Three Models for the Description of Language

Matt Willsey

May 10, 2006

1 Introduction

Noam Chomsky hopes to accomplish two goals with the model developed herein. First, the model must be simple, yet sophisticated enough to generate sentences. Second, he uses the developed model of the language to understand basic, underlying principles to the structure of a language. The model is developed by observing patterns in a finite set of sentences and extended such that it is capable of describing every sentence in the entire language. The author then evaluates the performance of the model by testing it on clear grammatical and ungrammatical sentences. Ideally, the model should be capable of describing grammatical sentences and not capable of describing ungrammatical sentences. For example consider the two sentences as given in [1].

(1) John ate a sandwich.
(2) Sandwich a ate John.

A correct model describing English would is capable of describing (1) but not (2).

In his paper, Noam Chomsky analyzes three models to describe language structure, which are described in more detail in this review. The first model analyzed is a method for generating English sentences with finite-state Markov processes. Conceptually, this method is very simple; but unfortunately, Chomsky concludes using finite-state Markov processes to generate sentences does properly model the English language. Next Chomsky develops the concept of "phrase structure" in the attempt to develop a more powerful model for the English language. By using this model, very simple English sentences can be generated. Unfortunately, this limited model can only generate a small subset of sentences in the complete set of English sentences. However, this model can be generalized to first use phrase structure to generate a basic sentence kernel and then any number of transformations can be applied. Using this extended model, practically all English sentences can be generated. Moreover, this model also provides insight into the basic structure of the English language.

2 Finite State Markov Processes

Before describing finite-state Markov processes, it is necessary to precisely define terms used in Chomsky's paper. First, language is defined as "a set (finite or infinite) of sentences, each of finite length, all constructed from a finite alphabet of symbols [1]." Secondly, for an alphabet, A, a string is any concatenation of the symbols of A. Lastly, grammar of the language, L, is a device that produces all the sentences of L and only the sentences of L.

With these definitions in place, it is possible to describe the first, and conceptually the simplest, model of a language by defining a finite-state grammar, G, using finite-state Markov processes. To do so consider the finite states, $S_0,\ldots,S_q$, a set of transition symbols, $A=\{a_{i,j,k} | 0 \le i, j \le q; 1 \le k \le N_{i,j}$ for each i, j$\}$, and finally a set of connected pairs of states of G, $C = \{(S_i, S_j)\}$, where $N_{i,J}$ is the total number of transition symbols when transitioning from S_i to S_J. As the process moves from one state to the next, it produces a symbol $a_{i,j,k}$ which is contained in A. Thus to produce a sentence in the language (L_G), we run through the states $S_{\alpha_1},\ldots,S_{\alpha_m}$ where $\alpha_1 = \alpha_m = 0$. This sequence of states will yield the string, $s_1 = a_{\alpha_1,\alpha_2,k_1} \wedge a_{\alpha_2,\alpha_3,k_2} \wedge \cdots \wedge a_{\alpha_{m-1},\alpha_m,k_{m-1}}$ for $k_i \le N_{\alpha_i,\alpha_{i+1}}$, where the string is the sentence in L_G.

However, neither this model nor any other finite-state model can generate every possible sentence in the English language. Unfortunately, the strings in English have interdependencies among words. For example, consider the sentences given in (3) where S_1 and S_2 are English strings.

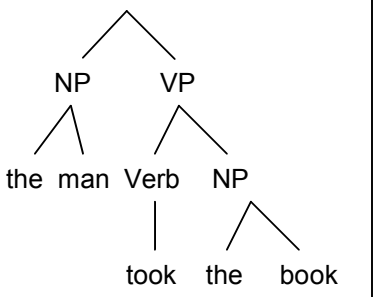
(3)(i) If S_1, then S_2.
(ii) Either S_3, or S_4.
(iii) If either if S_5, then S_6, or S_7, then S_8.

As shown in (3)(i) and (ii), the words "if"-"then" and "either"-"or" are dependent, since every "if" requires a "then" (although it could be an implied "then") and every "either" requires an "or." Moreover, these sentences can be nested an infinite number of times and still form a grammatically-correct English sentence. A case of nesting three dependencies is shown in (3)(iii), and it can be easily extended to nesting an infinite number of dependencies. When nesting dependencies, memory is required to remember the presence of an "if" or "either." Thus, no finite state machine can capture this property of nesting an infinite number of dependencies; therefore, no finite-state machine can completely describe the English language. Chomsky comments that it is not helpful to limit the nesting to finite numbers of dependencies; since once it is assumed that the sentence structure is finite, it is obvious that a finite state machine is capable of describing the structure. However, it still provides little insight into the underlying structure of English, since it cannot provide insight to a set of grammatically correct sentences.

3 Phrase-Structure Grammar

After demonstrating that finite-state Markov processes are incapable of modeling the English language, Chomsky explains a syntactic description commonly referred to as "immediate constituent analysis." The description breaks apart a sentence into its constituent parts. For example, the string "sentence" given in (4) can be first separated into a noun phrase (NP) and a verb phrase (VP). The NP can be broken down into "the man," and the VP can be separated in V and NP, which can be broken down into "took" and "the book," respectively. This sentence description can be formalized to describe mostly all English sentences. Note that "#" just refers to the terminal symbol of the vocabulary.

(4) #^Sentence^#



A phrase-structured grammar is then described as a finite vocabulary or alphabet (V_P) and a finite set of initial strings ($\sum$), and a finite set of rules (F) of the form $X \rightarrow Y$, where both X and Y are contained in V_P. Note that some rules $\in$ F could be mandatory while others may be optional. We refer to this grammar as a [$\sum$, F] grammar.

Now consider a [$\sum$, F] grammar, which accounts for the syntactic description of English sentences. The description is given in (5) and a derivation for "the man took the book" is given in (6).

(5) $\sum$: # ^ Sentence ^ #
F: Sentence → NP ^ VP
VP → Verb ^ NP
NP → the ^ man, the ^ book
Verb → took

(6) D: # ^ Sentence ^ #
^ NP ^ VP ^
^ the ^ man ^ VP ^
^ the ^ man ^ Verb ^ NP ^
^ the ^ man ^ took ^ NP ^
^ the ^ man ^ took ^ the ^ book ^

As can be seen (5), the initial string, "Sentence," is generated from $\sum$, and the rules of F are applied to this initial string. In the derivation in (6), consecutive rules $\in$ F are applied to the initial string, "Sentence" $\in \sum$, to arrive at the final output sentence, "the man took the book."

An important point for phrase-structured grammars is that they are more powerful than finite-state Markov processes. For example, a [$\sum$, F] grammar can be defined in (7) where rule (i) applies when $j \neq 0$ and rule (ii) applies when $j=0$. Again $k \leq N_{i,J}$:

(7) $\sum$: $\{S_0\}$
F: (i) $S_i \rightarrow a_{i,J,k} \wedge S_J$
(ii) $S_i \rightarrow a_{i,0,k}$

It is clear that this [$\sum$, F] grammar describes a language that is equivalent to L_G as described in section 2, meaning that [$\sum$, F] grammars are at least as good as finite-state Markov processes. However, [$\sum$, F] grammars are also capable of describing nesting infinite numbers of dependencies, unlike L_G. For example, a [$\sum$, F] grammar could be defined as

$\sum$: Z
F: Z → "if" ^ W ^ "then" ^ S
W → "either" ^ Z ^ "or" ^ S
Z → "if" ^ S ^ "then" ^ S

where each S is an independent, arbitrary string. Thus, it is clear that nesting an infinite number of dependencies (of the pattern specified in (3)(iii)) can be modeled by this grammar, which makes it more powerful than L_G.

4 Transformational Grammar

However, it can be shown that [$\sum$, F] grammars are still too limited to describe the English language. In the example in (6), "Verb" was replaced by "took," but there are several other possible replacements for "Verb" including: "takes," "has taken," "has been taken," etc… These strings have various interdependencies with the original sentence as well as interdependencies (redundancies) within the string itself (i.e. neither "the man are taking the book" nor "the man is taken the book" is grammatically correct sentence). Describing these interdependencies may be very complicated and most likely not the way the English language structured in the brain when producing a sentence. However, by selecting discontinuous strings of "Verb" as independent symbols, it is possible to conceptually simplify the model of "Verb." Therefore a transformation in this new grammar is actually given below:

(8)(i) Verb → Auxiliary ^ V
(ii) V → run, walk, take, …
(iii) Auxiliary → C(M) {(have ^ en), (be ^ ing) }
(iv) M → will, can, shall, may, must
(v) C → past, present

(9) Af ^ v → v ^ Af ^ #

Now, (8) can be applied to a "Verb" by breaking it apart as shown in (i) and applying (ii) to "V" to arrive at, for example, "take." Next we apply (iii) to "Auxiliary" by decomposing it into "C(M)" and by selecting zero, one, or two of the bracketed strings. Next a value for "C" is chosen and one or zero symbols for M are chosen. For example, "Verb" can be replaced with a string of the form "past ^ have ^ en ^ be ^ ing ^ take." Now we apply the transformation defined in (9), where {"en", "ing"} $\in$ "Af"

and {“V”,”M”, “have”, “be”} ∈”v.” Applying this grammatical transformation to the aforementioned string yields “have ^ past ^ # ^ be ^ en ^ # ^ take ^ ing ^ #.” As described in linguistic theory, this string is a concatenation of morphemes, which are the raw symbols of a sentence. A list of rules is applied to this sentence to give the phonemic spelling of each morpheme, which is what the native speaker will recognize as an English sentence. Applying this last set of morphological rules, the string is converted to “had ^ been ^ taking.”

The description above is indescribable using [∑, F] grammars, because to apply the transformation in (9), it is necessary to know that “took” is a verb. The only way to know that “took” is a verb is to look at the history of the rules being applied and observe that “v” → “took.,” which implies “took” is a verb. Thus the operation given in (9) is not a rule as explained earlier but a grammatical transformation. This notion leads to the concept of transformational grammar. The procedure to generating a sentence is to take the original string, apply a set of rules, apply a set of grammatical transformations, and then apply the morphological rules to yield a string of phonemes, which gives rise to two classes of sentences. The first class is called the kernel sentence, which is produced by applying a [∑, F] grammar to the original string of “# ^ sentence ^ #.” The kernel sentence contains the idea being communicated, since all later grammatical transformations (more or less) preserve the meaning. A series of mandatory and optional transformations are applied to the kernel, and then morphological rules have been applied, which yields the second class of sentences. This class of sentences is the final output sentences to be written down or spoken.

5 Discussion

The transformational grammar developed above is a tool that can be used to insightfully describe the structure of a language. A few details have been omitted for conciseness and clarity such as the inability of “be” to be concatenated with a transitive verb, which means that passive sentence are derived by applying a grammatical transformation. Moreover, this analysis was performed for simple declarative sentences. To arrive at questions, compound sentences, and more, additional grammatical transformation are needed.

A few key concepts can be taken from this paper. The first concept is that the world’s best computer does not necessarily provide insight into the underlying structure of a language. This statement means that the best finite-state Markov model still does not accurately represent the structure of a language in the sense that it cannot generate all possible English-like sentences nor can it simply discriminate grammatically incorrect sentences. For example, any finite-state process does not have infinite memory, which will prevent the computer from generating sentences with sentences nesting large numbers of dependencies. On the other hand, [∑, F] grammars can handle some of these cases. Moreover, the structure of a finite-state Markov chain does not easily encapsulate grammatical errors. For example, it is difficult to understand why the sting “the man a walked dog.” To eliminate the generation using Markov processes to specify the words, it could be arbitrarily decreed that “a” can never precede “walk.” However, this decree cannot explain the deeper principle in English that articles are contained in noun phrases and always precede nouns, which is, in fact, encapsulated in the concept of a phrase structure.

Chomsky argues that the English language is incapable of being modeled as a finite-state Markov model due to the inability of Markov chains at handling cases of infinite numbers of nested dependencies. However, the Lempel-Ziv algorithm assumes the sequence to be compressed is generated by a Markovian source. Since English is argued not to be Markov process, one might conclude that Lempel-Ziv cannot be used to compress the bit/symbol rate of English down to the entropy of a symbol. Nevertheless, for all practical purposes, the number of dependencies nested in an English sentence is always finite. The reason for this is that humans are generating these English sentences to communicate and nesting too many dependencies is confusing and never done. Thus for long strings, a Markov source approximation for English is still valid, and Lempel-Ziv can be used to compress English strings. Therefore, even though the English language is not accurately modeled as a finite-state Markov processes, approximating English as a Markov process is valid for practical compression.

Chomsky’s model of English perhaps offers insight to informational theorists about how the human brain encodes and decodes ideas to communicate to others. For example, an idea is conceived in the brain, and the idea is expanded and inserted into the string “# ^ Sentence ^ #” using a process akin to a [∑, F] grammar to yield a kernel sentence, which contains the idea and can be transformed into a phonological sentences such that it can be communicated by others. A listener receives the sentence and might decode/compress by converting the received sentence into a kernel sentence and then processing out the information.

It would be interesting to consider cases when human communication errors, i.e. when sentences are ambiguous. We then ask whether the model is also illustrates an ambiguous sentence. For example, consider the sentence, “they are flying planes.” Is the idea being communicated that “they” meaning “pilots” are flying some planes, or does it mean that “those specks on the horizon” are flying planes. The sentence is ambiguous, and it is impossible to understand what is meant by the sentence from this sentence alone. The phrase-structured model proposed by Chomsky also outputs an ambiguous sentence, and it is easy to see why. For the string “NP^VP^NP,” depending on the idea could be desribed by either of the two following situations: 1) NP→they, VP→”are^flying”, and NP→planes, or 2) NP→they, VP→are, and NP→”flying^planes.” With Chomsky’s model either of the two ideas could have been encoded to be

"they^are^flying^planes," which means that there is not a unique idea for the syntax of this sentence.

Finally, using a [∑, F] grammar, compression can be easier in certain cases. For example if the vocabulary includes the symbols "a^b", "a^a^b^b", "a^a^a^b^b^b", and so on. A [∑, F] grammar can be defined as ∑ = Z, Z→"a^Z^b", and Z→"a^b." Thus to decode which string is sent, the only information needed is the number of "a's" in the message. If this number, n, is received, the message can be reproduced by applying the rule "Z→'a^Z^b'" n-1 times and then applying "Z→'a^b'" once. It can be shown the number of bits needed to transmit an integer, n, is less than $\log_2 n+\log_2\log_2 n+C$ where C is an arbitrary constant. Thus if the length of the string in the alphabet is L, it can be compressed down to $\log_2 n+\log_2\log_2 n+C$ where n=L/2.

5 References

[1] Chomsky, N., "Three Models for the Description of Language," IEEE Trans. on Info. Theory, Vol. 2, No. 3, Nov. 1956.