Tampere University

Natural Language for Communication

CHAPTER 23 IN THE TEXTBOOK

Tampere University

Communication

- The intentional exchange of information brought about by the production and perception of signs drawn from a shared system of conventional signs
- Most animals use signs to represent important messages: food here, predator nearby, approach, withdraw, let's mate.
- In a partially observable world, an agent can learn information that is observed or inferred by others
- Humans are the most chatty of all species, and if computer agents are to be helpful, they'll need to learn to speak the language.
- We look at language models for communication.
- Models aimed at deep understanding of a conversation necessarily need to be more complex than the simple models aimed at, say, spam classification.
- We start with grammatical models of the phrase structure of sentences, add semantics to the model, and then apply it to machine translation and speech recognition.

Phrase Structure Grammars

- The n-gram language models are based on sequences of words
- The big issue for these models is data sparsity
  - with a vocabulary of, say, 10^5 words, there are 10^{15} trigram probabilities to estimate
- A corpus of even a trillion words will not be able to supply reliable estimates for all of them
- Tackle the problem of sparsity through generalization
- “black dog” is more frequent than “dog black” + similar observations, generalize that adjectives tend to come before nouns in English (also exceptions)
- The notion of a **lexical category** (also known as a **part of speech**) such as *noun* or *adjective* is a useful generalization as such,
- Even more so when we string together lexical categories to form **syntactic categories** such as *noun phrase* or *verb phrase*, and combine them into trees representing the phrase structure of sentences: nested phrases, each marked with a category

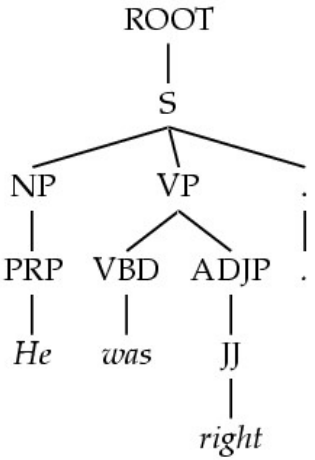
Tampere University

Probabilistic context-free grammar (PCFG)

- A grammar is a collection of rules that defines a language as a set of allowable strings of words
- “Context-free grammars” (or CFGs): the left-hand side consists of a single nonterminal symbol
- Each rule licenses rewriting the nonterminal as the right-hand side in any context
- “probabilistic” means that the grammar assigns a probability to every string. Here is a PCFG rule:

$$\text{VP} \rightarrow \text{Verb}\ [0.70]$$
$$\mid \text{VP NP}\ [0.30]\ .$$

- VP (verb phrase) and NP (noun phrase) are **non-terminal symbols**
- The grammar also refers to actual words, which are called **terminal symbols**
- This rule is saying that with probability 0.70 a verb phrase consists solely of a verb, and with probability 0.30 it is a VP followed by an NP
- We now define a grammar for a tiny fragment of English that is suitable for communication between agents exploring the wumpus world.
- We call this language $\mathcal{E}_0$
- We are unlikely ever to devise a complete grammar for English



Tampere University

The lexicon of $\mathcal{E}_0$

- Lexicon = list of allowable words
- Lexical categories: nouns, pronouns, and names to denote things; verbs to denote events; adjectives to modify nouns; adverbs to modify verbs; and function words:
  - articles (such as the),
  - prepositions (in), and
  - conjunctions (and)
- For nouns, names, verbs, adjectives, and adverbs, it is infeasible even in principle to list all the words
  - There are tens of thousands of members in each class, and new ones–like iPod or biodiesel—are being added constantly
  - These five categories are **open classes**
- For the categories of pronoun, relative pronoun, article, preposition, and conjunction we could have listed all the words with a little more work
  - These are **closed classes**; they have a small number of words (a dozen or so)
  - Closed classes change over the course of centuries, not months
  - For example, "thee" and "thou" were commonly used pronouns in the 17th century

Tampere University

- Noun → **stench** [0.05] | **breeze** [0.10] | **wumpus** [0.15] | **pits** [0.05] | . . .
- Verb → **is** [0.10] | **feel** [0.10] | **smells** [0.10] | **stinks** [0.05] | . . .
- Adjective → **right** [0.10] | **dead** [0.05] | **smelly** [0.02] | **breezy** [0.02] . . .
- Adverb → **here** [0.05] | **ahead** [0.05] | **nearby** [0.02] | . . .
- Pronoun → **me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] | . . .
- RelPro → **that** [0.40] | **which** [0.15] | **who** [0.20] | **whom** [0.02] ∨ . . .
- Name → **John** [0.01] | **Mary** [0.01] | **Boston** [0.01] | . . .
- Article → **the** [0.40] | **a** [0.30] | **an** [0.10] | **every** [0.05] | . . .
- Prep → **to** [0.20] | **in** [0.10] | **on** [0.05] | **near** [0.10] | . . .
- Conj → **and** [0.50] | **or** [0.10] | **but** [0.20] | **yet** [0.02] ∨ . . .
- Digit → **0** [0.20] | **1** [0.20] | **2** [0.20] | **3** [0.20] | **4** [0.20] | . . .

Tampere University

The Grammar of $\mathcal{E}_0$

- The next step is to combine the words into phrases
- Next slide shows a grammar for $\mathcal{E}_0$, with rules for each of the six syntactic categories and an example for each rewrite rule
- The following slide shows a **parse tree** for the sentence “Every wumpus smells.”
- It shows that the string of words is indeed a sentence according to the rules of $\mathcal{E}_0$
- The $\mathcal{E}_0$ grammar generates a wide range of English sentences such as the following:

  John is in the pit

  The wumpus that stinks is in 2 2

  Mary is in Boston and the wumpus is near 3 2

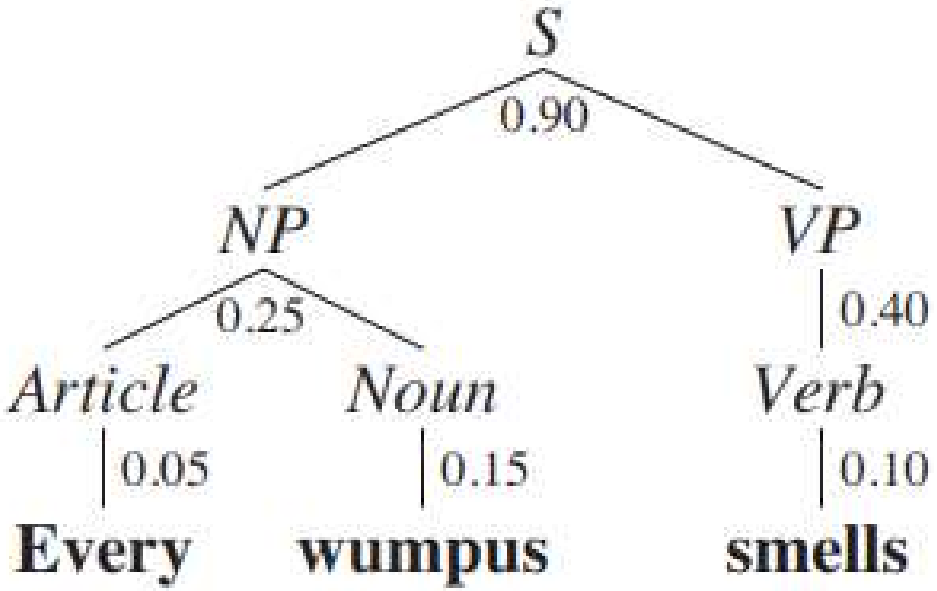
- Unfortunately, the grammar overgenerates: it generates sentences that are not grammatical, such as “Me go Boston” and “I smell pits wumpus John.”
- It also undergenerates: it rejects sentences of English, such as “I think the wumpus is smelly.”

Tampere University

$\mathcal{E}_0$:	S → NP VP [0.90]	I + feel a breeze
	| S Conj S [0.10]	I feel a breeze + and + It stinks
	NP → Pronoun [0.30]	I
	| Name [0.10]	John
	| Noun [0.10]	pits
	| Article Noun [0.25]	the + wumpus
	| Article Adjs Noun [0.05]	the + smelly dead + wumpus
	| Digit Digit [0.05]	3 4
	| NP PP [0.10]	the wumpus + in 1 3
	| NP RelClause [0.05]	the wumpus + that is smelly
	VP → Verb [0.40]	stinks
	| VP NP [0.35]	feel + a breeze
	| VP Adjective [0.05]	smells + dead
	| VP PP [0.10]	is + in 1 3
	| VP Adverb [0.10]	go + ahead
	Adjs → Adjective [0.80]	smelly
	| Adjective Adjs [0.20]	smelly + dead
	PP → Prep NP [1.00]	to + the east
	RelClause → RelPro VP [1.00]	that + is smelly

Tampere University

Parse tree for ”every wumpus smells”



Syntactic Analysis (Parsing)

- Analyze a string of words to uncover its phrase structure, according to the rules of a grammar
- We can start with the S symbol and search top down for a tree that has the words as its leaves
- We can also start with the words and search bottom up for a tree that culminates in an S
- Both top-down and bottom-up parsing can be inefficient, however, because they can end up repeating effort in areas of the search space that lead to dead ends
- Consider the following two sentences:

  Have the students in section 2 of Computer Science 101 take the exam.

  Have the students in section 2 of Computer Science 101 taken the exam?
- They share the first 10 words, but have very different parses, because the first is a command and the second is a question
- A left-to-right parsing algorithm would have to guess whether the first word is part of a command or a question and will not be able to tell if the guess is correct until at least the eleventh word, take or taken
- If the algorithm guesses wrong, it will have to backtrack all the way to the first word and reanalyze the whole sentence under the other interpretation

Tampere University

List of items	Rule
S	
NP VP	S → NP VP
NP VP Adjective	VP → VP Adjective
NP Verb Adjective	VP → Verb
NP Verb **dead**	Adjective → **dead**
NP **is dead**	Verb → **is**
Article Noun **is dead**	NP → Article Noun
Article **wumpus is dead**	Noun → **wumpus**
the wumpus is dead	Article → **the**

- We can use dynamic programming to avoid this source of inefficiency: every time we analyze a substring, store the results so we won't have to reanalyze it later
- For example, once we discover that "the students in section 2 of Computer Science 101" is an NP, we can record that result in a data structure known as a chart.
- Algorithms that do this are called chart parsers
- Because we are dealing with CFGs, any phrase that was found in the context of one branch of the search space can work just as well in any other branch of the search space
- There are many types of chart parsers; we describe a bottom-up version called the CYK algorithm, after its inventors, Cocke, Younger, and Kasami.

The CYK algorithm

- The CYK algorithm requires a grammar with all rules in one of two very specific formats: lexical rules of the form $X \rightarrow word$, and syntactic rules of the form $X \rightarrow Y\ Z$
- Any context-free grammar can be automatically transformed into **Chomsky Normal Form**
- The CYK algorithm uses space of $O(n^2 m)$ for the P table, where n is the number of words in the sentence, and m is the number of nonterminals in the grammar, and takes time $O(n^3 m)$
- No algorithm can do better for general context-free grammars, although there are faster algorithms on more restricted grammars
- In fact, it is quite a trick for the algorithm to complete in $O(n^3)$ time, given that it is possible for a sentence to have an exponential number of parse trees.
- Consider the sentence **Fall leaves fall and spring leaves spring**
- It is ambiguous because each word (except “and”) can be either a noun or a verb, and “fall” and “spring” can be adjectives as well
- For example, one meaning of “Fall leaves fall” is equivalent to “Autumn abandons autumn”

- With $\mathcal{E}_0$ the sentence has four parses:

  [S [S [NP **Fall leaves**] **fall**] **and** [S [NP **spring leaves**] **spring**]

  [S [S [NP **Fall leaves**] **fall**] **and** [S **spring** [VP **leaves spring**]]

  [S [S **Fall** [VP **leaves fall**]] **and** [S [NP **spring leaves**] **spring**]

  [S [S **Fall** [VP **leaves fall**]] **and** [S **spring** [VP **leaves spring**]]
- If we had c two-ways-ambiguous conjoined subsentences, we would have 2^c ways of choosing parses for the subsentences
- How does the CYK algorithm process these 2^c parse trees in $O(c^3)$ time?
- It doesn't examine all the parse trees; all it has to do is compute the probability of the most probable tree
- The subtrees are all represented in the P table, and with a little work we could enumerate them all (in exponential time), but the beauty of the CYK algorithm is that we don't have to enumerate them unless we want to

```
function CYK-PARSE(words, grammar) returns P, a table of probabilities
  N ← LENGTH(words)
  M ← the number of nonterminal symbols in grammar
  P ← an array of size [M, N, N], initially all 0
  /* Insert lexical rules for each word */
  for i = 1 to N do
    for each rule of form (X → words_i [p]) do
      P[X, i, 1] ← p
  /* Combine first and second parts of right-hand sides of rules, from short to long */
  for length = 2 to N do
    for start = 1 to N − length + 1 do
      for len1 = 1 to N − 1 do
        len2 ← length − len1
        for each rule of the form (X → Y Z [p]) do
          P[X, start, length] ← MAX(P[X, start, length],
                                    P[Y, start, len1] × P[Z, start + len1, len2] × p)
  return P
```

- In practice we are usually not interested in all parses; just the best one or best few
- Think of the CYK algorithm as defining the complete state space defined by the “apply grammar rule” operator
- It is possible to search just part of this space using A^* search
- Each state in this space is a list of items (words or categories), as shown in the bottom-up “The wumpus is dead” parse table
- The start state is a list of words, and a goal state is the single item S
- The cost of a state is the inverse of its probability as defined by the rules applied so far, and there are various heuristics to estimate the remaining distance to the goal; the best heuristics come from machine learning applied to a corpus of sentences
- With the A^* algorithm we don’t have to search the entire state space, and we are guaranteed that the first parse found will be the most probable.

Learning probabilities for PCFGs

- A PCFG has many rules, with a probability for each rule
- Learning the grammar from data might be better than a knowledge engineering approach
- Learning is easiest if we are given a corpus of correctly parsed sentences, commonly called a **treebank**
- The Penn Treebank is the best known
  - consists of 3 million words which have been annotated with part of speech and parse-tree structure, using human labor assisted by some automated tools
- Next slide shows an annotated tree from the Penn Treebank
- Given a corpus of trees, we can create a PCFG just by counting (and smoothing)
- In the example above, there are two nodes of the form [S[NP . . .][VP . . .]]. We would count these, and all the other subtrees with root S in the corpus.
- If there are 100,000 S nodes of which 60,000 are of this form, then we create the rule:

$$S \rightarrow NP\ VP\ [0.60]$$

[[S [NP-SBJ-2 **Her eyes**]
[VP **were**
[VP **glazed**
[NP *-2]
[SBAR-ADV **as if**
[S [NP-SBJ **she**]
[VP **did n't**
[VP [VP **hear** [NP *-1]]
or
[VP [ADVP **even**] **see** [NP *-1]]
[NP-1 **him**]]]]]]]]
.]

- What if a treebank is not available, but we have a corpus of raw unlabeled sentences?
- It is still possible to learn a grammar from such a corpus, but it is more difficult
- We actually have two problems:
  - learning the structure of the grammar rules
  - and learning the probabilities associated with each rule
- We'll assume that we're given the lexical and syntactic category names
- If not, we can just assume categories $X_1, \ldots, X_n$ and use cross-validation to pick the best value of n
- We can then assume that the grammar includes every possible $(X \rightarrow Y\ Z)$ or $(X \rightarrow word)$ rule, although many of these rules will have probability 0 or close to 0
- We can then use an expectation–maximization (EM) approach, just as we did in learning HMMs.
- The parameters we are trying to learn are the rule probabilities; we start them off at random or uniform values.
- The hidden variables are the parse trees: we don't know whether a string of words $w_i, \ldots, w_j$ is or is not generated by a rule (X → . . .).
- The E step estimates
- the probability that each subsequence is generated by each rule. The M step then estimates
- the probability of each rule. The whole computation can be done in a dynamic-programming fashion with an algorithm called the inside–outside algorithm in analogy to the forward–backward algorithm for HMMs.

Tampere University

Comparing context-free and Markov models

- The problem with PCFGs is that they are context-free.
- The difference between P ("$eat\ a\ banana$") and P("$eat\ a\ bandanna$") depends only on $P(Noun \rightarrow$ "$banana$"$)$ vs. $P(Noun \rightarrow$ "$bandanna$"$)$; not on the relation between "eat" and the respective objects
- A Markov model of order two or more, given a sufficiently large corpus, will know that "eat a banana" is more probable
- We can combine a PCFG and Markov model to get the best of both.
- The simplest approach is to estimate the probability of a sentence with the geometric mean of the probabilities computed by both models
- Then we would know that "eat a banana" is probable from both the grammatical and lexical point of view.
- But it still wouldn't pick up the relation between "eat" and "banana" in "eat a slightly aging but still palatable banana" because here the relation is more than two words away.
- Increasing the order of the Markov model won't get at the relation precisely; to do that we can use a lexicalized PCFG, as described in the next section.

- Another problem with PCFGs is that they tend to have too strong a preference for shorter
- sentences
- In a corpus such as the Wall Street Journal, the average length of a sentence is about 25 words
- But a PCFG will usually assign fairly high probability to many short sentences, such as

  “He slept,”
- whereas in the Journal we’re more likely to see something like

  “It has been reported by a reliable source that the

  allegation that he slept is credible.”
- It seems that the phrases in the Journal really are not context-free; instead the writers have an idea of the expected sentence length and use that length as a soft global constraint on their sentences
- This is hard to reflect in a PCFG