowhere. From out of nowhere, he came.

- Same meaning but different ordering: non-Markovian.
- How do we understand that both sentences have similar meaning?
- Look at sentence structure: “from out of nowhere” and “he came”

Today:
1. Using sentence structure in NLP
2. Machine translation
3. Speech recognition (no time, see textbook)
Exit survey: Natural Language Processing I

- What is a good method for identifying foreign languages?
- How do we improve bag of words to learn word sequences?

Entry survey: Natural Language Processing II (0.25 pts)

- Give some examples of why learning sentence structure may be useful.
- What was the most useful machine translation tool you ever used?
Uses of Sentence Structure in NLP

Can be useful for:

- Disambiguation of phrases
Uses of Sentence Structure in NLP

Can be useful for:
- Disambiguation of phrases
- Understanding meaning
Uses of Sentence Structure in NLP

Can be useful for:

- Disambiguation of phrases
- Understanding meaning
- Translation
Strike a match.
Disambiguation

Strike a match.
Disambiguation

Strike a match.
How Can We Use the Sentence Structure?

Hint:
How Can We Use the Sentence Structure?

Hint: Strike a match
How Can We Use the Sentence Structure?

Hint:

Verb Phrase

Noun Phrase

Verb

Noun

Noun

Strike

a

match
How Can We Use the Sentence Structure?

Hint:

Verb Phrase

Noun Phrase

Verb

Noun

Noun

Strike a match

Noun

Noun

Noun

Noun Phrase
Where Do the Trees Come From?

From the forest?

Seriously, from:

The grammar:

\[ S \rightarrow \text{VP} | \text{NP} \]

\[ \text{VP} \rightarrow \text{V NP} | \text{V} \]

\[ \text{NP} \rightarrow \text{N} | \text{N N} | \text{N N N} \]

\[ \text{N} \rightarrow \text{strike} | \text{match} \]

\[ \text{V} \rightarrow \text{strike} | \text{match} \]

Results in multiple possible parses of the same sentence.
Where Do the Trees Come From?

From the forest?

S → VP | NP
VP → V NP | V
NP → N | N N | N N N
N → strike | match
V → strike | match

Results in multiple possible parses of the same sentence.
Where Do the Trees Come From?

From the forest?

Seriously, from:

The grammar:

\[
\begin{align*}
S & \rightarrow VP|NP \\
VP & \rightarrow V NP|V \\
NP & \rightarrow N|N N|N N N \\
N & \rightarrow \text{strike}|\text{match} \\
V & \rightarrow \text{strike}|\text{match}
\end{align*}
\]
Where Do the Trees Come From?

From the forest?
Seriously, from:

The grammar:

\[
\begin{align*}
S & \rightarrow VP | NP \\
VP & \rightarrow V NP | V \\
NP & \rightarrow N | N N | N N N \\
N & \rightarrow \text{strike} | \text{match} \\
V & \rightarrow \text{strike} | \text{match}
\end{align*}
\]

Results in **multiple possible parses** of the same sentence.
Multiple Possible Parsleys

Parses, parsings, or parsleys (whatever)

"strike a match" can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?
1. Omitting a good parsley (false negative): #1 above
2. Including a bad parsley (false positive): #2 or #3 above

Solutions?
1. Use probabilities
2. Use word associations
3. Unambiguous grammar
Multiple Possible Parsleys

Parses, parsings, or parseys (whatever)

"strike a match" can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?

1. Omitting a good parsley (false negative): #1 above
2. Including a bad parsley (false positive): #2 or #3 above

Solutions?

1. Use probabilities
2. Use word associations
3. Unambiguous grammar
Multiple Possible Parsleys

Parses, parsings, or parseleys (whatever)

“strike a match” can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?
Multiple Possible Parsleys

Parses, parsings, or parseleys (whatever)

"strike a match" can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?

1. Omitting a good parsley (false negative): #1 above
Multiple Possible Parsleys

Parses, parsings, or parsleys (whatever)

“strike a match” can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?

1. Omitting a good parsley (false negative): #1 above
2. Including a bad parsley (false positive): #2 or #3 above
Multiple Possible Parsleys

Parses, parsings, or parses, or parses, or parsleys (whatever)

“strike a match” can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?

1. Omitting a good parsley (false negative): #1 above
2. Including a bad parsley (false positive): #2 or #3 above

Solutions?

1. Use probabilities
2. Use word associations
3. Unambiguous grammar
Multiple Possible Parsleys

Parses, parsings, or parleys (whatever)

“strike a match” can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?

1. Omitting a good parsley (false negative): #1 above
2. Including a bad parsley (false positive): #2 or #3 above

Solutions?

1. Use probabilities
2. Use word associations
3. Unambiguous grammar
Multiple Possible Parsleys

Parses, parsings, or parseys (whatever)

“strike a match” can be parsed as:

1. verb noun noun
2. noun noun noun
3. noun noun verb

Problems?

1. Omitting a good parsley (false negative): #1 above
2. Including a bad parsley (false positive): #2 or #3 above

Solutions?

1. Use probabilities
2. Use word associations
3. Unambiguous grammar
context-free grammar: Words are expanded without context (e.g., $S \rightarrow VP|NP$). Used with programming languages.
context-free grammar: Words are expanded without context
(e.g., $S \rightarrow VP|NP$). Used with programming languages.

“strike a match”

The probabilistic grammar:

$S \rightarrow VP(0.7)|NP(0.3)$
context-free grammar: Words are expanded without context (e.g., $S \rightarrow VP|NP$). Used with programming languages.

“strike a match”

The probabilistic grammar:

- $S \rightarrow VP(0.7)|NP(0.3)$
- $VP \rightarrow V NP(0.6)|V(0.4)$
context-free grammar: Words are expanded without context (e.g., $S \rightarrow VP|NP$). Used with programming languages.

“strike a match”

The probabilistic grammar:

$S \rightarrow VP(0.7)|NP(0.3)$
$VP \rightarrow V\,NP(0.6)|V(0.4)$
$NP \rightarrow N(0.6)|NN(0.3)|NNN(0.1)$
context-free grammar: Words are expanded without context (e.g., $S \rightarrow VP|NP$). Used with programming languages.

“strike a match”

The probabilistic grammar:

- $S \rightarrow VP(0.7)|NP(0.3)$
- $VP \rightarrow V \ NP(0.6)|V(0.4)$
- $NP \rightarrow N(0.6)|N N(0.3)|N N N(0.1)$
- $N \rightarrow strike(0.4)|match(0.7)$
Use Probabilities and Grammar Together

**context-free grammar:** Words are expanded without context (e.g., $S \rightarrow VP|NP$). Used with programming languages.

“strike a match”

**The probabilistic grammar:**

- $S \rightarrow VP(0.7)|NP(0.3)$
- $VP \rightarrow V\,NP(0.6)|V(0.4)$
- $NP \rightarrow N(0.6)|N\,N(0.3)|N\,N\,N(0.1)$
- $N \rightarrow strike(0.4)|match(0.7)$
- $V \rightarrow strike(0.6)|match(0.3)$
context-free grammar: Words are expanded without context (e.g., $S \rightarrow VP|NP$). Used with programming languages.

“strike a match”

The probabilistic grammar:

\[
\begin{align*}
S & \rightarrow VP(0.7)|NP(0.3) \\
VP & \rightarrow V \, NP(0.6)|V(0.4) \\
NP & \rightarrow N(0.6)|N \, N(0.3)|N \, N \, N(0.1) \\
N & \rightarrow \text{strike}(0.4)|\text{match}(0.7) \\
V & \rightarrow \text{strike}(0.6)|\text{match}(0.3)
\end{align*}
\]

It’s called a probabilistic context-free grammar (PCFG)
The probabilistic grammar:

\[
S \rightarrow \text{VP}(0.7) | \text{NP}(0.3)
\]

\[
\text{VP} \rightarrow \text{V NP}(0.6) | \text{V}(0.4)
\]

\[
\text{NP} \rightarrow \text{N}(0.6) | \text{N N}(0.3) | \text{N N N}(0.1)
\]

\[
\text{N} \rightarrow \text{strike}(0.4) | \text{match}(0.7)
\]

\[
\text{V} \rightarrow \text{strike}(0.6) | \text{match}(0.3)
\]
The probabilistic grammar:

\[ S \rightarrow \text{VP}(0.7) | \text{NP}(0.3) \]
\[ \text{VP} \rightarrow V \text{NP}(0.6) | V(0.4) \]
\[ \text{NP} \rightarrow N(0.6) | N N(0.3) | N N N(0.1) \]
\[ N \rightarrow \text{strike}(0.4) | \text{match}(0.7) \]
\[ V \rightarrow \text{strike}(0.6) | \text{match}(0.3) \]
PCFG Example

The probabilistic grammar:

S \rightarrow VP(0.7) | NP(0.3)
VP \rightarrow V NP(0.6) | V(0.4)
NP \rightarrow N(0.6) | N N(0.3) | N N N(0.1)
N \rightarrow strike(0.4) | match(0.7)
V \rightarrow strike(0.6) | match(0.3)
The probabilistic grammar:

\[
S \rightarrow VP(0.7)|NP(0.3)
\]

\[
VP \rightarrow V NP(0.6)|V(0.4)
\]

\[
NP \rightarrow N(0.6)|NN(0.3)|NNN(0.1)
\]

\[
N \rightarrow strike(0.4)|match(0.7)
\]

\[
V \rightarrow strike(0.6)|match(0.3)
\]

\[
P(\text{Verb Phrase}) = 0.0756
\]

\[
P(\text{Noun Phrase}) = 0.0084
\]
How to Get Grammar Probabilities?

I made them up :)
Can we count them?

First need a model of grammar, but problems:
- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?
- Machine learning

But where's the data?
- Need to pay people to build databases (e.g., Penn Tree Bank)

Can you think of a better solution?
How to Get Grammar Probabilities?

I made them up :)  
Can we count them? No, they are ambiguous out in the wild.
I made them up :)
Can we count them? **No**, they are ambiguous out in the wild.
First need a model of grammar, but problems:
I made them up :)  
Can we count them? **No**, they are ambiguous out in the wild. 
First need a model of grammar, but problems:

- Grammars are biologically evolved
How to Get Grammar Probabilities?

I made them up :)  
Can we count them? No, they are ambiguous out in the wild.  
First need a model of grammar, but problems: 

- Grammars are biologically evolved  
- They are complex and rough
How to Get Grammar Probabilities?

I made them up :)

Can we count them? No, they are ambiguous out in the wild.

First need a model of grammar, but problems:

- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?

Machine learning

But where's the data?

Need to pay people to build databases (e.g., Penn Tree Bank)

Can you think of a better solution?

Understand context first?
How to Get Grammar Probabilities?

I made them up :) Can we count them? **No**, they are ambiguous out in the wild. First need a model of grammar, but problems:

- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?
How to Get Grammar Probabilities?

I made them up :)  
Can we count them? No, they are ambiguous out in the wild. 
First need a model of grammar, but problems:

- Grammars are biologically evolved  
- They are complex and rough  
- Neat rules all have exceptions

Solution?  
- Machine learning
How to Get Grammar Probabilities?

I made them up :) Can we count them? No, they are ambiguous out in the wild. First need a model of grammar, but problems:

- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?

- Machine learning

But where’s the data?
How to Get Grammar Probabilities?

I made them up :)  
Can we count them? No, they are ambiguous out in the wild.  
First need a model of grammar, but problems:

- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?

- Machine learning

But where’s the data?

- Need to pay people to build databases (e.g., Penn Tree Bank)
How to Get Grammar Probabilities?

I made them up :) Can we count them? **No**, they are ambiguous out in the wild. First need a model of grammar, but problems:

- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?

- Machine learning

But where’s the data?

- Need to pay people to build databases (e.g., Penn Tree Bank)

Can you think of a better solution?
How to Get Grammar Probabilities?

I made them up :) Can we count them? **No**, they are ambiguous out in the wild. First need a model of grammar, but problems:

- Grammars are biologically evolved
- They are complex and rough
- Neat rules all have exceptions

Solution?

- Machine learning

But where’s the data?

- Need to pay people to build databases (e.g., Penn Tree Bank)

Can you think of a better solution?

- Understand context first?
Example Grammar

(S
  (NP-SBJ (DT The) (NN move))
  (VP (VBD followed)
    (NP
      (NP (DT a) (NN round))
      (PP (IN of)
        (NP
          (NP (JJ similar) (NNS increases))
          (PP (IN by)
            (NP (JJ other) (NNS lenders))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans)))))
      (, ,))
    (S-ADV
      (NP-SBJ (-NONE- *))
      (VP (VBG reflecting)
        (NP
          (NP (DT a) (VBG continuing) (NN decline))
          (PP-LOC (IN in)))))}
Back to Disambiguation with Learned Grammar

Lexicalized grammar: Probabilities of where words belong (can get help from dictionaries).

NP
  | PRP
  | saw
  | V
  | NP
  | with
  | DT
  | NN

VP
  | NP
  | saw
  | the
  | man
  | NP
  | a
  | telescope

S
  | NP
  | VP
  | PP
  | a
  | telescope

S
  | NP
  | VP
  | PP

Natural Language Processing II (Ch. 23) Spring 2013 13 / 18
Back to Disambiguation with Learned Grammar

Lexicalized grammar: Probabilities of where words belong (can get help from dictionaries).

Günay ()

Natural Language Processing II (Ch. 23)

Spring 2013 13 / 18
Lexicalized PCFG (LPCFG)

OMG! That’s a long acronym.

Probabilities based on actual words:

\[ P(\text{VP} \rightarrow \text{V NP NP} | \text{V} = \text{gave}) = 0.8 \text{ (common: gave me something)} \]

\[ P(\text{VP} \rightarrow \text{V NP NP} | \text{V} = \text{kiss}) = 0.1 \text{ (rare: kiss me goodbye)} \]

But telescope example still hard to solve. But we can use:

Smoothing Abstractions
Lexicalized PCFG (LPCFG)

OMG! That’s a long acronym.

Probabilities based on actual words:

\[ P(\text{VP} \rightarrow \text{V NP NP}|\text{V} = \text{gave}) = 0.8 \] (common: gave me something)

\[ P(\text{VP} \rightarrow \text{V NP NP}|\text{V} = \text{kiss}) = 0.1 \] (rare: kiss me goodbye)
OMG! That’s a long acronym.

Probabilities based on actual words:

\[ P(\text{VP} \rightarrow V \text{ NP NP}|V = \text{gave}) = 0.8 \text{ (common: gave me something)} \]
\[ P(\text{VP} \rightarrow V \text{ NP NP}|V = \text{kiss}) = 0.1 \text{ (rare: kiss me goodbte)} \]

But telescope example still hard to solve. But we can use:

- Smoothing
- Abstractions
So we have all the information now. How to parse language into trees?
So we have all the information now. How to parse language into trees?

Two options:

1. Start from words (bottom up); like starting from initial state

Context-free grammars have advantage of parsing parts of the tree independent of the rest. That is, we can divide and conquer.
So we have all the information now. How to parse language into trees? Two options:

1. Start from words (bottom up); like starting from initial state
2. Start from sentence (top down); like starting from goal state
So we have all the information now. How to parse language into trees? Two options:

1. Start from words (bottom up); like starting from initial state
2. Start from sentence (top down); like starting from goal state

So it becomes like a regular tree search!
So we have all the information now. How to parse language into trees?

Two options:

1. Start from words (bottom up); like starting from initial state
2. Start from sentence (top down); like starting from goal state

So it becomes like a regular tree search!

Note:

- Context-free grammars have advantage of parsing parts of the tree independent of the rest. That is, we can divide and conquer.
The Penn Treebank Project annotates naturally-occurring text for linguistic structure. Most notably, we produce skeletal parses showing rough syntactic and semantic information -- a bank of linguistic trees.
The Penn Treebank Project annotates naturally-occurring text for linguistic structure. Most notably, we produce skeletal parses showing rough syntactic and semantic information -- a bank of linguistic trees.
The Penn Treebank Project annotates naturally-occurring text for linguistic structure. Most notably, we produce skeletal parses showing rough syntactic and semantic information — a bank of linguistic trees.

Penn Treebank Projesi dil yapısı için doğal olarak oluşan metin not alınır. Dil ağaçları bir banka - En önemlisi, biz iskelet kaba söz dizimsel ve semantik bilgilerini gösteren aynıştırır üretmek.

Penn Treebank Project annotates language to the structure of naturally occurring text. Language trees, a bank - Most importantly, we produce skeletal parses showing rough syntactic and semantic information.
Multi-level pyramid of machine translation (by Vauquois):

1. Word by word
2. Phrase
3. Tree
4. Meaning (semantic)
Multi-level pyramid of machine translation (by Vauquois):

1. Word by word
2. **Phrase**
3. Tree
4. Meaning (semantic)

We’ll concentrate on #2, but others are used on the field, too.
Phrase Translation

$$P(e|g) = P(\{g_i\}|g) \prod_i P(\bar{e}_i|\bar{g}_i) P(a_i - b_{i-1})$$

- Segmentation
- Translation
- Distortion
Calculate $p(e)$ from LPCFG and check if translated sentence is likely.

What else to improve?
What else to improve?

- Calculate $p(e)$ from LPCFG and check if translated sentence is likely.