CHAPTER 11

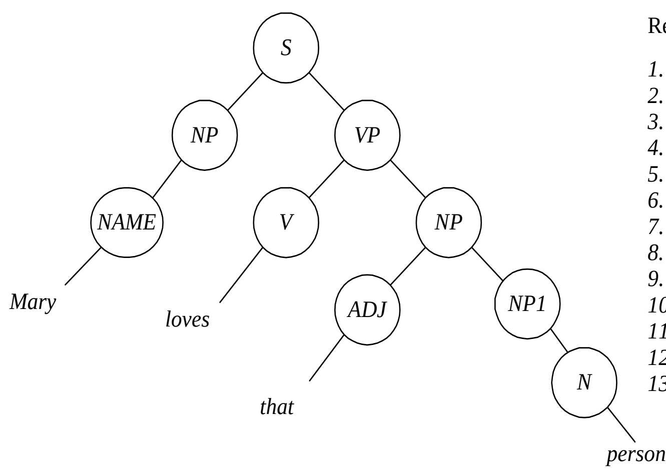
Language Modeling

Acoustic pattern matching, as discussed in Chapter 9, and knowledge about language are equally important in recognizing and understanding natural speech. Lexical knowledge (i.e., vocabulary definition and word pronunciation) is required, as are the syntax and semantics of the language (the rules that determine what sequences of words are grammatically well-formed and meaningful). In addition, knowledge of the pragmatics of language (the structure of extended discourse, and what people are likely to say in particular contexts) can be important to achieving the goal of spoken language understanding systems. In practical speech recognition, it may be impossible to separate the use of these different levels of knowledge, since they are often tightly integrated.

In this chapter we review the basic concept of Chomsky's formal language theory and the probabilistic language model. For the formal language model, two things are fundamental: the grammar and the parsing algorithm. The *grammar* is a formal specification of the permissible structures for the language. The *parsing* technique is the method of analyzing the sentence to see if its structure is compliant with the grammar. With the advent of bodies of text (*corpora*) that have had their structures hand-annotated, it is now possible to generalize the formal grammar to include accurate probabilities. Furthermore, the probabilistic relationship among a sequence of words can be directly derived and modeled from the corpora with the so-called stochastic language models, such as *n*-gram, avoiding the need to create broad coverage formal grammars. Stochastic language models play a critical role in building a working spoken language system, and we discuss a number of important issues associated with them.

11.1. FORMAL LANGUAGE THEORY

In constructing a syntactic grammar for a language, it is important to consider the generality, the selectivity, and the understandability of the grammar. The *generality* and *selectivity* basically determine the range of sentences the grammar accepts and rejects. The *understandability* is important, since it is up to the authors of the system to create and maintain the grammar. For SLU systems described in Chapter 17, we need to have a grammar that covers and generalizes to most of the typical sentences for an application. The system also needs to distinguish the kind of sentences for different actions in a given application. Without understandability, it is almost impossible to improve a practical SLU system that typically involves a large number of developers to maintain and refine the grammar.



Rewrite Rules:

1. $S \rightarrow NP\ VP$
2. $VP \rightarrow V\ NP$
3. $VP \rightarrow AUX\ VP$
4. $NP \rightarrow ART\ NP1$
5. $NP \rightarrow ADJ\ NP1$
6. $NP1 \rightarrow ADJ\ NP1$
7. $NP1 \rightarrow N$
8. $NP \rightarrow NAME$
9. $NP \rightarrow PRON$
10. $NAME \rightarrow Mary$
11. $V \rightarrow loves$
12. $ADJ \rightarrow that$
13. $N \rightarrow person$

Figure 11.1 A tree representation of a sentence and its corresponding grammar

The most common way of representing the grammatical structure of a sentence, "*Mary loves that person*", is by using a tree, as illustrated in Figure 11.1. The node labeled *S* is the parent node of the nodes labeled *NP* and *VP* for noun phrase and verb phrase respectively. The *VP* node is the parent node of node *V* and *N* - for verb and noun, respectively. Each leaf is associated with the word in the sentence to be analyzed. To construct such a tree for a sentence, we must know the structure of the language so that a set of rewrite rules can be used to describe what tree structures are allowable. These rules, as illustrated in Figure 11.1, determine that a certain symbol may be expanded in the tree by a sequence of symbols. The grammatical structure helps in determining the meaning of the sentence. It tells us that *that* in the sentence modifies *person*. Mary loves *that* person.

11.1.1. Chomsky Hierarchy

In Chomsky's *formal language theory* [1, 14, 15], a grammar is defined as $G = (V, T, P, S)$, where V and T are finite sets of *non-terminals* and *terminals*, respectively. V contains all the *non-terminal* symbols. We often use upper-case symbols to denote them. In the example discussed here, *S, NP, NP1, VP, NAME, ADJ, N*, and *V* are non-terminal symbols. The *terminal* set T contains *Mary, loves, that,* and *person*, which are often denoted with lower-case symbols. P is a finite set of *production (rewrite) rules*, as illustrated in the rewrite rules in Figure 11.1. S is a special non-terminal, called the *start symbol*.

Table 11.1 Chomsky hierarchy and the corresponding machine that accepts the language.

Types	Constraints	Automata
Phase structure grammar	$\alpha \rightarrow \beta$. This is the most general grammar.	Turing machine
Context-sensitive grammar	A subset of the phrase structure grammar. $\lvert\alpha\rvert \le \lvert\beta\rvert$, where |.| indicates the length of the string.	Linear bounded automata
Context-free grammar (CFG)	A subset of the context sensitive grammar. The production rule is $A \rightarrow \beta$, where A is a non-terminal. This production rule is shown to be equivalent to Chomsky normal form: $A \rightarrow w$ and $A \rightarrow BC$, where w is a terminal and B, C are non-terminals.	Push down automata
Regular grammar	A subset of the CFG. The production rule is expressed as: $A \rightarrow w$ and $A \rightarrow wB$.	Finite-state automata

The language to be analyzed is essentially a string of terminal symbols, such as "*Mary loves that person*." It is produced by applying production rules sequentially to the start symbol. The production rule is of the form $\alpha \rightarrow \beta$, where α and β are arbitrary strings of grammar symbols V and T, and the α must not be empty. In formal language theory, four major languages and their associated grammars are hierarchically structured. They are referred to as the Chomsky hierarchy [1] as defined in Table 11.1. There are four kinds of automata that can accept the languages produced by these four types of grammars. Among these automata, the finite-state automaton is not only the mathematical device used to implement the regular grammar but also one of the most significant tools in computational linguistics. Variations of automata such as finite-state transducers, hidden Markov models, and *n*-gram models are important examples in spoken language processing.

These grammatical formulations can be compared according to their generative capacity, i.e., the range that the formalism can cover. While there is evidence that natural languages are at least weakly context sensitive, the context-sensitive requirements are rare in practice. The context-free grammar (CFG) is a very important structure for dealing with both machine language and natural language. CFGs are not only powerful enough to describe most of the structure in spoken language,[1] but also restrictive enough to have efficient parsers to analyze natural sentences. Since CFGs offer a good compromise between parsing efficiency and power in representing the structure of the language, they have been widely applied to natural language processing. Alternatively, regular grammars, as represented with a finite-state machine, can be applied to more restricted applications. Since finite-state grammars are a subset of the more general context-free grammar, we focus our discussion on context free grammars only, although the parsing algorithm for finite-state grammars can be more efficient.

As discussed in Section 11.1.2, a parsing algorithm offers a procedure that searches through various ways of combining grammatical rules to find a combination that generates a tree to illustrate the structure of the input sentence, which is similar to the search problem in speech recognition. The result of the parsing algorithm is a parse tree,[2] which can be regarded as a record of the CFG rules that account for the structure of the sentence. In other words, if we parse the sentence, working either top-down from *S* or bottom-up from each word, we automatically derive something that is similar to the tree representation, as illustrated in Figure 11.1.

A push-down automaton is also called a *recursive transition network* (RTN), which is an alternative formalism to describe context-free grammars. A transition network consists of nodes and labeled arcs. One of the nodes is specified as the initial state *S*. Starting at the initial state, we traverse an arc if the current word in the sentence is in the category on the arc. If the arc is followed, the current word is updated to the next word. A phrase can be parsed if there is a path from the starting node to a *pop* arc that indicates a complete parse for all the words in the phrase. Simple transition networks without recursion are often called *finite-state machines* (FSM). Finite-state machines are equivalent in expressive power to regular grammars and, thus, are not powerful enough to describe all languages that can be described by CFGs. Chapter 12 has a more detailed discussion on RTNs and FSMs into speech recognition.

[1] The effort to prove natural languages is noncontext free is summarized in Pullman and Gazdar [54].

[2] The result can be more than one parse tree since natural language sentences are often ambiguous. In practice, a parsing algorithm should not only consider all the possible parse trees but also provide a ranking among them, as discussed in Chapter 17.

11.1.2. Chart Parsing for Context-Free Grammars

Since Chomsky introduced the notion of context-free grammars in the 1950s, a vast literature has arisen on the parsing algorithms. Most parsing algorithms were developed in computer science to analyze programming languages that are not ambiguous in the way that spoken language is [1, 32]. We discuss only the most relevant materials that are fundamental to building spoken language systems, namely the chart parser for the context-free grammar. This algorithm has been widely used in state-of-the-art spoken language understanding systems.

11.1.2.1. Top Down or Bottom Up?

Parsing is a special case of the search problem generally encountered in speech recognition. A parsing algorithm offers a procedure that searches through various ways of combining grammatical rules to find a combination that generates a tree to illustrate the structure of the input sentence, as illustrated in Figure 11.1. The search procedure can start from the root of the tree with the *S* symbol, attempting to rewrite it into a sequence of terminal symbols that matches the words in the input sentence, which is based on *goal-directed search*. Alternatively, the search procedure can start from the words in the input sentence and identify a word sequence that matches some non-terminal symbol. The bottom-up procedure can be repeated with partially parsed symbols until the root of the tree or the start symbol *S* is identified. This *data-directed search* has been widely used in practical SLU systems.

A top-down approach starts with the *S* symbol, then searches through different ways to rewrite the symbols until the input sentence is generated, or until all possibilities have been examined. A grammar is said to accept a sentence if there is a sequence of rules that allow us to rewrite the start symbol into the sentence. For the grammar in Figure 11.1, a sequence of rewrite rules can be illustrated as follows:

```
S
→ NP VP (rewriting S using S→NP)
→NAME VP (rewriting NP using NP→NAME)
→Mary VP (rewriting NAME using NAME→Mary)
…
→Mary loves that person (rewriting N using N→person)
```

Alternatively, we can take a bottom-up approach to start with the words in the input sentence and use the rewrite rules backward to reduce the sequence of symbols until it becomes S. The left-had side or each rule is used to rewrite the symbol on the right-hand side as follows:

```
→NAME loves that person (rewriting Mary using NAME→Mary)
→NAME V that person (rewriting loves using B→loves)
…
→NP VP (rewriting NP using S→NP VP)
→S
```

A parsing algorithm must systematically explore every possible state that represents the intermediate node in the parsing tree. As discussed in Chapter 12, if a mistake occurs early on in choosing the rule that rewrites *S*, the intermediate parser results can be quite wasteful if the number of rules becomes large.

The main difference between top-down and bottom-up parsers is the way the grammar rules are used. For example, consider the rule $NP \rightarrow ADJ\ NP1$. In a top-down approach, the rule is used to identify an *NP* by looking for the sequence *ADJ NP1*. Top-down parsing can be very predictive. A phrase or a word may be ambiguous in isolation. The top-down approach may prevent some ungrammatical combinations from consideration. It never wastes time exploring trees that cannot result in an *S*. On the other hand, it may predict many different constituents that do not have a match to the input sentence and rebuild large constituents again and again. For example, when the grammar is *left-recursive* (i.e., it contains a non-terminal category that has a derivation that includes itself anywhere along its leftmost branch), the top-down approach can lead a top-down, depth-first left-to-right parser to recursively expand the same non-terminal over again in exactly the same way. This causes an infinite expansion of trees. In contrast, a bottom-up parser takes a sequence *ADJ NP1* and identifies it as an *NP* according to the rule. The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules. It checks the input only once, and only builds each constituent exactly once. However, it may build up trees that have no hope of leading to *S* since it never suggests trees that are not at least locally grounded in the actual input. Since bottom-up parsing is similar to top-down parsing in terms of overall performance and is particularly suitable for robust spoken language processing as described in Chapter 17, we use the bottom-up method as our example to understand the key concept in the next section.

11.1.2.2. Bottom-Up Chart Parsing

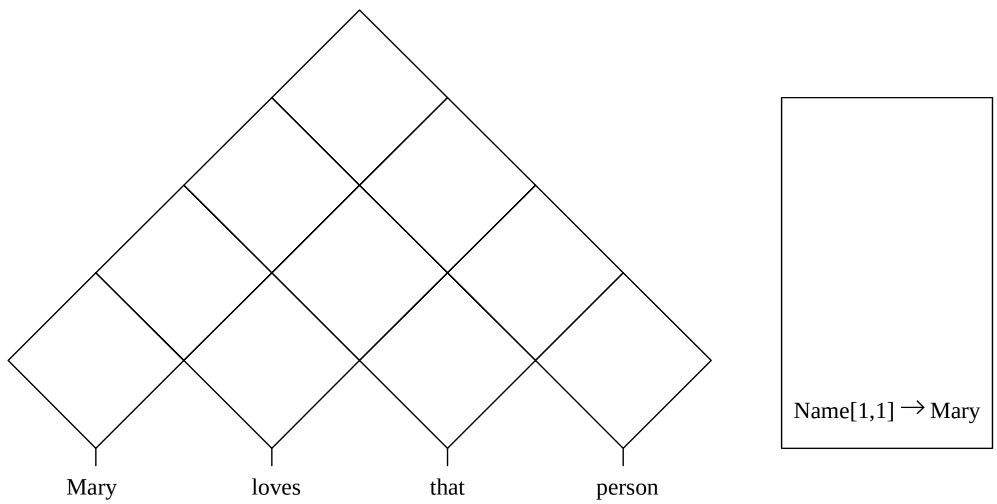
As a standard search procedure, the state of the search consists of a symbol list, starting with the words in the sentence. Successor states can be generated by exploring all possible ways to replace a sequence of symbols that matches the right-hand side of a grammar rule with its left-hand side symbol. A simple-minded solution enumerates all the possible matches, leading to prohibitively expensive computational complexity. To avoid this problem, it is necessary to store partially parsed results of the matching, thereby eliminating duplicate work. This is the same technique that has been widely used in dynamic programming, as described in Chapter 8. Since chart parsing does not need to be from left to right, it is more efficient than the graph search algorithm discussed in Chapter 12, which can be used to parse the input sentence from left to right.

A data structure, called a *chart*, is used to allow the parser to store the partial results of the matching. The chart data structure maintains not only the records of all the constituents derived from the sentence so far in the parse tree, but also the records of rules that have matched partially but are still incomplete. These are called *active arcs*. Here, matches are always considered from the point of view of some *active constituents*, which represent the subparts that the input sentence can be divided into according to the rewrite rules. Active constituents are stored in a data structure called an *agenda*. To find grammar rules that match a string involving the active constituent, we need to identify rules that start with the active constituent or rules that have already been started by earlier active constituents and require the current constituent to complete the rule or to extend the rule. The basic operation of a chart-based parser involves combining these partially matched records (active arcs) with a completed constituent to form either a new completed constituent or a new partially matched (but incomplete) constituent that is an extension of the original partially matched constituent. Just like the graph search algorithm, we can use either a depth-first or breadth-first search strategy, depending on how the agenda is implemented. If we use probabilities or other heuristics, we take a best-first strategy discussed in Chapter 12 to select constituents from the agenda. The chart-parser process is defined more precisely in Algorithm 11.1. It is possible to combine both top-down and bottom-up. The major difference is how the constituents are used.

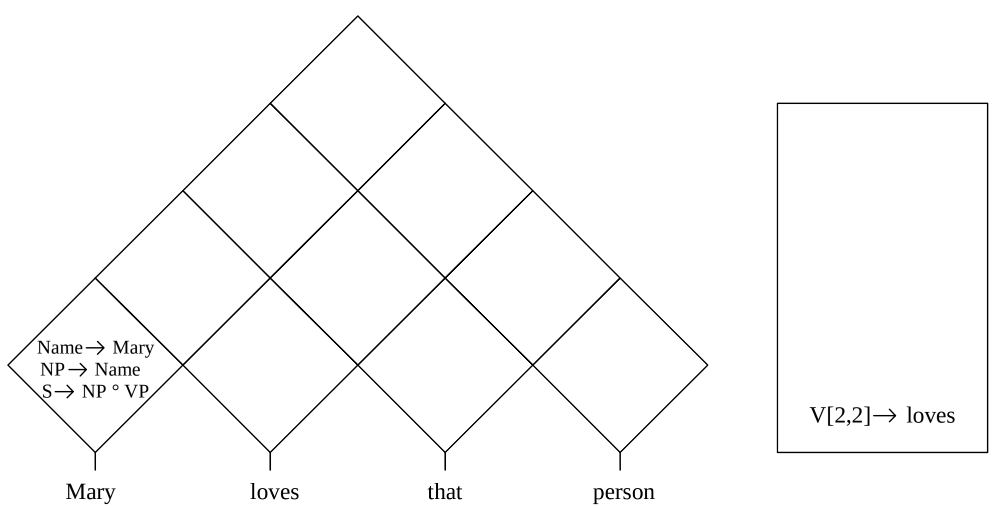
ALGORITHM 11.1 A BOTTOM-UP CHART PARSER

Step1: **Initialization**: Define a list called chart to store active arcs, and a list called an agenda to store active constituents until they are added to the chart.
Step 2: **Repeat**: Repeat Step 2 to 7 until there is no input left.
Step 3: **Push and pop the agenda**: If the agenda is empty, look up the interpretations of the next word in the input and push them to the agenda. Pop a constituent *C* from the agenda. If *C* corresponds to position from w_i to w_j of the input sentence, we denote it *C[i,j].*
Step 4: **Add *C* to the chart**: Insert *C[i,j]* into the chart.
Step 5: **Add key-marked active arcs to the chart**: For each rule in the grammar of the form $X \to C\ Y$, add to the chart an active arc (partially matched constituent) of the form $X[i,j] \to °CY$, where ° denotes the critical position called the key that indicates that everything before ° has been seen, but things after ° are yet to be matched (incomplete constituent).
Step 6: **Move ° forward**: For any active arc of the form $X[1,j] \to Y\ldots°C\ldots Z$ (everything before w_i) in the chart, add a new active arc of the form $X[1,j] \to Y\ldots C°\ldots Z$ to the chart.
Step 7: **Add new constituents to the agenda**: For any active arc of the form $X[1,I] \to Y\ldots\#C$, add a new constituent of type *X[1,j]* to the agenda.
Step 8: **Exit:** If *S*[*1,n*] is in the chart, where n is the length of the input sentence, we can exit successfully unless we want to find all possible interpretations of the sentence. The chart may contain many *S* structures covering the entire set of positions.

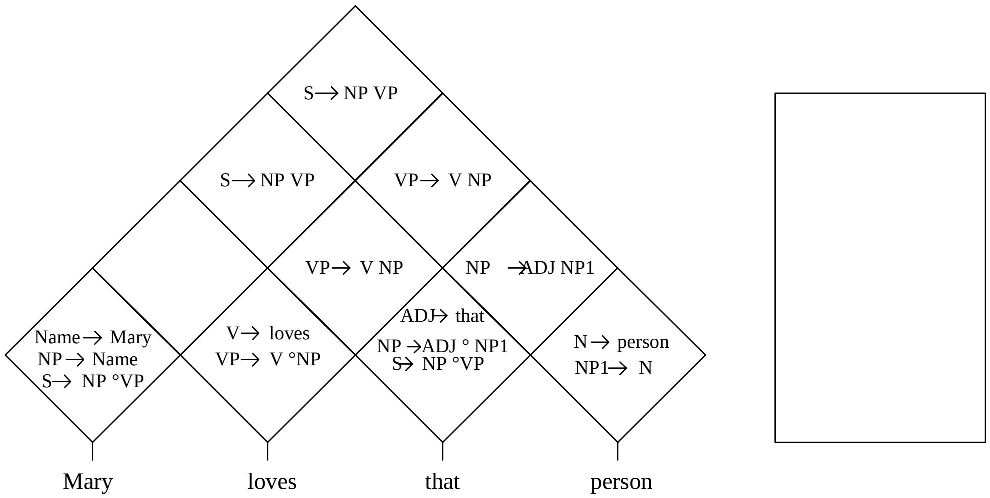
Let us look at an example to see how the chart parser parses the sentence *Mary loves that person* using the grammar specified in Figure 11.1. We first create the chart and agenda data structure as illustrated in Figure 11.2 (a), in which the leaves of the tree-like chart data structure corresponds to the position of each input word. The parent of each block in the chart covers from the position of the left child's corresponding starting word position to the right child's corresponding ending word position. Thus, the root block in the chart covers the whole sentence from the first word *Mary* to the last word *person*. The chart parser scans through the input words to match against possible rewrite rules in the grammar. For the first word, the rule *Name→Mary* can be matched, so it is added to the agenda according to Step 3 in Algorithm 11.1. In Step 4, *Name→Mary* is added to the chart from the agenda. After the word Mary is processed, we have *Name→Mary*, *NP→Name*, and *S→NP°VP* in the chart, as illustrated in Figure 11.2 (b). *NP°VP* in the chart indicates that ° has reached the point at which everything before ° has been matched (in this case *Mary* matched *NP*) but everything after ° is yet to be parsed. The completed parsed chart is illustrated in Figure 11.2 (c).



(a) The chart is illustrated on the left, and the agenda is on the right. The agenda now has one rule in it according to Step 3, since the agenda is empty.



(b) After Mary, the chart now has rules *Name→Mary*, *NP→Name*, and *S→NP°VP*.



(c) The chart after the whole sentence is parsed. S→ NP VP covers the whole sentence, indicating that the sentence is parsed successfully by the grammar.

Figure 11.2 An example of a chart parser with the grammar illustrated in Figure 11.1. Parts (a) and (b) show the initial chart and agenda to parse the first word.; part (c) shows the chart after the sentence is completely parsed.

A parser may assign one or more parsed structures to the sentence in the language it defines. If any sentence is assigned more than one such structure, the grammar is said to be ambiguous. Spoken language is, of course, ambiguous by nature.[3] For example, we can have a sentence like *Mary sold the student bags*. It is unclear whether *student* should be the modifier for *bags* or whether it means that *Mary* sold the *bags* to *the student*.

Chart parsers can be fairly efficient simply because the same constituent is never constructed more than once. In the worst case, the chart parser builds every possible constituent between every possible pair of positions, leading to the worst-case computational complexity of $O(n^3)$, where n is the length of the input sentence. This is still far more efficient than a straightforward brute-force search.

In many practical tasks, we need only a partial parse or shallow parse of the input sentence. You can use cascades of finite-state automata instead of CFGs. Relying on simple finite-state automata rather than full parsing makes such systems more efficient, although finite-state systems cannot model certain kinds of recursive rules, so that efficiency is traded for a certain lack of coverage.

[3] The same parse tree can also mean multiple things, so a parse tree itself does not define meaning. "*Mary loves that person*" could be sarcastic and mean something different.

11.2. Stochastic Language Models

Stochastic language models (SLM) take a probabilistic viewpoint of language modeling. We need to accurately estimate the probability $P(\mathbf{W})$ for a given word sequence $\mathbf{W} = w_1 w_2 ... w_n$. In the formal language theory discussed in Section 11.1, $P(\mathbf{W})$ can be regarded as 1 or 0 if the word sequence is accepted or rejected, respectively, by the grammar. This may be inappropriate for spoken language systems, since the grammar itself is unlikely to have a complete coverage, not to mention that spoken language is often ungrammatical in real conversational applications.

The key goal of SLM is to provide adequate probabilistic information so that the likely word sequences should have a higher probability. This not only makes speech recognition more accurate but also helps to dramatically constrain the search space for speech recognition (see Chapters 12 and 13). Notice that SLM can have a wide coverage on all the possible word sequences, since probabilities are used to differentiate different word sequences. The most widely used SLM is the so call *n*-gram model discussed in this chapter. In fact, the CFG can be augmented as the bridge between the *n*-gram and the formal grammar if we can incorporate probabilities into the production rules, as discussed in the next section.

11.2.1. Probabilistic Context-Free Grammars

The CFG can be augmented with probability for each production rule. The advantages of the probabilistic CFGs (PCFGs) lie in their ability to more accurately capture the embedded usage structure of the spoken language to minimize the syntactic ambiguity. The use of probability becomes increasingly important to discriminate many competing choices when the number of rules is large.

In the PCFG, we have to address the parallel problems we discussed for HMMs in Chapter 8. The *recognition problem* is concerned with the computation of the probability of the start symbol S generating the word sequence $\mathbf{W} = w_{1,} w_{2,} \ldots w_T$, given the grammar G:

$$P(S \Rightarrow \mathbf{W}|G) \tag{11.1}$$

where $\Rightarrow$ denotes a derivation sequence consisting of one or more steps. This is equivalent to the chart parser augmented with probabilities, as discussed in Section 11.1.2.2.

The *training problem* is concerned with determining a set of rules in G based on the training corpus and estimating the probability of each rule. If the set of rules is fixed, the simplest approach to deriving these probabilities is to count the number of times each rule is used in a corpus containing parsed sentences. We denote the probability of a rule $A \to \alpha$ by $P(A \to \alpha|G)$. For instance, if there are m rules for left-hand side non-terminal node A: $A \to \alpha_1, A \to \alpha_{2,} \ldots A \to \alpha_m$, we can estimate the probability of these rules as follows:

$$P(A \to \alpha_j \mid G) = C(A \to \alpha_j) / \sum_{i=1}^{m} C(A \to \alpha_i) \tag{11.2}$$

where *C(.)* denotes the number of times each rule is used.

When you have hand-annotated corpora, you can use the maximum likelihood estimation as illustrated by Eq. (11.2) to derive the probabilities. When you don't have hand-annotated corpora, you can extend the EM algorithm (see Chapter 4) to derive these probabilities. The algorithm is also known as the *inside-outside* algorithm. As what we discussed in Chapter 8, you can develop algorithms similar to the Viterbi algorithm to find the most likely parse tree that could have generated the sequence of words $P(\mathbf{W})$ after these probabilities are estimated.

We can make certain independence assumptions about rule usage. Namely, we assume that the probability of a constituent being derived by a rule is independent of how the constituent is used as a subconstituent. For instance, we assume that the probabilities of *NP* rules are the same no matter whether the *NP* is used for the subject or the object of a verb, although the assumptions are not valid in many cases. More specifically, let the word sequence $\mathbf{W}=w_{1,}w_{2.}\ldots w_T$ be generated by a PCFG *G*, with rules in Chomsky normal form as discussed in Section 11.1.1:

$$A_i \rightarrow A_m A_n \text{ and } A_i \rightarrow w_l \tag{11.3}$$

where A_m and A_n are two possible non-terminal that expand A_i at different locations. The probability for these rules must satisfy the following constraint:

$$\sum_{m,n} P(A_i \rightarrow A_m A_n \mid G) + \sum_l P(A_i \rightarrow w_l \mid G) = 1, \quad \text{for all } i \tag{11.4}$$

Equation (11.4) simply means that all non-terminals can generate either pairs of non-terminal symbols or a single terminal symbol, and all these production rules should satisfy the probability constraint. Analogous to the HMM forward and backward probabilities discussed in Chapter 8, we can define the inside and outside probabilities to facilitate the estimation of these probabilities from the training data.

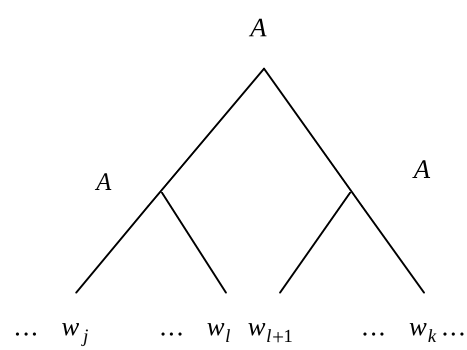


Figure 11.3 Inside probability is computed recursively as sum of all the derivations.

A non-terminal symbol A_i can generate a sequence of words $w_j w_{j+1} \ldots w_k$; we define the probability of $Inside(j, A_i, k) = P(A_i \Rightarrow w_j w_{j+1} ... w_k \mid G)$ as the *inside constituent probability*, since it assigns a probability to the word sequence inside the constituent. The inside probability can be computed recursively. When only one word is emitted, the transition rule of the form $A_i \rightarrow w_m$ applies. When there is more than one word, rules of the form $A_i \rightarrow A_j A_k$ must apply. The inside probability of $inside(j, A_i, k)$ can be expressed recursively as follows:

$$
\begin{aligned}
& inside(j, A_{i,} k) = P(A_i \Rightarrow w_j w_{j+1} \ldots w_k) \\
& = \sum_{n,m} \sum_{l=j}^{k-1} P(A_i \rightarrow A_m A_n) P(A_m \Rightarrow w_j \ldots w_l) P(A_n \Rightarrow w_{l+1} \ldots w_k) \\
& == \sum_{n,m} \sum_{l=j}^{k-1} P(A_i \rightarrow A_m A_n) inside(j, A_m, l) inside(l+1, A_n, k)
\end{aligned}
\tag{11.5}
$$

The inside probability is the sum of the probabilities of all derivations for the section over the span of *j* to *k*. One possible derivation of the form can be drawn as a parse tree shown in Figure 11.3.

Another useful probability is the *outside* probability for a non-terminal node A_i covering w_s to w_t, in which they can be derived from the start symbol *S*, as illustrated in Figure 11.4, together with the rest of the words in the sentence:

$$
outside(s, A_i, t) = P(S \Rightarrow w_1 \ldots w_{s-1} A_i \; w_{t+1} \ldots w_T) \tag{11.6}
$$

After the inside probabilities are computed bottom-up, we can compute the outside probabilities top-down. For each non-terminal symbol A_i, there are one of two possible configurations $A_m \rightarrow A_n A_i$ or $A_m \rightarrow A_i A_n$ as illustrated in Figure 11.5. Thus, we need to consider all the possible derivations of these two forms as follows:

$$
\begin{aligned}
& outside(s, A_i, t) = P(S \Rightarrow w_1 ... w_{s-1} \; A_i \; w_{t+1} ... w_T) \\
& = \sum_{m,n} \left\{ \begin{array}{l} \displaystyle\sum_{l=1}^{s-1} P(A_m \rightarrow A_n A_i) P(A_n \Rightarrow w_l ... w_{s-1}) P(S \Rightarrow w_1 ... w_{l-1} \; A_m \; w_{t+1} ... w_T) + \\ \displaystyle + \sum_{l=t+1}^{T} P(A_m \rightarrow A_i A_n) P(A_n \Rightarrow w_{t+1} ... w_l) P(S \Rightarrow w_1 ... w_{s-1} \; A_m \; w_{l+1} ... w_T) \end{array} \right\} \\
& = \sum_{m,n} \left\{ \begin{array}{l} \displaystyle\sum_{l=1}^{s-1} P(A_m \rightarrow A_n A_i) inside(l, A_n, s-1) outside(l, A_m, t) + \\ \displaystyle + \sum_{l=t+1}^{T} P(A_m \rightarrow A_i A_n) inside(t+1, A_n, l) outside(s, \; A_m, l) \end{array} \right\}
\end{aligned}
\tag{11.7}
$$

The inside and outside probabilities are used to compute the sentence probability as follows:

$$P(S \Rightarrow w_1 \ldots w_T) = \sum_i inside(s, A_i, t)outside(s, A_i, t) \qquad \text{for any } s \le t \tag{11.8}$$

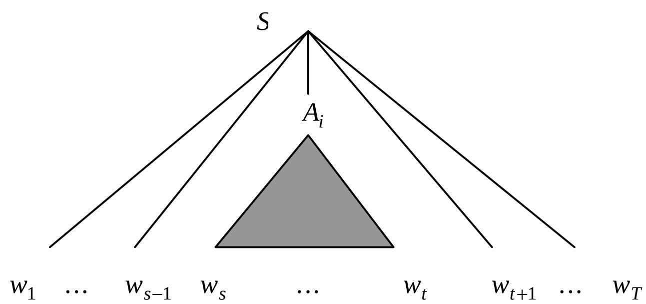


Figure 11.4 Definition of the outside probability.

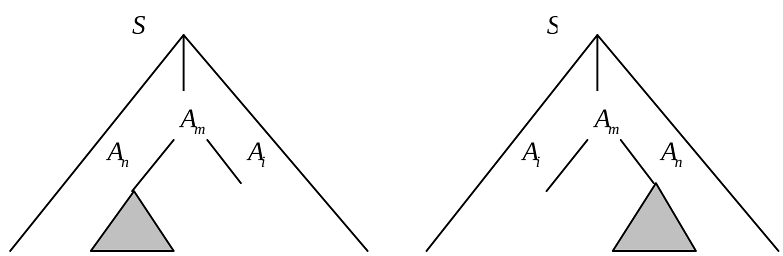


Figure 11.5 Two possible configurations for a non-terminal node A_m.

Since $outside(1, A_i, T)$ is equal to 1 for the starting symbol only, the probability for the whole sentence can be conveniently computed using the inside probability alone as

$$P(S \Rightarrow \mathbf{W}|G) = inside(1, S, T) \tag{11.9}$$

We are interested in the probability that a particular rule, $A_i \to A_m A_n$ is used to cover a span $w_s \ldots w_t$, given the sentence and the grammar:

$$\xi(i, m, n, s, t) = P(A_i \Rightarrow w_s ... w_t, A_i \to A_m A_n \mid S \Rightarrow \mathbf{W}, G)$$
$$= \frac{1}{P(S \Rightarrow \mathbf{W} \mid G)} \sum_{k=s}^{t-1} P(A_i \to A_m A_n \mid G) inside(s, A_m, k) inside(k+1, A_n, t) outside(s, A_i, t) \tag{11.10}$$

These conditional probabilities form the basis of the inside-outside algorithm, which is similar to the forward-backward algorithm discussed in Chapter 8. We can start with some initial probability estimates. For each sentence of training data, we determine the inside and outside probabilities in order to compute, for each production rule, how likely it is that the production rule is used as part of the derivation of that sentence. This gives us the number of counts for each production rule in each sentence. Summing these counts across sentences gives us an estimate of the total number of times each production rule is used to produce the sentences in the training corpus. Dividing by the total counts of productions used for each non-terminal gives us a new estimate of the probability of the production in the MLE framework. For example, we have:

$$P(A_i \to A_m A_n | G) = \frac{\sum_{s=1}^{T-1} \sum_{t=s+1}^{T} \xi(i,m,n,s,t)}{\sum_{m,n} \sum_{s=1}^{T-1} \sum_{t=s+1}^{T} \xi(i,m,n,s,t)} \tag{11.11}$$

In a similar manner, we can estimate $P(A_i \to w_m | G)$. It is also possible to let the inside-outside algorithm formulate all the possible grammar production rules so that we can select rules with sufficient probability values. If there is no constraint, we may have too many *greedy symbols* that serve as possible non-terminals. In addition, the algorithm is guaranteed only to find a local maximum. It is often necessary to use prior knowledge about the task and the grammar to impose strong constraints to avoid these two problems. The chart parser discussed in Section 11.1.2 can be modified to accommodate PCFGs [29, 45].

One problem with the PCFG is that it assumes that the expansion of any one non-terminal is independent of the expansion of other non-terminals. Thus each PCFG rule probability is multiplied together without considering the location of the node in the parse tree. This is against our intuition since there is a strong tendency toward the context-dependent expansion. Another problem is its lack of sensitivity to words, although lexical information plays an important role in selecting the correct parsing of an ambiguous prepositional phrase attachment. In the PCFG, lexical information can only be represented via the probability of pre-terminal nodes, such as verb or noun, to be expanded lexically. You can add lexical dependencies to PCFGs and make PCFG probabilities more sensitive to surrounding syntactic structure [6, 11, 19, 31, 45].

11.2.2. N-gram Language Models

As covered earlier, a language model can be formulated as a probability distribution $P(\mathbf{W})$ over word strings $\mathbf{W}$ that reflects how frequently a string $\mathbf{W}$ occurs as a sentence. For example, for a language model describing spoken language, we might have *P(hi)* = 0.01, since perhaps one out of every hundred sentences a person speaks is *hi*. On the other hand, we would have *P(lid gallops Changsha pop*) = 0, since it is extremely unlikely anyone would utter such a strange string.

$P(\mathbf{W})$ can be decomposed as

$$
\begin{aligned}
P(\mathbf{W}) &= P(w_1, w_2, \ldots, w_n) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\cdots P(w_n|w_1, w_2, \ldots, w_{n-1}) \\
&= \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1})
\end{aligned}
\tag{11.12}
$$

where $P(w_i|w_1, w_2, \ldots, w_{i-1})$ is the probability that w_i will follow, given that the word sequence $w_1, w_2, \ldots, w_{i-1}$ was presented previously. In Eq. (11.12), the choice of w_i thus depends on the entire past history of the input. For a vocabulary of size v there are v^{i-1} different histories and so, to specify $P(w_i|w_1, w_2, \ldots, w_{i-1})$ completely, v^i values would have to be estimated. In reality, the probabilities $P(w_i|w_1, w_2, \ldots, w_{i-1})$ are impossible to estimate for even moderate values of i, since most histories $w_1, w_2, \ldots, w_{i-1}$ are unique or have occurred only a few times. A practical solution to the above problems is to assume that $P(w_i|w_1, w_2, \ldots, w_{i-1})$ depends only on some equivalence classes. The equivalence class can be simply based on the several previous words $w_{i-N+1}, w_{i-N+2}, \ldots, w_{i-1}$. This leads to an *n-gram* language model. If the word depends on the previous two words, we have a *trigram*: $P(w_i|w_{i-2}, w_{i-1})$. Similarly, we can have *unigram:* $P(w_i)$, or *bigram:* $P(w_i|w_{i-1})$ language models. The trigram is particularly powerful, as most words have a strong dependence on the previous two words, and it can be estimated reasonably well with an attainable corpus.

In bigram models, we make the approximation that the probability of a word depends only on the identity of the immediately preceding word. To make $P(w_i|w_{i-1})$ meaningful for i = 1, we pad the *beginning of the sentence* with a distinguished token <s>; that is, we pretend w_0 = <s>. In addition, to make the sum of the probabilities of all strings equal 1, it is necessary to place a distinguished token </s> at the *end of the sentence*. For example, to calculate *P(Mary loves that person)* we would take

P(*Mary loves that person*) =
P(*Mary*|<s>)*P*(*loves*|*Mary*)*P*(*that*|*loves*)*P*(*person*|*that*)*P*(</s>|*person*)

To estimate $P(w_i|w_{i-1})$, the frequency with which the word w_i occurs given that the last word is w_{i-1}, we simply count how often the sequence (w_{i-1}, w_i) occurs in some text and normalize the count by the number of times w_{i-1} occurs.

In general, for a trigram model, the probability of a word depends on the two preceding words. The trigram can be estimated by observing the frequencies or counts of the word pair $C(w_{i-2}, w_{i-1})$ and triplet $C(w_{i-2}, w_{i-1}, w_i)$ as follows:

$$
P(w_i|w_{i-2,} w_{i-1}) = \frac{C(w_{i-2,} w_{i-1,} w_i)}{C(w_{i-2,} w_{i-1})} \tag{11.13}
$$

The text available for building a model is called a training corpus. For *n*-gram models, the amount of training data used is typically many millions of words. The estimate of Eq. (11.13) is based on the maximum likelihood principle, because this assignment of probabilities yields the trigram model that assigns the highest probability to the training data of all possible trigram models.

We sometimes refer to the value *n* of an *n*-gram model as its order. This terminology comes from the area of Markov models, of which *n*-gram models are an instance. In particular, an *n*-gram model can be interpreted as a Markov model of order *n*-1.

Consider a small example. Let our training data *S* be comprised of the three sentences *John read her book. I read a different book. John read a book by Mulan* and let us calculate *P(John read a book)* for the maximum likelihood bigram model. We have

$$P(John \mid <s>) = \frac{C(<s>, John)}{C(<s>)} = \frac{2}{3}$$

$$P(read \mid John) = \frac{C(John, read)}{C(John)} = \frac{2}{2}$$

$$P(a \mid read) = \frac{C(read, a)}{C(read)} = \frac{2}{3}$$

$$P(book \mid a) = \frac{C(a, book)}{C(a)} = \frac{1}{2}$$

$$P(</s> \mid book) = \frac{C(book, </s>)}{C(book)} = \frac{2}{3}$$

These trigram probabilities help us estimate the probability for the sentence as:

$$\begin{aligned} &P(John\ read\ a\ book) \\ &= P(John \mid <s>)P(read \mid John)P(a \mid read)P(book \mid a)P(</s> \mid book) \\ &\approx 0.148 \end{aligned} \tag{11.14}$$

If these three sentences are all the data we have available to use in training our language model, the model is unlikely to generalize well to new sentences. For example, the sentence "*Mulan read her book*" should have a reasonable probability, but the trigram will give it a zero probability simply because we do not have a reliable estimate for $P(read|Mulan)$.

Unlike linguistics, grammaticality is not a strong constraint in the *n*-gram language model. Even though the string is ungrammatical, we may still assign it a high probability if n is small.

11.3. COMPLEXITY MEASURE OF LANGUAGE MODELS

Language can be thought of as an information source whose outputs are words w_i belonging to the vocabulary of the language. The most common metric for evaluating a language model is the word recognition error rate, which requires the participation of a speech recognition system. Alternatively, we can measure the probability that the language model assigns to test word strings without involving speech recognition systems. This is the derivative measure of cross-entropy known as test-set *perplexity*.

The measure of cross-entropy is discussed in Chapter 3. Given a language model that assigns probability $P(\mathbf{W})$ to a word sequence $\mathbf{W}$, we can derive a compression algorithm that encodes the text $\mathbf{W}$ using $-\log_2 P(\mathbf{W})$ bits. The cross-entropy $H(\mathbf{W})$ of a model $P(w_i | w_{i-n+1} \dots w_{i-1})$ on data $\mathbf{W}$, with a sufficiently long word sequence, can be simply approximated as

$$H(\mathbf{W}) = -\frac{1}{N_{\mathbf{W}}} \log_2 P(\mathbf{W}) \tag{11.15}$$

where $N_{\mathbf{W}}$ is the length of the text $\mathbf{W}$ measured in words.

The perplexity $PP(\mathbf{W})$ of a language model $P(\mathbf{W})$ is defined as the reciprocal of the (geometric) average probability assigned by the model to each word in the test set $\mathbf{W}$. This is a measure, related to cross-entropy, known as test-set perplexity:

$$PP(\mathbf{W}) = 2^{H(\mathbf{W})} \tag{11.16}$$

The perplexity can be roughly interpreted as the geometric mean of the branching factor of the text when presented to the language model. The perplexity defined in Eq. (11.16) has two key parameters: a language model and a word sequence. The test-set[4] perplexity evaluates the generalization capability of the language model. The training-set perplexity measures how the language model fits the training data, like the likelihood. It is generally true that lower perplexity correlates with better recognition performance. This is because the perplexity is essentially a statistically weighted word branching measure on the test set. The higher the perplexity, the more branches the speech recognizer needs to consider statistically.

While the perplexity [Eqs. (11.16) and (11.15)] is easy to calculate for the *n*-gram [Eq. (11.12)], it is slightly more complicated to compute for a probabilistic CFG. We can first parse the word sequence and use Eq. (11.9) to compute $P(\mathbf{W})$ for the test-set perplexity. The perplexity can also be applied to nonstochastic models such as CFGs. We can assume they have a uniform distribution in computing $P(\mathbf{W})$.

A language with higher perplexity means that the number of words branching from a previous word is larger on average. In this sense, the perplexity is an indication of the complexity of the language if we have an accurate estimate of $P(\mathbf{W})$. For a given language, the difference between the perplexity of a language model and the true perplexity of the language is an indication of the quality of the model. The perplexity of a particular language model can change dramatically in terms of the vocabulary size, the number of states or grammar rules, and the estimated probabilities. A language model with perplexity *X* has roughly the same difficulty as another language model in which every word can be followed by *X* different words with equal probabilities. Therefore, in the task of continuous digit recognition, the perplexity is 10. Clearly, lower perplexity will generally have less confusion in

[4] We often distinguish between the word sequence from the unseen test data and that from the training data to derive the language model.

recognition. Typical perplexities yielded by *n*-gram models on English text range from about 50 to almost 1000 (corresponding to cross-entropies from about 6 to 10 bits/word), depending on the type of text. In the task of 5,000-word continuous speech recognition for the *Wall Street Journal*, the test-set perplexities of the trigram grammar and the bigram grammar are reported to be about 128 and 176 respectively[5]. In the tasks of 2000-word conversational Air Travel Information System (ATIS), the test-set perplexity of the word trigram model is typically less than 20.

Since perplexity does not take into account the acoustic confusability, we eventually have to measure speech recognition accuracy. For example, if the vocabulary of a speech recognizer contains the E-set of English alphabet: *B, C, D, E, G,* and *T,* we can define a CFG that has a low perplexity value of 6. Such a low perplexity does not guarantee we will have good recognition performance, because of the intrinsic acoustic confusability of the E-set.

11.4. N-Gram Smoothing

One of the key problems in *n*-gram modeling is the inherent data sparseness of real training data. If the training corpus is not large enough, many actually possible word successions may not be well observed, leading to many extremely small probabilities. For example, with several-million-word collections of English text, more than 50% of trigrams occur only once, and more than 80% of trigrams occur less than five times. Smoothing is critical to make estimated probabilities robust for unseen data. If we consider the sentence *Mulan read a book* in the example we discussed in Section 11.2.2, we have:

$$P(read \mid Mulan) = \frac{C(Mulan, read)}{\sum_w C(Mulan, w)} = \frac{0}{1}$$

giving us P(*Mulan read a book*) = 0.

Obviously, this is an underestimate for the probability of "*Mulan read a book*" since there is *some* probability that the sentence occurs in some test set. To show why it is important to give this probability a nonzero value, we turn to the primary application for language models, speech recognition. In speech recognition, if $P(\mathbf{W})$ is zero, the string **W** will never be considered as a possible transcription, regardless of how unambiguous the acoustic signal is. Thus, whenever a string ***W*** such that $P(\mathbf{W}) = 0$ occurs during a speech recognition task, an error will be made. Assigning all strings a nonzero probability helps prevent errors in speech recognition. This is the core issue of smoothing. Smoothing techniques adjust the maximum likelihood estimate of probabilities to produce more robust probabilities for unseen data, although the likelihood for the training data may be hurt slightly.

The name smoothing comes from the fact that these techniques tend to make distributions more uniform, by adjusting low probabilities such as zero probabilities upward, and

---

[5] Some experimental results show that the test-set perplexities for different languages are comparable. For example, French, English, Italian and German have a bigram test-set perplexity in the range of 95 to 133 for newspaper corpora. Italian has a much higher perplexity reduction (a factor of 2) from bigram to trigram because of the high number of function words. The trigram perplexity of Italian is among the lowest in these languages [34].

high probabilities downward. Not only do smoothing methods generally prevent zero probabilities, but they also attempt to improve the accuracy of the model as a whole. Whenever a probability is estimated from few counts, smoothing has the potential to significantly improve the estimation so that it has better generalization capability.

To give an example, one simple smoothing technique is to pretend each bigram occurs once more than it actually does, yielding

$$P(w_i | w_{i-1}) = \frac{1 + C(w_{i-1}, w_i)}{\sum_{w_i} (1 + C(w_{i-1}, w_i))} = \frac{1 + C(w_{i-1}, w_i)}{V + \sum_{w_i} C(w_{i-1}, w_i)} \tag{11.17}$$

where V is the size of the vocabulary. In practice, vocabularies are typically fixed to be tens of thousands of words or less. All words not in the vocabulary are mapped to a single word, usually called the *unknown word*. Let us reconsider the previous example using this new distribution, and let us take our vocabulary V to be the set of all words occurring in the training data S, so that we have V = 11 (with both <s> and </s>).

For the sentence *John read a book*, we now have

$$\begin{aligned} & P(John\ read\ a\ book) \\ & = P(John | <bos>)P(read | John)P(a | read)P(book | a)P(<eos> | book) \\ & \approx 0.00035 \end{aligned} \tag{11.18}$$

In other words, we estimate that the sentence *John read a book* occurs about once every three thousand sentences. This is more reasonable than the maximum likelihood estimate of 0.148 of Eq. (11.14). For the sentence *Mulan read a book*, we have

$$\begin{aligned} & P(Mulan\ read\ a\ book) \\ & = P(Mulan | <bos>)P(read | Mulan)P(a | read)P(book | a)P(<eos> | book) \\ & \approx 0.000084 \end{aligned} \tag{11.19}$$

Again, this is more reasonable than the zero probability assigned by the maximum likelihood model. In general, most existing smoothing algorithms can be described with the following equation:

$$\begin{aligned} & P_{smooth}(w_i | w_{i-n+1}...w_{i-1}) \\ & = \begin{cases} \alpha(w_i | w_{i-n+1}...w_{i-1}) & \text{if } C(w_{i-n+1}...w_i) > 0 \\ \gamma(w_{i-n+1}...w_{i-1}) P_{smooth}(w_i | w_{i-n+2}...w_{i-1}) & \text{if } C(w_{i-n+1}...w_i) = 0 \end{cases} \end{aligned} \tag{11.20}$$

That is, if an *n*-gram has a nonzero count we use the distribution $\alpha(w_i | w_{i-n+1} \dots w_{i-1})$. Otherwise, we backoff to the lower-order *n*-gram distribution $P_{smooth}(w_i | w_{i-n+2} \dots w_{i-1})$, where the scaling factor $\gamma(w_{i-n+1} \dots w_{i-1})$ is chosen to make the conditional distribution sum to one. We refer to algorithms that fall directly in this framework as *backoff models*.

Several other smoothing algorithms are expressed as the linear interpolation of higher- and lower-order *n*-gram models as:

$$P_{smooth}(w_i \mid w_{i-n+1}...w_{i-1})$$
$$= \lambda P_{ML}(w_i \mid w_{i-n+1}...w_{i-1}) + (1-\lambda) P_{smooth}(w_i \mid w_{i-n+2}...w_{i-1}) \tag{11.21}$$

where λ is the interpolation weight that depends on $w_{i-n+1}...w_{i-1}$.We refer to models of this form as interpolated models.

The key difference between backoff and interpolated models is that for the probability of *n*-grams with nonzero counts, interpolated models use information from lower-order distributions while backoff models do not. In both backoff and interpolated models, lower-order distributions are used in determining the probability of *n*-grams with zero counts. Now, we discuss several backoff and interpolated smoothing methods. Performance comparison of these techniques in real speech recognition applications is discussed in Section 11.4.4.

11.4.1. Deleted Interpolation Smoothing

Consider the case of constructing a bigram model on training data where we have that *C(enliven you)* = 0 *and C(enliven thou)* = 0. Then, according to both additive smoothing of Eq. (11.17), we have *P(you|enliven)* = *P(thou|enliven)*. However, intuitively we should have *P(you|enliven)* > *P(thou|enliven*), because the word *you* is much more common than the word *thou* in modern English. To capture this behavior, we can interpolate the bigram model with a unigram model. A unigram model conditions the probability of a word on no other words, and just reflects the frequency of that word in text. We can linearly interpolate a bigram model and a unigram model as follows:

$$P_I(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda) P(w_i) \tag{11.22}$$

where $0 \le \lambda \le 1$. Because *P(you|enliven) = P(thou|enliven)*=0 while presumably *P(you)* > *P(thou)*, we will have that $P_I(you \mid enliven) > P_I(thou \mid enliven)$ as desired.

In general, it is useful to interpolate higher-order *n*-gram models with lower-order *n*-gram models, because when there is insufficient data to estimate a probability in the higher-order model, the lower-order model can often provide useful information. An elegant way of performing this interpolation is given as follows

$$P_I(w_i|w_{i-n+1}\dots w_{i-1})$$
$$= \lambda_{w_{i-n+1...}w_{i-1}} P(w_i|w_{i-n+1}\dots w_{i-1}) + (1-\lambda_{w_{i-n+1...}w_{i-1}}) P_I(w_i|w_{i-n+2}\dots w_{i-1}) \tag{11.23}$$

That is, the *n*th-order smoothed model is defined *recursively* as a linear interpolation between the *n*th-order maximum likelihood model and the (*n*-1)th-order smoothed model. To end the recursion, we can take the smoothed first-order model to be the maximum likelihood distribution (unigram), or we can take the smoothed zeroth-order model to be the uniform distribution. Given a fixed $P(w_i|w_{i-n+1}\dots w_{i-1})$, it is possible to search efficiently for the interpolation parameters using the deleted interpolation method discussed in Chapter 9.

Notice that the optimal $\lambda_{w_{i-n+1}...w_{i-1}}$ is different for different histories $w_{i-n+1}...w_{i-1}$. For example, for a context we have seen thousands of times, a high λ will be suitable, since the higher-order distribution is very reliable; for a history that has occurred only once, a lower λ is appropriate. Training each parameter $\lambda_{w_{i-n+1}...w_{i-1}}$ independently can be harmful; we need an enormous amount of data to train so many independent parameters accurately. One possibility is to divide the $\lambda_{w_{i-n+1}...w_{i-1}}$ into a moderate number of partitions or buckets, constraining all $\lambda_{w_{i-n+1}...w_{i-1}}$ in the same bucket to have the same value, thereby reducing the number of independent parameters to be estimated. Ideally, we should tie together those $\lambda_{w_{i-n+1}...w_{i-1}}$ that we have a prior reason to believe should have similar values.

11.4.2. Backoff Smoothing

Backoff smoothing is attractive because it is easy to implement for practical speech recognition systems. The Katz backoff model is the canonical example we discuss in this section. It is based on the Good-Turing smoothing principle.

11.4.2.1. Good-Turing Estimates and Katz Smoothing

The Good-Turing estimate is a smoothing technique to deal with infrequent *n*-grams. It is not used by itself for *n*-gram smoothing, because it does not include the combination of higher-order models with lower-order models necessary for good performance. However, it is used as a tool in several smoothing techniques. The basic idea is to partition *n*-grams into groups depending on their frequency (i.e. how many time the *n*-grams appear in the training data) such that the parameter can be smoothed based on *n*-gram frequency.

The Good-Turing estimate states that for any *n*-gram that occurs *r* times, we should pretend that it occurs r^* times as follows:

$$r^* = (r+1)\frac{n_{r+1}}{n_r} \tag{11.24}$$

where n_r is the number of *n*-grams that occur exactly *r* times in the training data. To convert this count to a probability, we just normalize: for an *n*-gram *a* with *r* counts, we take

$$P(a) = \frac{r^*}{N} \tag{11.25}$$

where $N = \sum_{r=0}^{\infty} n_r r^*$. Notice that $N = \sum_{r=0}^{\infty} n_r r^* = \sum_{r=0}^{\infty}(r+1)n_{r+1} = \sum_{r=0}^{\infty} n_r r$ *i.e.* *N* is equal to the original number of counts in the distribution [28].

Katz smoothing extends the intuitions of the Good-Turing estimate by adding the combination of higher-order models with lower-order models [38]. Take the bigram as our example, Katz smoothing suggested using the Good-Turing estimate for nonzero counts as follows:

$$C^*(w_{i-1}w_i) = \begin{cases} d_r r & \text{if } r>0 \\ \alpha(w_{i-1})P(w_i) & \text{if } r=0 \end{cases} \tag{11.26}$$

where d_r is approximately equal to r^*/r. That is, all bigrams with a nonzero count r are *discounted* according to a discount ratio d_r, which implies that the counts subtracted from the nonzero counts are distributed among the zero-count bigrams according to the next lower-order distribution, e.g., the unigram model. The value $\alpha(w_{i-1})$ is chosen to equalize the total number of counts in the distribution, i.e., $\sum_{w_i} C^*(w_{i-1}w_i) = \sum_{w_i} C\ (w_{i-1}w_i)$. The appropriate value for $\alpha(w_{i-1})$ is computed so that the smoothed bigram satisfies the probability constraint:

$$\alpha(w_{i-1}) = \frac{1-\sum_{w_i:C(w_{i-1}w_i)>0} P^*(w_i|w_{i-1})}{\sum_{w_i:C(w_{i-1}w_i)=0} P(w_i)} = \frac{1-\sum_{w_i:C(w_{i-1}w_i)>0} P^*(w_i|w_{i-1})}{1-\sum_{w_i:C(w_{i-1}w_i)>0} P(w_i)} \tag{11.27}$$

To calculate $P^*(w_i|w_{i-1})$ from the corrected count, we just normalize:

$$P^*(w_i \mid w_{i-1}) = \frac{C^*(w_{i-1}w_i)}{\sum_{w_k} C^*(w_{i-1}w_k)} \tag{11.28}$$

In Katz implementation, the d_r are calculated as follows: large counts are taken to be reliable, so they are not discounted. In particular, Katz takes d_r = 1 for all $r > k$ for some k, say k in the range of 5 to 8. The discount ratios for the lower counts $r \le k$ are derived from the Good-Turing estimate applied to the global bigram distribution; that is, n_r in Eq. (11.24) denotes the total number of bigrams that occur exactly r times in the training data. These d_r are chosen such that

- the resulting discounts are proportional to the discounts predicted by the Good-Turing estimate, and
- the total number of counts discounted in the global bigram distribution is equal to the total number of counts that should be assigned to bigrams with zero counts according to the Good-Turing estimate.

The first constraint corresponds to the following equation:

$$d_r = \mu \frac{r^*}{r} \tag{11.29}$$

for $r \in \{1, \dots k\}$ with some constant μ. The Good-Turing estimate predicts that the total mass assigned to bigrams with zero counts is $n_0 \frac{n_1}{n_0} = n_1$, and the second constraint corresponds to the equation

$$\sum_{r=1}^{k} n_r (1 - d_r) r = n_1 \tag{11.30}$$

Based on Eq. (11.30), the unique solution is given by:

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} \tag{11.31}$$

Katz smoothing for higher-order *n*-gram models is defined analogously. The Katz *n*-gram backoff model is defined in terms of the Katz (*n*-1)-gram model. To end the recursion, the Katz unigram model is taken to be the maximum likelihood unigram model. It is usually necessary to smooth n_r when using the Good-Turing estimate, e.g., for those n_r that are very low. However, in Katz smoothing this is not essential because the Good-Turing estimate is used only for small counts *r<=k*, and n_r is generally fairly high for these values of *r*. The procedure of Katz smoothing can be summarized as in Algorithm 11.2.

In fact, the Katz backoff model can be expressed in terms of the interpolated model defined in Eq. (11.23), in which the interpolation weight is obtained via Eq. (11.26) and (11.27).

ALGORITHM 11.2 KATZ SMOOTHING

$$P_{Katz}(w_i \mid w_{i-1}) = \begin{cases} C(w_{i-1}w_i)/C(w_{i-1}) & \text{if } r > k \\ d_r C(w_{i-1}w_i)/C(w_{i-1}) & \text{if } k \ge r > 0 \\ \alpha(w_{i-1}) P(w_i) & \text{if } r = 0 \end{cases}$$

where $d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$ and $\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : r>0} P_{Katz}(w_i \mid w_{i-1})}{1 - \sum_{w_i : r>0} P(w_i)}$

11.4.2.2. Alternative Backoff Models

In a similar manner to the Katz backoff model, there are other ways to discount the probability mass. For instance, *absolute discounting* involves subtracting a fixed discount $D <= 1$ from each nonzero count. If we express the absolute discounting in term of interpolated models, we have the following:

$$P_{abs}(w_i | w_{i-n+1} \ldots w_{i-1}) \\ = \frac{\max\{C(w_{i-n+1} \ldots w_i) - D, 0\}}{\sum_{w_i} C(w_{i-n+1} \ldots w_i)} + (1 - \lambda_{w_{i-n+1} \ldots w_{i-1}}) P_{abs}(w_i | w_{i-n+2} \ldots w_{i-1}) \quad (11.32)$$

To make this distribution sum to 1, we normalize it to determine $\lambda_{w_{i-n+1} \ldots w_{i-1}}$. Absolute discounting is explained with the Good-Turing estimate. Empirically the average Good-Turing discount $r - r^*$ associated with n-grams of larger counts (r over 3) is largely constant over r.

Consider building a bigram model on data where there exists a word that is very common, say *Francisco*, that occurs only after a single word, say *San*. Since *C(Francisco)* is high, the unigram probability *P(Francisco)* will be high, and an algorithm such as absolute discounting or Katz smoothing assigns a relatively high probability to occurrence of the word *Francisco* after novel bigram histories. However, intuitively this probability should not be high, since in the training data the word *Francisco* follows only a single history. That is, perhaps *Francisco* should receive a low unigram probability, because the only time the word occurs is when the last word is *San*, in which case the bigram probability models its probability well.

Extending this line of reasoning, perhaps the unigram probability used should not be proportional to the number of occurrences of a word, but instead to the number of different words that it follows. To give an intuitive argument, imagine traversing the training data sequentially and building a bigram model on the preceding data to predict the current word. Then, whenever the current bigram does not occur in the preceding data, the unigram probability becomes a large factor in the current bigram probability. If we assign a count to the corresponding unigram whenever such an event occurs, then the number of counts assigned to each unigram is simply the number of different words that it follows. In Kneser-Ney smoothing [40], the lower-order n-gram is not proportional to the number of occurrences of a word, but instead to the number of *different* words that it follows. We summarize the Kneser-Ney backoff model in Algorithm 11.3.

ALGORITHM 11.3 KNESER-NEY BIGRAM SMOOTHING

$$P_{KN}(w_i \mid w_{i-1}) = \begin{cases} \dfrac{\max\{C(w_{i-1}w_i) - D, 0\}}{C(w_{i-1})} & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}) P_{KN}(w_i) & \text{otherwise} \end{cases}$$

where $P_{KN}(w_i) = \mathbb{C}(\bullet w_i) / \sum_{w_i} \mathbb{C}(\bullet w_i)$, $\mathbb{C}(\bullet w_i)$ is the number of unique words preceding w_i.

$\alpha(w_{i-1})$ is chosen to make the distribution sum to 1 so that we have:

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : C(w_{i-1}w_i) > 0} \dfrac{\max\{C(w_{i-1}w_i) - D, 0\}}{C(w_{i-1})}}{1 - \sum_{w_i : C(w_{i-1}w_i) > 0} P_{KN}(w_i)}$$

Kneser-Ney smoothing is an extension of other backoff models. Most of the previous models used the lower-order *n*-grams trained with ML estimation. Kneser-Ney smoothing instead considers a lower-order distribution as a significant factor in the combined model such that they are optimized together with other parameters. To derive the formula, more generally, we express it in terms of the interpolated model specified in Eq. (11.23) as:

$$\begin{aligned} & P_{KN}(w_i \mid w_{i-n+1} \ldots w_{i-1}) \\ & = \frac{\max\{C(w_{i-n+1} \ldots w_i) - D, 0\}}{\sum_{w_i} C(w_{i-n+1} \ldots w_i)} + (1 - \lambda_{w_{i-n+1} \ldots w_{i-1}}) P_{KN}(w_i \mid w_{i-n+2} \ldots w_{i-1}) \end{aligned} \tag{11.33}$$

To make this distribution sum to 1, we have:

$$1 - \lambda_{w_{i-n+1} \ldots w_{i-1}} = \frac{D}{\sum_{w_i} C(w_{i-n+1} \ldots w_i)} \mathbb{C}(w_{i-n+1} \ldots w_{i-1} \bullet) \tag{11.34}$$

where $\mathbb{C}(w_{i-n+1} \ldots w_{i-1} \bullet)$ is the number of unique words that follow the history $w_{i-n+1} \cdots w_{i-1}$. This equation enables us to interpolate the lower-order distribution with all words, not just with words that have zero counts in the higher-order distribution.

Now, take the bigram case as an example. We need to find a unigram distribution $P_{KN}(w_i)$ such that the marginal of the bigram smoothed distributions should match the marginal of the training data:

$$\frac{C(w_i)}{\sum_{w_i} C(w_i)} = \sum_{w_{i-1}} P_{KN}(w_{i-1} w_i) = \sum_{w_{i-1}} P_{KN}(w_i \mid w_{i-1}) P(w_{i-1}) \tag{11.35}$$

For $P(w_{i-1})$, we simply take the distribution found in the training data

$$P(w_{i-1}) = \frac{C(w_{i-1})}{\sum_{w_{i-1}} C(w_{i-1})} \tag{11.36}$$

We substitute Eq. (11.33) in Eq. (11.35). For the bigram case, we have:

$$\begin{aligned}
& C(w_i) \\
& = \sum_{w_{i-1}} C(w_{i-1})[\frac{\max\{C(w_{i-1}w_i) - D, 0\}}{\sum_{w_i} C(w_{i-1}w_i)} + \frac{D}{\sum_{w_i} C(w_{i-1}w_i)} \mathbb{C}(w_{i-1}\bullet) P_{KN}(w_i)] \\
& = \sum_{w_{i-1}:C(w_{i-1}w_i)>0} C(w_{i-1}) \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} + \sum_{w_{i-1}} C(w_{i-1}) \frac{D}{C(w_{i-1})} \mathbb{C}(w_{i-1}\bullet) P_{KN}(w_i) \\
& = C(w_i) - \mathbb{C}(\bullet w_{i-1}) D + D P_{KN}(w_i) + D P_{KN}(w_i) \sum_{w_{i-1}} \mathbb{C}(w_{i-1}\bullet)
\end{aligned} \tag{11.37}$$

Solving the equation, we get

$$P_{KN}(w_i) = \frac{\mathbb{C}(\bullet w_i)}{\sum_{w_i} \mathbb{C}(\bullet w_i)} \tag{11.38}$$

which can be generalized to higher-order models:

$$P_{KN}(w_i \mid w_{i-n+2} \ldots w_{i-1}) = \frac{\mathbb{C}(\bullet w_{i-n+2} \ldots w_i)}{\sum_{w_i} \mathbb{C}(\bullet w_{i-n+2} \ldots w_i)} \tag{11.39}$$

where $\mathbb{C}(\bullet w_{i-n+2} \ldots w_i)$ is the number of different words that precede $w_{i-n+2} \ldots w_i$.

In practice, instead of using a single discount D for all nonzero counts as in Kneser-Ney smoothing, we can have a number of different parameters (D_i) that depend on the range of counts:

$$\begin{aligned}
& P_{KN}(w_i \mid w_{i-n+1} \ldots w_{i-1}) \\
& = \frac{C(w_{i-n+1} \ldots w_i) - D(C(w_{i-n+1} \ldots w_i))}{\sum_{w_i} C(w_{i-n+1} \ldots w_i)} + \\
& + \gamma(w_{i-n+1} \ldots w_{i-1}) P_{KN}(w_i \mid w_{i-n+2} \ldots w_{i-1})
\end{aligned} \tag{11.40}$$

This modification is motivated by evidence that the ideal average discount for *n*-grams with one or two counts is substantially different from the ideal average discount for *n*-grams with higher counts.

11.4.3. Class *n*-grams

As discussed in Chapter 2, we can define classes for words that exhibit similar semantic or grammatical behavior. This is another effective way to handle the data sparsity problem. Class-based language models have been shown to be effective for rapid adaptation, training on small data sets, and reduced memory requirements for real-time speech applications.

For any given assignment of a word w_i to class c_i, there may be many-to-many mappings, *e.g.*, a word w_i may belong to more than one class, and a class c_i may contain more than one word. For the sake of simplicity, assume that a word w_i can be uniquely mapped to only one class c_i. The *n*-gram model can be computed based on the previous *n*-1 classes:

$$P(w_i|c_{i-n+1}\ldots c_{i-1}) = P(w_i \mid c_i)P(c_i|c_{i-n+1}\ldots c_{i-1}) \tag{11.41}$$

where $P(w_i|c_i)$ denotes the probability of word w_i given class c_i in the current position, and $P(c_i|c_{i-n+1}\ldots c_{i-1})$ denotes the probability of class c_i given the class history. With such a model, we can learn the class mapping $w \to c$ from either a training text or task knowledge we have about the application. In general, we can express the class trigram as:

$$P(\mathbf{W}) = \sum_{c_1\ldots c_n}\prod_i P(w_i \mid c_i)P(c_i \mid c_{i-2,}c_{i-1}) \tag{11.42}$$

If the classes are nonoverlapping, i.e. a word may belong to only one class, then Eq. (11.42) can be simplified as:

$$P(\mathbf{W}) = \prod_i P(w_i \mid c_i)P(c_i \mid c_{i-2,}c_{i-1}) \tag{11.43}$$

If we have the mapping function defined, we can easily compute the class *n*-gram. We can estimate the empirical frequency of each word $C(w_i)$, and of each class $C(c_i)$. We can also compute the empirical frequency that a word from one class will be followed immediately by a word from another $C(c_{i-1}c_i)$. As a typical example, the bigram probability of a word given the prior word (class) can be estimated as

$$P(w_i|w_{i-1}) \doteq P(w_i|c_{i-1}) = P(w_i \mid c_i)\, P(c_i|c_{i-1}) = \frac{C(w_i)}{C(c_i)}\frac{C(c_{i-1}c_i)}{C(c_{i-1})} \tag{11.44}$$

For general-purpose large vocabulary dictation applications, the class-based *n*-gram has not significantly improved the recognition accuracy. It is mainly used as a backoff model to complement the lower-order *n*-grams for better smoothing. Nevertheless, for limited domain speech recognition, the class-based *n*-gram is very helpful as the class can efficiently encode semantic information for improved key word spotting and speech understanding accuracy.

11.4.3.1. Rule-Based Classes

There are a number of ways to cluster words together based on the syntactic-semantic information that exists for the language and the task. For example, part-of-speech can be generally used to produce a small number of classes although this may lead to significantly increased perplexity. Alternatively, if we have domain knowledge, it is often advantageous to cluster together words that have a similar semantic functional role. For example, if we need to build a conversational system for air travel information systems, we can group the name of different airlines such as *United Airlines*, *KLM*, and *Air China*, into a broad *airline class*. We can do the same thing for the names of different airports such as *JFK*, *Narita*, and *Heathrow*, the names of different cities like *Beijing*, *Pittsburgh*, and *Moscow*, and so on. Such an approach is particularly powerful, since the amount of training data is always limited. With generalized broad classes of semantically interpretable meaning, it is easy to add a new airline such as *Redmond Air* into the classes if there is indeed a start-up airline named *Redmond Air* that the system has to incorporate. The system is now able to assign a reasonable probability to a sentence like "*Show me all flights of Redmond Air from Seattle to Boston*" in a similar manner as "*Show me all flights of United Airlines from Seattle to Boston*". We only need to estimate the probability of *Redmond Air*, given the airline class c_i. We can use the existing class *n*-gram model that contains the broad structure of the air travel information system as it is.

Without such a broad interpretable class, it would be extremely difficult to deal with new names the system needs to handle, although these new names can always be mapped to the special class of the unknown word or proper noun classes. For these new words, we can alternatively map them into a word that has a similar syntactic and semantic role. Thus, the new word inherits all the possible word trigram relationships that may be very similar to those of the existing word observed with the training data.

11.4.3.2. Data-driven Classes

For a general-purpose dictation application, it is impractical to derive functional classes in the same manner as the domain-specific conversational system that focuses on a narrow task. Instead, data-driven clustering algorithms have been used to generalize the concept of word similarities, which is in fact a search procedure to find a class label for each word with a predefined objective function. The set of words with the same class label is called a cluster. We can use the maximum likelihood criterion as the objective function for a given training corpus and a given number of classes, which is equivalent to minimizing the perplexity for the training corpus. Once again, the EM algorithm can be used here. Each word can be initialized to a random cluster (class label). At each iteration, every word is moved to the class that produces the model with minimum perplexity [9, 48]. The perplexity modifications can be calculated independently, so that each word is evaluated as if all other word classes were held fixed. The algorithm converges when no single word can be moved to another class in a way that reduces the perplexity of the clustered *n*-gram model.

One special kind of class *n*-gram models is based on the decision tree as discussed in Chapter 4. We can use it to create equivalent classes for words in the history, so that can we have a compact long-distance *n*-gram language model [2]. The sequential decomposition, as expressed in Eq. (11.12), is approximated as:

$$P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i | E(w_1, w_2, \ldots, w_{i-1})) = \prod_{i=1}^{n} P(w_i | E(\mathbf{h})) \tag{11.45}$$

where $E(\mathbf{h})$ denotes a many-to-one mapping function that groups word histories **h** into some equivalence classes. It is important to have a scheme that can provide adequate information about the history so it can serve as a basis for prediction. In addition, it must yield a set of classes that can be reliably estimated. The decision tree method uses entropy as a criterion in developing the equivalence classes that can effectively incorporate long-distance information. By asking a number of questions associated with each node, the decision tree can classify the history into a small number of equivalence classes. Each leaf of the tree, thus, has the probability $P(w_i | E(w_1 \ldots w_{i-1}))$ that is derived according to the number of times the word w_i is found in the leaf. The selection of questions in building the tree can be infinite. We can consider, not only the syntactic structure, but also semantic meaning to derive permissible questions from which the entropy criterion would choose. A full-fledged question set that is based on detailed analysis of the history is beyond the limit of our current computing resources. As such, we often use the membership question to check each word in the history.

11.4.4. Performance of *n*-gram Smoothing

The performance of various smoothing algorithms depends on factors such as the training-set sizes. There is a strong correlation between the test-set perplexity and word error rate. Smoothing algorithms leading to lower perplexity generally result in a lower word error rate. Among all the methods discussed here, the Kneser-Ney method slightly outperforms other algorithms over a wide range of training-set sizes and corpora, and for both bigram and trigram models. Albeit the difference is not large, the good performance of the Kneser-Ney smoothing is due to the modified backoff distributions. The Katz algorithms and deleted interpolation smoothing generally yield the next best performance. All these three smoothing algorithms perform significantly better than the *n*-gram model without any smoothing. The deleted interpolation algorithm performs slightly better than the Katz method in sparse data situations, and the reverse is true when data are plentiful. Katz's algorithm is particularly good at smoothing larger counts; these counts are more prevalent in larger data sets.

Class *n*-grams offer different kind of smoothing. While clustered *n*-gram models often offer no significant test-set perplexity reduction in comparison to the word *n*-gram model, it is beneficial to smooth the word *n*-gram model via either backoff or interpolation methods. For example, the decision-tree based long-distance class language model does not offer significantly improved speech recognition accuracy until it is interpolated with the word trigram. They are effective as a domain-specific language model if the class can accommodate domain-specific information.

Smoothing is a fundamental technique for statistical modeling, important not only for language modeling but for many other applications as well. Whenever data sparsity is an issue, smoothing can help performance, and data sparsity is almost always an issue in statistical modeling. In the extreme case, where there is so much training data that all parameters can be accurately trained without smoothing, you can almost always expand the model, such as by moving to a higher-order n-gram model, to achieve improved performance. With more parameters, data sparsity becomes an issue again, but a proper smoothing model is usually more accurate than the original model. Thus, no matter how much data you have, smoothing can almost always help performance, and for a relatively small effort.

11.5. ADAPTIVE LANGUAGE MODELS

Dynamic adjustment of the language model parameter, such as *n*-gram probabilities, vocabulary size, and the choice of words in the vocabulary, is important, since the topic of conversation is highly nonstationary [4, 33, 37, 41, 46]. For example, in a typical dictation application, a particular set of words in the vocabulary may suddenly burst forth and then become dormant later, based on the current conversation. Because the topic of the conversation may change from time to time, the language model should be dramatically different based on the topic of the conversation. We discuss several adaptive techniques that can improve the quality of the language model based on the real usage of the application.

11.5.1. Cache Language Models

To adjust word frequencies observed in the current conversation, we can use a dynamic *cache* language model [41]. The basic idea is to accumulate word *n*-grams dictated so far in the current document and use these to create a local dynamic *n*-gram model such as bigram $P_{cache}(w_i|w_{i-1})$. Because of limited data and nonstationary nature, we should use a lower-order language model that is no higher than a trigram model $P_{cache}(w_i|w_{i-2}w_{i-1})$, which can be interpolated with the dynamic bigram and unigram. Empirically, we need to normally give a high weight to the unigram cache model, because it is better trained with the limited data in the cache.

With the cache trigram, we interpolate it with the static *n*-gram model $P_s(w_i|w_{i-n+1}\dots w_{i-1})$. The interpolation weight can be made to vary with the size of the cache.

$$\begin{aligned}&P_{cache}(w_i \mid w_{i-n+1}...w_{i-1})\\&=\lambda_c P_s(w_i \mid w_{i-n+1}...w_{i-1})+(1-\lambda_c)P_{cache}(w_i \mid w_{i-2}w_{i-1})\end{aligned} \tag{11.46}$$

The cache model is desirable in practice because of its impressive empirical performance improvement. In a dictation application, we often encounter new words that are not in the static vocabulary. The same words also tend to be repeated in the same article. The cache model can address this problem effectively by adjusting the parameters continually as recognition and correction proceed for incrementally improved performance. A noticeable benefit is that we can better predict words belonging to fixed phrases such as *Windows NT*, and *Bill Gates*.

11.5.2. Topic-Adaptive Models

The topic can change over time. Such topic or style information plays a critical role in improving the quality of the static language model. For example, the prediction of whether the word following the phrase *the operating* is *system* or *table* can be improved substantially by knowing whether the topic of discussion is related to computing or medicine.

Domain or topic-clustered language models split the language model training data according to topic. The training data may be divided using the known category information or using automatic clustering. In addition, a given segment of the data may be assigned to multiple topics. A topic-dependent language model is then built from each cluster of the training data. Topic language models are combined using linear interpolation or other methods such as maximum entropy techniques discussed in Section 11.5.3.

We can avoid any pre-defined clustering or segmentation of the training data. The reason is that the best clustering may become apparent only when the current topic of discussion is revealed. For example, when the topic is hand-injury to baseball player, the pre-segmented clusters of topic *baseball* & *hand-injuries* may have to be combined. This leads to a union of the two clusters, whereas the ideal dataset is obtained by the intersection of these clusters. In general, various combinations of topics lead to a combinatorial explosion in the number of compound topics, and it appears to be a difficult task to anticipate all the needed combinations beforehand.

We base our determination of the most suitable language model data to build a model upon the particular history of a given document. For example, we can use it as a query against the entire training database of documents using *information retrieval* techniques [57]. The documents in the database can be ranked by relevance to the query. The most relevant documents are then selected as the adaptation set for the topic-dependent language model. The process can be repeated as the document is updated.

There are two major steps we need to consider here. The first involves using the available document history to retrieve similar documents from the database. The second consists of using the similar document set retrieved in the first step to adapt the general or topic-independent language model. Available document history depends upon the design and the requirements of the recognition system. If the recognition system is designed for live-mode application, where the recognition results must be presented to the user with a small delay, the available document history will be the history of the document user created so far. On the other hand, in a recognition system designed for batch operation, the amount of time taken by the system to recognize speech is of little consequence to the user. In the batch mode, therefore, a multi-pass recognition system can be used, and the document history will be the recognizer transcript produced in the current pass.

The well-known information retrieval measure called *TFIDF* can be used to locate similar documents in the training database [57]. The term frequency (TF) tf_{ij} is defined as the frequency of the *j*th term in the document D_i, the unigram count of the term *j* in the document D_i. The inverse document frequency (IDF) idf_j is defined as the frequency of the *j*th term over the entire database of documents, which can be computed as:

$$idf_j = \frac{\text{Total number of documents}}{\text{Number of documents containing term } j} \tag{11.47}$$

The combined TF-IDF measure is defined as:

$$TFIDF_{ij} = tf_{ij} \log(idf_i) \tag{11.48}$$

The combination of TF and IDF can help to retrieve similar documents. It highlights words of particular interest to the query (via TF), while de-emphasizing common words that appear across different documents (via IDF). Each document including the query itself, can be represented by the TFIDF vector. Each element of the vector is the TFIDF value that corresponds to a word (or a term) in the vocabulary. Similarity between the two documents is then defined to be the cosine of the angle between the corresponding vectors. Therefore, we have:

$$Similarity(D_{i,} D_j) = \frac{\sum_k tfidf_{ik} * tfidf_{jk}}{\sqrt{\sum_k (tfidf_{ik})^2 * \sum_k (tfidf_{jk})^2}} \tag{11.49}$$

All the documents in the training database are ranked by the decreasing similarity between the document and the history of the current document dictated so far, or by a topic of particular interest to the user. The most similar documents are selected as the adaptation set for the topic-adaptive language model [46].

11.5.3. Maximum Entropy Models

The language model we have discussed so far combines different *n*-gram models via linear interpolation. A different way to combine sources is the maximum entropy approach. It constructs a single model that attempts to capture all the information provided by the various knowledge sources. Each such knowledge source is reformulated as a set of constraints that the desired distribution should satisfy. These constraints can be, for example, marginal distributions of the combined model. Their intersection, if not empty, should contain a set of probability functions that are consistent with these separate knowledge sources. Once the desired knowledge sources have been incorporated, we make no other assumption about other constraints, which leads to choosing the flattest of the remaining possibilities, the one with the highest entropy. The maximum entropy principle can be stated as follows:

- Reformulate different information sources as constraints to be satisfied by the target estimate.

- Among all probability distributions that satisfy these constraints, choose the one that has the highest entropy.

Given a general event space $\{\mathbf{X}\}$, let $P(\mathbf{X})$ denote the combined probability function. Each constraint is associated with a characteristic function of a subset of the sample space, $f_i(\mathbf{X})$. The constraint can be written as:

$$\sum_{\mathbf{X}} P(\mathbf{X}) f_i(\mathbf{X}) = E_i \tag{11.50}$$

where E_i is the corresponding desired expectation for $f_i(\mathbf{X})$, typically representing the required marginal probability of $P(\mathbf{X})$. For example, to derive a word trigram model, we can reformulate Eq. (11.50) so that constraints are introduced for unigram, bigram, and trigram probabilities. These constraints are usually set only where marginal probabilities can be estimated from a corpus. For example, the unigram constraint can be expressed as

$$f_{w_1}(w) = \begin{cases} 1 & \text{if } w{=}w_1 \\ 0 & \text{otherwise} \end{cases} \tag{11.51}$$

The desired value E_{w_1} can be the empirical expectation in the training data, $\sum_{w \in training\ data} f_{w_1}(w)/N$, and the associated constraint is

$$\sum_{\mathbf{h}} P(\mathbf{h}) \sum_{w} P(w \mid \mathbf{h}) f_{w_1}(w) = E_{w_1} \tag{11.52}$$

where $\mathbf{h}$ is the word history preceding word w.

We can choose $P(\mathbf{X})$ to diverge minimally from some other known probability function $Q(\mathbf{X})$, that is, to minimize the divergence function:

$$\sum_{\mathbf{X}} P(\mathbf{X}) \log \frac{P(\mathbf{X})}{Q(\mathbf{X})} \tag{11.53}$$

When $Q(\mathbf{X})$ is chosen as the uniform distribution, the divergence is equal to the negative of entropy with a constant. Thus minimizing the divergence function leads to maximizing the entropy. Under a minor consistent assumption, a unique solution is guaranteed to exist in the form [20]:

$$P(\mathbf{X}) \propto \prod_i \mu_i^{f_i(\mathbf{X})} \tag{11.54}$$

where μ_i is an unknown constant to be found. To search the exponential family defined by Eq. (11.54) for the μ_i that make $P(\mathbf{X})$ satisfy all the constraints, an iterative algorithm called generalized iterative scaling exists [20]. It guarantees to converge to the solution with some arbitrary initial μ_i. Each iteration creates a new estimate $P(\mathbf{X})$, which is improved in the sense that it matches the constraints better than its [20]. One of the most effective applications of the maximum entropy model is to integrate the cache constraint into the language model directly, instead of interpolating the cache *n*-gram with the static *n*-gram. The new constraint is that the marginal distribution of the adapted model is the same as the lower-order *n*-gram in the cache [56]. In practice, the maximum entropy method has not offered any significant improvement in comparison to the cache modle discussed in Section 11.5.1.

11.6. PRACTICAL ISSUES

In a speech recognition system, every string of words $\mathbf{W} = w_1 w_2 \ldots w_n$ taken from the prescribed vocabulary can be assigned a probability, which is interpreted as the *a priori* probability to guide the recognition process and is a contributing factor in the determination of the final transcription from a set of partial hypothesis. Without language modeling, the entire vocabulary must be considered at every decision point. It is impossible to eliminate many candidates from consideration, or alternatively to assign higher probabilities to some candidates than others to considerably reduce recognition costs and errors.

11.6.1. Vocabulary Selection

For most speech recognition systems, an inflected form is considered as a different word. This is because these inflected forms typically have different pronunciations, syntactic roles, and usage patterns. So the words *work*, *works*, *worked*, and *working* are counted as four different words in the vocabulary.

We prefer to have a smaller vocabulary size, since this eliminates potential confusable candidates in speech recognition, leading to improved recognition accuracy. However, the limited vocabulary size imposes a severe constraint on the users and makes the system less flexible. In practice, the percentage of the Out-Of-Vocabulary (OOV) word rate directly affects the perceived quality of the system. Thus, we need to balance two kinds of errors, the OOV rate and the word recognition error rate. We can have a larger vocabulary to minimize the OOV rate if the system resources permit. We can minimize the expected OOV rate of the test data with a given vocabulary size. A corpus of text is used in conjunction with dictionaries to determine appropriate vocabularies.

The availability of various types and amounts of training data, from various time periods, affects the quality of the derived vocabulary. Given a collection of training data, we can create an ordered word list with the lowest possible OOV curve, such that, for any desired vocabulary size *V*, a minimum-OOV-rate vocabulary can be derived by taking the most frequent *V* words in that list. Viewed this way, the problem becomes one of estimating unigram probabilities of the test distribution, and then ordering the words by these estimates.

As illustrated in Figure 11.6, the perplexity generally increases with the vocabulary size, albeit it really does not make much sense to compare the perplexity of different vocabulary sizes. There are generally more competing words for a given context when the vocabulary size becomes big, which leads to increased recognition error rate. In practice, this is offset by the OOV rate, which decreases with the vocabulary size as illustrated in Figure 11.7. If we keep the vocabulary size fixed, we need more than 200,000 words in the vocabulary to have 99.5% English words coverage. For more inflectional languages such as German, larger vocabulary sizes are required to achieve coverage similar to that of English.[6]

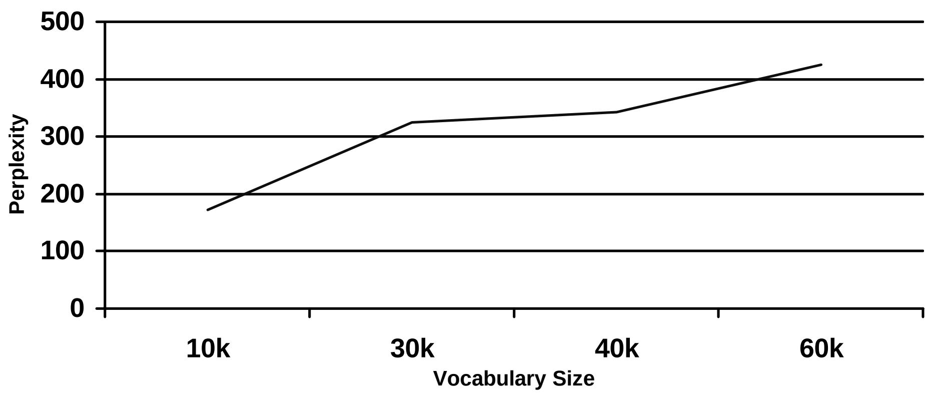


Figure 11.6 The perplexity of bigram with different vocabulary sizes. The training set consists of 500 million words derived from various sources, including newspapers and email. The test set comes from the whole Microsoft Encarta, an encyclopedia that has a wide coverage of different topics.

In practice, it is far more important to use data from a specific topic or domain, if we know in what domain the speech recognizer is used. In general, it is also important to consider coverage of a specific time period. We should use training data from that period, or as close to it as possible. For example, if we know we will talk only about air travel, we benefit from using the air-travel related vocabulary and language model. This point is well illustrated by the fact that the perplexity of the domain-dependent bigram can be reduced by more than a factor of five over the general-purpose English trigram.

For a user of a speech recognition system, a more personalized vocabulary can be much more effective than a general fixed vocabulary. The coverage can be dramatically improved as customized new words are added to a starting static vocabulary of 20,000. Typically, the coverage of such a system can be improved from 93% to more than 98% after 1000-4000 customized words are added to the vocabulary [18].

[6] The OOV rate of German is about twice as high as that of English with a 20k-word vocabulary [34].

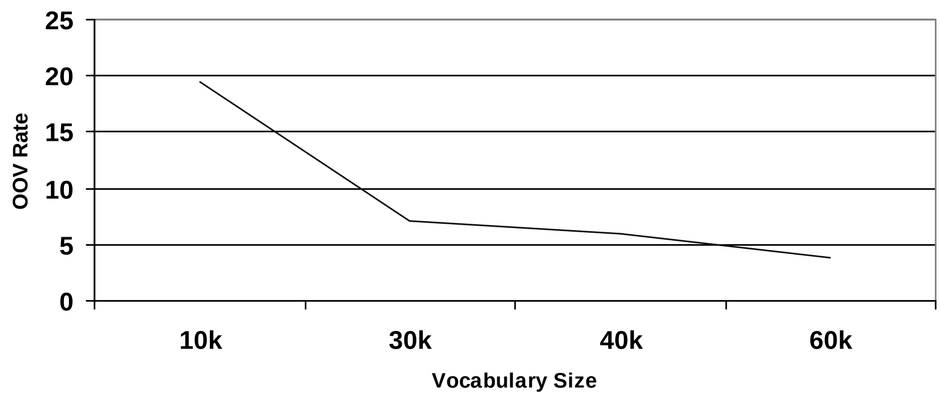


Figure 11.7 The OOV rate with different vocabulary size. The training set consists of 500 million words derived from various sources including newspaper and email. The test set came from the whole Microsoft Encarta encyclopedia.

In North American general business English, the least frequent words among the most frequent 60,000 have a frequency of about 1:7,000,000. In optimizing a 60,000-word vocabulary we need to distinguish words with frequency of 1:7,000,000 from those that are slightly less frequent. To differentiate somewhat reliably between a 1:7,000,000 word and, say, a 1:8,000,000 word, we need to observe them enough times for the difference in their counts to be statistically reliable. For constructing a decent vocabulary, it is important that most such words are ranked correctly. We may need 100,000,000 words to estimate these parameters. This agrees with the empirical results, in which as more training data is used, the OOV curve improves rapidly up to 50,000,000 words and then more slowly beyond that point.

11.6.2. *N*-gram Pruning

When high order *n*-gram models are used, the model sizes typically become too large for practical applications. It is necessary to prune parameters from *n*-gram models such that the relative entropy between the original and the pruned model is minimized. You can chose *n*-grams so as to maximize performance (i.e., minimize perplexity) while minimizing the model size [39, 59, 64].

The criterion to prune *n*-grams can be based on some well-understood information-theoretic measure of language model quality. For example, the pruning method by Stolcke [64] removes some *n*-gram estimates while minimizing the performance loss. After pruning, the retained explicit *n*-gram probabilities are unchanged, but backoff weights are recomputed. Stolcke pruning uses the criterion that minimizes the distance between the distribution embodied by the original model and that of the pruned model based on the *Kullback-Leibler distance* defined in Eq. (3.175). Since it is infeasible to maximize over all possible subsets of *n*-grams, Stolcke prunning assumes that the *n*-grams affect the relative entropy roughly independently, and compute the distance due to each individual *n*-gram. The *n*-grams are thus ranked by their effect on the model entropy, and those that increase relative entropy the least are pruned accordingly. The main approximation is that we do not consider possible interactions between selected n-grams, and prune based solely on relative entropy due to removing a single *n*-gram. This avoids searching the exponential space of *n*-gram subsets.

To compute the relative entropy, $KL(p \parallel p')$, between the original and pruned *n*-gram models p and p', there is no need to sum over the vocabulary. By plugging in the terms for the backed-off estimates, the sum can be factored as shown in Eq. (11.55) for a more efficient computation.

$$\begin{aligned} KL(p \parallel p') = -P(h)\{ & P(w \mid h)[\log P(w \mid h') + \log \alpha'(h) - \log P(w \mid h)] \\ & +[\log \alpha'(h) - \log \alpha(h)](1 - \sum_{w_i \in \neg Backoff\,(w_i h)} P(w_i \mid h))\} \end{aligned} \tag{11.55}$$

where the sum in $\sum_{w_i \in \neg Backoff\,(w_i h)} P(w_i \mid h)$ is over all non-backoff estimates. To compute the revised backoff weights $\alpha'(h)$, you can simply drop the term for the pruned *n*-gram from the summation (Backoff weight computation is illustrated in Algorithm 11.1).

In practice, pruning is highly effective. Stolcke reported that the trigram model can be compressed by more than 25% without degrading recognition performance. Comparing the pruned 4-gram model to the unpruned trigram model, it is better to use pruned 4-grams than to use a much larger number of trigrams.

11.6.3. CFG vs *n*-gram Models

This chapter has discussed two major language models. While CFGs remain one of the most important formalisms for interpreting natural language, word *n*-gram models are surprisingly powerful for domain-independent applications. These two formalisms can be unified for both speech recognition and spoken language understanding. To improve portability of the domain-independent *n*-gram, it is possible to incorporate domain-specific CFGs into the domain-independent *n*-gram that can improve generalizability of the CFG and specificity of the *n*-gram.

The CFG is not only powerful enough to describe most of the structure in spoken language, but also restrictive enough to have efficient parsers. $P(\mathbf{W})$ is regarded as 1 or 0 depending upon whether the word sequence is accepted or rejected by the grammar. The problem is that the grammar is *almost always incomplete*. A CFG-based system is good only when you know what sentences to speak, which diminishes the system's value and usability of the system. The advantage of CFG's structured analysis is, thus, nullified by the poor coverage in most real applications. On the other hand, the *n*-gram model is trained with a large amount of data, the *n*-word dependency can often accommodate both syntactic and semantic structure seamlessly. The prerequisite of this approach is that we have enough training data. The problem for *n*-gram models is that we need a lot of data and the model may not be specific enough.

It is possible to take advantage of both rule-based CFGs and data-driven *n*-grams. Let's consider the following training sentences:

```
Meeting at three with Zhou Li.
Meeting at four PM with Derek.
```

If we use a word trigram, we estimate *P(Zhou|three with)* and *P(Derek|PM with)*, etc. There is no way we can capture needed long-span semantic information in the training data. A unified model has a set of CFGs that can capture the semantic structure of the domain. For the example listed here, we have a CFG for {name} and {time}, respectively. We can use the CFG to parse the training data to spot all the potential semantic structures in the training data. The training sentences now look like:

```
Meeting {at three:TIME} with {Zhou Li:NAME}
Meeting {at four PM:TIME} with {Derek: NAME}
```

With analyzed training data, we can estimate our *n*-gram probabilities as usual. We have probabilities, such as P({name}|{time} with), instead of *P(Zhou|three with)*, which is more meaningful and accurate. Inside each CFG we also derive P("*Zhou Li*"|{name}) and P("*four PM*"|{time}) from the existing *n*-gram (*n*-gram probability inheritance) so that they are normalized. If we add a new name to the existing {name} CFG, we use the existing *n*-gram probabilities to renormalize our CFGs for the new name. The new approach can be regarded as a standard *n*-gram in which the vocabulary consists of words and structured classes, as discussed in Section 11.4.3. The structured class can be very simple, such as {date}, {time}, and {name}, or can be very complicated, such as a CFG that contains deep structured information. The probability of a word or class depends on the previous words or CFG classes.

It is possible to inherit probability from a word *n*-gram LM. Let's take word trigram as our example here. An input utterance $\mathbf{W} = w_1 w_2 ... w_n$ can be segmented into a sequence $\mathbf{T} = t_1 t_2 ... t_m$, where each t_i is either a word in **W** or a CFG non-terminal that covers a sequence of words $\bar{u}_{t_i}$ in **W**. The likelihood of **W** under the segmentation **T** is, therefore,

$$P(\mathbf{W}, \mathbf{T}) = \prod_i P(t_i \mid t_{i\text{-}1}, t_{i\text{-}2}) \prod_i P(\bar{u}_{t_i} \mid t_i) \tag{11.56}$$

$P(\bar{u}_{t_i} \mid t_i)$, the likelihood of generating a word sequence $\bar{u}_{t_i} = [u_{t_i 1} u_{t_i 2} ... u_{t_i k}]$ from the CFG non-terminal t_i, can be inherited from the domain-independent word trigram. We can essentially use the CFG constraint to condition the domain-independent trigram into a domain-specific trigram. Such a unified language model can dramatically improve cross-domain performance using domain-specific CFGs [66].

In summary, the CFG is widely used to specify the permissible word sequences in natural language processing when training corpora are unavailable. It is suitable for dealing with structured command and control applications in which the vocabulary is small and the semantics of the task is well defined. The CFG either accepts the input sentence or rejects it. There is a serious coverage problem associated with CFGs. In other words, the accuracy for the CFG can be extremely high when the test data are covered by the grammar. Unfortunately, unless the task is narrow and well-defined, most users speak sentences that may not be accepted by the CFG, leading to word recognition errors.

Statistical language models such as trigrams assign an estimated probability to any word that can follow a given word history without parsing the structure of the history. Such an approach contains some limited syntactic and semantic information, but these probabilities are typically trained from a large corpus. Speech recognition errors are much more likely to occur within trigrams and (especially) bigrams that have not been observed in the training data. In these cases, the language model typically relies on lower-order statistics. Thus, increased *n*-gram coverage translates directly into improved recognition accuracy, but usually at the cost of increased memory requirements.

It is interesting to compute the true entropy of the language so that we understand what a solid lower bound is for the language model. For English, Shannon [60] used human subjects to guess letters by looking at how many guesses it takes people to derive the correct one based on the history. We can thus estimate the probability of the letters and hence the entropy of the sequence. Shannon computed the per-letter entropy of English with an entropy of 1.3 bits for 26 letters plus space. This may be an underestimate, since it is based on a single text. Since the average length of English written words (including space) is about 5.5 letters, the Shannon estimate of 1.3 bits per letter corresponds to a per-word perplexity of 142 for general English.

Table 11.2 N-gram perplexity and its corresponding speaker-independent speech recognition word error rate.

Models	Perplexity	Word Error Rate
Unigram Katz	1196.45	14.85%
Unigram Kneser-Ney	1199.59	14.86%
Bigram Katz	176.31	11.38%
Bigram Kneser-Ney	176.11	11.34%
Trigram Katz	95.19	9.69%
Trigram Kneser-Ney	91.47	9.60%

Table 11.2 summarizes the performance of several different *n*-gram models on a 60,000-word continuous speech dictation application. The experiments used about 260 million words from a newspaper such as The Wall Street Journal. The speech recognizer is based on Whisper described in Chapter 9. As you can see from the table, when the amount of training data is sufficient, both Katz and Kneser-Ney smoothing offer comparable recogni-

tion performance, although Kneser-Ney smoothing offers a modest improvement when the amount of training data is limited.

In comparison to Shannon's estimate of general English word perplexity, the trigram language for The Wall Street Journal is lower (91.4 vs. 142). This is because the text is mostly business oriented with a fairly homogeneous style and word usage pattern. For example, if we use the trigram language for data from a new domain that is related to personal information management, the test-set word perplexity can increase to 378 [66].

11.7. HISTORICAL PERSPECTIVE AND FURTHER READING

There is a large and active area of research in both speech and linguistics. These two distinctive communities worked on the problem with very different paths, leading to the stochastic language models and the formal language theory. The linguistics community has developed tools for tasks like parsing sentences, assigning semantic relations to the parts of a sentence, and so on. Most of these parser algorithms have the same characteristics, that is, they tabulate each sub-derivation and reuse it in building any derivation that shares that sub-derivation with appropriate grammars [22, 65, 67]. They have polynomial complexity with respect to sentence length because of *dynamic programming* principles to search for optimal derivations with respect to appropriate evaluation functions on derivations. There are three well-known dynamic programming parsers with a worst-case behavior of $O(n^3)$, where n is the number of words in the sentence: the Cocke-Younger-Kasami (CYK) algorithm (a bottom-up parser, proposed by J. Cocke, D. Younger, and T. Kasami) [32, 67], the Graham-Harrison-Ruzzo algorithm (bottom-up) [30], and the Earley algorithm (top-down) [21].

On the other hand, the speech community has developed tools to predict the next word on the basis of what has been said, in order to improve speech recognition accuracy [35]. Neither approach has been completely successful. The formal grammar and the related parsing algorithms are too brittle for comfort and require a lot of human retooling to port from one domain to another. The lack of structure and deep understanding has taken its toll on statistical technology's ability to choose the right words to guide speech recognition.

In addition to those discussed in this chapter, many alternative formal techniques are available. Augmented context-free grammars are used for natural language to capture grammatical natural languages such as agreement and subcategorization. Examples include generalized phrase structure grammars and head-driven phrase structure grammars [26, 53]. You can further generalize the augmented context-free grammar to the extent that the requirement of *context free* becomes unnecessary. The entire grammar, known as the *unification grammar,* can be specified as a set of constraints between feature structures [62]. Most of these grammars have only limited success when applied to spoken language systems. In fact, no practical domain-independent parser of unrestricted text has been developed for spoken language systems, partly because disambiguation requires the specification of detailed semantic information. Analysis of the Susanne Corpus with a crude parser suggests that over 80% of sentences are structurally ambiguous. More recently, large *treebanks* of parsed texts have given impetus to statistical approaches to parsing. Probabilities can be estimated from treebanks or plain text [6, 8, 24, 61] to efficiently rank analyses produced by modified chart parsing algorithms. These systems have yielded results of around 75% accuracy in assigning analyses to (unseen) test sentences from the same source as the unambiguous training material. Attempts have also been made to use statistical induction to *learn* the correct grammar for a given corpus of data [7, 43, 51, 58]. Nevertheless, these techniques are limited to simple grammars with category sets of a dozen or so non-terminals, or to training on manually parsed data. Furthermore, even when parameters of the grammar and control mechanism can be learned automatically from training corpora, the required corpora do not exist or are too small for proper training. In practice, we can devise grammars that specify directly how relationships relevant to the task may be expressed. For instance, one may use a phrase-structure grammar in which nonterminals stand for task concepts and relationships and rules specify possible expressions of those concepts and relationships. Such *semantic grammars* have been widely used for spoken language applications as discussed in Chapter 17.

It is worthwhile to point out that many natural language parsing algorithms are *NP-complete*, a term for a class of problems that are suspected to be particularly difficult to process. For example, maintaining lexical and agreement features over a potentially infinite-length sentence causes the unification-based formalisms to be NP-complete [3].

Since the predictive power of a general-purpose grammar is insufficient for reasonable performance, *n*-gram language models continue to be widely used. A complete proof of Good-Turing smoothing was presented by Church *et al*. [17]. Chen and Goodman [13] provide a detailed study on different *n*-gram smoothing algorithms. Jelinek's Eurospeech tutorial paper [35] provides an interesting historical perspective on the community's efforts to improve trigrams. Mosia and Giachin's paper [48] has detailed experimental results on class-based language models. Class-based model may be based on parts of speech or morphology [10, 16, 23, 47, 63]. More detailed discussion of the maximum entropy language model can be found in [5, 36, 42, 44, 52, 55, 56].

One interesting research area is to combine both n-grams and the structure that is present in language. A concerted research effort to explore structure-based language model may be the key for significant progress to occur in language modeling. This can be done as annotated data becomes available. Nasr *et al*. [50] have considered a new unified language model composed of several local models and a general model linking the local models together. The local model used in their system is based on the stochastic FSA, which is estimated from the training corpora. Other efforts to incorporate structured information are described in [12, 25, 27, 49, 66].

You can find tools to build *n*-gram language models at the CMU open source Web site[7] that contains the release of CMU's language modeling toolkit and documentation of SRI's language modeling toolkit.[8]

[7] http://www.speech.cs.cmu.edu/sphinx/
[8] http://www.speech.sri.com/projects/srilm/download.html

REFERENCES

[1] Aho, A.V. and J.D. Ullman, *The Theory of Parsing, Translation and Compiling*, 1972, Englewood Cliffs, NJ, Prentice-Hall.

[2] Bahl, L.R., *et al.*, "A Tree-Based Statistical Language Model for Natural Language Speech Recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 1989, **37**(7), pp. 1001-1008.

[3] Barton, G., R. Berwick, and E. Ristad, *Computational Complexity and Natural Language*, 1987, Cambridge, MA, MIT Press.

[4] Bellegarda, J., "A Latent Semantic Analysis Framework for Large-Span Language Modeling," *Eurospeech*, 1997, Rhodes, Greece pp. 1451-1454.

[5] Berger, A., S. DellaPietra, and V. DellaPietra, "A Maximum Entropy Approach to Natural Language Processing," *Computational Linguistics*, 1996, **22**(1), pp. 39-71.

[6] Black, E., *et al.*, "Towards History-based Grammars: Using Richer Models for Probabilistic Parsing," *Proc. of the Annual Meeting of the Association for Computational Linguistics*, 1993, Columbus, Ohio, USA pp. 31--37.

[7] Briscoe, E.J., ed. *Prospects for Practical Parsing: Robust Statistical Techniques*, in Corpus-based Research into Language: A Feschrift for Jan Aarts, ed. P.d. Haan and N. Oostdijk, 1994, Amsterdam. 67-95, Rodopi.

[8] Briscoe, E.J. and J. Carroll, "Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-based Grammars," *Computational Linguistics*, 1993, **19**, pp. 25-59.

[9] Brown, P.F., *et al.*, "Class-Based N-Gram Models of Natural Language," *Computational Linguistics*, 1992(4), pp. 467-479.

[10] Cerf-Danon, H. and M. El-Bèze, "Three Different Probabilistic Language Models: Comparison and Combination," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada pp. 297-300.

[11] Charniak, E., "Statistical Parsing with a Context-Free Grammar and Word Statistics," *AAAI-97*, 1997, Menlo Park pp. 598-603.

[12] Chelba, C., A. Corazza, and F. Jelinek, "A Context Free Headword Language Model" in *Proc. of IEEE Automatic Speech Recognition Workshop"* 1995, Snowbird, Utah, pp. 89--90.

[13] Chen, S. and J. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," *Proc. of Annual Meeting of the ACL*, 1996, Santa Cruz, CA.

[14] Chomsky, N., *Syntactic Structures*, 1957, The Hague: Mouton.

[15] Chomsky, N., *Aspects of the Theory of Syntax*, 1965, Cambridge, MIT Press.

[16] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of 2nd Conf. on Applied Natural Language Processing.*, 1988, Austin, Texas pp. 136-143.

[17] Church, K.W. and W.A. Gale, "A Comparison of the Enhanced Good-Turing and Deleted Estimation Methods for Estimating Probabilities of English Bigrams," *Computer Speech and Language*, 1991, pp. 19-54.

[18] Cole, R., *et al.*, *Survey of the State of the Art in Human Language Technology*, eds. http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html, 1996, Cambridge University Press.

[19] Collins, M., "A New Statistical Parser Based on Bigram Lexical Dependencies," *ACL-96*, 1996, pp. 184-191.

[20] Darroch, J.N. and D. Ratcliff, "Generalized Iterative Scaling for Log-Linear Models," *The Annals of Mathematical Statistics*, 1972, **43**(5), pp. 1470-1480.

[21] Earley, J., *An Efficient Context-Free Parsing Algorithm*, PhD Thesis 1968, Carnegie Mellon University, Pittsburgh.

[22] Earley, J., "An Efficient Context-Free Parsing Algorithm," *Communications of the ACM*, 1970, **6**(8), pp. 451-455.

[23] El-Bèze, M. and A.-M. Derouault, "A Morphological Model for Large Vocabulary Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, NM pp. 577-580.

[24] Fujisaki, T., *et al.*, "A probabilistic parsing method for sentence disambiguation," *Proc. of the Int. Workshop on Parsing Technologies*, 1989, Pittsburgh.

[25] Galescu, L., E.K. Ringger, and a.F. Allen, "Rapid Language Model Development for New Task Domains," *Proc. of the ELRA First Int. Conf. on Language Resources and Evaluation (LREC)*, 1998, Granada, Spain.

[26] Gazdar, G., *et al.*, *Generalized Phrase Structure Grammars*, 1985, Cambridge, MA, Harvard University Press.

[27] Gillett, J. and W. Ward, "A Language Model Combining Trigrams and Stochastic Context-Free Grammars," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[28] Good, I.J., "The Population Frequencies of Species and the Estimation of Population Parameters," *Biometrika*, 1953, pp. 237-264.

[29] Goodman, J., *Parsing Inside-Out*, PhD Thesis in *Computer Science* 1998, Harvard University, Cambridge.

[30] Graham, S.L., M.A. Harrison, and W. L.Ruzzo, "An Improved Context-Free Recognizer," *ACM Trans. on Programming Languages and Systems*, 1980, **2**(3), pp. 415-462.

[31] Hindle, D. and M. Rooth, "Structural Ambiguity and Lexical Relations," *DARPA Speech and Natural Language Workshop*, 1990, Hidden Valley, PA, Morgan Kaufmann.

[32] Hopcroft, J.E. and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 1979, Reading, MA, Addision Wesley.

[33] Iyer, R., M. Ostendorf, and J.R. Rohlicek, "Language Modeling with Sentence-Level Mixtures," *Proc. of the ARPA Human Language Technology Workshop*, 1994, Plainsboro, NJ pp. 82-86.

[34] Jardino, M., "Multilingual Stochastic N-gram Class Language Models," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, GA pp. 161-163.

[35] Jelinek, F., "Up From Trigrams! The Struggle for Improved Language Models" in *Proc. of the European Conf. on Speech Communication and Technology* 1991, Genoa, Italy, pp. 1037-1040.
[36] Jelinek, F., *Statistical Methods for Speech Recognition*, 1998, Cambridge, MA, MIT Press.
[37] Jelinek, F., *et al.*, "A dynamic language model for speech recognition" in *Proc. of the DARPA Speech and Natural Language Workshop* 1991, Asilomar, CA.
[38] Katz, S.M., "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer," *IEEE Trans. Acoustics, Speech and Signal Processing*, 1987(3), pp. 400-401.
[39] Kneser, R., "Statistical Language Modeling using a Variable Context" in *Proc. of the Int. Conf. on Spoken Language Processing* 1996, Philadelphia, PA, pp. 494.
[40] Kneser, R. and H. Ney, "Improved Backing-off for M-gram Language Modeling" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1995, Detroit, MI, pp. 181-184.
[41] Kuhn, R. and R.D. Mori, "A Cache-Based Natural Language Model for Speech Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1990(6), pp. 570-582.
[42] Lafferty, J.D. and B. Suhm, "Cluster Expansions and Iterative Scaling for Maximum Entropy Language Models" in *Maximum Entropy and Bayesian Methods*, K. Hanson and R. Silver, eds. 1995, Kluwer Academic Publishers.
[43] Lari, K. and S.J. Young, "Applications of Stochastic Context-free Grammars Using the Inside-Outside Algorithm," *Computer Speech and Language*, 1991, **5**(3), pp. 237-257.
[44] Lau, R., R. Rosenfeld, and S. Roukos, "Trigger-Based Language Models: A Maximum Entropy Approach," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN pp. 108-113.
[45] Magerman, D.M. and M.P. Marcus, "Pearl: A Probabilistic Chart Parser," *Proc. of the Fourth DARPA Speech and Natural Language Workshop*, 1991, Pacific Grove, California.
[46] Mahajan, M., D. Beeferman, and X.D. Huang, "Improved Topic-Dependent Language Modeling Using Information Retrieval Techniques," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1999, Phoenix, AZ pp. 541-544.
[47] Maltese, G. and F. Mancini, "An Automatic Technique to Include Grammatical and Morphological Information in a Trigram-based Statistical Language Model," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, CA pp. 157-160.
[48] Moisa, L. and E. Giachin, "Automatic Clustering of Words for Probabilistic Language Models" in *Proc. of the European Conf. on Speech Communication and Technology* 1995, Madrid, Spain, pp. 1249-1252.
[49] Moore, R., *et al.*, "Combining Linguistic and Statistical Knowledge Sources in Natural-Language Processing for ATIS," *Proc. of the ARPA Spoken Language Sys-*

tems Technology Workshop, 1995, Austin, Texas, Morgan Kaufmann, Los Altos, CA.

[50] Nasr, A., *et al.*, "A Language Model Combining N-grams and Stochastic Finitie State Automata," *Proc. of the Eurospeech*, 1999, Budapest, Hungary pp. 2175-2178.

[51] Pereira, F.C.N. and Y. Schabes, "Inside-Outside Reestimation from Partially Bracketed Corpora," *Proc. of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992 pp. 128-135.

[52] Pietra, S.A.D., *et al.*, "Adaptive Language Model Estimation using Minimum Discrimination Estimation," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, CA pp. 633-636.

[53] Pollard, C. and I.A. Sag, *Head-Driven Phrase Structure Grammar*, 1994, Chicago, University of Chicago Press.

[54] Pullum, G. and G. Gazdar, "Natural Languages and Context-Free Languages," *Linguistics and Philosophy*, 1982, **4**, pp. 471-504.

[55] Ratnaparkhi, A., S. Roukos, and R.T. Ward, "A Maximum Entropy Model for Parsing," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan pp. 803--806.

[56] Rosenfeld, R., *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*, Ph.D. Thesis in *School of Computer Science* 1994, Carnegie Mellon University, Pittsburgh, PA.

[57] Salton, G. and M.J. McGill, *Introduction to Modern Information Retrieval*, 1983, New York, McGraw-Hill.

[58] Schabes, Y., M. Roth, and R. Osborne, "Parsing the Wall Street Journal with the Inside-Outside Algorithm," *Proc. of the Sixth Conf. of the European Chapter of the Association for Computational Linguistics*, 1993 pp. 341-347.

[59] Seymore, K. and R. Rosenfeld, "Scalable Backoff Language Models," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, PA pp. 232.

[60] Shannon, C.E., "Prediction and Entropy of Printed English," *Bell System Technical Journal*, 1951, pp. 50-62.

[61] Sharman, R., F. Jelinek, and R.L. Mercer, "Generating a Grammar for Statistical Training," *Proc. of the Third DARPA Speech and Natural Language Workshop*, 1990, Hidden Valley, Pennsylvania pp. 267-274.

[62] Shieber, S.M., *An Introduction to Unification-Based Approaches to Grammars*, 1986, Cambridge, UK, CSLI Publication, Leland Stanford Junior University.

[63] Steinbiss, V., *et al.*, "A 10,000-word Continuous Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, NM pp. 57-60.

[64] Stolcke, A., "Entropy-based Pruning of Backoff Language Models," *DARPA Broadcast News Transcription and Understanding Workshop*, 1998, Lansdowne, VA.

[65] Tomita, M., "An Efficient Augmented-Context-Free Parsing Algorithm," *Computational Linguistics*, 1987, **13**(1-2), pp. 31-46.

[66] Wang, Y., M. Mahajan, and X. Huang, "A Unified Context-Free Grammar and N-Gram Model for Spoken Language Processing," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey pp. 1639-1642.

[67] Younger, D.H., "Recognition and Parsing of Context-Free Languages in Time n^3," *Information and Control*, 1967, **10**, pp. 189-208.