SMOOTHING A PROBABILISTIC LEXICON VIA SYNTACTIC TRANSFORMATIONS

Jason Michael Eisner

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2001

Professor Mitch Marcus
Supervisor of Dissertation

Professor Val Tannen
Graduate Group Chair

Abstract

SMOOTHING A PROBABILISTIC LEXICON VIA SYNTACTIC TRANSFORMATIONS

Jason Michael Eisner

Supervisor: Professor Mitch Marcus

Probabilistic parsing requires a lexicon that specifies each word's syntactic preferences in terms of probabilities. To estimate these probabilities for words that were poorly observed during training, this thesis assumes the existence of arbitrarily powerful transformations (also known to linguists as lexical redundancy rules or metarules) that can add, delete, retype or reorder the argument and adjunct positions specified by a lexical entry. In a given language, some transformations apply frequently and others rarely. We describe how to estimate the rates of the transformations from a sample of lexical entries. More deeply, we learn which properties of a transformation increase or decrease its rate in the language. As a result, we can smooth the probabilities of lexical entries. Given enough direct evidence about a lexical entry's probability, our Bayesian approach trusts the evidence; but when less evidence or no evidence is available, it relies more on the transformations' rates to guess how often the entry will be derived from related entries.

Abstractly, the proposed "transformation models" are probability distributions that arise from graph random walks with a log-linear parameterization. A domain expert constructs the parameterized graph, and a vertex is likely according to whether random walks tend to halt at it. Transformation models are suited to any domain where "related" events (as defined by the graph) may have positively covarying probabilities. Such models admit a natural prior that favors simple regular relationships over stipulative exceptions. The model parameters can be locally optimized by gradient-based methods or by Expectation-Maximization. Exact algorithms (matrix inversion) and approximate ones (relaxation) are provided, with optimizations. Variations on the idea are also discussed.

We compare the new technique empirically to previous techniques from the probabilistic parsing literature, using comparable features, and obtain a 20% perplexity reduction (similar to doubling the amount of training data). Some of this reduction is shown to stem from the transformation model's ability to match observed probabilities, and some from its ability to generalize. Model averaging yields a final 24% perplexity reduction.

Contents

List of Tables

List of Figures

Chapter 1

Overview: An Executive Summary

1.1 Context of the Work

Humans are experts in the languages that they speak. So too must be machines that use language. Yet computational linguists are increasingly loath to build language-specific "expert systems" by hand. Such systems tend to be both expensive and fragile.

Instead, the dominant strategy of the past decade has been to build machines that learn statistically from raw or annotated language data. The computational linguist's job is to design an appropriately expressive statistical model whose parameters can be estimated, using appropriate algorithms, from a reasonable amount of data. These algorithms then learn the nuances of a particular language by tuning the parameters of the model to match known data from the language.

To increase the linguistic expertise of such machines, one must design statistical models that are sensitive to additional linguistic phenomena. But parameter estimation is harder for more complex models—so in practice it is a challenge to add linguistic sophistication that helps more than it hurts.

In the domain of syntax, the community has been adding sophistication carefully, one step at a time. Statistical models have been extended to capture a succession of linguistic phenomena, usually well enough in practice to exploit them for performance gains:

1. collocations (n-gram Markov models)

2. word categories (hidden Markov models)
3. trees (probabilistic context-free grammars)
4. subcategorization (lexicalized context-free grammars)
5. selectional preferences (various bilexical grammars)
6. long-distance movement (statistical gap passing)
7. arbitrary tree properties (stochastic LTAG, DOP, log-linear LFG, etc.)

These are mainly models of what Chomsky calls the "surface structure" of syntax trees.

This thesis builds on that progress by offering a working statistical treatment of Chomsky's "deep structure"—the transformational relationships *among* syntactic constructions that are related in meaning. We will see that a statistical model can detect such relationships in an existing sample of syntactic constructions, and use them to better predict the probabilities of constructions in the sample as well as new constructions that it has never seen before. It is in this sense that the model does "transformational smoothing" of probabilities.

In short, the model finds and applies language-specific generalizations about deep syntax. The approach is declarative. First we quantify a notion of a good grammar. Then we search in the continuous space of all possible grammars, trying to maximize the Bayesian or MDL product

$$\Pr(\text{training data} \mid \text{grammar}) \cdot \Pr(\text{grammar}) \tag{1.1}$$

The first factor emphasizes description, the second generalization. The first factor is high if the grammar assigns high probability to the constructions observed so far. The second, trickier factor is the *a priori* probability of the grammar itself; we will define it to be high if the grammar has strong, consistent internal structure with few stipulative exceptions.

The rest of this chapter is an overview of the scope and content of the work. It concludes with a detailed guide to the rest of the thesis.

1.2 Motivations

Let us begin by motivating the work from the interests of four different fields: engineering, linguistics, statistics, and child language acquisition.

1.2.1 The Engineering Problem: Learning More Syntax from Less Data

1.2.1.1 Probabilistic Parsing

Accurate parsing of natural language—discovering the recursive structure of an arbitrary sentence—is a major roadblock for systems that attempt to understand or translate text. A great many word meanings and grammatical constructions are *possible* (if rare) in the language. The typical sentence therefore has combinatorially many parses. The task is to output the *right* parse.

The trick is to search not for *possible* parses but for *probable* ones. A dynamic-programming parser builds a parse tree by composing context-free rules, or other small "chunks" of tree, according to some theory of syntax. One can define the tree's probability in terms of the probabilities of its component chunks (§5.3.3). The dynamic program can then find the most probable tree that parses a given input. A parser with access to probabilities can also run faster, by heuristically ignoring low-probability regions of the search space.

Probably the best-known recent English parsers of this sort are (Collins, 1997) and (Charniak, 2000), which are interesting, accurate and robust. However, this framework boasts many other practical and theoretical papers worthy of mention, and the techniques have been ported beyond English, e.g., (Collins et al., 1999).

1.2.1.2 The Need to Generalize

The trouble is that the chunk probabilities are hard to estimate from even a large sample of (expensively annotated) data. Many chunks that are needed to parse test data correctly are never observed at all in training data. In our data, derived from *The Wall Street Journal* via the Penn Treebank (§6.2), a remarkable 49% of sentences in the development set required for their correct parse at least one subcategorization frame (broadly construed to include slots for all arguments and adjuncts) that never appeared with *any* headword in training data (§6.3). The parser cannot recover the correct parse if it assigns these novel frames a zero or negligible probability.

Hence, the parser must be able to generalize from chunks it has seen during training to new chunks that it has never seen. The list of chunks that actually appears in training data tends to be large, unorganized, redundant, and—most harmfully—incomplete. To parse, we need to reconstruct the *true* list of chunks in the language, and their true probabilities, as well as possible. This means looking at internal chunk structure.

Parsers in the literature usually deal with this problem by building the "atomic" chunks (or equivalently the parse trees) from smaller "subatomic" units that are more likely to recur in training data: binary-branching rules or single dependencies rather than complete subcategorization frames. However, even weak versions of this atom-splitting lead to overly strong independence assumptions (§6.6.3). Other generative models for chunks have occasionally been considered (Collins, 1997, model 2), although these were not the best performers in our experiments either (§6.7.1).

This thesis explores a more sophisticated and apparently better-performing approach to generalizing from old (observed) chunks to new ones. The approach is inspired by both linguistic tradition (§1.2.2) and an analysis of the data (§2.4.2.1).

1.2.1.3 An Example

Concretely, consider the table in Table 1.1. It shows 6 words with all the subcategorization frames they headed in our training data.[1] (Since the words are uninflected verbs, they only appear in infinitive or present-tense form in our unmorphologized data.) The symbol "——" stands for the headword in the subcategorization frame. For example, the second row says the lexicalized context-free rule `S` $\rightarrow$ `To encourage NP PP` was used once in the hand-constructed training parses, while `S` $\rightarrow$ `To fund NP PP` was used twice, and so forth.

In order to parse, we need to replace the counts in this table with probabilities Pr(frame | headword), as shown in Table 1.3, such that

[1]For reasons discussed in §2.4.1, this thesis interprets "subcategorization frame" broadly to include all dependents—including adjuncts (such as some PP's) and subjects, which have traditionally been excluded from subcategorization frames. To avoid confusion we will simply use the term "frame" in future.

	encourage	question	fund	merge	repay	remove
S $\rightarrow$ To —— NP	1	1	5	1	3	2
S $\rightarrow$ To —— NP PP	1	1	2	2	1	1
S $\rightarrow$ To AdvP —— NP						1
S $\rightarrow$ To AdvP —— NP PP						1
S $\rightarrow$ NP —— NP .		2				
S $\rightarrow$ NP —— NP PP .	1					
S $\rightarrow$ NP Md —— NP	1					
S $\rightarrow$ NP Md —— NP PP-TMP					1	
S $\rightarrow$ NP Md —— PP PP						1
S $\rightarrow$ To —— PP				1		
S $\rightarrow$ To —— S	1					
S $\rightarrow$ NP —— SBAR .		2				

Table 1.1: Counts of all the S $\rightarrow \cdots$ frames that appear with six particular words in training data. The words were not specially chosen, except that they were all required to appear with S $\rightarrow$ To —— NP PP (so that they would be related) and to appear 4–7 times (so that this example table would be small but not trivial).

$\Pr(RHS \mid headword, LHS)$	encourage	question	fund	merge	repay	remove
S $\rightarrow$ To —— NP	0.200	0.167	0.714	0.250	0.600	0.333
S $\rightarrow$ To —— NP PP	0.200	0.167	0.286	0.500	0.200	0.167
S $\rightarrow$ To AdvP —— NP	0	0	0	0	0	0.167
S $\rightarrow$ To AdvP —— NP PP	0	0	0	0	0	0.167
S $\rightarrow$ NP —— NP .	0	0.333	0	0	0	0
S $\rightarrow$ NP —— NP PP .	0.200	0	0	0	0	0
S $\rightarrow$ NP Md —— NP	0.200	0	0	0	0	0
S $\rightarrow$ NP Md —— NP PP-TMP	0	0	0	0	0.200	0
S $\rightarrow$ NP Md —— PP PP	0	0	0	0	0	0.167
S $\rightarrow$ To —— PP	0	0	0	0.250	0	0
S $\rightarrow$ To —— S	0.200	0	0	0	0	0
S $\rightarrow$ NP —— SBAR .	0	0.333	0	0	0	0
(other)	0	0	0	0	0	0

Table 1.2: The probabilistic lexicon most likely to generate the counts in Table 1.1. While it is trivial to compute, unfortunately it is an improbable lexicon *a priori*. It would be a poor engineering choice since it assigns 0 probability to many plausible frames.

Pr($RHS \mid headword, LHS$)	encourage	question	fund	merge	repay	remove
`S` $\rightarrow$ `To` — `NP`	0.088	0.043	0.398	0.127	0.171	0.131
`S` $\rightarrow$ `To` — `NP PP`	0.030	0.016	0.101	0.114	0.032	0.048
`S` $\rightarrow$ `To AdvP` — `NP`	0.00017	0.000064	0.00019	0.00010	0.00020	0.033
`S` $\rightarrow$ `To AdvP` — `NP PP`	0.000057	0.000023	0.000049	0.000092	0.000038	0.012
`S` $\rightarrow$ `NP` — `NP .`	0.013	0.104	0.0021	0.0053	0.0063	0.0034
`S` $\rightarrow$ `NP` — `NP PP .`	0.016	0.0068	0.00095	0.0019	0.0021	0.0010
`S` $\rightarrow$ `NP Md` — `NP`	0.023	0.0020	0.0055	0.0030	0.035	0.034
`S` $\rightarrow$ `NP Md` — `NP PP-TMP`	0.00056	0.000061	0.000031	0.000079	0.0047	0.00030
`S` $\rightarrow$ `NP Md` — `PP PP`	0.00014	0.000032	0.000024	0.000089	0.00029	0.0018
`S` $\rightarrow$ `To` — `PP`	0.016	0.015	0.052	0.112	0.031	0.051
`S` $\rightarrow$ `To` — `S`	0.034	0.0061	0.010	0.016	0.012	0.0053
`S` $\rightarrow$ `NP` — `SBAR .`	0.018	0.111	0.0029	0.0076	0.0089	0.0048
(other)	0.762	0.696	0.428	0.613	0.695	0.675

Table 1.3: Part of the probabilistic lexicon induced from training data by the best model from previous literature, a smoothed bigram model (§6.6.1.2). There exist many more rows (in fact, infinitely many) and columns. Each column sums to 1 (if we consider only the `S` $\rightarrow \cdots$ rows).

Pr($RHS \mid headword, LHS$)	encourage	question	fund	merge	repay	remove
`S` $\rightarrow$ `To` — `NP`	0.142	0.117	0.397	0.210	0.329	0.222
`S` $\rightarrow$ `To` — `NP PP`	0.077	0.064	0.120	0.181	0.088	0.080
`S` $\rightarrow$ `To AdvP` — `NP`	0.00055	0.00047	0.0011	0.00082	0.00091	0.079
`S` $\rightarrow$ `To AdvP` — `NP PP`	0.00018	0.00015	0.00033	0.00037	0.00026	0.050
`S` $\rightarrow$ `NP` — `NP .`	0.022	0.161	0.0078	0.0075	0.0079	0.0075
`S` $\rightarrow$ `NP` — `NP PP .`	0.079	0.0085	0.0026	0.0027	0.0026	0.0026
`S` $\rightarrow$ `NP Md` — `NP`	0.090	0.0021	0.0024	0.0020	0.024	0.0026
`S` $\rightarrow$ `NP Md` — `NP PP-TMP`	0.0018	0.00016	0.00017	0.00016	0.069	0.00019
`S` $\rightarrow$ `NP Md` — `PP PP`	0.00010	0.000027	0.000027	0.000038	0.000078	0.059
`S` $\rightarrow$ `To` — `PP`	0.0092	0.0065	0.012	0.126	0.010	0.0091
`S` $\rightarrow$ `To` — `S`	0.098	0.0016	0.0043	0.0039	0.0036	0.0027
`S` $\rightarrow$ `NP` — `SBAR .`	0.0034	0.190	0.0032	0.0032	0.0032	0.0032
(other)	0.478	0.449	0.449	0.461	0.461	0.482

Table 1.4: Part of the probabilistic lexicon induced from training data by transformational smoothing.

1. Each column of the table sums to 1.
2. The observed counts constitute a likely sample from the probabilities.
3. The pattern of probabilities is *a priori* plausible.

The result is called a conditionalized "probabilistic lexicon."

The maximum-likelihood estimate (Table 1.2) satisfies the first two properties but not the third; it is too stipulative. A smoothing method such as our transformational smoothing (Table 1.4) is designed to "smear" these estimates down the column, generalizing from one row to another. This lets it place non-zero probabilities on rules that appear in the correct parses of test data, including novel rules.

We would like a smoothing method to place high probabilities on the test rules so that it can correctly choose the correct parses, which contain those rules. Thus, our cross-entropy evaluation measure is (the log of) the product of the test probabilities (§6.1).

1.2.2 The Linguistic Problem: Stochasticizing Deep Structure

1.2.2.1 Background: Lexical Redundancy

Now let us turn to the linguistic motivation. Many modern theories of syntax are *lexicalist.* They agree with the dynamic-programming parser of §1.2.1.1: a legal syntax tree is formed by composing "chunks" of syntax, each of which is governed by some lexical item. The collection of chunks is called the **lexicon**, and each individual chunk is called a **lexical entry**.

The form of the chunks and the compositional mechanisms vary from theory to theory (see Fig. 2.1 on p. 36 for examples), but the basic idea remains the same: there is no language-specific grammar outside the lexicon.

This thesis responds to a problem faced by all such theories, known as "lexical redundancy." Suppose English has a lexical entry for the word `fippened` that calls for a subject and a direct object: *The beast fippened its dinner.* It is then almost certain that the English lexicon also lists a passive entry for `fippened`: *The dinner was fippened by a beast.* One might also accept reasonable odds on a bet that English lists an intransitive entry: *The beast fippened (all day)*, or perhaps, *The dinner fippened (slowly).*

The basis for these bets is that many *other* words in English, such as `cooked`, exhibit the same pattern of lexical entries.[2] A simple lexical theory would take these common patterns to be coincidences, and would have to stipulate them over and over in order to describe the behavior of `fippened`, `cooked`, and other words.

1.2.2.2 Lexical Redundancy Rules

Linguists prefer not to enlarge the grammar with such repeated stipulation. Occam's Razor calls for modeling the patterns somehow—both to improve the lexicon's elegance and to explain why a learner who observes the new word `fippened` in a single linguistic context is immediately comfortable using it in the other contexts predicted above.

So lexicalist theories generally allow rules that *derive* lexical entries from one another. Again, the form of these rules may vary from theory to theory (Fig. 2.1). They are the spiritual descendants of transformations in transformational grammar, the main difference being that they cannot modify any chunk bigger than a lexical entry, and do their work before the chunks are assembled. There is still no language-specific grammar outside the lexicon—but there is now a language-specific grammar *inside* the lexicon.

Thus, a grammar consists of a lexicon plus a system of lexical redundancy rules. The lexicon is still the final arbiter of syntax, since it may include both positive and negative exceptions to the rules. However, the rules encode the regular patterns that underlie most of the lexicon.

1.2.2.3 Introducing Rule Probabilities

What this thesis contributes to linguistics is a sensible *stochastic* treatment of these lexical redundancy rules, their exceptions, and their acquisition. Just as one can attach probabilities to the lexical entries—so as to better model competence and performance—we will attach probabilities to the rules.

[2] Transitive verbs' ability to passivize is almost without exception in English, although they do not all passivize equally often. The tendency of some transitive verbs to boast intransitive entries as well is weaker, but still systematic and worth learning. Such "Object Drop" entries are used in the Brown corpus (Kucera and Francis, 1967) about 55% of the time with `eat` and `drink`, 25% of the time with `finish`, and never with `devour`. It is useful for a learner to recognize that Object Drop is at least plausible for a novel verb, and then to be able to tune its rate of application from verb to verb.

The stochastic framework will provide an attractive perspective on a number of linguistic issues. In particular it allows a unified and gradient treatment of language-specific syntactic generalizations at all levels: exceptions, subregularities (patterns of exceptions), rules, and tendencies shared by many rules.

Just as important, the stochastic framework yields up statistical methods to *learn* all these generalizations and exceptions from data. Learning the rules is of particular interest, because they encode the deep structure of the language and because they often have semantic consequences.

1.2.2.4 An Example

Consider again the data of Table 1.1. There are clear structural relationships among the frames that appear with these six words.

The first two rows are heavily correlated, as discussed further in §1.2.3.3 below. They correspond to frames that are related by a simple edit: the addition of a `PP` (prepositional phrase) at the right edge. We can regard this as a `PP`-adjunction that transforms *to encourage loyalty* ⇒ *to encourage loyalty [with frequent-flier miles]*. More precisely, it transforms the lexical entry that licenses the former phrase into an entry that licenses the latter. Indeed, the data suggest that a word must have the first entry to have the second.[3]

The same transformation relates rows 3 and 4. It so happens that `remove` likes to be modified by an `AdvP` (adverbial phrase): *to* {*completely,properly,surgically,...*} *remove the asbestos.* The same `PP`-adjunction transformation that can be learned from rows 1 and 2 (and other data) predicts that `remove`'s mildly idiosyncratic entry in row 3 predicts its mildly idiosyncratic entry in row 4.

Meanwhile, rows 1 and 2 are themselves related to rows 3 and 4, by an `AdvP`-insertion transformation. Notice that this transformation tends to apply in certain contexts: it prefers to insert the `AdvP` immediately before the headword ——, even at the cost of splitting an infinitive, as here. We will see below how to model this preference.

Rows 1 and 2 are also related to rows 5 and 6, by a more linguistically interesting

[3]Having the second entry was required for a word to be included in Table 1.1; see the caption. The motivation for lexical redundancy rules is to explain the "coincidence" that these words all also appeared with the first entry as well.

sequence of transformations: *to <u>encourage</u> loyalty* $\Rightarrow^*$ *[Frequent-flier miles] <u>encourage</u> loyalty .* First `encourage` acquires an `NP` (noun phrase) subject at the left edge. A sentence with a subject is allowed to acquire tense as well, so a second transformation drops the infinitive marker `To`. And tensed sentences are allowed to serve as main sentences, so a third transformation can now add a final period.[4]

The next few rows undergo a similar process, except that they acquire tense differently: not by dropping the infinitive marker `To`, but by substituting a modal `Md` (*will*, *can*, *should* ...). This is a productive transformation in English.

Finally, the observations in the last three rows are lexically idiosyncratic exceptions. Unlike in previous rows, most of the zero counts here are not accidental gaps in the data, but actually reflect very low probabilities:

to <u>merge</u>/<u>encourage</u>/*<u>question</u> [with First Hospital]*

to <u>encourage</u>/<u>question</u>/*<u>merge</u> [students to consider teaching]*

Experts <u>question</u>/<u>encourage</u>/*<u>merge</u> [whether the magazine can risk it] .*

Though Table 1.1 shows only a small and somewhat artificial subset of the real data, the similarity of the frames is striking. Regarded as strings, they are all closely related by edit distance. If we ignore the edit needed to turn `PP` into `PP-TMP` (a temporal `PP`), then each frame is at edit distance 1 from some other frame, and only two of the frames achieve an edit distance of 3 from the first two frames. §2.4.2 will use this observation to define a particularly simple set of "edit" transformations to be used in the experiments.

Even the lexically idiosyncratic frames in the last few rows appear to be licensed by simple edit transformations. For example, the last row is a transformation of the fifth row: some verbs, like `question`, can type-shift their `NP` complement (*management's credibility*) to an `SBAR` complement (*whether the magazine can risk it*). The table shows that `question` tends to allow main-verb lexical enties of both sorts (but does not particularly like `PP` modifiers). So it is an idiosyncratic verb, but its idiosyncrasies do follow a pattern. We would like our statistical account of lexical redundancy rules to be able to capture such

[4]This particular sequence of three transformations is not the only reasonable way to describe this process. It would in any case be preferable to use separate LHS nonterminals $\mathtt{S}_{\mathtt{inf}}$ and $\mathtt{S}_{\mathtt{tensed}}$, so that period-insertion does not have to look non-locally in the frame's RHS to discover whether it is licensed. (Separate LHS nonterminals would also be desirable because other context-free rules subcategorize for one or the other.) See §5.5.4.3 for more on how transformations that manipulate features might work.

semi-productive processes.

1.2.3 The Statistics Problem: Capturing Covariance of Related Events

In language and in life,[5] one often needs to estimate a probability distribution over a finite or countably infinite set of mutually exclusive events. Some of the events in this set might be related in ways that are known *a priori* to a domain expert. For example, linguistic constructions may be related by lexical redundancy rules, web pages may be related by links, movies may be related by their common personnel, and so forth.

One therefore wants a model that can, if necessary, capture the fact that related events have related probabilities. Every type of *potential* relationship in the domain should correspond to a parameter that models the *actual* strength of such relationships in the data.

1.2.3.1 Previous Approaches

An obvious attempt is to use a log-linear model, also known as a maximum-entropy or Gibbs distribution. Each event is characterized by a set of features. Increasing a feature's weight will equally scale up the probabilities of all events bearing that feature (after which the probability distribution over the full set of events must be renormalized).

However, this is not quite a solution. The log-linear model learns probabilities, not relationships. A feature weight does not really model the strength of the relationship between two events e, e'. It only influences both events' probabilities. If the probability of e is altered by some *unrelated* factor (another feature), then the probability of e' does not respond unless it happens to share this additional factor (feature). So the relationship between e and e' is a 100% correlation if all other parameters are fixed,[6] but weakens quickly as the other parameters are allowed to vary.

An alternative is to use a directed graphical model (Bayesian network). Such a model can directly capture the causal relationship between e and e'. However, a graphical model has the wrong form. It describes a joint distribution over multiple random variables—so

[5]My life, anyway, and probably the reader's.

[6]Assuming that the feature with variable weight appears just as often on e' as on e. More generally, if it appears $0, 1, 2, \ldots$ times as often on e', then the relation will be absent, linear, quadratic, etc. The number of appearances of each feature on each event is traditionally set by a domain expert who defines the features, although learning it is an interesting possibility.

e and e' are not mutually exclusive and their probabilities may sum to more than 1—whereas we are interested in the distribution over mutually exclusive values $e, e', \ldots$ of a single random variable. In addition, graphical models disallow causal cycles. So they cannot model a situation where extraneous factors affecting e' will affect e *as well as* vice-versa;[7] furthermore, learning the direction of causality requires learning the discrete topology of the model.

1.2.3.2 Transformation Models

This thesis tries to fill the bill by introducing a class of "transformation models," together with natural priors and parameter estimation algorithms. The conceit in transformation models is that events can spontaneously transform into one another. If e regularly transforms into e', then any effects that raise $\Pr(e)$ also raise $\Pr(e')$. The strength of the relationship between events e and e' is represented by how often e transforms into e'.

The distribution being modeled is presumed to be a snapshot of the events after all transformations have stopped. Observing this distribution provides evidence from which we infer the parameters of the underlying transformational process. In particular, suppose that *a priori*, $e_1, e_2, e_3, \ldots$ have some relationship to $e'_1, e'_2, e'_3, \ldots$ respectively, and that empirically, variance among the $\Pr(e_i)$ explains part of the variance among the $\Pr(e'_i)$. Then we have evidence that this type of relationship is strong and each e_i tends to transform into e'_i.

Transformation models do have resemblances to log-linear and graphical models. Each relationship's strength (i.e., each transformation's probability) is modeled log-linearly in terms of features of the relationship; in fact log-linear models are a special case of transformation models.[8] And since $e' \Rightarrow e$ may have a different probability than $e \Rightarrow e'$, the model describes asymmetric causal relationships just as directed graphical models do, with

[7] Unless one adds arcs to e from all ancestors of e', and so forth. But then it would require a complex parameterization to ensure that these arcs were mediated by the strength of the relationship between e' and e.

[8] To convert a log-linear model into a transformation model, allow the initial event START to transform randomly into any other event (where the process then stops), with a probability that depends log-linearly on features of the latter. The probability of this transformation is then the probability of the event. See §7.3.4.2 for a kind of converse: a variant kind of transformation model can be regarded as a log-linear model over events that have been augmented with derivational history.

similar conditional independence and abductive ("explaining away") properties.

A terminological note: Discussions of this work can help support web searches by sticking to the term "transformation model" rather than "transformational model," as the latter collocation is already in online use (e.g., in nursing). I do however use "transformational smoothing" to refer to the effect of a transformation model.

1.2.3.3 An Example

There is a relationship between rows 1 and 2 of Table 1.1. Across the board, row 2 is (let us say) about half of row 1. We can model such a relationship by saying that for every three instances of $\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP}$, one transforms into $\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP PP}$, leaving the observations in a 2:1 ratio.

Of course, row 2 is not *exactly* half of row 1. Three explanations are available within the model:

- The counts are merely samples of the true probabilities. The small counts observed in rows 1–2 are certainly consistent with a true 2:1 ratio. This explanation becomes less attractive as the counts get large.

- Row 2 is not solely derived from row 1. There may exist other transformations that can also produce row 2's $\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP PP}$. If the probabilities of all such transformations are held fixed while others vary, then row 2's $\Pr(\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP PP})$ is a fixed linear combination of row 1's $\Pr(\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP})$ *and* the probabilities of other entries. While $\frac{1}{2}$ may be the exact coefficient of row 1's $\Pr(\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP})$ in this linear combination, other summands exist and affect row 2's $\Pr(\texttt{S} \to \texttt{To} \text{ —— } \texttt{NP PP})$ as well. So the *ratio* of the two probabilities is not $\frac{1}{2}$, although they are 100% correlated. (The slope of the regression line is $\frac{1}{2}$ but the intercept is not 0.)

- The six transformations that turn the six row-1 entries into their corresponding row-2 entries have probabilities that are loosely tied (because they share many features) but need not be identical. $\texttt{S} \to \texttt{To merge NP}$ may transform into $\texttt{S} \to \texttt{To merge NP PP}$ with some probability $> \frac{1}{3}$, thanks to headword-specific features that model the exceptional properties of `merge`, in particular its tendency to take a `PP` argument (*with* `NP` or *into* `NP`).

1.2.4 The Language-Learning Problem

The long-term motivation of this thesis is to contribute to models of language learning by human or computer. This section speculates on how this work might fit into that program.

The Achilles' heel of current statistical parsing work is its dependence on a corpus of sample parses. One would like to reduce or eliminate that dependence. Children learn language from raw speech signals, situated in a real-world environment. Could a computer induce a grammar from raw text? What statistical biases would help it do so?

1.2.4.1 Enriching EM with a Transformational Prior

For learning context-free grammars from raw text, Inside-Outside reestimation (Baker, 1979) looms large in the intellectual landscape. Yet it works poorly on language text (Lari and Young, 1990; de Marcken, 1995).

One kind of improvement is to use a prior distribution over possible grammars. Since Inside-Outside is an instance of the EM algorithm, it is possible to constrain it with a prior. Using a linguistically sensible prior may help align EM's goal of maximizing probability with language learning's goal of maximizing interpretability.

A transformation model of the lexicalized grammar offers a natural prior, which says that low-cost lexicons—in §§1.3.2–1.3.3's sense of having broad generalizations with few exceptions—are *a priori* more likely. One could also modify the prior to include substantive linguistic biases.

1.2.4.2 Failure-Driven Learning

A prior by itself is unlikely to rescue EM. One also needs some way of avoiding local maxima. Let us consider a variant of EM that grows a labeling of the data slowly from a small seed model. The strategy is inspired both by a reasonable perspective on human learning and by several successful unsupervised learning algorithms for NLP (see §8.2.4).

All the time that children are learning language from the examples they hear, they have a more pressing concern: trying to make sense of those examples. It is well known that children comprehend a wider variety of constructions than they can produce, or produce correctly. Indeed, the same is true of adults, who can successfully guess the syntax and semantics of novel words in context.

This suggests failure-driven learning: Suppose a child is unable to parse a sentence according to her current grammar. If the sentence is hopeless gibberish to her, she ignores it. But if the sentence lies at the edge of her grammar, so that a slight change to the grammar would let her parse it and hence understand it, then she has an incentive to make that change.

For example, *The beast fippened all day* requires a lexical entry e that licenses `fippened` as an intransitive verb (§1.2.2.1). If the child does not have such an entry, she may be still be able to settle on the best parse with reasonable confidence. This requires "stretching" the grammar and adding the new entry to it.

1.2.4.3 Transformational Plausibility in Failure-Driven Learning

Of course, there may be multiple ways to stretch the grammar. An alternative is to treat *all day* as a noun phrase rather than an adverbial. So for the child to have reasonable confidence in the first parse, she must consider it to require a much less painful stretch.

Hence, the child must estimate how common it is for `fippened` to be used intransitively: $\Pr(e \mid \texttt{fippened})$. If she has seen `fippened` a million times and never as an intransitive, then her estimate should be small (probably $< 10^{-6}$). But if she has seen `fippened` only once or twice, as a transitive verb, then she should be willing to back off to *other* verbs, using the fact that many transitive verbs in English can also be used intransitively.

The transformation model of this thesis can make exactly such a smoothed estimate. It interpolates in Bayesian fashion between the actual observations of `fippened` and the generalizations it has extracted about English as a whole.

Our perspective is that all possible lexical entries are always in the lexicon. The requisite entry e has been dormant in the lexicon all along, with some low probability. It may be recruited into action to parse the newly observed datum *The beast fippened all day.*

If the parse using e is much more probable overall than alternative parses, then an EM-style algorithm will consider it likely that e has now been observed. This new observation raises the probability of e for future use.

Similarly, Pinker (1989) notes that adults readily interpret novel lexical entries as the result of applying semi-productive transformations to known entries. Again, our perspective is that these entries were dormant in the lexicon all along.

1.3 The New Idea

1.3.1 A Transformation Model of the Lexicon

Triangulating from the various motivations in §1.2, the shape of the solution should now be clear.

If linguistic theory is right, then the distribution of chunks (lexical entries) in a parser's training data (§1.2.1) should provide statistical evidence of lexical redundancy transformations (§1.2.2). These transformations spontaneously select chunks for production and convert them into one another at various rates. Several can apply in sequence. Of course, these choices are not "really" spontaneous—they reflect the speaker's intentions—but they appear spontaneous to a listener or a model with no prior knowledge of those intentions.[9]

One can build a transformation model of this process (§1.2.3). The details of the model—the form of the lexical entries, the set of possible transformations, and the parameterization of transformation rates—should be chosen on linguistic grounds. They specify the substance of universal grammar. From a learning viewpoint (§1.2.4), they describe the kinds of generalizations that a learner should be sensitive to in training data.

Learning a particular language is then a matter of estimating the parameters of the model: the language-specific rates of the transformations. Once discovered, these non-zero rates predict the existence and probability of an unbounded number of new chunks. They also yield smoothed probability estimates for old chunks. A statistical parser (or parse reranker) can therefore request a probability estimate for any potential chunk it is considering.

[9] It is very common and very useful to account for the effects of unmodeled factors as some kind of random noise, even in a deterministic physical process (Kalman, 1960).

1.3.2 Stochastic Rules and Exceptions

We will have to attach probabilities to lexical redundancy rules. A typical low-probability rule in English is Yiddish-movement: *Breakfast, the beast fippened rarely.* It is a completely regular process but simply does not apply very often, so that the entry it produces for `fippened` (or any verb) has low probability, is rarely used in generation, and is avoided in comprehension (when possible) in favor of better entries. A high-probability rule corresponds to an obligatory or nearly-obligatory transformation.

The plan that Sofia will swallow illustrates the use of such probabilities in comprehension. To disambiguate this `NP`, the hearer must ask herself, among other things, whether *swallow* is more likely to extract its object or drop it.[10] Evidence for the extraction reading is that the extraction rule has higher probability in English—at least in the Penn Treebank, where more verbs undergo extraction than object drop, both by type and by token.

We treat not only rules but exceptions in a graded fashion. Taken together, the rules predict probabilities for all lexical entries (from the probabilities of other entries). An entry is exceptional *to the degree* that its probability deviates from prediction. Entries that are "listed" in the lexicon are simply entries that occur much more often than predicted. All other entries are derived.

We are concerned with the *cost* of a grammar rather than its size. The lexicon holds *every* possible lexical entry,[11] in the sense of assigning it a non-zero (but usually negligible!) probability. This is perfectly fine so long as only a few of these probabilities need to be strongly listed and the rest are highly predictable.

Our cost model for a grammar follows Occam's Razor: it penalizes both generalizations and exceptions. Exceptions are expensive, so the lexicon avoids adding strong exceptions unless justified by a good deal of evidence. Adding a rule is expensive, for a learner or a linguist, but worth it if it makes several lexical entries less exceptional. In short, a linguist or learner will try to account for the data by deriving all lexical entries from a few listed entries with a few rules. The minimum-cost lexicon is the one that multiplies entities to

[10] On the object extraction reading, Sofia will swallow the plan; on the object drop reading, the plan's content is that Sofia will make a swallowing noise, perhaps to distract the guards.

[11] "Possible" according to the syntactic theory being used—that is, the Universal Grammar that determines the allowable form of entries and rules.

adjoin PP to S following NP	adjoin AdvP to S following NP	···
adjoin PP to S following $\bar{\texttt{S}}$	adjoin AdvP to S following $\bar{\texttt{S}}$	···
adjoin PP to S at left edge	adjoin AdvP to S at left edge	···
⋮	⋮	

Figure 1.1: A large set of rules characterized by a smaller set of underlying features (in this case, *what* is being adjoined and *where*).

necessity but not beyond.

1.3.3 Rules and Their Features

A twist is that we do not take the transformational rules to be the deepest elements of the grammar. Again, let us concern ourselves with the cost of the grammar rather than its size. The grammar includes *every* possible rule with some non-zero (but usually negligible!) probability. Again, this is fine so longer as the rules' probabilities are predictable from a small, finite set of rule features.

The motivation for rule features is to allow a more sensitive treatment of rule probabilities. Instead of having one rule each for PP-adjunction and AdvP-adjunction, we can have many of each kind, roughly as shown in Fig. 1.1. This improves both the descriptive and the explanatory power of the theory. By splitting PP-adjunction into several context-sensitive rules, which share only some of their features, we can assign them different probabilities according to *where* the PP is adjoined. And since rows as well as columns in Fig. 1.1 share features, we can also capture generalizations that cut across PP and AdvP: in English the probabilities decrease downward in the table, just as they decrease rightward. (In practice, we use tables with many more than two dimensions.)

Now it is not really adding a rule that is expensive; it is adding a feature. A feature is influential in the grammar to the extent that its weight is far from zero. So a linguist or learner will prefer to keep the number of influential features small. In other words, a low-cost grammar is one that is described by a few broad generalizations.

Finally, rule features can unify the treatment of rules and exceptions. We can take such a fine-grained view that PP-adjunction to two separate entries e and e' is regarded as

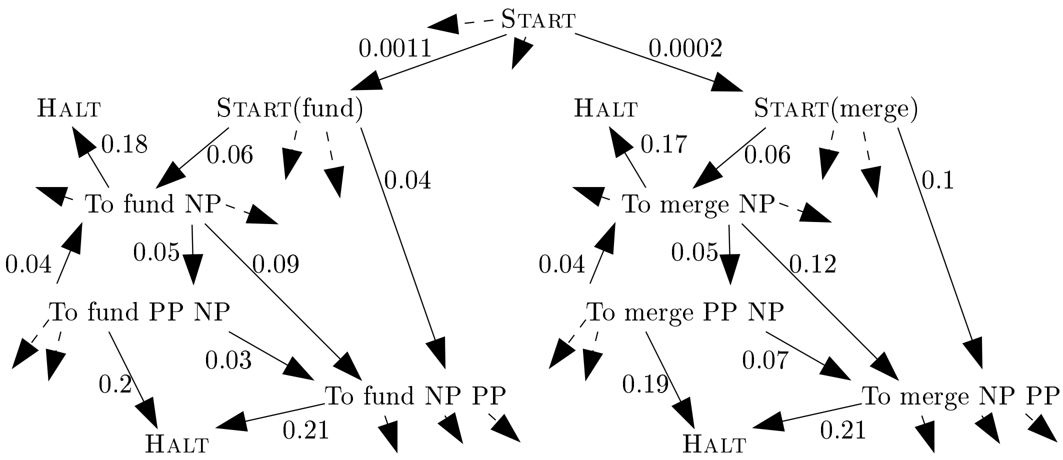


Figure 1.2: A fragment of a transformation model, showing arc probabilities. (Dashed arrows stand for other arcs not shown in this figure.)

two separate rules. Even if e and e' are identical except for their headword, the two rules' probabilities can differ if we have features that are specific to e and e'. These features suffice to describe exceptions.

So the grammar of English, say, is simply a collection of feature weights, which together describe exceptions, rules, and generalizations about the rules. Occam's Razor reduces to a simple principle: *keep all feature weights close to zero.* To learn a language, we try to find small feature weights that do a good job of modeling the data.

1.4 A Sketch of Transformation Models

We now see by example how the idea of the above section is turned into mathematics. For example, "low cost" in the above section will be realized as "high prior probability."

1.4.1 The Transformation Graph

Part of a transformation model for the lexicon is shown in Fig. 1.2. The vertices are linguistically possible lexical entries; the arcs (directed edges) represent linguistically possible transformations. The set of arcs leaving any given vertex has total probability 1.

A transformation model gives a language-specific probability distribution over the lexical entries. To sample from this distribution, take a random walk from the special vertex START to the special node HALT. The last lexical entry reached before HALT is the sample.

For example, the random walk might reach `S` $\rightarrow$ `To` `fund` `NP` in two steps and simply halt there. This happens with probability $0.0011 \cdot 0.06 \cdot 0.18$. Or, having arrived at `S` $\rightarrow$ `To` `fund` `NP`, it might transform it into `S` $\rightarrow$ `To` `fund` `PP` `NP` and then further to `S` $\rightarrow$ `To` `fund` `NP` `PP` before halting at the latter.

The probability distribution over lexical entries is entirely determined by the probabilities on the arcs. Crucially, if one increases the probability of an arc that goes to `S` $\rightarrow$ `To` `fund` `NP`, then the distribution allocates more probability mass not only to `S` $\rightarrow$ `To` `fund` `NP` but also to its child `S` $\rightarrow$ `To` `fund` `NP` `PP` (not to mention *its* children).[12] It is in this sense that the graph captures covariance among the entries' probabilities.

1.4.2 Parameter Tying

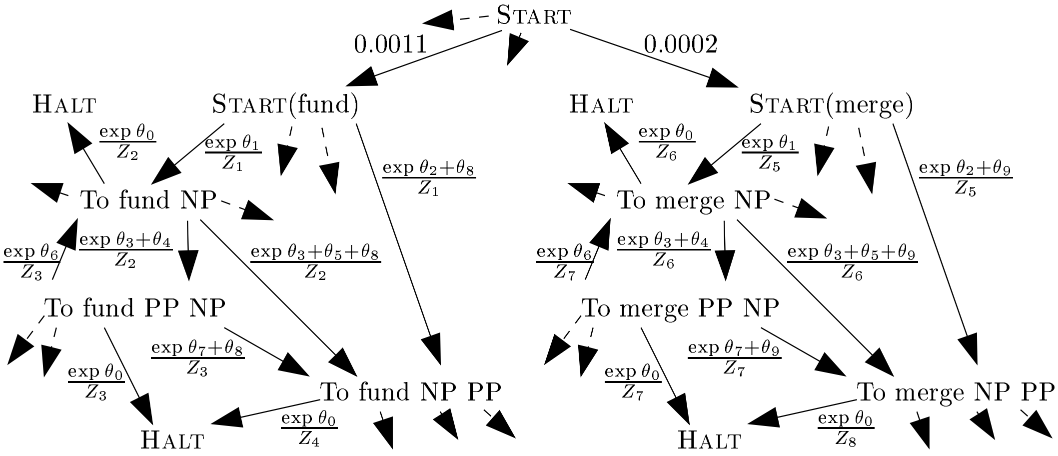
The graph in Fig. 1.2 is language-independent, but the arc probabilities are language specific. In principle the graph is infinite, since it assumes that lexical entries may take arbitrarily many descendants.

Does this mean that a language learner must estimate infinitely many language-specific probabilities, one for each arc in the graph? No, because many of the arcs have something in common. Arcs that represent the same transformation will be constrained to have the same probability. Arcs that represent similar transformations will be constrained to have similar probabilities.

To accomplish this, we parameterize the arc probabilities with a small, finite set of feature weights $\theta_1, \theta_2, \ldots \in \mathbb{R}$. The probabilities are defined in terms of these weights as shown in Fig. 1.3. Notice that this is a conditional log-linear (maximum-entropy) model of $\Pr(\text{arc} \mid \text{vertex})$, i.e., of $\Pr(\text{transformation} \mid \text{entry to be transformed})$.

For example, four transformations shown in Fig. 1.3 have the feature that they insert a prepositional phrase (`PP`) somewhere or other. That feature's weight θ_3 is used for all such transformations' arcs. The transformations available at each lexical entry compete with

[12] I.e., raising the probability of the arc to `S` $\rightarrow$ `To` `fund` `NP` increases the chances of sampling that entry and its descendants. The random walk is more likely to reach them, hence more likely to halt at them.



θ_0	halts	θ_3	inserts `PP`	θ_6	deletes `PP`
θ_1	chooses `To` —— `NP`	θ_4	inserts `PP` before `NP`	θ_7	moves `NP` right past `PP`
θ_2	chooses `To` —— `NP PP`	θ_5	inserts `PP` before right edge	θ_8	yields `To fund NP PP`
				θ_9	yields `To merge NP PP`

Figure 1.3: How the arc probabilities in Fig. 1.2 were determined from feature weights $\vec{\theta}$. The Z values are chosen so that the arcs leaving each vertex have total probability 1.

one another; and if θ_3 is positive, then `PP`-inserting transformations have an advantage in this competition.

If `PP`-insertion is common among the frequent words where we have a chance to observe it, we will learn that θ_3 is large in this language. This generalization also raises the probabilities of `PP`-insertion transformations that have never been attested, simply because these transformations also have θ_3. So it affects our estimates of lexical entries even for poorly-observed words.

As another example, the weight θ_9 is known as a "per-event" weight. It appears on all and only the arcs to `S` $\to$ `To merge NP PP`. Large θ_9 says that that lexical entry—*and its descendants in the graph* (§1.4.1)— have higher probabilities than one would otherwise expect. Put another way, this single parameter value says that `merge` is listed as a transitive verb with a role for a `PP`, and (other things equal) enjoys all the ordinary privileges of such verbs.

To flesh out the first paragraph of this section: The vertex labels, graph topology, and arc parameters of Fig. 1.3 are language-independent. More bluntly, this figure constitutes universal grammar. Learning a specific language's syntax is now just a matter of learning the weight vector $\vec{\theta}$. In other words, it is matter of learning what features of a linguistically possible transformation make it likely in the language.

1.4.3 The Prior

"With three parameters I can fit an elephant," wrote Lord Kelvin. There is always a danger of having so many parameters that one overfits the training data. On the other hand, having too few parameters is also dangerous, since elephants are sometimes the rule rather than the exception. Our approach is to permit very many degrees of freedom, but to use a prior to discourage their indiscriminate use.

While our use of feature weights (§1.4.2) may reduce the number of parameters, it still leaves too many parameters to estimate them freely from the available data. Indeed, the use of per-event weights such as θ_9 means that there are usually more parameters than training data—an underdetermined model.

Simply setting the weights to maximize the likelihood of training data would therefore lead to serious overfitting. Rather than smooth the data transformationally, it would just yield the maximum-likelihood multinomial, in which the estimated probability of a lexical entry is simply proportional to the number of times it was observed (perhaps zero). This is exactly the situation we were trying to avoid in §1.2.1.2.

The antidote is to assume a prior distribution over the parameters. We can then learn by maximizing not the likelihood, but the joint probability of the training data and parameters:

$$\underbrace{\Pr(\text{observed lexical entries} \mid \theta)}_{\substack{\text{“likelihood”} \\ = \Pr(\text{training data}|\text{grammar})}} \cdot \qquad \underbrace{\Pr(\theta)}_{\substack{\text{“prior”} \\ = \Pr(\text{grammar})}} \tag{1.2}$$

This is simply an instantiation of equation (1.1).

What prior $\Pr(\theta)$ will we use? If a feature has weight 0, it has no effect on the probabilities of arcs that bear it. We expect *a priori* that most features are innocuous in most languages: $\theta_i \sim N(0, \sigma^2)$, so crosslinguistically feature weights tend to be close to 0. It takes a fair amount of evidence to convince us that in the language we are learning, the unexpected is true and a particular feature θ_i has high weight.

If PP-insertion before NP has a specially high or low probability, the parameter θ_4 lets us model that probability accurately. But without such evidence we will try to respect our prior belief that $\theta_4 \approx 0$, meaning that PP is about as likely to insert before NP as anywhere else. Similarly, without a lot of evidence that S $\to$ To merge NP PP has a special probability, we will tend to assume $\theta_9 \approx 0$, meaning that Pr(S $\to$ To merge NP PP) is predicted by the same transformational processes as Pr(S $\to$ To fund NP PP.

In practice, the evidence about S $\to$ To merge NP PP is rarely either plentiful or absent. So rather than choose the empirically observed or the transformationally predicted probability, we must interpolate between them to maximize equation (1.2). This is the transformational smoothing effect.

It is really the prior that encourages transformational smoothing to find generalizations. Since the model is underdetermined, there are multiple ways to account for the training data with parameters. If during learning, S $\to$ To fund NP PP and S $\to$ To merge NP PP are both observed more often in training data than predicted by the model, then one has two alternatives:

- Raise their probabilities separately by increasing the per-event weights θ_8, θ_9. This makes two stipulations.
- Raise their probabilities together by raising θ_4 (or perhaps even θ_3). This expresses a broader generalization about PP insertion.

Both alternatives equally help the likelihood term of equation (1.2). So the latter is preferred, other things equal, because it hurts the prior term less.

1.5 Results

1.5.1 Algorithmic Results

In Fig. 1.2, the arc probabilities define a recurrence relation in which each lexical entry's probability is a linear function of its parents' probabilities. One can obtain the distribution over entries by solving this linear system.[13] Using sparse matrix methods, this can be done in $O(n|F|)$ time and $O(n)$ space,[14] where n is the number of vertices and $|F|$ is the total number of feature weight tokens in Fig. 1.3 (the "size" of the model).

One can improve the objective function equation (1.2) by adjusting θ, using gradient-based optimization or Expectation-Maximization. One step of either method—computing the gradient or the expected traversals of each feature—corresponds to solving a linear system, again in $O(n|F|)$ time and $O(n)$ space.

If n is large or infinite it is necessary to use approximations for all these methods (solving the model, computing the gradient, and running EM). One can consider only random walks of some length T or less. Then the runtime and space become $O(T|F|_T)$ and $O(nT + k)$, where $|F|_T$ is the size of the part of the model that can be reached from START by paths of length $\leq T$, and k is the number of feature types. The algorithms are closely related to propagation and back-propagation through time in "unrolled" recurrent neural networks (§8.5.4), with some additional optimizations. Improving on this, a more flexible class of algorithms is derived from relaxation methods.

Because our transformation model of the lexicon has a particular (not uncommon) form, some additional model-specific optimizations are possible. Most importantly, one can exploit the fact that the subgraphs corresponding to different headwords have isomorphic topology and similar probabilities.

1.5.2 Empirical Results on the Test Set

§6.7 compares several modeling techniques, using comparable features, according to how well they predict $\texttt{S} \to \cdots$ lexical entries in the Penn Treebank. A simple transformation model, estimated without great care in the smoothing parameters, reduces test-set perplexity by 20% over the best model from the literature. This reduction is comparable to doubling the amount of training data.

The reduction is somewhat greater on a smaller training set, highlighting the model's

[13]If the transformation graph is acyclic, the solution is considerably faster. However, disallowing cycles would force the model designer to decide in advance whether it is intransitives that are derived from transitives, or vice versa, rather than learning this from data.

[14]Alternatively, $O(|F| + n|P|)$ time and $O(|P|)$ space, where $|P|$ is the number of arcs in Fig. 1.3.

ability to generalize from sparse data, and can be increased somewhat by model averaging.

The improvement due to transformation modeling seems to stem from three sources: fitting the training data more exactly, generalizing better, and weighting the backoff evidence in a Bayesian fashion.

These experiments using comparable features show that transformational smoothing can beat other methods even without wiring any real linguistics into the transformation graph. In future work, we hope to increase its advantage—especially by adding "interesting" transformations like extraction, passivization, and tense formation, which the other methods have little hope of modeling—and translating this advantage into gains in parsing performance, particularly on limited amounts of data.

1.5.3 Empirical Results on the Example

One can get a qualitative sense of how transformational smoothing performs, relative to the best model from previous literature, by comparing their assessments of the lexical entries in Tables 1.3 and 1.4.

- The average entry in the table is 1.44 times as likely as before (geometric mean). Equivalently, the set of table entries has 31% lower perplexity.[15]

- The transformation model fits the training data more closely (because it can model exceptions). It assigns 53% lower perplexity to the training set in Table 1.1. Moreover, on these test data it separates observed and unobserved entries, assigning probabilities > 0.05 to the former and < 0.025 to the latter. Such a separation is not observed for the competing model.

- The transformation model also assigns higher probabilities to the unobserved entries in this table, reducing the perplexity of that set by 62%.

- The transformation model wastes considerably less probability on the "other" line—the multitude of frames that did not appear with any of these words. Indeed, it allocates more than 50% as much *total* probability to the entries in the table.

[15] Of course, this is not a typical test set: common and uncommon entries alike appear exactly once. As a result it has twice the perplexity of the real test set.

- One can observe transformational generalizations at work between S $\rightarrow$ NP —— NP . and S $\rightarrow$ NP —— NP PP . (the frames in rows 5–6 of Table 1.4). These frames receive probabilities of about 0.0076 and 0.0026 when neither has been observed with a word (columns 4–6). But columns 1–2 show that observing either one increases the probability of the other.

  To be precise, observing S $\rightarrow$ NP encourage NP PP . increases its estimated probability by quite a lot. Thanks to a Delete-PP transformation, S $\rightarrow$ NP encourage NP . then automatically increases by 18% as much. There is also an Insert-PP transformation giving the converse effect: twice observing S $\rightarrow$ NP question NP . increases its estimated probability; then thanks to Insert-PP, S $\rightarrow$ NP question NP PP . automatically increases by about 4% as much.[16]

- Transformational smoothing smooths the probabilities of observed entries, not just unobserved ones. In rows 3–4, as in rows 5–6, the expected probabilities of entries with and without PP are in about 3:1 ratio. But S $\rightarrow$ To AdvP remove NP and S $\rightarrow$ To AdvP remove NP PP have been *observed* in 1:1 ratio. The estimates in the table interpolate between these: they have the compromise ratio of 1.6 : 1.

The first few bullet points above highlight the fact that the competing model of lexical entries, a "bigram model," is relatively indiscriminate as to where it throws probability. §6.6.4 remedies this by improving the competing model, instead using maximum likelihood with backoff from the headword and backoff to the bigram model.

[16]It is surprising, but consistently the case, that deletions emerge with higher probability than insertions. There are two possible explanations. One is that this untraditional result should be regarded as correct, on the grounds that most dependents fill semantic roles and are more like arguments than adjuncts. (Then most variation among frames could be regarded as arising from deleting subcategorized roles rather than inserting adjuncts.) The other explanation is that it is harder to *learn* insertions. For a given lexical entry, a few deletion transformations are competing with literally hundreds of insertion transformations that want to insert various nonterminals at various positions in the entry. This necessarily leads to a generic bias against insertions, which the model must learn to overcome for particular kinds of insertions.

In principle, a transformation model will perform abduction (just like a Bayes net), reasoning backwards from effects to their causes. Thus, even if Insert-PP has probability 0, an observation of f_2 = S $\rightarrow$ NP question NP . is still weak indirect evidence of f_1 = S $\rightarrow$ NP question NP PP . since the f_1 is a possible precursor of the observation f_2 under Delete-PP. However, the model used in the experiments of this thesis does not have enough free parameters to effectively abduce lexical entries like f_1 that were never observed. (As §6.4.3 explains, only other observed entries are permitted to have exceptional probabilities that could be tuned to account for f_2.)

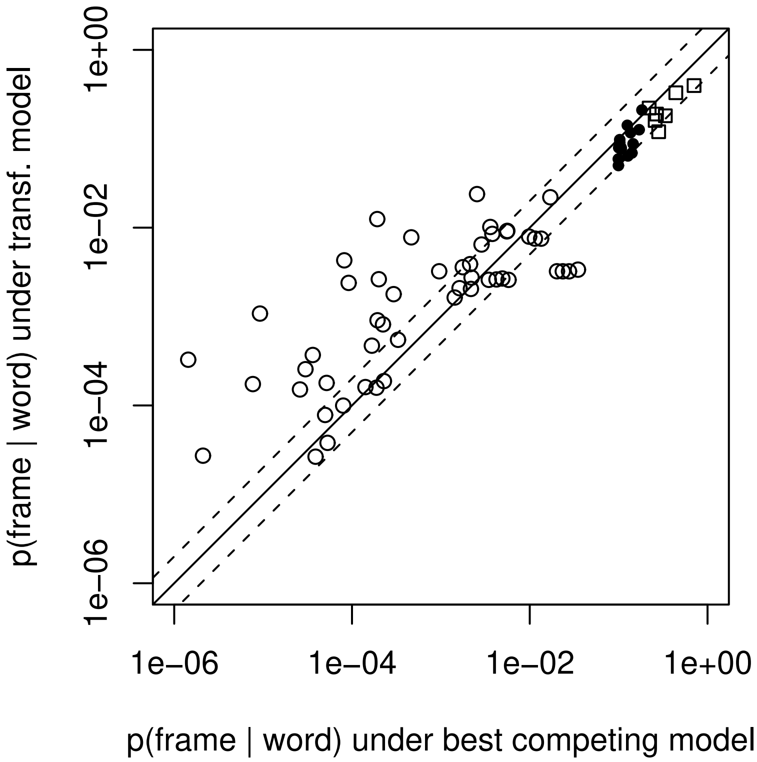


Figure 1.4: All probabilities in Table 1.4, plotted against the corresponding probabilities from the best competing model (§6.6.4—an improvement on the model discussed in most of §1.5.3). Lexical entries receiving a higher probability from the transformation model fall above the middle diagonal. The dashed outer diagonals mark a factor-of-2 difference between the two models. The plotting symbol indicates the number of training observations: $\circ = 0$, $\bullet = 1$, $\square \geq 2$.

In the final results (§6.7.1), that improved model was the best of the transformation-free models. Fig. 1.4 shows that it allocates slightly more probability to training data, but that the transformation model quite consistently assigns more probability to the novel "test" data in our example. The exceptions (slightly below the diagonal) are novel entries for the three frames that end in periods.[17] The biggest winners under the transformation model are some of the novel entries for `fund`, which was observed with the fewest and most similar frames in training data; the transformation model extrapolated more aggressively from these to the related frames in the table.[18]

1.6 Structure of the Thesis

To a large extent, the remaining chapters can be read independently, since they lay out different facets of the work: linguistics, statistics, algorithms, parsing, and experimental evaluation. However, the chapters are arranged in a natural sequence and sometimes refer to one another.

This introductory **Chapter 1** used an example (Table 1.1) to show how four problems converge on a single approach: the engineering problem of generalizing better from data when training a statistical parser (§1.2.1), the linguistic problem of attaching probabilities to lexical redundancy rules and exceptions (§1.2.2), the statistical problem of modeling positive covariance in the probabilities of mutually exclusive events (§1.2.3), and the learning problem of stretching a grammar to interpret new utterances (§1.2.4). Taken together, these problems motivate a new approach to grammar cost (§1.3) that can be formalized as a new kind of statistical model (§1.4). The approach is algorithmically manageable (§1.5.1) and yields an empirical improvement over work from the literature (§§1.5.2–1.5.3). So much for the overview.

Chapter 2 presents the linguistic idea of the thesis. It begins by describing lexicalized theories of syntax (§2.1) and motivating them (§2.2). Such a theory describes a language

[17]The transformation model did not learn a high probability for period-insertion. Even in general it is tricky to learn high rates for insertions (footnote 16), and in this case, the model did not have access to the "tensed verb" feature on which period-insertion should be conditioned (see footnote 4 on p. 10).

[18]Part of the reason is that the backoff technique (§6.6.2) in the competing model did not back off very far from the observed data for `fund`, taking the lack of 1's in `fund`'s column in Table 1.1 as a sign that `fund` had been well observed.

on four levels (§2.1.5): (1) strings, (2) a lexicon of "chunks" of surface structure, (3) a language-specific deep structure that interrelates the "chunks," and (4) a Universal Grammar that constrains and influences the form of the deep structure. Statistics is arguably part of humans' knowledge of language, and can be used to enrich each level (§2.3).[19] In particular, chunks have probabilities (§2.3.4), and we propose that deep structure concerns *correlated* probabilities and can be modeled by stochastic transformations on the lexical entries (§2.3.5). Since real grammars contain idiosyncrasies, these correlations are often imperfect; but the imperfections (which are matters of degree) can be modeled as well, and in such a way that a learning algorithm will attempt to characterize the lexicon in terms of the strongest correlations and the mildest exceptions possible (§2.3.6).

Chapter 2 continues by outlining the particular lexicalized theory to be used in the experiments. In this simple theory, lexical entries are "flat" context-free rules (§2.4.1). Transformations act to insert, delete, replace, or permute the nonterminals in a rule (§2.4.2). Such transformations serve to modify semantic argument structure, and their effects are evident in the training data (§2.4.2.1). Other linguistic transformations could also be useful (§2.4.3). The chapter closes by reviewing related work on lexical organization (§2.5.1), lexicon smoothing (§2.5.2), the extraction of subcategorization frames from text (§2.5.3), the modeling of optionality in subcategorization frames (§2.5.4), translation and summarization based on local syntactic transformations (§2.5.5), and approaches to grammar learning based on edit distance (§2.5.6) or priors on grammars (§2.5.7).

Chapter 3 presents the statistical framework (without reference to the linguistic application), just as Chapter 2 presented the linguistic framework. It begins with a review of probability notation and Bayesian smoothing (§3.1). Then it proposes a new class of statistical models (§3.2). These "transformation models" are related to random walks (§3.3). The chapter continues with practical advice about how to compute with transformation models (§3.4), how to design priors for them (§3.5), and how to design models that can capture exceptions (§3.6).

Transformation models are designed to model probability distributions in which "related" events have correlated probabilities (§3.7.1). §3.7.2 sketches the application to

[19]§2.3.7 meditates on the consequences for grammaticality judgments.

smoothing of a syntactic lexicon. §3.8 analyzes a very simple transformation model in detail, showing that the prior prefers to capture generalizations (§3.8.4) but that the evidence can motivate exceptions (§3.8.5). Estimating the parameters of the model yields a smoothed distribution (§3.8.6). Qualitatively, frequent events in the training data influence the smoothing of infrequent events more than vice-versa, but (fortunately) not in direct proportion to their frequency; rather, they are influential to the extent that their probabilities are nailed down by the evidence (§3.8.7). Parameter estimation can be difficult, since while the prior is a unimodal distribution over distributions (§3.8.8), the posterior unfortunately has local maxima. The chapter closes by proposing a variation, "perturbed" models (§3.9, which model exceptions in a slightly different way (§3.9.3): such models introduce extra parameters that correspond to flow multipliers in generalized network flow problems.

Chapter 4 is the computational heart of the thesis. It presents parameter estimation methods for arbitrary transformation models. §4.1 defines transformation models again, this time with a concise matrix notation that is used throughout the chapter. It also restates the objective function to be maximized during training (§4.1.4) and the evaluation function for testing (§4.1.5).

It is possible to approximate the objective function by a simple and flexible relaxation algorithm (§4.2), and to find the gradient of this approximation by "back-relaxation" (§4.3), for which a correctness proof is given (§4.3.4). The method can be extended without much difficulty to handle perturbed models as well (§4.4).

Chapter 4 closes with some ways to make relaxation and back-relaxation more efficient. Many arcs in the transformation graph do not affect the computation much, or at all, and can be ignored for speed (§4.5.2). Another important optimization applies to a common class of transformation models that have repetitive topology (§4.5.3). For example, most headwords agree on the probabilities of most syntactic transformations, so rather than repeat all computations for each headword, it is only necessary to propagate a few differences from a template.

Chapter 5 places the work in the context of statistical natural-language parsing. It sketches how one might practically use a transformationally smoothed syntactic lexicon in a parsing system. While transformational smoothing could be used with any lexicalized theory of grammar, the framework assumed here is lexicalized, flattened context-free grammar (§2.4.1).[20] §§5.1–5.3 formally define this framework. In particular, they define the probabilities of syntax trees in terms of probabilities in the lexicon, using a standard generative model in which nonterminals are recursively expanded into full lexical entries. Each expansion can be broken down into two steps (§5.4.1), both of which can exploit the smoothed lexicon. The first step is a contextually appropriate choice of headword for the nonterminal; since this choice must be compatible with the nonterminal (representing the headword's maximal projection) as well as with the governing lexical item, it is partly a syntactic choice, and the syntactic lexicon can help determine its probability (§5.4.3). The second step is to choose a sequence of dependent nonterminals for that headword; specifying the probability of this choice is the main job of the syntactic lexicon (§5.4.2). The two steps can be cleanly combined under certain Naive-Bayes assumptions (§5.4.5).

The rest of Chapter 5 is "armchair linguistics," conducted in the interest of concreteness and with an eye toward future experiments. It argues that with appropriate tricks, the lexicalized, flattened context-free approach would in principle be good enough to accommodate and exploit a variety of linguistic devices: category-changing transformations (§5.4.4), morphemes (§5.5.1.1), semantic clusters of words (§5.5.1.2), word senses (§5.5.1.3), long-distance movement (§5.5.3), syntactic as well as semantic heads (§§5.5.4.1–5.5.4.2), feature agreement and checking (§5.5.4.3), dependencies on more than one word (§5.5.4.4), thematic roles (§5.5.4.5), and dependence on greater amounts of context (§5.5.5).

Chapter 6 contains the experimental evaluation. Following the generative model of syntax trees in the previous chapter, the measure used for comparison is the perplexity of the right-hand-side of a context free rule (§6.1). The training and test data were lexical entries extracted from the Penn Treebank (§6.2); there were substantial amounts of novelty in the test data (§6.3).

The chapter continues by specifying the details of the transformation model (§6.4) and the methods for estimating its parameters (§6.5) that were used in the experiments. The

[20] That is, each context-free rule (lexical entry) rewrites a maximal projection in one step as a headword together with all its dependents. The resulting trees are similar in spirit to dependency trees, except that the subtrees are labeled with nonterminals rather than semantic roles.

competing models include a few from the literature (§6.6.1), with appropriate smoothing (§6.6.2), and tested using both flat and non-flat lexical entries (§6.6.3). Some new hybrid models are also considered (§6.6.4).

The experimental results examine the competing models and the overall perplexity reduction (§6.7.1), as well as the consistency (§6.7.2), difficulty (§6.7.3), and training-size sensitivity (§6.7.4) of this reduction. It used various additional experiments to investigate the contributions of memorizing the training examples (§6.7.1), using a Bayesian scheme to weight backoff evidence (§6.7.5), and generalizing better (§6.7.6). Finally, it briefly looks at the time course of learning and the distribution of learned weights (§6.7.7).

The experiments having been described, **Chapter 7** picks up where Chapter 3 left off, and makes some further remarks on transformational smoothing. It draws connections from transformation models to Markov processes (§7.1.1), finite-state machines (§7.1.2), recurrent neural networks (§7.1.3), graphical models (§7.1.4), and Bayesian backoff (§7.1.5). It also gives some non-linguistic applications of transformation models (§7.2). Finally, it offers a number of possible variations on the formalism. One can modify the form of the model (§7.3.1) or the form of the prior (§7.3.2). In fact, a simple modification to the prior yields Zipfian (power-law) behavior of a sort often observed in languages (§7.3.3). One can also interpret the model parameters differently so that the transformational process is blessed with "lookahead" (§7.3.4): there are at least three ways to do this, including global normalization of path probabilities (§7.3.4.2) and a "transformational circuit" approach inspired by current flow in electrical networks (§7.3.4.3).

Chapter 8 similarly augments Chapter 4, offering some additional algorithms for estimating the parameters of transformation models. §8.1 shows that the objective function and its gradient can be computed in closed form if desired (as opposed to the converging relaxation algorithm of Chapter 4). Moreover, the computations can be made to exploit the sparsity of the transformation graph. An alternative to gradient descent is to use Expectation-Maximization (§8.2), where the hidden variables are the paths in the transformation graph that generated the observed events. The M step of EM uses these paths to estimate the parameters of a log-linear distribution (§8.2.1), using a standard method like Improved Iterative Scaling (§8.2.2). The E step of EM reconstructs the paths for use by the M step: it is conceptually related to the forward-backward algorithm, and consists of solving a single sparse linear system that is defined by the training dataset and current parameters (§8.2.3). Some variations on EM are discussed in §8.2.4, including an algorithm for the Viterbi approximation and confidence-weighted approaches, and §8.3 shows that the central computation of EM can be efficiently approximated using the very same back-relaxation algorithm proposed in Chapter 4.

§8.4 discusses when, why and how one might wish to renormalize the probability distribution $\vec{p}$ defined by a transformation model. The chapter closes with somewhat tedious generalizations of two of the ideas from Chapter 4. The (back-)relaxation algorithm can be replaced with a pedagogically simpler (back-)propagation algorithm (§8.5) that is related to back-propagation in recurrent neural networks. The "template" approach of §4.5.3 can also be generalized to a variety of other circumstances (§8.6), for both exact and approximate algorithms.

Chapter 9 offers some brief conclusions and a discussion of future work.

Chapter 2

Lexicalized Syntax and Probabilities

A major goal of this thesis is to recast certain long-standing linguistic intuitions in statistical terms. Statistical models in NLP have gotten increasingly sophisticated over the past decade. Incorporating linguistic insights is one way to continue improving their accuracy and the usefulness of their output. Conversely, adding statistics to linguistic models is a way for linguists to describe competence in a way that supports performance and learning.

This chapter lays out the basic linguistic territory and sketches how statistics will enter the picture. The statistical approach is developed and tested in later chapters.

2.1 The Shape of the Lexicon

2.1.1 What Goes Into a Linguistic Lexicon?

The Oxford English Dictionary defines `lexicon` as follows:

> 2. Linguistics. The complete set of meaningful units in a language; the words, etc., as in a dictionary, but without the definitions. (Opp. grammar sb.)

However, the modern use of the term is broader. No pure or computational linguist would exclude definitions or grammar from the (mental) lexicon. Rather, the lexicon is the repository of all *word-specific information.*

This thesis focuses on just the *syntactic* information associated with words. (More precisely, with morphemes, though we use the term "word" for convenience.)

2.1.2 Regular vs. Irregular Forms

The original conception of the lexicon was that it should be as small as possible, listing only *unpredictable* word-specific information. Bloomfield (1933) wrote: "The lexicon is really an appendix of the grammar, a list of basic irregularities." The morphologists DiSciullo and Williams (1987) famously characterized it as "a prison—it contains only the lawless, and the only thing that its inmates have in common is lawlessness."

But a competing view is that the lexicon is a convenient repository for regular as well as irregular items. Once the prison is built, it makes a serviceable shelter for everyone, so why bother building houses too? As many syntactic constructions are governed by word-specific requirements of their headwords—e.g., a verb selects its complements, and `whom` requires object extraction from its sentential complement—the lexicon turns out to be a convenient place to stow information about syntax in general.

2.1.3 Lexicalized Theories of Syntax

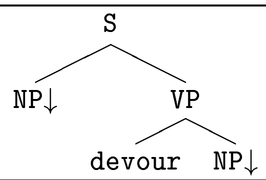
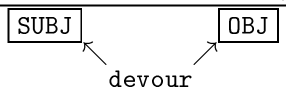
Adopting the latter argument, recent "lexicalized" syntactic theories tend to push *all* language-specific syntax into the lexicon.[1] That is, they regard all language-specific syntax as resulting from the properties (principally subcategorization) of particular function and content words.

The lexicon is then a collection of language-specific building blocks. The mechanisms that combine these building blocks into utterances are simple, language-independent, and universally specified once and for all by the formalism.

Fig. 2.1 shows sample lexical entries and compositional mechanisms for several lexicalized theories.

The simplicity and uniformity of these lexicalized theories is both a virtue and an apparent vice. On the one hand, they are easy to understand and to compute with. On the other, there is now a great deal of unexplained redundancy in the lexicon. For

[1] As well as morphology: see e.g. (Butterworth, 1983).

theory	entry for `devour`	composition mechanisms	transformation mechanism
flattened lexicalized CFG (as used in this thesis)	$\texttt{S} \rightarrow \texttt{NP}$ —— $\texttt{NP}$	context-free rewrites	transformations on CF rules
LFG (Bresnan and Kaplan, 1982)	V, $(\uparrow\text{TENSE}) = \text{INF}$ $(\uparrow\text{PRED}) = \text{`DEVOUR}\langle(\uparrow\text{SUBJ})(\uparrow\text{OBJ})\rangle\text{'}$	unification under control of CF rules	lexical rules
CG (e.g., (Lambek, 1958; Steedman, 1990))	$\texttt{(S\\NP)/NP}$ $\lambda y.\lambda x.\text{DEVOUR}(x,y)$	application, composition	lexical rules (Carpenter, 1992)
LTAG (Joshi and Schabes, 1991)		substitution, adjunction	e.g., lexical redistribution rules (Xia et al., 1999)
link grammar (Sleator and Temperley, 1993)		linking	?
HPSG (Pollard and Sag, 1994)	$\begin{bmatrix} \text{HEAD} & \text{verb[inf]} \\ \text{SUBJ} & \langle \boxed{1}\text{NP} \rangle \\ \text{COMPS} & \langle \boxed{2}\text{NP} \rangle \\ \text{ARG-S} & \langle \boxed{1},\boxed{2} \rangle \end{bmatrix}$	unification under control of universal schemata	metarules
minimalism (Chomsky, 1995)	`[+AgrS(1s)] [+AgrO] [+Acc] [+EPP]`	move, merge	?

Note: Some other lexicalized formalisms resemble the first one in the above table: notably Dependency Grammar (Tesnière, 1959; Mel'čuk, 1988; Milward, 1994) and Head Automaton Grammar (Alshawi, 1996; Eisner and Satta, 1999).

Figure 2.1: The lexical entry that licenses the unmarked use of `devour` (as in *Sara devoured the apple*), under each of several popular theories. (In these entries, DEVOUR and ARG-S describe semantics, not syntax.)

instance, all regular transitive verbs (such as `devour`) "happen" to list both active and passive entries. Many also list intransitive variants.

2.1.4 Mechanisms for Lexical Redundancy

Every lexicalized theory solves the redundancy problem by positing additional mechanisms that operate *within* the lexicon. Gazdar et al. (1985, p. 65) describe this as "using a grammar to generate one's grammar."[2]

For example, a lexical-functional grammar (LFG) for English would include Bresnan's (1978) rule of Passivization. This generates a lexical entry for passive `eaten` from the lexical entry for active `eat`, roughly by reordering the arguments. (See Fig. 2.6 on p. 63.) Corresponding mechanisms in some other lexicalized theories are named in Fig. 2.1.

2.1.5 An Explanatory Hierarchy

Thus, a lexicalized theory of syntax describes a language on four levels:

1. At the simplest level, a language is merely a set of strings—an extremely simple and uniform representation.

2. To explain regularities in the above set, it is taken to be generated (under universal combination operations) from a more concise and perhaps finite lexicon of language-specific syntactic substructures.

3. Regularities in the above lexicon are explained, in turn, via language-specific transformations on lexicon entries.

4. Finally, a language's system of transformations is itself licensed by Universal Grammar. We will use this to explain regularities in the set of transformations.

Let us call this the **explanatory hierarchy**. Levels 2 and 3 are the lexicalized versions of Chomsky's surface and deep structure, respectively, and level 4 corresponds to

[2]They attribute the idea first to the syntax of ALGOL 68 (van Wijngaarden, 1969). Wilson (in progress) makes a similar point about Lisp and Scheme, which have an extremely simple context-free surface syntax plus a deeper "transformational" syntax, in the form of code-manipulating macros, that the compiler must also know about.

Chomsky's Universal Grammar. §2.3.2 will show how to introduce probabilities at all levels.

Notice that each successive level results in a more detailed *analysis* of sentences. Level 1 treats sentences as unstructured strings. But level 2 describes each string as the result of at least one derivation, typically tree-structured, from which it is possible to infer a surface semantics: e.g., that a particular `NP` is the subject of `taken`. Level 3 allows a deeper semantics by identifying the subject of (passive) `taken` with the object of (active) `take`. Finally, our weak treatment of level 4 will relate the transformations themselves, explaining `PP`-adjunction to `NP` as reflecting more general tendencies toward nominal modification, `PP`-adjunction, and right-adjunction in the language. (See §1.3.3. Such "patterns in [the set of] metarules" were treated non-statistically by Becker (1994; 2000).) This explanation provides a better analysis of the strings because the semantic or pragmatic effect of a transformation is often influenced by these more general features that describe it.

2.2 Benefits of a Lexicalized Theory

So lexicalized theories are possible. But what are their benefits other than homogeneity? That is, is it really best to store regular (derived) syntactic building blocks alongside irregular ones? And why should these building blocks be lexicalized, i.e., specialized to particular words? There are several arguments:

statistical While the original role of the lexicon was to list only unpredictable information, nearly every entry is unpredictable in a probabilistic grammar. Lexical entries that are regular in form may still be exceptional in their probability. For example, verbs that passivize or drop their objects tend to do so at different rates. To exploit this fact, every lexical entry—whether regular (derived) or irregular—should list a probability that may not be fully predictable.

(The *reasons* for these exceptional probabilities are genuinely extragrammatical. Roland et al. (2000) studied the rate of intransitivization for 64 "single-sense" verbs. Not only did the rate vary from verb to verb, but for 9 of the verbs it varied significantly across three corpora because of the topics being discussed. For example, in balanced corpora, `soften` is transitive 70% of the time (*soften butter*), whereas in financial news it is only 43% transitive, because when *prices soften* there is no obvious agent to mention.)

The use of lexically specific subcategorization probabilities was pioneered in (Schabes, 1992; Jones and Eisner, 1992; Lafferty et al., 1992; Hindle and Rooth, 1993) and investigated further in (Collins and Brooks, 1995). It is now used in virtually all statistical parsing work.

semantic Although lexical semantics and pragmatics fall beyond the scope of this work, they provide an analogous argument. Syntactically regular entries may acquire not only idiosyncratic probabilities, but also idiosyncratic connotations that need to be listed in the lexicon.. For example, `retarded` often has the implicit modifier "mentally" and can have perjorative force, which are not properties derived from the verb `retard`, but rather have accrued from the word's usage history.

paradigmatic We saw above that seemingly regular entries may have idiosyncratic statistics or semantics. The converse also holds: seemingly irregular entries may be governed by subregularities. In DiSciullo and Williams's metaphor (§2.1.2), it is not really true that the prisoners have nothing in common but lawlessness: some of them are doing time for the same crime. One would like to account for both regularities and subregularities by a single mechanism (here, probabilistic transformations; see also the neural-network approach first proposed by Rumelhart and McClelland (1986)).

Examples abound in the morphology of Romance languages, which boast multiple conjugations and declensions as well as subfamilies of irregulars ("*comprendre*: conjugate like *prendre*"). Similarly in English: *ring/rang/rung*, *sing/sang/sung*, *swim/swam/swum*. In the syntactic domain, Levin (1993) lists many subfamilies of English verbs that share unusual syntactic paradigms, such as certain verbs that cover a surface with a mass or plural noun: *load hay (on the truck) / load the truck (with hay)*, *spray paint (on the wall) / spray the wall (with paint)*, similarly *lay*, *wipe*, *paint*, *spread*, etc.

The subregularities are themselves subject to exception: *bring* is not conjugated like *ring* and *sing*, and that the *load/spray* class does not include *lay out* or *bedeck* or even *cover*. Our view of the lexicon is that it is ultimately just *a set of entries related by a number of loose tendencies* (§1.3.3), and that learning the tendencies can help us learn the entries.

computational Simple, efficient, dynamic-programming parsers are possible because the model of generating sentences is so simple: lexical entries combine only by universal operations. The parsing algorithms do not need any special handling for language-specific transformations, in contrast to parsers for transformational grammar or similar non-lexicalized theories (Zwicky et al., 1965; Hobbs and Grishman, 1976; Wanner and Maratsos, 1978; Marcus, 1980; Duffy, 1987; Fong, 1991). Recent improved algorithms are available for lexicalized CFG and dependency grammar (Eisner and Satta, 1999), lexicalized TAG (Eisner and Satta, 2000), combinatory categorial grammar (Vijay-Shanker and Weir, 1990; Eisner, 1996a), ...

Bangalore and Joshi (1999) show that lexicalized grammars also allow efficient parsing heuristics. Given an input sentence s, one can narrow down the grammar to just those lexical entries most likely to be used in the correct parse of s—as predicted from the words in s using a cheaper n-gram or dependency model. More generally, one could use these cheap local predictions to help guide best-first or agenda-based parsing (Caraballo and Charniak, 1998).

Of course, any lexicalized parser needs access to the lexical entries, and it may be costly to use transformations to generate these on the fly as needed. However, they may be cached once generated (Bresnan, 1978)—a lazy version of Aronoff's (1976) "full-entry" theory of the lexicon.

syntactic A lexicalized theory makes an empirical claim about natural-language syntax. It regards a syntax tree as carved up into local domains, each of which is the size of a word (or idiom chunk) and its modifier slots. It limits both regular transformational processes *and* listed exceptions to a domain of this size.

Thus, lexicalized theories are more constrained than early transformational approaches (Chomsky, 1965) or the generic Move-α ("move anything anywhere") transformation of Chomsky (1981). The syntactic effect of a lexical entry tranformation is limited to changing or rearranging the complement and modifier (argument and adjunct) positions that the entry permits for a given headword, and perhaps changing the syntactic type of the resulting phrase or the morphological markings on the headword. This restriction captures the intuition that *language-specific transformations operate over the domain local to a single word:* each transformation is licensed by some word and can only affect the phrases governed by that word.

Note that this locality restriction does not preclude long-distance effects. A local transformation on a lexical entry may turn out to have non-local effects on the surface tree. For example, a transformation may introduce gaps that affect the way the entry can combine with other entries (Fig. 2.5 on p. 63). In general, the theory's universal mechanisms for *combining* lexical entries may include mechanisms such as adjunction (Joshi et al., 1975) or categorial gap-passing (Bresnan and Kaplan, 1982; Gazdar et al., 1985). So long-distance effects such as extraction or question formation arise from the interaction between (1) language-specific, local transformations *within* lexical entries and (2) language-independent, non-local composition *among* lexical entries.

Separately, Bresnan (1982a, p. 21) makes a rule-ordering argument for lexicalized syntax. She observes that `unsold` is not the past participle of `*unsell`. Rather, syntax precedes morphology here: $\texttt{sell}_{\texttt{V}} \stackrel{\text{syntax}}{\Longrightarrow} \texttt{sold}_{\texttt{Vpastpart}} \stackrel{\text{morph.}}{\Longrightarrow} \texttt{sold}_{\texttt{Adj}} \stackrel{\text{morph.}}{\Longrightarrow} \texttt{unsold}_{\texttt{Adj}}$. Assuming that the morphology acts on lexical entries *before* they are assembled into sentences, so must a syntactic operation that precedes morphology.[3]

[3]Alternatives to this assumption are conceivable but may require more powerful mechanisms. One possibility is to treat both morphology and syntax with sentential rather than lexical transformations: $\cdots \Rightarrow$ *goods that are not sold* $\Rightarrow \cdots \Rightarrow$ *unsold goods.* A more recent alternative would be to interleave the operations of transforming and assembling phrases, as in the Minimalist Program (Chomsky, 1995); `sold` could be produced by tranformation before it is inserted as a complement of `un-`. In general, minimalism's (overly?) powerful ability to transform arbitrary subtrees before inserting them allows (among other things) something quite like transformations of lexical entries, only with the arguments already instantiated.

2.3 A Statistical Approach to Lexical Redundancy

2.3.1 The New Idea

Recall the explanatory hierarchy of §2.1.5. The contribution of this thesis is a statistical treatment of levels 3–4. Previous *statistical* lexicalized theories have considered only levels 1–2. The thesis shows how to enrich such theories with lexicon-internal regularities, by allowing them for the first time to capture the lexical transformations or redundancy rules at level 3. The transformations are induced automatically from a sample of (known or hypothesized) lexical entries at level 2, under the guidance of weak universal-grammar preferences at level 4.

Knowing the transformations is important for learning a large lexicon from a sparse sample. If a lexical entry is predictable by transformation, it need not be observed directly in order to learn it.

The following sections sketch the solution, explaining how the statistical approach works out at each level of explanation.

2.3.2 The Probabilistic Framework

While one could attempt purely symbolic learning and application of lexical transformations, it is preferable to use a probabilistic framework. Language learning, like parsing, leaves us flailing in a sea of uncertainty. Probabilities have a role as beacons to guide us to a probably correct shore.

In a probabilistic framework, our linguistic knowledge at every level of the explanatory hierarchy (§2.1.5) must become gradient:

- At level 1, instead of describing strings as in or out of the language, we describe them as more or less probable. (This is not the same as more or less grammatical: see §2.3.7.)

- At level 2, instead of learning that entries are in or out of the lexicon, we learn that they are common, not-so-common, or extremely rare.

- At level 3, the transformations or lexical redundancy rules predict unobserved lexical entries *and their probabilities*. Since the lexicon is specified by a collection of probabilities. the redundancy rules must capture redundancies *among probabilities*: for example, the fact that a typical English transitive verb's passive entry is about $\frac{1}{20}$ as likely as its active entry.[4]

  Hence our transformations at level 3 must themselves be probabilistic. They must derive new entries from old *at particular rates*. To explain the pattern above, one could say that passivization has a rate of $\frac{1}{21}$; then about 1 in every 21 tokens of a typical transitive verb passivizes, and the other 20 remain active.

- Finally, at level 4, by turning Universal Grammar into a probabilistic prior that prefers certain systems of transformations, we can learn the transformations within a statistical framework. In other words, Universal Grammar includes soft preferences (notably preferences for consistency) as well as hard constraints.

To expand on the last point, it may help to point out a strict analogy between statistical grammar learning and statistical parsing. If we let "level $1\frac{1}{2}$" denote the set of trees of the language, then levels 1, $1\frac{1}{2}$, and 2 in parsing correspond to levels 2, 3, and 4 in learning:

- Probabilistic parsing lets us distinguish good parses from bad, by recovering likely level-$1\frac{1}{2}$ derivations of the level-1 strings under the probabilistic grammatical expectations imposed by level 2. This procedure can simultaneously recover likely reconstructions of any uncertain parts of the level-1 string (uncertain owing to speech recognition, say).

- Similarly, probabilistic grammar learning should let us distinguish good generalizations from bad, by recovering likely level-3 derivations of the set of level-2 lexical entries under the probabilistic universal-grammar expectations imposed by level 4. This procedure can simultaneously recover likely reconstructions of any uncertain parts of the level-2 lexicon (uncertain owing to sparse, noisy, or indirect data). Indeed, fleshing out or smoothing the lexicon in this way is a primary goal.

[4] Capturing this generalization helps us estimate the probabilities of any particular verb's passive entry, taking into account both the typical $\frac{1}{20}$ pattern and any evidence we have for the particular verb.

We now consider in more detail how statistics benefit levels 1, 2, 3, and 4 of the explanatory hierarchy.

2.3.3 Level 1: Strings

In formal language theory, a language is a set of strings. In the probabilistic approach, it is a probability distribution over strings.

Most strings have very low probability, since a human language has very few fixed utterances. Still, even among strings that have not yet been uttered in human history, some have lower probability than others, in that they contain many unlikely words or constructions. An extreme example (now rendered obsolete by frequent quotation) is Chomsky's contrast between *Colorless green ideas sleep furiously* and *Furiously sleep ideas green colorless.* Abney (1996) shows that many apparent "word salad" examples like the latter are actually grammatical—just very, very improbable, in a way that apparently interferes with human language comprehension. (See §2.3.7 for more on ungrammaticality.)

Relative string probabilities are useful for applications where it is necessary to choose one string over another, such as speech recognition or machine translation.

But level 1 has little practical use in applications such as parsing, where the string is a fixed input, and one cares only about the relative probabilities of different derivations of it. This takes us to the next level of the explanatory hierarchy. We will model the level-1 string probabilities by saying that the strings arise through a derivational process, such as the composition of randomly chosen lexical entries. Some strings are more likely than others to arise through this process.[5]

2.3.4 Level 2: The Stochastic Lexicon

The practical benefit of probabilities at level 2—a stochastic lexicon—has been well established over the past several years (see §1.2.1.1). In parsing, a parse can be deemed likely to the extent that it is composed of likely lexical entries. A probabilistic parser then returns the most likely parse whose yield (i.e., fringe) matches the input sentence.

[5]There may be multiple derivations that yield the same string α, in which case the probability of α is the total probability of all these derivations.

Formally the parse's probability is defined as some kind of product of the probabilities of its component lexical entries. This ensures, for example, that the parse will have zero probability if any of its components does.

The exact form of the product depends on the lexicalized theory of grammar we are using. It is often motivated by a generative model of parse structures. For example, in a lexicalized context-free approach, it is straightforward to use a stochastic lexicon to generate random parse trees from the top down. One randomly chooses a lexical entry (i.e., a lexicalized context-free rule) that specifies the head of the sentence and its argument/adjunct slots, and then chooses entries to fill those slots, continuing recursively until no empty slots are left. The probability of having chosen the resulting tree is the product of the (conditional) probabilities of all the choices of lexical entries made along the way.

This probabilistic context-free (PCFG) model of tree probabilities is detailed in §5.3.3. It is used in the experiments of this thesis and in much other work that also uses the Penn Treebank as data. But the same approach applies directly to any grammar formalism in Fig. 2.1 that has tree-structured derivations, including non-context-free formalisms such as CCG and TAG.

Some grammatical formalisms such as LFG, HPSG, and link grammar have more complicated derivations that are not tree-structured, because mechanisms such as unification allow for cyclic dependencies. However, they can still be accomodated within the general approach of this thesis:

- One option is to use a more complicated but still sequential model of generation. For example, each entry is chosen conditionally to fit the requirements of *two* previous entries rather than one (Lafferty et al., 1992).

- A more general solution (Riezler et al., 2000) is to assign non-negative "weights" to the entries in the lexicon. One can then directly define the probability of a parse as proportional to the product of the weights of its component lexical entries.[6]

  This is known as a log-linear model, since the log-probability of a parse is a linear function of the log-weights of various local features. Scoring parses in this simple

[6] And perhaps also the weights of other features of the parse that cross lexical entries, as Johnson et al. (1999) propose.

way is an old idea. However, the feature weights were once typically chosen by hand or by some unspecified psychological procedure (Schubert, 1984). Interpreting linear "scores" of parses as log-probabilities makes it possible to estimate appropriate weights automatically from a random sample of parses.

The weights may not have an interpretation as probabilities, and are harder to estimate from parses. But the transformational approach of Chapter 3 could still be used to smooth them, by redistributing weight rather than probability from one lexical entry to another.

2.3.5 Level 3: Stochastic Transformations

Now for a brief sketch of how transformations apply to entries in the stochastic lexicon. Full details are given in Chapter 3 (which uses a different notation).

A transformation is a partial function $t : E \to E$ for some set of objects E.[7] Thus, it can apply to certain objects and turn them into other objects.

For our purposes, E is a set of lexical entries allowed by some universal syntactic theory, and transformations are rules that can derive new lexical entries from old ones. (Examples are given in §2.4.2.) Transformations in other domains include general rewrite rules on strings (Chomsky, 1959), production rules on declarative memories in ACT-R (Anderson, 1993), operators on states in SOAR (Newell, 1990),[8] and the action of an input symbol on the state of a deterministic finite-state automaton.

The transformation system specifies a set of allowable transformations t_i, including a special transformation t_{Halt}. A given object $e \in E$ may be in the domain of several transformations.

Imagine a process that begins with the special object $\textsc{Start} \in E$, and passes it through an arbitrary series of transformations ending with t_{Halt}. The object to which t_{Halt} was applied is considered to be **derivable**. Formally, an object e is derivable under the grammar, with derivation $\langle t_1, \ldots t_n, t_{\text{Halt}} \rangle$, if it can be expressed as $e = t_n \circ t_{n-1} \circ \cdots \circ t_1(\textsc{Start})$ and

[7] Called Events in Chapter 3.

[8] In addition to operators, SOAR also has productions, which fire to indicate that the state is more or less amenable to the application of this or that operator. While SOAR's operators correspond to our transformations, its productions correspond to the features we assign to a transformation, which determine its probability (§3.2). (The assignment of features to transformations is the level-4 structure of §2.1.5.)

also $t_{\mathrm{Halt}}(e)$ is defined. (Usually START itself will not be in the domain of t_{Halt}, so will not be derivable.)

Now let us make this a *stochastic* process—one that randomly chooses the transformation to apply at each step. This will tell us not only which objects in E (e.g., lexical entries) are derivable, but the *probabilities* of deriving them. The probability of transforming e by t depends only on e itself, not on e's derivational history.

For each transformation t and object e, the transformation system must specify the probability $\Pr(t \mid e)$ that t will be the transformation chosen apply to e. This must be a true probability distribution: $\sum_t \Pr(t \mid e) = 1$. Of course $\Pr(t \mid e)$ should be 0 if e is not in the domain of t. The classical notions of **optional transformation** and **obligatory transformation** correspond to the cases $0 < \Pr(t \mid e) < 1$ and $\Pr(t \mid e) = 1$, respectively.

Now the probability of deriving e with derivation $\langle t_1, \ldots t_n, t_{\mathrm{Halt}} \rangle$ may be written as

$$\Pr(t_1|\textsc{Start}) \cdot \Pr(t_2 \mid t_1(\textsc{Start})) \cdot \Pr(t_3 \mid t_2(t_1(\textsc{Start}))) \cdots \Pr(t_{\mathrm{Halt}} \mid e) \tag{2.1}$$

$\Pr(e)$ denotes the total probability of all such derivations of e. This is the level-2 probability of the lexical entry e.

The idea is that for any word w of the language, its lexical entries are derived from START (and hence at intermediate steps from one another) by such a transformation system. Another word w' will use another transformation system—identical to the first except for a small number of stipulated changes to the probabilities, as described in the next section.

A "shallow" learner would learn only the probabilities $\Pr(e)$ of the lexical entries. In the terminology of §2.1.5, this is level-2 structure. However, shallow learning is hard in the presence of sparse data. It is hard to determine $\Pr(e)$ if e is too rare to have been observed. The system described here is therefore a "deep" learner. It tries to infer the transformation probabilities $\Pr(t \mid e)$, in an effort to get better estimates of $\Pr(e)$. This is level-3 structure: not just "what" is in the lexicon but "why."

2.3.6 Level 4: Suffering Lexical Idiosyncrasy

In transformational accounts of grammar, there is a tension between regularity and idiosyncrasy. Generic transformations cannot perfectly predict every entry in the lexicon.

They do not eliminate the need for lexically specific information. Words differ:

- A verb will tend to start the transformational process with different entries than a noun (or a different verb). These entries are "listed" for the verb. Some of them have higher probability than others and so are "listed more strongly."

- Other entries must be "delisted" since derivations can be idiosyncratically blocked: `let` in the sense of "allow" does not have a passive form (*We let her play* $\not\Rightarrow$ **She was let (to) play*).

- Some transformations such as Object Drop apply sporadically to some lexical items but not others (see footnote 2 on p. 8). As noted in §2.2, they are the syntactic analogue to morphological subregularities.

- Finally, even a transformation that applies regularly may apply at different rates to different words (e.g., see footnote 2 on p. 8). So even derived lexical entries must be weakly "listed" or "delisted" as occurring more or less frequently than the norm.

We will therefore store a positive or negative amount of "listedness" with every lexical entry. Positive listedness inflates an entry's probability over what would be predicted by transformation; negative listedness deflates it.[9] Transformations apply at the usual rate to a (de)listed entry, so transformed versions of it also gain or lose probability.

Listedness comes at a cost, however. §3.7.2 will use a simple notion of universal grammar (i.e., level 4) that acts much like a human linguist. It has a preference for general principles over narrowly tailored exceptions. It consists in a prior probability distribution—a universal belief that the lexicon being learned is *a priori* more likely to be simple and regular. The more listedness in a hypothesized lexicon, the lower the lexicon's prior probability, and the harder the grammar learning algorithm will try to avoid it. Just like a linguist, the learner tries to explain the data with as little stipulation as possible.

While this section focuses on lexically specific exceptions, the prior of §3.5 actually acts to discourage any narrow stipulation that disrupts a natural class of transformations. The approach was laid out in §1.3.3.

[9] Two different formalizations will be offered in §3.6.1 and §3.9. The options are to adjust the entry's probability by adjusting the transformations or through a separate mechanism.

It is of course an old notion that complicated grammars or lexicons with many listed exceptions have high cost and should be avoided by a learning algorithm (McNeill, 1970). But it is worth emphasizing that in probabilistic lexicons, listedness is a matter of degree. Following the usual dictum that a lexicon need not list what it can derive, we need to list only the *discrepancy* between the observed and predicted probabilities. To gradually reduce the amount of listedness for a lexical entry, we can increase the probability of deriving it.

In the specific model to be presented in §3.6.1 (or the alternative in §3.9), the effect of a given "amount" of listedness is multiplicative. This is a deliberate design decision: the prior assigns cost to the *ratio* between observed and predicted probabilities.[10] The consequence is that it is not arbitrary which entries tend to be listed. The learning algorithm finds it *a priori* more plausible to list high probabilities for lexical entries that are comparatively easy to derive by transformation.

For example, it costs the lexicon much less in prior probability to inflate an entry's probability from predicted 0.31 to observed 0.51 (a ratio of < 2) than to inflate it from 0.01 or 0.0001 to 0.21 (a ratio of 21 or 2100). This makes it relatively inexpensive to add 0.2 probability mass to a lexical entry that is already predicted to be common. It also means that among entries that are not predicted to be common, the ones that are at least transformationally plausible (predicted probability 0.01) are much cheaper to list as probable than those that can be derived from START only by long or dubious transformational sequences (predicted probability 0.0001). As an extreme case, it is impossible to list a high probability for an underivable, 0-probability entry: no amount of listedness can rescue it.

The multiplicative cost model has several positive consequences:

- Lexicons that seem intuitively similar are close in parameter space and close in cost. The intuition is that 0.31 and 0.51 are indeed more similar than 0.01 and 0.21. In absolute terms, common frames vary more in probability from word to word than uncommon frames do.

- When a lexical entry is transformationally plausible, then we are willing to list it

[10]The cost is assigned roughly as follows: a ratio of r will decrease the log of the prior probability by a constant times $(\ln r)^2$. For example, $(\ln 2)^2$ to double the predicted probability, $(-\ln 2)^2$ to halve it, and $(\ln 1)^2 = 0$ to leave it alone.

further on the strength of not much additional evidence, since the cost is low for increasing its probability to a given level. (See §3.1.3 for a Bayesian formulation.)

Consider especially the case of a learner parsing a difficult sentence (§1.2.4), who is forced to posit entry e_1 or entry e_2, neither of which has ever been observed before. Suppose the two parses containing these entries are equally likely overall. Then an EM learner will consider the sentence to have provided half an observation of each entry. But these equally weighted observations will increase $\Pr(e_1)$ by more than $\Pr(e_2)$, if e_1 originally had higher probability *in isolation*—i.e., if e_1 is easy to derive transformationally. This reflects the learner's prior bias toward a transformationally regular lexicon.[11]

- An old linguistic principle of Structure Preservation (Emonds, 1976) says that transformations tend to output structures that are already common in the language. One manifestation is that *listed* entries tend to be the same as transformational outputs. The multiplicative cost model makes it cheap to list just those entries.[12]

- Transformational explanations are favored. The reason (explained more carefully in §3.7.2.3) is that the multiplicative cost model—unlike an additive one—results in an economy of scale for listedness. Suppose several correlated entries all need to be inflated (listed). It is costly to inflate all their probabilities by 50% separately. But inflating an entry's probability by 50% has the same cost regardless of whether that probability is large or small. So it is most economical to suppose that all the probability for these entries starts out pooled at one of them—where it can be inflated "in a batch" by 50%—and is then redistributed to the others by transformation. This savings motivates the learner to find transformations that connect these corelated entries.

[11] In other words, the greater transformational plausibility of e_1 increases not just the probability of the parse containing e_1, so that it ties with the otherwise superior competing parse, but also the listability of e_1 in response to such parses. It is only the second effect that arises from the multiplicative cost model.

[12] The full effects of Structure Preservation can be captured more completely and directly by other means (§3.7.2.5).

2.3.7 Ungrammaticality and the Explanatory Hierarchy

To close this section on adding probabilities to linguistics, a remark on ungrammaticality is in order. Although §2.3.2 substituted string probability for string grammaticality, the former—however deeply it is modeled—is not really a *replacement* for the latter. It is true that people's grammaticality judgments are often gradient, but they do not seem to be judgments of overall probability. It is an old observation that a long sentence with many rare or nonsense words, or many rare constructions, may be judged grammatical although it is extremely improbable. Conversely, a short sentence may be done in by a single bad construction.

Modeling grammaticality judgments accurately would probably require a theory of human sentence processing. But I suspect that such judgments depend more on the *local* grammaticality of the level-2 structure that we recover for a string. If the best parse we can manage to humanly recover has any "bad" lexical entries, then our judgment of the sentence stems from our judgment of the *worst* of these entries.

Even our judgment of a single lexical entry is not a simple matter of its probability, since the grammatical sentence *Jack and Jill sprossified the cat* contains at least one very low-probability lexical entry. The judgment might however depend on the (estimated) *conditional* probability of the entry given its headword. This should be fairly high for `sprossified`—above the "grammaticality threshold"—since new words are quite often transitive verbs.

With the statistical treatment in this thesis of levels 3–4, one could even go one step further toward the standard linguistic notion of a legal grammatical derivation: namely, one in which all moves are licensed. Suppose that we humans do not merely use a precompiled lexicon, but have real-time access to our level-3 derivations of the lexical entries. (For example, when parsing a difficult sentence, we try to derive new entries on the fly as necessary.) In particular, suppose we can reconstruct the best way D to obtain a given entry by derivation from an entry that is listed (or at least moderately likely given its headword). Perhaps our judgment of the derived entry stems from the probability of derivation D, or of the lowest-probability transformation in derivation D. Then one poor transformation is enough to torpedo a lexical entry, and hence the whole parse in which it appears.

This is a very traditional approach: all moves in a derivation must be independently acceptable. What probabilities contribute is a gradient notion of grammaticality. The prediction is that a derivation is unacceptable if its worst transformation is unacceptable—but marginally acceptable if its worst transformation has just enough probability to be "vaguely plausible."

An example of a "vaguely plausible" transformation is one that is being applied in unusual circumstances:

- It is being applied to the wrong kind of lexical entry. In this case it is really a new transformation that resembles an existing one: they select for slightly different properties. But even if the new transformation is unattested, it will derive some plausibility by having features in common with the existing one. (See §3.2 and §3.5 for how this is arranged.)

- It is being applied to the wrong lexical entry. Some transformations (e.g., dropping a noun's determiner, or changing its gender feature) are lexically selective: they are observed to apply only to some arbitrary selection of lexical entries (e.g., the nouns that can be used as mass nouns).

  The entries that result from appropriate applications of a lexically selective transformation have to be explicitly listed. A conventional grammar would therefore not include the transformation at all. But in the statistical framework of §2.3.6, the learner will prefer to assign the transformation some non-negligible probability, since we have seen that this reduces the *cost* of listing its results (see §2.3.6). So the transformation is to some small degree part of the grammar, and it is bad but not crashingly bad to apply it to other lexical entries. Indeed, the probability assigned to it increases with the evidence for it: so if many nouns are observed to also allow mass-noun uses, then mass is not a particularly distinctive feature in the language, and coercing other count nouns to mass nouns is marginally acceptable: ?*Less desk means less computer.*

To move beyond armchair linguistics on this question, it would be interesting to fit a transformation model to data, as in Chapter 6 then try to use the above approach to predict human grammaticality judgments.

2.4 A Flattened Lexicalized Context-Free Approach

The reader may find it useful to have a specific grammatical theory in mind when reading the next chapters. Let us now, therefore, describe the choices that will be used in the statistical model sketched in §3.7.2, the syntactic framework of Chapter 5, and the experiments of Chapter 6—as well as their linguistic shortcomings (and how those shortcomings might be remedied).

What do the lexical entries look like? Transformational smoothing does not much care. The stochastic lexicon that it estimates is just a probability distribution $\Pr(\cdot)$ over all possible lexical entries. These could really be any objects—a fact exploited by §7.2, which suggests other useful probability distributions that could be transformationally smoothed. Our approach applies so long as we start with a set of possible entries and possible transformations on them, plus some observations of actual entries.

To put the approach into practice, however, we need to take the entries and transformations from some specific theory of grammar. For purposes of the experiments, we will make simple choices.

2.4.1 Defining Lexical Entries as "Flat" Rules

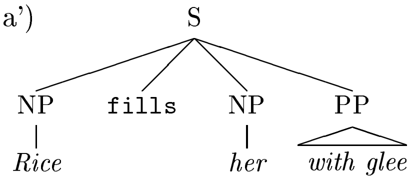
The experiments in this thesis derive their data from the Penn Treebank (Marcus et al., 1993), a collection of parses in a context-free style. So following much previous work (§2.5), the experiments take lexical entries to be lexicalized context-free rules as shown in Fig. 2.2a.[13]

But a choice that was not forced by the Treebank was to make the rules "flat." For example, a lexical entry for a verb describes how to construct a maximal projection of a verb—usually a sentence—from the verb together with *all* its arguments and adjuncts.

[13]To extract *lexicalized* rules from the Treebank, it is necessary to identify constituent heads. See §6.2 for the method used.

(a) $\texttt{fills: S} \rightarrow \texttt{NP fills NP PP}$

(a')



(b) $\texttt{fills: S} \rightarrow \texttt{NP } \underline{\texttt{VP}}$
$\texttt{fills: VP} \rightarrow \underline{\texttt{VP}}\texttt{ PP}$
$\texttt{fills: VP} \rightarrow \underline{\texttt{V}}\texttt{ NP}$
$\texttt{fills: V} \rightarrow$ ——

(c) $\texttt{fills: S} \rightarrow \texttt{NP [}_{\texttt{VP}}\texttt{ [}_{\texttt{VP}}\texttt{ [}_{\texttt{V}}$ —— $\texttt{] NP] PP]}$

(b'),(c')

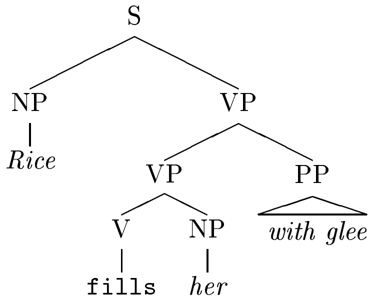


Figure 2.2: (a) Flat and (b) non-flat lexical entries for generating SVOX, and (c) a flat entry that generates a non-flat structure (as in LFG, TAG, and other lexicalized theories). In all cases, the probability of a rewrite rule is conditioned on the knowledge that the headword of the phrase will be `fills`. The resulting tree structures are shown as (a'),(b'),(c').

There are several reasons behind the decision to use flat rules. Most crucially, a flat lexical entry corresponds to the *local domain of a headword*—the word together with all its semantic arguments and modifiers. As discussed in §2.2, this is the smallest appropriate domain over which to *define transformations* and also to *list exceptions*.

One could also allow larger lexical entries, as in the treatment of idiom chunks in LTAG (Abeillé, 1988) or TSG (Bod and Scha, 1996). The arguments below are only against using *smaller* entries for purposes of transformational smoothing. Thus, the kind of straw man being rejected (Fig. 2.2b) is that the grammar builds an SVO structure from at least two separate entries or rules—one for the `VP`, one for the full sentence—and that these entries have independent probabilities given their common head.

locus of transformations If instead we were to adopt the traditional phrase-structure approach of Fig. 2.2b, in which `fills` has a separate lexical entry for `VP`, then many natural transformations could not be defined over single lexical entries. Passivization, unaccusative movement, and heavy-shift (past the `PP`) all move the `NP` object out of its

VP. Handling this in the statistical framework would require the use of long-distance-movement mechanisms (§2.4.3). To make long-distance movement preserve bilexical dependencies (see §5.3.3.2) has extra computational cost. With flat entries, however, these local transformations only move the object within the subcategorization frame.

locus of exceptions Since lexical entries in the same parse are assumed to be statistically independent of one another (§5.3.3), capturing dependencies among argument positions requires putting them in the same entry. Johnson (1999) shows that flattening trees aids PCFG parsing for exactly this reason, and §6.7.1 will obtain another such empirical result.

The canonical example is PP-adjunction. In the traditional structure of Fig. 2.2b, each PP adjoins to the VP at a new level. Hence in the parsers of Collins and Charniak, each instance of the context-free rule VP $\rightarrow$ VP PP is independent of the others and equally likely. In particular, adjoining a second PP to a given verb has the same probability as adjoining the first PP. This is empirically false: the number of PPs attached to a verb is not exponentially distributed.

By contrast, our use of flat rules (lexical entries) allows separate 2-PP, 1-PP, and 0-PP rules for each word. Their probabilities need not be related, and *can* be learned separately given enough evidence. (For example, the 2-PP rule is quite likely for verbs of motion, which often specify both a "from" and a "to.") But they *need* not be learned separately. If there is not enough evidence, then transformational smoothing assumes each is derived from the next by a PP-adjunction transformation. The model can even learn that the transformation's rate varies according to the number of PPs already in the sentence. The rate of a PCFG adjunction rule VP $\rightarrow$ VP PP is not allowed to vary in this way.

other linguistic arguments Recent work in linguistics (especially HPSG) has argued for a uniform treatment of complements and adjuncts, including lexical rules to insert optional adjuncts. (These rules are analogous to our Insert transformation in §2.4.2 below.)

The grounds—cogently reviewed by Przepiórkowski (1999, chapter 9)—include facts about obliqueness and agreement (Miller, 1992), approaches that flatten verb clusters (van Noord and Bouma, 1994), the fact that complements and adjuncts (and subjects) pattern together with respect to selection and extraction (Bouma et al., 2001), an ordering effect (shown by semantics) that adjunction can optionally precede passivization or causativization (McConnell-Ginet, 1982; Iida et al., 1994), and finally, theoretical parsimony given the weakness of evidence for a syntactic distinction (Przepiórkowski, 1999).

availability of data Flat structures are easier for human annotators to produce, and easy to recover from almost any syntactic annotation. In particular, they do not have to make the traditional argument-adjunct distinction, which has many borderline cases. All dependents attach at the same level. This is why the Penn Treebank uses some flat structures (Marcus et al., 1993, p. 323), and part of the reason that several recent European annotation projects have used dependency grammars. The experiments in Chapter 6 take their data from the Penn Treebank.

Several other attempts to learn subcategorization frames, such as (Li and Abe, 1996; Carroll and Rooth, 1998), have also included both adjuncts and arguments on an equal footing.

lexicalization One may prefer lexicalized theories of syntax on principle (see §2.2 and Fig. 2.1). In such theories, a single word token projects a single lexical entry and so has only one place to specify all its syntactic requirements, such as argument slots. This rules out the articulated derivational structure in Fig. 2.2b, where `fills` heads separate `V`, `VP`, `VP`, and `S` entries, in favor of a flat one.

A lexicalized theory need not be quite as flat as Fig. 2.2a. Lexicalized tree-adjoining grammar (LTAG) (Joshi and Schabes, 1991) and combinatory categorial grammar (CCG) (Steedman, 1990) adopt a convention whereby optional modifiers such as adjuncts select for the phrases that they adjoin to, rather than other way around.[14] Then the `PP` *with glee* would not be part of the entry for `fills`. Rather, the preposition `with` would have its own entry (roughly, `VP` $\rightarrow$ `VP` —— `NP`) that combined with

[14] An advantage of this convention is that the entry for a given Romance adjective can directly specify whether that adjective selects for a noun on its right or its left.

`fills`'s entry $S \to NP$ —— NP[15] by a special "adjunction" operation not available in context-free grammar. It is possible to attach sensible statistics to this operation (Schabes, 1992).

The lexical entries of LTAG or CCG still require lexical redundancy rules, which could be treated by statistical transformations as in this thesis. But for statistical reasons, we prefer not to treat adjuncts as heads as they do, preferring instead to insert optional modifiers into an entry by transformation (as first proposed by Miller (1992)).[16] As discussed earlier in this section, it is too strong an independence assumption to assume that adjuncts attach to a phrase independently of one another. In any case, the LTAG/CCG approach could not have completely unflattened our structures anyway: the category of a transitive verb must still specify both its arguments, subject and object.[17]

generality Finally, nothing appears to be sacrificed by the use of flat entries. This remark requires some explanation.

There is of course much linguistic evidence that a maximal projection can have internal structure. Internal units such as VP and $\bar{N}$ may be targeted by transformations, semantically modified by other internal units such as adverbs, and indexed for anaphoric reference. In fact, the internal structure may be recursive[18] (Hornstein and Lightfoot, 1981).

Fortunately, this internal structure can be marked in the entry itself, as in Fig. 2.2c.

[15]Or actually $S \to NP\ [_{VP}\ [_{V}$ —— $]\ NP\ ]$: it must mark an internal VP node so that `with`'s VP entry knows where it can adjoin. See below.

[16]The same proposal was made by Bouma and van Noord (1994) on empirical grounds. In Dutch, an SOV language, any adverb can left-adjoin to any verb, regardless of the verb's nonterminal category (transitive, intransitive, S-complement, etc.). This particular kind of polymorphism in the adverb is easy to describe with mechanisms in CCG and TAG. But in plain categorial grammar (CG) it is problematic, leading Bouma and van Noord, who were working in CG, to reject the idea that adjuncts select for their targets. Instead they adopted an "adjuncts as arguments" approach, positing a transformation that essentially inserts an adverb anywhere into a flat verbal frame, just like our Insert transformation in §2.4.2.

[17]Heads with multiple arguments cannot reasonably be eliminated altogether. If no word in a sentence could have more than one dependent, then the dependency structure or derivation tree of the sentence would be a non-branching tree (i.e., a straight line). Equivalently, each nonterminal node in the phrase-structure tree would have to branch into one terminal and one nonterminal child. This representation is not rich enough to describe natural language.

[18]Although always left- or right-recursive, as far as I am aware, so that it can be described and modified by finite-state methods.

(See also Fig. 6.7 on p. 199.) It is even possible to build up such a structured entry by a series of (perhaps obligatory) transformational steps that correspond to the rewrites of Fig. 2.2b. The traditional unflattened tree can be fully recovered from the complete entry. So on this interpretation, a lexical entry is a tree fragment that represents the high-level structure of a maximal phrase (cf. Tree Substitution Grammar (Bod and Scha, 1996)).

The advantage of marking internal structure is that transformations might be sensitive to it in some way. This is a subject for future experiments; the transformation model evaluated in Chapter 6 just takes lexical entries to follow the simple structure Fig. 2.2a.[19]

One particularly interesting and useful kind of transformation on Fig. 2.2c is the adjunction of other (non-lexicalized) subtrees into the structure, in the sense of tree-adjoining grammar (TAG). Thus, Fig. 2.2c itself might be taken to have been derived from $\mathtt{S} \rightarrow \mathtt{NP}\ [_{\mathtt{VP}}\ [_{\mathtt{V}}\ \text{——}\ ]\ \mathtt{NP}\ ]$ by adjoining $\mathtt{VP} \rightarrow \underline{\mathtt{VP}}\ \mathtt{PP}$. (On a subsequent step, the `PP` would then select a head **`with`** and expand with the conventional lexical entry $\mathtt{PP} \rightarrow \mathbf{\mathtt{with}}\ \mathtt{NP}$.

This **adjunction-as-transformation** approach differs from the conventional LTAG or CCG approach described above, in that its result is a bona fide lexical entry with full privileges. In particular, Fig. 2.2c may have an idiosyncratic probability that can be learned, and can also be subjected to further transformations (heavy-shift, standard question formation with or without pied-piping, etc.). So the approach retains the benefits of flat rules that were first mentioned in this section: that they are the locus of exceptions and the locus of transformations.

It has often been observed (Carroll and Rooth, 1998) that the distinction between arguments and adjuncts is not particularly useful for parsing. A verb selects for both to some degree, and both need to be modeled.

[19] Chapter 6 does evaluate some *non*-transformational methods that use Fig. 2.2b. (This turns out to hurt the performance, compared to versions that versions that use the flat Fig. 2.2a.)

Insert fill a semantic role	S $\rightarrow$ NP see NP *I see you*	$\Rightarrow$	S $\rightarrow$ NP see NP PP *I see you with my own eyes*
Delete suppress a role	S $\rightarrow$ NP see NP *I see you*	$\Rightarrow$	S $\rightarrow$ NP see *I see*
Insert type-shift a role	S $\rightarrow$ NP see NP *I see you*	$\Rightarrow$	S $\rightarrow$ NP see SBAR *I see that it's love*
Swap heavy-shift, etc.	S $\rightarrow$ NP see SBAR PP *I see that it's love with my own eyes*	$\Rightarrow$	S $\rightarrow$ NP see PP SBAR *I see with my own eyes that it's love*

Figure 2.3: Edit transformations.

2.4.2 Defining Transformations as String Edits

The flat lexical entries of Fig. 2.2a can be regarded as strings. The transformations allowed in the experiments correspond to single edit operations (Levenshtein, 1966) on the right-hand side (RHS)[20] of such strings:

- Insert a single nonterminal into the RHS
- Delete a single nonterminal from the RHS
- Substitute a single nonterminal for another in the RHS
- Swap two adjacent elements in the RHS
  (one may be a terminal, namely the headword of the entry)

Their effects are illustrated in Fig. 2.3.

These transformations were selected by manually examining the training data drawn from the Treebank (see §6.2 for details of data preparation). Presumably, they are common in language because they manipulate the argument (or adjunct) structure of a headword in useful ways: adding or deleting arguments, changing their type, or changing their order.

Indeed, Carpenter (1991) remarks:

> The form of lexical rules that we propose to add to the basic categorial system are what Keenan and Faltz (1985) have termed **valency affecting operations**. These operations allow the permutation, addition, or subtraction of

[20] That is, the substring written to the right of $\rightarrow$, consisting of a terminal symbol and 0 or more nonterminals.

> complements and the modification of the head or functor category. ...The same general lexical rule format has been proposed by virtually everyone considering the lexicon from a categorial perspective ... [and by] recent work in HPSG, which admits lexical rules that do the same work as the ones employed here.

Some authors have, like us, allowed these operations to act on adjuncts as well as complements: see in §2.4.1 the material on "other linguistic arguments" and footnote 16.

The edit transformations are nicely "complete" in that any change to the RHS of a rule can be carried out by some sequence of single edits.[21] The flip side is that the generative power of the system is very high: Carpenter (1991) shows that any recursively enumerable set of lexical entries can be generated from a finite starting lexicon using a finite set of edit-like transformations on unbounded flat frames.

2.4.2.1 Two Empirical Sanity Checks on These Transformations

Table 2.1 illustrates just how common the simple edit transformations are in training data (§6.2), among the relatively frequent "frames" (as defined in §5.1). These frequent frames will provide most of the evidence from which the smoothing model learns the probabilities of transformations.

How about rare frames, whose probabilities are most in need of smoothing?[22] Are the same transformations that we can learn from frequent cases appropriate for predicting the rare cases? The very rarity of these frames makes it impossible to create a table like Table 2.1.

However, rare frames can be measured in the aggregate, and the result suggests that the same kinds of transformations are indeed useful—perhaps even more useful—in predicting them. Let us consider the set S of 2,809,545 possible lexical entries that stand at edit distance 1 from the set of entries observed in training data. That is, an lexical entry is in

[21] When such a sequence is common, however, it is useful to package it up as a new transformation that gets its own probability. This is why we do not simply get by with the complete set {Insert, Delete}: it is so common to follow a deletion with a nearby insertion that we introduce Substitute and Swap, rather than accept equation (2.1)'s assumption that the deletion and insertion are independent. See §2.4.3 for remarks on even more powerful transformations.

[22] More precisely, what we smooth is the probabilities of lexical entries that use these frames.

MI	α	β	MI	α	β
9.01	[NP — ADJP-PRD]	[NP — RB ADJP-PRD]	2.04	[TO — NP]	[NP — S]
8.65	[NP — ADJP-PRD]	[NP — PP-LOC-PRD]	1.99	[TO — NP]	[NP — NP]
8.01	[NP — ADJP-PRD]	[NP — NP-PRD]	1.69	[TO — NP]	[NP — NP .]
7.69	[NP — ADJP-PRD]	[NP — ADJP-PRD .]	1.68	[TO — NP]	[NP — NP PP .]
8.49	[NP — NP-PRD]	[NP — NP-PRD .]	1.03	[TO — NP]	[— NP]
7.91	[NP — NP-PRD]	[NP — ADJP-PRD .]	5.54	[TO — NP PP]	[NP TO — NP]
7.01	[NP — NP-PRD]	[NP — ADJP-PRD]	5.25	[TO — NP PP]	[NP MD — NP .]
8.45	[NP — ADJP-PRD .]	[NP — PP-LOC-PRD]	4.67	[TO — NP PP]	[NP MD — NP]
8.30	[NP — ADJP-PRD .]	[NP — NP-PRD .]	4.62	[TO — NP PP]	[TO —]
8.04	[NP — ADJP-PRD .]	[NP — NP-PRD]	3.19	[TO — NP PP]	[TO — NP]
7.01	[NP — ADJP-PRD .]	[NP — ADJP-PRD]	2.05	[TO — NP PP]	[— NP]
7.01	[NP — SBAR]	[NP — SBAR . ”]	5.08	[— NP]	[ADVP-TMP — NP]
4.75	[NP — SBAR]	[NP — SBAR .]	4.86	[— NP]	[ADVP — NP]
6.94	[NP — SBAR .]	[“ NP — SBAR .]	4.53	[— NP]	[— NP PP-LOC]
5.94	[NP — SBAR .]	[NP — SBAR . ”]	3.50	[— NP]	[— NP PP]
5.90	[NP — SBAR .]	[S , NP — .]	3.17	[— NP]	[— S]
5.82	[NP — SBAR .]	[NP ADVP — SBAR .]	2.28	[— NP]	[NP — NP]
4.68	[NP — SBAR .]	[— SBAR]	1.89	[— NP]	[NP — NP .]
4.50	[NP — SBAR .]	[NP — SBAR]	4.89	[NP — NP .]	[NP ADVP-TMP — NP .]
3.23	[NP — SBAR .]	[NP — S .]	4.57	[NP — NP .]	[NP ADVP — NP .]
2.07	[NP — SBAR .]	[NP —]	4.51	[NP — NP .]	[NP — NP PP-TMP]
1.91	[NP — SBAR .]	[NP — NP .]	3.35	[NP — NP .]	[NP — S .]
1.63	[NP — SBAR .]	[NP — NP]	2.99	[NP — NP .]	[NP — NP]
6.13	[TO — NP]	[TO — NP SBAR-TMP]	2.96	[NP — NP .]	[NP — NP PP .]
5.72	[TO — NP]	[TO — NP PP PP]	2.25	[NP — NP .]	[— NP PP]
5.36	[TO — NP]	[NP MD RB — NP]	2.20	[NP — NP .]	[— NP]
5.16	[TO — NP]	[TO — NP PP PP-TMP]	4.82	[NP — S .]	[— S]
5.11	[TO — NP]	[TO — NP ADVP]	4.58	[NP — S .]	[NP — S]
4.85	[TO — NP]	[TO — NP PP-LOC]	3.30	[NP — S .]	[NP —]
4.84	[TO — NP]	[MD — NP]	2.93	[NP — S .]	[NP — NP .]
4.49	[TO — NP]	[NP TO — NP]	2.28	[NP — S .]	[NP — NP]
4.36	[TO — NP]	[NP MD — S]	4.76	[TO — S]	[— S]
4.36	[TO — NP]	[NP TO — NP PP]	4.17	[TO — S]	[TO — NP PP]
4.26	[TO — NP]	[NP MD — NP PP]	2.77	[TO — S]	[TO — NP]
4.26	[TO — NP]	[TO — NP PP-TMP]	4.75	[S , NP — .]	[NP — SBAR .]
4.21	[TO — NP]	[TO — PRT NP]	4.52	[NP — S]	[NP — S .]
4.20	[TO — NP]	[NP MD — NP]	4.27	[NP — S]	[— S]
3.99	[TO — NP]	[TO — NP PP]	3.36	[NP — S]	[NP —]
3.69	[TO — NP]	[NP MD — NP .]	2.66	[NP — S]	[NP — NP .]
3.60	[TO — NP]	[TO —]	2.37	[NP — S]	[NP — NP]
3.56	[TO — NP]	[TO — PP]	2.56	[NP — NP]	[NP — NP .]
2.56	[TO — NP]	[NP — NP PP]	2.20	[NP — NP]	[— NP]

Table 2.1: The most predictive pairs of sentential frames. (A frame, §1.2.1.3 and §5.1, is a template for lexical entries: the headword is to be filled in at —.) If $\mathtt{S} \to \alpha$ occurs in training data at least 5 times with a given headword, then $\mathtt{S} \to \beta$ also tends to appear at least once with that headword. MI measures the mutual information of these two events, computed over all words. When MI is large, as here, the edit distance between α and β tends to be strikingly small (1 or 2), and certain linguistically plausible edits are extremely common.

S if it did not appear in training data itself, but could be derived by a single transformation from some lexical entry that did appear.

A simple "bigram model" of lexical entry probabilities (see §6.6.1.2) was used to identify 2,714,763 rare entries in S—those that were predicted to occur with probability < 0.0001 given their headwords. 79 of these rare entries actually appeared in a development-data set of 1423 entries. The bigram model would have expected only 26.2 appearances, given the lexical headwords in the test data set. The difference is significant: the bigram model would be quite unlikely to draw as many as 79 appearances by chance ($p < 0.001$ by a non-parametric test).

In other words, the bigram model underpredicts the transformational neighbors of observed entries by a factor of 3. Similar results are obtained when we examine just one particular kind of edit transformation (or lexical entries of one particular size). One can therefore hope to use these edit transformations to improve on the bigram model. For example, the edit transformations correctly recognize what these experiments show: that if $\cdots$ `X Y Z` $\cdots$ is common, then $\cdots$ `X Z` $\cdots$ is plausible even if the bigram `X Z` has not previously been observed.

2.4.2.2 Edit Distance is Weighted and Context-Sensitive

If the edit distance between α and β is k, then by definition, $\mathtt{S} \rightarrow \alpha$ can be transformed into $\mathtt{S} \rightarrow \beta$ by a sequence of k transformations. Other sequences of length $\geq k$ may also do the job. The probability of transforming $\mathtt{S} \rightarrow \alpha$ into $\mathtt{S} \rightarrow \beta$ is the total probability of all these sequences.

But the probability of a given transformational sequence is determined by more than just the number of edits (e.g., k). This is because different edits have different probabilities, as illustrated in Table 2.1. For instance, insertions are more probable than swaps, insertions of optional categories are more probable than insertions of required categories, and insertions are more likely in some contexts than others. The log-probability of the sequence is therefore a *weighted* edit distance (Bahl and Jelinek, 1975; Ristad and Yianilos, 1996; Tiedemann, 1999) with context-dependent weights.

What determines the probability of applying a given transformation to a given lexical

Possessivize	$\texttt{NP} \rightarrow \texttt{Det lamb}$ *the lamb*	$\Rightarrow$	$\texttt{NP}_{\texttt{poss}} \rightarrow \texttt{Det lamb 's}$ *the lamb's*
Nominalize	$\texttt{S}_{\texttt{stem}} \rightarrow \texttt{NP bleat}$ *the lamb bleat*	$\Rightarrow$	$\texttt{NP} \rightarrow \texttt{NP}_{\texttt{poss}}\ \texttt{bleat -ing}$ *the lamb's bleating*
Tense	$\texttt{S}_{\texttt{stem}} \rightarrow \texttt{NP bleat}$ *the lamb bleat*	$\Rightarrow$ $\Rightarrow$	$\texttt{S}_{\texttt{tensed}} \rightarrow \texttt{NP bleat -ed}$ *the lamb bleated* $\texttt{S}_{\texttt{tensed}} \rightarrow \texttt{NP did bleat}$ *the lamb did bleat*
Invert	$\texttt{S}_{\texttt{tensed}} \rightarrow \texttt{NP did bleat}$ *the lamb did bleat*	$\Rightarrow$	$\texttt{S}_{\texttt{tensed,question}} \rightarrow \texttt{did NP bleat}$ *did the lamb bleat*

Figure 2.4: Some transformations that change the category of the phrase. See §5.5.4.3 for a more principled treatment of the morphology.

entry? A preview: Any instance of a transformation has certain descriptive features. Its probability is determined from these features—or rather, from weights that have been learned for the features. §1.3.3 and §1.4.2 gave examples; §3.2 will explain how this is done in general; §6.4.4 will describe the particular features used in the experiments.

2.4.3 Other Possible Transformations (future work)

The above section shows that the single-edit transformations are attested in real data. But it does not show that they are the *only* attested transformations. Indeed they are not. They are merely a reasonable starting point for experimental work.

What other linguistic transformations could we consider in future work, and how would we apply them to lexical entries in §2.4.1's "flat rule" format? Some key examples:

category changing Transformations should be able to affect the left-hand-side (LHS) nonterminal category of a lexical entry. (§5.4.4 discusses the statistical benefits of considering such transformations.)

Many languages feature category-changing transformations (Fig. 2.4): $\texttt{S}_{\texttt{stem}}$ becomes NP (nominalization), $\texttt{S}_{\texttt{tensed}}$ (finitization), or $\texttt{S}_{\texttt{inv}}$ (inversion). We can also regard feature-changing transformations such as pluralization or possessivization as category-changing.[23]

[23]One might imagine that the feature-changing cases are simple enough to handle with a weaker mechanism such as feature passing: that is, we estimate identical probabilities for $\texttt{NP[}\alpha\texttt{]} \rightarrow \texttt{Det N[}\alpha\texttt{] Sbar}$ regardless of whether $\alpha = -\texttt{plural}$ or $\alpha = +\texttt{plural}$. However, this simple design is too restrictive. It

ProduceGap	S $\rightarrow$ NP detect NP	$\Rightarrow$	S/NP $\rightarrow$ NP detect
	I detect enthusiasm		*I detect*
	PP $\rightarrow$ for NP	$\Rightarrow$	PP/NP $\rightarrow$ for
	for rice		*for*
PassGap	NP $\rightarrow$ enthusiasm PP	$\Rightarrow$	NP/NP $\rightarrow$ enthusiasm PP/NP
	enthusiasm for rice		*enthusiasm for*
	S $\rightarrow$ NP detect NP	$\Rightarrow$	S/NP $\rightarrow$ NP detect NP/NP
	I detect enthusiasm for rice		*I detect enthusiasm for*
	Note: Most gaps are consumed by lexical entries such as SBAR $\rightarrow$ that S/NP that subcategorize for them, as in the rice *that [I detect enthusiasm for]*$_{\mathtt{S/NP}}$. More such entries are created by FillGap.		
FillGap	S $\rightarrow$ NP detect NP AdvP	$\Rightarrow$	S $\rightarrow$ NP detect NP/PP AdvP PP
	I detect enthusiasm today		*I detect enthusiasm today for rice*
	S $\rightarrow$ NP rose AdvP	$\Rightarrow$	S $\rightarrow$ NP/PP rose AdvP PP
	enthusiasm for rice rose today		*enthusiasm rose today for rice*
			(cf. **Spirits rose today for rice*)
	S $\rightarrow$ it seems $\mathtt{S}_{\mathtt{tensed}}$	$\Rightarrow$	S $\rightarrow$ NP seems $\mathtt{S}_{\mathtt{inf}}\backslash\mathtt{NP}$
	it seems she likes rice		*she seems to like rice*

Figure 2.5: Gapping transformations; these too change the category of a phrase. Following GPSG and CCG, S/NP (respectively S\NP) is the category of a sentence (S) missing a post-head (respectively pre-head) noun phrase (NP). (See Fig. 5.2 for more examples of ProduceGap.)

SubjDrop (unaccusative)	NP $\rightarrow$ $\mathtt{NP}_{\mathtt{agent}}$ AdvP sink $\mathtt{NP}_{\mathtt{patient}}$	$\Rightarrow$	S $\rightarrow$ $\mathtt{NP}_{\mathtt{patient}}$ AdvP sink
	mice often sink boats		*boats often sink*
ObjDrop	S $\rightarrow$ $\mathtt{NP}_{\mathtt{agent}}$ AdvP eat $\mathtt{NP}_{\mathtt{patient}}$	$\Rightarrow$	S $\rightarrow$ $\mathtt{NP}_{\mathtt{agent}}$ AdvP eat
	mice often eat rice		*mice often eat*
Passivize	NP $\rightarrow$ $\mathtt{NP}_{\mathtt{agent}}$ eat $\mathtt{NP}_{\mathtt{patient}}$	$\Rightarrow$	S $\rightarrow$ $\mathtt{NP}_{\mathtt{patient}}$ is eat -en
	mice eat rice		*rice is eaten*
	NP $\rightarrow$ $\mathtt{NP}_{\mathtt{agent}}$ eat $\mathtt{NP}_{\mathtt{patient}}$	$\Rightarrow$	S $\rightarrow$ $\mathtt{NP}_{\mathtt{patient}}$ is eat -en [$_{\mathtt{PP}}$ by $\mathtt{NP}_{\mathtt{agent}}$]
	mice eat rice		*rice is eaten by mice*

Figure 2.6: Linguistically complex transformations: unaccusativity and passivization. Again, see §5.5.4.3 for a more principled approach to the morphology. The theta-role subscripts $_{\mathtt{agent}}$ and $_{\mathtt{patient}}$ are discussed in §5.5.4.5.

Notice that many of these transformations have morphological reflexes. They do not merely modify the argument structure and the type of the result, but they insert extra morphemes to show they have done so. See §5.5.4.3 for how to treat this more cleanly.

gapping Long-distance and control phenomena, as noted in §2.2, can be handled by passing gaps through "slashed" nonterminal categories (Gazdar et al., 1985). This would require further category-changing transformations like those in Fig. 2.5.[24] The fact that gap-passing is mediated by a lexical entry, which may have a low probability, allows for island effects.

See §5.5.4.3 for additional uses of gapping transformations. The transformation ProduceGap has appeared (under other names) in early GPSG work (Gazdar, 1981) and in HPSG (Pollard and Sag, 1994, chapter 9). Collins (1997) gave the first statistical treatment of gap-passing.

We leave experiments with gapping transformations to future work. In the meantime, given only the more impoverished transformations of §2.4.2, the model can only model extraction and control as deletion. In our current datasets, the gapped lexical entries of Fig. 2.5 are given left-hand-side nonterminals of `S` rather than the more accurate `S/NP` (thereby conflating two kinds of phrase that have rather different distributions). As a result, the model guesses that these entries were derived from the non-gapped entries by subject deletion rather than subject extraction.

unaccusativity and passivization Various other linguistically useful transformations are also not considered in the current experiments. Even when they only affect the

appropriately recognizes that singular and plural uses of an `NP` are correlated—e.g., both `rumor` and `rumors` can take sentential complements—but goes too far by assuming that they are identical. Singular and plural `NP` rules do differ in more than their $\pm$`plural` feature: plural `NP`s are more likely to lack determiners. (A more interesting example: possessive `NP`s are more likely to be head-final.) So we wish to estimate separate—but related—frame distributions for different kinds of `NP`s.

rephrase without talking about frames: need to talk about conditioning on LHS.

[24]Multiple gaps in the same phrase are possible. Things are somewhat complicated by the possibility of recursive gaps, as in *the enthusiasm that [I detect for rice today]*$_{\texttt{S/(NP/PP)}}$—a simplified version of an example that actually appears in the Penn Treebank—or *enthusiasm seems [to have risen for rice yesterday]*$_{\texttt{Sinf}\backslash\texttt{(NP/PP)}}$.

RHS and therefore can be simulated by edit sequences (§2.4.2), it would be helpful to add them explicitly in future (see footnote 21 on p. 59).

We focus here, as above, on examples that are attested in English. Fig. 2.6 shows that while object drop is a simple edit to the RHS, unaccusative movement and passivization require longer movement of arguments within an RHS. (Unaccusative movement is a version of passivization, available to some verbs, that has no morphological reflex.)

Notice that object drop is distinguished from unaccusative movement in Fig. 2.6 only by the thematic-role subscripts. The subscripts indicate that for unaccusative movement, the object in $\mathtt{S} \to \mathtt{NP}$ —— $\mathtt{NP}$ tends to have the same heads as the subject in $\mathtt{S} \to \mathtt{NP}$ ——. That is, the subscripts serve to help describe the selectional preferences of the head (§5.5.4.5). Using a model of selectional preferences, we could in principle distinguish the two transformations (§5.5.4.5).

What if the agent of a passive is displaced to a `by`-phrase? One possibility is to handle this in the semantics: i.e., the `by`-phrase is an ordinary instrumental `PP` that is interpreted as specifying the agent because there is no other agent. Handling it in the syntax instead is stickier because of the notion that transformations are local to a frame. It requires that the agent and patient switch places, with `by` inserted as an additional morphological reflex. This requires something like the structure at the end of Fig. 2.6, where the `by-PP` has been flattened into the `S` frame.

To regard the latter move as an instance of Structure Preservation (mentioned in §2.3.6), one would have to represent all `PP`'s as flattened into their parents in this way (or simply regard `PP`'s as a species of case-marked `NP`). This is not a wholly unreasonable representation, as it would allow our selectional preference model (§5.4.1) to capture a statistical dependence in between `saw` and `telescope`, the parent and child of the preposition, in *saw the man with the telescope*. The preposition `by` or `with` becomes "transparent." Such statistical dependence does exist, not just in the case of passivization but more generally. It has been exploited by Collins and Brooks (1995) and in subsequent work on `PP` attachment disambiguation.

Some additional transformations are discussed in §5.5.4.2.

2.5 A Review of Related Work

There has been much work over the past several years on probabilistic lexicalized grammars, including the induction of subcategorization frames or dependency frames.

However, there has been relatively little work on smoothing the probabilities of lexical entries. In particular, there is little work assigning non-zero probability to lexical entries that contain novel dependency frames. Such entries are surprisingly common (§1.2.1.2).

2.5.1 Non-Statistical Approaches with Similar Concerns

The idea of uniformly representing generalizations, partial generalizations, and exceptions in a computational lexicon (§§1.3.2–1.3.3) goes back a long way, at least to the non-monotonic multiple-inheritance hierarchies that Flickinger (1987) introduced to HPSG. His approach is similar to object-oriented programming. Like an object-oriented class, a lexical entry type may inherit all its properties from its parents, or define some of its own.

The defeasible inheritance relationships in Flickinger's HPSG hierarchies can be regarded as transformations that specialize and partially override the parent to derive the child. HPSG also allows more general transformations, called lexical rules or metarules. While these were originally exceptionless, Briscoe and Copestake (1999) show how to make them defeasible as well.

Evans et al. (2000) make a very similar set of arguments, and propose to represent LTAG lexicons efficiently using the DATR formalism, which allows them to encode both generalizations and exceptions in a unified fashion.

Schütze (1994) studies only the dative alternation between *give her a cat* and *give a cat to her*, but shares our interest in modeling lexical idiosyncrasy. He uses a feed-forward neural network that maps a verb's name and its semantic features to output units that correspond to the two possible frames. Such a model can learn arbitrary exceptions. Schütze introduced a learning bias in favor of generalizations over exceptions. He artificially made the verb's name pass through an extra network layer, so that it had less influence than the semantic features during the initial stages of back-propagation training. Hence the network initially paid more attention to the verb's semantic features than to its specific identity (and passed through a stage of overgeneralization, as children do).

2.5.2 Previous Statistical Methods for Smoothing Lexical Entries

Methods that can generalize to arbitrary novel frames—those that were not observed in training data—are at the core of this thesis. For the sake of brevity they are discussed only in §6.6.1, where they are replicated. The reader is welcome to peek ahead to that section now.

This thesis focuses on using *syntactic* similarity to smooth lexical entries. By contrast, Korhonen (2000) uses *semantic* similarity. As in Chapter 6 here, she does estimate Pr(frame | headword). But whereas we exploit similar frames, she exploits similar headwords, by backing off to Pr(frame | Levin class of headword). This approach relies on Levin's (1993) insight that semantically similar verbs tend to appear with the same frames (see §5.5.1.2).

While Korhonen defined semantic similarity by using Levin's existing verb classes, it is possible to discover semantic clusters of verbs or verb senses by more automatic means. §5.5.1.2 notes several papers, such as (Rooth et al., 1999), that cluster word types or tokens by their lexical selectional preferences, and obtain clusters that are indeed semantically coherent. Charniak (1997) achieved a small improvement in predicting subcategorization frames by backing off to such clusters.

It is worth noting that one could also combine backoff to semantic clusters with syntactic methods for predicting subcategorization frames. Any of the models in §6.6.1 could readily back off from word to cluster. In the case of transformational smoothing, the method of (Rooth et al., 1999) could for example be used to replace each word *token* in a parsed corpus with a coarse sense category corresponding to that token's best cluster. One could then apply transformational smoothing to the "lexical" entries extracted from this altered corpus. Such "lexical" entries would be headed by quasi-semantic categories, not words. Underlying this scheme is Levin's notion that frames and transformations are in fact properties of such categories.

Finally, unbeknownst to us, Briscoe and Copestake (1999) independently arrived at something like the high-level idea of this thesis. They too suggested smoothing Pr(frame | headword) by redistributing probability mass among lexical entries according to (something like) the relative rates of transformations. However, their smoothing technique was self-consciously ad hoc:

- Specify the complete, finite set of frame types $f_1, f_2, \ldots f_n$ and all legal transformations among them. (This is analogous to the transformation graph of §1.4.1, but without arc probabilities or features or an unbounded number of novel frames. Briscoe and Copestake explicitly reject the notion that adjuncts might be inserted by transformation (see §2.4.1), which could make n infinite.)

- For each transformation $f_i \Rightarrow f_j$, where f_i and f_j are subcategorization frames, define the transformation's **productivity** as

$$\text{Prody}(f_i \Rightarrow f_j) \stackrel{\text{def}}{=} \frac{\#\text{ of word types observed at least once with } f_i \text{ and } f_j}{\#\text{ of word types observed at least once with } f_i} \tag{2.2}$$

  For example, in Table 1.1 on p. 5, the productivity of a transformation from row 1 $\Rightarrow$ row 3 would be 1/6, and the productivity of row 5 $\Rightarrow$ row 6 would be 0/1. There is no significance testing for this ratio.

- Now, given a lexical entry (w, f_j), one wishes to estimate $\Pr(f_j \mid w)$. The maximum likelihood (ML) estimate would be $s_j / \sum_k s_k$, where s_j is the number of times (w, f_j) appeared in training data. What Briscoe and Copestake suggest is a way to avoid zeroes by a Katz-style variation on add-one smoothing.[25] Let $m < n$ be the number of frames unattested for w (that is, the number of s_j that equal 0). Briscoe and Copestake first suggest artificially increasing each of those m frames' counts from 0 to 1, but this would not differentiate among the unattested frames at all. So instead they divide up the m extra counts among the unattested frames by a kind of lower-order model (by analogy with (Katz, 1987)). That is,

$$\Pr(f_j \mid w) \text{ is estimated as } \begin{cases} \frac{s_j}{m+\sum_k s_k} & \text{if } s_j > 0 \\ \frac{m}{m+\sum_k s_k} \cdot \Pr_{\text{lower}}(f_j) & \text{if } s_j = 0 \end{cases} \tag{2.3}$$

[25]The connections to add-one smoothing and (Katz, 1987) are not tight. 1 is added only to 0 counts, not to all counts; and then only these extra 1 counts are redistributed.

The lower-order model is a probability distribution over just the unattested frames, and tries to favor frames that are commonly produced by transformation. Let $t_j \stackrel{\text{def}}{=} (f_i \Rightarrow f_j)$ denote the unique transformation that derives f_j. (It is unclear how to define t_j uniquely if there are zero or two choices for f_i, or whether f_i can be considered if $s_i = 0$.) Then define

$$\Pr_{\text{lower}}(f_j) = \frac{\text{Prody}(t_j)}{\sum_{k \text{ such that } s_k=0} \text{Prody}(t_k)} \tag{2.4}$$

Notice that this technique (unlike ours) does not redistribute mass *along the transformation arcs.* That is, it does not redistribute mass from the frames attested with w to their children or other descendants in the transformation graph. Rather, it redistributes mass from all frames attested with w to all frames unattested with w. Other things equal, any of the following conditions help to increase $\Pr(f_j \mid w)$ for unattested f_j: (1) w was poorly observed (small $\sum_k s_k$). (2) f_j appears to have been frequently derived (with other words). (3) The other unattested frames for w appear not to have been frequently derived (with other words).

Our transformational smoothing is a conceptual improvement in several ways. For example:

- It redistributes mass along the transformation arcs from attested frames. So $\Pr(f_j \mid w)$ will be increased if (1) f_j's parents or other ancestors are frequently attested *with* w, and (2) the necessary transformations appear to be probable (with other words).
- Mass is redistributed to f_j as above even if f_j is itself attested. So positive probabilities can be smoothed upward, not just downward, as appropriate to correct for sampling error.
- The amount of mass redistributed is not determined by the raw number of unattested frames (allowing us to have arbitrarily many unattested frames without sucking all the probability from the attested ones).
- It is possible to derive novel frames that were not attested with *any* word. Indeed, n need not be finite.

- Parameters can be tied interestingly across transformations (§1.3.3), so that evidence for one transformation may increase the probabilities of similar transformations.
- A prior discourages spurious generalizations from little evidence.
- Sequences of transformations are allowed.
- Transformations compete only if they apply to the same lexical entry.
- The approach can in principle do abduction and explaining-away of frames, as in Bayes nets.

2.5.3 Extracting Subcategorization Frames from Text

A great many recent papers have tried to automatically build dictionaries of subcategorization frames. (Sarkar and Zeman (2000, Table 2) give a comparison.) Such dictionaries could serve as input to our generalization methods. In particular, an attempt to bootstrap the acquisition of syntax, as sketched in §1.2.4.2, should be initialized with some reasonable (if imperfect and incomplete) guess as to the correct grammar, so that it is less likely to get stuck in a poor local maximum.

In the case of English, there already exist manually-constructed subcategorization dictionaries, such as COMLEX (Grishman et al., 1994), FrameNet (Johnson et al., 2001), and their predecessors intended for human consumption (Procter, 1978; Sinclair, 1987; Hornby, 1989). However, it is still useful to augment such dictionaries from large or domain-specific corpora (Manning, 1993). In addition, some researchers have been interested in the language-learning problem for its own sake.

This line of work began with Brent (e.g., (1993; 1994)), who used heuristic, English-specific surface cues to extract a few types of frames from transcribed speech to children. The work of Ushioda et al. (1993) is similar. Manning (1993) followed the same scheme but used newspaper text that had been automatically tagged and chunked, allowing him to design cues for a wider variety of frames.

Subsequent researchers have used text that was parsed automatically or else semi-automatically (Ersan and Charniak, 1995; Delisle and Szpakowicz, 1997; Korhonen, 1997;

Carroll and Rooth, 1998; Green, 1997; Arriola et al., 1999; Sarkar and Zeman, 2000). It is then possible to simply read the subcategorization frames off the parse trees, as we will do in §6.2. Most of these systems did not attempt any further generalization or smoothing. However, Ersan and Charniak (1995) made an effort to normalize the observed frames by deleting adjuncts, and some systems zero out the counts of observed frames not on a pre-approved list (Ersan and Charniak, 1995; Korhonen, 1997; Carroll and Rooth, 1998). or require a user in the loop during frame extraction (Delisle and Szpakowicz, 1997; Arriola et al., 1999).

Loosely related are the systems of Siskind (1996) and Thompson and Mooney (1999). These greedily construct lexical entries by "aligning" raw sentences with detailed semantic representations, which are assumed to be given.

All these methods can in principle discover novel frames, if the input cannot be accounted for in any other way (see §1.2.4.2 on failure-driven learning). However, they do not generalize from one frame to another, nor do they learn frame probabilities.

Most of these systems try to correct for noise that arises from fallible heuristics or fallible parsing of the input. They use various significance-testing techniques to filter the output. (Brent's (1993) original binomial significance test is most common; (Korhonen et al., 2000; Marques et al., 2000; Sarkar and Zeman, 2000) compare some techniques used.) The result is generally a dictionary of lexical entries with no probabilities and no rare entries.

In principle, one could try to correct for noise by instead producing smoothed probabilities or confidence intervals, which would be more useful to a statistical parser. However, it would remain difficult to distinguish rare entries from spurious (or, in our terms, extremely rare) ones, and impossible to include frames that were not in the data. Such behavior requires procedures for generalizing from one frame to another, such as the transformational smoothing proposed here.

2.5.4 Modeling Optionality in Verb Subcategorization

The work in the previous section tries to compile arbitrary lists of subcategorization frames for each headword.

In a variation, a number of researchers have tried to induce more structured representations of each headword's syntactic preferences. These researchers conceptualize a lexical entry as a headword together with a core ordered list or unordered set of roles to be filled. Roles are typically identified by case marker (e.g., preposition) or syntactic position. For example, an English verb may have roles "subject," "object," "for," "in," "to," etc.

A given token of a headword may appear with only some subset of these roles (the others being realized as the empty string $\emptyset$, leaving the hearer to infer them from context). It may also appear with additional adjuncts not in the core role set. So these researchers allow operations that either suppress roles from this set (realizing them as the null string ϕ) or add roles to the set. As a result, they get a degree of generalization from lexical entry to lexical entry, an idea at the core of this thesis.

2.5.4.1 Non-Stochastic Optionality

Webster and Marcus (1989) considered input that was parsed and annotated with roles; they simply kept track of which roles for a verb appeared to be obligatory or optional in the input.

Basili et al. (1997) used a similar approach in the pursuit of distinguishing verb senses. Observed role sets were organized into a subsumption lattice. The senses were derived from this lattice and mapped onto dictionary senses. Thus, lexical entries connected by small edit distance were considered to be related, as in transformational smoothing.

Sarkar and Zeman (2000) assumed that an observed lexical entry was derived by adjunction if the frame and headword were not sufficiently associated—if they might well have occurred together by chance. In this case, they chose a putative adjunct in the frame, and reclassified the observation as if the adjunct were deleted. Smaller lexical entries could thereby accumulate more observations than they would have received by chance, allowing them to survive. Maragoudakis et al. (2000) ported this approach from Czech to Modern Greek.

Buchholz (1998) used memory-based learning to train a system that classified each nonterminal in a verb frame as either an argument (presumably obligatory) or adjunct (presumably optional). She found that the nonterminal, its closed-class headword if any, and the previous nonterminal were the most important predictors.

2.5.4.2 Stochastic Optionality

More relevant to our work are approaches that determine the contextual *probability* that an optional role will be realized or suppressed.

Collins (1997) used a hard-coded heuristic (rather than learning a classifier à la Buchholz) to distinguish arguments from adjuncts in each training frame. He used this to train an interesting model in which unboundedly many adjuncts could be probabilistically inserted among the arguments. We will discuss and replicate this approach in §6.6.1.3.

The "slot-based model" in (Li and Abe, 1996) models the contextual probabilities of role suppression. The model says that for a given verb, each role depends on at most one single "previous" role: its suppression probability is conditioned on whether that previous role was suppressed. In other words, the model is a tree-structured Bayesian network (a dendroid distribution). The tree topology specifies which roles depend on one another; it is chosen as a minimum spanning tree of a graph in which roles with high mutual information are considered to be close together.

Since Li and Abe learned a separate dendroid distribution for each verb, sparse data considerations limited them to high-frequency verbs ($\geq$ 50 training tokens). (Typically they learned that a verb was more likely to take certain `PP`-arguments in the presence of a direct-object `NP`.) Miyata et al. (1997), who extended Li and Abe's approach to Bayesian networks with more complicated structure, considered only verbs of even higher frequency ($\geq$ 300 training tokens). However, one could presumably share parameters across verbs in order to extrapolate patterns for low-probability verbs.

Utsuro et al. (1998) also treated role suppression, modeling lexical entries using a rather complicated log-linear technique. A verb's lexical entry is the set containing the verb and its roles. They greedily grew a collection of increasingly larger subsets from which one could assemble each training lexical entry by disjoint union. For example, one might derive an training entry as the union of a basic subcategorization frame with various adjuncts, or with pairs of adjuncts such as *from ... to*. Any putative lexical entry then has zero or more derivations as a disjoint union of such subsets. The entry's features in the log-linear model are exactly the subsets that appear among its derivations (or a special feature "underivable"). Notice that growing the collection of available subsets amounts to feature selection; they add subsets in general-to-specific order, and use an MDL criterion to decide when to stop growing.

These papers are actually more sophisticated than described here. Except for Collins's, they try to learn not only the roles for a verb, but the selectional restrictions on those roles with respect to a semantic hierarchy. For example, the object role of `drink` can only be filled by liquids. Learning these selectional restrictions is part of the interest of the papers, though we have ignored it here for simplicity's sake.

2.5.5 Other Uses of Syntactic Transformations

While the present work uses probabilistic syntactic transformations *within* the grammar of a language, they can also be used to model the process of translating to a different language.

Yamada and Knight (2001) focus on transformations that permute the subconstituents of a constituent. They model Pr(Japanese tree | English tree) as a noisy channel that modifies each maximal projection in the English tree by (among other things) stochastically reordering its children.[26] Like us (§2.4.1), they operate over flattened structures such as $[_{\mathtt{S}}\ \mathtt{NP}\ \mathtt{V}\ \mathtt{NP}\ ]$, since this is a natural domain over which to define cross-language as well as within-language transformations. Thus, they are concerned with estimating probabilities such as Pr(Japanese $[_{\mathtt{S}}\ \mathtt{NP}_1\ \mathtt{NP}_2\ \mathtt{V}\ ]$ | English $[_{\mathtt{S}}\ \mathtt{NP}_1\ \mathtt{V}\ \mathtt{NP}_2\ ]$), which is the probability of transforming SVO into SOV word order. This is similar to the probability of Swap in our §2.4.2.

More relevantly, the same translation approach can be applied to rephrasings within a

[26] Their noisy channel also inserts, translates, and deletes nonterminals. It is worth drawing a contrast between their approach and synchronous grammars. Recall that the noisy-channel approach to translation (Berger et al., 1994) translates Japanese to English by trying to undo the effect of a putative noisy channel that stochastically translated English to Japanese. In this case, this means choosing the English tree that maximizes the product Pr(English tree) $\cdot$ Pr(Japanese string | English tree). Rather than assume separate processes for generating English (the first factor) and translating it into Japanese (the second factor), an alternative is to assume that the English and Japanese were generated in parallel by a synchronous grammar (Wu, 1997); in this case the "original" English and "transformed" Japanese phrase structure rules are generated together with some joint probability, and there is no separately modeled transformation probability.

language. This is close to Chomsky's original idea of transformations. Instead of regarding Japanese sentences as noisy translations of English ones, Knight and Marcu (2000) regarded long English sentences as noisy expansions of short English sentences. Removing the noise then corresponds to summarization rather than translation. They are concerned with estimating probabilities such as Pr(long $[_{\mathtt{S}}$ `NP VP PP ]` | English $[_{\mathtt{S}}$ `NP VP ]`). This is similar to the probability of Insert in our §2.4.2.

Both the above problems differ from ours in that they are formally translation problems. The training data were clearly separated into paired "before" and "after" sentences, where each pair was assumed to be related by a single transformation. By contrast, we are concerned with the case where any training datum may be derived from any other by a sequence of transformations.

2.5.6 Edit-Distance Methods

The method of this thesis is to notice when frames that are similar—e.g., in the sense of having small weighted edit distance from one another (§2.4.2)—tend to appear with the same words.

Weighted edit distance has sometimes been employed in other natural-language tasks, such as modeling pronunciation variation within a language (Ristad and Yianilos, 1996; Ristad and Yianilos, 1998) and either detecting (Tiedemann, 1999) or predicting (Knight and Graehl, 1997) cognates across languages.

The alignment-based learning (ABL) technique of van Zaanen (2000) is like us concerned with inducing linguistic structure, though at level 2 rather than level 3 of the grammar (§2.1.5). The technique uses a (slightly hacked) unweighted edit distance to detect and align similar sentences in a raw corpus.[27] If two sentences are similar, their best alignment partitions each of the sentences into substrings that are preserved and substrings that are replaced when editing one sentence to yield the other. If substring x is replaced with y, then ABL guesses that x and y (either of which may be empty) are constituents of the same nonterminal type.

[27] The work uses the ATIS and OVIS corpora, which fortunately contain many similar sentences. Speech to children also contains much near-repetition.

The analogy-based lexical acquisition system (ABLAS) (Calzolari et al., 1998) tries to find new subcategorization frames in chunked text. It does so by alignment with already-known subcategorization frames.

It may be worth noting that memory-based learning (Zavrel and Daelemans, 1997) can be regarded as using a kind of weighted edit distance between feature vectors. Feature vectors have fixed length, so the only edit allowed is Substitute, but some feature substiutions have greater cost than others. Other k-nearest-neighbor methods such as similarity-based smoothing (Dagan et al., 1997) have a comparable flavor.

What all these methods crucially have in common is that they attend to the superficial relations among surface phenomena in an unstructured set. They may or may not posit underlying mechanisms, but the basis for their learning is *pairs* of similar observed events. A caveat is that such pairs may be hard to find in very small corpora (see footnote 27 on p. 75), so it may sometimes be helpful to incorporate or back off to some generative model of *single* events.

2.5.7 Priors on Grammars

One of the distinctive aspects of the present work is the assignment of a prior probability to each possible grammar §1.3, so that learning can favor grammars that we have deemed *a priori* likely. This idea was first proposed by Solomonoff (1964) in the form of Minimum Description Length (MDL) learning.

In MDL, the log prior probability of a grammar is considered to be the number of bits it takes to describe the grammar in some notation (code). While any prior can be implemented by use of a sufficiently convoluted notation, certain simple styles of notation are traditional in the MDL community.

Several recent attempts have been made to learn context-free grammars from text, using the MDL framework.[28] The descriptions chosen lead to priors that are rather different from ours:

[28] Cracking this problem seems to be a common grad-student ambition: four of the five citations here are to theses or portions thereof!

- Stolcke and Omohundro (1994b) consider a non-lexicalized, probabilistic context-free grammar to be likely if (1) the total length of all its rules is small—i.e., it contains few rules of non-zero probability and these rules have short right-hand sides—and (2) the several rules that rewrite a given nonterminal are roughly equiprobable.

- Chen (1996, chapter 3) takes a context-free grammar of restricted form to be likely if it has few nonterminals and few rules, and the rules are about equiprobable.[29] Certain types of rule are more probable than others since they can be coded economically.

- de Marcken (1996, chapter 4) takes a hierarchical lexicon of strings to be likely if the lexicon lists few strings, and the strings it does list are predicted to have fairly high probability even without such a listing. (That is, listed strings would be expected to arise frequently anyway by random selection and concatenation of other listed strings.) It also helps if most listed strings e are listed with "standard" probabilities (specifically, common values of the integer $\lceil \log_2 \Pr(e) \rceil$).

  De Marcken also suggests (his Figure 4.1) that when a complex lexical entry is not fully predictable from its parts, then its exceptional properties—perhaps including an exceptional probability—should be listed with it at extra cost. But he does not make the cost of such an exceptional probability depend on the *degree* of its exceptionality, the way we do (§2.3.6).

- Grünwald (1996) takes a non-lexicalized context-free grammar to be likely if it has few productions.

- Osborne and Briscoe (1997) take a lexicalized, probabilistic categorial grammar to be likely if (1) the total length of all its frames is small—i.e., it contains few frames of non-zero probability and these frames have short right-hand-sides—and (2) each word tends to list only a small number of frames and these frames are frequent. They argue that (2) can be ignored in practice.

[29]The latter point arises from the fact that Chen codes the rule probabilities in terms of observed counts in training data, and it is cheaper to code equal counts (that add up to a fixed training data length) than unequal ones.

The prior used in this thesis is quite different in spirit. Rather than try to keep the number of rules in the grammar small, it takes the view that any grammar contains *all* possible rules or lexical entries (§1.2.4.3). Grammars differ only as to the probability distributions they impose over these rules or lexical entries. In general, all grammars assign non-zero probability to everything.

Given that framework, it would still be in the spirit of the above methods to assert that a good *a priori* probability distribution is one with high entropy, or low entropy, or some other figure of merit that depends only on the histogram of event probabilities. But that is *not* the assertion of this thesis. Rather, the prior proposed here says that a good probability distribution is one that is "internally coherent," in the sense that most of the probabilities are well-predicted from other probabilities in the distribution, using a small and simple set of transformational generalizations that take the events' structure into account.

Chapter 3

Smoothing Using Transformation Models

This chapter introduces **transformation models**, a new class of parameterized probability distributions. We will see that such models allow us to do **transformational smoothing** of sparse data.

The rest of the thesis applies this new statistical technique to one important problem in computational linguistics. The technique may also have other uses inside or outside computational linguistics (see §7.2 for examples).

The reader who just wants the equations in a concise form should look ahead to §4.1.

3.1 Preliminaries

Let us begin by spelling out the conceptual framework of Bayesian smoothing, as well as the notation used in this thesis. The ideas and notation will be familiar to many readers.

Most previous work in grammar learning has *not* used this Bayesian framework. However, (Stolcke and Omohundro, 1994b; Grünwald, 1996; Chen, 1995; de Marcken, 1996; Osborne and Briscoe, 1997) have followed Solomonoff (1964) and used the Minimum Description Length (MDL) framework, which is technically equivalent—although MDL philosophy and practice favor a style of prior different from the one used in this thesis.

3.1.1 Probability Distributions

Suppose we are considering a set Events of events whose probabilities are of interest, such as the set of possible sentences, syntax trees, or poker hands. (In the main application of this thesis, Events consists of all possible lexical entries: that is, objects that have the correct form to serve as entries in a human syntactic lexicon, under some linguistic theory of the lexicon.)

A **probability distribution** over Events is any function $\Pr : \mathsf{Events} \to [0, 1]$ such that $\sum_{e \in \mathsf{Events}} \Pr(e) = 1$. It is extended to event sets $E \subseteq \mathsf{Events}$ by defining $\Pr(E) \stackrel{\text{def}}{=} \sum_{e \in E} \Pr(e)$.[1] As a matter of notation, the set E is usually specified by a predicate or partial description that matches just the elements of E: so $\Pr(\mathtt{S})$ might mean the total probability of all lexical entries of category $\mathtt{S}$. Commas stand for conjunction: $\Pr(\mathtt{S}, \mathtt{devour})$ is the total probability of all lexical entries that have category $\mathtt{S}$ *and* headword $\mathtt{devour}$.

The conditional probability $\Pr(E \mid F)$ is defined as $\Pr(E \cap F)/\Pr(F)$. Finally, if $\vec{e}$ is a finite sequence (vector) of n events, then $\Pr(\vec{e})$ is usually defined as $\prod_{i=1}^{n} \Pr(e_i)$, that is, the probability of obtaining $\vec{e}$ by drawing a sequence of n independent samples from Pr. When we have such a sequence of samples, let $\#(e)$ denote the number of times event e was sampled, i.e., the number of components of $\vec{e}$ that are equal to e.

We will sometimes use mnemonic names for distributions, such as $\Pr^{\text{lex}}(w, f)$ for a particular probability distribution over (w, f) pairs.

The functional notation for Pr faces a standard ambiguity. For fixed w, the notation $\Pr(f \mid w)$ might refer to a particular probability (when f is also fixed). Or it might refer to the entire distribution, a function that maps any f to $\Pr(f \mid w)$. We sometimes use the conventional notation $\Pr(\cdot \mid w)$ to emphasize the latter interpretation.

3.1.2 Parameterized Probability Distributions

Often one wishes to consider a **parameterized family of probability distributions** together with prior probabilities on the parameters. Such a family-with-prior may be

[1] Actually, this intuitive definition only works if Events is countable (or has countable support), unless we find some way of defining uncountable summations over infinitesimals (the same problem faced by integral calculus). The solution (measure theory) can be found in any textbook on probability theory. The details need not concern us here—although this thesis will in fact have to consider distributions over uncountable sets of events, namely the the parameter space $\mathbb{R}^k$ and the set of walks in a finite graph.

formalized as a single probability distribution Pr over $\Theta \times \mathsf{Events}$, where Θ is a set called the **parameter space**. Each element $\theta \in \Theta$ is called a **parameter value** (but is typically a fixed-length real vector that specifies several scalar parameters). For each θ, let $\Pr_\theta(e) \stackrel{\text{def}}{=} \Pr(e \mid \theta)$, and note that this is a probability distribution over Events—giving us a family of distributions $\Pr_\theta$ for different values of θ. Now for $\theta \in \Theta, e \in \mathsf{Events}$, it is true by definition that

$$\Pr(\theta, e) = \Pr(\theta) \cdot \Pr(e \mid \theta) \tag{3.1}$$

When defining a parameterized family in practice, the two factors in equation (3.1) are defined independently. That is, one must define

- a parameterized family $\{\Pr_\theta : \theta \in \Theta\}$ of distributions over Events
- a prior probability distribution $\Pr_{\text{prior}}$ over Θ

Then one can set $\Pr(\theta) = \Pr_{\text{prior}}(\theta)$ and $\Pr(e \mid \theta) = \Pr_\theta(e)$, in equation (3.1), to define the parameterized family-with-prior, a distribution over $\Theta \times \mathsf{Events}$:

$$\Pr(\theta, e) = \Pr_{\text{prior}}(\theta) \cdot \Pr_\theta(e) \tag{3.2}$$

What this really defines is a prior probability distribution over probability distributions! $\Pr_{\text{prior}}(\theta)$ is a prior distribution over values of θ, each of which defines a distribution $\Pr_\theta$.

3.1.3 Smoothing via Bayes' Theorem

Smoothing is a traditional term for estimating a distribution from a sequence $\vec{e}$ of independent samples of that distibution. An "unsmoothed" estimate sets $\Pr(e)$ to be proportional to $\#(e)$, the number of occurrences of e in $\vec{e}$: in particular it sets $\Pr(e) = 0$ if e happened not to be sampled. Smoothing is any technique for improving on that naive approach.

The unsmoothed estimate of Pr has the virtue that it maximizes the likelihood $\Pr(\vec{e})$. Smoothing methods guess that some other distribution is nonetheless more plausible, because of its smoother, more reasonable shape. Such methods essentially make prior assumptions about the distribution class and parameters. If such prior assumptions can be stated explicitly, they immediately give rise to a principled (though perhaps computationally expensive) smoothing method via Bayes's Theorem.

Specifically, suppose one believes *a priori* that $\vec{e}$ consists of independent samples drawn from the *same* distribution in some parameterized family: that is, drawn from $\Pr_\theta$ for some single but unknown parameter vector θ. One also has a prior distribution $\Pr_{\text{prior}}(\theta)$. The goal is to guess the underlying θ.

Bayes' Theorem helps make such a guess. It tells us how probable each possible θ is given what we know. The **likelihood** of a given θ is defined as

$$\Pr(\vec{e} \mid \theta) = \Pr_\theta(\vec{e}) = \prod_{i=1}^{n} \Pr_\theta(e_i) \tag{3.3}$$

and the **posterior probability** of θ is then

$$\Pr(\theta \mid \vec{e}) = \frac{\Pr(\theta) \cdot \Pr(\vec{e} \mid \theta)}{\Pr(\vec{e})} \propto \Pr(\theta) \cdot \Pr(\vec{e} \mid \theta) = \Pr_{\text{prior}}(\theta) \cdot \Pr_\theta(\vec{e}) \tag{3.4}$$

where $\propto$ is read "proportional to." It is convenient to guess that the underlying distribution is $\Pr_{\hat{\theta}}$ where $\hat{\theta}$ maximizes $\Pr(\hat{\theta} \mid \vec{e})$ (or, more conveniently, its logarithm). This is known as a **maximum *a posteriori*** or **semi-Bayesian** estimate.[2] By equation (3.4), this estimate

[2]As contrasted to the **full-Bayesian** estimate $\Pr^{\text{opt}}$, defined by

$$\Pr^{\text{opt}}(e) = \int_\theta \Pr_\theta(e) \Pr(\theta \mid \vec{e}) d\theta \tag{3.5}$$

which is superior in principle and sometimes in practice, but which is far harder to compute and which usually does not itself fall into the parameterized family. For the record, Markov chain Monte Carlo (MCMC) techniques such as Gibbs sampling can be used to estimate $\Pr^{\text{opt}}(e)$.

In general, any parameter of a model can be (a) integrated out in full-Bayesian style, or it can be (b) estimated by maximizing equation (3.4), (c) estimated by maximizing the learned model's performance on held-out data, or (d) set by hand.

Sometimes it is even useful to treat different parameters differently. Suppose we have partitioned θ into four subvectors $\vec{a}, \vec{b}, \vec{c}, \vec{d}$, to be respectively treated in these four ways. We are given a joint prior on θ, and $\vec{d}$ is constant. Given an observed sample $\vec{e}$ of $\Pr_{\vec{a},\vec{b},\vec{c},\vec{d}}(\cdot)$, define

$$\begin{aligned}
\Pr(\vec{a} \mid \vec{b}, \vec{c}, \vec{d}, \vec{e}) &\stackrel{\text{def}}{=} \frac{\Pr_{\text{prior}}(\vec{a} \mid \vec{b}, \vec{c}, \vec{d}) \cdot \Pr_{\vec{a},\vec{b},\vec{c},\vec{d}}(\vec{e})}{\int_{\vec{a}'} \Pr_{\text{prior}}(\vec{a}' \mid \vec{b}, \vec{c}, \vec{d}) \cdot \Pr_{\vec{a}',\vec{b},\vec{c},\vec{d}}(\vec{e})} && \text{(analogous to equation (3.4))} && (3.6)\\
\Pr_{\vec{b},\vec{c},\vec{d}}(e) &\stackrel{\text{def}}{=} \int_{\vec{a}} \Pr_{\vec{a},\vec{b},\vec{c},\vec{d}}(e) \cdot \Pr(\vec{a} \mid \vec{b}, \vec{c}, \vec{d}, \vec{e}) && \text{(analogous to equation (3.5))} && (3.7)\\
\Pr(\vec{b}, \vec{c}, \vec{d} \mid \vec{e}) &\stackrel{\text{def}}{=} \Pr_{\text{prior}}(\vec{b}, \vec{c}, \vec{d}) \cdot \Pr_{\vec{b},\vec{c},\vec{d}}(\vec{e}) \,/\, \text{const} && \text{(analogous to equation (3.4))} && (3.8)\\
\vec{b}_{opt} &\stackrel{\text{def}}{=} \operatorname*{argmax}_{\vec{b}} \Pr(\vec{b}, \vec{c} \mid \vec{e}) && && (3.9)\\
\vec{c}_{opt} &\stackrel{\text{def}}{=} \operatorname*{argmax}_{\vec{c}}(\text{some performance measure of } \Pr_{\vec{b}_{opt},\vec{c},\vec{d}} \text{ on held-out data}) && && (3.10)
\end{aligned}$$

We then estimate

$$\Pr_{opt}(e) \stackrel{\text{def}}{=} \Pr_{\vec{b}_{opt},\vec{c}_{opt},\vec{d}}(e) \tag{3.11}$$

See (MacKay, 1996) for discussion and experiments.

must score reasonably high on the prior and also do a reasonably good job of predicting the evidence.

3.1.4 Transformational Smoothing via Bayes' Theorem

Transformational smoothing is precisely the above strategy when our parameterized probability distribution $\Pr_\theta$ is a transformation model (as defined in §3.2 below). It is appropriate when our knowledge of the problem domain allows us to identify a plausible transformation model. (That is, it should be plausible *a priori* that a parameter value θ will be chosen according to the model's prior, and that all our evidence and test data will then be generated from $\Pr_\theta$.)

Specifying such a model requires us to specify a prior distribution $\Pr_{\text{prior}}(\theta)$. Specifying priors by hand is ordinarily a black art. Fortunately, we will see that transformation models admit natural priors that implement a form of Occam's Razor. These priors do nothing but favor distributions $\Pr_\theta$ that are "simple and regular," so that equation (3.4) comes to mean "do not posit a distribution that is more complex than necessary to fit the data."

In the case of syntax, the transformation model—a family of distributions—corresponds to the space of possible human languages (Universal Grammar). Different values of θ correspond to particular human grammars, and each grammar θ defines a probabilistic language $\Pr_\theta$. Given a monolingual sample $\vec{e}$ drawn from $\Pr_\theta$—which is roughly what most children observe—the learner's job is to guess θ. The learner's prior distribution $\Pr_{\text{prior}}(\cdot)$ is one that expects to encounter grammars with simple regular rules and few exceptions. (In principle, the prior could also be modified to favor some kinds of rules over others, if we wished to give the learner substantive biases toward linguistic universals or near-universals.)

Having guessed that $\theta = \hat{\theta}$, the learner can use the probabilistic language $\Pr_{\hat{\theta}}$ to assess what *other* events in Events are probable in the language. This is useful for interpreting novel inputs, for disambiguating ambiguous inputs, and perhaps for language production.

An instructive application of the Bayesian strategy to bigram smoothing is (MacKay and Peto, 1995), which is sketched below in §7.1.5 but is worth reading in full.

3.1.5 Other Forms of Evidence

The above discussion of smoothing suffices for many applications. But with an eye toward the main application of this thesis, and other applications sketched in §7.2, it is worth generalizing the notion of "evidence."

We sometimes have evidence of a particular θ that does not come in the form of a random sample $\vec{e}$ from $\Pr_\theta$. For example, if the parameters θ correspond to observable mechanisms, then we might have direct evidence of them. Or perhaps $\Pr_\theta$ indirectly influences someone's actions, and we observe those actions.

Equation (3.4) can easily deal with such cases. We need only be able to assess how well θ explains whatever we have observed. Merely replace the likelihood factor $\Pr(\vec{e} \mid \theta)$ with $\Pr(\text{evidence} \mid \theta)$.

As an example that we will actually use in §6.1, suppose we are only able to obtain n "restricted" samples of $\Pr_\theta$. The i^{th} sample, e_i, was chosen randomly not from all of Events, but only from a restricted subset $E_i \subseteq \mathsf{Events}$. That is, it was chosen from $\Pr_\theta(\cdot \mid E_i)$. Then replace equation (3.3) with the **conditional likelihood**

$$\Pr(\text{evidence} \mid \theta) = \prod_{i=1}^{n} \Pr_\theta(e_i | E_i) = \prod_{i=1}^{n} \frac{\Pr_\theta(e_i)}{\sum_{e \in E_i} \Pr_\theta(e)} \tag{3.12}$$

3.2 Transformation Models

This short section formally defines transformation models. (Some readers may prefer the brief matrix presentation in §4.1.) An example is shown in Fig. 1.2 and Fig. 1.3, and the next section (§3.3) offers some simple intuitions.

3.2.1 Specification of a Transformation Model

A (log-linear) transformation model is specified by a tuple

$$\langle \mathsf{Events}, \textsc{Start}, \textsc{Halt}, \mathsf{Features}, \mathsf{Arcs} \rangle$$

where

- Events is the (finite or infinite) set of events whose probabilities are of interest,

- $\textsc{Start} \in \mathsf{Events}$ is a distinguished event,
- $\textsc{Halt}$ is a distinguished object not in Events,
- $\mathsf{Features} = \{t_1, t_2, \ldots t_k\}$ is an arbitrary set of finite size k, whose elements are known as **features**,
- $\mathsf{Arcs} \subseteq \mathcal{P}(\mathsf{Events} \times (\mathsf{Events} \cup \{\textsc{Halt}\}) \times \mathcal{P}(\mathsf{Features}))$ is a set whose elements are known as **arcs**.[3]

The model may be loosely regarded as a **transformation graph**. This is a directed multigraph[4] whose edges are labeled and whose vertex set is $\mathsf{Events} \cup \{\textsc{Halt}\}$. The edges are given by Arcs: the triple $\langle e, e', F\rangle \in \mathsf{Arcs}$ represents a directed edge from e to e' labeled by the feature set $F \subseteq \mathsf{Features}$. $\textsc{Start}$ and $\textsc{Halt}$ are distinguished source and sink nodes.

We ordinarily impose two additional conditions on a transformation model:

- **Finite-fanout property**. There are only finitely many directed edges from any vertex e.
- **Coaccessibility property**. If there is a path from $\textsc{Start}$ to a vertex e, then there must also be a path from e to $\textsc{Halt}$. (Equivalently, all accessible vertices of the multigraph must be co-accessible.)

The next section will assign probabilities to the arcs. The finite-fanout property ensures that the probabilities are well-defined. The coaccessibility property helps ensure that a random walk from $\textsc{Start}$ on the transformation graph reaches $\textsc{Halt}$ in finite time with probability 1.[5]

[3] This condition and some of the notation below imply that Arcs is a set. In practice we relax this to a multiset, i.e., there is no objection to Arcs containing duplicates if convenient. The set notation is merely more familiar and easier to follow.

[4] A multigraph is a slight generalization of a graph. Given two vertices e, e', a graph has at most one edge from e to e'. A multigraph may have multiple such edges (perhaps labeled differently). Allowing multiple edges is only a matter of convenience and adds no extra power to the formalism: see footnote 1 on p. 114.

[5] The coaccessibility property is necessary for random walks to halt with probability 1. In the case of finite graphs the coaccessibility property also turns out to be sufficient. For infinite graphs it turns out not to be sufficient or even, in general, computable (see footnote 7 on p. 90), although models for real applications can generally be shown to have the desired properties. In any case the issue is moot in practice, since one must use approximation algorithms (especially for infinite graphs) that consider only a subset of possible paths.

3.2.2 The Parameterized Probability Distribution Defined by a Transformation Model

The above transformation model yields a parameterized probability distribution $\Pr_\theta$ over Events, as follows.

The parameter space Θ is simply $\mathbb{R}^k$, where $k = |\mathsf{Features}|$. Given a length-k real vector $\theta \in \Theta$, first define $G_\theta : \mathsf{Arcs} \to \mathbb{R}^+$ to assign a positive ***G*-value** to every edge in the transformation graph:

$$G_\theta(\langle e, e', F\rangle) \stackrel{\text{def}}{=} \exp(\sum_{i:t_i \in F} \theta_i) > 0 \tag{3.13}$$

For example, an edge labeled by the feature set $F = \{t_3, t_8, t_{21}\}$ would have a G-value of $e^{\theta_3+\theta_8+\theta_{21}}$. The parameter θ_i is called the **weight** of the feature t_i. If several edges bear feature t_i, their G-values rise and fall together as θ_i rises or falls. (The variable G is meant to suggest conductance, an analogy developed in the variant model of §7.3.4.3. However, the G-values in the present section do not behave quite like conductances.)

Now define a normalized version $P_\theta : \mathsf{Arcs} \to (0, 1]$ that assigns a **transition probability** to each edge, by scaling the G-values such that the edges leaving each vertex have total probability of 1:

$$\mathsf{competitors}(\langle e, e', F\rangle) \stackrel{\text{def}}{=} \{\langle e, e'', F'\rangle : \langle e, e'', F'\rangle \in \mathsf{Arcs}\} \tag{3.14}$$

$$P_\theta(A) \stackrel{\text{def}}{=} G_\theta(A) / \sum_{A' \in \mathsf{competitors}(A)} G_\theta(A') \tag{3.15}$$

(The purpose of the finite-fanout property (§3.2.1) is to ensure that A has only finitely many competitors, so that the sum in equation (3.15) is finite.)

In other words, the parameter vector θ defines for each vertex e a log-linear probability distribution P_θ over the arcs leaving e. Features with positive (or negative) weights raise (or lower) the transition probabilities of the arcs on which they appear.

(Such conditional log-linear distributions (here conditioned on e) also play a central role in the maximum-entropy framework that has lately become popular in statistical natural language processing. We will take advantage of the connection in §8.2).

Next let us extend P_θ over paths in the multigraph.[6] A **path** is a finite-length vector

$$\vec{A} = \langle\langle e_0, e_1, F_1\rangle, \langle e_1, e_2, F_2\rangle, \ldots \langle e_n, e_{n+1}, F_{n+1}\rangle\rangle$$

where each $e_i \in \mathsf{Events}$ and each $A_i \stackrel{\text{def}}{=} \langle e_{i-1}, e_i, F_i\rangle$ is in Arcs. The probability of the path is defined as the product of its component arcs' probabilities:

$$P_\theta(\vec{A}) \stackrel{\text{def}}{=} \prod_{i=1}^{n+1} P_\theta(A_i) \tag{3.16}$$

Note that path probabilities are strictly positive. If $e_0 = \textsc{Start}$ and $e_{n+1} = \textsc{Halt}$, then the path $\vec{A}$ is said to **halt at** e_n.

All this notation lets us define the parameterized probability distribution specified by the model. For $\theta \in \Theta$ and $e \in \mathsf{Events}$, *we define $\Pr_\theta(e)$ to be the total probability of all paths that halt at e*:

$$\Pr_\theta(e) \stackrel{\text{def}}{=} \sum_{\vec{A} \in \{\text{paths that halt at } e\}} P_\theta(\vec{A}') \tag{3.17}$$

Some variants are discussed in §7.3.

As usual for a parameterized family of distributions, we will also want a prior distribution $\Pr_{\text{prior}}(\theta)$ over the parameters; see §3.5.

3.3 Some Simple Intuitions About Transformation Models

How should one think about transformation models? Here we give the most important intuitions, in terms of random walks on a graph. (For the reader who prefers other formalisms, §7.1 gives connections to other familiar mathematical devices: finite-state machines, Markov models, neural networks, graphical models, and Bayesian backoff.)

3.3.1 An Interpretation Using Random Walks

Sampling an event from $\Pr_\theta$ can be regarded as describing the outcome of a random walk on the transformation graph. The random walk is a stochastic process that defines a sequence of events $\vec{e} = e_0, e_1, e_2, \ldots$:

[6] In this thesis the terms "path" and "walk" (as in "random walk") are both taken to allow cycles. This follows the usage of "path" in automata theory (as contrasted with graph theory, where paths are usually defined to be acyclic walks). Here the two terms are essentially synonymous, although we tend to use "path" when the representation is a list of edges, and "walk" when it is a list of vertices.

1. Set $i = 0$. Let $e_0 = \text{START}$.

2. Randomly choose an arc of the form $\langle e_i, e', F \rangle$—that is, an arc leaving e_i. The probability of choosing a given arc A is $P_\theta(A)$ as defined by equation (3.15).

3. If $e' = \text{HALT}$, then return e_i as the sample. Else:

4. Let $e_{i+1} = e'$.

5. Increment i and return to step 2.

Note that the probability distribution over arcs at step 2 is a log-linear distribution determined by θ and the arc features. $\Pr_\theta(e)$ may be regarded simply as the probability that the walk halts "at vertex e," that is, immediately after generating e.

To put this another way, we may regard Θ as a random variable that indirectly determines the arc probabilities. $\vec{E} = \langle E_0, E_1, \ldots E_N \rangle$ is a random variable describing the random walk; note that the walk's length N is also a random variable. Then $\Pr(\theta, e)$ is the probability that the random variable $\langle \Theta, E_N \rangle$ has value $\langle \theta, e \rangle$. (See §3.1.2.) Conditionalizing this distribution yields $\Pr_\theta(e) \stackrel{\text{def}}{=} \Pr(e \mid \theta)$ as desired.

(On whether the random walk is guaranteed to halt with probability 1, see §3.2.1, especially footnote 5.)

3.3.2 Random Walks as Transformational Processes

The model is called a *transformation* model because of the following interpretation of the random walk. (See §2.3.5 for the linguistic relevance.) The stochastic process begins at START and randomly chooses an event e_1. It may halt there and output e_1, or it may randomly decide to *transform* e_1 into some e_2. In the latter case, it may halt and output e_2, or it may decide to transform it further into e_3, and so forth.

The topology of the model (i.e., Arcs) determines which actions are *available* when the current event is e_i (namely, halting and various transformations). Meanwhile, the *probabilities* of those actions are specified by P_θ, and hence determined by the parameter vector θ together with the features F associated with each action, according to a log-linear model.

Note the Markovian independence assumption here: the final event is produced by a sequence of *independent* transformations. The actions available when the current event is e_i, and their probabilities, depend only on e_i. They do not otherwise depend on the previous transformations that produced e_i in the first place. (Of course, the transformation model may be designed so that e_i encodes some or all of this history, at the cost of increasing the size of the event space Events. Recall that this space can be infinite.)

3.4 Solving a Transformation Model

To **solve** a transformation model means to find the distribution $\Pr_\theta$, given θ. (This is easier than the separate problem of estimating θ from evidence.)

Even with a finite transformation graph, infinitely many paths may halt at e (thanks to cycles). Nonetheless, it is straightforward here to solve for their total probability $\Pr_\theta(e)$, given θ.

For any node e, let $I_\theta(e)$ denote the **flow to** e. (The variable I is meant to suggest current, an analogy developed further in §7.3.4.3.) This is defined to be the total probability of all paths from $\textsc{Start}$ to e. Equivalently, it is the expected number of times e is generated during the random walk: that is, the number of times that the walk visits vertex e.

The following recurrence relates $I_\theta(e)$ to the values $I_\theta(e')$, where e' ranges over the parents of e in the transformation graph:

$$I_\theta(e) = \sum_{A=\langle e',e,F\rangle \in \mathsf{Arcs}} I_\theta(e') \cdot P_\theta(A) + \delta(e = \textsc{Start}) \tag{3.18}$$

where $\delta(\phi)$ has value 1 if ϕ is true and 0 otherwise.

Each vertex e in the transformation graph contributes one variable $I_\theta(e)$ and one linear equation that is an instance of equation (3.18). The linear equations may be solved simultaneously for the $I_\theta(e)$ values in cubic time, using standard matrix methods (see §4.1 for details). Then for any arc $A = \langle e, e', F\rangle$, we can define the **flow along** A:

$$I_\theta(A) \stackrel{\text{def}}{=} I_\theta(e) \cdot P_\theta(A) \tag{3.19}$$

which is the expected number of times that the random walk traverses arc A.

Now we can write

$$\begin{aligned} \Pr_\theta(e) &= \sum_{A=\langle e, \text{HALT}, F\rangle \in \mathsf{Arcs}} I_\theta(A) \\ &= I_\theta(e) \cdot \sum_{A=\langle e, \text{HALT}, F\rangle \in \mathsf{Arcs}} P_\theta(A) \end{aligned} \tag{3.20}$$

This sum is the total flow from e to Halt. It describes how often the random walk reaches e and then halts immediately via some arc from e to Halt—which is the definition of $\Pr_\theta(e)$, as desired.

For an infinite transformation graph, solving the above recurrence for $\Pr_\theta(e)$ may be harder or even uncomputable, depending on the structure of the model.[7] But there are general techniques for approximate solution, discussed in §4.2 and §8.5.

3.5 Priors For Transformation Models

The previous sections discussed the distribution $\Pr_\theta$ when the parameter vector θ is known. (Recall that $\theta = \langle \theta_1, \theta_2, \ldots \theta_k\rangle$ specifies weights for the features $t_1, t_2, \ldots t_k$ that appear on the various arcs.) This section discusses the prior distribution $\Pr_{\text{prior}}(\theta)$, which helps us estimate θ given sparse evidence.

A natural prior for the parameter vector θ of a transformation model is specified in terms of a variance σ^2. We simply say that the weights $\theta_1, \theta_2, \ldots \theta_k$ are independent samples from the normal distribution with mean 0 and variance σ^2:

$$\Theta \sim \underbrace{N(0,\sigma^2) \times N(0,\sigma^2) \times \cdots \times N(0,\sigma^2)}_{k} \tag{3.21}$$

[7]Uncomputability is not surprising since the approach is designed to be able to capture lexical redundancy rules. Uszkoreit and Peters (1985) famously showed that context-free grammars augmented with simple lexical redundancy rules are r.e.-complete. Carpenter (1991) found similar results for categorial and HPSG grammars.

Here, uncomputability shows up by reduction from the halting problem. Given a deterministic Turing machine and an input, define Events as the infinite set of $\langle$state, tape contents$\rangle$ pairs of the TM. Each event has a single child, possibly Halt, that represents the next configuration of the TM. The single child of Start is the initial configuration $\langle$initial state, input$\rangle$.

The coaccessibility property holds iff the TM halts on this input, so is uncomputable in general.

Now we will see that even if the coaccessibility property is known to hold, it is uncomputable to determine $\Pr_\theta$. To see this, modify the previous transformation graph so it satisfies the coaccessibility property: Replace Halt with a new event H, and make Halt the second child of every event. Notice that no arcs have any features, so both children of an event have probability 1/2. Now $\Pr_\theta(H)$ is 2^{-n} iff the TM halts on this input in n steps. Thus it is uncomputable even to determine whether $\Pr_\theta(H)$ is positive.

or equivalently, that θ is drawn from the multivariate Gaussian with mean $\vec{0}$ and diagonal covariance matrix $\sigma^2 I$:

$$\Theta \sim N(\vec{0}, \sigma^2 I) \tag{3.22}$$

That is, for $\theta \in \Theta$,

$$\Pr(\theta) = (2\pi\sigma^2)^{-k/2} \exp -(\sum_{i=1}^{k} \theta_i^2 / 2\sigma^2) \tag{3.23}$$

$$\ln \Pr(\theta) = -||\theta||^2 / 2\sigma^2 + \text{const} \tag{3.24}$$

This prior says that *each weight is expected to be near zero.* In other words, each feature is *a priori* unlikely to have much influence on the probabilities of the arcs that bear it.

The smaller the variance σ^2, the stronger the prior, and the more evidence is necessary to override the prior's preference for a given weight to be close to zero. This prevents overfitting, i.e., choosing strongly positive or negative weights simply to fit one or two examples. Thus the prior is responsible for feature selection, effectively turning off any features that do not sufficiently help fit the data.

As a shorthand, we can say that the prior favors **sparse** parameter vectors θ—those in which most of the components are zero or at least close to zero. More precisely, it favors *short* parameter vectors, since equation (3.23) falls off with exponentially with the (squared) Euclidean length of the vector θ.

This prior has already been proposed (originally by John Lafferty) for the feature weights used in conditional maximum-entropy models (Chen and Rosenfeld, 1999). As already hinted in §3.2.2, our transition probability function P_θ is really a conditional maximum-entropy model. We will take advantage of this connection later to obtain an EM algorithm for parameter estimation (§8.2.1).

We will assume this prior unless otherwise stated. Some other possible priors are offered in §7.3.2.

3.6 Per-Event Features and Output Features

3.6.1 Per-Event Features

A particularly useful architecture for a transformation model includes **per-event features**. In a model with per-event features, for every $e \in \mathsf{Events}$ there is an associated feature $t_e \in \mathsf{Features}$ with weight θ_e. This feature appears on *all* arcs that lead *to* e. Formally, if $\langle e', e, F\rangle \in \mathsf{Arcs}$, then $t_e \in F$.

Such a model may include other features as well. But while the other features typically capture generalizations, the per-event features serve to capture exceptions. Increasing θ_e by $\pm \ln 2$ serves, roughly, to double or halve $\Pr(e)$, respectively.[8] (See §2.3.6 for motivation.)

However, notice that increasing θ_e by $\ln 2$ will roughly double not just $\Pr(e)$ but $I(e)$ (defined in §3.4 above as the "flow" to e). The effect of this extra flow is not limited to e. It is partitioned among e's outgoing arcs, not just its HALT arc, and thereby flows through to e's descendants in the transformation graph. We will see shortly why this is useful.

(The increase in the probabilities of this complex of events comes at the expense of e's parents and siblings in the graph, and *their* descendants.)

If a transformation model has per-event features, it has more parameters than data, and can fit nearly any probability distribution.[9] Indeed, it can do so in multiple ways. Multiple θ vectors will exist such that $\Pr_\theta$ is the maximum likelihood distribution given the observed data. So will other θ vectors such that $\Pr_\theta$ is a smoothed distribution. As always, the prior chooses among all these θ vectors, favoring those that are "sparse."

In essence, the per-event features make the model sufficiently tunable that we can capture any idiosyncrasies that are sufficiently well-observed that we need to capture them. It is as if a little knob θ_e is attached to each event e so that we can match its predicted probability to its observed probability. But a crucial feature of transformation models—unlike, say, multinomial models—is that tuning a single event through its knob will also

[8] The actual effect may be less, since the probabilities of arcs to e (especially if large) will not quite double or halve. The numerator of equation (3.15) does double or halve for each such arc A, but so does one summand in the denominator. On the other hand, this consideration can be outweighed by the feedback if e is one of its own descendants. Such feedback can sometimes magnify the effect of θ_e enough that increasing it by $\pm \ln 2$ *more* than doubles or halves $\Pr(e)$.

[9] For this reason it is usually not necessary to specialize further to per-*arc* features. Having a separate feature on every arc would really introduce far too many parameters for comfort, even though it is possible to construct artificial data that motivate such targeting. (In the lexicon smoothing application, per-arc features would allow a language θ to make a particular transformation target one specific lexical entry.)

tune the correlates of that event. (That is, the knobs at the event's children automatically turn as well, in proportion to the probability of the parent-child arcs. This turns the grandchildren's knobs, and so forth.)

While those correlates could then be "untuned" by turning the children's knobs in the opposite direction, this has a greater prior cost (imagine that all knobs have friction) and so would require additional evidence. The default is therefore to preserve the correlations.

(In the terms of §2.3.6, we can list any event, and regular processes will derive new events from the listed one.)

So any correlations captured by more widespread features remain important even in a model with per-event features. If a feature t_0 appears on many arcs, then giving it a high weight θ_0 captures the fact that the parent and child for each such arc tend to covary. If something (even a per-event weight) tunes the parent upward, then the corresponding child is *a priori* likely to follow at a rate determined by θ_0. That is, it is *a priori* unlikely that a given child will itself be tuned far away from following its parent, since the prior says $\theta_{\text{child}} \sim N(0, \sigma^2)$.

The qualitative effects of per-event features are further examined in §3.7.2 and §3.8.

3.6.2 Output Features

Output features are a generalization of per-event features. They can tune the probabilities of classes of events, not just single events.

A particular output feature appears on a transformational arc from e to e' just if the "output" of the transformation, namely e', has some particular property.

3.7 A Sample Application

Before turning to the detailed behavior and manipulation of transformation models, it may help to sketch the model of the lexicon that will be used in the thesis. More sample applications are sketched in §7.2, which the reader is free to glance at now.

3.7.1 What Applications Are Appropriate?

What do the sample applications below and in §7.2 all have in common? In each domain, events that are "related" in certain ways tend to have positively covarying probabilities. By choosing the edges and features of the transformation graph, a domain expert can describe possible patterns of relations among the events. Then feature weights θ can be estimated from actual data. By assigning the arcs large or small probabilities, these weights indicate which *kinds* of related events really do tend to covary in the data—i.e., which of the possible relations actually do predict high covariance.

The correlations have a causal interpretation, which is why the transformation graph has *directed* edges (as in Bayesian networks: see §7.1.4). Suppose we fix just the weights on the arcs to and from event e, but allow other weights to vary randomly according to the prior. Then the variance in $\Pr_\theta(e)$ is explained by the total variance at its parents (not its children). A consequence is that for e to be probable under a given choice of θ, all of its correlated children—but perhaps only one of its correlated parents—need be probable. In other words, only one cause of an event need be present for the event to occur and trigger all its effects.

§1.2.3 argued that transformation models are more appropriate than other kinds of models for capturing this kind of covariance.

It is worth remarking that transformation models are designed to capture only positive correlations, not negative ones. (Negative *weights* push covariance toward 0, not $-\infty$, since they push arc probabilities toward 0.) Of course, since probabilities must sum to 1, it is true that the covariance between unrelated events will be slighly negative. But these models cannot capture any more negative covariance than that.[10] They are suited to situations where events cause or transform into one another, not to "winner-take-all" situations where events inhibit one another (i.e., where a high probability for event e implies an especially low probability for event e'). Examples of the latter are mentioned in §7.2.2 and in footnote 18 on p. 167.

[10] Actually this is not *quite* true. Siblings compete for a share of the probability arriving at their common parent. If the parent's probability is relatively invariant with respect to θ, the siblings therefore do covary negatively over different values of θ, inded more negatively than pairs of unrelated events. But if the parent's probability itself varies widely, as is usual, then the effect is outweighed by the siblings' positive covariance with the parent and hence with one another. See also §3.9.

3.7.2 Syntactic Lexicons

The motivating example of this thesis is the syntactic lexicon of a language. (See Table 1.4 and Chapter 2.) The events are possible lexical entries, over which $\Pr_\theta$ specifies a probability distribution.

Put another way, $\Pr_\theta$ specifies the lexicon of a particular language. It says which of the universally possible entries are common in the language and which ones are extremely unlikely (i.e., probably ungrammatical, though available in a pinch). Different parameters θ yield different languages $\Pr_\theta$. The rest of this section explains how.

3.7.2.1 The Lexicon's Transformation Graph

In the transformation model $\Pr_\theta$, the arcs are instances of linguistic transformations that might relate lexical entries for the same word. Whether they *do* relate the entries depends on whether θ assigns them high probabilities. A random walk in the transformation graph chooses an initial version of a lexical entry by following an arc from START, and passes this initial form through a path of zero or more randomly chosen transformations (arcs) before choosing to halt. Each transformation arc's probability depends on θ together with the arc's features.

Broadly speaking, arcs have similar features if they serve to apply similar linguistic transformations in similar contexts. A transformation arc's features describe salient properties of the transformation, such as the type of transformation (`PP` adjunction) and the syntactic context in which it is being applied (at the right edge of an intransitive `S`). Arcs with similar features are *a priori* likely to have similar probabilities.[11]

To capture the data at the granularity of a linguistic theory, a complicated feature space (including disjunctive features) would probably be necessary, in order to ensure that any language's clear-cut transformations are linearly separable from its clear-cut non-transformations. Without linear separability in the feature space, no log-linear model can

[11] Indeed, for arcs to have similar probabilities, it is typically enough for them to share their "broad" features. Features that are more specific (selective) tend to have weights close to zero. This is because they appear on few arcs. Unless there is a great deal of relevant evidence $\vec{e}$ compared to the strength of the prior ($1/\sigma^2$), then even the few such features that are genuinely predictive are unlikely to have enough explanatory power to overcome the prior of §3.5. This is why we say that the prior implicitly does feature selection.

capture this distinction exactly—although it may do a passable job.

The model also has per-event features (§3.6.1), which by definition are specific to single lexical entries, headwords and all. But otherwise, the arcs relating lexical entries for word w have exactly the same features as the arcs relating the corresponding lexical entries for word w'. That is, transformations on a lexical entry are not sensitive to the entry's headword. (§4.5.3 will use this fact to speed up parameter estimation.)

(See §6.4 for more details of the model, in particular, the arcs from START and to HALT and their features.)

3.7.2.2 How Per-Event Features "List" Lexical Entries

It is the per-event features that allow words to differ from one another. They allow individual entries to be tuned to their observed frequencies, to the extent that these can be determined from the data. (The prior discourages overfitting if they cannot be determined well.)

Stipulating that a particular lexical entry's per-event weight is strongly positive has the effect of "listing" that entry as having a high probability (§2.3.6). Stipulating a strongly negative weight "delists" the entry, driving its probability close to zero.

As motivated in §2.3.6 and explained in §3.6.1, these effects are multiplicative, so that it is easier to idiosyncratically (de)list an entry that already has non-negligible probability.

What makes the model attractive is that if an entry is listed (or delisted), transformations continue to apply to it at the usual rate. So as discussed in §3.6.1, listing an entry has an effect that flows through the graph: it also increases the probabilities of derived entries for the same word. The prior makes it relatively cheap to make all these increases at once, since they are effected simultaneously by increasing a single weight.

For example, given enough evidence to list word w as a transitive verb, the prior encourages us to also list derived entries for word w, such as passive and extracted forms, with appropriate probabilities. The optimal lexicon (equation (3.4)) will do so unless the data strongly suggest other probabilities for the derived entries.

θ specifies the lexicon by defining which transformations, listings, and delistings are common in the language. If θ itself is *a priori* likely (§3.5), then $\Pr_\theta$ is by definition a likely language. Assuming the prior of §3.5, this is so iff θ uses relatively few large weights (including per-event weights). The language's lexicon can therefore be described by stipulating a small number of entries and transformational principles.

3.7.2.3 How Multiplicative Variation Encourages Transformations

§2.3.6 sketched why transformations were encouraged by our multiplicative cost model of listedness, in which increasing the per-event weight θ_e by $\ln c$ roughly *multiplies* $\Pr(e)$ by c. Let us now work out an example in the lexicon smoothing domain.

When the lexical entry $\texttt{S} \to \texttt{NP}\ w$ is listed for some word w (an intransitive verb), so is the entry $\texttt{S} \to \texttt{NP}\ w\ \texttt{PP}$. Suppose that that without listing, their probabilities given w are $(0.2, 0.1)$.

In a stochastic lexicon where the second entry is entirely derived from the first by PP-adjunction,[12] all 0.3 of the probability initially flows into the first entry. 0.2 of this flow continues along the arc to HALT, and the remaining 0.1 flows to the second frame and then to HALT. (There may also be additional flow to the first frame that ends up elsewhere.)

The two entries tend to be listed together. When w is an (intransitive) verb, they both have higher probability given w—say 50% higher for $w =$ `rise`, i.e., $(0.3, 0.15)$. In a stochastic lexicon $\Pr_\theta$ with the PP-adjunction transformation, it is only necessary to list the first entry, $\texttt{S} \to \texttt{NP rise}$. Raising its per-event feature by about $\ln 1.5$ will increase the flow to it from 0.3 to 0.45, thereby increasing the probabilities of both frames. In a competing lexicon $\Pr_{\theta'}$ that has no such transformation, the per-event features of *both* events must be raised by $\ln 1.5$.

The extra per-event weight specified by θ' reduces the second lexicon's log-probability (by $(\ln 1.5)^2/2\sigma^2$). The first lexicon's log-probability is also somewhat reduced by the need for θ to specify extra weights of its own to encode the transformation; but the benefit of these weights is amortized over all the intransitive verbs w. If there are enough such verbs, the prior favors the first, transformational lexicon, $\Pr_\theta$.

Under an additive cost model of listedness, the transformational lexicon would have no such advantage. The first lexicon adds 0.15 to the first entry and then shifts some

[12]The opposite transformation (PP-deletion) is equally plausible in this example. When more entries are involved, it is possible to detect causality and so distinguish the two transformations: see §3.7.1.

of it by transformation; the second adds 0.1 and 0.05 to the two entries separately, for the same total change. Under an additive cost model these would be equally cheap. The first lexicon would therefore be *disfavored* because of the extra prior cost of specifying the transformation. It is only under the multiplicative cost model that scaling the total flow of 0.3 to 0.45, "all at once," constitutes an economy of scale that lets the first lexicon win.

3.7.2.4 Per-Event Features vs. Features on START Arcs

One might be tempted to entertain an alternative scheme for listing lexical entries. Instead of placing the special per-event feature t_e on *all* arcs to event e, one might consider placing it only on the arc from START to e.

Then different words would select for different *initial* frames, which would then be transformed at a fixed rate regardless of headword. To list or delist a derivable entry like `S` $\rightarrow$ `NP rise PP`, the learner would have to manipulate just the probability of starting with that entry (the arc from START) rather than the probability of deriving it (the arc from `S` $\rightarrow$ `NP rise`). The start probabilities would be the only locus of variation.

There are reasons to avoid this alternative. Suppose the entry e is frequently derived but has a start probability close to zero. Then under this scheme, only a drastic increase in θ_e would have much effect on $\Pr(e)$, and even a drastic decrease in θ_e could reduce $\Pr(e)$ by at most the start probability. Yet in practice, the probability of such derived entries seems to vary rather freely. To consider two canonical examples:

- An entry of the form `S` $\rightarrow$ `NP` w `NP PP` is presumably derived from `S` $\rightarrow$ `NP` w `NP`. But the relative probabilities of these two entries vary substantially among verbs w, according to whether w provides a useful role for a `PP` argument. (For example, `S` $\rightarrow$ `NP` w `NP PP` has increased probability for verbs of movement such as `move`, `push` or `bring`.)

- Passive-verb entries also presumably fall into this category, as we would like to regard each as derived from an active entry for the same verb. But a few such entries are delisted: `let` does not appear in the passive (§2.3.1).

Merely adjusting the start probabilities cannot delist passive `let` (without also delisting active `let`), since its start probability is already close to zero. There is no way to block or weaken the transformation that yields most of passive `let`'s probability.

Both examples would militate against positing strong transformations in the first place. Derived entries would be permitted little variability under this alternative scheme, so to explain the variability in the example entries above, we would have to abandon the idea that they are usually derived. Per-event features seem preferable.

Another alternative approach would be to place the feature t_e only on the arc from e to HALT. Raising the weight of t_e would then increase $\Pr(e)$. However, it would do so at the expense of e's descendants in the transformation graph: listing e would tend to delist its children. This anti-correlation seems to be the opposite of the empirical fact (§2.4.2.1), which is that transformationally related lexical entries are *positively* correlated.

3.7.2.5 How Output Features Allow Structure Preservation

Structure Preservation (mentioned in §2.3.6) is the old observation that transformations in a language tend to produce structures, or types of structures, that are already common in that language for other reasons. This suggests a conspiracy among one or more transformations and listings to produce the same outputs.

How to specify the model so that the prior encourages such conspiracies? §2.3.6 noted that if the model has per-event features, a transformation that outputs the exact entry e' necessarily makes it cheaper to list e'. This might explain some conspiracies between transformation and listing. But it cannot explain why two English transformations discussed in §2.4.3—passivization and `PP`-adjunction—conspire *with each other*, both producing frames that contain $[_{\texttt{PP}}$ `by NP ]`.

The observation of §2.3.6 is also incomplete for another reason. Structure Preservation in practice often involves conspiracies of multiple mechanisms to produce outputs that are merely similar, not identical. For example, the lexically specific English "tough-movement" transformation turns *It is cruel to eat babies* into *Babies are cruel to eat*. The resulting frame for `cruel` is syntactically the same as the frame for `tender` in *Babies are tender to eat*, but the two lexical entries are not exactly the same. They have different headwords and also different semantics: the latter is semantically related to *Babies are tender*, rather than to *It is tender to eat babies.*

Output features provide a reasonable mechanism for modeling these conspiracies. In English, a transformation's probability is increased if it outputs an entry that contains $\texttt{[}_{\texttt{PP}}$ $\texttt{by NP]}$, or an entry of the form $\texttt{S}_{\texttt{stem}} \rightarrow \texttt{NP be}$ —— $\texttt{S\textbackslash NP/NP}$.[13]

3.8 Qualitative Behavior of Transformational Smoothing

This section carefully examines a simple transformation model, in order to demonstrate some of the properties, virtues, and vices of the approach:

- The transformation graph ensures that related events have related probabilities.
- Feature weights far from 0 correspond to stipulated generalizations or exceptions.
- When we estimate those feature weights in a semi-Bayesian way (§3.1.3), our prior (§3.5) enforces Occam's Razor: namely, stipulate as little as necessary to explain the observed probabilities.
- If those "observed" probabilities are not known precisely because of sparse evidence, they too are estimated. They are smoothed away from the maximum-likelihood model toward a model that has as little stipulation as possible.
- In the example, the smoothing backs off from the observed (maximum-likelihood) probability of event e by considering events related to e. More precisely, if e and e' are transformationally related events, then the smoothing adjusts the *ratio* $\Pr(e)/\Pr(e')$ toward the corresponding ratio for other pairs of events related by the same transformation. The scheme resembles traditional interpolated backoff, but
  - The backoff estimate is more like a type-weighted than a token-weighted average of the related ratios. (In fact it is a principled middle road.)

[13] Of course, these output features do not count for much by themselves. The transformation's probability remains negligible unless it also has various other features. But multiple features that encourage the transformation cooperate multiplicatively (equation (3.13)), including per-event features that encourage listedness of the output.

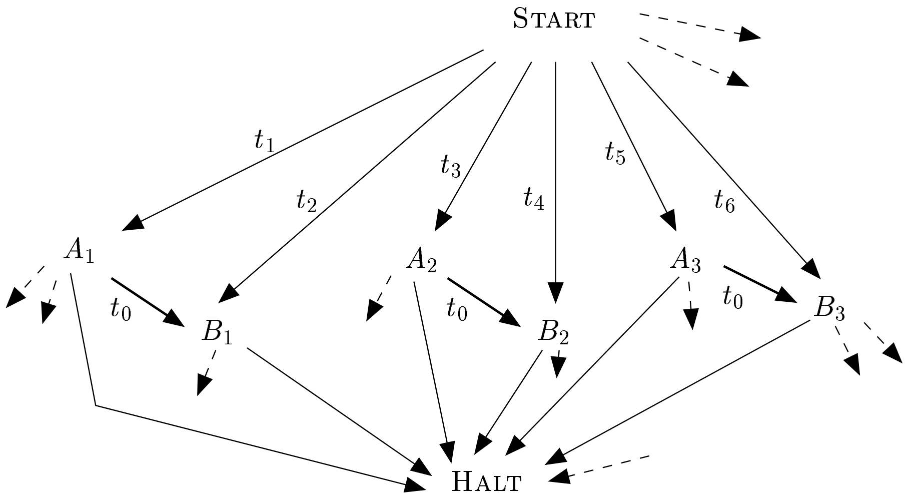


Figure 3.1: A canonical transformation model on the $2k$ events $(A_1, B_1, \ldots A_k, B_k)$. The transformations from START to A_i or B_i all have independent features, but the transformations from A_i to B_i all have the same feature.

- It can be arranged that if the related ratios vary widely, then the backoff estimate is less influential. Intuitively, such a backoff estimate is unreliable because the related events do not form a natural class.[14]

- Competing transformations are smoothed to have similar probabilities, absent sufficient evidence to differentiate them. This avoids overfitting and winner-take-all behavior.

- Optimizing the parameters unfortunately faces a pervasive problem of local maxima.

3.8.1 A Canonical Example

A sample transformation model is shown in Fig. 3.1. Recall that the vertices of this graph correspond to events. Not all features are shown on the arcs.

For the sake of the discussion, assume that if the feature weights $\theta_0, \ldots \theta_{2k}$ are 0, then the arcs shown have small probabilities, so the events $A_1, \ldots A_k, B_1, \ldots B_k$ are all unlikely. (In other words, it is unnecessary to drive a weight negative to effectively "turn off" its

[14] As usual, the backoff estimate is also less influential when smoothing a well-observed ratio.

arc.) This will be the case if each of the arcs we care about has additional weights that are strongly negative, or has competitors with strongly positive weights.

3.8.2 The Connection to Lexicon Smoothing

The example is a thinly disguised simplification of the lexicon-smoothing application. Imagine that each $i = 1, 2, \ldots k$ represents a different word. A_i is a lexical entry licensing the use of that word as an transitive verb, $\texttt{S} \to \texttt{NP}\ i\ \texttt{NP}$, and B_i is a related entry licensing its use as an intransitive one, $\texttt{S} \to \texttt{NP}\ i$. We have to learn the probability distribution over these $2k$ lexical entries.

The model would be uninteresting except for the A_i-to-B_i arcs. These encode a possible transformation that detransitivizes a verb. The model is designed so that the transformation applies at the same (possibly negligible) rate to all the A_i. The rate depends on θ_0 and might be negligible; part of our job is to learn it.

(Note: it is a pedagogical simplification for the transformation to operate at the same rate always. In a real model, the B_i entries would have per-event features that could vary the rate for different values of i. See §3.8.5 for discussion.)

3.8.3 The Effect of the Transformation Arcs

To the extent that θ_0 is large, evidence that increases the estimate of $\Pr(A_i)$ implies a proportionate increase in the estimate of $\Pr(B_i)$. Why? Because if there is a good chance that the random walk will reach A_i and halt, then there is a proportionately good chance (depending on θ_0) that it will reach A_i and continue to B_i and halt. Any new evidence about A_i "flows through" the graph to affect B_i (and nodes beyond).

To formalize this argument in the terms of §3.4, $\Pr_\theta(B_i)$ is a linear function of $\Pr_\theta(A_i)$, because the flow to B_i, $I_\theta(B_i)$, is a linear function of $I_\theta(A_i)$:

$$\begin{aligned}
\Pr_\theta(B_i) &= \text{const} \cdot I_\theta(B_i) && \text{(by equation (3.20))} \\
&= \text{const} \cdot (\text{const} + \text{const} \cdot I_\theta(A_i)) && \text{(by equation (3.18))} \\
&= \text{const} \cdot (\text{const} + \text{const} \cdot (\Pr_\theta(A_i)/\text{const})) && \text{(by equation (3.20))} \\
&= \text{a linear function of } \Pr_\theta(A_i)
\end{aligned}$$

Thus, raising θ_5 will not only make $\Pr_\theta(A_3)$ increase but will also make $\Pr_\theta(B_3)$ increase proportionately.[15]

The strength of this effect increases with θ_0. Conversely, our estimate of θ_0 will increase with the observed strength of the effect.

3.8.4 Fitting Regular Patterns

Suppose the evidence sample $\vec{e}$ includes many examples of both "A" and "B" events, and that for each i, the number of observations of B_i is about half the number of observations of A_i. Qualitatively speaking, there are at least two ways to fit the parameter vector θ so that these data look likely, i.e., so that $\Pr(\vec{e} \mid \theta)$ is large:

- **Coincidence:** Raise all the weights $\theta_1, \theta_2, \ldots \theta_{2k}$ so that all the A_i and B_i events become probable.

  This approach independently stipulates that each of the events is probable.

- **Correlation:** Raise just the weights $\theta_1, \theta_3, \ldots \theta_{2k-1}$, so that the events A_i become probable. Then raise the single weight θ_0 so that each token of A_i has a good chance, say $\frac{1}{3}$, of transforming into a token of B_i. This accounts for the 2-to-1 ratio of observations: for every 3 tokens of A_i, 2 halt at A_i and one transforms into B_i before halting.

  This approach independently stipulates that each A event is probable, and then stipulates a single additional rule transforming a fraction of A events into B events.

The first approach requires raising $2k$ parameters above zero. The second approach requires raising only $k + 1$ parameters. This means that the natural prior of §3.5 tends to favor the latter: it is *a priori* more probable to have only a small number of non-zero parameters. *In other words, the prior prefers the approach with less stipulation.*

When can the model discover the regular transformation required, which turns $\frac{1}{3}$ of A_i's into B_i's for each i? It is the model's topology that encodes the *possibility* of such a transformation: Arcs specifies that the A-to-B arcs exist and have a common feature t_0.

[15] Unless raising θ_5 substantially changes the constants in the linear function. This could happen if feature t_5 also appears on any of the arcs leaving A_3 or B_3.

Meanwhile, the model's prior is biased toward assuming that *some* regular transformation, rather than some large set of features, carries most of the explanatory burden of the grammar.

Of course, while the model's structure is therefore amenable to this transformation *a priori*, it does not by itself choose a transformation rate of $\frac{1}{3}$ (rather than 10^{-9}, say). That depends on our data-driven estimate of θ and especially θ_0. The evidence sample $\vec{e}$ favors $\theta_0 > 0$ to the extent that $\vec{e}$ supports a correlation between the $\Pr(A_i)$ and $\Pr(B_i)$ values—for example, if several pairs (A_i, B_i) are sufficiently well observed to establish their probabilities, and these pairs demonstrate that $\Pr(B_i)$ tends to increase linearly with $\Pr(A_i)$. The parameters can fit these $\Pr(B_i)$ values via either separate stipulation or a common transformation, and the prior favors the latter approach.

3.8.5 Fitting Exceptions

A smaller set of parameters is easier to estimate accurately from a sparse sample of evidence. In the above example, the second, transformational account requires only $k + 1$ non-zero parameters rather than $2k$.

Formally, however, the model has fully $2k + 1$ degrees of freedom $(\theta_0, \ldots \theta_{2k})$. Indeed, there are enough parameters in this case to model *any* distribution with arbitrary accuracy.[16] The prior merely discourages the use of the extra parameters by favoring values close to 0. The smaller σ^2, the greater the burden of evidence necessary to justify a weight of given magnitude. Thus, some parts of the distribution space are *a priori* less likely than others.

The additional degrees of freedom are still available, and may come into play in order to fit exceptions. If θ is chosen as described in the previous section, with θ_0 set so that the probability of a transformation is about $\frac{1}{3}$, then $\Pr(B_i) = \frac{1}{2}\Pr(A_i)$ for *every* $i = 1, 2, \ldots k$. But what if this ratio is not exactly the same for all i? If the evidence indicates that $\Pr(B_7) > \frac{1}{2}\Pr(A_7)$, say, then other weights must be adjusted to arrange this result:

[16] That is, $\{\Pr_\theta : \theta \in \Theta\}$ is dense in the space of all distributions over $\{A_1, B_1, A_2, \ldots\}$. In fact, something even stronger holds: any distribution can be *exactly* modeled so long as it assigns probability > 0 to every event. (To arrange $\Pr_\theta(B_7) = 0$ we would need at least one $-\infty$ weight on every path of the form START, $\ldots B_7$, HALT.)

- One could increase θ_{14} above 0. This stipulates that B_7 is probable, just as would be necessary if there were no A_7-to-B_7 arc, or if its probability were low. However, since that arc is already delivering a good deal of probability mass to B_7, θ_{14} need only be raised enough to pick up the slack.

  The arc-specific weight θ_{14} only needs to be large if $\Pr(B_7)$ is well *above expectation*. Its magnitude corresponds to the degree to which B_7 is exceptional or "listed" (in the sense of §2.3.6). The prior functions to discourage "listedness."

- Alternatively, one could increase the weight of some feature other than t_0 on the A_7-to-B_7 arc (or some other arc to B_7). This raises the probability of that arc. As a side effect, it also raises the probability of other arcs—if any—on which that feature occurs, and lowers the probability of their competitors. Whether these changes help or harm the overall likelihood of the evidence, $\Pr(\vec{e} \mid \theta)$, depends on whether the feature is indeed broadly predictive of "useful" transformations.

Using per-event features (§3.6.1) allows a transformation model to handle both "positive" and "negative" exceptions (that is, "listing" and "delisting") quite gracefully. In the present example model, as discussed in §3.7.2.4, "negative" exceptions are harder to handle. This is because t_{14} does not appear on *all* arcs to B_7 and so is not a true per-event feature for B_7. We cannot lower $\Pr(B_7)$ below $\frac{1}{2}\Pr(A_7)$ simply by manipulating θ_{14}. B_7 already gets very little probability from START, and making θ_{14} negative cannot reduce this probability below zero. The only solution in the example model is along the lines of the latter strategy above: decrease the weight of some other feature that is peculiar to the A_7-to-B_7 arc. (Or one could simply reduce θ_0 and treat all the other B_i as positive exceptions, but this would greatly reduce the prior probability of θ!)

3.8.6 The Smoothing Effect

The discussion so far has assumed that the exact probability of each A_i and each B_i is known from the evidence, and the weights must be adjusted to model them exactly.

But of course, the evidence sample is generally too small to get good direct estimates of event probabilities. Indeed, the point of transformational smoothing is to get better estimates!

As in the previous section, suppose there is evidence of an exception. We have estimated θ such that $\Pr_\theta(B_i) = \frac{1}{2}\Pr_\theta(A_i)$ for all i, but B_7 was observed *more* than half as often as A_7 in the data: $\#(B_7) > \frac{1}{2}\#(A_7)$.

The question for the model is whether this discrepancy at (A_7, B_7) is a real exception or can be written off to the vagaries of sampling. If it is real, then we should adopt one of the solutions of the previous section, and adjust the weight vector θ so as to exactly fit the observed frequencies of A_7 and B_7. But if it is a sampling error, we should *not* fit the observed frequencies, but rather smooth them, supposing that for the true distribution, $\Pr_\theta(B_7)$ really does equal $\frac{1}{2}\Pr_\theta(A_7)$.

Which behavior is selected by our estimation procedure? Fitting the observations exactly would reduce the prior $\Pr_{\text{prior}}(\theta)$ because it recruits additional weights. On the other hand, ignoring the observations would reduce the likelihood $\Pr(\vec{e} \mid \theta)$. The maximum-posterior estimate selects θ to maximize the product of both quantities (equation (3.4)), so it strikes a compromise. In other words, it smooths the observed count ratio *somewhat* toward the general 2-to-1 ratio.

How much smoothing happens depends—as always for a (semi-)Bayesian estimate—on the relative strengths of prior and likelihood. In particular, small counts are smoothed more, since a failure to fit them exactly makes less difference to the likelihood of the data.

Thus, suppose A_7 and B_7 were observed just once each in a sample of 40 events. The maximum-likelihood estimates of their probability would be $\frac{1}{40}$ and $\frac{1}{40}$. But since those estimates are supported by scant evidence, transformational smoothing would guess probabilities closer to $\frac{1}{30}$ and $\frac{1}{60}$ (totalling $\frac{2}{40}$, and in 2-to-1-ratio). Note the abductive inference here: transformational smoothing guesses that A_7 was *more* likely than in the sample, because this helps explain why we observed B_7. Smoothing the probabilities in this way does not badly reduce $\Pr(\vec{e} \mid \theta)$ since $\frac{1}{30} \cdot \frac{1}{60}$ is not much less than $\frac{1}{40} \cdot \frac{1}{40}$.

By contrast, if A_7 and B_7 were observed 10 times each in a sample of 400 events, then transformational smoothing would put their probabilities closer to $\frac{1}{40}$ and $\frac{1}{40}$, since there is more evidence here to motivate an exception. It is simply unlikely that a true 2-to-1 ratio would yield equal counts as large as 10 and 10: $(\frac{1}{30} \cdot \frac{1}{60})^{10}$ is several times less than

$(\frac{1}{40} \cdot \frac{1}{40})^{10}$.

Globally, the degree of smoothing can be controlled by setting σ^2. It should be small for strong smoothing (i.e., we expect weights to be small, so there are heavy penalties for modeling exceptions by stipulating large weights).

3.8.7 Type-Weighted vs. Token-Weighted Smoothing

The above section assumed that θ_0 is such that the probability of A_i-to-B_i arcs is $\frac{1}{3}$, so that $\Pr_\theta(B_i) = \frac{1}{2}\Pr_\theta(A_i)$ by default. But of course θ_0 is not handed down from the heavens to yield an arc probability of $\frac{1}{3}$. Together with the other weights, it is chosen to allow a reasonable fit to *all* the observed A_i and B_i counts. In other words, θ_0 is chosen by combining the various $(\#(A_i), \#(B_i))$ count pairs.

How much influence does $(\#(A_7), \#(B_7))$ have on this combined estimate? The higher these counts, the more we trust them and their ratio as accurate—up to a point of diminishing returns!—so the more important it is for θ_0 to fit them well. If these counts are small enough to be unreliable, then they have less influence on θ_0.

This is an attractive compromise between an weighted average of the $(\#(A_i), \#(B_i))$ data, where i with higher counts have *proportionately* more influence, and an unweighted average, where all i are treated equally. Essentially the same compromise was used by the Bayesian model of backoff proposed by MacKay and Peto (1995). We defer further discussion to §7.1.5, which will describe that related model and the connection to the present canonical example.

3.8.8 Effect of the Prior on Competing Arcs

A natural question about Fig. 3.1 concerns the Start node. Each of the $2k$ competing arcs leaving Start has its own single weight, which does not appear anywhere else in the transformation graph. In this canonical situation, does the prior prefer the competing arcs to have similar probabilities or different probabilities?

The latter case would be unfortunate from a practical standpoint:[17] the prior would

[17]Of course, such a prior (which is *not* used in this thesis) could still be appropriate from a modeling standpoint; this depends on the problem domain. In the linguistic application of §3.7.2, where the arcs from a node correspond to the transformations on a lexical entry, this would impose an *a priori* bias toward

be "U"-shaped and multimodal, further complicating optimization. Gradient ascent, for example, would function as a winner-take-all competition in which one arc from START would seize early advantage and suck most of the probability from its competitors. This would make the result of optimization even more sensitive to the starting point.

Fortunately, the prior we have chosen (§3.5) prefers the probabilities on the arcs from START to be as *similar* as possible. Suppose $\vec{p} = \langle p_1, \ldots p_{2k} \rangle$ is a probability distribution over the $2k$ arcs. Put $\theta_i = \ln p_i$. From equation (3.13) and equation (3.15), $\vec{p}$ must have arisen from the weight vector $\vec{\theta} - d$ for some $d \in \mathbb{R}$, where $(\vec{\theta} - d)_i \stackrel{\text{def}}{=} \theta_i - d = \ln p_i - d$.

Let n, μ and v respectively denote the length ($= 2k$), mean, and variance (with denominator n) of $\vec{\theta}$. (Or equivalently, of $\vec{\theta} - d$ for any d.)

The prior probability of $\vec{\theta} - d$ is

$$(2\pi\sigma^2)^{-n/2} \exp - \sum_i (\theta_i - d)^2 / 2\sigma^2 \tag{3.25}$$

Integrating this over all $d \in \mathbb{R}$, we obtain the total prior probability of $\vec{p}$, which works out to[18]

$$n^{-1/2} (2\pi\sigma^2)^{-(n-1)/2} \exp -nv/2\sigma^2 \tag{3.26}$$

In practice we will not consider this integral, since we only have enough compute power to do semi-Bayesian smoothing (see §3.1.3). The semi-Bayesian or maximum *a posteriori* approach seeks the *single* weight vector with maximum posterior probability. To this end, for any $\vec{p}$ we need only consider the corresponding $\vec{\theta} - d$ with the greatest prior probability, since a different value of d would lower the prior but leave $\vec{p}$—and hence the likelihood[19]—unchanged. Maximizing equation (3.25) over all $d \in \mathbb{R}$ (rather than integrating), we set $d = \mu$ and obtain

$$(2\pi\sigma^2)^{-n/2} \exp -nv/2\sigma^2 \tag{3.27}$$

obligatory or nearly obligatory transformations (and halts), including the "transformation" of START that chooses an initial lexical entry. More concretely, it would smooth rarely observed lexical entries toward probability 0, reallocating their observations to the most common entries. By contrast, we will now see that the prior of §3.5 reallocates observations in exactly the reverse way, from the most-observed to the least-observed events.

[18] The trick is to rewrite the exponential term of equation (3.25) as

$$\exp -v/(2\sigma^2/n) \cdot \exp -(d-\mu)^2/(2\sigma^2/n)$$

The first factor is independent of d, and the integral of the second factor is $\sqrt{2\pi\sigma^2/n}$.

[19] The likelihood of the data depends on the weights $\theta_1, \ldots \theta_{2k}$ through $\vec{p}$ alone, since by assumption none of the weights are used on other arcs.

Both equation (3.26) and equation (3.27) fall off exponentially as v increases. No other term in these equations depends on $\vec{p}$. Thus, regardless of whether we integrate or maximize over the free variable d, the prior always pushes us to adjust $\vec{p}$ so as to reduce v as much as possible. Remember that v is the variance of the log-probabilities on the arcs. It is small just if the arc probabilities $\vec{p}$ cluster tightly around their geometric mean.

More precisely, the prior falls off exponentially with $nv/2\sigma^2$. Thus, as usual, the prior's preferences are strongest when σ^2 is small. The appearance of nv rather than v ensures that a given outlier arc offends the prior equally no matter how many competitors it has. (v is the *average* squared difference of the log-probabilities from their mean, which would obscure the outlier for large n, but nv is the *total* squared deviation.)

For some applications, one might have a prior expectation that the probabilities of competing arcs will obey Zipf's Law. This can actually be arranged by a simple change to the prior: see §7.3.3.

3.9 Variation: Perturbed Transformation Models

Per-event features (§3.6.1) are not the only way to tune the probabilities of individual events. An alternative is to perturb the event probabilities directly. In fact, this is the strategy used in the experiments of Chapter 6.

3.9.1 Specifying the Perturbed Model

A **perturbed transformation model** is a transformation model with some additional parameters. For each event e that we wish to be tunable, there is an additional parameter π_e, known as a **perturbation**.

The value $\exp \pi_e$ serves as a **flow multiplier**: the idea is that the outflow of vertex e does not equal its inflow (compare §7.3.4.3), but rather equals its inflow times $\exp \pi_e$. In the linear programming community, this framework is known as **generalized network flow**.[20]

[20] The closest version of the problem is the single-source generalized shortest-paths problem, also known as the "restricted generalized uncapacitated transshipment problem" (Oldham, 1999). Given $\vec{\pi}$, a legal flow is one that respects the multipliers (except at the source and sink), and the classical problem is to find a legal flow that minimizes $\sum_A cost(A) \cdot flow(A)$. A Viterbi approximation to the E step of

The perturbation π_e is a replacement for the per-event weight θ_e. Their behavior is extremely similar:

- Both parameters are usually given a Gaussian prior of the form $N(0, \sigma^2)$.
- Either parameter can be used to "list" or "delist" the event e, in the sense of §2.3.6 and §3.7.2.2.
- Transformations apply in the ordinary way to listed events, regardless of whether it was θ_e or π_e that accomplished the listing: so the effect of listing e "flows through" the graph to affect e's descendants.
- Either parameter has a multiplicative effect on $\Pr(e)$. Increasing π_e or θ_e by $\ln 2$ roughly doubles $\Pr(e)$; decreasing π_e or θ_e by $\ln 2$ roughly halves it.

π_e may be regarded as a slack variable. Without it, $\Pr(e)$ is constrained to be exactly a linear combination of the probabilities of e's parents. The coefficients of this linear combination are determined by θ; they are not specific to e unless θ includes per-event features. π_e captures the difference between the observed value of $\Pr(e)$ and its value as predicted from θ. The prior says that this difference is probably small, which helps us guess either $\Pr(e)$ and θ, or both, in case of uncertainty.

3.9.2 Solving the Perturbed Model

Given values for the θ and π parameters, the perturbed transformation model defines a probability distribution $\Pr_{\theta,\pi}(\cdot)$ over Events. The new π parameters affect this distribution by requiring the outflow of each event vertex—both to other events and to HALT—to deviate in a particular way from its inflow. When solving for $\Pr_{\theta,\pi}$ as in §3.4, we replace the recurrence relation of equation (3.18) with the following, where $I_{\theta,\pi}$ represents inflow:

$$I_{\theta,\pi}(e) = \sum_{A=\langle e',e,F\rangle \in \mathsf{Arcs}} I_{\theta,\pi}(e') \cdot \exp \pi_{e'} \cdot P_\theta(A) + \delta(e = \textsc{Start}) \tag{3.28}$$

Expectation-Maximization would require a non-linear variant: Let the cost of the arc from e to HALT be the number of observations of e, and all other arcs have cost zero. Then find a legal flow that minimizes $\sum_A cost(A) \cdot (-\ln flow(A))$, under the constraint that the total flow into the sink is 1. Things are more complicated if $\vec{\pi}$ is unknown and must be estimated.

This "artificially" increases the outflow from e by a factor of $\exp pi_e$ over its inflow. That is, e can visit its children more often (if $pi_e > 0$) than the number of times it is visited from its parents.

As before, if Events is finite, this linear system of equations can be exactly solved for the vector $I_{\theta,\pi}$ by matrix methods (see §8.4). If Events is infinite, we can use approximation techniques (§4.2, §8.5).

We would then like to obtain $\Pr_{\theta,\pi}$ by replacing equation (3.20) as follows, again respecting the flow multiplier on the flow to Halt:

$$\Pr_{\theta,\pi}(e) = I_\theta(e) \cdot \exp \pi_e \cdot \sum_{A=\langle e,\textsc{Halt},F\rangle \in \mathsf{Arcs}} P_\theta(A) \tag{3.29}$$

However, this is not in general a probability distribution. It does not sum to 1 because the perturbations affect the total amount of probability mass flowing through the transformation graph.

It is therefore necessary to renormalize the "distribution" given by equation (3.29). Rather than following equation (7.12) and computing the normalizing factor to divide by as $I_\theta(\textsc{Halt})$, which is the amount of unnormalized probability mass leaving the graph, it is sometimes more convenient to compute it as the amount of unnormalized probability mass *entering* the graph,

$$\underbrace{1}_{\substack{\text{probability of}\\ \text{starting at }\textsc{Start}}} + \sum_{e \in \mathsf{Events}} \underbrace{I_{\theta,\pi}(e) \cdot (\exp \pi_e - 1)}_{\text{extra probability injected at node } e} \tag{3.30}$$

since this only requires summing over e such that $\pi_e \neq 0$, and in practice this may be a small set of events,[21] whereas if Events is large (or infinite) then $I_\theta(\textsc{Halt})$ may be hard to compute.

3.9.3 Perturbations vs. Per-Event Weights

The discussion above exposes how perturbations work. A positive perturbation lists the event e "directly." It allows the transformational process to start at e rather than Start,

[21] E.g., one may have tied π_e to zero for events that were not observed or well-observed.

with some probability. As a result, the probabilities of starting at START and other nodes must be scaled down.

Similarly, a negative perturbation makes it less likely that the transformational process will continue from e. It can be interpreted as making the process quit at e with some probability, and restart from scratch. (It might choose to restart at START or at any node with a positive perturbation, just as if it were starting for the first time.)

As with per-event weights, the prior chance of shifting a given amount of probability mass to e depends on the existing probability of e. That is, the transformational process will not mind being asked to start often at e if it would have reached e often anyway from other starting points.[22]

How do such perturbations differ from per-event weights? Raising the perturbation π_e makes e (and its descendants) more likely at the expense of *all* other events. By contrast, raising the per-event weight θ_e improves e's ability to compete with its siblings. It makes e (and its descendants) more likely at the expense of e's siblings and parents only.[23]

Thus the perturbed model eliminates some of the anti-correlation between sibling events. (See footnote 10 on p. 94.) Since transformational arcs from the same node still compete, siblings still compete with one another for a share of the flow to their common parent. But that applies only to the regular processes described by θ, not to the exceptions described by π. Being listed as a starting point is a competition among all nodes of the transformation graph (including START), and is less sensitive to the details of the model topology.

[22] Or from e itself by a self-loop or other cyclic path in the transformation graph. It is an unfortunate artifact of perturbed transformation models that if a cyclic path has high probability, then a "feedback effect" means that small perturbations to events on that path can increase their probabilities greatly.

As an extreme case, suppose the transformation graph contains a cyclic path with probability 1. If the path is accessible from START, there is no solution to the recurrence (3.28), and if it is not accessible, there are infinitely many solutions. In either case, equation (3.28) calls for the inversion of a singular matrix.

[23] The flow to the parents does not change, but when the random walk reaches a parent of e, it is more likely to continue to e and therefore less likely to halt at the parent. Put another way, HALT may be one of the siblings that e competes with.

Chapter 4

Efficient Parameter Estimation for Transformation Models

Chapter 3 introduced transformation models (defined more concisely below). Given a transformation model of a domain, the transformations and exceptions in a given training dataset are to be captured by an appropriate choice of parameters $\theta = \langle \theta_1, \theta_2, \ldots \theta_k \rangle \in \mathbb{R}^k$. To fit the model to data in this way, we choose θ that maximizes Bayes' Theorem (equation (3.4)). (See footnote 2 on p. 82 for alternatives.)

But how can we carry out this maximization efficiently in practice? The experiments of Chapter 6 simply use gradient descent. As preparation for reading about them, this chapter develops the relevant methods for computing (1) an approximation to the objective function and (2) the gradient of this approximation.

Later, Chapter 8 will give a fuller presentation of possible algorithms for these tasks—slow exact algorithms as well as faster approximate ones. One important result postponed to that chapter is an EM algorithm, which is closely connected to the gradient computation. That chapter will also draw algorithmic connections to the solution of linear systems and to back-propagation in neural networks. Finally, that chapter will show how to handle the variant models to be discussed in Chapter 7.

4.1 A Matrix Definition of the Objective Function

Let us first summarize the definitions of Chapter 3 using a concise matrix notation for transformation models. This notation suffices when the transformation graph is a finite directed graph. (Not a multigraph, although this restriction implies no loss of generality,[1] and in any case the algorithms we describe with the matrix notation generalize trivially to multigraphs.)

4.1.1 Model Specification

Let $1, 2, \ldots n$ be an enumeration of Events, with 1 representing START. Let 0 represent HALT $\notin \mathsf{Events}$. The model is entirely specified (cf. §3.2.1) by a three-dimensional array $F \in \mathbb{R}^{(k+1)\times(n+1)\times(n+1)}$. (Array indices start at 0.) We will write $|F|$ for the number of non-zero elements in F. The element $F^h_{ij} \in \mathbb{R}$ represents the coefficient of feature t_h on the arc from event i to event j, that is, the number of times it appears on that arc. (For the basic formalism of §3.2.1, $F^h_{ij} \in \{0, 1\}$, but it is no trouble to allow arbitrary real coefficients, as discussed in §7.3.1.1.)

Formally, if the transformation graph has no arc from event i to event j, we need to put $F^0_{ij} = 1$ (and $F^h_{ij} = 0$ for $h > 0$). This specifies a dummy arc bearing just the special feature t_0, whose weight $\theta_0 \stackrel{\text{def}}{=} -\infty$, ensuring that the arc has zero probability. Such a dummy or missing arc is distinct from an arc that has no features.[2] There must be no arcs from event $i = 0$ (HALT), and at least one arc from each event $i > 0$.

In practice, it is not necessary to store F explicitly or have random access to its elements. (This is good because F may be infinite and defined by some policy.) It can be represented as any object that supports the following two methods:

[1] Given a multigraph transformation model in the notation of §3.2.1, one can easily construct a graph transformation model that defines the same parameterized probability distribution. To eliminate an "extra" arc $\langle e, e', F\rangle$, add a new vertex e'' to Events and replace the arc with a pair of arcs $\langle e, e'', F\rangle$ and $\langle e'', e', \emptyset\rangle$.
This construction fails if $e' =$ HALT, since then the random walk will halt at the new vertex e'' instead of e. So if there are multiple arcs from e to HALT, some preprocessing is required. Add a new vertex $\hat{e}$ to Events; replace e with $\hat{e}$ throughout the model; replace *every* arc of the form $\langle \hat{e}, \text{HALT}, F\rangle$ with $\langle \hat{e}, e, F\rangle$; and finally add the arc $\langle e, \text{HALT}, \emptyset\rangle$. Now the multiple arcs go from $\hat{e}$ to e, and the extras can be eliminated by the previous construction.

[2] An arc with no features has positive probability since $G_\theta(\text{arc}) = 1$. See equation (3.13).

- Given a vertex i, it must be able to enumerate the arcs ij that exist in the transformation graph (i.e., for which $F^0_{ij} = 0$).

- Given an arc ij, it must be able to enumerate the features on the arc. Each feature is a modifiable record that stores the feature's current weight F^h_{ij} and perhaps some other information about the feature, such as the partial derivative $\partial f / \partial F^h_{ij}$.

It is convenient to implement these enumerators as "lazy list" objects so that other functions can iterate over arcs and their features and perform arbitrary operations on them.

4.1.2 Defining Transition Probabilities

Given a parameter vector $\vec{\theta} \in \mathbb{R}^k$ (conventionally written just as θ), we define the matrix $P_{\vec{\theta}} \in \mathbb{R}^{(n+1)\times(n+1)}$ to represent the transitional probabilities of the transformation graph. We will write $|P|$ for the number of non-zero entries in P.

F and θ affect the computations of this chapter (and Chapter 8) only through P_θ. Ordinarily P_θ is defined as shown below, following §3.2.2, although we will sometimes vary the definition as discussed later:

$$(G_\theta)_{ij} \stackrel{\text{def}}{=} \exp(\vec{\theta} \cdot \vec{F_{ij}}) = \exp \sum_{h=0}^{k} \theta_h F^h_{ij} \qquad \text{(unnormalized probabilities)} \tag{4.1}$$

$$(P_\theta)_{ij} \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} (G_\theta)_{ij} / \sum_{j'=0}^{n} (G_\theta)_{ij'} & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{array} \right\} \qquad \text{(transition probabilities)} \tag{4.2}$$

(The special case for $i = 0$ is necessary to prevent division by zero: there are no arcs from event 0 (HALT), so the zero$^{\text{th}}$ row of G_θ is identically zero and P_θ follows this.[3])

P is often too large to store in memory (indeed infinite). However, the non-zero elements in each row can be enumerated on demand, via the above equations and the methods supported by F (§4.1.1). It is very useful for each vertex i to cache the value of $\sum_{j'=0}^{n} (G_\theta)_{ij'}$; this value is used many times each time that that row i is enumerated. The cached value becomes stale when θ changes.

[3]One might try to clean this presentation up by giving HALT a self-loop so that *all* rows of P_θ would sum to 1. But then the flow to HALT, I_0, would be infinite, with the result that $(1 - P_\theta)$ in equation (4.5) below would have no inverse.

4.1.3 Model Solution

Following §3.4, we now define the row vectors $\vec{b}, \vec{I_\theta} \in \mathbb{R}^{n+1}$, the latter by a matrix equation:

$$\vec{b} \stackrel{\text{def}}{=} \langle 0, 1, 0, 0, 0, \ldots \rangle \qquad \text{(initial probabilities)} \tag{4.3}$$

$$\vec{I_\theta} \stackrel{\text{def}}{=} \vec{I_\theta} \cdot P_\theta + \vec{b} \tag{4.4}$$

whence

$$\vec{I_\theta} = \vec{b} \cdot (1 - P_\theta)^{-1} \qquad \text{(flow to vertices)} \tag{4.5}$$

(here 1 denotes the identity matrix of order $n + 1$)

Recall that I_i is called the inflow to vertex i, and denotes the expected number of times that a random walk will visit i.

Equation (4.5) requires a matrix inversion that is impractical to compute for large models. Avoiding such inversions will be the main goal of the algorithms in this chapter and Chapter 8.

The probability distribution over events that the model specifies is now

$$(p_\theta)_i \stackrel{\text{def}}{=} (I_\theta)_i \cdot (P_\theta)_{i0} \qquad \text{(observable probabilities)} \tag{4.6}$$

which is the pointwise product of the flow vector $\vec{I_\theta}$ with the halt probabilities in the 0^{th} column of P_θ. We can more neatly rewrite the last equation as

$$\vec{p_\theta} \stackrel{\text{def}}{=} \vec{I_\theta} \odot (P_\theta)_{\cdot 0} \tag{4.7}$$

using the **pointwise product operator** $\odot$, defined by $(\vec{a} \odot \vec{b})_i \stackrel{\text{def}}{=} a_i \cdot b_i$ (or equivalently $\vec{a} \odot \vec{b} \stackrel{\text{def}}{=} \mathsf{diag}(\vec{a}) \cdot \vec{b}$), and the convention that $P_{\cdot i}$ represents the i^{th} column of matrix P.

4.1.4 Objective Function

Following §3.1.3, our objective function is the logarithm of the posterior probability of θ. Let s_i denote the number of times event i was observed in the training sample (elsewhere denoted $\#(i)$). Using the Gaussian prior of §3.5 together with Bayes' Theorem (equation (3.4)), the quantity to maximize is

$$f(\theta) \stackrel{\text{def}}{=} -\sum_{h=1}^{k} \theta_h^2 / 2\sigma^2 + \sum_{i=1}^{n} s_i \cdot \ln \frac{(p_\theta)_i}{Z} \tag{4.8}$$

Here the normalizing factor $Z \stackrel{\text{def}}{=} \sum_j (p_\theta)_j$. It can be ignored for the time being, since it equals 1 under the definitions so far. We will need it later, for instance to renormalize the perturbed models of §3.9 (see §4.4).

4.1.5 Evaluation Function

The success of the method can be evaluated by passing the result of optimization, $\tilde{\theta}$, to an evaluation function g. Our current evaluation function measures the log-probability of a test sample under the fitted model $p_{\tilde{\theta}}$. If u_i denotes the number of observations of event i in the test sample, then

$$g(\tilde{\theta}) \stackrel{\text{def}}{=} \sum_{i=1}^{n} u_i \cdot \ln \frac{(p_{\tilde{\theta}})_i}{Z} \tag{4.9}$$

Larger values are better.

When reporting the objective and evaluation functions, it is customary to divide them by $-n \ln 2$, yielding a positive value measured in bits per test entry. In particular, $-g(\tilde{\theta})/n \ln 2$ is called the **cross-entropy** of the test sample under the fitted model. Another convention is to report $\exp -(g(\tilde{\theta})/n)$, known as the **perplexity** of the test sample under the fitted model. (Smaller values of cross-entropy and perplexity are better.) However, the intermediate computations are easier to perform and describe in terms of natural logarithms, hence the definitions (4.8) and (4.9).

4.1.6 A Caveat on Numerical Accuracy

One must be unusually careful when solving numerically or approximately for the flow vector $\vec{I}_\theta$, and hence for the probability vector $\vec{p}_\theta$. As noted, the probability vector must sum to at most 1, to avoid giving the method an unfair advantage in evaluation. And because the objective and evaluation functions (equations (4.8) and (4.9)) are sensitive to the *logarithm* of probability, relative error in every component of $\vec{p}_\theta$ must be kept small. If event i has been observed, and especially if it has been observed several times, then even a small value of $(p_\theta)_i$, such as 2×10^{-8}, must be computed accurately. Computing it as 1×10^{-8} due to numerical error will decrease the objective or evaluation function by the observation count of i. Computing it as 0 can drive the objective or evaluation function to

$-\infty$. And computing it as a small negative number makes it impossible to interpret the results as probabilities at all.

4.2 Solving the Model by Relaxation

We wish to compute an approximation $\vec{p}$ to $\vec{p}_\theta$ without carrying out the matrix inversion of §4.1.3. The following **relaxation algorithm** gradually increases the elements of $\vec{p}$ toward their desired values.

We can then substitute $\vec{p}$ for $\vec{p}_\theta$ in equation (4.8), thereby defining an approximation $\tilde{f}$ to the objective function f; and similarly in equation (4.9) to define an approximation $\tilde{g}$ to the evaluation function g.

4.2.1 The Relaxation Algorithm

Each vertex $i \neq 0$ stores its current estimate of p_i. This represents the total probability of all random walks considered so far that have halted at i (i.e., taken the arc from i to HALT).

Each vertex $i \neq 0$ also defines I_i, which represents the total flow (§3.4) that has been propagated to i so far. This value need not actually be stored. What vertex $i \neq 0$ does store is a value $J_i \in [0, I_i]$, which is the portion of I_i that it has still not propagated to its children.

Initially, $\vec{p} = 0$ and $\vec{J} = \vec{b}$ (and $\vec{I} = \vec{b}$), where $\vec{b}$ is the start vector defined by equation (4.3). A step of the algorithm consists of selecting any vertex $i \neq 0$ such that $J_i \neq 0$ and propagating J_i to the children of i, including HALT:

1. Let $z = J_i$
2. Reset $J_i = 0$
3. Increment p_i by $z \cdot P_{i0}$
4. For each vertex $j \neq 0$ that is a child of i (that is, $P_{ij} \neq 0$), increment J_j (and I_j) by $z \cdot P_{ij}$

This is called **relaxing** vertex i. Notice that P_{i0} is considered separately from P_{ij} for $j \neq 0$. Full pseudocode for relaxation will be given in §4.3.3 below.

The algorithm can be motivated from the random-walk interpretation of the transformation model (§3.3.1). Picture a large army of N ants that begins at START. The ants take independent random walks on the graph—but not synchronously. When a vertex i is kicked (relaxed), its ants swarm one step away from the vertex along its out-arcs ij, in proportion to the arcs' probabilities P_{ij}. Then $J_i N$ represents the number of ants currently at i (with $\sum_i J_i = 1$ since there are always N ants in the graph), and $I_i N$ represents the total number of ant visits to i so far. $p_i N$ represents the number of ants that have followed the $i0$ arc (from i to HALT) so far.

Mohri (2000) independently proposed the same relaxation algorithm,[4] working not with the reals but in a different class of "k-closed" semirings in which the algorithm was guaranteed to converge.[5] He also noted a connection to shortest-path algorithms. His application was to compute ϵ-closures for weighted finite-state automata, a problem whose connection to transformation models will be elaborated in §8.2.3.

4.2.2 Disciplines for Relaxation

Different strategies are possible for selecting the vertex i to relax at each step. A straightforward one is to choose i such that J_i is maximal. This can be arranged by maintaining a priority queue of the vertices $\{i \neq 0 : J_i \neq 0\}$, with the priority of i being J_i. (Footnote 15 on p. 133 gives an improved definition of priority.)

The algorithm will never succeed in emptying the queue if the graph is cyclic, but it may be stopped at any time, with longer runs giving better approximations.[6] A straightforward

[4]Aside from notation, Mohri's version differs only in that he accumulates $\vec{I}$ instead of $\vec{p}$. (In principle we could likewise accumulate $\vec{I}$, and define $\vec{p}$ by equation (4.7) only after the algorithm terminated. However, that would force us to hold the halt probabilities constant throughout the algorithm, whereas it is sometimes useful to vary them: see §6.5.3 and §8.4.)

[5]Mohri (p.c.) did note that the algorithm might be used as an approximation in the reals, which do not form a k-closed semiring.

[6]As the algorithm runs, it effectively considers more and more prefixes of possible random walks. p_i accumulates the total probability of prefixes that are actually complete walks that have halted at i. It is not hard to see from this that p_i increases monotonically, and that it converges to the true value if every vertex is relaxed infinitely often, so that every random walk prefix is eventually considered. (The same argument also demonstrates that I_i converges from below to its true value.)

criterion is to stop when J_0 is close to 1 (i.e., most of the ants have reached HALT).[7] Alternatively, one can stop when the priorities of all remaining vertices are below some threshold. For example, if the priority of vertex i is defined to be J_i, this means stopping when J_i is close to 0 for every $i \neq 0$: that is, when no single node has enough ants to make it worth kicking.

Relaxing vertex i takes time proportional to its number of out-arcs. The overall runtime of the algorithm depends on how long one chooses to run it, e.g., how long it takes to converge with a given model and given parameters θ.

It is sometimes useful to exploit information about the transformation graph's topology when choosing an order for relaxation. To minimize the number of relaxations, one would like to wait until a lot of ants have accumulated at a vertex before relaxing that vertex. In the case of an acyclic graph, relaxing the vertices in topologically sorted order will empty the queue with only one relaxation per vertex. A generalization of this trick applies to cyclic graphs, as noted by Mohri (2000). Any graph can be decomposed into its strongly connected components, and the components of this graph may be topologically sorted into an order $C_1, C_2, C_3, \ldots$.[8] Then a good strategy is to relax only vertices in C_1 until some stopping criterion is reached, then relax only vertices in C_2, and so on. This strategy ensures that when a vertex is relaxed, it has already received as many ants as it ever will from other strongly connected components.

The version of relaxation used in the actual experiments took advantage of the particular transformation graph used, in order to arrange a natural way of stopping the algorithm. See §6.5.3 for details.

[7] Above we only defined J_i for $i > 0$. However, one can also maintain J_0 by allowing j to also take value 0 at step 4 of relaxation, above. Note that $J_0 N$ represents the total number of ants that have halted so far, and equals $I_0 N$ because vertex 0 is never relaxed (i.e., all the ants at 0 stay there forever).

[8] A **strongly connected component** of a graph is a maximal subset S of vertices such that for any two vertices $i, j \in S$, there is a directed path from i to j. Any cycle in the graph falls within some strongly connected component. If the strongly connected components are not known, they may be found in linear time (for a finite graph) by Tarjan's (1972) algorithm. The order $C_1, C_2, C_3, \ldots$ is also found by that algorithm; definitionally, this order results from contracting each component to a single vertex and topologically sorting the resulting acyclic graph.

4.2.3 Relaxation Produces a Deficient Solution

Because the relaxation algorithm must be halted before convergence, in general $\vec{p}$ will be a deficient distribution, leading to a pessimistic approximation $\tilde{g}$ of the model's performance. However, §6.5.3 and §8.4 will later describe ways to avoid this deficiency if desired; the strategy of §6.5.3 is actually used in the experiments reported in Chapter 6.

In particular, if p_i is underestimated as 0 for some i observed in training or test data, then $\tilde{f}$ or $\tilde{g}$ will be underestimated as $-\infty$. Several solutions are possible:

- Simply ignore such vertices i during training, treating s_i as 0 in the definition of $\tilde{f}$. This at least allows training to go forward: optimizing $\tilde{f}$ tries to raise the joint probability of θ and the part of the observed data that we can account for. However, it does not necessarily help in testing.
- Take care to continue relaxation until $\tilde{f}$ or $\tilde{g}$ becomes finite.[9] This is loosely related to the strategy actually used in the experiments (§6.5.3).
- Back off to a lower-order model $\vec{q}$ that assigns non-zero probability to all events: replace the estimated distribution $\vec{p}$ in the definition of $\tilde{f}$ with $\alpha\vec{p}^{(T)} + (1-\alpha)\vec{q}$.

4.3 Computing the Gradient

We wish to optimize the approximated objective function $\tilde{f}$. A gradient-based numerical optimization technique, such as gradient descent or conjugate gradient, takes as inputs the function $\tilde{f}(\theta)$ and its gradient $\nabla\tilde{f}(\theta)$. Notice that this is the *exact* gradient of the approximate objective, in order to guarantee that the optimizer will work correctly. It is not just some approximate gradient of the exact objective.

The technique for computing this gradient is similar to back-propagation in recurrent neural networks—a connection that will be further explored in §8.5.4.

[9] This is fair provided that $\vec{p}$ is not renormalized, even though the computation of p_i uses the knowledge that p_i is in the test set. The computed p_i is still an underestimate of the true value that would be found if relaxation were allowed to converge. It is not exaggerated at the cost of some other p_j. Nonetheless, this evaluation procedure does raise concerns about replicability, since p_i might receive a different approximation in the context of some other test set that arose in practice. One could answer this concern by defining the approximation p_i to be the first positive value of p_i after relaxation had run for at least some preset amount of time, ignoring any further increases to p_i.

4.3.1 Notation

For any real variable x, let $\mathsf{g}x$ denote $\partial\tilde{f}/\partial x$, the partial of the approximated objective function with respect to x. The chain rule for derivatives is then $\mathsf{g}x = \sum_y \mathsf{g}y \cdot \frac{\partial y}{\partial x}$. We will write $\mathsf{g}\vec{x}$ for $\langle \mathsf{g}x_1, \mathsf{g}x_2, \ldots \rangle$.

Some more notation will be developed in §4.3.4.

4.3.2 Approach

We will implicitly use a common trick for computing partial derivatives. If a variable x is used several times in the computation of $\tilde{f}$, then we may regard these mentions as if they were separate variables $x^{(1)}, x^{(2)}, \ldots$, and compute $\mathsf{g}x$ as $\mathsf{g}x^{(1)} + \mathsf{g}x^{(2)} + \cdots$.[10]

For example, the transition probability P_{ij} may be used several times during the relaxation algorithm (since i may be relaxed several times). Suppose these mentions are called $P_{ij}^{(1)}, P_{ij}^{(2)}, \ldots P_{ij}^{(T)}$. We will compute the partials with respect to these mentions in a natural order—in fact, in the reverse of the order in which the probabilities were used during relaxation. As we compute $\mathsf{g}P_{ij}^{(T)}, \ldots \mathsf{g}P_{ij}^{(2)}, \mathsf{g}P_{ij}^{(1)}$, we add them into an accumulator for $\mathsf{g}P_{ij}$ that was initialized to 0.

4.3.3 Back-Relaxation

Let us first give a basic version of the algorithm in which the transition matrix is a fixed matrix P. This basic version builds up a matrix of partials $\mathsf{g}P$ as described above. It is the version that is most closely analagous to back-propagation: the transition probability matrix P corresponds to the weight matrix in a neural network.

(§4.3.5 will extend this to the full case, in which P so is itself parameterized by the more compact $\vec{\theta}$ (i.e., $P = P_\theta$); then what we really want is $\mathsf{g}\vec{\theta}$.)

To compute the gradient it is necessary to work backwards from the final computation of $\tilde{f}$, assessing how the various uses of P_{ij} will eventually affect $\tilde{f}$. To enable this, the

[10] Formally, we have a function $\tilde{f}(\ldots, x, \ldots)$. Define a new function $F(\ldots, x^{(1)}, x^{(2)}, \ldots, x^{(T)}, \ldots)$ that is computed in exactly the same way but with the T mentions of x replaced respectively with $x^{(1)}, x^{(2)}, \ldots x^{(T)}$. Then $\tilde{f}(\ldots, x, \ldots) = F(\ldots, \underbrace{x, x, \ldots x,}_{T} \ldots)$. By the chain rule, $\frac{\partial \tilde{f}}{\partial x} = \sum_{i=1}^{T} \frac{\partial F}{\partial x^{(i)}} \cdot \frac{\partial x^{(i)}}{\partial x} = \sum_{i=1}^{T} \frac{\partial F}{\partial x^{(i)}}$.

computation of $\tilde{f}$ must be modified to do some additional bookkeeping (the new line 5 below):

RELAXATION():

1. $\mathit{Stack}1 := \emptyset$ (* *Stack*2 *and* $\mathbf{g}\vec{J}$ *retain their values from previous* BACK-RELAXATION, *if any* *)
2. $\vec{p} := 0;\ \vec{J} := \vec{b}$
3. **repeat until** stopping criterion met
4. choose $i \neq 0$ such that $J_i \neq 0$ (* *e.g., by popping priority queue or Stack2; see §4.5.1, §4.5.2* *)
5. push (i, J_i) onto $\mathit{Stack}1$
6. $z := J_i;\ J_i := 0$
7. increment p_i by $z \cdot P_{i0}$
8. **for** each child $j \neq 0$ of i
9. increment J_j by $z \cdot P_{ij}$
10. **return** $\vec{p}$

To compute partial derivatives is to play a game of hypotheticals: how would the result $\vec{I}$ of the above relaxation, and the objective function $\tilde{f}$ that is computed from this result, have differed if a P_{ij} value had been infinitesimally different during relaxation?

Having run relaxation to compute $\tilde{f}$, we can now run relaxation backwards to play this game. (Ignore the use of *Stack*2 for the time being.) Each vertex i must maintain a value $\mathbf{g}J_i$. The quantity $\mathbf{g}p_i$ is defined as

$$\mathbf{g}p_i = s_i/p_i \tag{4.10}$$

by differentiation of equation (4.8).[11]

BACK-RELAXATION():

1. $\mathit{Stack}2 := \emptyset$ (* *Stack*1 *retains its value from end of previous* RELAXATION *)
2. $\mathbf{g}\vec{J} := 0;\ \mathbf{g}P := 0$
3. **repeat until** $\mathit{Stack}1 = \emptyset$

[11] In the event that the denominator $p_i^{(T)} = 0$, then $\mathbf{g}p_i^{(T)}$ is 0 or $+\infty$ according to whether $s_i = 0$ or not. The latter case forces $\tilde{f} = -\infty$. Modifications to make $\tilde{f}$ finite were discussed in §4.2.3, and it is straightforward to modify $\mathbf{g}p_i^{(T)}$ accordingly:

- If we treat s_i as 0 in the definition of $\tilde{f}$, then we should also do so in the definition of $\mathbf{g}p_i^{(T)}$.
- Suppose we replace $\vec{p}^{(T)}$ in the definition of $\tilde{f}$ with a backed-off model $\vec{p} \stackrel{\text{def}}{=} \alpha\vec{p}^{(T)} + (1-\alpha)\vec{q}$. Then $\mathbf{g}p_i^{(T)} = \alpha s_i/p_i$. (Also, $\mathbf{g}\alpha = (p_i^{(T)} - q_i)s_i/p_i$, a fact that can be used to optimize α.)

pop (i, J_i) from $Stack1$ (* *this sets* J_i *for use below* *)
push $(i, \mathrm{g}J_i)$ onto $Stack2$ (* *for possible use by a subsequent call to* RELAXATION *)
$z := 0$
increment $\begin{cases} z & \text{by } P_{i0} \cdot \mathrm{g}p_i \\ \mathrm{g}P_{i0} & \text{by } J_i \cdot \mathrm{g}p_i \end{cases}$
for each child $j \neq 0$ of i
increment $\begin{cases} z & \text{by } P_{ij} \cdot \mathrm{g}J_j \\ \mathrm{g}P_{ij} & \text{by } J_i \cdot \mathrm{g}J_j \end{cases}$
$\mathrm{g}J_i := z$
return $\mathrm{g}P$

RELAXATION visits vertices in an order recorded on $Stack1$, and BACK-RELAXATION visits those vertices in reverse order by popping the stack. The two algorithms run equally fast, up to a constant, even though the latter is computing a larger set of values.

4.3.4 Correctness of Back-Relaxation

The correctness of back-relaxation holds by induction, as we will now see. (This section may be skipped on a first reading, but the notation will be used in later sections as well.)

The proof requires a notation for referring to the values of variables at different times t. Thus, as a preliminary step, let us formally rewrite the relaxation and back-relaxation algorithms to use different names for these different values. Also, $P_{ij}^{(t)}$ refers to the mention of P_{ij} on iteration t, in the sense of §4.3.2. Finally, $\mathrm{g}p_i$ refers to $\mathsf{g}(p_i^{(T)})$ and is computed just as before. It is easy to see by inspection that the new versions do the same thing as the old ones.

RELAXATION(): (* *transformed version, for correctness proof and discussion ONLY* *)
$t := 0$; $Stack1 := \emptyset$ (* $Stack2$ *and* $\mathrm{g}\vec{J}$ *retain their values from previous* BACK-RELAXATION, *if any* *)
$\vec{p}^{(t)} := 0$; $\vec{J}^{(t)} := \vec{b}$
repeat until stopping criterion met
$t := t + 1$
(* *Now construct* $\vec{I}^{(t)}, \vec{J}^{(t)}$ *from* $\vec{I}^{(t-1)}, \vec{J}^{(t-1)}$, *and* $P^{(t)}$. *)
choose $i^{(t-1)} \neq 0$ such that $J_{i^{(t-1)}} \neq 0$ (* *e.g., by popping priority queue or* $Stack2$ *)
let $i := i^{(t-1)}$; push $(i, J_i^{(t-1)})$ onto $Stack1$

$p_i^{(t)} := p_i^{(t-1)} + J_i^{(t-1)} \cdot P_{i0}^{(t)}$

  for $j := 1, \ldots n$

    $J_j^{(t)} := \left\{ \begin{array}{ll} J_j^{(t-1)} & \text{if } j \neq i \\ 0 & \text{if } j = i \end{array} \right\} + J_i^{(t-1)} \cdot P_{ij}^{(t)}$

    increment J_j by $z \cdot P_{ij}$

$T := t$

return $\vec{p}^{(T)}$

Back-Relaxation(): *(* transformed version, for correctness proof and discussion ONLY *)*

$Stack2 := \emptyset$ *(* t and $Stack1$ retain their values from end of* Relaxation*; in particular $t = T$ *)*

$(\mathsf{g}\vec{J})^{(t)} := 0$; $(\mathsf{g}P)^{(t)} := 0$

repeat until $Stack1 = \emptyset$ *(* equivalently, until $t = 0$ *)*

  (Now construct $(\mathsf{g}P)^{(t-1)}, (\mathsf{g}\vec{J})^{(t-1)}$ from $(\mathsf{g}P)^{(t)}, (\mathsf{g}\vec{J})^{(t)}$, and $P^{(t)}$. *)*

  pop $(i^{(t-1)}, J_i^{(t-1)})$ from $Stack1$; let $i := i^{(t-1)}$

  push $(i, (\mathsf{g}J)_i^{(t)})$ onto $Stack2$ *(* for possible use by a subsequent call to* Relaxation **)*

  (Initialize new values to be the same as on previous pass *)*

  $(\mathsf{g}J)^{(t-1)} := (\mathsf{g}\vec{J})^{(t)}$

  $(\mathsf{g}P)^{(t-1)} := (\mathsf{g}P)^{(t)}$

  (Now tweak row i of the new values. *)*

  $(\mathsf{g}J)_i^{(t-1)} := P_{i0}^{(t)} \cdot \mathsf{g}p_i + \sum_{j=1}^{n} P_{ij}^{(t)} \cdot (\mathsf{g}J)_j^{(t)}$

  $(\mathsf{g}P)_{i0}^{(t-1)} := (\mathsf{g}P)_{i0}^{(t)} + J_i^{(t-1)} \cdot \mathsf{g}p_i$

  for $j := 1, \ldots n$

    $(\mathsf{g}P)_{ij}^{(t-1)} := (\mathsf{g}P)_{ij}^{(t)} + J_i^{(t-1)} \cdot (\mathsf{g}J)_j^{(t)}$

  $t := t - 1$

return $(\mathsf{g}P)^{(0)}$

The formal claim to be proved by induction is that for all $t = T, T-1, \ldots 0$,

$$(\mathsf{g}\vec{J})^{(t)} = \mathsf{g}(\vec{J}^{(t)}) \tag{4.11}$$

$$(\mathsf{g}P)^{(t)} = \mathsf{g}(P^{(t+1)}) + \mathsf{g}(P^{(t+2)}) + \cdots + \mathsf{g}(P^{(T)}) \tag{4.12}$$

Then the second equation at $t = 0$ asserts that the return value of Back-Relaxation correctly implements the strategy in §4.3.2.

What do equations (4.11) and (4.12) mean? Their left-hand sides are variables computed by Back-Relaxation. But their right-hand sides are actual gradients of quantities used in RELAXATION.

For example, what does $\mathtt{g}(J_i^{(t)})$ mean? Suppose one interrupted RELAXATION at line line 3 and increased $J_i^{(t)}$ by an infinitesimal ϵ. (Here t denotes the value of t at the time of the interruption.) If one then resumed RELAXATION and used its return value $\vec{p}^{(T)}$ to compute $\tilde{f}$, the final effect on $\tilde{f}$ would be an increase of $\epsilon \cdot \mathtt{g}(J_i^{(t)})$, by the definition of $\mathtt{g}(J_i^{(t)})$.

If one were to increase P_{ij} by ϵ under the same circumstances, so that the increase affected all subsequent mentions $(P^{(t+1)}, P^{(t+2)}, \ldots P^{(T)})$, then the increase to $\tilde{f}$ would by definition be $\mathtt{g}(P^{(t+1)}) + \mathtt{g}(P^{(t+2)}) + \cdots + \mathtt{g}(P^{(T)})$. It is exactly these values that BACK-RELAXATION is claimed to compute.

The base case for the induction is $t = T$. The claim holds here: in both equations, the left-hand side is zero by the initialization of BACK-RELAXATION, and the right-hand side is zero by definition. In particular, $\mathtt{g}(\vec{J}^{(T)}) = 0$ because $\vec{J}^{(T)}$ is computed in the last iteration of RELAXATION but never subsequently used in the computation of $\tilde{f}$.

For the inductive step, assume that the claim holds for t. We wish to show it for $t-1$. Let $i = i^{(t-1)}$. Now

$$\begin{aligned}
\mathtt{g}(J_i^{(t-1)}) &= \mathtt{g}p_i \cdot \frac{\partial p_i}{\partial J_i^{(t-1)}} + \sum_{j=1}^{n} \mathtt{g}(J_j^{(t)}) \cdot \frac{\partial J_j^{(t)}}{\partial J_i^{(t-1)}} \text{ by the chain rule of §4.3.1} && (4.13)\\
&= \mathtt{g}p_i \cdot P_{i0}^{(t)} + \sum_{j=1}^{n} \mathtt{g}(J_j^{(t)}) \cdot P_{ij}^{(t)} \text{ by lines 8–10 of RELAXATION (p. 125)} && (4.14)\\
&= \mathtt{g}p_i \cdot P_{i0}^{(t)} + \sum_{j=1}^{n} (\mathtt{g}J)_j^{(t)} \cdot P_{ij}^{(t)} \text{ by inductive hypothesis} && (4.15)\\
&= (\mathtt{g}J)_i^{(t-1)} \text{ by line 11 of BACK-RELAXATION (p. 125)} && (4.16)
\end{aligned}$$

while for $j \neq i$, similarly

$$\mathtt{g}(J_j^{(t-1)}) = \mathtt{g}(J_j^{(t)}) \cdot \frac{\partial J_j^{(t)}}{\partial J_j^{(t-1)}} = \mathtt{g}(J_j^{(t)}) = (\mathtt{g}J)_i^{(t)} = (\mathtt{g}J)_i^{(t-1)} \tag{4.17}$$

This proves the first part (4.11) of the claim. As for the second part (4.12), the definition of the entries of $\mathtt{g}(P^{(t)})$ splits into three cases. Observe first that by a trivial induction on line 8 of RELAXATION (p. 125),

$$p_i \stackrel{\text{def}}{=} p_i^{(T)} = \sum_{t=1}^{T} J_i^{(t-1)} \cdot P_{i0}^{(t)} \tag{4.18}$$

whence

$$\mathsf{g}(P_{i0}^{(t)}) = \mathsf{g}p_i \cdot \frac{\partial p_i}{\partial P_{i0}^{(t)}} = \mathsf{g}p_i \cdot J_i^{(t-1)} \tag{4.19}$$

The corresponding formula for $j \neq 0$ comes directly from line 10 of RELAXATION (p. 125):

$$\begin{aligned}
\mathsf{g}(P_{ij}^{(t)}) = \mathsf{g}(J_j^{(t)}) \cdot \frac{\partial J_j^{(t)}}{\partial P_{ij}^{(t)}} &= \mathsf{g}(J_j)^{(t)} \cdot J_i^{(t-1)} && (4.20)\\
&\ (\mathsf{g}J)_j^{(t)} \cdot J_i^{(t-1)} \text{ by inductive hypothesis} && (4.21)
\end{aligned}$$

Finally, consider $k \neq i$ and any j: $P_{kj}^{(t)}$ is never used in the computation, so

$$\mathsf{g}(P_{kj}^{(t)}) = 0 \tag{4.22}$$

In summary,

$$\mathsf{g}(P_{kj}^{(t)}) = \begin{cases} 0 & \text{if } k \neq i \\ J_i^{(t-1)} \cdot \mathsf{g}p_i & \text{if } k = i \text{ and } j = 0 \\ J_i^{(t-1)} \cdot (\mathsf{g}J)_j^{(t)} & \text{if } k = i \text{ and } j \neq 0 \end{cases} \tag{4.23}$$

Line 9 and lines 12–14 of BACK-RELAXATION (p. 125) handle exactly these three cases, respectively. In light of equation (4.23), they may be read as arranging that for all k and j,

$$\begin{aligned}
(\mathsf{g}P)_{kj}^{(t-1)} &= \mathsf{g}(P_{kj}^{(t)}) + (\mathsf{g}P)_{kj}^{(t)} && (4.24)\\
&= \mathsf{g}(P_{kj}^{(t)}) + \mathsf{g}(P_{kj}^{(t+1)}) + \cdots + \mathsf{g}(P_{kj}^{(T)}) \text{ by inductive hypothesis} && (4.25)
\end{aligned}$$

demonstrating the second part (4.12) of the claim.

4.3.5 Computing the Gradient With Respect to θ Instead

As explained at the start of §4.3.3, the basic back-relaxation algorithm applies to a model parameterized by the transition probabilities P_{ij}. It computes the gradient with respect to those probabilities.

But in a standard transformation model, the transition probability matrix P is defined as P_θ from equation (4.2). That is, the P_{ij} are themselves parameterized by $\vec{\theta}$. So we only really care about finding $\nabla\tilde{f} = \mathsf{g}\vec{\theta}$.

It is straightforward to convert $\mathbf{g}P$ to $\mathbf{g}\vec{\theta}$ by differentiating equation (4.8) (whose first term is independent of P, and whose second term depends on $\vec{\theta}$ only through P):

$$\begin{aligned}\mathbf{g}\theta_h &= -\theta_h/\sigma^2 + \sum_{ij} \mathbf{g}P_{ij} \cdot \frac{\partial P_{ij}}{\partial \theta_h} &(4.26)\\ &= -\theta_h/\sigma^2 + \sum_{ij} \mathbf{g}P_{ij} \cdot P_{ij} \cdot (F_{ij}^h - \sum_{j'} P_{ij'} F_{ij'}^h) &(4.27)\end{aligned}$$

However, it would be a waste of space to compute and store $\mathbf{g}P$ explicitly. Instead of accumulating a large matrix $\mathbf{g}P$ of partial derivatives, we would like to accumulate only the vector $\mathbf{g}\vec{\theta}$, which is usually much smaller.

To do so, recall that $\mathbf{g}P_{ij}$ is computed as a sum $\sum_t \mathbf{g}(P_{ij}^{(t)})$. We can apply the summands toward $\mathbf{g}\vec{\theta}$ as soon as we discover them, rather than wait until they are added up. Rewriting equation (4.26),

$$\mathbf{g}\theta_h = -\theta_h/\sigma^2 + \sum_{i,j,t} \mathbf{g}(P_{ij}^{(t)}) \cdot \frac{\partial P_{ij}}{\partial \theta_h} \qquad (4.28)$$

This suggests that we modify Back-Relaxation as follows. There is no matrix $\mathbf{g}P$ but rather a vector $\mathbf{g}\vec{\theta}$. Initialize each $\mathbf{g}\theta_h$ to $-\theta_h/\sigma^2$. Whenever the original version of Back-Relaxation added a term to some accumulator $\mathbf{g}P_{ij}$—call this term $\mathbf{g}P_{ij}^{(t)}$—the new version should instead add $\mathbf{g}P_{ij}^{(t)} \cdot \frac{\partial P_{ij}}{\partial \theta_h}$ to $\mathbf{g}\theta_h$, for each h.

While this solution saves space as promised, it is far too slow to be practical. Each simple addition in the original version has been replaced with a nested loop! Not only does each addend $P_{ij}^{(t)}$ now require an iteration over all features h, but each of those features h demands an inner loop over the arcs ij' that compete with ij, since $\frac{\partial P_{ij}}{\partial \theta_h} = P_{ij} \cdot (F_{ij}^h - \sum_{j'} P_{ij'} F_{ij'}^h)$.

So a further trick is necessary. Rewrite equation (4.27) in terms of an intermediate vector $\vec{m}$:

$$\begin{aligned}m_i &= \sum_{t,j} \mathbf{g}P_{ij}^{(t)} \cdot P_{ij} &(4.29)\\ \mathbf{g}\theta_h &= -\theta_h/\sigma^2 + \left(\sum_{i,j,t} \mathbf{g}P_{ij}^{(t)} P_{ij} F_{ij}^h\right) - \left(\sum_{ij'} m_i P_{ij'} F_{ij'}^h\right) &(4.30)\end{aligned}$$

This leads to the following, much faster modification of Back-Relaxation:

- Initialize $\vec{m} = 0$ and $\mathbf{g}\theta_h = -\theta_h/\sigma^2$ for each h.
- Whenever the original version of BACK-RELAXATION added a term $\mathbf{g}P_{ij}^{(t)}$ to some accumulator $\mathbf{g}P_{ij}$, the new version should instead
  - add $\mathbf{g}P_{ij}^{(t)} \cdot P_{ij}$ to m_i.
  - also add $\mathbf{g}P_{ij}^{(t)} P_{ij} F_{ij}^h$ to $\mathbf{g}\theta_h$, for each h. This only requires an single loop, not a nested loop, and crucially it is only necessary to loop over the small set of features h that actually appear on arc ij (that is, $F_{ij} \neq 0$).
- After BACK-RELAXATION finishes, iterate over all vertices i such that $m_i \neq 0$. For each such vertex, consider each of its out-arcs ij', and for each feature h of that out-arc (that is, $F_{ij'}^h \neq 0$), subtract $m_i P_{ij'} F_{ij'}^h$ from $\mathbf{g}\theta_h$.

This computation of $\vec{\mathbf{g}\theta}$ is still slower than the original computation of $\mathbf{g}P$, but only by a factor equal to the average number of features per arc. That is simply the increase in the size of the model. After all, the new computation must consider all the non-zero entries F_{ij}^h, not just the non-zero entries P_{ij}, and this increase exactly accounts for the slowdown.

4.3.6 Allowing P to Change

It is not strictly necessary for P to remain constant as the relaxation algorithm runs. We will actually take advantage of this fact in the experiments (see §6.5.3).

In the metaphor of §4.2.1, the weather may change at any time between kicks of the anthills, changing the ants' preferences about which arcs to take. This does not change the number of ants in the graph, so it does not threaten to yield a non-normalized probability distribution. What is important is that if P changes, its row sums remain at 1.

In the terminology of §4.3.4, the different mentions $P_{ij}^{(t)}$ of P_{ij} are formally different variables, and need not have the same value. So it is all right to change P between iterations t of the main loop.

If P is changed during relaxation, how does this affect back-relaxation? The changes are minor. Obviously back-relaxation must use the same $P_{ij}^{(t)}$ values as relaxation did, so it is necessary to keep some record (analogous to $Stack1$) of when and how P changed.

Otherwise, the only changes are to §4.3.5, which assumes a particular fixed definition of P, not a changing one. In particular, equation (4.26) must be changed to compute $\frac{\partial P_{ij}^{(t)}}{\partial \theta_h}$ as appropriate.

4.4 Handling Perturbed Models

The experiments of Chapter 6 use the "perturbed transformation models" of §3.9, since in early experiments, perturbations led to slightly better results than per-event features.

There are different ways of adapting the algorithms to handle perturbed models; see §8.4.1 for another. The method used in the experiments is to reparameterize the flow multiplier parameters π_i as "flow adder" parameters δ_i. The advantage of this technique is that simple stopping conditions for relaxation (§4.2.2) remain sensible ones.[12]

The effect of a perturbation π_i is to inflate the outflow of vertex i by a factor of $\exp \pi_i$. In other words, $I_i N$ ants enter the node and $(\exp \pi_i) \cdot I_i N$ ants leave it. This is equivalent to injecting $\delta_i N$ extra ants into the graph at node i, where

$$\delta_i \stackrel{\text{def}}{=} (\exp \pi_i - 1) I_i \tag{4.31}$$

δ_i is essentially a **flow adder**, as opposed to the flow multiplier $\exp \pi_i$. That is, it measures the difference between outflow and inflow instead of their ratio. Instead of adjusting the π_i values, our strategy is to adjust the δ_i values. This does not change the model, only the parameterization. Thus the prior probability of a δ_i value is still defined by the probability (under a Gaussian) of the π_i value that would give rise to it, namely

$$\pi_i = \ln(1 + \frac{\delta_i}{I_i}) \tag{4.32}$$

Perturbed models need to be normalized because of the extra mass added to the graph. Equation (3.30) is equivalent to defining equation (4.8)'s normalizing factor Z as $1 + \sum_{i=1}^{n} \delta_i$.

[12] In relaxation, we usually gamble that small J_i need not be propagated. This is sometimes a losing gamble, since the logarithmic objective function can be quite sensitive to small absolute errors in $\vec{p}$ (see §4.1.6). Flow multipliers make the gamble slightly riskier, because a small J_i can be multiplied as it is propagated; one cannot tell in advance whether kicking a small anthill will eventually turn it into an important army. With flow adders, by contrast, it is known from the start how many extra ants are being added to the graph, and it only remains to propagate them. What we lose with flow adders is knowledge of the exact prior cost of the extra ants; in general I_i is underestimated so this cost is overestimated.

To run relaxation with flow adders, we conceptually just want to initialize $\vec{J}$ not to $\vec{b}$ but to $\vec{b}+\vec{\delta}$, and relax as usual. However, this may lead to negative elements in the vector $\vec{J}$. In this case a subtle danger arises: the elements of $\vec{p}$ may subsequently decrease as well as increase during the course of relaxation (compare §4.2). When relaxation is halted, the elements of $\vec{p}$ may therefore be overestimates rather than underestimates, so possibly $\sum_i p_i > 1$, giving the model an unfair advantage in evaluation.

One possible solution is to normalize $\vec{p}$ more carefully, correcting for any (positive or negative) probability mass that was left unpropagated in $\vec{J}$ when relaxation halts. However, even this scheme is still not entirely safe. It can lead to a "probability distribution" $\vec{p}$ that does sum to 1 but is negative at some vertices. If the test data happens not to include observations of those vertices, one might not notice a problem, but the probabilities of the remaining vertices would be unfairly inflated.

The safe solution is to make sure that $\vec{J}$ is, in fact, uniformly non-negative at all times. This is done by introducing the extra flow δ_i into the graph not necessarily at the start of relaxation, but sometime during it. In other words, when $\delta_i < 0$, the algorithm should wait to remove ants from i until they have actually arrived there. Usually δ_i is positive, and can be added to J_i at the start. If not, up to $-J_i$ of it can be added whenever J_i becomes positive, and the rest, if any, waits around until J_i becomes positive again:

```
(* J_i has just been changed *)
if δ_i ≠ 0
    d := max(δ_i, −J_i) pp
    decrement δ_i by d
    increment J_i by d
    increment Z by d
```

Note that only the part of δ_i that is used is added to the normalizing factor Z (which is initially 1).

Back-relaxation must now compute $\mathbf{g}\delta_i$ as part of the gradient (as well as still computing $\mathbf{g}P$ or $\mathbf{g}\vec{\theta}$). If we ignore the normalizing factor and the prior, this is already computed: it is simply $\mathbf{g}J_i^{(t)}$, where t is the step on which δ_i becomes 0 (is fully discharged).[13] The full

[13] If δ_i was never fully discharged, then it must have represented a flow adder that was more negative than the inflow I_i was positive. Such an extreme flow adder could never arise legally from any flow multiplier (i.e., equation (4.32) has no solution). So in this case, relaxation is trying to evaluate the function $\tilde{f}(\vec{\theta}, \vec{\delta})$

result is

$$\mathbf{g}\delta_i = \mathbf{g}J_i{}^{(t)} + \underbrace{-\frac{\sum_i s_i}{Z}}_{=\mathbf{g}Z\cdot\frac{\partial Z}{\partial \delta_i}} + \underbrace{-\frac{\pi_i}{\tau^2}\cdot\frac{1}{I_i+\delta_i}}_{=\mathbf{g}p_i\cdot\frac{\partial \pi_i}{\partial \delta_i}} \tag{4.33}$$

where π_i is defined by equation (4.32) and τ^2 is the variance of the Gaussian prior on π_i.

4.5 Optimizations Used in the Experiments

4.5.1 The Double Stack

§4.2.2 recommended using a priority queue to determine the order in which to relax nodes when evaluating the objective function $\tilde{f}(\theta)$. However, this means that the relaxation order will depend on θ, possibly making $\tilde{f}$ a bumpy, nondifferentiable surface on which optimization is difficult.

To keep the surface smooth, it helps to use the same relaxation order for many evaluations of $\tilde{f}$ in a row. One implementation exploits the mirror symmetry between relaxation and back-relaxation. Just as Back-Relaxation pops *Stack*1 to visit vertices in the reverse order of the previous Relaxation, Relaxation can pop *Stack*2 to visit vertices in the reverse order of the previous call to Back-Relaxation. (This is why the latter was defined to push vertices on *Stack*2 as it processed them.)

Thus, a series of evaluations $\tilde{f}(\theta), \nabla\tilde{f}(\theta), \tilde{f}(\theta'), \nabla\tilde{f}(\theta'), \ldots$ first creates *Stack*1 by a priority queue, and then shifts the all nodes back and forth between *Stack*1 and *Stack*2 like a Slinky. The double stack can be regarded instead as an array, with relaxation traversing the array forward (writing J-values) and back-relaxation traversing it backward (writing $\mathbf{g}J$-values, whose use we will see in a moment).[14]

4.5.2 Path Pruning

A significant optimization comes from a graphical view of the algorithms. Relaxation may be regarded as exploring the graph from Start (or more generally from $\vec{b}$), in search

outside its legal domain. The gradient descent algorithm that presumably called relaxation has strayed out of the legal domain of $\tilde{f}$; it must be able to handle this case, e.g., by reducing its stepsize and backtracking.

[14]However, in practice an array implementation this would make it difficult to drop useless vertices heuristically from the stack (footnote 15 on p. 133). See footnote 23 on p. 140 for a note on implementing the necessary stacks.

of vertices i such that p_i affects the objective function (during training) or the evaluation function (during testing). A path that does not eventually lead to such a vertex is not worth following, because it will have no effect on the function being computed (equation (4.8) or (4.9)).

The first call to RELAXATION must explore the graph blindly unless the graph is known to have some special structure, but subsequent calls to RELAXATION only need to follow paths that turned out to be fruitful. In fact, BACK-RELAXATION specifically propagates back the information about whether any paths from vertex j will be fruitful, and indeed how fruitful. If ${\rm g}J_j{}^{(t)} = 0$, then it is not worth propagating probability mass to vertex j on step t, since the variable $J_j^{(t)}$ affected by this propagation will have no effect on the objective function.

Thus, RELAXATION can benefit at step t from access to the value of ${\rm g}\vec{J}^{(t)}$ from the previous call to BACK-RELAXTION. It can ensure such access by always popping $(i^{(t-1)}, {\rm g}J_i{}^{(t)})$ from $Stack2$ into the variables $(i, {\rm g}J_i)$ at line 6 of RELAXATION (p. 124). This policy ensures inductively that during step $t = 1, 2, \ldots T$ of relaxation, the variable ${\rm g}\vec{J}$ equals ${\rm g}\vec{J}^{(t)}$ from the previous call, since ${\rm g}\vec{J}^{(t)}$ is identical to ${\rm g}\vec{J}^{(t-1)}$ in all but its $i^{\rm th}$ component.

Thus, there is no need at all to propagate from i to j on step 0 if ${\rm g}J_j{}^{(t)} = 0$. One could also heuristically skip such propagation if ${\rm g}J_j{}^{(t)}$ is non-zero but small. In fact, the importance to $\tilde{f}$ of that propagation can be measured by $J_i^{(t-1)} \cdot P_{ij}^{(t)} \cdot {\rm g}J_j{}^{(t)}$, a product that one can use to measure whether propagation along ij is worthwhile.[15]

To compute the gradient quickly and accurately, the subsequent back-relaxation must remember which arcs were ignored by relaxation—i.e., which entries of $P^{(t)}$ were treated

[15] Before computing any such products, it is worth checking first whether $J_i^{(t-1)} \cdot {\rm g}J_i{}^{(t-1)}$ is big enough that it is worth relaxing i at all. If this value is small, then J_i is not going to have much effect anyway on the objective function, so we may as well skip it entirely. In this case we simply avoid pushing i onto $Stack1$, so it will not be considered during back-relaxation or the subsequent relaxations. Of course, none of these tricks can be used when running relaxation for the first time, because no estimate of ${\rm g}J_i{}^{(t-1)}$ is available yet.

$J_i^{(t-1)} \cdot {\rm g}J_i{}^{(t-1)}$ also seems at first like a good definition of the priority of vertex i when it is time to determine a new relaxation order. Unfortunately ${\rm g}\vec{J}$ is not available when choosing the new order (since it is found by popping $Stack2$, whereas the new order is found by popping a priority queue). Nor would ${\rm g}J_i{}^{(t-1)}$ as determined from the old order be a plausible guess as to its true value during the (quite different) new order. A reasonable approach in the same spirit is to define the priority of i (on step t) as the product $J_i^{(t-1)} \cdot (\epsilon + \max_{t'=1}^{T} {\rm g}J_i{}^{(t'-1)})$ for some $\epsilon > 0$. This says that i is an especially useful vertex to relax in the new order if it was ever a useful vertex to relax during the old order; the use of ϵ also allows new useful vertices to emerge.

as if they were zero—and ignore those arcs too.[16] In fact, it is usually more economical to store the list of arcs that were *not* ignored. In the experiments of Chapter 6, most arcs can be ignored most of the time. (After all, given the number of crazy transformations in the graph, most paths from a given frame do not hit any legitimate lexical entry for English, let alone one that was actually observed in the dataset.)

Thus, in the double stack, $i^{(t-1)}$ should carry around with it not only the value $J_i^{(t-1)}$ or $\mathrm{g}J_i{}^{(t)}$, but also the list of arcs ij that should be considered when $i^{(t-1)}$ is relaxed at step t. (In other words, $Stack1$ and $Stack2$ now consist of triples (vertex, real, arc list).) This list is used again during the corresponding step of back-relaxation; in both cases, the line "for each child $j \neq 0$ of i" simply iterates over this list.

For efficiency, the same arc lists may be reused for many passes in a row without being recomputed.[17] This also keeps $\tilde{f}(\theta)$ smooth (differentiable), just as keeping the same relaxation order from pass to pass does (§4.5.1). One does not want infinitesimal changes in θ to have a discrete effect on either the relaxation order or the choice of arcs to skip, as this would lead to discrete jumps in $\tilde{f}(\theta)$.

4.5.3 Templates: Exploiting Redundancy in the Transformation Graph

A final optimization makes use of redundancy in the computation. For example, in the lexicon smoothing model used in the experiments of Chapter 6, most words do not differ much from the "typical" word. We would like to save time by computing a **template** for the typical word, and then handle each word by computing only its differences from the template.

[16] If relaxation only skips arcs where $J_i^{(t-1)} \cdot P_{ij}^{(t)} \cdot \mathrm{g}J_j{}^{(t)}$ is actually zero, which is the policy in the experiments of Chapter 6, then there is no effect on $\tilde{f}$, so the only reason for back-relaxation to skip the same arcs is to achieve the same speedup. If relaxation also heuristically skips arcs where this product is small, then back-relaxation *must* skip the same arcs in order to get an accurate gradient of the approximation $\tilde{f}$.

[17] As a result, one should be somewhat conservative about ignoring arcs or halting relaxation early. It has a long-term effect when one chooses not to propagate J_i along an arc from i. There may be good reasons to skip that arc on this pass (or simply to halt relaxation before relaxing i), but weak paths may sometimes become strong on later passes, as the parameters $\vec{\theta}, \vec{\delta}$ change.

For example, suppose $\delta_i = 0$ on the current pass. Then $J_i^{(0)} = 0$ so $J_i^{(0)} \cdot P_{ij} \cdot \mathrm{g}J_j{}^{(1)}$ is certainly zero, and one might therefore wish to ignore all arcs from i on step 1 on the grounds that they are not contributing to the objective function on this relaxation pass. But in fact the arcs may be useful to *future* passes. *Not* ignoring them may result in a non-zero value for $\mathrm{g}\delta_i$ during back-relaxation, yielding $\delta_i \neq 0$ (and so making them useful) on the next pass.

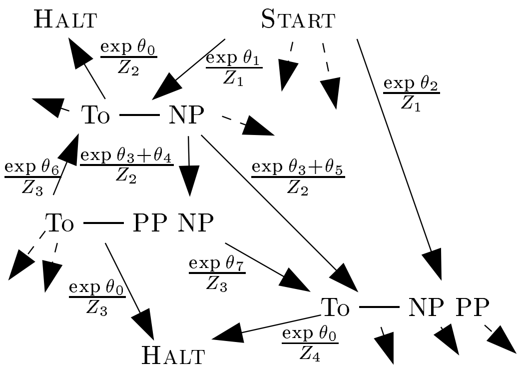


Figure 4.1: A fragment of a transformation model whose events are frames rather than lexical entries. Perturbation parameters are not shown.

For clarity, let us confine the discussion to that example. A more general presentation will be given in §8.6.

4.5.3.1 Motivation for Templates

The experiments need to evaluate $\Pr(f \mid \ell)$ for various frames f and headwords ℓ. They use a (perturbed) transformation model of lexical entries like that shown in Fig. 1.3 on p. 21,[18] where the first step of a random walk samples $\Pr(\ell)$ (by transitioning to $\textsc{Start}_\ell$) and the remaining steps sample $\Pr(f \mid \ell)$.

Because the subgraphs reachable from the various $\textsc{Start}_\ell$ vertices are *isomorphic*, the process of sampling $\Pr(f \mid \ell)$ is largely independent of ℓ. One can regard it as sampling f from the subgraph shown in Fig. 4.1, whose transition probabilities are independent of ℓ but whose perturbation parameters are specific to ℓ.[19] In other words, we will identify every vertex in Fig. 1.3 (except START) with the corresponding vertex in Fig. 4.1.

To compute the distribution $\Pr(f \mid \ell)$, one may perform relaxation on Fig. 4.1, initializing $\vec{J}^{(0)}$ to include the flow adders that are specific to ℓ. The desired distribution is

[18] Fig. 1.3 actually shows an ordinary model, not a perturbed one. In the perturbed model actually used in the experiments and discussed here, the per-event arc weights such as θ_8 and θ_9 are replaced by perturbations π_8 and π_9.

[19] To put this more formally, f is sampled from a perturbed transformation model of frames: a family of distributions $\Pr_{\theta,\pi}(f)$, defined by Fig. 4.1, where $\vec{\theta}$ effectively specifies the language and $\vec{\pi}$ effectively specifies the headword. $\vec{\theta}$ must be learned for the overall language, and an appropriate $\vec{\pi}$ must be learned for each headword, so that $\Pr(f \mid \ell)$ can be defined as some $\Pr_{\theta,\pi}(f)$. The training data are samples of frames from multiple headwords of the same language.

returned in $\vec{p}$. Such a relaxation must be performed for many different values of ℓ in order to evaluate the objective function. However, a good deal of the work is repeated for every ℓ: namely, 1 is always propagated along the many paths from START. The idea is to do this part of the work in advance and reuse it for every ℓ.

4.5.3.2 Relaxation Using a Template

The "template computation" of $\vec{p}$ is defined to simply set $\vec{J}^{(0)} = \vec{b}$ and run relaxation on Fig. 4.1. This computes $\vec{p}$ for the "typical" word—one with no perturbations at all—by propagating 1 from START. The later propagation of flow adders for each ℓ then refers to the *Stack*1 that was set by the template computation.

We will use $\mathtt{t}x$ to refer to the value of a variable x in the template computation. $\mathtt{s}_\ell x$ denotes the value x would have if we were to do the complete relaxation corresponding to subgraph ℓ in the ordinary way: that is, if we were to set $\vec{J}^{(0)} = \vec{b} + \mathtt{s}_\ell\vec{\delta}$ and run relaxation, where $\mathtt{s}_\ell\vec{\delta}$ denotes the vector of flow adders for word ℓ. Finally, $\Delta_\ell x \stackrel{\text{def}}{=} \mathtt{s}_\ell x - \mathtt{t}x$. In general it is more efficient to compute $\mathtt{s}_\ell x$ as $\mathtt{t}x + \Delta_\ell x$ than to compute it from scratch.

The strategy is to compute each $\mathtt{s}_\ell\vec{p}$ by relaxation, using exactly the same vertex order that was used in the computation of $\mathtt{t}\vec{p}$. This order was conveniently stored in $\mathtt{t}Stack1$. Since $\mathtt{t}Stack1$ also stores the values $\mathtt{t}J_i$, the computation has access to those values, and can save time by propagating only the differences $\Delta_\ell J_i$. Thus the relaxation for $\mathtt{s}_\ell\vec{p}$ is as follows (cf. RELAXATION on p. 123):

$\Delta_\ell Stack1 := \emptyset$
$\Delta_\ell\vec{p} := 0$; $\Delta_\ell\vec{J} := 0$; $\Delta_\ell Z := 0$
for each $(i, \mathtt{t}J_i)$ on $\mathtt{t}Stack1$ *(* in order they were pushed (not reversed), and without popping *)*
    if $\delta_i \neq 0$ *(* this portion is essentially copied from §4.4 *)*
        $d := \max(\delta_i, -(\mathtt{t}J_i + \Delta_\ell J_i))$
        decrement δ_i by d
        increment $\Delta_\ell J_i$ by d
        increment $\Delta_\ell Z$ by d
    if $\Delta_\ell J_i \neq 0$
        push $(i, \Delta_\ell J_i)$ onto $\Delta_\ell Stack1$
        $z := \Delta_\ell J_i$; $\Delta_\ell J_i := 0$

12. increment $\Delta_\ell p_i$ by $z \cdot P_{i0}$
13. **for** each child $j \neq 0$ of i
14. increment $\Delta_\ell J_j$ by $z \cdot P_{ij}$
15. **return** $\mathtt{t}\vec{p} + \Delta_\ell \vec{p}$ (* *also return* $\mathtt{t}Z + \Delta_\ell Z$ *)

Line 9 is actually a little more complicated than shown. If $\Delta_\ell J_i = 0$ but could become non-zero with a change to the parameters, then we do want to relax i, specifically so that i will be pushed on $\Delta_\ell Stack1$. Under the discipline of §4.5.3.4, this ensures that back-relaxation will compute and send back $\mathtt{g}\Delta_\ell J_i \neq 0$, allowing the parameters to change and $\Delta_\ell J_i$ to become nonzero. It also ensures that i will be relaxed again on the next pass—at which time relaxation will actually be useful since $\Delta_\ell J_i \neq 0$.

Conceptually, one should regard a "contingent" zero—such as a free parameter δ_i that is zero only at the moment—as an infinitesimal. Infinitesimals act like nonzeros in their effect on the algorithm's bookkeeping, but they are so close to zero that they have no effect on the algorithm's answer.[20]

Thus, an infinitesimal acts just like zero except for purposes of line 4 or line 9, and except that it yields infinitesimals rather than zeroes when added to zero or multiplied by non-zero. If δ_i is an infinitesimal at line 4, then $\Delta_\ell J_i$ becomes nonzero (and δ_i becomes 0) and i must now be relaxed at line 9. Relaxing i makes the children of i have nonzero J-values, so they will have to be relaxed in turn (and pushed on $\Delta_\ell Stack1$).

4.5.3.3 Benefits of Templates

Thanks to the use of the template, each computation of $\tilde{f}$ in the experiments took less time to compute the differences for all 3607 training headwords, *in total*, than to compute the template. The algorithm would have been far slower if the computation of the template had essentially been repeated for all 3607 headwords.

The benefit comes at line 9, which in most cases allows the relaxation of i to be skipped entirely. For example, relaxing START enqueues all children of START. A straightforward computation of $\Pr(\cdot \mid \ell)$ would have to then relax all children of START, whereas line 9

[20] In the methods without templates, there is no such thing as a contingent zero, because $P_{ij} \neq 0$ iff the arc ij exists, regardless of parameters.

makes it possible to relax only the children that have flow adders, since the rest are no different from the template. In the experiments of Chapter 6, this is the difference between relaxing all observed frames while computing $\Pr(\cdot \mid \ell)$ and relaxing only those frames that were observed *with headword* ℓ.

Another important benefit emerges from a synergy with path pruning (§4.5.2). Recall that the first call to relaxation is slow, because it must explore the graph from START in search of observed vertices. It would be a waste to explore each subgraph of Fig. 1.3 separately, since they are identical. Again, the work can be combined across subgraphs. Running back-relaxation immediately on the template (see below) will detect the many arcs that are useless for *all* ℓ. That is, they do not appear in a path to any observed frame in Fig. 4.1.

Back-relaxation on the template records lists of just the useful arcs, as usual, on $\mathtt{t}Stack1$. Now the subgraphs can take advantage of this work. Even the first relaxation for subgraph ℓ need only explore along the arcs that were useful in the template—a significant savings. Back-relaxation on subgraph ℓ will winnow this set of arcs further, to just those that appear in a path to some observed lexical entry headed by word ℓ.

4.5.3.4 Back-Relaxation Using a Template

To perform back-relaxation, the computation must be run in reverse as usual. Thus, one must perform back-relaxation on each subgraph ℓ, and finally back-relaxation on the template.[21]

Back-relaxation on the subgraphs is essentially like back-relaxation for any model: it propagates the base-case gradient $\mathtt{g}\Delta_\ell p_i = \mathtt{gs}_\ell p_i = \frac{\mathtt{s}_\ell s_i}{\mathtt{s}_\ell p_i}$ backward from the frames observed with headword ℓ. It differs in the obvious ways from BACK-RELAXATION on p. 123:

$\Delta_\ell Stack2 := \emptyset$

$\mathtt{g}\vec{\Delta}_\ell J := 0$ *(* should set* $\mathtt{g}P := 0$ *only once before back-relaxing all subgraphs and template *)*

repeat until $\Delta_\ell Stack1 = \emptyset$

  pop $(i, \Delta_\ell J_i)$ from $\Delta_\ell Stack1$

[21] The remark about path pruning in §4.5.3.3 suggested an initial run relaxation and back-relaxation on the template, solely to find potentially useful paths. The gradients computed by this are not valid and should be ignored. Thereafter, the discipline is to relax and back-relax all subgraphs in between relaxation and back-relaxation of the template.

push $(i, \mathtt{g}\Delta_\ell J_i)$ onto $\Delta_\ell Stack2$

$z := 0$

increment $\begin{cases} z & \text{by } P_{i0} \cdot \mathtt{g}\Delta_\ell p_i \\ \mathtt{g}P_{i0} & \text{by } \Delta_\ell J_i \cdot \mathtt{g}\Delta_\ell p_i \end{cases}$

for each child $j \neq 0$ of i

  increment $\begin{cases} z & \text{by } P_{ij} \cdot \mathtt{g}\Delta_\ell J_j \\ \mathtt{g}P_{ij} & \text{by } \Delta_\ell J_i \cdot \mathtt{g}\Delta_\ell J_j \end{cases}$

$\mathtt{g}\Delta_\ell J_i := z$

(* *no return value; should return the final* $\mathtt{g}P$ *only after back-relaxing all subgraphs and template* *)

Back-relaxation on the template is exactly the same, except that Δ_ℓ is replaced by $\mathtt{t}$ throughout. Here the base-case gradient is given by $\mathtt{gt}p_i = \sum_\ell \mathtt{gs}_\ell p_i \cdot \frac{\partial \mathtt{s}_\ell p_i}{\partial \mathtt{t} p_i} = \sum_\ell \mathtt{gs}_\ell p_i$. The summation indicates that this is generally larger than for any single subgraph: after all, the template computation affects the final probabilities of lexical entries in many different subgraphs.

Notice that just as relaxation for subgraph ℓ only had to follow a few paths (the ones along which Δ_ℓ was propagated), back-relaxation for subgraph ℓ only has to follow those same paths backwards. Thus its pop move is to pop $(i, \Delta_\ell J_i)$ from $\Delta_\ell Stack1$. Moreover, it uses the popped $\Delta_\ell J_i$ in place of J_i when computing $\mathtt{g}P_{ij}$, since it is measuring the effect of the parameters P only on the subgraph-ℓ computation. This measurement is combined with back-relaxation on the template, which measures the effect of P on the template computation.

Subgraph ℓ is generally much faster to process than the template. This is because $\Delta_\ell Stack1$ is generally considerably shorter than $\mathtt{t}Stack1$. Also, once path pruning has been applied to $\Delta_\ell Stack1$ (§4.5.2), the arc lists it carries around are generally considerably shorter than the arc lists for corresponding entries on $\mathtt{t}Stack1$. This is because the subgraph-ℓ problem only needs to follow paths in Fig. 4.1 to frames that appear with word ℓ, whereas the template problem has to follow paths to frames that appear with any word.

A subtlety is glossed over in the pseudocode above. We would like to use our usual double-stack trick (§4.5.1), so that the next call to relaxation on subgraph ℓ will obtain its vertex order by repeatedly popping $(i, \mathtt{g}\Delta_\ell J_i)$ from $\Delta_\ell Stack2$. However, remember that relaxation will also need access to the corresponding values of $\mathtt{t}J_i$ on $\mathtt{t}Stack1$.[22] It is therefore necessary for vertex instances on the subgraph-ℓ stacks to maintain pointers to the corresponding vertex instances on the template stacks.[23]

[22] To avoid problems with flow adders, and also for a more direct reason if templates are generalized (§8.6.2.2).

[23] A good implementation is for every vertex token created during template or subgraph relaxation to be a record stored at some fixed address in memory (no matter what stack it is on). The record specifies the vertex; a real J or $\mathtt{g}J$ value as appropriate; the arc list; a pointer to the underlying template record; and a pointer to the next vertex on the same stack.

Chapter 5

Trees and Transformations

Previous chapters defined transformational lexicons and transformation models. In particular, §2.4 described a particular format for a transformational lexicon, and §3.7.2 sketched how to model it statistically with a transformation model.

But what is the practical use of such a model? In particular, how could we use it to parse better? This brief chapter fleshes out the answer (following §2.3), before we move on to actually fitting such a model in the next chapter. Old hands may want to skip ahead to §5.4.

5.1 Dependency Frames

We take lexical entries to be the flat, lexicalized context-free rules of §2.4.1. A lexicon lists the lexical entries of a language.

A **frame** (formally defined in §5.3.2) is a kind of template for lexical entries (not in the sense of §4.5.3!). It looks just like a lexical entry, except that the headword is a variable, written as —. For example, the lexical entries `S` $\rightarrow$ `NP ate NP PP` and `S` $\rightarrow$ `NP drank NP PP` are both instantiations of the same frame, `S` $\rightarrow$ `NP — NP PP`.

A frame specifies the dependents of the headword, their order, and their position relative to the head. The flat representation means that unlike a traditional subcategorization frame, it mentions all dependents—specifiers and adjuncts as well as complements. (The term "dependency frame" alludes to dependency grammar (Tesnière, 1959; Mel'čuk, 1988).)

A lexical entry can be regarded as an element (w, f) of $\mathsf{Words} \times \mathsf{Frames}$. Thus, the lexicon specifies for each word the set of frames that it can project. The transformations of §2.4.2 and §2.4.3 convert these frames into one another without changing the headword.

5.2 Probabilistic Lexicons

A probabilistic lexicon is a probability distribution $\Pr^{\text{lex}}(w, f)$ over the space of lexical entries, $\mathsf{Events} \stackrel{\text{def}}{=} \mathsf{Words} \times \mathsf{Frames}$.

Our principal interest is in the conditionalized distribution $\Pr^{\text{lex}}(f \mid w)$, which specifies for each word the frames that it is likely to project. Note that for any w, $\Pr^{\text{lex}}(\cdot \mid w)$ is itself a probability distribution, with $\sum_f \Pr^{\text{lex}}(f \mid w) = 1$. Transformational smoothing adjusts this distribution by shifting probability mass among the frames for w,[1] on the assumption that correlations observed among other words' frames also hold of w's frames.

For learning (smoothing), we will have to compare multiple possible lexicons. In this case we will write a given lexicon as a function $\Pr^{\text{lex}}_\theta(w, f)$, where θ is a large but finite vector of parameters that fully determines the function $\Pr^{\text{lex}}_\theta$ according to a transformation model (§3.2.2).

This chapter describes how to do three related things with a probabilistic lexicon:

assign probabilities to trees As shown below in §5.4.4, we may use the conditional distributions $\Pr^{\text{lex}}(f \mid w)$—together with a second function that describes the contextual probabilities of each word w—to construct a distribution $\Pr^{\text{tree}}$ over syntax trees. This is useful for training, parsing, and evaluation (§5.3.1).

induce a lexicon The goal of grammar learning is to determine θ such that $\Pr^{\text{lex}}_\theta$ is likely to be the lexicon of the target language. Given some evidence (sample lexical entries or trees), we choose the lexicon $\Pr^{\text{lex}}_\theta$ that has highest posterior probability according to Bayes' Theorem (§3.1.3).

[1]Why is probability mass shifted only among entries that share a headword w? Only because we do not currently consider transformations that can change the headword of an entry.

favor transformationally regular lexicons Bayes' Theorem recognizes that some lexicons are *a priori* more likely than others. We will model $\Pr^{\text{lex}}_\theta$ using a transformation model (Chapter 3). The prior on θ plays the role of Universal Grammar. The result is that

- *Any* θ has positive prior probability ($\Pr_{\text{prior}}(\theta) > 0$), so any lexicon $\Pr^{\text{lex}}_\theta$ can be learned given a sufficiently large data sample.
- The prior probability $\Pr_{\text{prior}}(\theta)$ is high to the extent that the lexicon $\Pr^{\text{lex}}_\theta$ is "transformationally regular." This means that many entries in the lexicon (i.e., individual probabilities $\Pr^{\text{lex}}_\theta(f \mid w)$) are roughly predictable from other entries, using a "small" set of transformations that apply at consistent rates throught the lexicon, transforming frames for a word into other frames for that word.

5.3 Tree Probabilities

5.3.1 The Importance of Tree Probabilities

The probabilistic lexicon $\Pr^{\text{lex}}(f \mid w)$ is useful only insofar as it helps us define a probability distribution over syntax trees, $\Pr^{\text{tree}}(T)$. We write $\Pr^{\text{tree}}_\theta$ for the particular distribution constructed from $\Pr^{\text{lex}}_\theta$.

- It is $\Pr^{\text{tree}}$ that is of primary interest for parsing. One can parse a sentence by choosing its parse T for which $\Pr^{\text{tree}}(T)$ is greatest.
- We can also use $\Pr^{\text{tree}}$ to define the likelihood $\prod_i \Pr^{\text{tree}}(T_i)$ of a corpus of trees $T_1, T_2, \ldots T_n$. This is an appropriate measure for evaluating and comparing statistical models of syntax, even models that do *not* make an explicit estimate of our probabilistic lexicon $\Pr^{\text{lex}}(f \mid w)$. A model is better than another if it assigns higher likelihood (equivalently, lower perplexity) to the correct parse trees for a set of randomly chosen test sentences.
- Finally, $\Pr^{\text{tree}}$ is also important for learning the probabilistic lexicon $\Pr^{\text{lex}}_\theta$. Ideally,

we would choose θ that maximizes the posterior probability

$$\begin{aligned}
\Pr(\theta \mid T_1, \ldots T_n) &\propto \Pr(T_1, \ldots T_n \mid \theta) \cdot \Pr(\theta) && (5.1)\\
&\stackrel{\text{def}}{=} \prod_i \Pr_\theta^{\text{tree}}(T_i) \cdot \Pr_{\text{prior}}(\theta) && (5.2)
\end{aligned}$$

where $\Pr_\theta^{\text{tree}}$ is constructed from $\Pr_\theta^{\text{lex}}$ and the trees $T_1, T_2 \ldots T_n$ are an observed sample from $\Pr_\theta^{\text{tree}}$. We cannot observe the transformational parameters θ directly, so we have to infer them from this sample.

The following sections develop a context-free model of $\Pr^{\text{tree}}$ that is defined as much as possible in terms of $\Pr^{\text{lex}}(f \mid w)$. Better techniques for estimating $\Pr^{\text{lex}}(f \mid w)$ from sparse data—such as transformational smoothing—should then lead to better estimates of $\Pr^{\text{tree}}$, and thus, one hopes, to more accurate parsing.

5.3.2 Formal Definition of Syntax Trees and Frames

Let Words be the set of terminal symbols and Cats a set of nonterminal categories.

A **frame** is an object of the form $X \to Y_1 \ldots Y_\ell$ —— $Z_1 \ldots Z_r$, for some $\ell \geq 0, r \geq 0$, and $X, Y_1, \ldots Y_\ell, Z_1, \ldots Z_r \in \mathsf{Cats}$.

We take a **syntax tree** to be any 4-tuple $T = \langle T.\mathsf{w}, T.\mathsf{f}, T.\mathsf{leftkids}, T.\mathsf{rightkids}\rangle$, where

- $T.\mathsf{w} \in \mathsf{Words}$ is the tree's **headword**;
- $T.\mathsf{f} = X \to Y_1 \ldots Y_\ell$ —— $Z_1 \ldots Z_r$ is the tree's **root frame**, with $X \in \mathsf{Cats}$ being called the tree's **category** and denoted by $T.\mathsf{cat}$ or $\mathsf{LHS}(T.\mathsf{f})$;
- $T.\mathsf{leftkids}$ is a sequence of syntax trees whose categories are $Y_1, Y_2 \ldots Y_\ell$ respectively;
- $T.\mathsf{rightkids}$ is a sequence of syntax trees whose categories are $Z_1, Z_2 \ldots Z_\ell$ respectively.

The sentence of which the tree is a parse is the tree's **yield**, found by reading off the sequence of headwords during an infix traversal:

$$\begin{aligned}
\mathsf{yield}(T) = {}& \mathsf{concat}(\mathsf{yield}(T.\mathsf{leftkids}[1]), \ldots \mathsf{yield}(T.\mathsf{leftkids}[\ell]), && (5.3)\\
& T.\mathsf{w}, \mathsf{yield}(T.\mathsf{rightkids}[1]), \ldots \mathsf{yield}(T.\mathsf{rightkids}[r]))
\end{aligned}$$

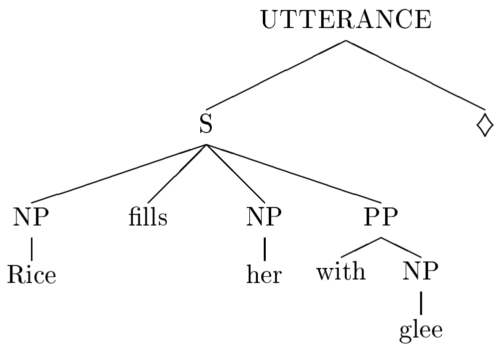


Figure 5.1: A sample tree generated as described in the text. $\diamondsuit$ is a special "end of sentence" symbol that serves as the headword of utterances.

An example is shown in Fig. 5.1. As motivated in §2.4.1, in this style of tree, each of the frames is "flat," in contrast to the more articulated "X-bar" style of representation. The `S` tree does not have explicit subtrees `VP`, $\overline{\texttt{V}}$, or `V` that share its head. Rather, all complements, subjects, and adjunct modifiers attach simultaneously to the headword **`fills`**, yielding the maximal projection `S`.

5.3.3 Stochastic Generation of Syntax Trees

5.3.3.1 A Context-Free Model

We first consider a lexicalized PCFG model of trees. A random syntax tree of category X_0 may be generated by the following multiply-recursive stochastic process (cf. (Charniak, 1997)).[2]

The process is defined by a set of **expansion probabilities** $\Pr^{\text{exp}}(w, f \mid X)$ where $w \in \mathsf{Words}$, $X \in \mathsf{Cats}$, and f is a frame with $X = \mathsf{LHS}(f)$. It has the following steps:

1. Choose a pair (w_0, f_0) at random, conditioned on X_0, by sampling from the distribution $\Pr^{\text{exp}}(w, f \mid X_0)$. Concretely, suppose that f_0 is $X_0 \to Y_1 \ldots Y_\ell$——$Z_1 \ldots Z_r$.

2. Recursively generate a list *leftkids* of ℓ syntax trees whose respective categories are $Y_1 \ldots Y_\ell$.

[2]Unfortunately, as is well known, this process is not guaranteed to terminate everywhere; so for some parameters of the stochastic process, the total probability of all (finite) trees can be < 1. We ignore this issue; Chi (1999) provides a formal remedy through renormalization, and also shows that the issue does not arise for maximum-likelihood estimates of the PCFG parameters.

3. Recursively generate a list *rightkids* of r syntax trees whose respective categories are $Z_1 \ldots Z_r$.

4. Return $\langle w_0, f_0, \mathit{leftkids}, \mathit{rightkids} \rangle$ as the generated tree.

Any parse tree T can be generated in this way by a unique sequence of expansion moves. The tree's probability, given its category, is the product of the probabilities of all the moves in this sequence, that is:

$$\begin{aligned} \Pr^{\text{tree}}(T \mid T.\mathsf{cat}) &= \Pr^{\text{exp}}(T.\mathsf{w}, T.\mathsf{f} \mid T.\mathsf{cat}) \cdot \prod_{T' \in T.\mathsf{leftkids} \cdot T.\mathsf{rightkids}} \Pr^{\text{tree}}(T' \mid T'.\mathsf{cat}) && (5.4) \\ &= \prod_{T' \text{ a subtree of } T} \Pr^{\text{exp}}(T'.\mathsf{w}, T'.\mathsf{f} \mid T'.\mathsf{cat}) && (5.5) \end{aligned}$$

We are especially interested in the special root category $X_0 = \mathtt{UTTERANCE}$. Define a **complete utterance** as a tree of this category. Abusing notation slightly, we use $\Pr^{\text{tree}}(T)$ as an abbreviation for $\Pr^{\text{tree}}(T \mid \mathtt{UTTERANCE})$, i.e., the probability that a random complete utterance has structure T as its correct analysis. In particular, this was the meaning of $\Pr^{\text{tree}}(T)$ in §5.3.1 above.

The example tree in Fig. 5.1 has probability

$$\begin{aligned} &\Pr(\Diamond : \mathtt{UTTERANCE} \rightarrow \mathtt{S} \text{ —— } \mid \mathtt{UTTERANCE}) \cdot \Pr(\mathtt{fills} : \mathtt{S} \rightarrow \mathtt{NP} \text{ —— } \mathtt{NP\ PP} \mid \mathtt{S}) \\ &\cdot \Pr(\mathtt{rice} : \mathtt{NP} \rightarrow \text{ —— } \mid \mathtt{NP}) \cdot \Pr(\mathtt{her} : \mathtt{NP} \rightarrow \text{ —— } \mid \mathtt{NP}) \\ &\cdot \Pr(\mathtt{with} : \mathtt{PP} \rightarrow \text{ —— } \mathtt{NP} \mid \mathtt{PP}) \cdot \Pr(\mathtt{glee} : \mathtt{NP} \rightarrow \text{ —— } \mid \mathtt{NP}) && (5.6) \end{aligned}$$

5.3.3.2 Enriching the Model With Context

The procedure just described makes the independence assumption that a subtree's head-word w and root frame f depend only on the subtree's category $X = \mathsf{LHS}(f)$. This is the standard PCFG assumption. However, recent statistical parsing models have also allowed dependence on other parts of the tree that govern the node in question.

We simply redefine the expansion probabilities to condition on more information: they now have the form $\Pr^{\text{exp}}(w, f \mid X, \gamma)$, where γ is (the result of) some function that picks out relevant contextual features from the part of the tree that was generated previously.

A common use of γ is to model "bilexical" selectional preferences (Eisner and Satta, 1999). For example, in Fig. 5.1, we might want to capture the fact that `rice` is a good subject of `fills`. We may do so by conditioning the choice of `rice` (and its frame $\texttt{NP} \to$ ——) not only on $X = \texttt{NP}$ but also on the lexical governor $\gamma = \texttt{fills}$. (See §5.5.4 below for more sophisticated variants.)

In general, we formally modify equation (5.4) to

$$\Pr^{\text{tree}}(T \mid T.\mathsf{cat}, \gamma) = \Pr^{\text{exp}}(T.\mathsf{w}, T.\mathsf{f}, \mid T.\mathsf{cat}, \gamma) \cdot \prod_{T_i \in T.\mathsf{leftkids} \cdot T.\mathsf{rightkids}} \Pr^{\text{tree}}(T_i \mid T_i.\mathsf{cat}, \gamma_i)$$

where γ_i may depend on $\gamma, T.\mathsf{w}, T.\mathsf{f}$, and the left sisters of T_i (defined as the trees that strictly precede T_i in the sequence $\mathsf{concat}(T.\mathsf{leftkids}, T.\mathsf{rightkids})$ of T's children).

For the bilexical case mentioned above, we would simply define γ_i (regardless of i) to be T.w, the headword of the minimal supertree of T'. This is the word on which T' depends: T' fills one of its adjunct or argument roles.

5.4 From Lexical Probabilities to Expansion Probabilities

5.4.1 Insertion and Projection

We have just made an independence assumption that allows us to compute $\Pr^{\text{tree}}$ as a product of expansion probabilities $\Pr^{\text{exp}}(w, f \mid X, \gamma)$, where $X = \mathsf{LHS}(f)$. The only remaining question is how to model the expansion probabilities (and what this has to do with probabilistic lexicons).

The usual approach in the literature (e.g., see §1.2.1.1) is to first decompose each expansion probability as follows, via the laws of conditional probability:

$$\Pr(w, f \mid X, \gamma) = \Pr(w \mid X, \gamma) \cdot \Pr(f \mid w, X, \gamma) \tag{5.7}$$

or, giving mnemonic names to the conditionalized distributions,

$$\Pr^{\text{exp}}(w, f \mid X, \gamma) = \Pr^{\text{ins}}(w \mid X, \gamma) \cdot \Pr^{\text{proj}}(f \mid w, X, \gamma) \tag{5.8}$$

The first factor is a **lexical insertion probability** that controls the choice of headword w, according to the selectional preferences of γ, and the second is a **frame projection probability** that controls the choice of arguments and modifiers for the headword w.

Applying this decomposition to equation (5.5), the probability of a tree or corpus T has the form

$$\prod_{T_i} \Pr^{\text{ins}}(w_i \mid X_i, \gamma_i) \cdot \prod_{T_i} \Pr^{\mathsf{proj}}(f_i \mid w_i, X_i, \gamma_i) \tag{5.9}$$

where T_i ranges over subtrees of T and $(w_i, f_i, X_i) = (T_i.\mathsf{w}, T_i.\mathsf{f}, T_i.\mathsf{cat})$.

The recent probabilistic parsing literature (see §1.2.1.1) is largely about how to model the conditional distributions $\Pr^{\text{ins}}$ and $\Pr^{\mathsf{proj}}$.

5.4.2 Smoothing the Projection Probabilities

The main focus of this thesis is $\Pr^{\mathsf{proj}}$ and how to smooth it using a transformation model.

A common independence assumption is that it is insensitive to γ:

$$\Pr^{\mathsf{proj}}(f \mid w, X, \gamma) = \Pr^{\mathsf{proj}}(f \mid w, X) \tag{5.10}$$

This means that each subtree's root frame f must match the headword w and LHS category X already generated for it, but that the choice is not further influenced by anything else in the tree.

We will adopt this assumption, which is intuitive and reasonable on linguistic grounds. (See §5.5.5 for further discussion.)

Some techniques for estimating $\Pr^{\mathsf{proj}}(f \mid w, X)$ are as follows:

- The simplest technique is to use a multinomial model, as if $\Pr^{\mathsf{proj}}(f \mid w, X)$ were the probability of landing on f when rolling a many-sided weighted die associated with (w, X). The maximum-likelihood estimate would be a ratio of observed counts,

$$\widehat{\Pr}^{\mathsf{proj}}(f \mid w, X) = \frac{\#(f, w, X)}{\#(w, X)} \tag{5.11}$$

- Standard backoff or discounting techniques can adjust the above maximum-likelihood estimate to compensate for the sparsity of the observed data, in particular to avoid estimating the probabilities of unseen events as 0.

- §6.6.1 and §6.6.3 discuss some more sophisticated approaches that take the internal structure of the frames f into account, so that "similar" frames are assigned similar probability estimates.

- Transformational smoothing, the contribution of this thesis, also takes the internal structure of the frames into account. It redistributes probability mass among frames that are transformationally related. Related frames receive related—but not necessarily similar—probability estimates.

  Given a transformationally smoothed lexicon $\Pr_\theta^{\text{lex}}(w, f)$, or just its conditionalization $\Pr_\theta^{\text{lex}}(f \mid w)$, we compute projection probabilities as

$$\Pr^{\text{proj}}(f \mid w, X) = \frac{\Pr_\theta^{\text{lex}}(f \mid w)}{\Pr_\theta^{\text{lex}}(X \mid w)} = \frac{\Pr_\theta^{\text{lex}}(f \mid w)}{\sum_{f':\mathsf{LHS}(f')=X} \Pr_\theta^{\text{lex}}(f \mid w)} \tag{5.12}$$

  where $X = \mathsf{LHS}(f)$.

Equation (5.12) is especially easy to compute if the transformation model has no transformations that change the category X (or headword w). In that case, the denominator $\Pr_\theta^{\text{lex}}(X \mid w)$ can usually be computed directly without a summation. It is simply the probability that if the transformational random walk (§3.3.1) begins by choosing a frame with headword w, then that frame's LHS is X. This can usually be determined by looking at the probabilities of arcs from START.

5.4.3 Smoothing the Insertion Probabilities

Surprisingly, transformational smoothing can also help improve the estimate of the insertion probabilities $\Pr^{\text{ins}}(w \mid X, \gamma)$.

Typically γ is a governing word, so that $\Pr^{\text{ins}}(\cdot \mid X, \gamma)$ is a distribution over words w that can fill the X role of γ. Just as for projection probabilities, several estimation techniques are available:

- As before, the simplest technique is to compute the maximum-likelihood estimate $\#(w, X, \gamma)/\#(X, \gamma)$.

- As before, the maximum-likelihood estimate can be improved with standard backoff or discounting techniques.

- One can back off using semantic clusters of words; see §5.5.1.2.

- Another backoff estimate is the **Naive Bayes** estimate:

$$\Pr(w \mid X, \gamma) = \frac{\Pr(w) \cdot \Pr(X \mid w) \cdot \Pr(\gamma \mid w)}{Z(X, \gamma)} \tag{5.13}$$

  where $Z(X, \gamma)$ is a normalizing factor chosen to make $\sum_w \Pr(w \mid X, \gamma) = 1$.

  One can estimate the "reversed" insertion term $\Pr(\gamma \mid w)$ using the same clustering or transformational smoothing techniques as above. The payoff is in the term $\Pr(X \mid w)$, which is obtained simply by marginalizing the probabilistic lexicon:

$$\Pr(X \mid w) \stackrel{\text{def}}{=} \sum_{f':\mathsf{LHS}(f')=X} \Pr_\theta^{\text{lex}}(f \mid w) \tag{5.14}$$

Transformational smoothing of the lexicon helps equation (5.14) just if the transformation model includes category-changing transformations. We now examine this point.

5.4.4 The Benefits of Category-Changing Transformations

Cross-category transformations, while linguistically useful (§2.4.3), are not used in the experiments presented here (see §2.4.2). However, they can help estimate both projection and insertion probabilities. This would be particularly helpful for derived categories like S/NP, where observed data tend to be sparse.

First consider projection probabilities. Suppose that during the stochastic generation of a tree (§5.3.3), we need to sample $\Pr^{\mathsf{proj}}(\cdot \mid \texttt{devoured}, \texttt{S/NP})$, in order to project an S/NP frame headed by devoured. This distribution is partly predictable from $\Pr^{\mathsf{proj}}(\cdot \mid \texttt{devoured}, \texttt{S})$.

The two distributions are not similar in the sense of small divergence: we cannot back off from one to the other. But they are related by an extraction transformation. If w is a transitive verb, so that $\Pr^{\mathsf{proj}}(\texttt{S} \to \texttt{NP} \;—\; \texttt{NP} \mid w, \texttt{S})$ is large, then we would expect extracted frames to have largish projection probabilities as well: $\Pr^{\mathsf{proj}}(\texttt{S/NP} \to \texttt{NP} \;— \mid w, \texttt{S/NP})$ and $\Pr^{\mathsf{proj}}(\texttt{S/NP} \to \texttt{NP} \;—\; \texttt{NP/NP} \mid w, \texttt{S/NP})$.

The transformation graph of Fig. 5.2 captures this expectation. When sampling from $\Pr^{\mathsf{proj}}(\cdot \mid \texttt{devoured}, \texttt{S/NP})$, we might start a random walk with the frame $\texttt{S} \to \texttt{NP} \;—\; \texttt{NP}$ and transform it into $\texttt{S/NP} \to \texttt{NP} \;—$ by object extraction. The second frame inherits

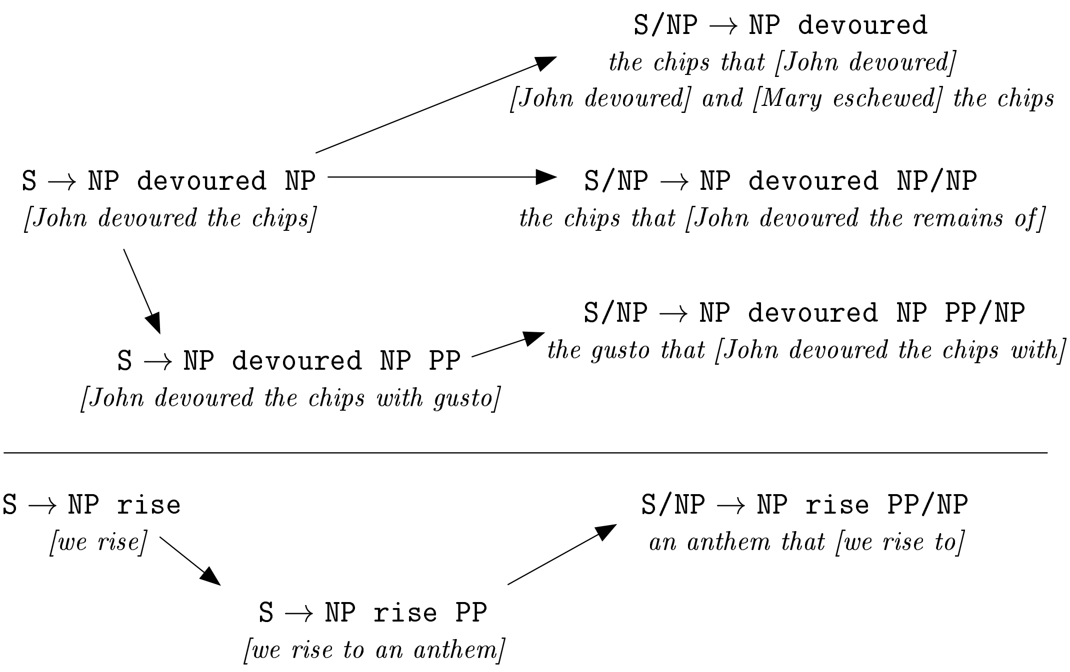


Figure 5.2: A transitive verb has more ways than an intransitive one to end up as the head of an `S/NP`. So $\Pr^{\text{proj}}$ will tend to select different `S/NP` frames for it, and $\Pr^{\text{ins}}$ should be more likely to choose it as the head of an `S/NP`.

probability from the first even though they have different LHS categories.[3]

Now suppose we were sampling instead from $\Pr^{\text{proj}}(\cdot \mid \texttt{rise}, \texttt{S/NP})$. Since `rise` is intransitive, $\texttt{S} \rightarrow \texttt{NP}$—— rather than $\texttt{S} \rightarrow \texttt{NP}$—— `NP` would be a far more probable choice of initial frame given the head. This cannot undergo object extraction, so we would have to take a more roundabout route to convert it into an `S/NP` frame. We might choose to transform it into $\texttt{S} \rightarrow \texttt{NP}$—— `PP` by adjunction and then $\texttt{S/NP} \rightarrow \texttt{NP}$—— `PP/NP` by extraction: *an anthem that [we rise to]*$_{\texttt{S/NP}}$.

So the transformation model says that `rise` tends to select for different `S/NP` frames than `devoured` does, precisely because it selects for different `S` frames.

How about insertion probabilities? It is also true that `S/NP` frames are rarely headed

[3]Note that to sample from $\Pr^{\text{proj}}(\cdot \mid \texttt{devoured}, \texttt{S/NP})$, a single random walk on the transformation graph does not suffice. Not all random walks halt at an entry of the form $\texttt{S/NP} \rightarrow \cdots \texttt{devoured} \cdots$. To sample from this conditional distribution, as defined in equation (5.12), we need a process that takes repeated random walks from START, trying again and again until it does halt at an entry of the appropriate form.

by `rise` in the first place. That is, the insertion probability $\Pr^{\text{ins}}(\texttt{rise} \mid \texttt{S/NP}, \gamma)$ is small. (Intuitively, this is because none of the `S/NP` frames headed by `rise` are easy to derive.) If the data are too sparse to learn this fact directly for a given γ, the Naive Bayes backoff estimate of §5.4.3 predicts it perfectly. A random walk among the frames for intransitive `rise` is unlikely to end up at an `S/NP` frame, so $\Pr(\texttt{S/NP} \mid \texttt{rise})$ is small in equation (5.14); this makes $\Pr^{\text{ins}}$ small in equation (5.13), as desired. For transitive `devoured` the estimate will be larger.

So the transformation model of the lexicon helps estimate insertion probabilities as well as projection probabilities.

5.4.5 Recombining Insertion and Projection

When using the Naive Bayes estimate of insertion probabilities, as above, it is actually convenient to estimate $\Pr^{\text{exp}} = \Pr^{\text{ins}} \cdot \Pr^{\text{proj}}$ directly:

$$
\begin{aligned}
\Pr(w, f \mid X, \gamma) &= \Pr(w \mid X, \gamma) \cdot \Pr(f \mid w, X) \text{ by equations (5.7) and (5.10)} && (5.15)\\
&= \frac{\Pr(w) \cdot \Pr(\gamma \mid w) \cdot \Pr(f \mid w, X) \cdot \Pr(X \mid w)}{Z(X, \gamma)} \text{ by equation (5.13)} && (5.16)\\
&= \frac{\Pr(w, \gamma) \cdot \Pr(f, X \mid w)}{Z(X, \gamma)} && (5.17)\\
&= \frac{\Pr(w, \gamma) \cdot \Pr(f \mid w)}{Z(X, \gamma)} \quad \text{(provided } \mathsf{LHS}(f) = X\text{)} && (5.18)\\
&= \frac{\Pr(w \mid \gamma) \cdot \Pr(f \mid w)}{Z'(X, \gamma)} && (5.19)
\end{aligned}
$$

where $Z'(X, \gamma) = Z(X, \gamma)/\Pr(\gamma)$.

Giving the distributions mnemonic names as usual,

$$\Pr^{\text{exp}}(w, f \mid X, \gamma) = \frac{\Pr^{\text{revins}}(w \mid \gamma) \cdot \Pr^{\text{lex}}(f \mid w)}{Z'(X, \gamma)} \tag{5.20}$$

If the denominator Z' has been precomputed, this equation figures the full expansion probability with just one access to the transformationally smoothed lexicon $\Pr^{\text{lex}}$. The other term $\Pr^{\text{revins}}$ can be estimated by any of several methods (§5.4.3).

To find $Z'(X, \gamma)$ more explicitly, observe that the distribution must sum to 1 over all words w' and all frames f' such that $\mathsf{LHS}(f') = X$. So we may rewrite as

$$\Pr^{\text{exp}}(w, f \mid X, \gamma) = \frac{\Pr^{\text{revins}}(w \mid \gamma) \cdot \Pr^{\text{lex}}(f \mid w)}{\sum_{w'} \Pr^{\text{revins}}(w' \mid \gamma) \cdot \sum_{f' : \mathsf{LHS}(f') = X} \Pr^{\text{lex}}(f' \mid w')} \tag{5.21}$$

when $\mathsf{LHS}(f) = X$ (and of course $\Pr^{\text{exp}} = 0$ otherwise).

To understand the distribution defined by equation (5.21), note that it may be sampled via rejection sampling (just as in footnote 3 on p. 150). We are given X and γ.

1. Choose a headword w from the multinomial distribution $\Pr^{\text{revins}}(w \mid \gamma)$. This word will be appropriate to γ but perhaps not to X.

2. Choose a frame f projected by w, using the transformationally smoothed distribution $\Pr^{\text{lex}}(f \mid w)$. This requires a random walk on the lexical entries having headword w. Set f to be the frame of the entry where the walk halts.

3. If $\mathsf{LHS}(f) \neq X$ then discard both w and f and return to step 1 to try again.

4. Return (w, f).

This procedure captures the linguistic idea of the previous section: even if we have few observations of `S/NP` constituents, we should still be able to tell that intransitive `rise` is an unlikely head for one. If we attempt to choose `rise` as head, the rarity of extraction from intransitive frames means that `rise` will usually generate an `S` rather than `S/NP` frame, causing us to discard `rise` and try again. A transitive verb `devour` will manage to choose an `S/NP` frame much more often, so it will survive in this context at a higher rate.

The statistical idea behind equation (5.21) is that if $\Pr(w \mid X, \gamma)$ is difficult to estimate, then the transformational smoothing model can absorb the dependence of w on X. The reasonable Naive Bayes assumption about $\Pr(w \mid X, \gamma)$ (equation (5.13)) is what lets us separate out w's dependence on X from its dependence on γ.

A downside to equation (5.21) is that the denominator is complicated: the normalizing constants $Z'(X, \gamma)$ may be inconvenient to compute. On the other hand, even simpler uses of cross-category transformations (such as equation (5.12) used in the service of equation (5.8)) must do the trickiest part of that computation, namely the sum over f' of transformationally smoothed probabilities.

5.4.6 Summary

In short, we now have a way to define the probabilities $\Pr^{\text{exp}}$ that are multiplied to yield $\Pr^{\text{tree}}$. When it is possible to estimate $\Pr^{\text{ins}}$ accurately without backoff, then use $\Pr^{\text{ins}}$ and

$\mathrm{Pr}^{\mathrm{lex}}$ in equations (5.8) and (5.12). Otherwise, use $\mathrm{Pr}^{\mathrm{revins}}$ and $\mathrm{Pr}^{\mathrm{lex}}$ in equation (5.21).[4]

$\mathrm{Pr}^{\mathrm{ins}}$ and $\mathrm{Pr}^{\mathrm{revins}}$ may be estimated in any of several ways described in §5.4.3. $\mathrm{Pr}^{\mathrm{lex}}$ is transformationally smoothed.

5.5 Remarks on Linguistic Adequacy

Any statistical model of language makes some implicit claims about linguistics, in both its representations and its independence assumptions. A given model will be able to handle some linguistic phenomena easily, others by representational tricks, and others not at all.

This section discusses the motivation and use of the kind of flattened context-free approach given in §§2.4–3.7.2 and §§5.3.2–5.3.3. It includes a good deal of "armchair linguistics" for English. In particular, although the *experiments* reported in this thesis use relatively crude representations and transformations, we would like to hold out the hope that the *framework* is on the right track, in that it would remain appropriate—though perhaps not easily learnable—if one were to add more linguistic sophistication.

One could simply apply transformational smoothing to one's favorite lexicalized theory from linguistics, as in Chapter 2. (See especially §2.3.4.) But such theories sometimes take an *ad hoc* or *de minimis* approach to lexical transformations, and they are not always concerned with parsing efficiency, let alone with "soft" statistical issues such as word-to-word generalizations or the statistical independence of lexical entries from one another. It is therefore worth taking a few pages to develop our simple framework with respect to the statistical and linguistic concerns of this thesis.

5.5.1 What is a Word?

The model of trees in §§5.3.2–5.3.3 relies on a set Words. (The headwords of a syntax tree and its subtrees are drawn from this set; $\mathrm{Pr}^{\mathrm{lex}}$ specifies a distribution of frames for each $w \in \mathsf{Words}$; and $\mathrm{Pr}^{\mathrm{ins}}$ or $\mathrm{Pr}^{\mathrm{revins}}$ describes the contextual probabilities of each w.) But are these orthographic words, morphemes, word clusters, or word senses?

[4] Or for simplicity, just use equation (5.21) always.

In the experiments reported here, it was convenient to have Words consist of orthographic words. Below we will examine some possible extensions and their effects on $\Pr^{\text{ins}}$ (or $\Pr^{\text{revins}}$) and $\Pr^{\text{lex}}$.

5.5.1.1 Morphology

Taking Words as a set of unanalyzed orthographic words has the unfortunate consequence that *Popeye devours the spinach* and *Popeye has devoured the spinach* have different headwords. So far as the statistical model knows, these headwords are unrelated. The model must learn separately that both forms of `devour` are transitive ($\Pr^{\text{proj}}$, via $\Pr^{\text{lex}}$). It must also learn that they select, and are selected by, the same heads ($\Pr^{\text{ins}}$).

If external morphological knowledge is available, a simple solution to the would be to leave Words alone, but improve the models of $\Pr^{\text{ins}}(w \mid X, \gamma)$ to recognize the common root. For example, one might arrange that

$$\Pr^{\text{ins}}(w \mid X, \texttt{devours}) = \Pr^{\text{ins}}(w \mid X, \texttt{devoured})$$

and that

$$\Pr^{\text{ins}}(\texttt{devoured} \mid X, \gamma) = \Pr^{\text{ins}}(\texttt{devour}-? \mid \mathrm{X}, \gamma) \cdot \Pr^{\text{ins}}(?-\texttt{ed} \mid \mathrm{X})$$

However, that trick would improve only the insertion probabilities $\Pr^{\text{ins}}$, not the project probabilities $\Pr^{\text{lex}}$.

A more thoroughgoing solution is to assume that Words is a set of morphemes, and that the sentences presented to the learner have been preprocessed by a morphologizer (or human annotator) that splits orthographic words into their component morphemes. Then the correct analysis of the sentence *Popeye has devoured the spinach* would appear to the learner as a syntax tree with yield `Popeye have -Pres devour -PastPart the spinach`. We will discuss the form of this syntax tree further in §5.5.4.2 below.

Of course, a wholly unsupervised learner would have to learn the interword segmentation (de Marcken, 1996) and morphology (Goldsmith, 2001) along with the syntax.[5] In this

[5] A sensible approach (Eisner, 2000) is to regard the raw data as having been generated by the composition of a PCFG with a probabilistic finite-state transducer (PFST). The PCFG generates morpheme strings according to syntax, and the PFST post-processes them according to morphology. The learner must learn the parameters of both the PCFG and the PFST.

case, an EM-style learner might entertain multiple parses of an input sentence like "Popeye has devoured the spinach": parses with the fringe shown above, in which `devoured` is related to `devours`, as well as parses with differently analyzed or unanalyzed fringes such as `Popeye has devoured the spinach`. As with any statistically interdependent variables, the parameters governing morphology and syntax would bootstrap off each other during reestimation. If either morphology or syntax favors the correct analysis, the other benefits.

5.5.1.2 Semantic Clusters

One might wish the model to benefit from not only morphological but also semantic relationships among words. The selectional preferences of `devour` resemble not only those of `devoured` but also those of `eat`.

Thus, $\Pr^{\text{ins}}(w \mid \texttt{NP}, \texttt{devour}) \approx \Pr^{\text{ins}}(w \mid \texttt{NP}, \texttt{eat})$. Several researchers have used this observation to better predict w by aggregating the estimates from similar parent words such as `devour` and `eat` (Lee, 1997; Stetina and Nagao, 1997; Abney and Light, 1999; Li and Abe, 1998; Miyata et al., 1997; Utsuro et al., 1998). A variant also considers similar child words w (Rooth, 1995; Hofmann and Puzicha, 1998, §6.2). Some of these methods pick out the similar words in an entirely unsupervised way, while others use the hand-built WordNet hierarchy (Miller et al., 1990) as a source of constraint.

Clusters of words with similar selectional preferences can help estimate not only $\Pr^{\text{ins}}$ but also $\Pr^{\text{proj}}$. Rooth et al. (1999) have shown, by clustering techniques, that words (at least, verbs) with similar selectional preferences have similar meanings. Therefore the work of (Levin, 1993; Pinker, 1989; Grimshaw, 1994) suggests that they will tend to project the same frames: $\Pr^{\text{lex}}(f \mid \texttt{drink}) \approx \Pr^{\text{proj}}(f \mid \texttt{eat})$ (and hence $\Pr^{\text{proj}}(f \mid \texttt{NP}, \texttt{drink}) \approx \Pr^{\text{proj}}(f \mid \texttt{NP}, \texttt{eat})$).[6] Korhonen (2000) shows that Levin's insight holds up statistically even without word-sense disambiguation. There are many methods for clustering words by selectional preference (Webster and Marcus, 1989; Pereira et al., 1993; Dagan et al., 1997; Li and Abe, 1998; Rooth et al., 1999; Prescher et al., 2000), and Charniak (1997) actually achieved a small improvement in parsing accuracy by backing off to such semantic clusters of words in estimating both $\Pr^{\text{ins}}$ and $\Pr^{\text{proj}}$. Possibly such clusters would also aid

[6]The converse insight is also true: words that project the same frames tend to have the same meanings (Levin, 1993; Gleitman, 1990; Lapata and Brew, 1999; Schulte im Walde, 2000).

syntax learning or the semantic interpretation of parse trees.

To apply such techniques it is not necessary to change Words, only the procedures for estimating $\Pr^{\text{ins}}$ and $\Pr^{\text{lex}}$.

- Most obviously, one could back off just as Charniak does. For lexicon smoothing, this would set the backed-off estimate of $\Pr^{\text{lex}}(f \mid w)$ to a weighted average of values $\Pr^{\text{lex}}(f \mid w')$ over the words w' that are similar to w. That means estimating the individual distributions first, ignoring the influence that they will have upon one another once they are averaged.

- A more principled alternative is to augment the model with output features §3.6.2. Transformations yielding a lexical entry (w, f) should have not only a per-event feature specific to (w, f), but also a more general output feature sensitive to (C, f) for every word cluster C that contains w. This means that f tends to be listed or delisted for all words in the cluster together. In the case of soft clustering, the coefficient of this feature (§7.3.1.1) should depend on w's degree of membership in C, and might perhaps be learned (§7.3.2.2).

- Finally, one could capture the insight of similarity-based smoothing (Dagan et al., 1997) by allowing transformations that change the headword of a lexical entry.

5.5.1.3 Word Senses

Grouping words into clusters, whether by morphology or by semantics, has an opposite: splitting them into senses. One can certainly let Words be a set of senses, provided that the input to the learner is sense-disambiguated.[7] But this has less use in a transformational smoothing model than one might imagine.

Certainly there is no need to treat transitive and intransitive `eat` as different words. $\Pr^{\text{lex}}(f \mid \texttt{eat})$ specifies probabilities for both uses, and the whole point of transformational smoothing is to estimate these probabilities better by recognizing that they are related uses of the *same* word.

[7] Or that one is willing to treat the input senses as hidden variables and run EM to fit the parameters θ of the transformational smoothing model.

Even in cases where an orthographic word has clearly unrelated senses, it may not be particularly important to distinguish them for purposes of the model. Although one could certainly estimate separate distributions $\Pr^{\text{lex}}(f \mid \texttt{bank}_{\texttt{N1}})$, $\Pr^{\text{lex}}(f \mid \texttt{bank}_{\texttt{N2}})$, and $\Pr^{\text{lex}}(f \mid \texttt{bank}_{\texttt{V}})$ for the classical two noun senses and one verb sense of **bank**, there is little harm in simply estimating a single distribution $\Pr^{\text{lex}}(f \mid \texttt{bank})$ instead, which is a linear combination of the distributions for the three senses. Similarly, there is little harm in conflating the senses when estimating $\Pr^{\text{ins}}(w \mid \texttt{NP}, \texttt{bank})$ (which specifies words that can head NP arguments of bank—presumably $\texttt{bank}_{\texttt{V}}$) or $\Pr^{\text{ins}}(\texttt{bank} \mid \texttt{NP}, \gamma)$ (which specifies words whose NP arguments can be headed by bank—presumably $\texttt{bank}_{\texttt{N1}}$ or $\texttt{bank}_{\texttt{N2}}$).

There are, however, a few advantages to "doing it right" and distinguishing senses, particularly senses with the same part of speech:[8]

- When using the model $\Pr^{\text{tree}}$ to parse, it helps disambiguation if the input words have sharp selectional preferences. Rolling unrelated senses together makes the correct parse less certain.

- The transformational smoothing model learns transformations by looking at pairs of frames that tend to appear with the same words. If senses are conflated, then more frames appear with each word; the resulting spurious correlations can mislead the model.

- Under our transformational prior, the prior probability of a particular distribution $P = \Pr^{\text{lex}}(f \mid \texttt{bank})$ is not quite the same as the prior probability of obtaining separate distributions $\Pr^{\text{lex}}(f \mid \texttt{bank}_{\texttt{N1}})$, $\Pr^{\text{lex}}(f \mid \texttt{bank}_{\texttt{N2}})$, and $\Pr^{\text{lex}}(f \mid \texttt{bank}_{\texttt{V}})$ whose average (weighted by the senses' relative probabilities) is P. So distinguishing senses has some effect on the estimation of $\Pr^{\text{lex}}$.

- Under the independence assumptions of §5.3.3, ?*insure the grassy bank* appears to have high probability because (river) banks are likely to be grassy, and (financial) banks are likely to be insured. A model that distinguishes the two senses will not

[8]If this were not true, one would not even have to distinguish orthographic words. It would suffice to conflate them all into a single word um with 100,000 senses. This would certainly aid estimation of the transformational smoothing model, but it is not a good idea.

make this mistake: ?*insure the grassy bank*$_{\mathtt{N1}}$ is unlikely. In general, the sense of the word can mediate statistical correlations among the word's dependents and governor.

- Similarly, ?*kern the letter about your aunt* incorrectly appears to have high probability, since (typographical) letters can be kerned and (mailed) letters can be about something. In this case, a sense distinction is necessary to mediate a statistical correction between a word's frame and its governor.

- The Naive Bayes assumption used for $\Pr^{\text{ins}}(w \mid X, \gamma)$ in §5.4.3 is less plausible if senses are conflated. Naive Bayes implies that if w can occur with X and γ separately, then it can occur with them when they appear together. This is false if only one sense of w can occur with X and only another sense can occur with γ.[9] Since *a letter about ...* is a "picture-NP" that permits extraction, the Naive Bayes assumption would overestimate $\Pr^{\text{ins}}(\mathtt{letter} \mid \mathtt{NP/NP}, \mathtt{kerned})$, allowing ?*the subject you kerned [a letter about]*$_{\mathtt{NP/NP}}$.

On the other hand, it is also possible to make overly fine sense distinctions. If a sense $w \in \mathsf{Words}$ is so specific that it appears with only a single frame, then it gives the transformational smoothing model no evidence for transformations among frames. A remedy is to use semantic clusters of senses as described above (§5.5.1.2). In essence, this backs off to the orthographic word, by presuming that its senses will have similar if not identical distributions of frames.

5.5.2 The Use of Flat Frames

A distinctive and important point about the representation of §5.3.2 is that frames are flat. In the stochastic generation of a tree (§5.3.3), a frame such as !!! is generated all at once, as a single sample from $\Pr^{\text{lex}}$.

As a result, in the generated tree shown in Fig. 5.1, all arguments and adjuncts of `fills` attach at the same level. There is only a single subtree headed by `fills`. This contrasts with most of the other models currently used in the field for English parsing (exceptions

[9]However, if γ specifies not only a lexical governor but also a theta-role for w, as suggested in §5.5.4.5, then it is hard to find examples of this. The example given in the text works but is rather strained.

include the dependency parser of (Eisner, 1996c; Eisner, 1996b; Eisner, 1997; Eisner, 2000)). In those models, `fills` would project `S` and `VP` levels at separate, statistically independent steps of the generation process.

The use of flat frames was motivated at length in §2.4.1, and discussed further in §2.4.3 and §6.7.1.

5.5.3 Long-Distance Movement

As discussed in §2.2 and §2.4.3, long-distance movement can be handled in the style of GPSG (Gazdar et al., 1985) or categorial grammars (e.g., (Steedman, 1996)). The idea is to use structured "slashed" nonterminals to pass gaps up the tree. Such devices can increase the power of context-free grammar if they are allowed to introduce infinitely many nonterminals. Collins (1997) was the first to use this scheme in a statistical parser.

A problem with modeling long-distance movement is that semantic dependencies become non-local. The model of insertion probabilities in §5.4.3 tries to capture selectional preferences, but cannot do so in non-local cases. In particular, it cannot use the subject and object selectional preferences of `eat` to help disambiguate *the hungry orphan whom Bill wants to eat.*

One solution would be to require the stochastic generation process of §5.3.3 to expand gapped nonterminals before their fillers; then the headword of the filler could be conditioned on the lexical governor of the gap, which was previously generated. Another solution would be to modify §5.3.3 to allow not only expansion but also adjunction at nonterminal nodes. (Adjunction is used here in the sense of TAG (Joshi et al., 1975); it could be constrained to non-wrapping adjunction in order to preserve context-freeness (Schabes and Shieber, 1994).)

5.5.4 Capturing Bilexical Dependencies with $\mathbf{Pr}^{\mathrm{ins}}(w \mid X, \gamma)$

The generation model of §5.4.1 assumes that every word w is chosen from a distribution $\Pr^{\mathrm{ins}}(w \mid X, \gamma)$. So it depends only on X and γ, where γ is some previously generated context. Usually γ is assumed to be a *single* previously generated word. Indeed, γ is traditionally chosen to be w's parent in the tree (i.e., the tree headed by w is a child of the tree headed by γ). Is there a style of annotation for which this choice is linguistically apt?

5.5.4.1 Choosing Lexical Headwords

The first question is whether a phrase's designated headword should be the morpheme that contributes the phrase's *type* or its *core meaning*. Is *the tulips* headed by `the` (the DP hypothesis), `tulip`, or `-s`? Is *The sailor has devoured the spinach* headed by *has*'s present tense morpheme `-s`, or the main verb `devour`?

For the experiments in this thesis, the data are consistently annotated (§6.2) with *lexical* heads: `tulip`, `devour`.[10] The phrase's type is already reflected in its frame, so we are free to use the headword to reflect the phrase's meaning. This breaks with GB-style linguistics, in which a phrase with obligatory features (e.g., `IP`) is headed by those features (e.g., `I`). It also breaks to some extent with the parsers of Collins and Charniak, which take a verb phrase to be headed by the tensed verb even if it is an auxiliary.

The decision to use lexical headwords is motivated by their role in selectional preferences. For example, in *The sailor has devoured the spinach*, the important dependency—the one with high pointwise mutual information—is between `sailor` and `devour`. It is not between `the` and `has`, or `the` and -s. Thus, we would like to choose the subject of *has devoured the spinach* from the distribution $\Pr^{\text{ins}}(w \mid \texttt{NP+3s}, \texttt{devour})$. Moreover, to capture `devour`'s preference for animate, fierce subjects, we would like to choose the subject headword w that determines the animacy of the subject: a noun, not a determiner.

5.5.4.2 Function Morphemes Remain Influential as Nonterminal Features

We cannot ignore function "heads," however. Just as `devour` requires an animate subject, the verbal affix `-s` requires a third-person singular ("3s") subject. The subject of *has devoured the spinach* should be both animate and 3s. Does this mean that `-s` has an equal claim to be the headword?

To get the best of both worlds, the subject should depend on both `devour` and `-s`. Fortunately, in $\Pr^{\text{ins}}(w \mid X, \gamma)$—with or without the Naive Bayes assumption of §5.4.3—w

[10] Or actually, `tulips` and `devoured`, since there is no morphological preprocessing at present.

is chosen not only given $\gamma =$ `devour` but also given X, the category that w is to head. Above we supposed $X =$ `NP+3s`, which does the trick. The `+3s` feature of X stands proxy for the `-s` morpheme that has been suppressed from γ. This allows us to retain the formal notion that w depends on only one word γ.

In short, the "right" model of English is for `devour` to project a frame such as

S → NP+3s have -s ——-ed NP

5.5.4.3 Gap-Filler Conservation for Features

As the frame just mentioned is complex, we would like to consider how it can arise via regular processes. (Again, the experiments reported in Chapter 6 ignore morphology, so do not attempt anything so ambitious.)

Using structured nonterminals to handle agreement, we might write the above frame as something like

S → NP+3s have +Tense/3s ——-ed NP

Improving this further, the formalism of §5.3.2 suggests that rather than include a terminal symbol such as `have` directly in the frame, we ought to use a nonterminal that rewrites as `have`. Using further features for this, and encoding their semantic effect on the category of the phrase, we get

S+Perfect+Tense → NP+3s +Perfect/PastPart +Tense/3s —— +PastPart NP

The morphemes `have`, `-s`, and `-ed` will then be inserted as the respective headwords of `+Perfect/PastPart`, `+Tense/3s`, and `+PastPart`.[11]

[11] This postpones morphological fusion to a later step. A notational alternative—in the lexicalist spirit of precompiling all the hard work into the lexicon (§2.2)—is for the transformations that introduce the morphemes to carry out the morphology directly on the lexical entry, in the style of word-based morphology ((Aronoff, 1976; Anderson, 1992; Sehitoglu and Bozsahin, 1999)). Instead of changing `NP devour NP` ⇒ `NP +Perfect/PastPart devour +PastPart NP` as below, a transformation would change it ⇒ `NP have devoured NP`. (The headword would still be considered `devour`, for purposes of lexical insertion.) The same transformation would have to know about irregular morphology: `NP sing NP` ⇒ `NP have sung NP`. The resulting lexical entries permit a morphology-free account of generation and parsing. Aesthetically, although these entries obscure the filler-gap conservation that is supposed to license the transformation, they do have the advantage that affixes are never ordered with respect to their hosts. The transformation simply inserts the auxiliary and modifies the adjacent word. There is no need to explain why the modificational affix characteristically "hops" to the right edge of `devour` (or to the middle of `sing`).

The "+ and /" notation (adapted from Stabler's (1997) formalization of minimalism[12]) is intended to suggest that the feature `+3s` satisfies the requirement `/3s`, just as a constituent `NP` can fill a gap `/NP`. The pairing of fillers and gaps, which ensures that semantic roles not filled locally will be filled at a distance, is a central theme of categorial grammars ((Lambek, 1958; Steedman, 1990; Morrill, 1994)) and link grammars ((Sleator and Temperley, 1991)). Similarly, the pairing of requirements and features is a central theme of minimalist syntax ((Chomsky, 1995; Stabler, 1997; Epstein and Hornstein, 1999)); although it is not quite as symmetric, since every / requirement must be "checked off" by some + feature, but an + feature may check off zero or two / features in the course of a minimalist derivation.

While minimalism generates and represents sentences very differently from what we have assumed, embedding its ideas about features in our flattened CFG-like framework is straightforward. The + features of a phrase are specified in its lexical entry (on the LHS), and may or may not be introduced by transformation.

Recall that gap-filler transformations in Fig. 2.5 were careful to preserve the invariant that gaps must be resolved locally or passed up. Featural transformations are similar, modulo the asymmetry:

- A transformation may introduce a new LHS feature `+x`—provided it is not already present on the LHS—together with some RHS morpheme bearing `+x`. (This is analogous to ProduceGap in Fig. 2.5. There is no analogy to PassGap, which would propagate features up the syntax tree, because there is no need to propagate beyond the maximal projection already represented by a flat frame.)

- A transformation may introduce a new RHS requirement `/x` together with a requirement that some other nonterminal on the RHS have a satisfying feature `+x`. This may involve actually inserting such a nonterminal.

So in the final analysis, the frame above and its LHS features could be derived by transformation from *untensed* frames as follows:

[12] Stabler uses + and -. However, we prefer to reserve the notation `S-Tense` for another use, so that `NP hope S+Tense` and `NP want S-Tense` can be read as subcategorizing for sentences that are required to be tensed and untensed, respectively.

$\texttt{S} \rightarrow \texttt{NP}$ —— $\texttt{NP}$

Popeye devour the spinach

$\Rightarrow$ $\texttt{S+Perfect} \rightarrow \texttt{NP +Perfect/PastPart}$ —— $\texttt{+PastPart NP}$

Popeye have devoured the spinach

$\Rightarrow$ $\texttt{S+Perfect+Tense} \rightarrow \texttt{NP+3s +Perfect/PastPart +Tense/3s}$ —— $\texttt{+PastPart NP}$

Popeye has devoured the spinach

The transformation that inserts the tense-and-agreement morpheme `+Tense/3s` (e.g., `-s`) requires the presence of an agreeing subject. The `+3s` feature it introduces on the subject `NP` serves as a restriction on how that `NP` can subsequently expand: the putative lexical entry $\texttt{NP+3s} \rightarrow \texttt{I}$ has very low probability (except in Popeye's own dialect), so **I has devoured the spinach* is unlikely.

The above remarks about featural gaps and fillers are meant particularly to defend the naturalness of transformations that insert auxiliary verbs into a core SVX structure (where X represents any complements). They become more natural still if lexical entries have internal structure (Fig. 2.2c). In this case, these insertions can be described by transformational mechanisms that modify this internal structure—the "adjunction-as-transformation" approach of §2.4.1. The inspiration comes from Lexicalized Tree-Adjoining Grammar (LTAG), which also begins with a core SVX lexical entry and inserts auxiliaries. The motivation is the same: since S and V are in the same structure to begin with, V can impose selectional preferences on S, even though they will later be separated by auxiliaries. LTAG aficionados call this an "extended domain of locality." LTAG also provides a natural mechanism (wrapping adjunction) for inserting matched gaps and fillers on either side of a nonterminal, as in the Popeye example above.

5.5.4.4 Multiple Lexical Influences

There are various constructions in which a headword w arguably depends on *two* words from the context, which in turn may depend upon one another. Such cases have generally been ignored by statistical models of syntax, excepting statistical approaches to link grammar (Lafferty et al., 1992) and unification grammar Johnson et al. (1999).[13]

A Naive Bayes model (§5.4.3) can be used to capture the dependency on both words, just as it was used above to capture the dependency on both a word and a function morpheme. For example, in

$$\textit{Bluto convinced } \texttt{NP} \textit{ [to explode]}_{\texttt{Sinf\textbackslash NP}},$$

Lafferty et al. (1992) would condition the generation of the NP's headword jointly on `convinced` and `explode`. The resulting sparse-data problem could be handled by backing off to a Naive Bayes approximation.

It should be noted, however, that there is actually no good choice to head this NP. Such an NP would have to be at once a good object of `convince` (hence animate) and a good subject of `explode` (hence an explosive). For this reason, the entire template above should have been unlikely in the first place: that is, `explode` with an unaccusative frame is simply unlikely to be the complement of `convince`.[14] The model above fails to capture this fact; it will generate the template too often (and then be stuck with a choice among bad alternatives for NP).

A nice solution is the kind of whole-parse maximum-entropy model that has been proposed by Johnson et al. (1999) (although normalizing such models is difficult). Here the parse probability is separately affected by the presence of the three pairwise ("bilexical") relationships $\texttt{convince} \xleftarrow{comp} \texttt{explode}$, $\texttt{convince} \xleftarrow{patient} \text{head}(\texttt{NP})$, $\text{head}(\texttt{NP}) \xrightarrow{patient} \texttt{explode}$. (For any choice of head(NP), at least one of these relationships is unlikely and knocks down the parse probability, which captures the intuition described above that all such parses are bad.) As noted in §2.3.4, one might attempt to apply transformational smoothing to the weights of such a model.

[13] A link grammar would assign a cyclic dependency structure to the example below, while a unification grammar would allow the same NP to appear as an argument of both `convinced` and `explode`.

[14] However, `explode` with a transitive frame is all right:

$$\textit{Bluto convinced } \texttt{NP} \textit{ [to explode the bomb]}_{\texttt{Sinf\textbackslash NP}}$$

5.5.4.5 Theta Roles

Lexical insertion selects a headword to fill devour's "NP" slot, according to the distribution $\Pr^{\text{ins}}(\cdot \mid \texttt{devour}, \texttt{NP})$. A subtlety is that devour really has two NP slots, which should be distinguished because they are filled by different kinds of arguments—predators and prey. These arguments fill different **thematic roles** or **theta-roles** of the verb (Fillmore, 1968).

Fig. 2.6 on p. 63 illustrated the obvious way of distinguishing these slots.[15] There are three consequences for the model of insertion probabilities, $\Pr^{\text{ins}}$:

no generalization within a frame The distribution $\Pr^{\text{ins}}(\cdot \mid \texttt{devour}, \texttt{NP}_{agent})$ will be different from the superficially similar $\Pr^{\text{ins}}(\cdot \mid \texttt{devour}, \texttt{NP}_{patient})$. One selects for predators, the other for prey.

generalization across related frames As Rooth et al. (1999, p. 5) point out, $\Pr^{\text{ins}}(\cdot \mid \texttt{sink}, \texttt{NP}_{patient})$ is the same distribution whether it is used to select a head for the object of transitive $\texttt{S} \rightarrow \texttt{NP}_{\texttt{agent}}\ \texttt{sink}\ \texttt{NP}_{\texttt{patient}}$ or the subject of intransitive $\texttt{S} \rightarrow \texttt{NP}_{\texttt{patient}}\ \texttt{sink}$. This makes it possible

to generalize selectional preferences from the first (listed) entry to the second (derived) one.

generalization across headwords Insertion probabilities may need to be smoothed as in §5.4.3. Suppose theta-roles are considered to be comparable across different heads: that is, $\texttt{NP}_{\texttt{agent}}$ appears in the frames of eat as well as the frames of devour. Then one can back off from $\Pr^{\text{ins}}(\texttt{lion} \mid \texttt{devour}, \texttt{NP}_{\texttt{agent}})$ to $\Pr^{\text{ins}}(\texttt{lion} \mid \texttt{eat}, \texttt{NP}_{\texttt{agent}})$.[16]

In the limit—for example, the Naive Bayes model of §5.4.3—one would back off to $\Pr^{\text{ins}}(\texttt{lion} \mid \texttt{NP}_{agent})$, which asks whether lion is a good $\texttt{NP}_{agent}$ in some generic sense. This question is not nonsense. Any transitive verb imposes various "proto-agent" properties (animacy, volition, motion) to a greater degree on its agent than

[15] An alternative scheme (Charniak, 1997; Johnson, 1999) relies on the fact that subject and object NP's have different parent nonterminals (S and VP respectively). This scheme simply conditions the insertion probability on this parent nonterminal as well. However, this scheme does not work for our flattened frames, nor for frames in which the agent is not the subject.

[16] If theta-roles are not comparable, one can still back off to the less precise $\Pr^{\text{ins}}(\texttt{lion} \mid \texttt{eat}, \texttt{NP})$, as computed over all NP arguments of eat.

on its patient (Dowty, 1991)—this is what is *meant* by agent and patient—and `lion` (as opposed to `carcass`) is quite easy to reconcile with those properties.[17]

A learner that does not observe the subscripted theta-roles might be able to use the observed insertion probabilities to infer them (§2.4.3; Rooth et al. (1999)). Suppose, for instance, that the learner observes everything *but* the theta-roles: it sees a sample of trees from which the theta-roles have been removed. As usual, the learner tries to set hidden variables—theta-roles and transformations—to maximize the probability of these trees under $\Pr^{\text{tree}}$ (equation (5.2)).

As far as a learner is concerned, the correct tree for *The boat sank* might use either $\mathtt{S} \to \mathtt{NP}_{\mathtt{patient}}\ \mathtt{sink}$ or $\mathtt{S} \to \mathtt{NP}_{\mathtt{agent}}\ \mathtt{sink}$. Both entries are plausible: they can be derived from $\mathtt{S} \to \mathtt{NP}_{\mathtt{agent}}\ \mathtt{sink}\ \mathtt{NP}_{\mathtt{patient}}$ by transformations that are common in English (and which Bresnan (1982b) calls Activo-Passivization and Intransitivization, respectively).

However, the subject headword `boat` is likely only in a tree that uses the first entry. We know from observations of transitive `sink` that $\Pr^{\text{ins}}(\mathtt{boat} \mid \mathtt{sink}, \mathtt{NP}_{patient}) \gg \Pr^{\text{ins}}(\mathtt{boat} \mid \mathtt{sink}, \mathtt{NP}_{agent})$. More generally, we know from observations of other transitive verbs that $\Pr^{\text{ins}}(\mathtt{boat} \mid \mathtt{NP}_{patient}) \gg \Pr^{\text{ins}}(\mathtt{boat} \mid \mathtt{NP}_{agent})$. So the first entry produces a more likely tree for *The boat sank*. If the learner observes several such trees, it will choose to list the first entry but not the second in the lexicon.[18] Rooth et al. (1999) can do this.

[17] Dowty further points out that a verb with multiple arguments tends to realize the most agent-like argument in subject position. In English, where subjects are sentence-initial, this amounts to a bias toward the frame $\mathtt{S} \to \mathtt{NP}_{\mathtt{agent}}\ \text{——}\ \mathtt{NP}_{\mathtt{patient}}$. A transformation model can capture such a bias by learning the weight of an output feature (§3.6.2) that picks out entries with this frame.

[18] The two entries are to some extent in competition, as they are derived by competing transformations on $\mathtt{S} \to \mathtt{NP}_{\mathtt{agent}}\ \mathtt{sink}\ \mathtt{NP}_{\mathtt{patient}}$. Raising the probability of one such transformation reduces the probability of the others. However, one might suppose that the competition between these two entries is especially fierce because of functional pressure for *He sank* to have only one meaning. In other words, one might expect a prior bias against giving two lexical entries large probabilities in the same language if they are identical except for theta-role. While transformation models are poor at *learning* such anti-correlations from data (§3.7.1), this one should be universal across languages and so would be appropriate to hard-code into the model. For example, the arcs leading to a frame like $\mathtt{S} \to \mathtt{NP}_{\mathtt{patient}}\ \mathtt{sink}$ might be augmented with *negative* copies (§7.3.1.1) of some of the features on the arcs leading to the twin frame $\mathtt{S} \to \mathtt{NP}_{\mathtt{agent}}\ \mathtt{sink}$ (notably the per-event feature), and vice-versa. Then learning weights that list or derive one entry will automatically delist or block derivation of the other.

5.5.5 The Frame Independence Assumptions

Recall our lexicalized PCFG model for randomly generating trees (§5.3.3). It assumes that each frame is chosen freely and independently of the others, based only on its previously generated headword and LHS. This assumption is relaxed by some researchers.

5.5.5.1 Projection and Parent Nonterminals

In particular, Charniak (1997) conditions also on the LHS of the parent frame. Given the style of tree he uses (Penn Treebank), this allows an NP to expand differently according to whether its parent category is S, VP, or PP—that is, whether it is a subject, direct object, or object of a preposition. Similarly, an S may expand differently according to whether it is a matrix sentence and, if not, whether it has a complementizer.

Johnson (1999) shows that this trick improves the parsing performance of a non-lexicalized PCFG. To incorporate it into our model, we could follow Johnson and expand the set of nonterminals, annotating them with context. Thus, a token of $\mathtt{S} \to \mathtt{NP}$ —— $\mathtt{NP}$ $\mathtt{PP}$ that appears inside an SBAR would become $\mathtt{S}^{\mathtt{SBAR}} \to \mathtt{NP}^{\mathtt{S}-subject}$ —— $\mathtt{NP}^{\mathtt{S}-object}$ $\mathtt{PP}^{\mathtt{S}}$. The superscript on each nonterminal indicates the category of its parent's frame, and perhaps the position (*subject* or *object*) it occupies within that frame. The tranformations must be rejiggered to respect these superscript conventions; note that the RHS superscripts are fully predictable from the rest of the frame.

This approach also affects the lexical insertion probabilities, since they can also be made sensitive to the nonterminal superscripts. The insertion probability $\Pr^{\text{ins}}(\cdot \mid w, \mathtt{PP})$ becomes $\Pr^{\text{ins}}(\cdot \mid w, \mathtt{PP}^{\mathtt{S}})$. This recognizes that PP's that modify S might have different headwords than PP's that modify noun phrases.[19]

5.5.5.2 Projection and Other Contextual Features

More generally, one can make the projection probabilities sensitive to some additional tree context γ by abandoning the frame independence assumption of equation (5.10): thus,

[19] Charniak's model does this, as already mentioned in footnote 15. The superscripts are partly redundant with the theta-role subscripts of §5.5.4.5. The difference is that the superscripts encode syntactic role (*subject*) while the subscripts encode semantic role (*agent*). One would expect $\Pr^{\text{proj}}$ to be sensitive to the former and $\Pr^{\text{ins}}$ to the latter, when available. If nonterminals are annotated with both, this can be arranged by suitable independence assumptions.

$\Pr^{\text{proj}}(f \mid w, X, \gamma)$. This means that $\Pr^{\text{lex}}$ must be estimated over triples (w, f, γ).[20]

In the case described above, γ is the nonterminal superscript on $\mathsf{LHS}(f)$, so the transformations effectively relate frames of the form $\mathtt{S}^{\mathtt{SBAR}} \rightarrow \mathtt{NP}$ —— $\mathtt{NP}$ $\mathtt{PP}$. This merely drops the predictable RHS superscripts from the frames in the previous approach.

The main advantage to keeping those predictable superscripts, as Johnson does, is that the result is still a PCFG—with an expanded nonterminal set—and therefore allows the use of standard parsing algorithms. For example, a bottom-up chart parser will build not just an NP but an $\mathtt{NP}^{\mathtt{S}-subject}$, with a probability that reflects the typical form of subject NP's (for example, the fact that they are less likely to take sentential complements).

5.5.6 Semantics and World Knowledge

A final point is worth emphasizing. No matter how much syntax it takes into account, $\Pr^{\text{tree}}(\cdot)$ is ultimately an incomplete model of the trees that arise in a language. For example, it does not reflect the tendency of speakers to say things that are topical, sensible, interesting, easy to parse, semantically interpretable, or true.

The model does contain weak proxies for these phenomena. It knows how often people talk about devouring. It knows that when they do, they usually mention who or what gets devoured. And it knows what kinds of things do get devoured—at least up to their headwords.

These proxies suffice for the task attempted by most current parsing work: parsing reasonably unambiguous input (written text) using a well-trained grammar. They would do less well at disambiguating noisy speech data, resolving ambiguities of scope or reference, or reestimating a poor grammar in the course of language learning. For those tasks, semantics and world knowledge become increasingly important. Young children who are still learning the language appear to rely on such factors (Bever, 1970; Donaldson, 1978), as do adults processing speech in real time (Tanenhaus et al., 1995).

Perhaps the most obvious deficiency of the lexicalized-grammar definition of $\Pr^{\text{tree}}$ is

[20] Or, rather than designing a transformation model over these triples, one can back off to a Naive Bayes model of the form

$$\Pr^{\text{proj}}(f \mid w, X, \gamma) = \frac{\Pr(\gamma \mid f) \cdot \Pr^{\text{lex}}(f \mid w)}{\sum_{f' : \mathsf{LHS}(f') = X} \Pr(\gamma \mid f') \cdot \Pr^{\text{lex}}(f' \mid w)}$$

that it identifies entities or events with the headwords of the phrases used to describe them. A (far) more sophisticated generative model than §5.3.3's would generate phrases that refer felicitously to entities in the surrounding discourse and their plausible behaviors. In parsing, such a model would resolve these phrases' referents, and recognize from context that `it` or `the lion` is not just any `it` or `lion`, but is a particular tender-hearted, toothachey lion who would (almost) certainly never devour anybody.

Chapter 6

Experimental Evaluation

This chapter evaluates a number of models for estimating frame projection probabilities over a large or infinite set of frames. A frame projection probability (§5.4.2) is the probability of a lexical entry given its headword and LHS. Here we write it as

$$\Pr(\text{RHS} \mid \text{headword}, \text{LHS}) \tag{6.1}$$

For example, in the lexical entry `S` $\rightarrow$ `NP devoured NP PP`, this is

$$\Pr(\texttt{NP} \;\text{——}\; \texttt{NP PP} \mid \texttt{devoured}, \texttt{S}) \tag{6.2}$$

As explained in §5.4.1 with a different notation, such probabilities are useful in parsing. A parser requesting probability (6.2) wants to know: Assuming that the rest of the parse tree calls for an `S` headed by `devoured`, would `NP` —— `NP PP` be a plausible internal structure for that `S`?

We call estimating such probabilities the **frame prediction problem**. This chapter is almost the first to isolate this problem and evaluate solutions to it directly, using cross-entropy, rather than as part of a larger parser. Carroll and Rooth (1998) also did so; however, they spent their effort on creating a large training set (by developing a parser and parsing raw data), rather than on generalizing from the training set or otherwise smoothing it. Also, their test set used only 3 words and much more impoverished frames. Here we will compare smoothing methods using fixed training and test sets extracted from the Penn Treebank.

As hoped, a quite simple edit-distance transformation model had the best performance, even without exploring the additional parameter optimization techniques to be presented in Chapter 8. It achieved a 20% perplexity reduction on test data over the best model replicated from previous literature (namely (Eisner, 1996b)). Improving the models from the literature also yielded performance gains, but somewhat less than the transformation model's.

6.1 The Evaluation Task

6.1.1 Conditional Cross-Entropy (RHS Perplexity)

Following (Carroll and Rooth, 1998), the evaluation measure used to compare various frame-prediction models is conditional cross-entropy. That is, the test data is a collection of lexical entries, and we hope to achieve a small value of

$$\operatorname*{mean}_i \left(-\log_2 \Pr_\theta(\text{RHS of test entry } i \mid \text{headword of test entry } i, \text{LHS of test entry } i)\right) \tag{6.3}$$

Following the convention in language modeling for speech recognition, we actually report the perplexity of the RHS, defined as 2 to this conditional cross-entropy.

The justification comes from the previous chapter. If the dataset consists of all the lexical entries in a corpus of parse trees, then equation (5.9) defined the joint probability of the corpus. One might hope to set the transformation model's parameters to make this large (even though that is not quite a task-specific evaluation: see §6.1.2). If the first factor of equation (5.9) is to be estimated without reference to these parameters, then one can maximize equation (5.9) by minimizing equation (6.3).[1]

In the case of a transformation model, the conditional probabilities required by equation (6.3) are not the unconditioned probabilities Pr(headword, LHS, RHS) that would be computed by equation (4.7). However, computing the conditionalized versions requires only small adaptations to the algorithms of Chapter 4.[2]

[1]Granted the independence assumption (5.10). If one drops that assumption, or uses the transformation model as described in §5.4.3 to help estimate the insertion probabilities in the first factor of equation (5.9), then one might prefer to evaluate directly on equation (5.9). However, this complicates the evaluation and we do not pursue it here.

[2]In particular, working with conditional probabilities throughout is trivial for the transformation model

One could still fit a reasonable model by maximizing the unconditionalized objective function (4.8). Since it is possible, however, it is better to use the conditionalized version that corresponds to equation (6.3), that is,

$$\log_2 \Pr_{\text{prior}}(\theta) + \sum_i \log_2 \Pr_\theta(\text{RHS of training entry } i \mid \text{headword of training entry } i, \text{LHS of training entry } i) \quad (6.4)$$

This is exactly (the logarithm of) the conditional Bayesian paradigm described in §3.1.5.

6.1.2 Why Perplexity?

Perplexity is not the only reasonable evaluation measure. As noted in the chapter introduction, one could also evaluate a frame prediction model by embedding it in a statistical parser and measuring parse accuracy.[3] This is analogous to evaluating a language model by embedding it in a speech recognizer and measuring recognition accuracy (word error rate).

Perplexity directly measures a system's inability to make quantitative predictions of isolated data from the test set. However, parsing and speech recognition do not consider words or frames in isolation—they perform a global optimization in which the input data and the easy predictions can constrain the hard predictions—and at the end of the day they are evaluated only on the *qualitative* classifications that result from maximizing a predicted probability. As a result they can sometimes perform well even if they use high-perplexity models. The parser of (Charniak, 1997) was state-of-the-art even though its frame-prediction module assigned probability 0 to frames that were not observed in training data, yielding *infinite* perplexity.

The hope is nonetheless that reducing perplexity will eventually improve task performance. This is the working assumption of the speech community, even though small perplexity improvements in language modelling do not always improve word error rate.

evaluated in this chapter. The model lacks any transformations that would change the headword or LHS of a lexical entry. Indeed, any random walk commits to a headword and LHS on the first two arcs from START. The probability of the rest of the random walk generates RHS given the headword and LHS. It is therefore straightforward to evaluate and optimize just the probability $\Pr(\text{RHS} \mid \text{headword}, \text{LHS})$.

[3] Or as an approximation, one might use a weighted version of perplexity, where test data are weighted by some loss function that estimates how important it is to get them right.

There is some evidence for this hope in the parsing world as well. In the terms of §6.6.1 below, Collins (1997) improved parse accuracy by switching from a unigram model to an subcategorization model, and Charniak (2000) improved parse accuracy by switching from the aforementioned infinite-perplexity model to a bigram model. We will see in §6.7.1 that the former change reduced perplexity, and obviously the latter change did as well.

It is certainly true that frame prediction at a certain level is necessary in order to parse correctly. §6.3 below will note the high level of novel frames needed for the correct parses of test data. Briscoe and Carroll (1993) found that half the parse failures in their non-statistical, lexicalized wide-coverage parser were caused by inaccurate subcategorization information in the lexicon.[4]

Just as in language modelling, there are three reasons to focus on perplexity when trying to improve frame prediction. First, by isolating this problem from other parsing issues, it makes it easier for different researchers to compare results. Second, isolating the problem makes the choice of model less specialized to a particular task (e.g., parsing newspaper text vs. parsing speech, generating text, or learning syntax (§1.2.4), which was the original motivation of this work). Finally, as a continuous rather than a discrete measure, it is more sensitive than parse accuracy, and is therefore a more helpful indicator of whether a change to a model is promising.

6.2 Preparation of the Experimental Data

The experimental data, illustrated in Fig. 6.3, were derived from the Penn Treebank II (Marcus et al., 1993; Bies et al., 1995). The Treebank is a representative collection of hand-parsed sentences drawn from *The Wall Street Journal*, an English-language newspaper that focuses on business and politics.

Some relevant features of the Treebank:

- The terminal symbols are generally inflected forms; the Treebank project performed very little morphological analysis prior to hand-parsing.

- Punctuation symbols are treated like other terminals.

[4] As summarized by Briscoe and Carroll (1997).

```
( (S
    (NP-SBJ (JJ Big) (NN indexer) (NNP Bankers) (NNP Trust) (NNP Co.) )
    (ADVP (RB also) )
    (VP (VBZ uses)
      (NP (NNS futures) )
      (PP-LOC (IN in)
        (NP
          (NP (DT a) (NN strategy) )
          (SBAR
            (WHNP-1 (IN that) )
            (S
              (PP (IN on)
                (NP (NN average) ))
              (NP-SBJ (-NONE- *T*-1) )
              (VP (VBZ has)
                (VP (VBN added)
                  (NP (CD one) (NN percentage) (NN point) )
                  (PP-CLR (TO to)
                    (NP
                      (NP (PRP$ its) (JJ enhanced) (NN fund) (POS 's) )
                      (NNS returns) )))))))))
    (. .) ))
```

Figure 6.1: A “raw” sentence from the Penn Treebank II.

```
( (S
    (NP (JJ big) indexer (NPR (NNP Bankers) (NNP Trust) Co.) )
    (ADVP also)
    uses
    (NP futures)
    (PP-LOC in
      (NP
        (NP (DT a)
            strategy
            (SBAR that
              (S
                (PP on (NP average) )
                (VBZ has)
                added
                (NP (CD one) (NN percentage) point)
                (PP to
                  (NP
                    (NP$ (NP (PRP$ its) (JJ enhanced) fund) 's)
                    returns) ))))))
    (. .) ))
```

Figure 6.2: The sentence of Fig. 6.1 just after step 8 of data preparation. Notice the discovery of subconstituents `Bankers Trust Co.` (a restrictive appositive) and `its enhanced fund`, which were not marked in the Treebank, and the relabeling of `its enhanced fund 's` as a possessive.

headword	frame LHS		RHS
big	JJ	→	——
indexer	NP	→	JJ —— NPR
Bankers	NNP	→	——
Trust	NNP	→	——
Co.	NPR	→	NNP NNP ——
also	ADVP	→	——
uses	S	→	NP ADVP —— NP PP-LOC .
futures	NP	→	——
in	PP-LOC	→	—— NP
a	DT	→	——
strategy	NP	→	DT —— SBAR
that	SBAR	→	—— S
on	PP	→	—— NP
average	NP	→	——
has	VBZ	→	——
added	S	→	PP VBZ —— NP PP
one	QP	→	——
percentage	NN	→	——
point	NP	→	QP NN ——
to	PP	→	—— NP
its	PRP$	→	——
enhanced		→	JJ ——
fund	NP	→	PRP$ JJ ——
's	NP$	→	NP ——
returns	NP	→	NP$ ——
.	.	→	——

Figure 6.3: The list of lexical entries extracted from Fig. 6.2 at step 9. (No information is lost in this step; it is reversible.) Each entry is shown partitioned into a headword and a frame; we wish to estimate Pr(headword | frame). —— denotes the position of the headword in the frame. Notice the flat frames for the lexical verbs `uses` and `added`, and the fact that `added` is missing its subject.

- Traces and other phonologically null elements are marked in the Treebank, so the maximal projection of a verb is always a sentence or `SBAR` (perhaps with null subject), never a `VP`.[5]

- Noun phrases and particularly noun-noun compounds are left relatively flat (a situation that we partly correct below).

- There are detailed conventions for where to attach punctuation in a tree. These are complicated by the fact that conventional English punctuation processes like comma absorption and quote ordering were not undone before hand-parsing.

Each Treebank tree was passed through the following sequence of modifications (compare (Eisner, 1996b)), each of which was implemented as a Perl script.[6] Some key steps in the transformation are shown in Figures 6.1–6.3.

1. *Canonicalize nonterminals.* The Treebank has a large nonterminal set because nonterminals sometimes include function tags or movement indices that indicate their semantic role. For example, `tomorrow` is a temporal noun phrase `NP-TMP`, which functions effectively as an adverb. Drop these tags and indices, except for those that indicate the allowable syntactic role of the phrase: namely, `TMP`, `LOC`, `ADV`, `PRD`, and the tags `NOM`, `SBJ`, `PRP`, `TPC` on sentential constituents. (Buchholz (1998) found that function tags were helpful in making the argument-adjunct distinction.)

2. *Articulate.* Assign more articulated structure to certain kinds of subtrees that the Treebank leaves flat. The idea is to group sibling phrases into a single phrase. Put

[5]Except in the case of reduced relative clauses: `funds [VP tracked` t `by the report]`, `a request [VP seeking approval]`. This convention for reduced relatives is of course arbitrary, and could be changed to introduce a sentential projection with empty subject. (Whether this projection should be `SBAR` or $\mathtt{S}_{\mathtt{part}}$ is a matter of current debate among linguists. The reduced relative would end up with either the same category `SBAR\NP` as full relatives, or a slightly different category $\mathtt{S}_{\mathtt{part}}\backslash\mathtt{NP}$. The first account would require a classical "whiz-deletion" transformation that removes the complementizer and `be`-auxiliary from `SBAR\NP`. The second would require a transformation that turns an `SBAR\NP` postmodifier into an $\mathtt{S}_{\mathtt{part}}\backslash\mathtt{NP}$, licensing the latter wherever the former appears.)

[6]These scripts had many other options to allow the preparation of different styles of corpus. For example, they can use the traces and other null elements in the Treebank to create a version of the corpus that indicates long-distance movement as the passing of gaps (and occasional "antigaps") through slashed nonterminals, in the style of GPSG. However, the experiments reported here did not attempt to model extraction transformations and so did not use this version of the corpus.

another way, we are "unflattening" the RHS of overly flat rules that do *not* consist of a head child and its dependents, as they should for our purposes.

(a) group sequences of proper nouns (`NNP`) into a proper noun phrase (`NPR`)

(b) group sequences of cardinal numbers (`CD`) into a quantity phrase (`QP`)

(c) group `$ QP` into `QPMONEY`[7]

(d) relabel `QP` as `QPMONEY` if it appears to be headed by a `QPMONEY` subphrase (e.g., `[as much as [$ 200 million]]`)

(e) when a sequence of common nouns `NN` immediately follows a proper noun phrase `NPR`, group the `NN` sequence into `NP`

(f) Correct some common annotator and tagger errors:

- following the first nominal in a noun phrase (`NP`), group any sequence starting with a determiner or preposition[8] into an `NP` or `PP` respectively.
- following the first preposition or adverb in a prepositional phrase (`PP`), group any sequence starting with a determiner or preposition into an `NP` or `PP` respectively.
- relabel `POS` as `VBZ` at the start of a phrase. This corrects a common tagger error of mistagging `'s` as a possessive morpheme when it is being used as a contraction of `is`.
- group any maximal sequence preceding `POS` into a noun phrase (`NP`), and group this phrase together with the `POS` into a possessive noun phrase (`NP$`).[9]

3. *Normalize case.* Change the case of each sentence-initial word, and each word in all-caps, to the most common case pattern for that word. For example, `THE` or sentence-initial `The` becomes `the`, but `IBM` and `WordStar` are left alone.

[7] Usually this `QPMONEY` is the head child of its `NP`, in which case the `QPMONEY` node will disappear at step 8. Before that, however, it is available for pattern-matching by the head-prediction rules at step 6.

[8] Up to the next determiner or preposition, or the end of the phrase. A `PP` created by this process is given the substructure `P NP`.

[9] This rule is applied recursively in case of nested possessives.

4. *Discard conjunctive sentences.* Discard sentences that contain conjunctions. This is because the scope of conjunctions is not clearly marked in the Treebank; see (Goldberg, 1999) for examples and a possible solution. Ignoring conjunctions also circumvents the usual problems with defining the head of a conjunctive phrase.

5. *Discard suspicious sentences.* Discard sentences that use phrase-structure rules in a list of "suspicious" rules that usually indicate annotation errors. These rules were discovered while writing patterns to choose heads.

6. *Predict heads.* Identify the head child of every nonterminal, based on the nonterminal tag and its child tags. The head child is usually identified by a set of hand-written patterns, but a list of hand-written exceptions can override these patterns. (See below for details.) If no head can be found, the rightmost child is chosen as the head.

7. *Delete traces.* Remove phonologically null constituents. If this removes a phrase's head, the rightmost child is chosen as the new head.

8. *Flatten structure.* For each nonterminal X whose head child Y is also a nonterminal, replace Y by Y's sequence of 1 or more children. This generally increases X's number of children and gives X a new head child (the old head child of Y), which is replaced recursively. In the resulting "flat" tree, each nonterminal X has a head child that is a terminal—the headword—as well as 0 or more nonterminal children, which are the dependents of the headword.

9. *Extract lexical entries.* It is simple to extract the phrase-structure rules that would be needed to generate this flat tree, such as $\texttt{S} \rightarrow \texttt{NP devour NP PP}$. These rules are our lexical entries (§2.4).

10. *Discard long rules.* To conserve memory and runtime, discard rules whose right-hand sides include more than 4 nonterminals in addition to the headword.[10] See footnote 14 on p. 183 for more on this.

[10] Otherwise, the training data would include a surprising number of inordinately long rules—not only for `NP` lists, but also for sentences whose headword has up to 15 dependents including punctuation. The transformational paths that derive such a rule by inserting 15 nonterminals into $S \rightarrow$ —— include $2^15 =$ $32,768$ entries, which all require storage and processing time. Some examples giving rise to long rules

The rules and exceptions used to choose heads (step 6) were developed as follows during an earlier project (Eisner, 1996b). I built an annotation tool for viewing phrase-structure rules from the corpus together with examples of each rule. Each rule was displayed with its annotated head child (if any) and its predicted head child (if any).

The annotated head could be changed by clicking. The predicted head could be changed by editing the head-prediction patterns in a Perl script, which were edited frequently to try to reduce the number of discrepancies between the annotated and predicted heads. The discrepancies that remained constituted the exception list.

The tool also made it possible to flag a phrase-structure rule as "suspicious," when no head could be annotated because the phrase-structure rule arose only (or usually) through annotator error. Sentences containing suspicious rules were removed from the corpus in step 5.

As the sole annotator, I found that it was quite fast to check whether predicted heads were correct, and if not, to correct them. I stopped annotating when the patterns had developed to the point where they seemed almost always right.

For each nonterminal LHS tag, the pattern for predicting the head child on the RHS was a short decision list something like this: "If there is exactly one child from $\{X_1, X_2, X_3\}$, then it is the head. Otherwise, if there is a final Y child, then it is the head. Otherwise, if there is at least one child from $\{Z_1, Z_2\}$, then the rightmost one is the head. Otherwise ..." If the RHS contained a comma, then the pattern was first used to try to find a head in the portion of the RHS before the first comma. If this failed, the pattern was reapplied to the entire RHS.

(headwords underlined, dependents in brackets):

- [“] [Who knows] [,] [”] [he says] [,] [“] [if he heard that soybeans make money today] [,] [he] [might] [be] <u>flying</u> [out to Chicago] [tomorrow] [.]
- [With fast-food outlets on every corner] [,] [he] [,] [like many] [,] [does] [n't] <u>think</u> [he has a choice in the price war] [:] [“] [Our customers say that ...] [.] [”]
- [The bank] [, he wrote back,] [was] [“] [immediately] [”] <u>lowering</u> [the rate] [by 3.5%] [,] [“] [as a concession to you] [.] [”]

There are various possible engineering solutions short of discarding such data (as here) or recoding it. One is to modify the transformation graph so that (for example) successive insertions must proceed from left to right, or be made simultaneously. (In the simultaneous case, the fan-out from each vertex of the transformation graph would be dramatic; each arc's probability would be determined from a complex feature vector, or from a finite-state transducer as in footnote 12 on p. 227.) This is slightly tricky to arrange but is probably doable.

The annotation convention was to select the semantic head of a phrase rather than the syntactic head. In particular, the head of a sentence was always the main verb, rather than a tensed auxiliary. Punctuation never served as the head of a larger phrase.

6.3 Some Properties of the Experimental Data

6.3.1 Datasets

The experiments used sections 0–15 of the Treebank for training, section 16 as a development set used to tune the models both automatically and manually, and section 17 for final testing. (The remaining sections were reserved for future experiments; neither they nor section 17 were looked at during the research.)

A very few of the entries in the development and test sets used nonterminals that never appeared in the training set. These entries were discarded since any of the methods would have assigned them 0 probability.

The present experiments considered only lexical entries of the form $\texttt{S} \rightarrow \cdots$, which represent a verb together with all its arguments (perhaps including a subject). Such entries are highly varied, making their prediction especially challenging. They are also key to successful parsing. Indeed, Johnson et al. (1999) evaluated parsing performance in part on whether the parser found the correct lexical entry for the matrix `S`. Basili et al. (1999) showed that a lexicon of `S` lexical entries, automatically acquired from the output of a shallow non-statistical parser, could be used to improve the parser's `PP`-attachment performance. Similarly, Carroll et al. (1998) showed that a lexicon of `S` lexical entries, automatically acquired from the output of a non-lexicalized statistical parser, could be used to improve the parser's precision at recovering labeled dependencies, without seriously hurting recall. (See §2.5.4 for other work that has focused on predicting `S` entries.)

It was possible to focus on just the `S` entries because the particular transformation model in the experiments did not allow category-changing transformations (§5.4.4). That is, there are no transformations between `S` entries and other entries.

Fitting LHS-specific models in this way reduces the memory footprint of the programs. It also means that parameters that control the strength of smoothing (see §3.5 and §6.6.2)

	all entries			$\texttt{S} \rightarrow \cdots$ entries		
	train	dev	test	train	dev	test
entry tokens	290903	24789	14941	18836	1588	973
entry types	66477	9188	5995	11565	1317	795
frame types	8732	1845	1361	2722	564	365
headword types	22084	4957	3610	3607	756	504
novel entry tokens		17.5%	17.7%		51.6%	47.8%
novel frame tokens		1.6%	1.5%		8.9%	6.3%
novel headword tokens		4.9%	5.4%		10.4%	10.2%
novel entry types		44.9%	42.1%		61.4%	57.5%
novel frame types		20.3%	15.4%		24.6%	16.4%
novel headword types		20.7%	18.9%		20.9%	18.8%
nonterminal types	121			78		
# transformations applicable to entry with RHS length $= n$	$244n-1$	$244n-1$	$244n-1$	$158n-1$	$158n-1$	$158n-1$

Figure 6.4: Properties of the experimental data. For example, the training data consisted of 290,903 (word,frame) entries (where frame=(LHS,RHS)), but only 66,477 of these entries and only 8,732 of the frames were distinct. "Novel" means "never observed in training."

are tuned specifically for estimation of a particular LHS. For example, S entries might require different smoothing from, say, JJ (adjective) entries since it is rare for an adjective to take dependents. On the downside, fitting LHS-specific models misses the potential synergies of training simultaneously on more heterogenous data, namely, the ability to combine evidence across LHS categories.[11]

6.3.2 Statistics

Statistics about the datasets are shown in Fig. 6.4—in particular, the degree to which novel events appeared in the development and test sets. (The test set was somewhat less novel.)

The rate of novel events is high in part because the dataset uses a large event space. As Fig. 6.3 showed, a lexical entry specifies a particular headword and nonterminal slots

[11] Such synergies arise if parameters are tied across different LHS categories. This is true in all the models in this chapter, making it possible for training frames $X \rightarrow \cdots$ to improve the estimation of frames $Y \rightarrow \cdots$. In the transformation model, some feature weights are independent of LHS (§6.4.4). In the other models, backoff from LHS is possible, although it happens only in the rare instances where the LHS category has been poorly observed.

for all arguments and adjuncts. However, a complex event need not be particularly hard to model. The models we tried were allowed to "back off" from the headword, and also to generate the frame piecemeal.

Fig. 6.4 also shows that there are still many novel frames even when headwords are ignored. §1.2.1.2 mentioned that 49% of all sentences contain such a frame. The rate of novel frames would have been still larger if the frames had included slashed nonterminals (footnote 6 on p. 177).

§2.4.2.1 demonstrated that frames that appeared with the same word were disproportionately related by simple edit-distance transformations.

6.4 Topology and Parameterization of the Transformation Model

To specify a transformation model (§3.2.1 or §4.1.1), one must specify the events; the arcs, which represent possible transformations; and the features on those arcs. The model used in the experiments was outlined and motivated in §2.4; this section gives details. A fragment of the model is drawn in Fig. 1.3 on p. 21.[12]

6.4.1 Events

The events are flat lexicalized context-free rules, such as `S` $\to$ `NP devour NP PP`, over a finite set of nonterminals.[13] As before, we refer to the nonterminal to the left of the arrow as the left-hand side (LHS), and the string of nonterminals to the right of the arrow as the right-hand side (RHS). Because the RHS can be arbitrarily long,[14] the graph is

[12]However, that figure depicts per-event arc weights such as θ_8 and θ_9. The results reported here used per-event perturbations π_8 and π_9 instead.

[13]We use the template strategy (§4.5.3) of processing the frames for each word separately. So we are primarily interested in storing and manipulating frames like `S` $\to$ `NP` —— `NP PP`. It is convenient to represent frames as strings of nonterminals; then standard string libraries may be used to copy, compare, and hash them. Each nonterminal should be represented as an arbitrary character, as should ——. A frame with k symbols on the RHS can then be represented with $k+2$ characters (LHS, RHS, terminator). When there are more than 255 nonterminals (e.g., because nonterminals can encode gaps or features), this requires library support for strings of so-called wide (2-byte or n-byte) characters. However, in C or C++, wide-character strings can actually be handled without special support, provided that all bytes of all characters are non-zero, except for the final byte of the terminator.

[14]The length of the RHS in training and test data was artificially limited by step 10 in §6.2. However, the model does not know to expect data with this property (nor does it have any features, such as output

conceptually infinite, although the relaxation algorithm explores only a finite portion of it during training and testing (§4.2).

We use the term **frame** as defined in §5.1 and §5.3.2, to refer to a lexical entry with the headword abstracted away.

There are also several special events that are not lexical entries. As always, there is a distinguished node $\textsc{Start}$; and for every word ℓ, there is a node $\textsc{Start}_\ell$.

6.4.2 Arcs

There are arcs from $\textsc{Start}$ to each $\textsc{Start}_\ell$. For each ℓ, there are arcs from $\textsc{Start}_\ell$ to all lexical entries whose headword is word ℓ and whose frame appeared in training data (perhaps with some other headword).

Finally, from each lexical entry e, there are arcs to $\textsc{Halt}$ and to all other lexical entries at edit distance 1 from e. These are entries with the same LHS and headword as e, but their RHS has been modified by a single edit from §2.4.2 (Insert, Delete, Substitute, Swap).

The resulting transformation graph is a multigraph. For example, there are two arcs from $\mathtt{X} \to \mathtt{Y}$ —— $\mathtt{Z}$ to $\mathtt{X} \to \mathtt{Y}$ —— $\mathtt{Z}$ $\mathtt{Z}$, since there are two distinct "Insert Z" transformations that turn the former into the latter. Since these two transformations insert Z into different contexts, they will in general have different probabilities: inserting Z at the right edge of a frame may be more probable than inserting it medially.

Self-loop Substitute arcs from e back to e are disallowed, both for efficiency (as such arcs would otherwise be pervasive) and to discourage the feedback effect for perturbed models mentioned in footnote 22 on p. 112.

Fig. 6.4 notes that the graph has many vertices and quite large fan-out. The implementation allocates storage for as few vertices as possible, and generates arcs on the fly when needed. Most arcs from each vertex can be ignored (§4.5.2), and the implementation benefits greatly from storing at each arc the short list of out-arcs that should be used for propagation or relaxation.

features in §3.6.2, that are sensitive to RHS length and so would enable parameter estimation to discover this property in the training data). In particular, the transformation graph still includes arcs to and from entries with longer RHS. These arcs have positive probability, so they do affect the performance of the model. They may also contribute to useful paths.

6.4.3 Perturbations and Per-Event Features

The implementation can handle both per-event features (§3.6.1) and perturbations (§3.9). These are used to tune the probabilities of individual lexical entries so as to better fit the observed data. (The effect flows through the transformation graph to other entries.) Without such tuning, frames and headwords would be statistically independent: $\Pr(f \mid w) = \Pr(f)$ regardless of w.

We allow only *observed* lexical entries to have per-event features or perturbations. This keeps the number of parameters finite.[15]

6.4.4 Features of Transformations

An arc from one lexical entry e to another entry e' represents an edit. For purposes of defining features, it is characterized by

- the **type** of edit (Insert, Delete, Substitute, or Swap)
- the **LHS** nonterminal common to e and e'
- the **target** nonterminal (that is being respectively inserted, deleted, replaced, or moved rightward)
- the **replacement** nonterminal (in the case of Substitute only)
- the nonterminal to the **left** of the target
- the nonterminal to the **right** of the target
- the **side** where the target appears (pre-head or post-head)

The left or right nonterminal may also be the special symbol —, which represents the headword, or the special symbol $\diamond$ to represent the edge of the rule. In the case of Swap, the target nonterminal may be —.

[15]However, it partly precludes two potential uses of tuning. In principle, parameter estimation can tune down $\Pr(e)$ to explain why e has been observed less than predicted, and it can tune up $\Pr(e)$ to explain economically why several of e's children have been observed more than predicted. These moves become impossible if e is denied a tuning parameter because it has never been observed at all.

(type) (type, left) (type, right) (type, left, right)	*(type, LHS) *(type, LHS, left) *(type, LHS, right)	(type, target) (type, target, left) (type, target, right)	*(type, LHS, target)
(type, side) (type, side, left) (type, side, right) (type, side, left, right)	*(type, side, LHS) *(type, side, LHS, left) *(type, side, LHS, right)	(type, side, target) (type, side, target, left) (type, side, target, right)	*(type, side, LHS, target)

Figure 6.5: Tuples that serve as templates for features on Insert and Delete arcs. (In the experiments of this chapter, where LHS was fixed, the tuples marked with * were suppressed as redundant.) The features of a given arc are found by instantiating these tuples with the arc's actual type, LHS, etc. Thus, $\vec{\theta}$ specifies a weight for each instantiated tuple.

Ignoring per-event features, each Insert or Delete arc has 22 features, corresponding to the tuples in Fig. 6.5. Each tuple specifies the arc type, at most two of the nonterminals {LHS, target, left, right}, and perhaps the side.

For example, the arc that transforms `S` $\rightarrow$ `NP devour NP` into `S` $\rightarrow$ `NP devour NP PP` has 22 features corresponding to these tuples. One of these features indicates that (type, target, right)=(Insert, PP, $\diamondsuit$), and appears on *all* arcs with that property. This feature has high weight in English—making all such arcs probable—because English does indeed like to insert PP at the right edge of a rule.

Each Swap arc has features corresponding to just those tuples above that do not mention "left." So the G-value of a Swap arc depends on the two nonterminals being swapped (target and right) but not their context.

A Substitute arc uses the same tuples as a Swap arc, except with "replacement" in place of "right." So the G-value of a Substitute arc depends on the substitution (the target and its replacement) but not its context, except as encoded by LHS or side.

6.4.5 Other Features

The features on arcs from START to START_ℓ do not matter for the frame prediction problem. The probability β_ℓ on such an arc is the probability of choosing word ℓ as the headword. As explained in §6.1.1, we are only concerned with a conditional probability *given* word

ℓ.[16]

The arc from START_ℓ to a lexical entry e has an output feature (§3.6.2) that depends on the frame of e. Thus, the arc from $\text{START}_{\texttt{devour}}$ to `S` $\rightarrow$ `NP devour NP PP` has the same feature as the arc from $\text{START}_{\texttt{regurgitate}}$ to `S` $\rightarrow$ `NP regurgitate NP PP`. This means—if we ignore per-event features on these arcs and any perturbation parameters—that these lexical entries have equal conditional probabilities given their headwords.

An arc from e to HALT has a single feature called the "HALT feature." (It has no features that are sensitive to the form of e.) All such arcs therefore have the same G-value, so that their probabilities are determined entirely by the features on their competitors.

Notice that if e has many out-arcs, then almost any one of them—including HALT—will have low probability. This happens in particular if e has a long RHS, because then there are many different positions into which to insert a nonterminal; see Fig. 6.4.

6.5 Details of Computing with the Transformation Model

This section gives some details that would be necessary to replicate the results reported here. The constants mentioned here were chosen informally, by trying a few values on the development set of `S` $\rightarrow \cdots$ entries.

6.5.1 Smoothing Parameters

The prior was the simple one specified in §3.5, where the feature weights θ_i are IID $\sim N(0, \sigma^2)$. All experiments took $\sigma^2 = 3$.

The special features on arcs from START_ℓ and arcs to HALT were treated exceptionally, however. They were given a flat prior (e.g., a normal with $\sigma^2 \rightarrow \infty$), so they were not biased toward zero at all.

[16]We use a template-structured computation (§4.5.3), which for ℓ a headword, computes $\Pr(\ell, \text{LHS}, \text{RHS})$ as a product

$$\underbrace{\beta_\ell}_{\Pr(\ell)} \cdot \underbrace{\mathtt{s}_\ell p_{(LHS,RHS)}}_{\Pr(\text{LHS},\text{RHS}|\ell)}$$

So the probability we want is actually $\mathtt{s}_\ell p_{(LHS,RHS)} / \sum_{RHS} \mathtt{s}_\ell p_{(LHS,RHS)}$. For the present experiments, the denominator is 1 since we built LHS-specific models.

These experiments used perturbations rather than per-event weights. The prior on perturbations π_i took them to be IID $\sim N(0, \sigma^2)$ where $\sigma^2 = 5$.

6.5.2 Parameter Initialization and Optimization

The ordinary parameters $\vec{\theta}$ and $\vec{\pi}$ had to be learned from data. They were optimized using adaptive gradient ascent on the objective function.

As the parameters changed during gradient ascent, the model's performance (cross-entropy) on the development set was monitored. Optimization was halted after the performance appeared not to be improving any longer.[17] The parameters used for testing were the ones that minimized cross-entropy of the development set during this run.

Although simply initializing $\vec{\theta}$ and $\vec{\pi}$ to 0 gave passable performance (see §6.7.7), a better starting guess for the parameters proved important for two reasons. First, it allowed gradient ascent to reach a better local maximum of the objective function. Second, it improved efficiency by giving the paths unequal probability, so that the relaxation algorithm was able to concentrate on propagating along the most promising paths.

To choose our initial value of $\vec{\theta}$, we performed a single pass of the EM algorithm that will be described in §8.2, but with an easy-to-compute heuristic guess in place of the E step. The intuition is that if two training-set entries differ by edit distance 1, then the features of the relevant edit should get a high weight.

Specifically, recall that s_i denotes the number of training instances of the lexical entry i. (If i is an event of the form START_ℓ, temporarily define s_i to be the constant 0.4.) For $j \neq 0$, let N_{ij}, the guessed number of transitions from i to j, be $(s_i / \sum_{i'} s_{i'}) s_j$, and let N_{j0} be s_j. The M step then chose feature weights $\vec{\theta}$ by Improved Iterative Scaling so that the arcs with high N_{ij} values also had high probability.

In other words, we guess that of the s_j random walks that were observed to halt at j (i.e., to traverse arc $j0$), the number that reached j from a given parent lexical entry i is proportional to the number of other walks that are known to have actually halted at i, namely s_i.[18] We also guess that the number of walks that reached j from the appropriate

[17] In some cases the cross-entropy leveled off. In others it started getting worse, apparently representing overfitting of the training data.

[18] This heuristic ignores the fact that the s_j walks ending with $\langle \ldots, i, j, \text{HALT} \rangle$ will have traversed other

START_ℓ vertex is proportional to 0.4.

Once the weight vector $\vec{\theta}$ was initialized in this way, the perturbation vector $\vec{\pi}$ was initialized such that the amount of flow added at vertex i (see §4.4), namely $\delta_i \stackrel{\text{def}}{=} (\exp \pi_i - 1)(I_{\theta,\pi})_i$, was $\delta_i = 3 \cdot (s_i / \sum_{i'} s_{i'})$. In this case s_i makes no special provision for START_ℓ nodes.

6.5.3 The Two-Stage Relaxation Strategy

To compute the objective and evaluation functions, it was necessary to solve the model for the distribution. The technique used was interesting. It defined a non-deficient probability distribution over the infinite lexicon such that the exact probability of any lexical entry could be found in finite time.

A relaxation strategy (§4.2) was employed, with the same double stack usually used for the entire run to avoid unnecessary bumpiness in the objective function (§4.5.1). The relaxation was divided into two stages that used different transition matrices P (see §4.3.6). The first stage propagates probability mass along all transformational paths of reasonably high probability. The second stage continues the propagation indefinitely along Insert and Halt arcs only.

The approximation here consists in the fact that the second stage does not use the "true" probability transition matrix P_θ. Instead it uses a modified (but still Markovian) version of this matrix, in which the probability of non-Insert arcs from each vertex i has been reallocated to the Halt arc from i.[19] So Delete, Substitute, and Swap arcs are only available early in a random walk (i.e., during the first stage).

The idea is that relaxation can now be run, in effect, forever. The first stage is halted early, but relaxation continues in a second stage that operates over an *acyclic* graph and so can be regarded *as if* it had run forever. How so? During the second stage, the relaxation priority queue is defined so that short RHS implies high priority. In other words, the vertices on the queue are relaxed in topologically sorted order. Now suppose k is the maximum length of any RHS in the test set. Then by the time all vertices with RHS

arcs in order to reach i.

[19] This requires a minor change to §4.3.5, since it changes the partials $\partial P_{ij}^{(t)} / \partial \theta_h$.

length $< k$ have been relaxed during the second stage, all the vertices in the test set have settled on their final probabilities *just as if* the relaxation had continued forever. At this point relaxation can be halted in practice, since we have our final answer.

Notice that vertex j will surely receive some probability mass during the second stage, so long as it has some "Insert ancestor" i on the queue when the first stage ends. In particular, if $\mathtt{S} \to$ —— is on the queue when the first stage ends, then all lexical entries $\mathtt{S} \to \cdots$ are guaranteed to end up with positive probability, preventing cross-entropy from being infinite.

The first stage ends when no vertex on the priority queue has priority ≥ 0.0005. The priority of vertex i is roughly defined as J_i. However, using J_i itself as the priority would prevent the discovery of paths between uncommon lexical entries (which are very useful clues to transformations).[20] Instead, we take the priority to be a new variable K_i, which is initialized and updated identically to J_i except that relaxing $i = \textsc{Start}$ or $i = \textsc{Start}_\ell$ is handled specially: each child j of such an i increases K_j not by $K_i P_{ij}$, as usual, but by K_i. In general K_i may be interpreted as the total probability of the edit sequences whereby J_i was propagated to i.[21]

6.6 Competing Models

6.6.1 Models from the Parsing Literature

This section lays out the basic models being compared with the transformation models. All of them are generative models. That is, $\Pr(\text{RHS} \mid \text{headword}, \text{LHS})$ is obtained as the probability that some stochastic RHS-generating process (whose probabilities are influenced by headword and LHS) would have generated RHS.

[20] To see why, note that the approximated objective function $\tilde{f}$ provides no incentive to adjust the weights on a path $i, i', i'', \ldots$ unless that path actually affects $\tilde{f}$ by carrying probability mass to an entry that was observed in training. For that to be so, $i, i', i'', \ldots$ must be dequeued in that order, and hence at a minimum must all have priority > 0.0005. But if there are no high-probability paths from $\textsc{Start}$ to these nodes—for example, if i has only been observed once in training data and its children not at all—then they will have small J_i; so we would like to define priority to be something larger.

[21] That is, the path probabilities along which J_i was propagated, if we consider only the part of the path corresponding to actual edits, without the initial arcs from $\textsc{Start}$ and $\textsc{Start}_\ell$.

6.6.1.1 Memorization Model

Some recent parsers (Charniak, 1997; Carroll and Rooth, 1998) have modeled Pr(RHS | headword, LHS) as a simple multinomial distribution (with backoff to Pr(RHS | LHS)).[22] That is, it generated RHS in a single step as if the RHS had no internal structure.

Charniak called this a "Treebank grammar" because the allowable phrase-structure rules were limited to the ones that had appeared in the training data from the Penn Treebank.

§6.1.2 already noted that this model assigns infinite perplexity to the full test set. However, it does an excellent job of modeling the non-novel frames in the test set. We will later combine it with other models.

6.6.1.2 Markov Models

In Fig. 6.3, the right-hand side (RHS) of each frame can be regarded as a string of nonterminals. A distribution over such strings can be modeled with an n^{th}-order Markov model, also known as an n-gram model. The idea is that the RHS is generated randomly from left to right, with each nonterminal symbol chosen with a probability that depends on the substring formed by the $n-1$ previously generated symbols. The generation process terminates upon generation of a special symbol $\diamondsuit$.[23]

There are two complications. First, the choice of RHS string should be conditioned on the headword and LHS. Second, the RHS string must contain exactly one copy of ——. So generation of each nonterminal should be conditioned on the headword, the LHS, and a boolean variable indicating whether —— has been generated yet.[24]

For example, the probability $\Pr(\texttt{PP VBZ} \text{ —— } \texttt{NP PP} \mid \texttt{added}, \texttt{S})$ would be modeled under a trigram (i.e., 3-gram) model as

$$\Pr(\texttt{PP} \mid \epsilon, \texttt{added}, \texttt{S}, false) \cdot \Pr(\texttt{VBZ} \mid \texttt{PP}, \texttt{added}, \texttt{S}, false)$$

[22] Actually, Charniak's parser also conditioned on additional information not available in the frame prediction problem. See §5.5.5.1.

[23] Which eventually happens with probability 1, if the model parameters were estimated in any sensible way from a training set of finite RHS strings.

[24] If this variable is $true$, then the probability of generating —— (again) is 0, so there can be at most one ——. If the variable is $false$, then the probability of generating $\diamondsuit$ is 0, so there must be at least one —— in any completed string.

$$\cdot \quad \Pr(\text{——} \mid \texttt{PP VBZ}, \texttt{added}, \texttt{S}, false) \cdot \Pr(\texttt{NP} \mid \texttt{VBZ} \text{——}, \texttt{added}, \texttt{S}, true)$$

$$\cdot\Pr(\texttt{PP} \mid \text{——}\ \texttt{NP}, \texttt{added}, \texttt{S}, true) \cdot \Pr(\Diamond \mid \texttt{NP PP}, \texttt{added}, \texttt{S}, true) \tag{6.5}$$

Notice that the string of previously-generated symbols does not reach length $n-1$ until the n^{th} factor.

There is some history in the literature of using Markov models for frame prediction. The model C parser in (Eisner, 1996c; Eisner, 1996b) used a bigram model for frame prediction. The model 1 parser of (Collins, 1997) used a unigram model. Charniak (2000) tried both a unigram and a bigram model and found that the latter led to higher parsing accuracy.

The head-automaton parser of (Alshawi, 1996) used a kind of hidden Markov model that will be explored further in §6.6.3. Finally, the memorization model (§6.6.1.1) can be regarded as an n-gram model with very large n and no backoff to smaller n (see §6.6.2).

6.6.1.3 Subcategorization Model

The model 2 parser of (Collins, 1997) relies on a heuristic distinction between argument and adjunct nonterminals in the RHS. It cleverly combines a memorization model of the argument sequence with a unigram model of the adjunct sequence.

Collins exploits the linguistic intuition that a verb subcategorizes for arguments but not adjuncts. A given verb may strongly constrain the number and order of subjects and objects (arguments) it allows, but not the number or order of adverbs and prepositional phrases (adjuncts). For example, if an adverb can appear at a particular position among the arguments with probability p, then a second adverb will appear there with probability p as well, and so on. But subjects cannot be iterated in the same way as adverbs.

To generate the sequence of RHS nonterminals to the right of ——, Collins's stochastic process first chooses, in one step, the unordered collection C_R of *arguments* that will appear in that sequence. In our example `PP VBZ —— NP PP`, this is just {`NP`}. It then generates the sequence of all RHS nonterminals to the right of ——, using a modified unigram model in which each nonterminal is chosen with a probability that depends not only on the headword and LHS, but also on the subset of C_R that has not yet been generated.[25] So

[25] Removing already-generated (close-to-the-head) arguments from C_R is similar to category cancellation

the NP is chosen given that arguments {NP} must still be generated, while the PP and $\diamondsuit$ are chosen given that {} is the set of arguments still to be generated (i.e., the generation process is at the edge of the subcategorization frame). Notice that NP, PP, and $\diamondsuit$ are likely choices in these respective contexts.

The sequence of LHS nonterminals is generated similarly, but in the opposite direction (from right to left): VBZ, PP, $\diamondsuit$.

Collins's model further conditions each nonterminal choice on a variable Δ, which encodes whether there were any previous nonterminals generated and, if so, whether any of them (transitively) dominated a verb or a comma. The latter test is not possible in the pure frame prediction task studied here, since the RHS in our dataset does not indicate which nonterminals dominate verbs or commas. However, we approximate the verb test by assuming that nonterminals beginning with the letter S or V dominate a verb, and others do not.

In replicating Collins's model, we were careful to mark argument nonterminals with a suffix -C, as he does, *before* processing the Treebank as in §6.2. This was important since his heuristics for finding arguments assume standard Treebank structure. The -C marks were preserved through the data preparation steps of §6.2, which otherwise ignored them.

6.6.1.4 Maximum Entropy Models

Charniak (2000) uses an approach that is "inspired" by maximum-entropy modeling, in the sense that each probability is modeled as a product of weights. (The weights in Charniak's approach are themselves ratios of smoothed probabilities.)

We do not replicate Charniak's technique here, because it does not yield normalized probabilities—making it incomparable with other techniques on a perplexity measure, as Charniak notes—and because crucial details are deferred to a longer version of the paper that is not yet available. However, it is worth noting that actual maximum-entropy (log-linear) approaches, which have not been tried for this problem, would be interesting competition for transformational approaches. Johnson et al. (1999) model entire trees using a maximum-entropy approach, and with an appropriate feature set the same approach

in categorial grammars, or the Valence Principle in HPSG.

could undoubtedly be used to model individual lexical entries.[26] §1.2.3 gave an intuition about why transformational approaches were more appropriate; however, maximum-entropy models have the advantage that they allow global optimization of parameters.

6.6.2 Backoff

Most of the probabilities required by the above models cannot be estimated directly from the amount of data available. For example, to estimate the following factor from equation (6.5),

$$\Pr(\mathtt{NP} \mid \mathtt{VBZ} \text{ ——}, \mathtt{added}, \mathtt{S}, true) \tag{6.6}$$

one would like to measure how often NP was generated in training data, in the context $\mathtt{VBZ} \text{ ——}, \mathtt{added}, \mathtt{S}, true$:

$$\frac{\#(\mathtt{NP}, \mathtt{VBZ} \text{ ——}, \mathtt{added}, \mathtt{S}, true)}{\#(\mathtt{VBZ} \text{ ——}, \mathtt{added}, \mathtt{S}, true)} \tag{6.7}$$

where $\#(e)$ is the count of e in training data (§3.1). But the denominator may be too small for this ratio to be reliable. Indeed the denominator is often zero.

The standard solution is to "back off" and combine equation (6.7) with a coarser probability that considers how often NP was generated in a variety of similar contexts as well:

$$\Pr(\mathtt{NP} \mid \text{——}, \mathtt{added}, \mathtt{S}, true) \tag{6.8}$$

This coarser probability may in turn be estimated by a second level of backoff, and so on recursively.

To give these models a good chance of success, we will consistently use the "one-count" backoff method devised by Chen and Goodman (1996), who compared a variety of backoff methods for n-gram language modeling and recommended this one for its superior performance and simplicity.[27] The sequence of backoff contexts chosen for each distribution

[26] Although computing the partition function (normalization constant) might be difficult. Johnson et al. (1999) circumvent this problem by adopting a conditional likelihood approach, where they maximize the correct parse tree's likelihood *relative to* competing parses. This approach is not obviously applicable unless one parses the input and obtains a set of competing lexical entries for each word.

[27] The backoff method actually used by (Collins, 1997; Charniak, 1997; Carroll and Rooth, 1998) was to interpolate directly among backed-off estimates. Carroll and Rooth condition their interpolation parameter on part of the denominator (the word); it is not clear what Collins and Charniak conditioned their interpolation parameters on, if anything. But even if they conditioned on the full denominator (Bahl et al., 1983), Chen and Goodman claim that one-count smoothing (which *also* conditions on the number of singletons) works better.

Model	memorization	trigram (other n-gram similar)	mem + n-gram
$\Pr(\cdots)$	RHS $\mid$ LHS, w	$R_i \mid (i < \text{head}), R_{i-1}, R_{i-2}, w$	RHS $\mid$ LHS, w
Level 1	RHS $\mid$ LHS	$R_i \mid (i < \text{head}), R_{i-1}, R_{i-2}$	RHS $\mid$ LHS
Level 2		$R_i \mid (i < \text{head}), R_{i-1}$	$\Pr_{n\text{-gram}}(\text{RHS} \mid \text{LHS}, w)$
Level 3		$R_i \mid (i < \text{head})$	

Model	subcategorization (Collins)		
$\Pr(\cdots)$	$R_{\text{head}} \mid \text{LHS}, w$	$C_{\text{side}} \mid \text{side}, \text{LHS}, R_{\text{head}}, w$	$R_i \mid \text{side}, C_i, \text{LHS}, R_{\text{head}}, \Delta_i, w$
Level 1	$R_{\text{head}} \mid \text{LHS}$	$C_{\text{side}} \mid \text{side}, \text{LHS}, R_{\text{head}}$	$R_i \mid \text{side}, C_i, \text{LHS}, R_{\text{head}}, \Delta_i$
Level 2		$C_{\text{side}} \mid \text{side}, \text{LHS}$	$R_i \mid \text{side}, C_i, \text{LHS}, R_{\text{head}}$
Level 3			$R_i \mid \text{side}, C_i$

Meaning of variables for `S` $\rightarrow$ `NP told NP PP SBAR` (*He told her with rice that she was lovely*):

w	`told`
LHS	`S`
RHS	`NP` —— `NP PP SBAR`
$R_1, R_2, \ldots R_6$	`NP`, ——, `NP`, `PP`, `SBAR`, $\diamondsuit$
head	2 (position of head child; possible in §6.6.3 for $R_{\text{head}} \neq$ ——)
C_R	{`NP`,`SBAR`} (complements to right of head)
C_3, C_4, C_5, C_6	{`NP`,`SBAR`},{`SBAR`},{`SBAR`},{} (C_i = subset of C_R (or C_L) not generated before R_i)
$\Delta_3, \Delta_4, \Delta_5, \Delta_6$	none,some,some,verb (Δ_i describes symbols intervening between R_{head} and R_i)

Figure 6.6: Backoff levels for all smoothed distributions. The $\Pr(\cdots)$ line describes the conditional distribution being estimated, and subsequent lines describe successively coarser backoff estimates.

is shown in Fig. 6.6.

The one-count method modifies equation (6.7) by adding kp to the numerator and k to the denominator, where p is the estimate of the coarser probability (6.8) and k is large if equation (6.7) is likely to be an unreliable estimate. Specifically, k is an (increasing) linear function of the number of different nonterminals x such that $\#(x, \texttt{VBZ ---}, \texttt{added}, \texttt{S}, true) = 1$.

The slope and intercept of this linear function must be specified for each backoff level. For example, if equation (6.8) is itself estimated by backoff, then there are 2 levels of backoff and 4 parameters to specify. We consistently used Powell's method (Press et al., 1992) to search for the parameter vector that maximized the conditional cross-entropy (6.3) on development data.

It is worth noting that the backed-off estimate $\frac{a+kp}{b+k}$ can be written as $\alpha\frac{a}{b} + (1-\alpha)p$, which interpolates between the "raw" estimate $\frac{a}{b}$ and the coarser estimate p. Here $\alpha = \frac{b}{b+k}$, so that for a given k, the raw estimate contributes more if it has a large denominator. Again, k depends on the number of singleton events in the given context.

6.6.3 Unflattened Models and Head Automata

It might be argued that the models of §6.6.1 are operating at a disadvantage on the task of this chapter. Most of them were developed to predict the RHS of Treebank-sized context-free rules.[28] By contrast, the rules in our dataset are larger, thanks to the flattening in step 8 of §6.2. For example, Collins's subcategorization model (§6.6.1.3) may have to generate larger argument sets C_L and C_R in order to generate our frames.

Without flattening, the frame $\texttt{S} \to \texttt{PP VBZ ---- NP PP}$ would have been built up by a sequence of context-free rewrites:

$$\texttt{S} \to \texttt{PP}\ \underline{\texttt{VP}} \qquad \texttt{VP} \to \texttt{VBZ}\ \underline{\texttt{VP}} \qquad \texttt{VP} \to \underline{\texttt{VBN}}\ \texttt{NP PP} \qquad \texttt{VBN} \to \texttt{----} \tag{6.9}$$

where the underlined head child in each rule is expanded by the next rule. The parsers discussed above therefore generally compute not $\Pr(\texttt{PP VBZ ---- NP PP} \mid \texttt{added}, \texttt{S})$ but

[28] Although Eisner and Alshawi did use dependency-style parses that were as flat as the ones considered here.

rather the more specific probability[29]

$$\begin{aligned}&\Pr(\texttt{PP [}_{\texttt{VP}}\texttt{ VBZ [}_{\texttt{VP}}\texttt{ [}_{\texttt{VBN}}\texttt{ ——] NP PP]]} \mid \texttt{added}, \texttt{S}) && (6.11)\\ &= \Pr(\texttt{PP } \underline{\texttt{VP}} \mid \texttt{added}, \texttt{S}) \cdot \Pr(\texttt{VBZ } \underline{\texttt{VP}} \mid \texttt{added}, \texttt{VP}) \\ &\quad \cdot \Pr(\underline{\texttt{VBN}}\texttt{ NP PP} \mid \texttt{added}, \texttt{VP}) \cdot \Pr(\text{——} \mid \texttt{added}, \texttt{VBN}) && (6.12)\end{aligned}$$

Is flattening a good thing or a bad thing? §1.2.1.2 suggested that it was a good thing because the articulated (non-flat) formula in equation (6.12) makes overly strong independence assumptions. For example, equation (6.12) clearly misses the statistical dependence between the past participle VBN and its auxiliary VBZ. Also, Johnson (1999) systematically tried flattening certain Treebank constructions, such as PP-modification, and found that this improved precision and recall of a non-lexicalized PCFG parser.

On the other hand, it might be argued that the articulated structure is better in certain ways. The memorization and subcategorization models can more easily estimate the probabilities of several small, common context-free rules than the probability of the big frame that the rules collectively generate. Moreover, the generative process that produces a frame by context-free rewrites is more powerful than any mechanism in the models of §6.6.1.[30] By encapsulating VP as a constituent, a rule such as $\texttt{S} \to \texttt{NP } \underline{\texttt{VP}}\ \texttt{.}$ can capture the "long-distance" generalization that a VP followed by a period is likely to be preceded by a subject, no matter how many arguments are in the expansion of the VP.

Indeed, the stochastic context-free rewrite model for generating a headword's RHS is equivalent to the head automaton model of Alshawi (1996). The internal nonterminals

[29] It should be noted that Charniak (2000) reports a substantial performance improvement by following (Collins, 1997) and replacing the product (6.12) with

$$\begin{aligned}\Pr(\texttt{VBN} \mid \texttt{added}) &\cdot \Pr(\texttt{PP } \underline{\texttt{VP}} \mid \texttt{added/VBN}, \texttt{S}) \cdot \Pr(\texttt{VBZ } \underline{\texttt{VP}} \mid \texttt{added/VBN}, \texttt{VP}) \\ &\cdot \Pr(\underline{\texttt{VBN}}\texttt{ NP PP} \mid \texttt{added/VBN}, \texttt{VP}) \cdot \Pr(\text{——} \mid \texttt{added/VBN}, \texttt{VBN}) \qquad (6.10)\end{aligned}$$

which we have not attempted to replicate (amusingly, since the above authors attribute the idea to (Eisner, 1996b)). Such a strategy of conditioning on the *tagged* headword, which among other things allows better backoff in the competing models, would probably also help the transformation model. For example, the transformation model learns that it is good to insert a modal before an untensed auxiliary verb VB (be slowing $\Rightarrow$ may be slowing), but it cannot generalize this to an untensed main verb VB because that tag does not appear on the main verb in our dataset. Taking advantage of headword part-of-speech tags for either model would require an augmented training set that mentioned those tags.

[30] Most dramatically, the process is capable of modeling the (linguistically useless) set of frames $\texttt{S} \to \texttt{A}^n$ —— $\texttt{B}^n$, although the fact that only the head child is ever rewritten—so rewrites must be nested—prevents it from describing arbitrary context-free languages, such as the Dyck language of matched parentheses.

correspond to the states of Alshawi's automaton for that headword.

For these reasons, and for comparison with previous literature, it seemed worth testing such articulated models:

training Each model in §6.6.1 was used to train probabilities like those in equation (6.12). This used the articulated training data in Fig. 6.7. (Thereby giving these models an unfair advantage in the form of additional supervision. Unlike the flat models, they had access to human annotation of the frames' internal structure. This indicated that (e.g.) `VP` was a natural class, and gave examples.)

development The smoothing parameters (§6.6.2) were set to maximize the cross-entropy of a similarly articulated version of the development set.

testing In testing, each frame projection probability such as Pr(`PP VBZ` —— `NP PP` | `added`, `S`) was computed as the *total* probability of all parses of the RHS. Equation (6.12) is only the probability of the single parse that agrees with the Treebank.[31]

The parsing procedure did consider rules it had never seen in training, evaluating them by the appropriate model in §6.6.1. However, for speed, the procedure did not consider parses in which X had a head child $\underline{Y}$ that had never served in training data as a head child of X. It did consider parses with nested unary rules, but usually not those with cycles of unary rules (e.g., $X \to \underline{Y} \to \underline{X}$).

6.6.4 Backed-Off Memorization Models

As we will see in §6.7.1, the transformation model performs rather better than the models shown so far. One explanation is that it does a better job of "memorizing" lexical entries that have been observed already, i.e., recognizing them as units to be awarded high probabilities. This is possible because of the arcs to observed frames and the parameters that are used to tune individual lexical entries. By contrast, the bigram model cannot throw

[31] Empirically, evaluating just the single parse that agrees with the Treebank turns out to be a poor choice. For example, replacing `VBN` (past participle) with `VBG` (present participle) in equation (6.12) yields another legal parse. The second parse has about the same probability as the first, if the model has not seen `added` before and does not know its correct part of speech. So counting both will double the estimated probability.

	frame		
headword	LHS		RHS
big	JJ	→	——
indexer	NP	→	$\mathtt{JJ}\ [_{\mathtt{NN}}\ \text{——}\ ]\ \mathtt{NPR}$
Bankers	NNP	→	——
Trust	NNP	→	——
Co.	NPR	→	$\mathtt{NNP\ NNP}\ [_{\mathtt{NNP}}\ \text{——}\ ]$
also	ADVP	→	$[_{\mathtt{RB}}\ \text{——}\ ]$
uses	S	→	$\mathtt{NP\ ADVP}\ [_{\mathtt{VP}}\ [_{\mathtt{VBZ}}\ \text{——}\ ]\ \mathtt{NP\ PP\text{-}LOC}\ ]\ .$
futures	NP	→	$[_{\mathtt{NNS}}\ \text{——}\ ]$
in	PP-LOC	→	$[_{\mathtt{IN}}\ \text{——}\ ]\ \mathtt{NP}$
a	DT	→	——
strategy	NP	→	$[_{\mathtt{NP}}\ \mathtt{DT}\ [_{\mathtt{NN}}\ \text{——}\ ]\ ]\ \mathtt{SBAR}$
that	SBAR	→	$[_{\mathtt{WHNP}}\ [_{\mathtt{IN}}\ \text{——}\ ]\ ]\ \mathtt{S}$
on	PP	→	$[_{\mathtt{IN}}\ \text{——}\ ]\ \mathtt{NP}$
average	NP	→	$[_{\mathtt{NN}}\ \text{——}\ ]$
has	VBZ	→	——
added	S	→	$\mathtt{PP}\ [_{\mathtt{VP}}\ \mathtt{VBZ}\ [_{\mathtt{VP}}\ [_{\mathtt{VBN}}\ \text{——}\ ]\ \mathtt{NP\ PP}\ ]\ ]$
one	QP	→	$[_{\mathtt{CD}}\ \text{——}\ ]$
percentage	NN	→	——
point	NP	→	$\mathtt{QP\ NN}\ [_{\mathtt{NN}}\ \text{——}\ ]$
to	PP	→	$[_{\mathtt{TO}}\ \text{——}\ ]\ \mathtt{NP}$
its	PRP\$	→	——
enhanced		→	JJ ——
fund	NP	→	$\mathtt{PRP\$\ JJ}\ [_{\mathtt{NN}}\ \text{——}\ ]$
’s	NP\$	→	$\mathtt{NP}\ [_{\mathtt{POS}}\ \text{——}\ ]$
returns	NP	→	$\mathtt{NP\$}\ [_{\mathtt{NNS}}\ \text{——}\ ]$
.	.	→	——

Figure 6.7: A version of Fig. 6.3 if frame-internal brackets are retained at step 8 of data preparation (§6.2).

probability mass at an observed lexical entry without splashing heavily onto other entries that contain many of the same bigrams.

To test this hypothesis, we can try a hybrid approach that starts with the memorization model of §6.6.1.1 but backs off to a lower-order Markov model, especially for rare words. Like the transformation model, this combines the abilities to memorize and generalize. It assigns positive probability to all possible frames. Carroll and Rooth (1998) used a similar technique for evaluation purposes, backing off to a poor "spelling model."

Two backoff techniques were implemented. Each technique discounts the maximum-likelihood ("memorized") probabilities of lexical entries with low counts, and reallocates their probability mass to a lower-order Markov model.

The first technique is Katz backoff (Katz, 1987), which applies the Good-Turing discounting formula to counts ≤ 5.

The second technique is one-count smoothing, with a backoff scheme as sketched in Fig. 6.6. For reasons described in §6.6.2, it may be interpreted as follows. We try to estimate $\Pr(\text{RHS} \mid \text{headword}, \text{LHS})$. To the extent that the word (or LHS) is rare, then we will back off to a weighted average. $\Pr_{\text{mem}}(\text{RHS} \mid \text{LHS})$ and $\Pr_{\text{lower-order}}(\text{RHS} \mid \text{headword}, \text{LHS})$. (The weight in this average depends only on LHS.) All of the smoothing weights were estimated simultaneously, so that the Markov model was smoothed specifically to perform well on rare words.

6.7 Results and Analysis

The empirical comparison should be regarded as a proof of principle. Once the bigram model was determined to be the best from previous literature, the transformation model's features (§6.4.4), including "side," were carefully designed to use only as much information as the bigram model. No feature could depend on more than two nonterminals. There were no specially linguistic features, and no output features (§3.6.2) that judged a transformation's probability by its output's well-formedness (or probability under another model).

At most, the feature weights could only indicate whether it would be good to transform

	basic		memorization+backoff		
			Katz	one-count	
	flat	non-flat	flat	flat	non-flat
1-gram	1774.9	86435.1	340.9	160.0	193.2
2-gram	**135.2**	199.3	127.2	**116.2**	174.7
3-gram	136.5	177.4	132.7	123.3	174.8
subcat	363.0	494.5	197.9		
transf	**108.6**				
combined	**102.3**				

Table 6.1: Perplexity of the test set of $\mathtt{S} \to \cdots$ entries under various models.

an S entry so that two given nonterminal children would (or would no longer) be adjacent on a particular side of the head.

So the comparison aimed to see whether the transformation model could match the bigram model, using the same kind of information in a very different way.

Of course one putative strength of transformation modeling is that it could incorporate more informative features, as well as more linguistically interesting transformations and representations (§2.4.3, §5.5). However, before doing such work, it is necessary to develop good estimation methods and show that transformation models are not sunk by their own complexity even on simple cases.

6.7.1 Basic Comparison of Models

The perplexity of the test set under various models is shown in Table 6.1. The smallest (best) perplexities appear in boldface. The key results:

- The transformation model was the winner, reducing perplexity by 20% over the best model replicated from previous literature (a bigram model).
- Much of this improvement could be explained by the transformation model's ability to model exceptions (§6.6.4). Adding this ability more directly to the bigram model reduced perplexity by 6% or 14%, depending on whether Katz or one-count backoff was used, as compared to the transformation model's 20%.
- Averaging the transformation model with the best competing model improves it by

an additional 6%.[32] This gives a total perplexity reduction of 24% over the best previous model from the literature and 12% over the best new model that does not use transformations.

- It is much better to predict flat frames directly than by summing over their possible articulated structures.
- One-count smoothing works better than Katz smoothing when backing off from the memorization model. (Katz appears to discount the observations too heavily.)

A useful qualitative comparison of the transformation and bigram models can be found in §1.5.3.

6.7.2 Consistency of the Improvement

Fig. 6.8 (analogous to Fig. 1.4) shows that averaging the transformation model with the memorization+bigram model improves the latter not merely on balance, but across the board. In other words, there is no evident class of phenomena for which incorporating transformations would be a bad idea.

- Transformations particularly helped on low-probability novel frames, as hoped.
- Transformations also helped on frames that were observed once in training with relatively infrequent words. (In other words, the transformation model does less backoff from singletons.)
- Transformations hurt slightly on balance for frames observed more than once, but the effect was tiny.

Of course, all these differences are slightly exaggerated if one compares the transformation model directly with the memorization+bigram model, without averaging.

[32] Model averaging is a common technique for exploiting the strengths of two probability distributions $\Pr_1$ and $\Pr_2$, each of which underestimates some probabilities. It defines a new distribution $\Pr(x) = (\Pr_1(x) + \Pr_2(x))/2$. This can be regarded as a mixture model in which each sample is drawn from either $\Pr_1$ or $\Pr_2$ according to a coin flip, so that x is fairly likely if either distribution deems it likely.

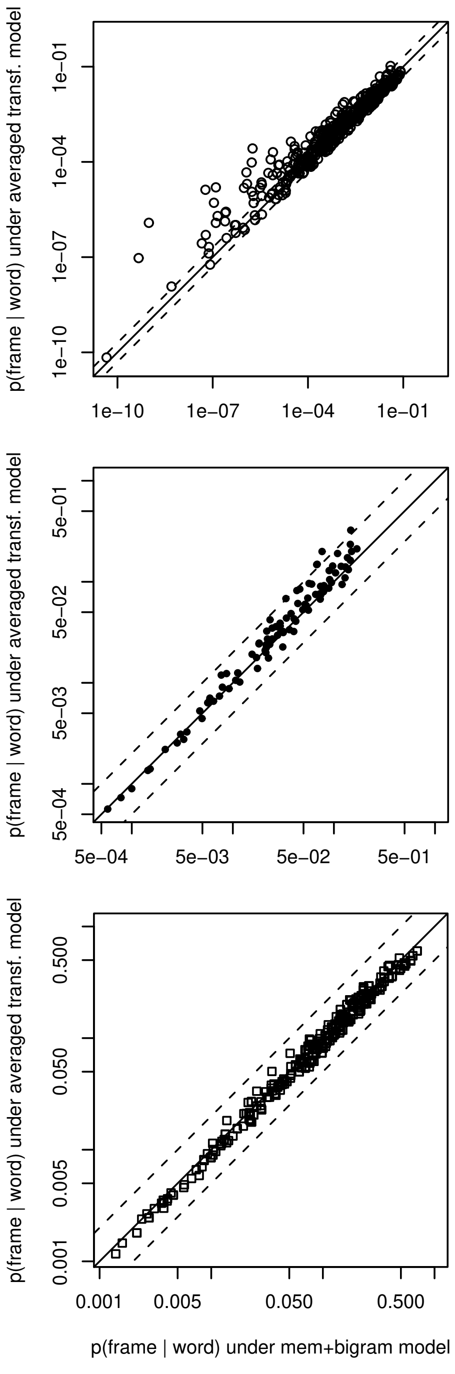


Figure 6.8: Probabilities of test set lexical entries under the averaged model, plotted against the corresponding probabilities under the best transformation-free model. Improvements fall above the main diagonal; dashed diagonals indicate a factor of two. The three plots (at different scales!) partition the entries by the number of training observations: $\circ = 0$, $\bullet = 1$, $\square \geq 2$.

	basic		memorization+backoff		
			Katz	one-count	
	flat	non-flat	flat	flat	non-flat
1-gram	1991.2	96318.8	455.1	194.3	233.1
2-gram	**162.2**	236.6	153.2	**138.8**	205.6
3-gram	161.9	211.0	156.8	145.7	208.1
subcat	414.5	589.4	242.0		
transf	**124.8**				
combined	**118.0**				

Table 6.2: A version of Table 6.1 when the amount of training data is reduced by half.

6.7.3 Annotation Cost

What would be the cost of achieving such a perplexity improvement by additional annotation? Training the transformation model on only the first half of the training set (Treebank sections 0–7), with no further tuning of any options, yielded a test set perplexity of 124.8. Similarly training an averaged model yielded 118.0.

So by using transformations, we can achieve about the same perplexity (118.0) as the best model without transformations (116.2), *even using only half as much training data.*

6.7.4 Graceful Degradation

Results for all models trained on only sections 0–7 are shown in Table 6.2. In general, Tables 6.1 and 6.2 have the same pattern of results.

On this halved training set, the perplexity of the bigram model and its memorization variant increase by 20%. (The inferior models also increase by about this much, except for unigrams without memorization, which can be well-estimated from a small sample, and the basic subcategorization model.) But the perplexity of the transformation model and the averaged model increase by only 15%. This suggests that the transformation model generalizes better from sparse data, as hoped.

As a result, on the smaller training set, the advantage of using transformations is increased by another 3 percentage points. Thus, its improvements over the bigram model and its memorization variant rise to 23% and 10% respectively (as compared with 20%

and 7% on the full training set). The averaged model has corresponding improvements of 27% and 15% (as compared with 24% and 12%).

These results are encouraging for the use of transformational smoothing in language learning. In EM-style language learning, the training data are much noisier than here, because the lexical entries must be inferred by parsing strings badly. §1.2.4.2 and §8.2.4 suggested using a thresholded version of EM that reestimates parameters from only the most confidently labeled data. In this case the data are less noisy, but sparser. The claim of §1.2.4.3 was that transformational smoothing might be useful in extrapolating from such sparse data.

6.7.5 Type-Weighting

Besides its ability to memorize, the transformation model was supposed to have other desirable properties. §3.8.7 hinted that it should not be led astray by common but atypical types such as function words (see §7.1.5 for full details).

To test this proposition, we considered the set of `SBAR` $\rightarrow \cdots$ frames. In our dataset these happen to form a heterogeneous set with two types of lexical entries, exemplified by `SBAR` $\rightarrow$ `that S` and `SBAR` $\rightarrow$ `NP MD remain PP-LOC`. The former type is headed by a closed-class complementizer `that`, `which`, or a question word. The latter type is headed by a lexical verb such as `remain`, and has the same form as an `S` $\rightarrow \cdots$ frame from the previous experiment.[33]

This dataset is challenging for backoff smoothing methods, including the new memorization model with one-count backoff. A poorly observed word w cannot get a good direct estimate of $\Pr(RHS \mid w, \texttt{SBAR})$. It must back off to the distribution $\Pr(RHS \mid \texttt{SBAR})$. But the latter distribution is heavily influenced by the first type of lexical entry and so predicts an RHS of —— `S` with probability $\frac{2}{3}$. The trouble is that a poorly observed word is likely to be an open-class word, not a lexical complementizer. The backoff distribution is helpful for predicting the frame of new word tokens, but not new word types.

By contrast, the transformation model should do better at using just a few perturbation

[33]The latter type began life with a null complementizer as `SBAR` $\rightarrow \emptyset$ `S`. Since our dataset does not allow null headwords, the data-preparation scripts were forced to take `S` as the head child. Therefore the `SBAR` inherited the headword and dependent slots of the `S`.

parameters to model the common lexical entries as exceptions. For example, it can learn that $\textsc{Start}_{\texttt{that}}$ has an especially high-probability arc to `SBAR` $\rightarrow$ `that S`. A word that is not known to be exceptional will be governed by more regular processes.

As predicted, the transformation model does do substantially better on this dataset. The bigram model achieves a perplexity of 8.3 (i.e., the average SBAR's RHS can be encoded with about half as many bits as the average S's RHS). The memorization model with one-count backoff to the bigram model only reduces the perplexity by 4% (to 8.0); after all, the bigram model can do a good job of learning —— `S`. But the transformation model reduces perplexity by 29% (to 5.9). So the margin of victory over the best competing model is substantially greater here.

This is true although the transformation model was trained with the same options and priors as before, with no further tuning to the development set. This presumably put it at a disadvantage against the competing models, whose smoothing parameters were chosen afresh to minimize perplexity on `SBAR` frames in the development set.

6.7.6 Generalization: The Forced-Match Task

Finally, the transformation model was supposed to be able to generalize better from the frames that are known for a word to new frames, which are plentiful in test data (Fig. 6.4). To test its success at this, we compared it to the bigram model on a pseudo-disambiguation task.

Each instance of the task consisted of a pair of lexical entries from test data, (w_1, f_1) and (w_2, f_2), such that f_1 and f_2 are "novel" frames that did *not* appear in training data (with any headword).

Each model was then asked: Does f_1 go with w_1 and f_2 with w_2, or is it the other way around? In other words, which is bigger, $\Pr(f_1 \mid w_1) \cdot \Pr(f_2 \mid w_2)$ or $\Pr(f_2 \mid w_1) \cdot \Pr(f_1 \mid w_2)$?

Since the frames were novel, the model had to make the choice according to whether f_1 or f_2 looked more like the frames that had actually been observed with w_1 in the past, and likewise w_2. What this means depends on the model. The bigram model takes two frames to look alike if they contain many bigrams in common. The transformation model takes two frames to look alike if they are connected by a path of probable transformations.

The test data contained 62 distinct lexical entries (w, f) in which f was a novel frame. This yielded $\frac{62 \cdot 61}{2} = 1891$ pairs of lexical entries, leading to 1811 task instances after obvious ties were discarded.[34]

Baseline performance on this difficult task is 50% (random guess). The bigram model chose correctly in 1595 of the 1811 instances (88.1%). Memorization does not help on this task, as it involves only novel frames, so the memorization+bigram model had the same performance.[35] By contrast the transformation model got 1669 of 1811 correct (92.2%), for a more-than-34% reduction in error rate.

Since the 1811 task instances were derived non-independently from just 62 novel lexical items, this result is really based on a small sample. However, some additional support for the transformation model is provided by the similar results on the development set rather than the (smaller) test set.[36] Here the bigram and transformation models got 8592 (82.1%) and 9184 (87.7%) of 10474 forced matches correct, meaning that the transformation model reduced error rate by over 31%.

6.7.7 Learning Curves and Weight Distributions

Let us close with a look inside the optimization. The long dashed line in Fig. 6.9 shows the continual improvement in the objective function $\tilde{g}$ for a typical run of optimization. (The function has been rescaled to units of bits-per-training-frame, so *smaller* values are better.)

In this run, $\vec{\theta}$ was initialized to 0 (except for the weights of the special features in §6.4.5), while perturbations $\vec{\pi}$ were initialized as in §6.5.2. The first 150 iterations of gradient descent are shown, enough to surpass the performance of the bigram model though not enough to converge. The lower and upper solid lines show cross-entropy on training and test data, respectively. As the optimization proceeds, performance on test data improves,

[34] An obvious tie is an instance where $f_1 = f_2$, or where both w_1 and w_2 were novel headwords. (The 62 lexical entries included 11 with novel headwords.) In these cases, neither the bigram nor the transformation model has any basis for making a forced-match decision: the probabilities being compared will necessarily be equal.

[35] To be precise, it got 1596 correct (still 88.1%). The trivial difference in performance can be attributed to the fact that headwords vary in how strongly they back off from memorization to the bigram model.

[36] The development set contained 147 rather than 62 lexical entries not seen in training, yielding more than five times as many task instances. The transformation model had been tuned on the development set to obtain good perplexity, but had not been tuned for the forced-match task.

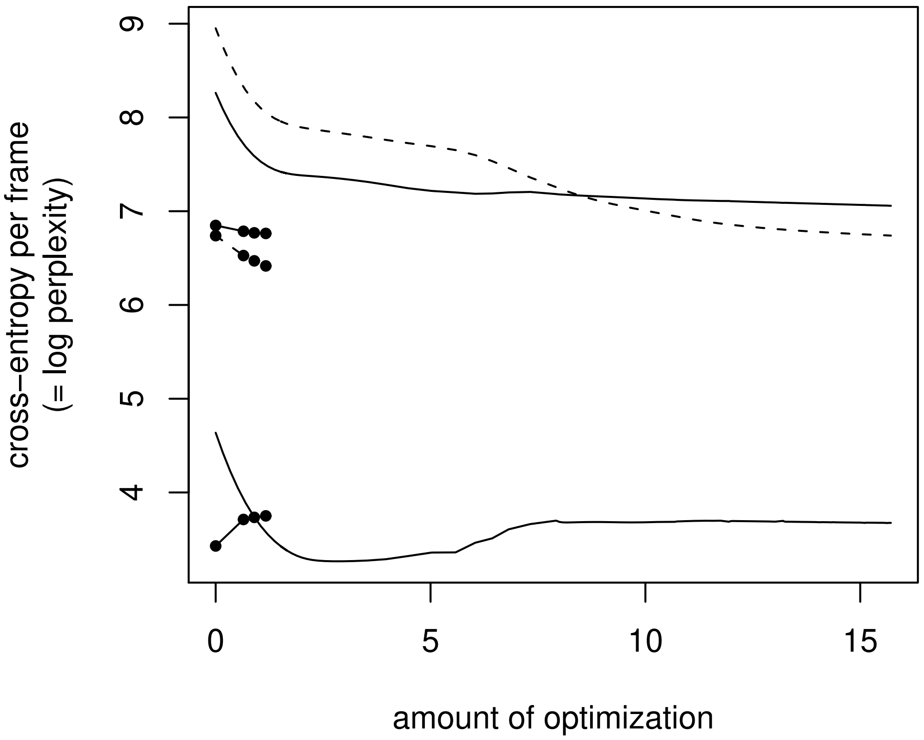


Figure 6.9: Learning curves during adaptive gradient descent (smaller y-values are better). Long and short runs correspond to different initialization schemes. Solid lines show cross-entropy per frame on training data (lower line) and test data (upper line). The dashed line represents the objective function: it is like the training line but adds in the cost of the prior, amortized over all training frames.

somewhat at the expense of training data—a smoothing effect, although as already noted, the model continues to assign rather high probability to training data.

Fig. 6.9 also displays such curves for a different run, identical except that this time $\vec{\theta}$ as well as $\vec{\pi}$ was initialized as described in §6.5.2. This was in fact the run reported in Table 6.1. The initial model was much better than the previous run's, although it did not have much room to improve locally: it was stopped after only 3 passes (based on its performance on held-out data, where after 3 passes it began to overfit training data).

The x-axis in the figure is not the number of steps of the gradient descent algorithm, but rather the cumulative length of the algorithm's trajectory in parameter space. This yields a smoother graph because the algorithm is adaptive, varying its stepsize from pass to pass.

Fig. 6.10 shows the distribution of weight magnitudes at the point where each of the two runs was stopped. There are about 70000 non-zero weights—including 2482 output-feature weights used on arcs from START_ℓ, which do not describe transformations. (The 11274 perturbation parameters are not shown here.)

For a weight to have become non-zero, it must have had some effect on the objective function. Specifically, it must correspond to a feature of an arc on a path that generates an observed lexical entry, or on an arc that competes with probability for such an arc. The competitor arcs are one common source of negative weights, so only about 11% of weights are positive, though these tend to be larger.

A few hundred weights are strongly positive or negative; all but a few thousand weights are negligible. For the long run whose transformational features' weights were initialized to 0, only 3109 of the 70000 weights end up with absolute value > 0.1, and this includes many (non-transformational) weights that were initialized that way.[37] For the shorter run, 12282 weights end up above 0.1. Notice that a single feature of weight ± 0.1 suffices to alter an arc's probability by about $\pm 10\%$.

One might wonder whether small weights gang up for a substantive combined effect. The answer turns out to be no.[38] As Fig. 6.10 shows, zeroing out *all* the small weights

[37]The long run has trouble escaping an apparent local maximum in which there is little transformational action. Transformational features having been initialized to 0, gradient descent preserves a rather sharp distinction between strong output features and weak transformational features.

[38]Remember that a given arc has only 22 features at most. This does not provide much opportunity for

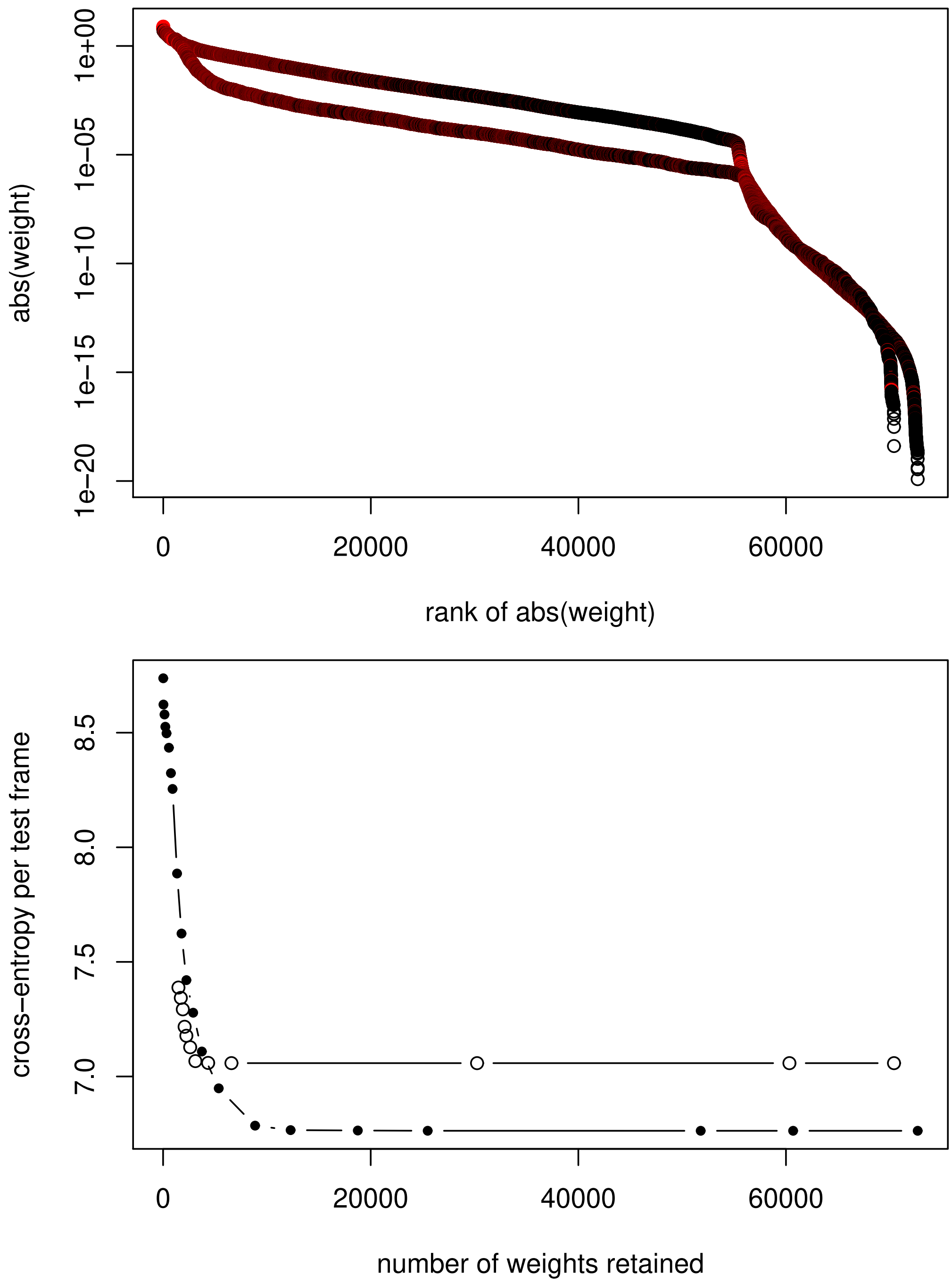


Figure 6.10: *Top:* Distribution of weight magnitudes at the end of each run from Fig. 6.9; the lower curve is for the long run. Positive weights are shown in red, negative ones in black. *Bottom:* Effect on cross-entropy of zeroing out the small weights. Here the white curve is for the long run.

(without retraining the others or the perturbations) has no discernible effect on the resulting distribution, or at least on its perplexity. This suggests that it may be possible to speed up the system by eliminating low-weight features. One must work up to zeroing out all weights of magnitude 0.1 or 0.2 before one sees even a small effect.

---

gang effects: the sum of 22 tiny weights would still be tiny.

Chapter 7

Variations and Applications of Transformation Models

Chapter 3 gave an introduction to transformation models as abstract devices. In this chapter, we return to that viewpoint and flesh it out a bit more with additional connections, applications, and variations.

7.1 Connections to Other Techniques

Transformation models have resemblances to other familiar mathematical devices, and to a Bayesian smoothing technique in the literature.

7.1.1 An Interpretation as a Markov Process

A transformation model may be interpreted as a Markov process whose state set is $\mathsf{Events} \times \{0,1\}$. A transition to $\langle e, 0\rangle$ corresponds to generating the event e in the random walk of §3.3.1. A transition from $\langle e, 0\rangle$ to the absorbing state $\langle e, 1\rangle$ corresponds to halting at e.

Formally, write $P_\theta(e, e')$ to abbreviate $\sum_{A=\langle e,e',F\rangle\in\mathsf{Arcs}} P_\theta(A)$. Then the transition probabilities are given as follows, where $e, e' \in \mathsf{Events}$:

- The transition probability from $\langle e, 0\rangle$ to $\langle e', 0\rangle$ is $P_\theta(e, e')$.

- The transition probability from $\langle e, 0\rangle$ to $\langle e, 1\rangle$ is $P_\theta(e, \textsc{Halt})$.

- The transition probability from $\langle e, 1\rangle$ to $\langle e, 1\rangle$ is 1. So $\langle e, 1\rangle$ is an absorbing state.
- All other transition probabilities are 0.

The initial distribution of the Markov process assigns probability 1 to $\langle \textsc{Start}, 0\rangle$ and probability 0 to all other states. If p_{lim} is the corresponding limit distribution of the process, then we define $\Pr_\theta(e) = p_{\text{lim}}(\langle e, 1\rangle)$. Footnote 5 on p. 85 briefly discussed when this distribution sums to 1 (that is, $(\forall e) p_{\text{lim}}(\langle e, 0\rangle) = 0$).

Note that the Markov process is non-ergodic because of the absorbing states $\langle e, 1\rangle$. So the limit distribution $\Pr_\theta(e)$ is sensitive to the initial distribution (which places all the probability mass at Start). Indeed, it is sensitive to the distribution on the following step: so the model's preference for choosing certain initial events e_1 to transform (§3.3.2) is reflected in the events that tend to emerge from the sequence of transformations.

7.1.2 An Interpretation as a Probabilistic Finite-State Machine

Another useful intuition about transformation models comes from the world of automata theory. Probabilistic finite-state machines have been the subject of much recent interest (e.g., (Mohri, 1997)). They are ordinary finite-state machines augmented with probabilistic information: each state specifies a probability distribution over the arcs leaving the state.

It is helpful to regard a probabilistic finite-state automaton as *writing* the labels of the arcs it traverses, rather than reading them. Such a machine may be regarded as stochastically choosing a string.[1]

A path in such an automaton from the start state to a final state, whose arcs are respectively labeled $\alpha_1, \alpha_2, \ldots \alpha_n$, is said to output the string $\alpha_1\alpha_2\cdots\alpha_n$. The probability of the path is the product of the probabilities of all its arcs (just as in a transformation model).[2] One is usually interested in the probability of generating a particular string α—i.e., the total probability of all paths that output that string.

Given a transformation model and parameter vector θ, the distribution $\Pr_\theta$ over Events

[1] Or equivalently, stochastically transducing the null string ϵ to some output.

[2] In so-called subsequential machines, this product also includes the **halt probability** of the final state. (The probability distribution specified by a state is modified to be a distribution over all the state's "exit options"—its out-arcs and, if the state is final, the option of halting.) But if ϵ arcs are allowed, as in our case, then introducing a Halt state lets us avoid this complication without loss of generality.

can be described using a probabilistic finite-state automaton, provided that Events is finite. The transformation graph described in §3.2.1 provides the automaton's topology (states and arcs). START is the start state, and HALT is the only final state. Any arc from a state e to HALT is labeled with e; all other arcs are labeled with ϵ. Finally, the probability of an arc A is given by $P_\theta(A)$.

Now the probability $\Pr_\theta(e)$ is defined to be the probability of generating the length-1 string e.

Eisner (2001) presents a general scheme for parameter estimation in finite-state recognizers and transducers, including those with ϵ-cycles. The method given here in §8.2 is a special case of that general scheme. In particular, an implementation of the general scheme could be applied to estimate the parameters of transformation models. Conversely, nearly all the algorithmic tricks given here could be applied in the general case of finite-state machines. See §8.2.3 and §8.3.1 for further discussion.

If Events is infinite, then a transformation model corresponds to an *infinite*-state machine. §4.2 showed how to approximate the necessary computations by exploring a finite portion of the automaton graph "on the fly" as necessary Sproat (1996, §6) calls this architecture a "Generalized State Machine."

7.1.3 An Interpretation as a Recurrent Neural Network

Like any Markov process, a transformation model may be interpreted as a possibly recurrent neural network. We will exploit this connection in one of our algorithms for training transformation models (§8.5.5), which computes the gradient in a way that resembles back-propagation.

The units, connections, and weights of the network correspond to the states, transitions, and transition probabilities of the Markov process described above (§7.1.1). The activation squashing function is the identity function rather than a sigmoidal threshold function.

The network always has the same input vector: the START node receives activation of 1, and no other node receives input. On each timestep, each unit's activation is propagated along its outgoing connections (scaled by the connection weights), and is removed from the original unit. In the limit, the network settles. $\Pr_\theta(e)$ is the limit activation of the unit $\langle e, 1\rangle$.

It is worth remarking that recurrent networks that settle are often used to implement associative memories that can complete an input activation vector (Hopfield, 1982). One could imagine using this approach to smooth a syntactic lexicon. The input activation vector would specify the observed probabilities of the lexical entries, and the network would be allowed to settle into a smoothed version. However, it is doubtful that this approach can be made to work.[3]

7.1.4 An Interpretation as a Cyclic Graphical Model

A transformation model can also be regarded, somewhat awkwardly, as a kind of directed graphical model (Pearl, 1988). Every vertex e in the transformation represents a real variable $\Pr(e)$. More precisely, it represents the real variable $I(e)$, defined in §3.4; this is proportional to $\Pr(e)$.[4] (Most graphical models are defined over boolean variables, but the formalism allows any kind of variable.) What a graphical model does is to define a joint prior distribution over these real variables. In this case, that amounts to a distribution over distributions (see §3.1.2).

The transformation graph makes conditional independence assertions about the individual variables, just as a graphical model does. The probability of e depends only on the probabilities of e's parents. Given the latter, knowing other probabilities in the graph cannot tell us anything about e (their conditional mutual information with e is zero).

Of course, the probability at e also depends on components of θ. So to turn the transformation model into a graphical model, we must give e some new parents. We add each θ_i as a new vertex and add links from these nodes to the original vertices as necessary.

This graphical model now has an odd property. The vertices θ_i are ordinary free

[3] If the idea is to run the model on different inputs (corresponding to different words, say) and get different outputs, then it is problematic that only a handful of outputs are possible (the fixed points; indeed, to get multiple fixed points the model must be made nonlinear). This is appropriate for a memory but not for a smoother. It is also unclear what the objective function should be for training the network, or the balance between representing exceptions in the network vs. in the input. Finally, this approach would require a different style of transformation graph, one in which transitions to HALT are not used and recurrent transitions become crucial.

[4] It has to represent $I(e)$ rather than $\Pr(e)$ because $\Pr(e)$ can always be recovered from $I(e)$ (by equation (3.20)), but not vice-versa (since if there is no arc from e to HALT then solving equation (3.20) reduces to $0/0$).

parameters; the prior of §3.5 says that they are normally distributed random variables with variance σ^2. But each $I(e)$—and hence each $\Pr(e)$—is merely a deterministic function of its parents. $\Pr(e)$ has no additional variance; its conditional distributions are spikes. This is legal for graphical models but highly unusual.

A *perturbed* transformation model, though, is not unusual in this way: there *is* additional variance at $I(e)$, provided by the perturbation π_e. Unlike θ_i, π_e is not a vertex of the graphical model. The usual graphical model property holds: the node e is conditionally independent of its parents. That is, the prior probability that $I(e)$ will have a particular value depends only on θ and on the values $\{I(e') : e' \text{ is a parent of } e\}$. Specifically, under the assumption that $\pi_e \sim N(0, \sigma^2)$, equation (3.28) says that $I(e)$ is distributed *log-normally* about a linear combination of those values: namely,

$$\sum_{A=\langle e', e, F\rangle \in \mathsf{Arcs}} I(e') \cdot p(A) + \delta(e = \textsc{Start})$$

A more important hitch in the analogy to graphical models is that our graphical model may inherit cycles from the transformation graph. Cycles are not standardly allowed in directed graphical models, because they may prevent the existence of any joint distribution that is consistent with all the conditional independence assumptions. Happily, however, this is not a problem for our model. The defintion in terms of path probabilities or random walks means that a unique consistent solution is guaranteed, as computed in §3.4.

7.1.5 Bayesian Backoff

Some of the desirable properties of transformational smoothing arise from the fact that it is a Bayesian method. It is influenced not only by *what* evidence is available but also by *how much.* The prior and the various pieces of evidence all put pressure on the parameters, so weak evidence is likely to be overruled (smoothed) to appease a coalition of strong evidence and the prior.

Other Bayesian methods have this property as well. For example, one of the nice properties of simple Bayesian backoff (MacKay and Peto, 1995) is that it treats frequently observed phenomena as trustworthy but not necessarily typical. This makes it a sensible compromise between two non-Bayesian techniques for choosing a backoff estimate, namely, averaging related estimates by type and by token. This section explains that comment and shows that it also applies to transformational smoothing, in particular the example model discussed throughout §3.8.

7.1.5.1 The Bigram Smoothing Task

Given a word w_j, suppose one wishes to smooth the distribution $\Pr(w_{j+1} \mid w_j)$, which predicts the word w_{j+1} that will follow w_j. The traditional approach[5] interpolates between the maximum-likelihood estimate $\Pr_{\text{ML}}(w_{j+1} \mid w_j)$, which is unreliable when there are few observed samples of the distribution (i.e., for infrequent w_j), and a generic backoff estimate $\Pr_{\text{backoff}}(w_{j+1})$:

$$\hat{\Pr}(w_{j+1} \mid w_j) \stackrel{\text{def}}{=} \alpha_{w_j} \Pr_{\text{ML}}(w_{j+1} \mid w_j) + (1 - \alpha_{w_j}) \Pr_{\text{backoff}}(w_{j+1}) \tag{7.1}$$

The interpolation parameter α_{w_j} depends on how well the distribution is observed: the more infrequent the conditioning word w_j, the more strongly the backoff estimate is weighted.

7.1.5.2 Token-Weighted Bigram Smoothing

In the traditional approach to bigram smoothing, the generic backoff estimate can be regarded as a weighted average over all context words w_j:

$$\Pr_{\text{backoff}}(w_{j+1}) \stackrel{\text{def}}{=} \Pr_{\text{ML}}(w_{j+1}) = \sum_{w_j} \Pr_{\text{ML}}(w_{j+1} \mid w_j) \cdot \Pr_{\text{ML}}(w_j) \tag{7.2}$$

The weighting factor $\Pr(w_j)$ indicates that it is a **token-weighted average**, i.e., more frequent words have more influence on the backoff estimate.

7.1.5.3 Type-Weighted Bigram Smoothing

This strategy is called into question by Baayen and Sproat (1996), who show (for a somewhat different problem) that for the rare words w_j where the backoff estimate actually matters, it is better to use a backoff estimate based only on other rare words. Baayen

[5]Encompassing both Jelinek-Mercer smoothing (Jelinek and Mercer, 1980; Bahl et al., 1983) and simple additive smoothing. Common alternatives such as Katz backoff (Katz, 1987) and similarity-based smoothing (Dagan et al., 1994) use the backoff estimate only when the maximum-likelihood estimate is 0.

and Sproat do not try using the **type-weighted average** where all words (both rare and frequent) are weighted equally,

$$\Pr_{\text{backoff}}(w_{j+1}) = \sum_{w_j} \Pr_{\text{ML}}(w_{j+1} \mid w_j) \cdot \frac{1}{n} \tag{7.3}$$

(n being the number of summands). However, this would presumably be similar to Baayen and Sproat's backoff estimate, since rare words tend to greatly outnumber frequent words when counted by type (Zipf's law).

7.1.5.4 Bayesian Bigram Smoothing

MacKay and Peto's approach can also be described as an instance of equation (7.1). It is derived by treating the words observed to follow each w_j as samples from the distribution $\Pr(\cdot \mid w_j)$, and by also treating *all* the distributions $\Pr(\cdot \mid w'_j)$ as vector-valued samples from a single Dirichlet function. The Dirichlet is itself a distribution (over distributions) whose parameters must also be estimated.

$\Pr_{\text{backoff}}(\cdot)$ emerges as the mean of the Dirichlet. If the Dirichlet has low variance, then each conditional distribution $\Pr(\cdot \mid w_j)$ is expected *a priori* to be close to this mean, and will therefore be smoothed aggressively toward it. But to the extent that a given w_j is well-observed, the evidence about w_j may overwhelm this prior expectation, so that $\Pr(\cdot \mid w_j)$ is modeled as exceptional. This is the same Bayesian idea discussed in §3.8.5.

What determines $\Pr_{\text{backoff}}$ in the MacKay and Peto model? Unlike the token-weighted average $\Pr_{\text{ML}}(w_{j+1})$ used by traditional backoff, the Dirichlet mean is effectively estimated as a type-weighted average. Each of the distributions $\Pr(\cdot \mid w_j)$ counts as a single sample from the same Dirichlet, without regard to how frequent w_j is. So the Dirichlet mean (and variance) must be chosen so that all these samples are plausible.

However, this type-weighted average is complicated by the fact that in practice it is an average of *unknown* distributions. Each word-specific distribution $\Pr(\cdot \mid w_j)$ in the average is "fuzzy" in the sense that it is not known precisely: it is being estimated along with the average. If w_j is rarely observed, then there is considerable "give" or uncertainty in that estimate. Indeed, it is exactly this uncertainty that smoothing resolves by expecting *a priori* that word-specific distributions tend to be close to the average. As an extreme case, if w_j has never been observed at all, then $\Pr(\cdot \mid w_j)$ is wholly underspecified, and is estimated as equal to the background distribution—its most likely value. It provides no information that can influence the background distribution; rather the influence is in the other direction.

So the MacKay-Peto model actually does grant a frequent word more influence than a rare word on the background distribution, but only because it asserts its preferences more confidently. As in traditional backoff smoothing, the flow of information is largely from frequent to rare words: frequent words help determine the behavior of the "typical" word, which in turn influences the estimated behavior of rare words.

The extra influence of frequent words in the MacKay-Peto model differs from their extra influence in the token-weighted average of traditional backoff. It does not increase linearly with the counts but rather faces diminishing returns once there are enough observations to allow a reasonably sharp probability estimate. (That is, a word with 100 observations has barely more influence than a word with 50, and because of type-weighting, both are outweighed by just a handful of words with one observation each.[6])

In short, the Bayesian approach combines the virtues of token-weighting and type-weighting. It correctly recognizes that frequent words tend to contribute more evidence about the language, without incorrectly supposing them to be more typical of words in general.

7.1.5.5 Bayesian Backoff Behavior in a Transformation Model

In the bigram smoothing example above, $\Pr(w_{j+1} \mid w_j)$ for each w_j is smoothed toward a common background distribution $\Pr_{\text{backoff}}(w_{j+1})$. Whether we use type-weighted, token-weighted, or Bayesian smoothing, poorly observed distributions are smoothed more aggressively.

[6] A concrete example: Let $\frac{k}{n}$ denote k observations of event A_i and $n-k$ observations of B_i. Suppose for $i = 1, 2, \ldots 8$ we observe $\frac{0}{1}, \frac{0}{1}, \frac{0}{1}, \frac{0}{1}, \frac{0}{1}, \frac{1}{1}, \frac{1}{1}, \frac{5000}{7000}$ respectively. The maximum-likelihood Dirichlet turns out to have a mean of $\langle 0.44, 0.56 \rangle$, indicating that the backoff distribution takes A events to be *less* likely than B events. So the several rare cases that total $\frac{2}{7}$ actually outweigh the frequent but atypical case $\frac{5000}{7000}$, which takes A events to be *more* likely. Nonetheless, the frequent case, being well-observed, does have more influence than in the straight type-weighted average, which would be $\langle 0.34, 0.66 \rangle$. (If the other cases were scaled up to be just as well-observed, so that sparse data were no longer a worry, the MacKay-Peto estimate would go to $\langle 0.31, 0.69 \rangle$, which is indeed in the ballpark of the type-weighted average though it is not quite the same thing.)

Recall the canonical transformation model of §3.8.1, which was discussed throughout §3.8. It must similarly smooth $(\Pr(A_i \mid i), \Pr(B_i \mid i))$ for each i toward a common background distribution $(\frac{2}{3}, \frac{1}{3})$. The transformation model is Bayesian, like the MacKay-Peto model for bigram smoothing. We now see (as §3.8.7 promised) that it strikes the same middle ground between type-weighting and token-weighting.

Stipulating an exception for any pair (A_i, B_i) is equally costly, regardless of how rarely A_i and B_i are observed. But if A_i and B_i are particularly rare, they may not insist as strongly on a particular exception. So rare event pairs have less influence on the transformation probability.

For example, the counts $\#(A_8) = 1, \#(B_8) = 0$ are consistent with many transformation probabilities, so this pair has little influence on θ_0 or other weights, and will be heavily smoothed. It would take many such $(1,0)$ pairs to establish that the transformation probability is small for the typical word.

But on the other hand, having many such pairs is in fact good evidence about the typical word, whereas a single well-observed pair (A_7, B_7) is not, even if it has more observations than all the other pairs combined. Indeed, if A_7 and B_7 are the only pair with an unusual probability ratio, then the optimal parameter vector θ will tend to use θ_0 to model the usual ratio and θ_{14} to model an exception for (A_7, B_7).

Multiplying all the evidence counts by a factor of 1000 (while keeping σ^2 constant) would ensure that all pairs (A_i, B_i)—even the *comparatively* infrequent ones—were supported by enough evidence that they would all insist, almost equally strongly, on having their observed ratios closely modeled. Such modeling would involve a type-weighted compromise for the parameter θ_0 that describes the typical case, together with enough other non-zero weights in $\theta_1, \ldots \theta_{2k}$ to capture how each pair has been observed to differ from this typical case. In other words, more evidence justifies more non-zero parameters.

7.1.5.6 How Does This Pattern Generalize to More Complex Models?

In a more complex transformation model, B_i might have several parents in the transformation graph, not just A_i and START. Then the model smooths probabilities so that each B_i is close to a linear combination of *all* its parents' probabilities. $(\Pr(B_i) \approx c \cdot \Pr(A_i)$ was a special case.)

The coefficients of the linear combination correspond to the "generic" probabilities of transforming the parents into B_i—that is, the probabilities determined by equations (3.13) and (3.15) if features peculiar to B_i are ignored. If B_i and its parents are well-observed, they have a stronger say in determining these generic coefficients. If they are not, their probabilities will be more aggressively smoothed by recourse to the coefficients.

If $\Pr(B_i)$ is exceptional and does not quite match the standard linear-combination prediction, parameter estimation can fit it by adjusting weights that are peculiar to B_i. (In our example θ_{2i} is such a weight; more typical models will adjust the weights of per-event features.) This modifies the coefficients for B_i in particular, picking up the slack in equation (3.18) so that $\Pr(B_i)$ is *exactly* equal to a linear combination with modified coefficients. The prior prefers these additional features to have low weights so that the modifications are not great.

7.1.5.7 Learning the Strength of Smoothing

In the MacKay-Peto approach to bigram smoothing, the interpolation parameter α_{w_j} of equation (7.1) is related not only to the count of w_j, as in traditional backoff, but also to the learned variance of the Dirichlet. If the aggregated distributions $\Pr(\cdot \mid w'_j)$ are quite different from one another—i.e., they do not form a natural class—then this variance will be estimated as high and backoff will automatically be weak. Notice that α_{w_j} is set without the use of held-out data.[7]

Thus, the MacKay-Peto approach learns the *degree* of consensus among the individual words' distributions (the Dirichlet variance), as well as the consensus distribution itself (the Dirichlet mean). Poor consensus among the individual distributions means that they will be smoothed less toward the consensus distribution.

Our example transformation model can be augmented to have a similar property. §7.3.2.2 describes how to learn a common variance for the weights $\theta_2, \theta_4, \ldots \theta_{2k}$. This variance estimates the degree of listability (idiosyncrasy) of the B_i events. If the B_i events

[7]Suppose the parameter vector of the Dirichlet is $\alpha\vec{m}$ where $\sum_i m_i = 1$. Then $\vec{m}$ is the mean and α is inversely related to the variance. The full-Bayesian posterior estimate of $\Pr(\cdot \mid w_j)$ is equivalent to setting $\Pr_{\text{backoff}} = \vec{m}$ and $\alpha_{w_j} = \#(w_j)/(\#(w_j) + \alpha)$. The latter is close to 1 if the variance is high.

appear to be derived at a constant rate from the A_i events, then the estimated variance will be low to explain why the aforementioned weights tend to be close to zero. In that case, smoothing will be more aggressive. But if the B_i events are highly (de)listable—their probabilities are hard to predict from the A_i events—then the estimated variance will be high, making smoothing weaker.

7.2 More Sample Applications

Recall that §3.7.1 described the kind of domain that are amenable to transformation modeling. We now examine a few more examples.

7.2.1 Google's PageRank

7.2.1.1 Original PageRank

A rather different example is the PageRank distribution used by the Google search engine to assess the quality of pages on the World Wide Web (Brin and Page, 1998). In Google's view, a web page is good if many other good pages link to it. More precisely, it is good if a web surfer randomly clicking on links would be likely to happen upon it (and stop and read it).

Imagine that the surfer takes a random walk on the web, where the pages are regarded as graph vertices and the links are edges. At any page, the walk has an 0.15 chance of halting, and an $0.85/k$ chance of continuing along any one of the page's k outgoing links. PageRank(e) is defined as the probability of halting at page e, just as in our §3.3.2.

PageRank can be described as a transformation model $\Pr_\theta$ with fixed θ. An arc representing a link from page e to page e' has no features (hence has G-value of $\exp(0) = 1$), and for every such arc, there is also an arc from e to HALT with a feature t_0. The weight of t_0 is $\theta_0 = \ln(0.15/0.85)$.[8] There are also featureless edges from START to every page e, so all pages are equiprobable start points in the random walk.

[8]The use of k parallel HALT arcs from a page with k links ensures that the page's halt probability is constant (at 0.15) regardless of k. An alternative solution is to give each page a second, auxiliary vertex in the graph. The page's original vertex has only two outgoing arcs, one to HALT (probability 0.15) and one to its auxiliary vertex (probability 0.85). The auxiliary vertex has k outgoing arcs corresponding to the k links.

7.2.1.2 Improving PageRank by Learning Features

One could presumably improve PageRank by using more features and learning their weights under a prior. This improves the random walk, by modeling a web-savvy surfer who has good intuitions about *which* links from a page are worth clicking on. The features help the model distinguish links of differing quality:

- An arc from e to e' ought to have features that might correlate (or anti-correlate) with the strength of page e's recommendation of page e'—for example, the placement of the link, words in the link anchor text, or similarities in the texts or URLs of the two pages.

- An arc from e to HALT should have features that correlate with page e's recommendation of itself as a place to stop and read. For example, a page that consists entirely of links (someone's bookmark collection) might be an excellent place to pass through but a poor place to halt.

- Finally, an arc from START to e (or better, all arcs to e: see §3.6.1) should have features that correlate with the "intrinsic" merit of page e, such as good spelling, a short URL, a recommendation from a third-party rating service, or—in the case of a specialized topical search engine—keywords that indicate topicality.

To learn the weights of these features—i.e., to model the savvy surfer—we need evidence of the desired PageRank distribution: that is, evidence as to which pages are "really" good or bad. This lets us choose among candidate models $\Pr_\theta$ according to how well they predict this evidence. A random sample from the desired distribution would be one form of evidence, but any correlates of the desired distribution will do (see §3.1.5). Some possible correlates include measures of where actual surfers spend their time (similar to a random sample of PageRank), third-party awards, manual annotations of a subset of the pages, Brin and Page's original PageRank distribution,[9] and user responses to search

[9] Of course, if we try to find θ such that $\Pr_\theta$ exactly reproduces the original distribution, we will end up choosing the same $\theta = \langle \ln(0.15/0.85), 0, 0, 0, \ldots \rangle$ that the original distribution used, unless we modify the prior so that it hates this θ. A different strategy is to randomly remove some arcs from the transformation graph when estimating the new θ. This demands that we find "smart" features that can reproduce the rankings of the "dumb" model despite having less help from web page authors. Once the new θ is estimated, the missing arcs can then be restored to yield a final definition of $\Pr_\theta$.

engine results.[10]

The smoothing attempts to predict this evidence using just the features that are available to it, with *a priori* reasonable feature weights, and thereby reconstructs a plausible PageRank distribution underlying the evidence. The size of $1/\sigma^2$ determines the importance placed on reasonableness of the feature weights.

Other refinements are possible. For instance, one could use the transformational lookahead of §7.3.4 below, which effectively downweights links to a page that recommends neither itself nor other pages very strongly.

7.2.1.3 Incorporating Finer-Grained Evidence Into PageRank

Suppose the available evidence about PageRank includes user-specific or query-specific page ratings. This finer-grained evidence makes it possible to learn more specific features. The trick is to define the events not as pages but as $\langle \text{user}, \text{page} \rangle$ or $\langle \text{query}, \text{page} \rangle$ pairs, where an arc connects $\langle x, y \rangle$ to $\langle x', y' \rangle$ iff $x = x'$ and page y links to page y'. The transformation graph now consists of many parallel copies of the web graph, one per user or query. In particular there are many arcs that correspond to the same web link. Any feature describing that link—even a feature wholly specific to that link—will now be shared by all those arcs. The model will now estimate a high weight for that feature if the two pages connected by the link have correlated ratings across the many users or queries, implying that surfers who are interested in the first page tend to follow the link to the second page.

Given user-specific or query-specific ratings, and the above transformation graph, we can even arrange for the same page to have *different ratings* depending on the user or the user's query. Such personalized ratings are potentially useful. They require additional features that are sensitive to the particular user or query. For example, per-event features (§3.6.1): by increasing the single weight $\theta_{\langle x,y \rangle}$, we can increase user x's predicted interest in page y and all of y's near descendants. Another example is a feature that appears on arcs to $\langle x, y \rangle$ just if the words in query x appear (prominently) on page y. A high weight

[10] Suppose the search engine recommends pages e, e' and the user clicks on e. This is weak evidence about PageRank: it is somewhat more likely if $\text{PageRank}(e) > \text{PageRank}(e')$ than vice-versa, so we would prefer a model $\Pr_\theta$ of PageRank with this property.

for this feature means that a query rates a page highly if the page is easy to reach by a random walk *through* pages that tend to mention words of the query.[11]

7.2.2 Collaborative Filtering

A related task is collaborative filtering, also known as recommender systems or "automated word of mouth" (Goldberg et al., 1992; Shardanand and Maes, 1995). An example is §7.2.1's proposed extension of the PageRank transformation model over $\langle \text{user}, \text{page} \rangle$ pairs, which estimates user-specific ratings of web pages. Exactly the same approach applies to other domains.

Suppose a video rental store plans to make recommendations. It wants to estimate how much each of its customers would like each of its videos. In other words, it wishes to estimate an interest function over $\langle \text{customer}, \text{movie} \rangle$ pairs. If this function is non-negative and is normalized to sum to 1, then it may be regarded as a probability distribution over such pairs.

Imagine that the distribution is generated as follows. A random customer walks into the store and randomly thinks of a past movie (or genre) that she liked. Her mind then drifts knowledgeably from that movie (or genre) to other movies of related interest, taking a random walk in cinematic space until she randomly decides to stop and rent the movie she is currently thinking of.

Such a distribution may be regarded as emerging from a transformation model over $\langle \text{customer}, \text{movie} \rangle$ pairs (and START). For example, if the store suspects that interest in movie_1 may translate into interest in movie_2, then it arranges for the transformation graph to include an arc from $\langle \text{customer}_i, \text{movie}_1 \rangle$ to $\langle \text{customer}_i, \text{movie}_2 \rangle$ for each i. Such an arc bears features that jointly describe the pair of movies, as well as output features (§3.6.2) that describe movie_2 or $\langle \text{customer}_i, \text{movie}_2 \rangle$.

Learning the feature weights will allow the store's computer's mind to drift knowledgeably on behalf of a customer who may not be so knowledgeable. For example, the model might learn that customer C likes cinematographer X, or that customers who like films

[11] By contrast, in Google (Brin and Page, 1998), a page is rated highly if it is easy to reach through any old random walk. The words in the page or query are taken into account only during a post-processing step, and the words on the intermediate pages of the random walk are never considered.

starring actor Y also tend to like the ones starring actor Z. (See §7.3.2.2 for an extension.)

A nice feature of the model is that it need not be misled into false generalizations by popular movies. Let all arcs leading to a given movie carry an output feature specific to that movie. If the movie is unusually popular within its genre—customers's minds tend to drift to it from related movies—the best explanation is to raise that specific feature's weight. It is not necessary to suppose that the movie's director or star is the reason for its success, unless other movies with the same director or star are also unusually popular.

As mentioned in §3.7.1, the transformational approach will not capture negative correlations: "If you liked this movie, you'll hate that one." For instance, suppose that many Monty Python fans also like grade-B sci-fi flicks, which lovers of art movies always hate. The model's weights can capture the fact that there is no particular crossover appeal directly from art movies to sci-fi, and even that a given customer tends to love art movies and Monty Python and hate sci-fi. But it cannot learn that *in general*, an interest in art movies will *inoculate* a Monty Python fan against the usual crossover to sci-fi. This weakness might or might not be a practical obstacle.

7.3 Variations on the Theme

To round out our formal discussion of transformation models, we now sketch several possible variations on the basic idea. Some of these variations might be useful for the syntactic smoothing task of this thesis, or for other linguistic or non-linguistic applications. For most of the variants it is obvious how to adapt the algorithms of Chapter 4; for others, algorithms will be considered in §8.4.

7.3.1 Other Ideas of the Transformation Graph

7.3.1.1 Variation: Non-Binary Features

In the basic transformation models of §3.2.1, each edge in the transformation graph is labeled with a subset F of $\mathsf{Features} = \{t_1, t_2, \ldots t_k\}$. For a given edge and a given feature t_i, either t_i appears in the edge's label or it does not.

A natural generalization is to allow the same feature to appear multiple times on an edge, or even fractionally or negatively many times.

In the general case, the label of an edge is a real vector $\vec{F} = \{F_1, F_2, \ldots F_k\} \in \mathbb{R}^k$, where the **coefficient** F_i is interpreted as the "number of times" that t_i appears on the edge, or the "strength" of t_i as a description of that edge. Our original definition restricted $F_i \in \{0, 1\}$. If F_i is allowed to range over the integers, non-negative reals, or reals, we simply generalize the definition of G_θ in equation (3.13) as follows:

$$G_\theta(\langle e, e', \vec{F}\rangle) \stackrel{\text{def}}{=} \exp(\sum_{i=1}^{k} F_i \cdot \theta_i) = \exp(\vec{F} \cdot \vec{\theta}) > 0 \tag{7.4}$$

§7.3.2.2 discusses the possibility of learning these coefficients.

7.3.1.2 Variation: Other Distributions Over Competing Edges

In §3.2 we defined log-linear transformation models. Instead of log-linear distributions, one might allow other classes of parameterized distributions over sets of competing edges. Note that in this case we may not need the finite-fanout property of §3.2.1.[12]

What is important is to tie the parameters so that edges that encode the "same" transformation, or similar transformations, will have similar probabilities. For example, in the case of log-linear distributions, such edges are labeled with identical or overlapping feature sets.

Like the topology of the model, the tying of parameters is up to the domain expert who designs the model. It is supposed to capture relevant facts about the domain.

One advantage of the log-linear approach is that features need not be orthogonal. Hence if the model designer is unsure whether a given set of edges should have correlated probabilities, then he or she can leave it up to the data: introduce a feature t_i shared by

[12]As an example, let Events be Σ^*, the set of all strings over some alphabet Σ, and use a probabilistic finite-state transducer to specify the model's arcs. If the transducer maps string α to string β with probability $p > 0$ (conditioned on α), then the model has an arc from α to β with probability p. Assuming that the transducer has the correct form, the arcs from α (perhaps infinitely many) will have total probability of 1.

The parameters of this model are simply the parameters of the transducer—namely, the transducer's arc probabilities or underlying variables used to define them. Although the distribution defined by the model is in general uncomputable, it can be approximated by composing several copies of the transducer. (Note that even one copy of a simple transducer can make random contextual edits throughout a string.)

One could similarly let Events be a set of trees and use a tree transducer, yielding something like a classical transformational grammar.

just those edges, impose a prior that encourages the weight θ_i to be close to zero, and let the learning algorithm decide whether the data justify a large $|\theta_i|$ that will drive all those edges up (or down) at once. Of course this approach will not work well if taken to extremes—for example, introducing a feature for each subset of edges. Unless the priors are very strong, having more features than nodes would make it harder to learn the model parameters than to learn the distribution over nodes directly.

7.3.1.3 Variation: More General Transductions

Recall from §7.1.2 that a transformation model can be regarded as a finite-state automaton that produces outputs—or equivalently, transduces ϵ to outputs.

One could in principle generalize to allow non-ϵ *input* symbols on the arcs of the transducer. This would allow the stochastic transformational process to be externally constrained. One would be interested in the conditional probability of transformationally generating event e given that the transformational process reads a particular string α. We do not pursue this possibility here.

Another interesting possibility is to allow non-ϵ output symbols to appear freely in the transducer. In this case, the transformational process would output a sequence of symbols, not just one.

7.3.1.4 Variation: Learning the Model Topology

Rather than fixing the topology of the model (i.e., the transformation graph) in advance, one might treat it as an additional parameter to learn, something not attempted here.

Similar problems have been studied. For example, there exist methods for learning small arbitrary topologies for hidden Markov models (Stolcke and Omohundro, 1994a), finite-state automata (Lang et al., 1998), and Bayesian networks (Heckerman, 1995).

7.3.2 Other Priors

While the simple Gaussian prior given in §3.5 is often sufficient, generalizations are possible. We consider several.

7.3.2.1 Variation: More General Gaussian Priors

If the features $t_i \in \mathsf{Features}$ fall into natural classes, one may wish to specify a different variance for each class (or, more precisely, for the weight parameters θ_i associated with that class). In general one can separately specify a variance for each component of θ:

$$\theta \sim N(0, \sigma_1^2) \times N(0, \sigma_2^2) \times \cdots \times N(0, \sigma_k^2)$$

One could also specify a mean for each component:

$$\theta \sim N(\mu_1, \sigma_1^2) \times N(\mu_2, \sigma_2^2) \times \cdots \times N(\mu_k, \sigma_k^2)$$

Why? If there is a prior reason to believe that arcs with feature t_i are more or less likely than their competitors, then one might set μ_i to be positive or negative, respectively. μ_i describes the size of this anticipated effect and σ_i describes the modeler's uncertainty in μ_i.

The above generalizations simply allow $\Pr_{\text{prior}}(\theta)$ to be any multivariate Gaussian with diagonal covariance matrix.[13]

7.3.2.2 Variation: Variances and Feature Classes

The previous section allowed that different classes of features might have different variances. One might wish to *learn* these variances from data. (These remarks also apply to the features of ordinary maximum entropy models with priors (Chen and Rosenfeld, 1999).)

[13] Generalizing further to an arbitrary covariance matrix would allow the modeler to specify first-order correlations among the feature values. If different values of θ correspond to different languages, then such a correlation is a cross-linguistic typological generalization. For example, θ_2 might be expected to rise and fall with θ_4, or more generally with a linear combination of several components of θ.

However, this does not really increase the power of the formalism, since a coordinate transform on θ can then restore us to the diagonal case (perhaps at some cost to efficiency). In the example, θ_4 could be added to every feature set F containing θ_2, and then θ_2 would only have to capture the difference between θ_2 and the old θ_4.

For such a transform to work in general, we must allow real feature vectors rather than just feature sets (see §7.3.1.1). Suppose we have a transformation model, and the prior is given by a multivariate Gaussian with mean $\vec{\mu}$ and covariance matrix Σ: then $\Pr_{\text{prior}}(\theta) \stackrel{\text{def}}{=} (\theta - \vec{\mu})^{\mathsf{T}} \Sigma (\theta - \vec{\mu})$. But by Singular Value Decomposition, we can find square real matrices Σ' and U, where Σ' is diagonal, U is invertible, and $\Sigma = U^{\mathsf{T}} \Sigma' U$. Define $\theta' = U\theta$, $\vec{\mu}' = U\vec{\mu}$, and $\Pr_{\text{prior}}{}'(\theta') = (\theta' - \vec{\mu}')^{\mathsf{T}} \Sigma' (\theta' - \vec{\mu}') = \Pr_{\text{prior}}(\theta)$. Construct a new transformation model that is identical to the old except that each arc A with feature set F has been replaced by a new arc A' with feature set FU^{-1}, so that $G_{\theta'}(A') = FU^{-1} \cdot \theta' = F \cdot \theta = G_\theta(A)$. The new model with prior $\Pr'_{\text{prior}}$ is therefore equivalent in all respects to the old model with prior $\Pr_{\text{prior}}$.

For example, in the collaborative filtering application of §7.2.2, a given customer C might place great stock in the director of a movie. Features that capture this customer's preference for particular directors—features of the form "customer is C and director of movie$_2$ is D"—tend to have weights far from 0. How to capture this generalization? We need to learn that these weights have high variance. That has the desired effect: that when the customer likes a new movie, we will prefer to explain this by raising these weights, guessing that the customer's interest will transfer better to other movies with the same director than with the same star.

As a final example, in the lexicon smoothing model of §3.7.2, some types of lexical entries are more idiosyncratic or "listable" than others. For example, active-verb entries are idiosyncratic in English. Some of the possible active-verb entries have large per-event weights (§3.6.1), meaning that their headwords are listed for use as active verbs, while others do not. By contrast, passive-verb entries are derived from active-verb entries at a fairly constant rate. Their derived probabilities tend to be correct, so their per-event weights can generally stay close to zero. We would like to learn that the per-event weights for active verbs have higher variance than the per-event weights for passive verbs. A similar case was discussed in §7.1.5.7.

If the features can be partitioned into natural classes, each with its own variance, then things are conceptually simple. We consider the variances to be additional parameters. We can impose a simple prior on them, for example, that they are independent and identically distributed according to some gamma distribution.[14] Then we estimate the parameters as before (§3.1, perhaps considering footnote 2).

If a feature can belong to multiple overlapping classes, each with its own variance parameter, then it is necessary to combine these different variances in some way into a single variance for the given feature. A reasonable combination function is the harmonic mean: that is, to define the reciprocal variance $1/\sigma^2$ for the feature, average the reciprocal variances $1/\sigma_i^2$ associated with the classes.[15]

[14] This independence assumption might not hold in all domains. In the word-sense disambiguation model, for example, the variances would presumably tend to decrease as one travels down the WordNet hierarchy.

[15] Motivation: If one class suggests that the feature's weight should be sampled from $N(0, \sigma_1^2)$ and the other suggests it should be sampled from $N(0, \sigma_2^2)$, we would like our sample to be consistent with both distributions. (That is, each class containing the feature is a source of constraint: if σ_1 is small, then we

If it is confusing to write $\Theta_i \sim N(0, \sigma_j^2)$ where σ_j is itself a random variable that depends on the class or classes of feature t_i, this so-called "hierarchical Bayes" technique can be avoided by reparameterizing slightly. Simply replace θ_i with $\sigma_j \theta_i$ in equation (3.13), and write $\Theta_i \sim N(0, 1)$. This formulation is equivalent. It also shows that learning the variances amounts to learning a multiplier for the coefficient of t_i (§7.3.1.1). Occurrences of t_i on different arcs may have different coefficients, fixed by the modeler, but they all use the same multiplier, unless they are considered to fall into different classes.

7.3.2.3 Variation: Asymmetric Gaussian Priors

It is not necessary to use normal (Gaussian) priors on feature weights. Keeping the assumption that each component of θ is independently sampled from some distribution, a useful distribution to use is the **asymmetric Gaussian**:

$$\Pr(x) = \begin{cases} c \exp(-x^2/2\sigma_+^2) & \text{for } x \geq 0 \\ c \exp(-x^2/2\sigma_-^2) & \text{for } x \leq 0 \end{cases} \tag{7.5}$$

where c is the normalizing factor $2/(\sqrt{2\pi\sigma_+^2} + \sqrt{2\pi\sigma_-^2})$. This density function is continuous and differentiable at 0.

If one believes that t_i is the kind of feature that might seriously help, but is unlikely to seriously harm, the probability of the arcs on which it appears, then one can give θ_i an asymmetric Gaussian prior with $\sigma_+^2 > \sigma_-^2$.

7.3.3 Variation: Exponential Priors and Zipf's Law

A further change to the prior is also worth considering. Our univariate Gaussian density in x is obtained by exponentiating a quadratic in x (i.e., e^{ax^2+bx+c}, $a < 0$). The variations

expect the weight θ_i to be close to 0 even if σ_2 would allow it to be far away.) So let us sample both distributions independently and require the samples to agree on θ_i: $\Pr(\Theta_i = \theta_i) \stackrel{\text{def}}{=} \Pr(X = Y = \theta_i \mid X = Y)$ where $X \sim N(0, \sigma_1^2)$ and $Y \sim N(0, \sigma_2^2)$. The density of Θ_i is then the normalized pointwise product of the two normal densities. However, it seems preferable to correct for the number of classes (two in this case) by taking the normalized pointwise geometric mean (rather than product). Then $\Theta_i \sim N(0, \text{harmmean}(\sigma_1^2, \sigma_2^2))$, as is easy to show. Note that the harmonic mean of k positive numbers lies between their minimum and k times their minimum—regardless of their maximum—so smaller variances can mostly overrule larger ones as hinted above.

above stayed within this framework, but we now consider exponentiating other powers of x.

7.3.3.1 Zipf's Law Exactly

The motivation comes from Zipf's Law. As in §3.8.8, consider the canonical situation where each of k competing arcs has a single feature and all k features are distinct. When the feature weights are IID normal (equation (3.21)), the resulting arc probabilities are *a priori* distributed log-normally about their geometric mean. But suppose the competing arcs are being used (for example) to choose a word at random. (This is true of the arcs from START in §6.4.2.) Then *a priori* we would expect the arcs' probabilities to fall according to a power law—since in practice the distribution of words in a language, like many other naturally occurring distributions, has this form (Zipf, 1932; Zipf, 1949). For example, the tails of the distribution should be longer than predicted by equation (3.21).

This could be arranged with a different prior on θ. Replace equation (3.21) with

$$\Theta \sim \underbrace{E(\lambda) \times E(\lambda) \times \cdots \times E(\lambda)}_{k} \tag{7.6}$$

so that the weights are independently exponentially distributed instead of independently normally distributed. That is, for $\theta \in \mathbb{R}^k_{\geq 0}$,

$$\Pr(\theta) = \lambda^k \exp(-\lambda \sum_{i=1}^{k} \theta_i) \tag{7.7}$$

Then a sample distribution over k competing arcs (for large k) will follow a power law with exponent $-1/\lambda$: i.e., the i^{th} most probable arc will have probability proportional to $i^{-1/\lambda}$.[16] To match the empirical distribution of word frequencies (Zipf, 1932; Zipf, 1949), we would want to put $\lambda \approx 1$.

Notice that we are modeling the Zipf distribution as a double sample. The first chooses

[16] Proof: Given an arc whose weight is θ_j. What is its expected rank i? That is, on average, how many of the k weights will be $\geq \theta_j$? Integrating equation (7.7), we obtain $i = k \cdot \exp(-\lambda\theta_j)$. Then $i^{-1/\lambda} = k^{-1/\lambda} \cdot \exp\theta_j$, which is indeed proportional to the arc's probability. It is easy to confirm this by generating such a sample distribution, say over $k = 10000$ arcs, and plotting the log-samples as a function of their log-ranks. The plot will be a straight line with slope $-1/\lambda$, indicating a power law.

a language (really a weighted vocabulary), by sampling feature weights $\theta_1, \ldots \theta_k$ independently from the prior (7.6), yielding a probability distribution over the k arcs, or equivalently k word types. The second chooses a corpus in that language, by sampling that probability distribution to obtain a number of word tokens. If both samples are large, then the corpus will look Zipfian.

This is not to claim that a competition among word types with independently chosen weights, most of them low, is necessarily the cause of Zipf's Law. (Many other explanations have been proposed, some of them (Li, 1992) even simpler.) But since our model assumes some sort of competition among word types, the prior of equation (7.6) would be at least a convenient way of favoring languages that agree with Zipf's Law.

The choice among word types is, of course, only one instance of a competition among arcs in our model. It is particularly simple because each arc has a single unique feature. It remains to be seen whether the exponential prior (7.6) is also useful for modeling other sets of competing arcs. Notice that if the set of competing arcs from a lexical entry tended to follow Zipf's Law directly, this would mean that one or a few transformations (perhaps including HALT) would have good probabilities of applying to that entry, with most of the rest having negligible probabilities.

7.3.3.2 Allowing Negative Weights

The main objection to the exponential prior of equation (7.6) is that it disallows negative weights, assigning them a probability of zero. Negative weights are not always as useful as positive ones (§7.3.2.3), but they remain useful, since some features may indicate that a transformational arc is less probable than its other features might suggest. Can we modify the prior to allow negative weights?

One solution is to replace equation (7.7) with the symmetric version

$$\Pr(\theta) = \frac{1}{2}\lambda^k \exp(-\lambda \sum_{i=1}^{k} |\theta_i|) \tag{7.8}$$

so that the weights 3 and -3 are equally likely.[17] In our simple case where each competing arc has a single, unique weight, the resulting distribution is hard to distinguish in practice

[17] One could however reduce the probability of the negative weights along the lines of §7.3.2.3.

from the Zipfian one, because it simply extends the tail with additional word types that are so rare as to be almost unmeasurable.[18]

Like the Gaussian prior of §3.8.8, either of these new priors—equation (7.7) or equation (7.8)—assigns a higher probability to a weight vector θ if the weights in the vector are close together. Given θ, recall that $\theta - d$ for any $d \in \mathbb{R}$ will yield the same probability distribution over competing arcs, but that the prior prefers some d values to others. If we maximize over d, the prior probability falls off exponentially with the total absolute distance of the weights θ from their minimum (i.e., $\sum_i(\theta_i - \min(\theta))$ in equation (7.7)) or from their median (i.e., $\sum_i |\theta_i - \text{median}(\theta)|$ in equation (7.8)). Integrating over d gives similar results.[19]

7.3.3.3 Differentiability and Curved Power-Law Distributions

Since equation (7.8) is not differentiable at zero, making it harder to maximize the posterior distribution of equation (3.4), one might prefer to substitute

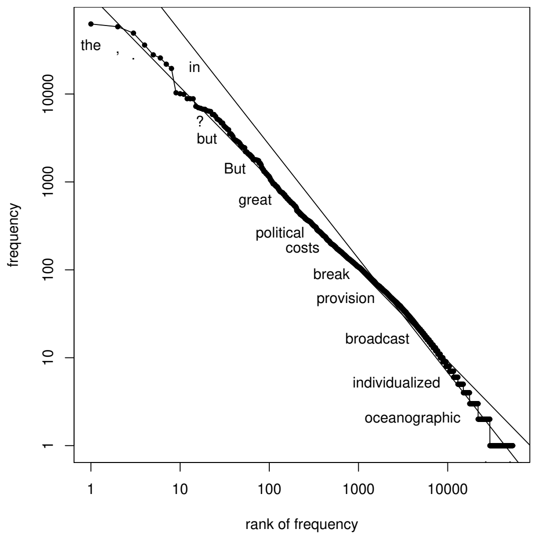
$$\Pr(\theta) = \frac{1}{Z}\exp(-\lambda \sum_{i=1}^{k} |\theta_i|^\alpha/\alpha) \qquad \text{(choosing } Z \text{ for normalization)} \tag{7.9}$$

for α slightly above 1. This density is everywhere singly differentiable if $\alpha \neq 1$.

Notice that the new density "interpolates" between equation (7.8) ($\alpha = 1$) and our original Gaussian prior ($\alpha = 2$). Increasing α towards 2 also has an empirical benefit: it captures the slight downward curvature that is characteristic of many Zipf distributions in nature. Fig. 7.1 shows how this slight curvature in the Brown corpus can be matched by using $\alpha = 1.323$. (Other methods of modeling this curvature are discussed by Montemurro (2001).)

[18] Not only are the individual word types that correspond to negative weights rare, but even their *aggregate* effect is hard to discern in practice. Yes, other things equal, the proportion of word types that appear only once is predicted to be slightly greater under equation (7.8) than under equation (7.7). But it takes a very large corpus to see that effect. When we generated many corpora experimentally according to equations (7.7) and (7.8)—even "large" corpora that contained as many as 10^5 or 10^6 word tokens of only 10^4 word types—the proportion of singletons was indeed slightly higher in the average vocabulary from equation (7.8), but this difference paled next to the variation among corpora generated by the same technique. (In any case other things are *not* equal: one can fit empirical distributions quite well by freely varying λ, k, and α in equation (7.8) or equation (7.9). See Fig. 7.1.)

[19] Integrating the exponential density simply scales it, giving $\Pr(\theta - \min(\theta))/\lambda k$. Integrating the symmetric version gives $\sum_{i=1}^k \Pr(\theta - \theta_i)((\frac{1}{\lambda(k-2i+2)} \text{ or } \theta_i) - (\frac{1}{\lambda(k-2i)} \text{ or } \theta_i))$, where "$x$ or y" denotes x if it is defined and y otherwise. This still requires clustering about the median, but penalizes outliers slightly more.



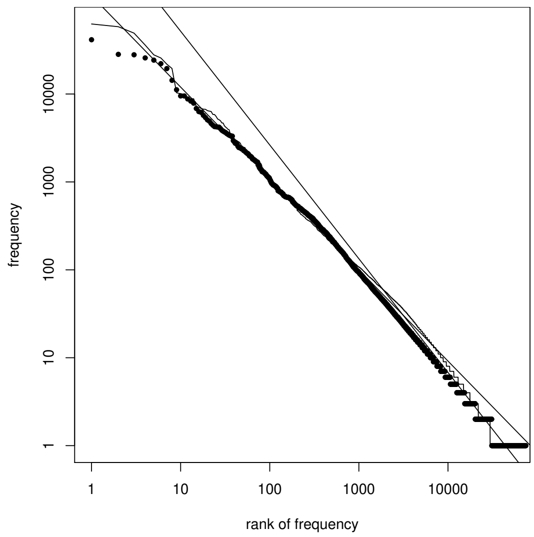


Figure 7.1: (a) Frequency counts of the 53849 word types in the Brown corpus, plotted on a log-log scale against their rank. The data approximately follow a power law with exponent -1.29 (the steeper regression line), but for the 1000 most common words, the exponent is more like -1.04 (the shallower line). (b) A very similar artificial "corpus" can be generated from equation (7.9) ($k = 1110384$ types, most of which never appear; $\alpha = 1.323$; $\lambda = 0.405$). The background lines and curve are copied from graph (a).

7.3.4 Variation: Transformational Lookahead

Recall the interpretation of transformation models as stochastic transformational processes §3.3.2. At each stage, the process stochastically chooses a new transformation to apply to the current event, or else chooses to halt and output the current event.

One might wish to equip this process with some sort of lookahead. For example, suppose we abandoned the coaccessibility property (§3.2). Then an arc from e to e' might have features with strongly positive weights, yet be a "poor route" because there is no path to get from e' to HALT. Then one might like the process to avoid transforming e to e' despite the high weights. This requires lookahead.

More subtly, even if there is a path from e' to HALT, it may still happen that (for given parameters) there is no "good" path from e' to HALT. For example, perhaps all the arcs leaving e' describe "poor" transformations in the sense that they have low G-values (§3.2.2).

Our original definition of transformation models simply normalizes those G-values to

P_θ probabilities that sum to 1. That is, the random walk is required to leave e' somehow, so it is only concerned with the relative values of the arcs from e', not their absolute values. But as an alternative, we might want to capture the latter notion. We might want to recognize that there is no way or no "good" way to leave e', or to get from e' to HALT, and have this affect the chance that the process will choose a path leading to e' in the first place.

We now consider three approaches to lookahead, focusing primarily on the case of finite transformation graphs. Estimation procedures will be given in §8.4, especially §8.4.1.

7.3.4.1 Renormalizing $\mathbf{Pr}_\theta$

First we consider a simple renormalization that does not avoid "poor" random walks, except to ensure that the random walk does get to HALT by some route.

As noted above and in §3.3.1, our original definition of $\Pr_\theta$ might yield a **deficient** probability distribution,[20] at least on an infinite transformation graph, because the probability may be < 1 that the random walk halts at all. This probability of halting is conveniently given by the flow (expected number of visits) to HALT: $I_\theta(\text{HALT})$.

Usually deficiency is not a problem. We are guaranteed that $(\forall\theta) I_\theta(\text{HALT}) = 1$ for many transformation graphs, such as finite graphs with our usual coaccessibility property (§3.2.1). Even if not, we may still be willing to fit a deficient distribution to the evidence. (We will tend to choose θ so that $I_\theta(\text{HALT}) \approx 1$ anyway, since the likelihood of the data tends to be higher when $\Pr_\theta$ is not very deficient.)

But in general, one can simply normalize the probability model, redefining $\Pr_\theta(e)$ as the probability that the random walk halts at e *given that* it halts at all. This means dividing the right-hand side of equation (3.20) by $I_\theta(\text{HALT})$:

$$\Pr_\theta(e) = \sum_{A=\langle e,\text{HALT},F\rangle\in\mathsf{Arcs}} \bar{I}_\theta(A) \tag{7.10}$$

$$= \bar{I}_\theta(e) \cdot \sum_{A=\langle e,\text{HALT},F\rangle\in\mathsf{Arcs}} P_\theta(A) \tag{7.11}$$

[20]That is, $\sum_{e\in\mathsf{Events}} \Pr_\theta(e) < 1$.

where we define the normalized flows $\bar{I}_\theta$ via

$$\bar{I}_\theta(e) \stackrel{\text{def}}{=} I_\theta(e)/I_\theta(\textsc{Halt}) \tag{7.12}$$

$$\bar{I}_\theta(A) \stackrel{\text{def}}{=} I_\theta(A)/I_\theta(\textsc{Halt}) \tag{7.13}$$

An interpretation of the normalized model is that when a random walk turns out to take infinite time, we throw it away and try another random walk, repeating until we finally get one that halts. The normalized flow $\bar{I}_\theta(e)$ can then be regarded as the expected number of visits to e in the random walk that we actually keep.

The notion of normalized flow will also come in handy in the lookahead models below.

7.3.4.2 Late Normalization

We now consider the more general problem of using lookahead to avoid "poor" transformations (not merely dead-ends).

One technique is to normalize G-values over competing paths rather than competing arcs. In §3.2.2, we normalized the value $G_\theta(A)$ to a probability $P_\theta(A)$, and then extended $P_\theta(A)$ over paths $\vec{A}$:

$$P_\theta(A) \stackrel{\text{def}}{=} G_\theta(A)/\sum_{A' \in \mathsf{competitors}(A)} G_\theta(A') \tag{7.14}$$

$$P_\theta(\vec{A}) \stackrel{\text{def}}{=} \prod_{i=1}^{n+1} P_\theta(A_i) \tag{7.15}$$

But an alternative way to define the path probability function P_θ is to extend the definition of G-value over paths first, and then normalize:

$$G_\theta(\vec{A}) \stackrel{\text{def}}{=} \prod_{i=1}^{n+1} G_\theta(A_i) \quad (= \exp(\textstyle\sum \text{all weights along path } \vec{A})) \tag{7.16}$$

$$P_\theta(\vec{A}) \stackrel{\text{def}}{=} G_\theta(\vec{A})/\sum_{\vec{A}' \in \{\text{paths from } \textsc{Start} \text{ to } \textsc{Halt}\}} G_\theta(\vec{A}') \tag{7.17}$$

Now there is no notion of arc probabilities, but only of path probabilities. A path is likely to the extent that strongly positive features anywhere along it outweigh strongly negative features anywhere along it. In short, we have a single log-linear model over the infinite set of paths, rather than a combination of separate log-linear models over finite sets of competing arcs. The same idea was independently proposed by (Lafferty et al., 2001) for parameterizing path probabilities in finite-state machines.

Indeed, we can regard the model as a kind of maximum-entropy model over the set Events. Like any maximum-entropy model, it must define the pertinent features of each event. The twist here is that the features of an event are primarily features of its "pedigree"—the path that derived the event. If an event has multiple pedigrees, each yields a different probability under the model, and these probabilities are summed.

In the case where Events is finite, the normalization over all paths from START to HALT is straightforward to carry out. Redefine the flow $I_\theta(e)$ by replacing $P_\theta(A) \in (0, 1]$ with $G_\theta(A) \in (0, \infty)$ in the recurrence relation of equation (3.18). We can still solve simultaneously for all the $I_\theta(e)$ values by matrix inversion. Now observe that the redefined flow $I_\theta(e)$ is the total G-value of all paths from START to e. In particular, $I_\theta(\text{HALT})$ is the desired denominator of equation (7.17).

As usual, $\Pr_\theta(e)$ is defined as the probability of choosing a path from START to e and then halting immediately (equation (3.17)). By the properties of $I_\theta(e)$ and $I_\theta(\text{HALT})$ just mentioned,

$$\Pr_\theta(e) = \frac{I_\theta(e) \cdot \sum_{A=\langle e, \text{HALT}, F\rangle \in \mathsf{Arcs}} G_\theta(A)}{I_\theta(\text{HALT})} \tag{7.18}$$

$$= \bar{I}_\theta(e) \cdot \sum_{A=\langle e, \text{HALT}, F\rangle \in \mathsf{Arcs}} G_\theta(A) \tag{7.19}$$

where $\bar{I}_\theta(e)$ denotes the normalized flow through e as in equation (7.12). Note that this definition of $\Pr_\theta$ is properly normalized so cannot be deficient.

The redefined $I_\theta(e)$ values are conceptually infinite sums. These sums will diverge (leading to negative solutions of the recurrence relation) if the transformation graph includes any cycles $\vec{A}$ of non-negative total weight, i.e., $G_\theta(\vec{A}) \geq \exp(0) = 1$. Values of θ that make I_θ (hence $\Pr_\theta$) diverge in this way should be regarded as assigning likelihood 0 to the observed evidence.[21]

The above discussion considers only finite event sets. If Events is infinite, we are left with methods for approximating $\Pr_\theta(e)$ (e.g., relaxation). However, those methods can

[21] Many values of θ do allow I_θ to converge. For example, if all feature weights $\theta_i < 0$, then G-values are in $(0, 1)$.

still be used under the "lookahead" definition of $\Pr_\theta$ (see §8.4.1).

7.3.4.3 Transformational Circuit Models

There is another, more elegant way to redefine transformation models so as to "allow lookahead." This approach is inspired by electrical circuits, which have the desired property: if a node has only high-resistance paths to ground (cf. poor paths to HALT), then less current will flow to that node.[22]

In a **transformational circuit model**, the transformation graph is undirected: that is, all transformations are reversible. The weighted features on an undirected transformation arc are used to define its conductance (i.e., 1/resistance). Electricity flows from START (a positive battery terminal) through the transformation graph to HALT (the negative battery terminal). The probability $\Pr_\theta(e)$ is defined to be the proportion of the current flowing into HALT that flows there directly from e.

Formally, a transformational circuit model has exactly the same form as an ordinary transformation model—it is a tuple $\langle \mathsf{Events}, \textsc{Start}, \textsc{Halt}, \mathsf{Features}, \mathsf{Arcs}\rangle$— except that the arcs are undirected: they have the form $\langle \{e, e'\}, F\rangle$ rather than $\langle e, e', F\rangle$, for $e \in \textsc{Start}, e' \in \textsc{Start} \cup \{\textsc{Halt}\}$. We also replace the coaccessibility condition of §3.2.1 with the weaker condition that there must be a path from START to HALT.

Given a parameter vector θ that assigns weights to the features in $\mathsf{Features}$, we define $G_\theta : \mathsf{Arcs} \to \mathbb{R}^+$ exactly as before. $G_\theta(A) > 0$ is now called the **conductance** of the arc A. For convenience, in case the transformation graph is a multigraph, we also define the total conductance between nodes e and e':[23]

$$G_\theta(e, e') \stackrel{\text{def}}{=} G_\theta(e', e) \stackrel{\text{def}}{=} \sum_{A = \langle \{e,e'\}, F\rangle \in \mathsf{Arcs}} G_\theta(A) \;\; \geq \; 0 \tag{7.20}$$

Notice that if there are no arcs between e and e', then $G_\theta(e, e') = 0$.

[22]Other kinds of network flow (liquid, traffic) behave similarly, but not identically: the edges of such networks are usually labeled with capacity (i.e., maximum flow) rather than with resistance to flow. Assuming that we identify the flow along an arc with its probability, it is not clear that this would be a good model of transformational processes.

[23]This will let us define the total current between nodes e and e' without having to define the current separately on each arc from e to e'. Such a definition would be conceptually just as straightforward, but would require slightly awkward notation to indicate the direction of current flow.

The flow of current through the graph is defined by Ohm's Law and Kirchoff's Current Law. For each $e \in \mathsf{Events} \cup \{\textsc{Halt}\}$, let $V_\theta(e) \in [0,1]$ be a variable called the **potential** at e. Also, for each pair e, e', let $I_\theta(e,e') \in \mathbb{R}$ be a variable called the **current** from e to e'. If this value is negative, it denotes current flowing in the opposite direction.

The non-negative quantity $\max(I_\theta(e,e'),0)$ is analogous to the flow from e to e', also denoted by I_θ, that we computed in the original definition of transformation models (equation (3.18)).

Ohm's Law ($\Delta V = I \cdot R$, where R is resistance) defines the currents if the potentials are known. We rewrite it as $I = \Delta V \cdot G$ where $G = 1/R$ is the conductance:

$$I_\theta(e,e') \stackrel{\text{def}}{=} (V_\theta(e) - V_\theta(e')) \cdot G_\theta(e,e') \tag{7.21}$$

Notice that $I_\theta(e,e') = -I_\theta(e',e)$.[24]

Kirchoff's Current Law says that for all nodes e besides the source and sink, the net current flow from e is 0, i.e., the inflow balances the outflow:

$$(\forall e \notin \{\textsc{Start}, \textsc{Halt}\}) \;\; 0 = \sum_{e'} I_\theta(e,e') \tag{7.22}$$

Combining these, we obtain the system of simultaneous linear equations

$$(\forall e \notin \{\textsc{Start}, \textsc{Halt}\}) \;\; 0 = \sum_{e'} (V_\theta(e) - V_\theta(e')) \cdot G_\theta(e,e') \tag{7.23}$$

At least in the case where Events is finite, we may use standard matrix methods to solve this linear system (see footnote 12 on p. 258). There is one variable $V_\theta(e)$ for every node e in the transformation graph, and one equation for every node except the source and sink nodes. To get a single solution to the system, we add two equations that assign arbitrary potentials to those nodes (simulating a 1-volt battery):

$$V_\theta(\textsc{Start}) = 1 \tag{7.24}$$

$$V_\theta(\textsc{Halt}) = 0 \tag{7.25}$$

It follows that $V_\theta(e) \in [0,1]$ for all e.

[24] Moreover, the current $I_\theta(e,e)$ around a self-loop is 0. This makes self-loops irrelevant in a transformational circuit model; they may be pruned.

Finally, having solved for the potentials V_θ, we obtain the currents I_θ by equation (7.21), and then define

$$\Pr_\theta(e) \stackrel{\text{def}}{=} \frac{I_\theta(e, \textsc{Halt})}{\sum_{e'} I_\theta(e', \textsc{Halt})} \tag{7.26}$$

$$= \frac{V_\theta(e) \cdot G_\theta(e, \textsc{Halt})}{\sum_{e'} V_\theta(e') \cdot G_\theta(e', \textsc{Halt})} \tag{7.27}$$

Is there a probabilistic interpretation of the transformational circuit model? Yes.[25] It corresponds to exactly the same kind of Markovian random walk as the original model (see §3.3.1). The only difference is in the more complicated way that the probabilities are determined from the feature weights.

- In the original model, a random walk at node e chooses its next step from among the directed arcs leaving e. The probability of an arc A is proportionate to $G_\theta(A)$.

- In the circuit model, a random walk at node e chooses its next step from among the undirected arcs that are incident on e. The probability of an arc A is proportionate to $\max(I_\theta(A), 0)$—where $I_\theta(A)$ is determined by solving a system of equations.

  To put this another way, suppose we orient all the arcs in the direction of positive current flow.[26] The random walk then chooses its next step from among the directed arcs leaving e, and the probability of an arc A is proportionate to $I_\theta(A)$.

In fact, the analogy to the original model may be made sharper. Let us define the total flow into node e (which equals the flow out of node e unless $e \in \{\textsc{Start}, \textsc{Halt}\}$):

$$I_\theta(e) = \sum_{e'} \max(I_\theta(e', e), 0) \tag{7.28}$$

Now we can define the normalized flow as in equation (7.12):

$$\bar{I}_\theta(e) \stackrel{\text{def}}{=} I_\theta(e)/I_\theta(\textsc{Halt}) \tag{7.29}$$

$$\bar{I}_\theta(e, e') \stackrel{\text{def}}{=} I_\theta(e, e')/I_\theta(\textsc{Halt}) \tag{7.30}$$

Just as in the normalized version of the original model (§7.3.4.1), these $\bar{I}_\theta$ values denote the expected number of times that the random walk passes through e or from e to e'. And

[25]Which suggests that there will also be an EM algorithm for maximizing its parameters, although this is left to future work.

[26]Arcs with zero current may be oriented arbitrarily, or eliminated.

just as in the original model, the random-walk transition probabilities may be computed as

$$P_\theta(e, e') = \bar{I}_\theta(e, e') / \bar{I}_\theta(e) \tag{7.31}$$

Also, we may rewrite equation (7.26) analogously to equation (7.10):

$$\Pr_\theta(e) \stackrel{\text{def}}{=} \bar{I}_\theta(e, \textsc{Halt}) \tag{7.32}$$

Of these quantities, only $P_\theta(e, e')$ is simpler to compute under the original model: under the circuit model it is essentially an output, not an input, of the matrix inversion that computes I_θ.

A pleasant property of a transformational circuit model is that it has zero probability of going around a transformational cycle (of any length). This is essentially Kirchoff's Cycle Law. In our formulation, it holds because Ohm's Law (equation (7.21)) guarantees that positive current flows in the direction of decreasing potential. So if we orient the edges in the direction of positive current flow, there are no directed cycles for the random walk to traverse.

7.3.4.4 Relationship Among the Lookahead Models

It is important to recognize that in all three approaches to lookahead considered above, solving for $\Pr_\theta$ requires the same amount of effort. In each approach, the main task is to invert a single matrix whose size and sparsity are dictated by the transformation graph.

We have already discussed the relationship between the first and third approaches. The second and third are also similar. In late normalization (§7.3.4.2), G-values are "unnormalized probabilities" in $\mathbb{R}^+$, which sum in parallel and multiply in series. In transformational circuits (§7.3.4.3), G-values are conductances in $\mathbb{R}^+$, which sum in parallel and add harmonically in series.[27] In either model, we use these facts to effectively collapse all paths of the form $\textsc{Start}, \ldots e, \textsc{Halt}$ into a single composite arc A_e from $\textsc{Start}$ to $\textsc{Halt}$, and

[27] The harmonic sum of $x, y \in \mathbb{R}^+$ is defined to be $1/(1/x + 1/y)$. Note that this is less than $\min(x, y)$; so is the product of x and y if they are restricted to $(0, 1)$, as recommended in §7.3.4.2. These properties ensure that in either model, the G-value of a path decreases with the length of the path, which keeps unbounded-length paths from having unbounded G-values and hence stealing all the probability.

to determine this arc's G-value, $G_\theta(A_e)$.[28] Both models end with a normalization step in which $\Pr_\theta(e)$ is defined to be proportional to $G_\theta(A_e)$.

[28] Simply knowing how circuits combine in series or in parallel is not enough to do this collapsing, however. There are circuits such as K_4 (the complete graph on 4 vertices) that cannot be simplified just by merging parallel or series arcs. That is why it is necessary to solve a recurrence relation for either model.

Chapter 8

Additional Algorithms

This chapter resumes the exploration of algorithms for transformation models. It focuses on a few key questions:

- Do transformation models admit exact algorithms to replace the approximation algorithms of Chapter 4? (Yes.)
- Do they admit EM algorithms to replace the gradient computation of Chapter 4? (Yes, both exact and approximate ones.)
- How in general might one approach renormalizing the computed distribution $\vec{p}$—either to use a variant model that requires a normalization step (§7.3.4.1), or else to salvage probability lost during the approximation?

The chapter then treats two more topics:

- A pedagogically simpler alternative to relaxation, called propagation. Propagation may be regarded as a parallel version of relaxation. It admits relaxation as a special case although the presentation of relaxation in §4.2 is more direct.
- Some details of how to use the template approach beyond the rather narrow presentation in §4.5.3.

8.1 Exact Algorithms by Solution of Sparse Linear Systems

The approximate relaxation approach of §4.2 is not the only way to solve a transformation model or compute its gradient. It is possible to compute the objective function f and its gradient exactly. The results could be passed to any general-purpose numerical optimizer. Below we see how to compute them as efficiently as possible.

This parameter estimation strategy is practical (on a single processor) only for models of moderate size. A large and dense transformation graph makes it slow to solve equation (4.4) as required, and a large feature set implies a large parameter space to search.

8.1.1 Model Solution

§4.1.3 actually already gave an exact algorithm for solving for $\vec{p}$ (in the service of defining the objective function $\tilde{f}$). The only trouble was that it involved inversion of a matrix (equation (4.5) on p. 116). This is quite slow. And since the inverse of a sparse matrix need not be sparse, general inversion methods do not save time or space on an arbitrary sparse input.

Press et al. (1992) discuss exact $O(n^3)$-time techniques for inverting matrices and solving linear systems, such as Gaussian elimination, LU decomposition, and QR decomposition.

Fortunately, it is not actually necessary to invert the matrix. Equation (4.5) has the form $\vec{x} = \vec{b}A^{-1}$. This equation is equivalent to $\vec{x}A = \vec{b}$, which—for a single $\vec{b}$—can be solved for $\vec{x}$ without computing A^{-1}. The usual iterative solvers such as biconjugate gradient or GMRES (Greenbaum, 1997; Press et al., 1992) are suitable:

- They multiply A or A^{T} by $O(n)$ different vectors for a total time of $O(n|A|)$, giving a proportional speedup over $O(n^3)$ to the extent that A is sparse.

  ($|A|$ denotes the number of non-zero entries in A. In our application, $A = 1 - P_\theta$ where 1 denotes the identity matrix; so it is almost exactly as sparse as P_θ. Using standard sparse matrix methods, one can multiply A by any vector in time $O(|A|)$.)

- They require only $O(n)$ space rather than $O(n^2)$ or even $O(|A|)$. (One might think that storing the transformation graph would require the larger amount of space anyway, but as §§4.1.1–4.1.2 noted, often the arcs of the transformation graph need not be stored but rather can be created on demand according to the model structure.)

- Since they converge iteratively on the solution $\vec{x}$ (in the course of n iterations), they can be stopped early when some convergence criterion is met, yielding an approximate solution.

8.1.2 Computing the Gradient of the Exact Objective

We have now seen how to compute the exact objective function f. Its gradient ∇f is also straightforward to find.

Fix $h \in [1, k]$. We wish to compute $f'(\theta) \stackrel{\text{def}}{=} df(\theta)/d\theta_h$ (the h^{th} component of the gradient). Differentiating the matrix equations from §§4.1.2–4.1.4, and rearranging each one a bit to make use of quantities already computed, we obtain

$$
\begin{aligned}
(P'_\theta)_{ij} &= (P_\theta)_{ij} \cdot (F^h_{ij} - \sum_{j'} (P_\theta)_{ij'} F^h_{ij'}) && (8.1)\\
\vec{I}'_\theta &= (\vec{I}_\theta \cdot P'_\theta) \cdot (1 - P_\theta)^{-1} && (8.2)\\
\vec{p}'_\theta &= \vec{I}'_\theta \odot (P_\theta)_{\cdot 0} + \vec{I}_\theta \odot (P'_\theta)_{\cdot 0} && (8.3)\\
f'(\theta) &= -\theta_h/\sigma^2 + \sum_{i=1}^{n} s_i \frac{(p'_\theta)_i}{(p_\theta)_i} && (8.4)
\end{aligned}
$$

As in §8.1.1, equation (8.2) can be solved either by matrix inversion or with an iterative solver. It must be solved k times, once with respect to each feature weight θ_h. This means that it may be worth doing extra work to decompose, precondition, or actually invert $1-P_\theta$. The extra work will be amortized over all the features (not to mention the computation of f in equation (4.5)). This makes it worthwhile if there are many features, or if the amount of extra work is small (e.g., for a dense matrix).

A small rearrangement is helpful by analogy to the rearrangement in §4.3.5. In equation (8.1), the summation is independent of j and so can be reused for different values of j. This helps us avoid an explicit computation of P'_θ, which is not usually sparse[1] but is "almost sparse" in that each row contains many identical negative values. We can write

[1]Intuitively, why not? If t_h appears on any arc from i, then raising θ_h will affect the probabilities of *all* arcs from i. So row i of the derivative matrix P'_θ is non-zero.

(still for fixed h, and θ also fixed)

$$R_{ij} \stackrel{\text{def}}{=} (P_\theta)_{ij} \cdot F^h_{ij} \qquad \text{(still a sparse matrix)} \tag{8.5}$$

$$r_i \stackrel{\text{def}}{=} \sum_j R_{ij} \qquad \text{(expected occurrences of } t_h \text{ on a random arc from } i) \tag{8.6}$$

$$P'_\theta = R - \mathsf{diag}(\vec{r}) \cdot P_\theta \tag{8.7}$$

and therefore replace the subexpressions using P'_θ with sparser, more efficient subexpressions using P_θ, R, and r:

$$\vec{I}_\theta \cdot P'_\theta = \vec{I}_\theta \cdot R - \vec{I}_\theta \cdot \mathsf{diag}(\vec{r}) \cdot P_\theta \qquad \text{(in equation (8.2))} \tag{8.8}$$

$$= \vec{I}_\theta \cdot R - (\vec{I}_\theta \odot \vec{r}) \cdot P_\theta \tag{8.9}$$

$$(P'_\theta)_{\cdot 0} = R_{\cdot 0} - \vec{r} \odot (P_\theta)_{\cdot 0} \qquad \text{(in equation (8.3))} \tag{8.10}$$

This speeds up the computation of $f'(\theta)$ and hence of ∇f.

8.2 An Exact Expectation-Maximization Algorithm

Since the objective function $f(\theta)$ (equation (4.8)) is (proportional to the log of) a posterior probability distribution, another way to find a local maximum is the Expectation-Maximization or EM algorithm (Dempster et al., 1977). Jamshidian and Jennrich (1993) show that EM is approximately a gradient ascent algorithm that does not have to compute the gradient.

For our problem, the EM setup is as follows:

parameters The weight vector θ.

observed data The observed sample of events $\vec{s}$ (where as before s_i denotes the number of observations of event i).

hidden data For each observed event e, the random walk $\langle \textsc{Start}, \ldots, e, \textsc{Halt} \rangle$ that generated it.[2]

[2] More precisely, we are interested in the path (list of arcs) traversed by the random walk. But in the case of a graph rather than a multigraph, this can be reconstructed from the walk (list of vertices).

Like most optimization algorithms, EM begins with a guess for θ. The **expectation step** (E step) computes the posterior distribution $\tilde{P}$ over the hidden data. Then the **maximization step** (M step) reestimates θ based on this distribution (or a sufficient statistic) together with the observed data and the prior. Specifically, it chooses θ to maximize the expected log-joint-probability if the hidden data are generated from $\tilde{P}$: namely,

$$\mathbf{E}_{\tilde{P}}[\ln(\Pr(\theta) \cdot \Pr(\text{hidden}, \text{observed} \mid \theta))] \tag{8.11}$$

The E and M steps are repeated indefinitely to continue improving the estimate of θ.[3]

We now develop the E and M steps for transformation models.

8.2.1 The Maximization Step

The maximization step will depend on existing methods for fitting log-linear models (also known as maximum-entropy models or Gibbs distributions) to data. We quickly review what such methods do, then see how they help us.

As §3.2.2 remarked, a transformation model makes use of a conditional log-linear model of arc probabilities. The events of this subsidiary model are the arcs of the transformation graph. For each arc ij (where i and j are event numbers as in §4.1), the parameter vector θ and equation (4.2) give us the conditional probability of ij given i, denoted $(P_\theta)_{ij}$.

Suppose we have actually observed a collection of arcs (the hidden data). The Improved Iterative Scaling (IIS) algorithm of (Pietra et al., 1997; Berger et al., 1996; Berger, 1997) would let us fit a conditional log-linear model to the collection. The algorithm chooses θ that globally maximizes the log-conditional-likelihood of the collection,

$$\sum_{ij} \#(ij) \ln(P_\theta)_{ij} \tag{8.12}$$

where $\#(ij)$ is the number of observations of ij in the collection.[4]

[3] This formulation of EM serves to maximize $\Pr(\text{observed} \mid \theta) \cdot \Pr(\theta)$, which is what we want. More commonly EM is presented as a scheme for maximizing just $\Pr(\text{observed} \mid \theta)$, in which case the factor $\Pr(\theta)$ is left out of equation (8.11) as well.

[4] The term "conditional likelihood" (§3.1.5) is properly used in place of "likelihood" because $(P_\theta)_{ij}$ is not the probability of arc ij, but rather the probability of ij conditioned on i. While IIS famously maximizes this value, it is usual to formulate it as the equivalent problem of maximizing the quantity $\sum_{ij} \tilde{p}(ij) \ln(P_\theta)_{ij}$ where $\tilde{p}$ is the observed probability of ij in the collection rather than its count in the collection. This simply divides through by the collection size. We cannot do this division (yet) because (a) the collection size is

Even better, Chen and Rosenfeld (1999), developing an idea of John Lafferty's, extend IIS to incorporate our Gaussian prior on θ (§3.5). Their version of IIS maximizes the log-posterior conditional probability of θ,

$$
\begin{aligned}
& \ln \Pr_{\text{prior}}(\theta) + \sum_{ij} \#(ij) \ln(P_\theta)_{ij} & (8.13)\\
= & -\sum_{h=1}^{k} \theta_h^2 / 2\sigma^2 + \sum_{ij} \#(ij) \ln(P_\theta)_{ij} & (8.14)
\end{aligned}
$$

We can use Chen and Rosenfeld's method for our EM maximization step. In this case, the collection of arcs is hidden (even its size is unknown). But the expectation step has already provided a distribution $\tilde{P}$ over possible collections, and the maximization step requires us to choose θ maximizing

$$
\begin{aligned}
& \mathbf{E}_{\tilde{P}}[\ln \Pr_{\text{prior}}(\theta) + \sum_{ij} \#(ij) \ln(P_\theta)_{ij}] & (8.15)\\
= & \ln \Pr_{\text{prior}}(\theta) + \sum_{ij} \underbrace{\mathbf{E}_{\tilde{P}}[\#(ij)]}_{\text{called } N_{ij}} \ln(P_\theta)_{ij} & (8.16)
\end{aligned}
$$

Since $N_{ij} \stackrel{\text{def}}{=} \mathbf{E}_{\tilde{P}}[\#(ij)]$ is just a non-negative real number, the problem has the same form as the one treated by Chen and Rosenfeld, and their modified IIS algorithm applies.

8.2.2 Details of the Maximization Step

Let us now spell out the details. The IIS algorithm requires us first to compute, for each feature t_h, the "observed" feature count $\sum_{ij} N_{ij} F_{ij}^h$. (The shudder quotes around "observed" are because we are using IIS as a step in EM, so that for us the arc and feature counts are inferred, not observed.) This is easy once we have obtained N_{ij} as described in the next section.

Then as IIS iteratively improves θ, we must repeatedly compute, for each feature t_h, the predicted feature count $\sum_{ij}(N_i(P_\theta)_{ij})F_{ij}^h$. This is the predicted count given both the new value of θ, which changes from iteration to iteration, and the "observed" vertex counts

not a harmless constant in our case but is itself a hidden variable, and (b) we are about to add a prior term that will need to be divided as well. (If we scaled the likelihood to sample size 1 before adding a prior, large samples would have no more ability than small ones to overcome the prior!)

$N_i \stackrel{\text{def}}{=} \sum_j N_{ij}$, which stay the same.[5] The predicted feature counts are straightforward to compute, but require $O(|F|)$ time *per iteration* of IIS, where $|F|$ denotes the total size of the model.

If different arcs can have different numbers of features, things are slightly more complicated. For each t_h and each integer $r \in$ [1,maximum number of features per arc], we must accumulate the r^{th} predicted feature count of t_h, defined by $\sum_{ij \text{ having } r \text{ features}} (N_i (P_\theta)_{ij}) F_{ij}^h$. All these counts can still be computed in total time $O(|F|)$.

IIS tries to modify θ so that the predicted feature counts match the "observed" ones. Each iteration of IIS simply computes the predicted feature counts given the current θ, and then increases each θ_h by the δ_h that solves the equation[6]

$$(\text{“observed” count of } t_h) = \frac{\theta_h + \delta_h}{\sigma^2} + \sum_r (r^{\text{th}} \text{ predicted count of } t_h) \cdot \exp(r\delta_h) \qquad (8.17)$$

Solving this equation is simple because it is equivalent to finding the root of a monotonically increasing function of δ_h.[7]

Notice the effect of the Gaussian prior: it acts to push θ_h back toward zero, the Gaussian mean. For example, suppose that IIS has converged, so that equation (8.17) holds for $\delta_h = 0$. If we now strengthen the prior by decreasing σ^2, then to bring equation (8.17) back to equality, we must make δ_h negative if θ_h is positive, or vice-versa, ensuring that on the next update θ_h will be pushed toward zero.

It is evident from Chen and Rosenfeld's derivation that their technique can actually be applied with any convex prior, including the alternatives considered in §7.3. Simply modify equation (8.17) by replacing $(\theta_i + \delta_i)/\sigma^2$ with $-\frac{\partial}{\partial \delta_i}(\ln \Pr_{\text{prior}}(\theta_i + \delta_i) - \ln \Pr_{\text{prior}}(\theta_i))$.

Since equation (8.16) (with any convex prior) is a convex function in θ, there are many other methods besides IIS that will easily maximize it. One could use hill-climbing methods

[5]The counts N_i are used because the conditional log-linear model of arc probabilities is conditioned on the originating vertex i. That is, in equation (8.16), $(P_\theta)_{ij}$ is the probability of arc ij given vertex i.

[6]The usual formulation divides the equation through by $N \stackrel{\text{def}}{=} \sum_{ij} N_{ij}$, so that the counts are replaced by conditional probabilities. (See footnote 4.) Interestingly, Chen & Rosenfeld neglect to divide the prior by N, so their choice of σ^2 is specific to a sample size.

[7]Solving the equations usually takes insignificant time even with a naive method like interval bisection; what is slow is computing the predicted counts. But since the function is locally exponential (it is a sum of an affine function and several exponentials, one of which dominates locally), a nice choice is Ridder's method (Press et al., 1992), which does repeated exponential interpolation (and converges in one step for a function of the form exponential × affine). Like Newton-Raphson, Ridder's method has superlinear convergence.

such as conjugate gradient, or a combination of such methods with IIS (Lau et al., 1993).

8.2.3 The Expectation Step

Now for the expectation step. In principle, this step produces a distribution $\tilde{P}$ over assignments of paths to the observations. However, we have just seen that the maximization step requires only a sufficient statistic from this distribution: the expected count $N_{ij} \stackrel{\text{def}}{=} \mathbf{E}_{\tilde{P}}[\#(ij)]$ of each arc ij. So we only need to compute these expected counts.

When we run the expectation step, θ is fixed. Suppose we observe an event m. How many times do we expect to have traversed ij in the path that generated m?

To answer this, consider the set $\mathcal{W}_m$ of all random walks that could have resulted in m. These are walks of the form $W = \langle 1, \ldots, m, 0 \rangle$ (recall that $1 = $ START and $0 = $ HALT). Their total probability $\sum_{W \in \mathcal{W}_m} \Pr(W)$ equals $(p_\theta)_m$ by definition. Given that we observed m, the posterior probability that we took each W is $\Pr(W)/(p_\theta)_m$, so the expected number of times that we traversed ij is

$$\sum_{W \in \mathcal{W}_m} \frac{\Pr(W)}{(p_\theta)_m} \cdot (\text{occurrences in } W \text{ of arc } ij) \tag{8.18}$$

Suppose for a moment that we have computed the inverted matrix discussed in the previous sections,

$$\begin{aligned} Q &\stackrel{\text{def}}{=} A^{-1} = (1 - P_\theta)^{-1} && (8.19)\\ &= 1 + P_\theta + P_\theta^2 + P_\theta^3 + \cdots, && (8.20) \end{aligned}$$

Then we could efficiently find the (possibly infinite) sum over paths W. The key is that since P_θ is a matrix of transition probabilities, Q_{rs} is the total probability of all paths from r to s. (For example, we have already seen that the flow $(I_\theta)_s$ is Q_{1s}.) This total may exceed 1 since the paths are not mutually exclusive: some are prefixes of others.

The total probability of all paths of the form $\langle 1, \ldots i, j, \ldots, m, 0 \rangle$ is

$$Q_{1i} \cdot (P_\theta)_{ij} \cdot Q_{jm} \cdot (P_\theta)_{m0} \tag{8.21}$$

This total may also exceed 1. This is not because the paths are prefixes of one another: they cannot be, since all of them end with 0 but none of them can pass through 0. Rather, it is because we are multiply counting a path that traverses ij multiple times and so matches the template $\langle 1, \ldots i, j, \ldots, m, 0\rangle$ in multiple ways. This is what we want. It means that the above expression computes $\sum_{W \in \mathcal{W}_m} \Pr(W)$(occurrences in W of arc ij). Dividing by $(p_\theta)_m$ obtains the expected number of traversals given m, as required by equation (8.18).

The above discussion considered a single observation of m. We now sum over all observations of all events to obtain our desired N_{ij}. Recall that s_m denotes the number of observations of m.

$$\begin{aligned}
N_{ij} &\stackrel{\text{def}}{=} \text{expected traversals of } ij \text{ given that we observed } \vec{s} && (8.22)\\
&= \sum_m s_m \cdot \text{expected traversals of } ij \text{ given that we observed } m && (8.23)\\
&= \sum_m s_m \cdot \frac{Q_{1i} \cdot (P_\theta)_{ij} \cdot Q_{jm} \cdot (P_\theta)_{m0}}{Q_{1m} \cdot (P_\theta)_{m0}} && (8.24)\\
&= Q_{1i} \cdot (P_\theta)_{ij} \cdot \sum_m Q_{jm} \frac{s_m}{Q_{1m}} && (8.25)\\
&= (I_\theta)_i \cdot (P_\theta)_{ij} \cdot (Q \cdot \vec{\hat{s}})_j \qquad \text{where } \hat{s}_m \stackrel{\text{def}}{=} \frac{s_m}{Q_{1m}} = \frac{s_m}{(I_\theta)_m} && (8.26)\\
\text{equivalently, } N &= \mathsf{diag}(\vec{I_\theta}) \cdot P_\theta \cdot \mathsf{diag}(Q \cdot \vec{\hat{s}}) && (8.27)\\
\text{exception: } N_{i0} &= s_i \ \ \text{(obvious)} && (8.28)
\end{aligned}$$

(The exception arises because if $j = 0$, then we cannot write the paths of interest in the form $\langle 1, \ldots i, j, \ldots, m, 0\rangle$ as assumed: the ij arc is the same as the $m0$ arc.)

We now see that, once again, we can avoid the computation and storage of the inverted matrix $Q = (1 - P_\theta)^{-1}$. Equation (8.26) does not use all of Q, but only the column vector $Q \cdot \vec{\hat{s}}$. This can be found by solving the single equation $(1 - P_\theta) \cdot \vec{x} = \vec{\hat{s}}$ for x, as usual using a sparse matrix solver (see §8.1.1). This is more efficient than matrix inversion, provided that the transformation graph (and hence P_θ) is indeed sparse.

In equation (8.24), one can regard Q_{1i} as a "forward probability" and $Q_{jm} \cdot (P_\theta)_{m0}$ as a "backward probability." In contrast to the forward-backward algorithm for (ϵ-free) HMMs (Baum, 1972), however, the forward and backward paths through the transformation graph are constrained by very few observed data (only 1, m, and 0) and have potentially unbounded length. This is why the probabilities require more computation.

Eisner (2001) gives a very general method of performing expectation steps for probabilistic finite-state recognizers and transducers. The above EM algorithm is a special case of that method. However, the special case has one major advantage over the general method: instead of solving one linear system per observed event, it combines the observed events into the vector $\hat{s}$ and solves a single linear system. This is possible because the observed events only constrain the very ends of the available paths, so most of the computation is independent of the observed events. In terms of §7.1.2's interpretation of a transformation model as a finite-state machine, the transformation graph uses only ϵ-transitions except when transitioning to HALT.

8.2.4 Variations on EM

There are several possible variations on the above EM algorithm:

GEM EM is especially slow here because it involves a nested loop. The outer loop alternates between E and M steps. The inner loop is the IIS algorithm that constitutes the M step. However, it is not necessary to run IIS to completion. The **Generalized EM** algorithm (Dempster et al., 1977) only requires the M step to increase equation (8.16) if possible, not necessarily to maximize it. Riezler (1998; 1999) suggests running only a single iteration of IIS during the M step, and proves that this yields a GEM algorithm, which he calls **Iterative Maximization**.[8]

incremental A common variation on EM is to run it incrementally. See (Neal and Hinton, 1998) for a theoretical justification, and §1.2.4 for a possible motivation from language learning in children.

Viterbi A common way to speed up the E step is to use the so-called Viterbi or winner-take-all approximation to $\tilde{P}$, in which the most likely hidden path for an observed event is assumed to be the one that did generate it. This is indeed faster here. A single run of Dijkstra's (1959) shortest-path algorithm on the transformation graph, taking an arc's cost to be its negative-log-probability, will find the most probable paths from START to all vertices. (This also provides a good first guess for $\vec{I}_\theta$ when iteratively solving equation (4.5).)

[8] One would have to adapt this proof for our slightly more complicated case, which involves a conditional model and a prior.

When implemented in its simplest form, with a binary heap, Dijkstra's algorithm takes time $O(|\mathsf{Arcs}| \log |\mathsf{Events}|)$ (Cormen et al., 1990)—in contrast to sparse matrix solution, which takes time $O(|\mathsf{Arcs}| \cdot |\mathsf{Events}|)$. It can be sped up further to $O(|\mathsf{Arcs}| + |\mathsf{Events}| \log |\mathsf{Events}|)$ by using a Fibonacci heap, but at the cost of a larger constant factor.

After running Dijkstra's algorithm, it is necessary to follow backpointers from the observed events back to START in order to collect the arc counts N_{ij} from these paths. This does not affect the asymptotic complexity. With a suitable marking scheme it is possible to avoid traversing any arc twice when doing this, so that the time requirement is proportional to the size of the union of all the best paths to the observed events (which may be few or short), and is therefore $O(|\mathsf{Arcs}|)$, often better.[9]

The Viterbi approximation accentuates the "winner-take-all" effect in EM's search for a local maximum. A transformation whose probability is initially low will tend not to appear on the best paths at all, so it is hard to find evidence to increase its probability. Its best hope for survival is for other transformations sharing its features to appear on best paths.

For a better approximation to $\tilde{P}$, one might modify this approach to use the $K > 1$ best paths to each observed event, weighted in proportion to their probabilities. These can be computed efficiently (Eppstein, 1998).

confidence-weighted Some recent successful unsupervised learning approaches resemble EM, but do not use all the data at each iteration. Instead they use an observed datum only if its corresponding hidden datum can be reconstructed "confidently."[10]

[9] For example, use two passes. All vertices are initially white. The first pass follows the best path from each observed vertex m backwards toward START, flipping the vertex colors to black. If it encounters a vertex that is already black, it increments that vertex's "pending count" (initially 0) by s_m and quits the path. The second pass follows exactly the same paths in the same order, flipping vertices back to white, and aborting a path if it encounters a vertex that is already white. While following the path from m, the second pass keeps a running total T_m of s_m plus the pending counts that it encounters (and resets to 0) along the way; as it traverses an arc ij it increments N_{ij} (initially 0) by the current total T_m.

[10] The confidence threshold should be lowered, temporarily or permanently, if the algorithm stops making progress. When the threshold reaches zero, the passes become identical to EM and the algorithm stops at a true local maximum, which is the primary (if weak) argument for standard EM.

Note that rather than thresholding the observed data with a strict cutoff, one might instead reweight

This might mean that the current model assigns high probability to the observed datum, or the distribution $\tilde{P}(\cdot \mid \text{observed datum})$ over the hidden datum given the observed datum is sharply biased toward one reconstruction (Hearst, 1991; Hindle and Rooth, 1993; Brill, 1995; Yarowsky, 1995), or that different classifiers agree on the reconstruction (Blum and Mitchell, 1998), or that the reconstruction of this hidden datum is expected to be accurate for some other reason (Melamed, 1997; Osborne, 2000). The idea is to avoid drowning the signal to the M step in the noise of many uncertain examples from the E step. The theory is that the noise moves the estimate of θ to a rather random part of the space with poor local maxima, and it is better to extend the base of training data conservatively from a small, accurate seed.

The notion of child language learning in §1.2.4 has the same idea: that children ignore adult language that they cannot make head or tail of. They learn primarily from sentences (or parts of sentences) "at the edge of their competence," those that they find surprising but for which they can settle on a clearly best interpretation in light of their current grammar and the discourse context. (1980) used a similar "failure-driven learning" approach to grow the ruleset of a deterministic parser. Barg and Walther (1998) apply a similar technique to learning a lexicalized (HPSG) grammar, and likewise Solomon and Wood (1994) (somewhat more statistically) for learning a lexicalized (categorial) grammar.

8.3 An Approximate EM Algorithm via Back-Relaxation

In Chapter 4, we used back-relaxation to find the gradient of the approximate objective function $\tilde{f}$ found by relaxation. An alternative use for back-relaxation is to optimize $\tilde{f}$ not with a gradient-based method, but by Expectation-Maximization. In fact there is a tight connection between the two computations.

them according to *how* confidently the corresponding hidden data can be reconstructed.

Things become slightly more interesting if the hidden data are complex. One may then need to salvage confident *portions* of a reconstruction. For example, it may be that only part of a parse, or only the end of a transformational path, can be reconstructed confidently.

8.3.1 The New E Step

§8.2 already described an EM algorithm to optimize the exact objective function $f(\theta)$. For EM to be applicable to $\tilde{f}(\theta)$ instead, the function $\tilde{f}(\theta)$ must be a posterior (log-)probability distribution. We can indeed regard it as such (even though it was originally constructed as a deficient approximation of another distribution). The idea is that a random walk that has not halted anywhere, by the time relaxation has stopped, is considered to have halted at a special event ∞. Adding ∞ to the event space in this way accounts for any missing probability while ensuring that the M step keeps the form in §8.2.1. Of course, ∞ is never observed in training data. The EM algorithm tries to maximize the likelihood of the events that *were* observed, at the expense of other events such as ∞.

Suppose $\vec{s}$ is an observed sample from this model (with s_i being the number of observations of i and $s_\infty = 0$). Then EM can be used to maximize the posterior probability of θ given $\vec{s}$, just as described in §8.2. The only difference is in the E step (§8.2.3), which is defined very simply using a quantity computed during back-relaxation. For given θ, the expected number of times we traverse arc kj on time step t turns out to be just

$$N_{kj}^{(t)} = P_{kj}^{(t)} \cdot \mathtt{g}(P_{kj}^{(t)}) \tag{8.29}$$

as derived in the next section. This is easy to compute during back-relaxation, since $\mathtt{g}(P_{kj}^{(t)})$ (equation (4.23)) is just the value added to $(\mathtt{g}P)_{kj}^{(t-1)}$ during step t of ordinary back-relaxation (equation (4.24)). It is non-zero only if $k = i$, the vertex being relaxed.

As discussed in §8.2.2, the maximization step only needs to know the expected number of times each vertex i and each feature t_h is "observed" during traversal. So when we compute $N_{ij}^{(t)}$ from $\mathtt{g}P_{ij}^{(t)}$ during back-relaxation, we do not need to store it. We merely increment the "observed" count of each feature t_h by $N_{ij}^{(t)} F_{ij}^{h}$. (This takes a loop over just the features on arc ij; there is no need to consider the features of competing arcs ij' as in §4.3.5, making feature counts for EM easier to accumulate than feature weight gradients for gradient descent.) We also increment N_i by $N_{ij}^{(t)}$.

As a side remark, the EM-gradient connection is also useful for the more general problem of optimizing the parameters of probabilistic finite-state machines (see §7.1.2). Eisner (2001) notes that the EM algorithm it presents for finite-state machines could be adapted to find the gradient, allowing one to optimize a machine's parameters by either EM or conjugate gradient. If one computes the approximate likelihood of the training data by relaxation (Mohri, 2000), then one should compute the *exact* gradient of this approximate likelihood by back-relaxation.

8.3.2 Conceptual Derivation of the Back-Relaxation Formula for EM

This section gives a quick intuitive justification of equation (8.29). (A more formal derivation is postponed to §8.5.6, where we will have some more notation available; that derivation exposes a connection to the Baum-Welch EM algorithm for hidden Markov models.)

Let us suppose that $\vec{s}$ consists of a *single* observation, of some event $m \neq \infty$. By definition, $p_m^{(T)}$ is the total probability of all paths from START that halt at m. Also by definition, $N_{kj}^{(t)}$ denotes the fraction of that total that is derived from paths whose t^{th} arc is kj. Multiplying $P_{kj}^{(t)}$ by $1+\epsilon$ would multiply the computed probability of those paths by $1+\epsilon$, and thereby multiply the total $p_m^{(T)}$ by $1+\epsilon N_{kj}^{(t)}$. This implies

$$\frac{\partial p_m^{(T)}}{\partial P_{kj}^{(t)}} = \frac{N_{kj}^{(t)} p_m^{(T)}}{P_{kj}^{(t)}} \tag{8.30}$$

since it says (rephrasing the multiplicative increases as additive ones) that increasing $P_{kj}^{(t)}$ by $\epsilon P_{kj}^{(t)}$ would increase $p_m^{(T)}$ by $\epsilon N_{kj}^{(t)} p_m^{(T)}$.

Therefore

$$\mathsf{g}(P_{kj}^{(t)}) = \mathsf{g}p_m^{(T)} \cdot \frac{\partial p_m^{(T)}}{\partial P_{kj}^{(t)}} = \frac{1}{p_m^{(T)}} \cdot \frac{N_{kj}^{(t)} p_m^{(T)}}{P_{kj}^{(t)}} \tag{8.31}$$

and rearranging yields equation (8.29) as desired. Notice that the logarithm used in our particular choice of objective function gave us EM's normalizing factor of $1/p_m^{(T)}$ "for free."

While the above derivation assumes that $\vec{s}$ is a single observation, we can extend to general $\vec{s}$ by regarding all three variables in equation (8.29) as functions of $\vec{s}$. Conceptually, N_{kj} and $\mathsf{g}(P_{kj}^{(t)})$ are then linear functions (the latter because $\tilde{f}$ is linear plus a constant), while $P^{(t)}$ is a constant, so we can simply add together single-observation versions of equation (8.29) to get a multiple-observation version.

8.4 On Renormalization of $\vec{p}$

In general, our computed probability distribution over events ($\vec{p}$) may not sum to 1. This may happen for any of several reasons:

1. We halt the relaxation algorithm when some vertices $i \neq 0$ still remain to be relaxed (i.e., $J_i \neq 0$). See §4.2.2.

2. For efficiency, we ignore some arcs that do not contribute much to the objective function. See §4.5.2.

3. We use a late normalization model (§7.3.4.2), which requires a final normalization step.

4. We use a perturbed model (§3.9), which requires a final normalization step (§3.9.2).

Sometimes it is desirable to renormalize the distribution $\vec{p}$ so that it sums to 1, using the factor Z provided for this purpose in equation (4.8). It is always necessary for items 3–4 above. For items 1–2, it is generally sensible to renormalize during testing of the model, modulo the caveat in §8.4.4 below. But during training, it may be better not to correct for items 1–2 above, lest one create perverse incentives when fitting the model.[11]

8.4.1 A Unified Approach to Renormalization

There exists a unified way of thinking about renormalization. It stems from the observation that $\vec{p}$ always *will* sum to 1 if two conditions are met:

- Relaxation is permitted to run to completion (i.e., to empty the queue) when computing $\vec{p}$.

[11] That is, perverse incentives that depend on the particular nodes and arcs used during the run of relaxation. For example, suppose vertex i happens not to have been relaxed by the time the relaxation algorithm halts. Then a renormalized version of the objective function will reallocate vertex i's probability mass J_i by scaling up $\vec{p}$ uniformly; such an objective function gives the learner an incentive to propagate a lot of probability to vertex i so that it can be usefully reallocated! By contrast, optimizing an unnormalized objective function gives the learner an incentive to propagate to vertices that will have a direct effect on $\tilde{f}$, rather than to vertices like i.

- Every row in the transition probability matrix P sums to 1 (other than row 0, representing HALT). Even if P changes during the course of relaxation (§4.3.6), it must always have this property.

The observation holds because if the row sums of P are 1 at every step of relaxation, then the relaxation algorithm (§4.2.1) is easily seen to have the following invariants, where J_0 is defined as in footnote 7 on p. 120:

$$\sum_{i=0}^{n} J_i = 1 \tag{8.32}$$

$$\sum_{i=1}^{n} p_i = J_0 \tag{8.33}$$

So if also relaxation is allowed to run to completion, then the probability distribution will be normalized:

$$\sum_{i=1}^{n} p_i = J_0 = 1 - \sum_{i=1}^{n} J_i = 1 - \sum_{i=1}^{n} 0 = 1 \tag{8.34}$$

(The unnormalized distributions in conditions 2–4 of §8.4, above, arise because the row sums of P are *not* 1. In 2, some elements of P have been zeroed out so that the rows no longer sum to 1. In 3, the model is defined without normalization of the rows. Finally, 4 may be treated as a special case of 3: the perturbation parameter π_i has precisely the effect of multiplying row i by $\exp \pi_i$.[12])

[12]This section therefore covers two of the three kinds of "transformational lookahead" model described in §7.3.4: late-normalization and perturbed models. For completeness, this footnote describes how to work with the third, transformational circuit models. In principle, a transformational circuit model is just like an ordinary transformation model, but with an unusual parameterization of P_θ, since §7.3.4.3 noted that a circuit model can be interpreted as a random walk. A more direct way to solve a circuit model is by setting

$$(H_\theta)_{ij} \stackrel{\text{def}}{=} \begin{cases} 1_{ij} & \text{if } i \in \{0,1\} \\ (G_\theta)_{ij} & \text{if } i \notin \{0,1\} \text{ and } i \neq j \\ (G_\theta)_{ij} - \sum_{j'} (G_\theta)_{ij'} & \text{if } i \notin \{0,1\} \text{ and } i = j \end{cases} \tag{8.35}$$

where G_θ is symmetric because a transformational circuit model F_{ij}^h is defined to be symmetric with respect to i, j. Now equations (7.23) and (7.24) become

$$H_\theta \cdot \vec{V} = \vec{b} \tag{8.36}$$

and the rest of the model solution is straighforward as described in §7.3.4.3. As for the gradient, a partial of $\vec{V}$ (with respect to θ_h, say) is found similarly to equation (8.2): differentiating both sides of equation (8.36),

$$H_\theta \cdot \vec{V}' = -H_\theta' \cdot \vec{V} \tag{8.37}$$

where H_θ' can be easily found by reducing it to G_θ', and then the linear equation can be solved for $\vec{V}'$. As usual, one can use iterative methods for solving the sparse linear systems (§8.1.1).

This observation leads to the following two tricks that can be used together for normalization:

- Make sure to run relaxation to completion. This may require changing P toward the end of relaxation so that the queue will empty. (Or one can use an exact method in place of relaxation: see §8.1.1.)

- Modify P as necessary so that the row sums are 1. This can always be done locally by normalizing the rows; or if the row sum is less than 1, by adding to column 0 (the probability of halting).

  A more interesting way to make the row sums 1 is to introduce a special new event ∞ (also mentioned in §8.3.1). $P_{i\infty}$ is defined to pick up the slack so that row i sums to 1: $\left(\sum_{j=0}^{n} P_{ij}\right) + P_{i\infty} = 1$. Unlike other entries of P, $P_{i\infty}$ can be negative.

The event ∞ is used to figure a *global* renormalizing constant Z. Let us say that ∞ itself has an arc of probability 1 to HALT (i.e., $P_{\infty 0} = 1$) and no other out-arcs. Modifying equation (8.34) to account for the existence of this extra event, we see that $(\sum_{i=1}^{n} p_i) + p_\infty = 1$ if relaxation is run to completion. So the normalizing constant $Z \stackrel{\text{def}}{=} \sum_{i=1}^{n} p_i$ can be expressed as $1 - p_\infty$.[13]

8.4.2 Global Renormalization and the Gradient Computation

Renormalizing $\tilde{f}$ affects the gradient $\nabla\tilde{f}$. Fortunately, the presentation above requires almost no changes to the back-relaxation computation.

Back-relaxation will automatically compute the effect on the objective function of paths that end at ∞. It is only necessary to extend equation (4.10)'s definition of $\mathrm{g}p_\infty$ (found by differentiating equation (4.8)):

$$\mathrm{g}p_i = \begin{cases} s_i/p_i & \text{if } i \neq \infty \\ \frac{1}{Z}\sum_{i=1}^{n} s_j & \text{if } i = \infty \end{cases} \tag{8.38}$$

As usual, back-relaxation must compute $\partial P_{ij}^{(t)}/\partial\theta_h$. While the basic case was given in

[13]For uniformity, one can also use p_∞ to deal with flow adders: in §4.4, just subtract from p_∞ instead of adding to Z.

§4.3.5, redefining P inevitably requires changes. For example, if $P_{i\infty} \stackrel{\text{def}}{=} 1 - \sum_{j=1}^{n} P_{ij}$, then

$$\frac{\partial P_{i\infty}^{(t)}}{\partial \theta_h} = - \sum_{j=1}^{n} \frac{\partial P_{ij}^{(t)}}{\partial \theta_h} \tag{8.39}$$

8.4.3 A Global Renormalization Example

As an example of all this, suppose we have a late normalization model. Introduce a new event ∞ and use it to correct P so that rows sum to 1, as above. Run relaxation until some stopping criterion is met.

Now it is necessary to relax all remaining vertices on the queue. To ensure that each vertex will only have to be relaxed one more time, first change P so that the only arcs of non-zero probability go to either 0 (i.e., HALT) or ∞. Then when vertex i is relaxed for the final time, its unpropagated probability mass will either be counted toward p_i (via arc $i0$) or toward renormalizing all of $\vec{p}$ (via arc $i\infty$). The vertices on the queue can now be relaxed in any order, with ∞ relaxed last, completing the algorithm.

Suppose in particular that P is set for this final relaxation such that the only arcs go to ∞ (and these have probability 1). Then this final relaxation has the simple effect of adding all the unpropagated mass on the queue, $\sum_i J_i$, to p_∞ and then subtracting this total from Z.

8.4.4 A Caveat About Global Renormalization

It is important that during testing, all probabilities be determined without knowledge of the test data. In particular, the renormalization constant $Z = 1 - p_\infty$ should be so chosen. Otherwise the test would be unfair.

For example, in the two-stage relaxation used in the experiments (§6.5.3), the second stage of relaxation was halted just when all test vertices had been relaxed. It would not have been fair to take any unpropagated mass left on the queue and subtract it from Z, as in the example §8.4.3. Rather, §6.5.3's policy was that the unrelaxed vertices left on the queue could have been relaxed if the test data had been different. It is only when a vertex is left unrelaxed (or an arc is ignored) for reasons *independent of the test set* that it is fair to count its probability mass as lost, and redirect that mass to something else.

It was also important in §6.5.3 that the matrix P used in the second stage of relaxation did not have (or need) any arcs to ∞. Since the test set determined how long the second stage ran, it would have influenced how many of the arcs to ∞ were actually used during relaxation, thereby unfairly affecting the normalization factor again.

8.5 Propagation: A Simpler Variant of Relaxation

There is a close connection between transformation models and recurrent neural networks (§7.1.3). As a result, Chapter 4's back-relaxation algorithm for transformation models is similar in spirit to the well-known back-propagation algorithm for neural networks (Werbos, 1974; Le Cun, 1985; Rumelhart et al., 1986). Both compute the gradient of a computed function by running the computation backwards, keeping track of how the parameters influence the current state and how the current state influenced the final result.

In fact, it is possible to formulate genuine propagation and back-propagation algorithms for transformation models. These are alternatives to relaxation and back-relaxation, which we develop here.

The main reason to mention these additional algorithms is expository. Propagation is not as flexible or adaptive a strategy as relaxation. However, it has a simple motivation, notation, and implementation; everything can be described in terms of operations on sparse matrices. The simplicity of the approach may also make its empirical behavior easier to analyze.

Propagation does have at least one substantative advantage. As a matrix algorithm, it can be efficiently implemented on parallel processors. Indeed, propagation is inherently a parallel algorithm. It can be regarded as a version of relaxation in which step t relaxes all vertices $1, 2, \ldots n$ simultaneously, rather than choosing a single vertex $i^{(t-1)}$ to relax.

8.5.1 The Propagation Idea

All of the exact algorithms in §§8.1–8.2 put $A \stackrel{\text{def}}{=} (1 - P_\theta)$ and make use of the inverted matrix $Q \stackrel{\text{def}}{=} A^{-1}$. They require us to evaluate products of the form $\vec{b}Q$ or $Q\vec{b}$. Even without finding Q explicitly, such a product is in general slow to compute exactly. An iterative solution by biconjugate gradient (§8.1.1) requires $O(n)$ $A\times$vector multiplications, for a total runtime of $O(n|F|)$, or $O(|F| + n|P_\theta|)$ if we have space to store P_θ for reuse. This is impractical for models of the size we consider in Chapter 6.

The propagation approach simply follows equation (8.20) and expands[14]

$$Q = 1 + P_\theta + P_\theta^2 + \cdots \tag{8.40}$$

$$\vec{b}Q = \vec{b} + \vec{b}P_\theta + (\vec{b}P_\theta)P_\theta + \cdots \tag{8.41}$$

and use only the first (say) T terms of equation (8.41), where T is a small constant. This approximation, $\tilde{Q}$, can be computed with a total of only $T-1$ $P_\theta\times$vector multiplications, for a total runtime of $O(T|F|)$, or $O(|F| + T|P_\theta|)$ if we have space to store P_θ for reuse. Moreover, if $\vec{b}$ is sparse (as in equation (4.5)), then at least the first few of these multiplications involve sparse vectors.

While simple, propagation shares with relaxation many of the advantages over the linear-system-solution methods of §8.1:

- It gives appropriate answers even if $1 - P_\theta$ is not actually invertible (for example, because the transformation graph contains cycles of probability 1).

  After all, the role of Q throughout this chapter is that Q_{ij} gives the total probability of all paths from i to j (which are not in general disjoint). This is more directly and generally stated by using equation (8.40), rather than a matrix inverse, to define Q.

- Unlike iterative solvers, it is guaranteed to return appropriate results. Its approximation cannot lead to negative approximations of the flows I_i, probabilities p_i, or arc counts N_{ij}, or to approximations of p_i that sum to more than 1. (See §4.1.6.)

- It is easy to run even if n is intractably large or infinite. (It explores as much of the transformation graph as it can reach from START within $T-1$ steps. Other methods could also be confined to this subgraph, but propagation automatically finds the relevant subgraph.)

[14] This expansion of Q converges only if $\lim_{t\to\infty} P_\theta^t = 0$. One might expect that to be false on the grounds that P_θ is a Markovian matrix, but actually it is not Markovian: its 0^{th} row is 0 since there are no transitions from HALT. Indeed, the coaccessibility property of (§3.2.1) guarantees that row i of P_θ^t will converge to 0 if i is accessible from START. So the expansion converges if all vertices are accessible. Even if not, equation (8.41) remains valid provided that the vertices $\{i : b_i \neq 0\}$ are accessible.

- The gradient of the approximated objective function $\tilde{f}$ can be computed exactly (by back-propagation).[15]

- This computation of $\nabla\tilde{f}$ is far faster than §8.1.2's component-by-component computation of ∇f (although the latter could be restructured).

8.5.2 Understanding Propagation

Propagation has a simple interpretation in the transformation graph. As noted at equation (8.20), Q_{ij} is the total probability of all paths from i to j. We have expanded it as $P_\theta^0 + P_\theta^1 + P_\theta^2 + \cdots P_\theta^{T-1}$, where $(P_\theta^t)_{ij}$ is the total probability of just the length-t paths from i to j.

Assuming that the start vector $J_0 = \vec{b} = \langle 0, 1, 0, 0, \ldots \rangle$ (equation (4.3)), the approximation to $\vec{I} = \vec{b}Q$ therefore sums the probabilities of just the length $< T$ paths from START. This yields a probability distribution $\vec{p} = \vec{I} \odot (P_\theta)_{\cdot 0}$ that considers only length $\leq T$ paths from START to HALT. The distribution is deficient unless the transformation graph is such that every random walk reaches HALT within T steps.

For a more general start vector such as $J_0 = \vec{b} + \vec{\delta}$ (§4.4), propagation considers paths from all the vertices having non-zero coefficients in J_0.

As noted above, propagation may be regarded as a version of relaxation in which step $t = 1, 2, \ldots T$ relaxes not just a single vertex $i^{(t-1)}$, but rather relaxes all the vertices in parallel. In the metaphor of §4.2.1, all the anthills are kicked at once, and the ants all swarm to their next destinations.

Conversely, relaxation may be regarded as a more flexible version of propagation, since it allows pieces of probability to be propagated through the graph in any convenient order rather than in lockstep. This gives relaxation two advantages:

- Probability propagated to vertex i at different times can be accumulated there, and can then be propagated further from i all at once. This "batching" is more efficient

[15] On the other hand, it might be possible to apply a form of back-propagation equally accurately and quickly to the kind of iterative solver used in §8.1. Suppose we are solving an equation $\vec{x}A = \vec{b}$ in order to compute the objective function (equation (4.5)). Let $\vec{x}^{(t)}$ be the approximation to $\vec{x}$ produced at iteration t of the biconjugate gradient method. It might be possible to "back-propagate" the partial derivatives of f with respect to $\vec{x}^{(t)}$ back through the computation of $\vec{x}^{(t)}$, obtaining the partials of f with respect to $\vec{x}^{(t-1)}$, and meanwhile updating the partials of f with respect to θ. This possibility is left to future work.

than propagating probability mass received at step t immediately on step $t+1$.

- High-probability paths can be explored further than low-probability paths, whereas in propagation, all paths are explored to the same length T regardless of how promising they are or how much effect they have on the objective function $\tilde{f}$.

However, our presentation of propagation will actually be general enough to include relaxation as a special case (see §8.5.4).

8.5.3 Implementing Propagation

Equation (8.41) translates into a simple algorithm for evaluating the objective function. As usual put $\vec{b} = \langle 0, 1, 0, 0, \ldots \rangle$. With θ fixed, adopt the following notation for each $t = 0, 1, \ldots T$:[16]

$$P \stackrel{\text{def}}{=} P_\theta \tag{8.42}$$

$$\vec{J}^{(t)} \stackrel{\text{def}}{=} \vec{b} \cdot P^t \qquad \text{(where } {}^t \text{ denotes matrix power, not transpose)} \tag{8.43}$$

$$\vec{p}^{(t)} \stackrel{\text{def}}{=} \vec{I}^{(t-1)} \odot P_{\cdot 0} \qquad (t\text{-step approximation to } \vec{p}_\theta\text{; cf. (4.7))} \tag{8.44}$$

$p_i^{(T)}$ then gives the total probability of all length $\leq T$ paths of the form $\langle \textsc{Start}, \ldots, i, \textsc{Halt} \rangle$. The pointwise product operator $\odot$ was defined in §4.1.3.

The algorithm sets $\vec{p}^{(0)} = 0$ and $\vec{J}^{(0)} = \vec{b}$, and then carries out the following update rules for each "time step" $t = 1, 2, \ldots T$:

$$\vec{p}^{(t)} = \vec{p}^{(t-1)} + \vec{J}^{(t-1)} \odot P_{\cdot 0} \tag{8.45}$$

$$\vec{J}^{(t)} = \vec{J}^{(t-1)} \cdot P \tag{8.46}$$

It returns $\vec{p}^{(T)}$. Note that $\vec{J}^{(T)}$ is never used and need not be computed.

Following §4.3.4, as a notation that will be helpful below when computing partial derivatives, let us replace P with $P^{(t)}$ in equations (8.45) and (8.46). So in principle, we

[16] One could also conceptually define the following quantity, which is not used in the computation:

$$\begin{aligned} \vec{I}^{(t)} &\stackrel{\text{def}}{=} \vec{b}(1 + P + \cdots + P^t) \qquad (t\text{-step approximation to } \vec{I}_\theta\text{; cf. (4.5))} \\ &= J^{(0)} + J^{(1)} + \ldots J^{(t)} \end{aligned}$$

might use different transition probability matrices $P^{(1)}, P^{(2)}, \ldots P^{(T)}$ on steps $1, 2, \ldots T$ of the random walk. In the usual case these matrices are all equal to P.

Just as in §4.2, we use the return value $\vec{p}^{(T)}$ to define an approximation $\tilde{f}$ to the objective function.

8.5.4 Computing the Gradient by Back-Propagation

To recap, we have $A = 1 - P$ with inverse $Q \stackrel{\text{def}}{=} A^{-1}$ and approximate inverse $\tilde{Q} \stackrel{\text{def}}{=} 1 + P + \cdots + P^{T-1}$. We saw above how to find $\vec{b}\tilde{Q}$ by propagation. We now consider its gradient.[17]

We now construct a **back-propagation** algorithm that computes this entire gradient in only about as much time as propagation itself. As promised above, it is morally equivalent to back-propagation in artificial neural networks (Werbos, 1974; Le Cun, 1985; Rumelhart et al., 1986)—whose resemblance to transformation models was detailed in §7.1.3. In particular, it resembles the "back-propagation through time" variant described by Rumelhart et al. (1986), which can handle recurrent (cyclic) networks by unrolling their loops up to some length T, just as we do in propagation.[18] Our version is simplified slightly by the fact that the vertices in a transformation model lack the thresholding or squashing functions common in neural networks. On the other hand, it is complicated by the fact that the arc probabilities are not free parameters as in neural networks, but are derived from lower-level parameters θ. We are also more concerned with optimizations that exploit the sparse and possibly redundant topology of the network (see §8.5.7).

[17] As noted in §4.3, we need the exact gradient of the approximation to pass to numerical optimization methods. It is perhaps tempting to observe that $Q' = -QA'Q$ (see equation (8.2)) and to approximate it by $-\tilde{Q}A'\tilde{Q}$, but while this is an approximation of Q' it is not exactly $(\tilde{Q})'$. (The equality $Q' = -QA'Q$ is obtained by differentiating the identity $AQ = 1$, and the inequality $\tilde{Q}' \neq -\tilde{Q}A'\tilde{Q}$ is equivalent to $\sum_{i+j+1<T} P^i P'_\theta P^j \neq \sum_{i<T, j<T} P^i P'_\theta P^j$.) The difficulty is more than just theoretical, since even very small changes to the approximation of Q can have a large impact on our objective function and its gradient (see §4.1.6).

[18] Another well-known algorithm for training recurrent neural networks propagates gradient information *forward* together with the activation (i.e., flow), so that it can learn online (Williams and Zipser, 1989). However, this strategy is targeted at cases where T is potentially unbounded, or is at least large compared to the size of the network. It must propagate many more gradients, computing and storing the partial derivative of each component of $\vec{I}^{(t)}$ with respect to *every* edge weight (or, for us, every feature weight) that might have affected it. While it would help to store and propagate only the non-zero partial derivatives, the Williams & Zipser method remains more expensive in time and storage unless T is much larger than the number of features. See (Williams and Zipser, 1995) for a review and comparison of several recurrent-net training methods.

One can obtain the relaxation algorithm as a special case of propagation. Step t of propagation can be made to relax just a single vertex i: just set $P^{(t)}$ to equal a modified identity matrix in which row i is equal to row i of P_θ. Back-relaxation then reduces to a special case of back-propagation.

8.5.5 Details of the Back-Propagation Algorithm

The idea is to propagate gradients backwards along the paths that were already used for propagation. If we already know how infinitesimal changes to $\vec{J}^{(t)}$ would have affected the objective function $\tilde{f}$, we can figure out how changes at the previous time step (to $\vec{J}^{(t-1)}$) would have affected $\tilde{f}$, via their effect on $\vec{J}^{(t)}$. We will use the same notation $\mathbf{g}x$ as in §4.3.1.

Since at the end of the day we only care about partials of the objective function (notably $\nabla\tilde{f} = \mathbf{g}\vec{\theta}$), we will avoid propagating the partials of other, intermediate quantities (cf. footnote 18). As a result, we will build up the gradient "top-down," and the computations will look a bit different than in the "bottom-up" approach that §8.1.2 used for closed-form solutions.

We use back-propagation to compute $\mathbf{g}J_i^{(t)} \stackrel{\text{def}}{=} \frac{\partial f}{\partial J_i^{(t)}}$ for all t and i. Since $\vec{J}^{(T)}$ is never used, we have $\mathbf{g}\vec{J}^{(T)} = 0$. Then for $t = T, T-1, \ldots 1$ we can use the following update rule:[19]

$$\mathbf{g}\vec{J}^{(t-1)} = P_{\cdot 0}^{(t)} \odot \mathbf{g}\vec{p}^{(T)} + P^{(t)} \cdot \mathbf{g}\vec{J}^{(t)} \tag{8.47}$$

where $\mathbf{g}p_i^{(T)}$ is defined as in equation (4.10) or (8.38) and $p_i^{(T)}$ comes from the return value of forward propagation (§8.5.3).

Of course, what we really want is the gradient of $\tilde{f}$ with respect to the feature weights.

[19] For a simple derivation of this update rule, eliminate the intermediate vectors $\vec{p}^{(1)}, \ldots \vec{p}^{(t-1)}$, collapsing the updates (8.45) into

$$\vec{p}^{(T)} = \sum_{t=1}^{T} \vec{J}^{(t-1)} \odot P_{\cdot 0}^{(t)}$$

Now for each i, the above equation together with equation (8.46) gives

$$\mathbf{g}J_i^{(t-1)} = \mathbf{g}p_i^{(T)} \cdot \frac{\partial p_i^{(T)}}{\partial J_i^{(t-1)}} + \sum_j \mathbf{g}J_j^{(t)} \cdot \frac{\partial J_j^{(t)}}{\partial J_i^{(t-1)}} = \mathbf{g}p_i^{(T)} \cdot P_{i0}^{(t)} + \sum_j \mathbf{g}J_j^{(t)} \cdot P_{ij}^{(t)}$$

As we back-propagate $\mathsf{g}J_j^{(t)}$ along the arc ij (i.e., when we multiply it by $P_{ij}^{(t)}$ during the update (8.47)), we can also use it to compute the gradient with respect to that arc's probability (cf. equation (4.23), where i was the single vertex being relaxed):[20]

$$\mathsf{g}P_{ij}^{(t)} = J_i^{(t-1)} \cdot \left\{ \begin{array}{ll} \mathsf{g}p_i^{(T)} & \text{if } j = 0 \\ \mathsf{g}J_j^{(t)} & \text{if } j \neq 0 \end{array} \right\} \tag{8.48}$$

§4.3.5 explains how each such arc probability gradient, as we compute it, in turn contributes a summand to the gradient of the weight θ_h, for every feature t_h that affects the arc probability.

8.5.6 Using Propagation to Justify the EM Formula

We now have the notation to wrap up some unfinished business. Recall the claim from §8.3 that

$$N_{ij}^{(t)} = P_{ij}^{(t)} \cdot \mathsf{g}(P_{ij}^{(t)}) \tag{8.49}$$

defines the E counts necessary to use EM for optimizing $\tilde{f}$, for all i and j. We can now justify that claim using the simpler notation provided by back-propagation.[21] In particular, we will use the fact that matrix products describe path probabilities. The (back-)relaxation version then holds as a corollary since relaxation can be treated as a special case of propagation (see §8.5.4).

Because it mentions $\mathsf{g}(P_{kj}^{(t)})$, equation (8.49) divides into cases $j \neq 0$ and $j = 0$ according to equation (8.48). If $j \neq 0$, then the claim can be rewritten as follows:

$$N_{ij}^{(t)} \tag{8.50}$$

$$= \quad P_{ij}^{(t)} \cdot \mathsf{g}P_{ij}^{(t)} \text{ by (8.49)} \tag{8.51}$$

[20]The derivation is similar to the one in footnote 19 and draws on the same propagation equations. For all ij,

$$\mathsf{g}P_{ij}^{(t)} = \mathsf{g}p_i^{(T)} \cdot \frac{\partial p_i^{(T)}}{\partial P_{ij}^{(t)}} + \mathsf{g}J_j^{(t)} \cdot \frac{\partial J_j^{(t)}}{\partial P_{ij}^{(t)}} = \mathsf{g}p_i^{(T)} \cdot \delta_{j0} \cdot J_i^{(t-1)} + \mathsf{g}J_j^{(t)} \cdot J_i^{(t-1)}$$

To obtain equation (8.48), note that the first term above falls away if $j \neq 0$, while the second term falls away if $j = 0$, since $\mathsf{g}J_0^{(t)} = 0$ by the absence of transitions from HALT.

[21]In §8.3, $\mathsf{g}(P_{kj}^{(t)})$ and $\tilde{f}$ were defined by (back-)relaxation. Here we will take them to be defined by (back-)propagation, which is formally more general.

$$= \underbrace{J_i^{(t-1)}}_{\text{forward probability}} \quad P_{ij}^{(t)} \quad \underbrace{\mathsf{g}J_j^{(t)}}_{\frac{\text{backward probability}}{\text{total probability}}} \qquad \text{by (8.48); cf. (8.26)} \tag{8.52}$$

$$= J_i^{(t-1)} \cdot P_{ij}^{(t)} \cdot \left(\sum_{t'=t+1}^{T} P^{(t+1)} \cdots P^{(t'-1)} (P_{\cdot 0}^{(t')} \odot \mathsf{g}\vec{p}^{(T)}) \right)_j \quad \text{by (8.47)} \tag{8.53}$$

$$= J_i^{(t-1)} \cdot P_{ij}^{(t)} \cdot \sum_{t'=t+1}^{T} \sum_m \left(P^{(t+1)} \cdots P^{(t'-1)} \right)_{jm} (P_{\cdot 0}^{(t')} \odot \mathsf{g}\vec{p}^{(T)})_m \tag{8.54}$$

$$= J_i^{(t-1)} \cdot P_{ij}^{(t)} \cdot \sum_{t'=t+1}^{T} \sum_m \left(P^{(t+1)} \cdots P^{(t'-1)} \right)_{jm} P_{m0}^{(t')} \cdot s_m / p_m^{(T)} \text{ by (4.10)} \tag{8.55}$$

$$= \sum_m s_m \frac{J_i^{(t-1)} \cdot P_{ij}^{(t)} \cdot \sum_{t'=t+1}^{T} \left(P^{(t+1)} \cdots P^{(t'-1)} \right)_{jm} P_{m0}^{(t')}}{p_m^{(T)}} \tag{8.56}$$

$$= \sum_m s_m \frac{\left(P^{(1)} \cdots P^{(t-1)} \right)_{1i} \cdot P_{ij}^{(t)} \cdot \sum_{t'=t+1}^{T} \left(P^{(t+1)} \cdots P^{(t'-1)} \right)_{jm} P_{m0}^{(t')}}{p_m^{(T)}} \tag{8.57}$$

This is simply a version of equation (8.24)—which defined N_{ij} for the exact EM algorithm (§8.2.3)—that has been restricted to paths of length T or less in which ij appears at step t, as desired.

If on the other hand $j = 0$, then

$$N_{i0}^{(t)} = P_{i0}^{(t)} \cdot \mathsf{g}P_{i0}^{(t)} \qquad \text{by (8.29)} \tag{8.58}$$

$$= J_i^{(t-1)} \cdot P_{i0}^{(t)} \cdot \mathsf{g}p_i^{(T)} \qquad \text{by (8.48)} \tag{8.59}$$

$$= \frac{\left(P^{(1)} \cdots P^{(t-1)} \right)_{1i} \cdot P_{i0}^{(t)} \cdot s_i}{p_i^{(T)}} \qquad \text{by (4.10)} \tag{8.60}$$

which is justified similarly; as a check, summing equation (8.59) over t gives

$$N_{i0} = \sum_{t=1}^{T} N_{i0}^{(t)} = \sum_{t=1}^{T} J_i^{(t-1)} \cdot P_{i0}^{(t)} \cdot s_i / p_i^{(T)} = p_i^{(T)} s_i / p_i^{(T)} = s_i \tag{8.61}$$

just as in the exact method (equation (8.28)).

The formulation of $N_{ij}^{(t)}$ as a forward-backward probability in equation (8.52) connects this algorithm to the closely related formulation of EM for hidden Markov models (Baum, 1972). The analogy is complicated by facts that have no analogue in HMMs: a logarithmic objective function, a log-linear parameterization of the transition probabilities, and variable-length paths. (If we considered only the paths of length *exactly* T, as in HMMs, then an observation of event m would correspond to the finite-length observed sequence $\underbrace{\epsilon\epsilon\cdots\epsilon}_{T-1}m$.[22]) But at least the hidden paths for a given input have bounded length in the propagation model, which is true in HMMs but not the exact model (§8.2.3), leading to the latter's need to solve a linear system.

8.5.7 Making (Back-)Propagation Efficient

Again, relaxation is generally preferable to propagation, for reasons described in §8.5.2. Propagation does appear to be easier to implement, but this is deceptive.

The matrix presentation of propagation is attractive and can be made efficient by using sparse matrices. Implementing P as a sparse matrix means only iterating over the arcs in the graph, and not wasting time on nonexistent zero-probability racs. Implementing $\vec{J}^{(t)}$ as a sparse vector means not bothering to propagate zeroes.

However, the matrix presentation of back-propagation is not efficient even if sparse matrices are used. That is because the vector $\mathbf{g}\vec{J}^{(t)}$ rapidly becomes non-sparse as t decreases. Sparse matrices ensure that we only bother to back-propagate along actual paths to the observed vertices, but they do not confine us to paths that originated in $\vec{b}$, that is, the paths that were actually used in propagation.

To make propagation efficient, it would therefore be necessary to use the relaxation optimizations in §4.5, such as the double stack that remembers the paths already used, and path pruning. The best approach is simply to implement propagation as a parallel version of relaxation, in which several vertices may be simultaneously relaxed (or back-relaxed) and put on the stack jointly during a given pass.

8.6 A General Presentation of Templates

§4.5.3 sketched a technique for taking advantage of redundancy in a transformation graph. The idea was that if the graph contained many isomorphic subgraphs, then one should find a way to combine work across those subgraphs.

[22]The transition probabilities are P. As for emissions, an arc from i to HALT emits i with probability 1, and and all other arcs emit the special symbol ϵ with probability 1, as suggested in §7.1.2.

What §4.5.3 sketched was a modified relaxation algorithm for the special case of perturbed models without per-event features. Here we will see that the technique applies to other models and algorithms as well. First, we formally describe the special kind of model structure where the technique applies, using matrix notation, and show that exact matrix techniques can take advantage of such model structure. Then we give an approximation algorithm that generalizes §4.5.3.

The approximation algorithm is usually more important in practical terms, and we begin with the exact algorithm (which may be skipped) merely to introduce the problem and notation.

8.6.1 Using Templates in an Exact Algorithm

§4.5.3 really hinged on *two* separate properties of the transformation model of the lexicon: (1) the transformation graph consisted of many separate subgraphs, one per headword, and (2) these subgraphs were isomorphic with similar or identical transition probabilities. Applications such as collaborative filtering (§7.2.2) also satisfy (1) and (2).

It is possible to exploit these properties in an exact algorithm as well, where the key operation is the solution of a linear system of equations (§8.1.1):

(1) means that the transition matrix P has a particularly tractable pattern of sparsity, called Schur complement form, for which faster solutions are possible. (2) means that a matrix technique called preconditioning can also be used to speed the solution.

8.6.1.1 Solving Schur Complement Systems

In general, specialized serial or parallel or approximate methods for solving $\vec{x}A = \vec{b}$ may apply if the transformation graph has some "nice" pattern of sparsity. This is the subject of considerable research in the numerical algorithms community, of which the present section provides only a taste.

We are particularly interested in block matrices of the following much-studied **Schur**

complement form:

$$P_\theta = \left(\begin{array}{c:cccc} C & S_1 & S_2 & S_3 & \dots \\ \hdashline H_1 & B_1 & & & \\ H_2 & & B_2 & & \\ H_3 & & & B_3 & \\ \vdots & & & & \ddots \end{array}\right) \qquad \text{(implying that } A = 1 - P_\theta \text{ also has this form)} \quad (8.62)$$

In the numerical algorithms community, this form arises frequently when doing domain decomposition in finite element methods.[23] In a transformation model, it arises when the transformation graph can be partitioned into several dense subgraphs that are connected only through a small set V of vertices (START, HALT, and perhaps others). The first row of blocks represents the transitions from V (including from START), the first column represents the transitions to V (including to HALT), and the diagonal element B_ℓ represents the ℓ^{th} dense subgraph.

For example, in the lexicon smoothing application (Fig. 1.2 and §3.7.2), a dense subgraph B_ℓ is induced by the lexical entries that share a given headword ℓ. The subgraphs for different headwords are not connected directly because there are no headword-changing transformations.[24]

A Schur complement matrix such as A in equation (8.62) does not in general have a sparse inverse. However, it admits efficient specialized techniques for solving $\vec{x}A = \vec{b}$ via Cholesky decomposition. These techniques require only inversion of the matrix blocks $B_1, B_2, \dots$ and E, where E is an intermediate matrix the same size as C. It is also possible to exploit sparsity in those blocks: instead of inverting $B_1, B_2, \dots E$, one can iteratively solve solutions of sparse linear systems involving them, just as in §8.1.1. The methods for solving Schur complement systems are called the **Element-by-Element (EBE) method** or **iterative substructuring**.

[23] Another potentially useful form has overlapping domains (overlapping dense subgraphs); these are handled by so-called Schwarz methods.

[24] If there are no category-changing transformations either, the subgraphs can be further partitioned by LHS category. This leads to a Schur complement form with even smaller blocks, allowing the method of this section to run faster, although these smaller blocks will generally be too dissimilar to one another for the techniques in §8.6.1.4 and §8.6.2 to apply.

8.6.1.2 Special Case of the Schur Complement

In the special case that $H_1 = H_2 = \cdots = 0$, a Schur complement matrix actually does have a sparse closed-form inverse, namely

$$\left(\begin{array}{c|cccc} C^{-1} & -C^{-1}S_1B_1^{-1} & -C^{-1}S_2B_2^{-1} & -C^{-1}S_3B_3^{-1} & \cdots \\ \hline & B_1^{-1} & & & \\ & & B_2^{-1} & & \\ & & & B_3^{-1} & \\ & & & & \ddots \end{array}\right) \tag{8.63}$$

The matrices that arise in practice are sufficiently close to this case for equation (8.63) to be indirectly quite useful. First, an intuition as to why. The H_i blocks in equation (8.62) usually only represent transitions to HALT. Remember that the inverse $A^{-1} = (1-P_\theta)^{-1} = 1 + P_\theta + P_\theta^2 + \cdots$ stores at position ij the total probability of all paths from i to j (§8.2.3, equation (8.20)). None of these paths can pass *through* HALT since it is a sink state; at best they can reach HALT (if $j = 0$). So the H_i blocks of P_θ—the transition probabilities to HALT—can affect only column 0 of A^{-1}. Since the computations in which we use A^{-1} generally ignore column 0,[25] we may as well pretend the H_ℓ blocks are zero and just use equation (8.63) directly.

Let us formalize a more general version of this argument. The idea is that transformation graphs of the Schur complement form (8.62) often have a more specific form that is close to the special case $H_1 = H_2 = \cdots = 0$:

$$P_\theta = \left(\begin{array}{cc|cccc} C_{00} & 0 & 0 & 0 & 0 & \\ C_{10} & C_{11} & S_{11} & S_{12} & S_{13} & \cdots \\ \hline H_{10} & 0 & B_1 & & & \\ H_{20} & 0 & & B_2 & & \\ H_{30} & 0 & & & B_3 & \\ \vdots & & & & & \ddots \end{array}\right) \tag{8.64}$$

[25] $(A^{-1})_{i0}$ represents the total probability of all paths from i to HALT. This is not used directly because what the transformation model cares about is the event that the path reaches immediately *before* HALT.

To put this qualitatively, the connecting set V consists of two disjoint subsets, a "source" V_1 and a "sink" V_0, such that V_1 receives no arcs from outside V_1 and V_0 sends no arcs to outside V_0.[26] Note that V_1 contains START and V_0 contains HALT; in the intuitive argument above, V_0 contained *only* HALT.

Now define P_1 as identical to P_θ except that it zeroes out transitions to all sink vertices V_0 (in particular, the H blocks), and set $P_2 = P_\theta - P_1$. The crucial fact is that $P_2 P_1 = 0$ because there are no paths that enter V_0 and leave it again. This fact lets us reduce the expansion $A^{-1} = (1-P_\theta)^{-1} = 1 + (P_1+P_2) + (P_1+P_2)^2 + \ldots$ to $(1+P_1+P_1^2+\ldots)(1+P_2)$, and hence to $(1-P_1)^{-1}(1+P_2)$.

This product $A^{-1} = (1-P_1)^{-1}(1+P_2)$ can be efficiently computed by using equation (8.63) to figure the inverse $(1-P_1)^{-1}$. It is sparse, and indeed has the Schur complement form. In fact it can be written as the sum of $(1-P_1)^{-1}$ and $(1-P_1)^{-1}P_2$, which are easily seen to have the same sparsity patterns as P_1 and P_2 respectively.[27]

8.6.1.3 Decomposition Into Strongly Connected Components

The previous section can be simplified and generalized. Ultimately, the reason that §8.6.1.2 admitted efficient solutions was that, granted the special property of V, the graph had a natural decomposition into strongly connected components $(V_0, V_1, B_1, B_2, B_3, \ldots)$.

In fact *any* transformation graph that decomposes into strongly connected components admits a modular exact solution, by means analogous to §4.2.2's approximate solution. This is a much more general strategy.

Let $B_1, B_2, B_3, \ldots$ denote the strongly connected components, in topologically sorted order (see §4.2.2). Then by choosing a vertex order that respects this order of the components, P_θ can be expressed as an upper triangular block matrix where $B_1, B_2, B_3, \ldots$ are the blocks along the diagonal. Such matrices are well-studied. It is particularly easy to

[26] So once a random walk on the transformation graph returns to V, it must stay in V until it halts. Any such walk in this kind of graph consists of a subwalk on the source vertices, followed by a subwalk on the B_ℓ vertices for some ℓ, followed by a subwalk on the sink vertices (usually just HALT). Any of these subwalks can be empty.

[27] This general solution for any V_0 makes use of P_2. The intuition at the start of this section came from the more restricted case $V_0 = \{\text{HALT}\}$. In that case we can ignore P_2, as claimed then, because P_2 and hence $(1-P_1)^{-1}P_2$ consists of just a 0^{th} column. If we do not care about adding that column into A^{-1}, we can pretend P_2 is 0.

solve for $\vec{b}(1-P_\theta)^{-1}$ by solving a sequence of subproblems of the form $\vec{J}_\ell(1-B_\ell)^{-1}$ for $\ell = 1, 2, 3, \ldots$. Iterative methods as in §8.1.1 may be used for each subproblem, to exploit the sparsity of B_ℓ.

The idea is that $\vec{J}_\ell$ represents the flow to vertices in subgraph ℓ from subgraphs preceding ℓ: that is, its i^{th} element is the total probability of all paths that terminate at the i^{th} vertex of subgraph ℓ but do not otherwise pass through subgraph ℓ. Now $\vec{I}_\ell \stackrel{\text{def}}{=} \vec{J}_\ell(1-B_\ell)^{-1}$ gives the total flow to vertices in subgraph ℓ, by extending those paths through subgraph ℓ in all possible ways; remember that by assumption, no path to subgraph ℓ can pass through any subgraph beyond ℓ.

Having computed $\vec{I}_\ell$, we can proceed with subgraph $\ell+1$, first obtaining $\vec{J_{\ell+1}}$ by multiplying the simple row vector

$$\left(\vec{I}_1 \ \vec{I}_2 \ \cdots \vec{I}_\ell \ 0 \ 0 \ \cdots \right) \tag{8.65}$$

by the block column of P_θ that contains $B_{\ell+1}$ (this column represents arcs to $B_{\ell+1}$), and adding the appropriate subvector of $\vec{b}$ (representing the starting flow at the vertices of $B_{\ell+1}$). This technique is also used to find the base case $\vec{J}_1$, when $\ell = 0$.

8.6.1.4 Preconditioning

A standard trick called **preconditioning** helps the iterative solvers of §8.1.1 converge more quickly and with less numerical error. The trick is to replace $\vec{x}A = \vec{b}$ with an equivalent equation of the same form, $\vec{x}(A\tilde{A}^{-1}) = \vec{b}\tilde{A}^{-1}$. $\tilde{A}$ is a **preconditioner** matrix that is close to A but more easily inverted. The system $A\tilde{A}^{-1}$ is therefore close to an identity matrix, which makes iterative solvers more stable. Moreover, $\vec{b}\tilde{A}^{-1}$ is close to the desired $\vec{x} = \vec{b}A^{-1}$ and so provides a fairly accurate starting guess for $\vec{x}$, allowing faster convergence.

There is a literature on appropriate preconditioners for sparse matrices, including matrices that exhibit (or nearly exhibit) particular patterns such as Schur complement (§8.6.1.1). As a naive example, if A has the Schur complement form, then we can construct $\tilde{A}$ by setting its H_ℓ blocks to zero. The result is invertible as in equation (8.63).

The preconditioner is one place to take advantage of further structure in the transformation model. Equation (8.63) and the other techniques of the previous section still requires us to invert all the square blocks $B_1, B_2, \ldots$ separately. But in practice most of the B_ℓ may have the same dimensions and be very similar to one another. If in our construction of $\tilde{A}$ we replace some of them with a common block B, then we will have fewer distinct blocks to invert. (Replacing some of the S_ℓ with a common S is also beneficial, but less so.)

For example, when equation (8.62) arises in the lexicon smoothing task, B_ℓ represents the transitional probabilities among lexical entries for a given word ℓ. These do not differ so much from word to word. If we ignored word-specific features when computing the transitional probabilities B_ℓ, we would obtain a reasonable word-independent approximation B. We can speed up our inversion of the preconditioner $\tilde{A}$ by substituting B for any or all of the B_ℓ in $\tilde{A}$. B is a particularly good substitute for B_ℓ if word ℓ has small word-specific feature weights. Happily, this is so for most words, thanks to the Zipfian fact that most words are not observed frequently enough to overcome the prior.

At least in principle, one might wish to use different preconditioning blocks B for different B_ℓ, based on properties of the B_ℓ blocks or the words they describe. For example, one might use different preconditioners for suspected nouns than for suspected verbs, based on solutions for a few typical nouns or verbs.

8.6.2 Using Templates with Propagation or Relaxation

We now turn to the use of approximate algorithms with models of this special form. To be explicit, we are concerned with transformation models that have *all* of the three properties discussed in the previous three sections:

- The model has the Schur complement form discussed in §8.6.1.1.
- The model also has the source-sink property discussed in §8.6.1.2, with a set V_1 of source vertices that contains START and a set V_0 of sink vertices.
- The model contains similar subgraphs as in §8.6.1.4:
  - Square blocks B_ℓ in the Schur form (8.62) have the same dimension and tend to be similar (for all θ) to a template B, as discussed in §8.6.1.4. The sense of

"similar" is stronger here: it means many entries that are not merely close but actually equal.

– Moreover, if possible, all the blocks H_ℓ tend to be similar to a template H.

– Finally, each block S_ℓ is similar to $\beta_\ell S$ where β_ℓ is a scalar and S is a template. By scaling S we can arrange that $\sum_\ell \beta_\ell = 1$. (The use of β_ℓ allows the model to have different probabilities of entering different subgraphs from V_1. For example, different words have different probabilities.)

Again, the intuition is that the subgraphs of the transformation graph that correspond to different words (see Fig. 1.3) are variations on a common template (Fig. 4.1). They are naturally isomorphic to the template, which represents the "typical" word; but some of their probabilities are different.

If we run the propagation algorithm on the template, we obtain a starting point for propagation through the various subgraphs of the actual transformation graph. We copy the template to each subgraph and re-propagate along just those subgraph paths whose probabilities differ from their analogues in the template. A similar approach works for back-propagation.

Notice that, unlike §4.5.3, we will not assume that the transformation probabilities are necessarily identical in each block: thus per-event features are possible.

As in §4.5.3, the notations $\mathtt{s}_\ell x$ and $\mathtt{t}x$ will refer to the values of a variable x in the computation on subgraph ℓ and the template, respectively. The computation of $\mathtt{s}_\ell x$ makes use of the previously computed $\mathtt{t}x$. Also, $\Delta_\ell x$ denotes $\mathtt{s}_\ell x - \mathtt{t}x$. Finally, $\mathsf{dim}\, x$ denotes the length of a vector x, or the length of the side of a square matrix x.

8.6.2.1 Propagation With a Template

The subgraph propagation or relaxation problems are defined as follows:[28]

$$\mathtt{t}P \stackrel{\text{def}}{=} \left(\begin{array}{c:c} C & S \\ \hdashline H & B \end{array}\right) \tag{8.66}$$

$$\mathsf{s}_\ell P \stackrel{\text{def}}{=} \left(\begin{array}{c:c} C & \beta_\ell^{-1} S_\ell \\ \hdashline H_\ell & B_\ell \end{array}\right) \tag{8.67}$$

$$\mathtt{t}\vec{b} = \mathsf{s}_\ell \vec{b} \stackrel{\text{def}}{=} \langle 0, 1, 0, 0, \ldots \rangle \text{ of length } \mathsf{dim}\, C + \mathsf{dim}\, B \tag{8.68}$$

We can separately apply the propagation or relaxation algorithm to the template and to each subgraph ℓ. In the case of a subgraph, let us write the result in block form:

$$\mathsf{s}_\ell \vec{p}^{(T)} = \left(\begin{array}{c:c} \mathsf{s}_\ell \vec{p}_C & \mathsf{s}_\ell \vec{p}_B \end{array}\right) \tag{8.69}$$

where $\mathsf{dim}\, \mathsf{s}_\ell \vec{p}_C = \mathsf{dim}\, C$ and $\mathsf{dim}\, \mathsf{s}_\ell \vec{p}_B = \mathsf{dim}\, B = \mathsf{dim}\, B_\ell$.

Thanks to the source-sink property, we can combine the subgraphs' results across different ℓ to get the result of propagating through the entire transformation graph:[29]

$$\begin{aligned} \vec{p}^{(T)} &= \left(\begin{array}{c:c:c:c} \vec{p}_C & \vec{p}_{B_1} & \vec{p}_{B_2} & \cdots \end{array}\right) && (8.70) \\ &= \left(\begin{array}{c:c:c:c} \sum_\ell \beta_\ell \mathsf{s}_\ell \vec{p}_C & \beta_1 \mathsf{s}_1 \vec{p}_B & \beta_2 \mathsf{s}_2 \vec{p}_B & \cdots \end{array}\right) && (8.71) \end{aligned}$$

[28] The propagation and relaxation algorithms do not require matrix rows to sum to 1, and indeed the top rows of these matrices may sum to something else. If for some reason we do want all the rows (except row 0) to sum to 1, then it suffices to ensure that each S_ℓ has the same row sums as $\beta_\ell S$, which can always be arranged. (If START is the only vertex in V_1, so that S and S_ℓ each consist of only one row, then satisfying the requirement is trivial by an appropriate choice of β_ℓ. If there are other vertices in V_1, they can be eliminated first by splitting each into ℓ copies and considering these to be part of the ℓ respective subgraphs.)

[29] Proof sketch: for reasons mentioned in footnote 26 on p. 273, the source-sink form of the graph guarantees that in the exact case (as $T \to \infty$),

$$\begin{aligned} \mathsf{s}_\ell \vec{p}^{(T)} &= (\mathsf{s}_\ell \vec{b}(1 + \mathsf{s}_\ell P + \mathsf{s}_\ell P^2 + \cdots)) \odot \mathsf{s}_\ell P_{\cdot 0} \\ \mathsf{s}_\ell \vec{p}_B &= \vec{b}(1 - C)^{-1} \beta_\ell^{-1} S_\ell (1 - B_\ell)^{-1} \odot (H_\ell)_{\cdot 0} \\ \mathsf{s}_\ell \vec{p}_C &= \vec{b}(1 - C)^{-1}(1 + \beta_\ell^{-1} S_\ell (1 - B_\ell)^{-1} H_\ell (1 - C)^{-1}) \odot C_{\cdot 0} \\ \vec{p}^{(T)} &= (\vec{b}(1 + P + P^2 + \cdots)) \odot P_{\cdot 0} \\ \vec{p}_{B_\ell} &= \sum_\ell \vec{b}(1 - C)^{-1} S_\ell (1 - B_\ell)^{-1} \odot (H_\ell)_{\cdot 0} \\ \vec{p}_C &= \vec{b}(1 - C)^{-1}(1 + \sum_\ell S_\ell (1 - B_\ell)^{-1} H_\ell (1 - C)^{-1}) \odot C_{\cdot 0} \end{aligned}$$

and since $\sum_\ell \beta_\ell = 1$, the claim follows immediately. The proof for finite T is similar.

So far we have not saved any work or used the template. The payoff comes when working on a subgraph. In the case of propagation, this computation can take advantage of the work already done on the template by the template-specific versions of equations (8.45) and (8.46), namely

$$\begin{aligned}\mathtt{t}\vec{p}^{(t)} &= \mathtt{t}\vec{p}^{(t-1)} + \mathtt{t}\vec{J}^{(t-1)} \odot \mathtt{t}P_{\cdot 0} &(8.72)\\ \mathtt{t}\vec{J}^{(t)} &= \mathtt{t}\vec{J}^{(t-1)} \cdot \mathtt{t}P &(8.73)\end{aligned}$$

Instead of propagating a subgraph's $\mathtt{s}_\ell\vec{p}^{(t)}$ and $\mathtt{s}_\ell\vec{J}^{(t)}$ directly by update rules just like the above, we can propagate only their differences from the template. Still phrasing this in terms of propagation, this means using sparser matrix multiplications, which are more efficient:

$$\begin{aligned}\Delta_\ell\vec{J}^{(t)} &= \mathtt{s}_\ell\vec{J}^{(t)} - \mathtt{t}\vec{J}^{(t)} &(8.74)\\ &= \mathtt{s}_\ell\vec{J}^{(t-1)} \cdot \mathtt{s}_\ell P^{(t)} - \mathtt{t}\vec{J}^{(t-1)} \cdot \mathtt{t}P^{(t)} \qquad \text{(by subgraph version of (8.46))} &(8.75)\\ &= (\mathtt{t}\vec{J}^{(t-1)} + \Delta_\ell\vec{J}^{(t-1)}) \cdot (\mathtt{t}P^{(t)} + \Delta_\ell P^{(t)}) - \mathtt{t}\vec{J}^{(t-1)} \cdot \mathtt{t}P^{(t)} &(8.76)\\ &= (\mathtt{t}\vec{J}^{(t-1)} \cdot \underbrace{\Delta_\ell P^{(t)}}_{\text{sparse}}) + (\underbrace{\Delta_\ell\vec{J}^{(t-1)}}_{\text{hopefully sparse}} \cdot \mathtt{s}_\ell P^{(t)}) &(8.77)\\ \Delta_\ell\vec{p}^{(t)} &= \Delta_\ell\vec{p}^{(t-1)} + (\mathtt{t}\vec{J}^{(t-1)} \odot \underbrace{\Delta_\ell P^{(t)}_{\cdot 0}}_{\text{sparse}}) + (\underbrace{\Delta_\ell\vec{J}^{(t-1)}}_{\text{hopefully sparse}} \odot \mathtt{s}_\ell P^{(t)}_{\cdot 0}) \quad \text{(similarly)} &(8.78)\end{aligned}$$

At the end we can compute the return value for subgraph ℓ,

$$\mathtt{s}_\ell\vec{p}^{(T)} = \mathtt{t}\vec{p}^{(T)} + \Delta_\ell\vec{p}^{(T)} \tag{8.79}$$

Note that the stored variables are of the form $\mathtt{t}x$ (for the template) and $\Delta_\ell x$ (for the subgraph, and sparser), with $\mathtt{s}_\ell x \stackrel{\text{def}}{=} \mathtt{t}x + \Delta_\ell x$ not stored.

8.6.2.2 Relaxation and Back-Relaxation with a Template

It is not difficult to see how to adapt the simple template relaxation and back-relaxation algorithms in §4.5.3 to the equations above. The new wrinkle is that $\Delta_\ell P$ may now be nonzero. That is, some transition probabilities in subgraph ℓ may differ from those in the template, for instance because of per-event features in subgraph ℓ. Let us briefly sketch the consequences without giving pseudocode.

For relaxation on subgraph ℓ, the possibility that $\Delta_\ell P \neq 0$ means that equations (8.77) and (8.78) now require access to $\mathtt{t}\vec{J}$ as well as $\mathtt{s}_\ell \vec{J}$ values. Fortunately, these values were already available in §4.5.3.2 (and see footnote 23 on p. 140). Notice that it is now necessary to relax i while processing subgraph ℓ if *either* $\Delta_\ell J_i \neq 0$ as before, *or else* $\mathtt{t}J_i \cdot \Delta_\ell P_{ij} \neq 0$ for some j.

Back-relaxation on subgraph ℓ also needs access to both $\mathtt{t}\vec{J}$ and $\mathtt{s}_\ell \vec{J}$. Back-relaxing vertex i at time step t computes $\mathtt{g}\Delta_\ell J_i^{(t-1)}$ and adds to $\mathtt{g}P_{i\cdot}^{(t)}$ (specifically, to $\mathtt{gs}_\ell P_{i\cdot}^{(t)}$) just as in §4.5.3.4. But now back-relaxing i must also add to $\mathtt{gt}J_i^{(t-1)}$ and add some more to $\mathtt{g}\Delta_\ell P_{i\cdot}{}^{(t)} = \mathtt{gs}_\ell P_{i\cdot}{}^{(t)}$, since the new rules for relaxing vertex i (equations (8.77) and (8.78)) are sensitive to $\mathtt{t}J_i^{(t-1)}$.

It follows that back-relaxation on the template will start with non-zero values for $\mathtt{gt}J_i{}^{(t-1)}$ that were accumulated during back-relaxation on the subgraphs. Such a value is stored with $i^{(t-1)}$ on $\mathtt{t}\mathit{Stack}1$. When step t of template back-relaxation pops ($i^{(t-1)}$, $\mathtt{t}J_i^{(t-1)}$) from $\mathtt{t}\mathit{Stack}1$, it will also pop $\mathtt{gt}J_i{}^{(t-1)}$. The step then accumulates terms into $\mathtt{gt}J_i{}^{(t-1)}$ starting at this popped value rather than at 0. (The additional terms arise from the mentions of $\mathtt{t}J_i{}^{(t-1)}$ in template relaxation, in the sense of §4.3.2, while the starting value arose from the mentions during subgraph relaxation.) Finally, the template back-relaxation of i adds to $\mathtt{gt}P_{i\cdot}^{(t)}$ rather than any $\mathtt{gs}_\ell P_{i\cdot}^{(t)}$.

8.6.2.3 Templates for Per-Event Features vs. Perturbations

Recall from §3.9 that $\mathtt{s}_\ell p_i$ can be made tunable in either of two ways. Vertex i in subgraph ℓ can be given a per-event feature or a perturbation parameter π_i.

Equation (8.78) tends to be appropriately sparse in the latter case. A perturbation π_i affects only the arcs from i, so it requires differences Δ_ℓ to be propagated to the descendants of i. (See §4.5.3 for the details.)

By contrast, a per-event feature for i affects the arcs to i *and their competitors.* In a particularly bad and common case, i is a child of $\textsc{Start}_\ell$.[30] So all arcs from $\textsc{Start}_\ell$ have different probabilities in subgraph ℓ than in the template, and it is necessary to propagate

[30] For example, the model used for the experiments of Chapter 6 gives special status to the events in subgraph ℓ (i.e., lexical entries headed by word ℓ) that were observed in training data. They serve both as the "tunable" events with per-event features and also as (some of) the children of $\textsc{Start}_\ell$.

differences from $\textsc{Start}_\ell$ to all descendants. This more than wipes out the advantage of the template: it would have been faster to solve the subgraph-ℓ problem by doing an ordinary propagation from $\textsc{Start}_\ell$ to its descendants in the first place, without using the template.

There is, however, a fix that makes it unnecessary to repropagate fully from $\textsc{Start}_\ell$ in this particular case. We can leave the probabilities on the arcs from $\textsc{Start}_\ell$ unnormalized, so that most arcs have the same "probability" in subgraph ℓ as in the template. Provided that every path from $\textsc{Start}$ to $\textsc{Halt}$ in the subgraph-ℓ problem passes exactly once through $\textsc{Start}_\ell$, as in Fig. 1.3, we can correctly compensate for this change by normalizing $\mathbf{s}\vec{_\ell p}$ *after* propagating through the subgraph.[31]

[31]The global renormalization constant is computed as usual (§8.4.1): set $P_{1\infty} \in \mathbb{R}$ so that $\sum_i P_{1i} = 1$.

Chapter 9

Conclusions

This thesis has developed a stochastic approach to lexicalized syntax—including the lexical redundancy rules that derive lexical entries from one another. Chapter 1 gave a self-contained overview of the work, its motivation, and its empirical performance on an example.

As explained in Chapter 1, the work attempts to appeal simultaneously to the interests of several communities. Like any ménage à trois, this marriage of linguistics, statistics, and engineering is not without its tensions—particularly the competing desires for fidelity and tractability in the model—but has its own existence and internal logic.

For linguists, the thesis has shown it possible to integrate statistics throughout a modern syntactic theory using a single mechanism, without either eviscerating the theory or making it ugly. Abney (1996) has argued that linguists should broaden their notion of human linguistic competence to include sensitivity to frequency. Transformation models of the lexicon extend this idea into the underlying fabric of the grammar, by modeling not only the frequency of different constructions, but also the strength of the grammar's transformational generalizations and their exceptions.

Indeed, the frequencies of constructions are here entirely derived from the fabric of generalizations and exceptions. All that the statistical grammar ultimately specifies are the properties of a lexical entry or transformation that make it more or less likely to be used while generating a sentence.

In statistics, the new idea of transformational modeling may have broader use. Transformation models address a general problem: estimating a probability distribution over a set of interrelated events, where the relationships are known in advance but their effects on the distribution are not. A simple prior encourages the model to use a small number of relationships to approximate a large number of probabilities. Such approximation results in smoothing the probabilities.

To facilitate future statistical work with transformation models—both of the lexicon and of other phenomena—the thesis has presented them in a general form. The presentation abstracts away from the details of the linguistic application, and includes a number of variations that may be useful for other problems. The algorithms for solving and estimating transformation models were also given in a general form and with variations. Indeed, they can be used to estimate the parameters of probabilistic finite-state machines (see §7.1.2).

For NLP, the experiments in Chapter 6 demonstrated transformational smoothing's ability to accurately predict lexical entries given sparse training data. This problem has been significant in the statistical parsing community, which has turned to lexicalized statistics over the past decade. The method achieved a healthy perplexity reduction over all competing methods from the literature. Indeed, it performed comparably to the next-best method using only half as much training data, and its advantage increased when the amount of training data was reduced. Its improvement applied not merely to the test set as a whole, but also to the individual test items, which were almost uniformly better predicted by using transformations. The reasons appeared to lie in the transformation model's ability to model exceptions, its Bayesian approach to smoothing, and its ability to model transformational relationships that were genuinely present in the data.

This empirical work is best regarded only as a starting point. In an effort to be fair to the competing methods, the evaluation used a bare-bones transformation model that was as much like the competing models as possible. In particular, the model did not consider any "linguistically interesting" transformations, such as gapping (in particular subject-extraction, which is very common). Nor did it parameterize the transformations in terms of linguistically substantive features. Yet in principle, one of the advantages of transformation models over past work is their greater capacity to model linguistic substance, as discussed elsewhere in the thesis. This is an obvious source of possible future improvements.

More immediately, there are many possible ways to improve the parameter estimation. One is speed, via EM (Chapter 8) or conjugate gradient. As for accuracy, the thesis has described various possible under-the-hood changes to the objective function and its approximation (notably in normalization and the prior). Most important would be ways to avoid local maxima—currently the Achilles' heel of transformation models.

In general, one could also try to make the approach simpler and more robust for engineering purposes:

- Greater use of held-out data (rather than the prior) in estimating θ; for example, leave-one-out training.
- Techniques for avoiding local maxima: annealing and incremental feature selection.
- Use of weighted, nondeterministic finite-state transducers to model the transformational process. A simple string-to-string transducer can model one step of the random walk, transforming an input frame such as $\texttt{S} \rightarrow \texttt{NP}$ —— $\texttt{NP}$ stochastically into an output frame. More significantly, a simple transducer can just as easily model an *unbounded sequence* of transformations (including insertions) that examine and modify non-overlapping substrings of the input frame.[1] The transformational process can be approximately modeled by composing a fixed number of such transducers, whose parameters can then be estimated (see (Eisner, 2001)).

Ultimately, the value of transformational smoothing to NLP will be measured not by the perplexity it assigns to a test set of lexical entries, but by its ability to improve performance on a "real" problem such as parsing, generation, or grammar induction. Chapter 5 outlined how the technique could be used in parsing for predicting both appropriate headwords and appropriate syntactic frames for them. It could also be used to rerank the output

[1] See (Ristad and Yianilos, 1996) for an edit-distance transducer, which could be modified to compute contextual edit distance. The requirement that the edits in the sequence be chosen locally rules out the use of per-event features during such a sequence, since such features would condition transducer actions on the entire frame. A reasonable option is to allow per-event features only on arcs from START. Then words can differ as to the initial frames they like to project, but the subsequent transformational process does *not* vary from word to word in its probabilities.

of a parsing or generation system. Finally, §1.2.4 discussed the application to grammar induction.

Let us close by returning to the context of the work. The statistical paradigm addresses the need for robustness and trainability in the face of uncertainty. In computational linguistics, it has been rapidly filtering upward from the speech-level tasks where it made its debut. The community has struggled, at length but often successfully, to apply statistical methods to ever-fuller descriptions of linguistic competence. The challenge, as for other formalization, is to find an appropriate angle from which to rethink the available linguistic theories and the data that underlie them.

This thesis sits squarely in that tradition. It is the first attempt to statistically rethink the "deep structure" of lexicalized syntax, which is crucial to linguistic competence and language learning, but whose details are often given short shrift even among pure linguists. Like other attempts to model linguistic complexity (e.g., PCFGs and their successors), the work has had to introduce new statistical techniques, new parameters, and new hidden variables.[2] Part of its contribution, however, is simply to place the problem on the table for further study. Other researchers who wish to join in are welcome to ask for the training and test data.

[2]Namely, a hidden web of transformations underlying a lexicon of parse fragments that are themselves at best partly observed.

References

Anne Abeillé. 1988. Parsing French with tree-adjoining grammar: Some linguistic accounts. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING '88)*, Budapest, August.

Steven Abney and Marc Light. 1999. Hiding a semantic hierarchy in a Markov model. In *Proceedings of the ACL'99 Workshop on Unsupervised Learning in Natural Language Processing*, pages 1–8.

Steven Abney. 1996. Statistical methods and linguistics. In Judith L. Klavans and Philip Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, chapter 1, pages 1–26. MIT Press.

Hiyan Alshawi. 1996. Head automata and bilingual tiling: Translation with minimal representations. In *Proceedings of the 34th ACL*, pages 167–176, Santa Cruz, CA.

S. R. Anderson. 1992. *A-morphous Morphology*. Cambridge University Press.

John R. Anderson. 1993. *Rules of the Mind*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Mark Aronoff. 1976. *Word Formation in Generative Grammar*. MIT Press.

Jose Mari Arriola, Xabier Artola, Aitor Maritxalar, and Aitor Soroa. 1999. A methodology for the analysis of verb usage examples in a context of lexical knowledge acquisition from dictionary entries. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora (LINC-99)*, Bergen, Norway, June.

Harald Baayen and Richard Sproat. 1996. Estimating lexical priors for low-frequency syncretic forms. *Computational Linguistics*, 22(2):155–166.

Lalit R. Bahl and Frederick Jelinek. 1975. Decoding for channels with insertions, deletions and substitutions with applications to speech recognition. *IEEE Transactions on Information Theory*, IT-21(4):404–411.

Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 5(2):179–190.

J. K. Baker. 1979. Trainable grammars for speech recognition. In Jared J. Wolf and Dennis H. Klatt, editors, *Speech Communication Papers Presented at the 97th meeting of the Acoustical Society of America*, MIT, Cambridge, MA, June.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2), June.

Petra Barg and Markus Walther. 1998. Processing unknown words in HPSG. In *Proceedings of COLING-ACL '98*, volume 1, pages 91–95.

R. Basili, M. T. Pazienza, and M. Vindigni. 1997. Corpus-driven unsupervised learning of verb subcategorization frames. In M. Lenzirini, editor, *AI*IA 97: Advances in Artificial Intelligence*, number 1321 in Lecture Notes in Artificial Intelligence. Springer-Verlag.

R. Basili, M. T. Pazienza, and F. M. Zanzotto. 1999. Lexicalizing a shallow parser. In *Proceedings of Le Traitment Automatique des Langues Naturelles (TALN 1999)*, Cargse, Corsica.

L. E. Baum. 1972. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. *Inequalities*, 3.

Tilman Becker. 1994. Patterns in metarules. In *Proceedings of the TAG+3 Workshop*, Paris.

Tilman Becker. 2000. Patterns in metarules. In *Tree Adjoining Grammars: Formal, Computational and Linguistic Aspects*. Stanford University.

Adam L. Berger, Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, John R. Gillett, John D. Lafferty, Robert L. Mercer, Harry Printz, and Luboš Ureš. 1994. The Candide system for machine translation. In *Proceedings of ARPA Conference on Human Language Technology*.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum-entropy approach to language modeling. *Computational Linguistics*, 22(1):39–71.

Adam Berger. 1997. The Improved Iterative Scaling algorithm: A gentle introduction. Tutorial note, Carnegie Mellon University. Available online.

Robert C. Berwick. 1980. Learning structural descriptions of grammar rules from examples. M.S. thesis, MIT.

Thomas G. Bever. 1970. The cognitive basis of linguistic structures. In J. R. Hayes, editor, *Cognition and the Development of Language*, pages 279–362. Wiley, New York.

Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. 1995. Bracketing guidelines for Treebank II style: Penn Treebank project. Technical Report MS-CIS-95-06, University of Pennsylvania, January.

L. Bloomfield. 1933. *Language*. Holt, Rinehart, and Winston, New York.

A. Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the 1998 Conference on Computational Learning Theory*, July.

Rens Bod and Remko Scha. 1996. Data-oriented language processing: An overview. IILC Technical Report LP-96-13, University of Amsterdam.

Gosse Bouma and Gertjan van Noord. 1994. Constraint-based categorial grammar. In *Proceedings of the 32nd Annual Meeting of the ACL*.

Gosse Bouma, Rob Malouf, and Ivan A. Sag. 2001. Satisfying constraints on extraction and adjunction. *Natural Language and Linguistic Theory*.

Michael R. Brent. 1993. From grammar to lexicon: Unsupervised learning of lexical syntax. *Computational Linguistics*, 19(2):243–262, June.

Michael R. Brent. 1994. Acquisition of subcategorization frames using aggregated evidence from local syntactic cues. *Lingua*, 92:433–470. Reprinted in L. Gleitman and B. Landau (Eds.), *Acquisition of the Lexicon*, MIT Press, Cambridge, MA.

Joan Bresnan and Ronald M. Kaplan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*. MIT Press.

Joan Bresnan. 1978. A realistic transformational grammar. In Morris Halle, Joan Bresnan, and George A. Miller, editors, *Linguistic Theory and Psychological Reality*. MIT Press, Cambridge, MA.

Joan Bresnan. 1982a. The passive in lexical theory. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

Joan Bresnan. 1982b. Polyadicity. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

Eric Brill. 1995. Unsupervised learning of disambiguation rules for part of speech tagging. In David Yarowsky and Kenneth Church, editors, *Proceedings of 3rd Workshop on Very Large Corpora*, MIT, June. Association for Computational Linguistics. Also to appear in *Natural Language Processing Using Very Large Corpora*, 1997.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of 7^{th} World Wide Web Conference.*

E. J. Briscoe and J. Carroll. 1993. Generalized probabilistic LR parsing for unification-based grammars. *Computational Linguistics*, 19(1):25–60.

E. J. Briscoe and J. Carroll. 1997. Automatic extraction of subcategorisation from corpora. In *Proceedings of the 5^{th} ACL Conference on Applied Natural Language Processing*, pages 356–363, Washington, DC.

Ted Briscoe and Ann Copestake. 1999. Lexical rules in constraint-based grammar. *Computational Linguistics*, 25(4):487–526, December.

Sabine Buchholz. 1998. Distinguishing complements from adjuncts using memory-based learning. In B. Keller, editor, *Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing*, pages 41–48.

B. Butterworth. 1983. Lexical representation. In B. Butterworth, editor, *Language Production Volume 2: Development, Writing and Other Language Process*. Academic Press, London.

Nicoletta Calzolari, Stefano Federici, Simonetta Montemagni, and Vito Pirrelli. 1998. An analogy-based approach to lexicon acquisition. Deliverable 5.1, LE-SPARKLE project. Istituto di Linguistica Computazionale, Consiglio Nazionale delle Ricerche, Rome. Available online.

Sharon A. Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics.*

Bob Carpenter. 1991. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics*, 17(3):301–313, September.

Bob Carpenter. 1992. Lexical and unary rules in categorial grammar. In B. Levine, editor, *Formal Grammar: Theory and Implementation*, volume 2 of *Vancouver Studies in Cognitive Science*. Oxford University Press. Online version updated 1995.

Glenn Carroll and Mats Rooth. 1998. Valence induction with a head-lexicalized PCFG. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

John Carroll, Guido Minnen, and Ted Briscoe. 1998. Can subcategorisation probabilities help a statistical parser? In *Proceedings of the ACL Workshop on Very Large Corpora*, Montréal, August.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Menlo Park. AAAI Press/MIT Press.

Eugene Charniak. 2000. A maximum-entropy inspired parser. In *Proceedings of NAACL-2000.*

Stanley Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques. In *Proceedings of the 34th Meeting of the Association for Computational Linguistics (ACL '96).*

Stanley F. Chen and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Carnegie Mellon University, February.

Stanley Chen. 1995. Bayesian grammar induction for language modeling. In *Proceedings of ACL.*

Stanley Chen. 1996. *Building Probabilistic Models for Natural Language.* Ph.D. thesis, Harvard University.

Zhiyi Chi. 1999. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.

Noam Chomsky. 1959. On certain formal properties of grammars. *Information and Control*, 2:137–167.

N. Chomsky. 1965. *Aspects of the Theory of Syntax.* MIT Press, Cambridge, MA.

Noam Chomsky. 1981. *Lectures on Government and Binding.* Foris, Dordrecht.

Noam Chomsky. 1995. *The Minimalist Program.* MIT Press, Cambridge, MA.

M. Collins and J. Brooks. 1995. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, Cambridge, MA.

Michael Collins, Jan Hajič, Lance Ramshaw, and Christoph Tillmann. 1999. A statistical parser for Czech. In *Proceedings of ACL.*

Michael J. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th ACL and 8th European ACL*, pages 16–23, Madrid, July.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms.* MIT Press.

Ido Dagan, Fernando Pereira, and Lillian Lee. 1994. Similarity-based estimation of word cooccurrence probabilities. In *Proceedings of ACL*, New Mexico State University, June.

Ido Dagan, Lillian Lee, and Fernando Pereira. 1997. Similarity-based methods for word sense disambiguation. In *Proceedings of the 35th ACL and 8th European ACL*, pages 56–63.

Carl de Marcken. 1995. On the unsupervised induction of phrase-structure grammars. In *Proceedings of the Third Workshop on Very Large Corpora.* Also known as: "Lexical heads, phrase structure, and the induction of grammar.".

Carl de Marcken. 1996. *Unsupervised Language Acquisition.* Ph.D. thesis, MIT.

Sylvain Delisle and Stan Szpakowicz. 1997. Extraction of predicate-argument structures from texts. In *Proceedings of the 2nd Conference on Recent Advances in Natural Language Processing (RANLP-97)*, pages 318–323, Tzigov Chark (Bulgaria), September.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. Ser. B*, 39(1):1–38. With discussion.

Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Anna-Marie DiSciullo and Edwin Williams. 1987. *On the Definition of Word.* MIT Press, Cambridge, MA.

Margaret C. Donaldson. 1978. *Children's Minds.* Norton.

David Dowty. 1991. Thematic proto-roles and argument selection. *Language*, 67(3):547–619.

G. Duffy. 1987. *Language, Politics, and Method: Grounding a Computational Hermeneutic.* Ph.D. thesis, Department of Political Science, Massachusetts Institute of Technology, August.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the 37th ACL*, pages 457–464, University of Maryland, June.

Jason Eisner and Giorgio Satta. 2000. A faster parsing algorithm for lexicalized tree-adjoining grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+5)*, Paris, May.

Jason Eisner. 1996a. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 79–86, Santa Cruz, June.

Jason Eisner. 1996b. An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, Institute for Research in Cognitive Science, Univ. of Pennsylvania.

Jason Eisner. 1996c. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 5th International Workshop on Parsing Technologies*, pages 54–65, MIT, Cambridge, MA, September.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers.

Jason Eisner. 2001. Expectation semirings: Flexible EM for finite-state transducers. In Gertjan van Noord, editor, *Proceedings of the ESSLLI Workshop on Finite-State Methods in Natural Language Processing.* Extended abstract.

Joseph Emonds. 1976. *A Transformational Approach to English Syntax: Root, Structure-Preserving, and Local Transformations.* Academic Press, New York.

David Eppstein. 1998. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673.

Samuel David Epstein and Norbert Hornstein, editors. 1999. *Working Minimalism*, volume 32 of *Current Studies in Linguistics.* MIT Press, Cambridge, MA.

Murat Ersan and Eugene Charniak. 1995. A statistical syntactic disambiguation program and what it learns. Technical Report CS-95-29, Department of Computer Science, Brown University, October.

Roger Evans, Gerald Gazdar, and David Weir. 2000. 'Lexical rules' are just lexical rules. In A. Abeillé and O. Rambow, editors, *Tree Adjoining Grammars: Linguistic, Formal and Computational Properties*, CSLI Lecture Notes, pages 71–100. Stanford University.

Charles Fillmore. 1968. The case for case. In Emmon Bach and Robert T. Harms, editors, *Universals in Linguistic Theory*, pages 1–88. Holt, Rinehart, and Winston, New York.

Daniel Paul Flickinger. 1987. *Lexical Rules in the Hierarchical Lexicon.* Ph.D. thesis, Stanford University.

Sandiway Fong. 1991. *Computational Properties of Principle-Based Grammatical Theories.* Ph.D. thesis, Massachussetts Institute of Technology.

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan Sag. 1985. *Generalized Phrase Structure Grammar.* Blackwell, Oxford.

Gerald Gazdar. 1981. Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, 12:155–184.

Lila Gleitman. 1990. Structural sources of verb learning. *Language Acquisition*, 1(1).

David Goldberg, Daniel Nichols, Brian M. Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 32(12), December.

Miriam Goldberg. 1999. An unsupervised model for statistically determining coordinate phrase attachment. In *Proceedings of the 37th ACL (Student Session)*, pages 610–614.

John Goldsmith. 2001. Unsupervised learning of the morphology of a natural language. *Computational Linguistics.* To appear.

Anders Green. 1997. CatLA: A system for automatic acquisition of subcategorization information. Master's thesis, Göteborg University.

Anne Greenbaum. 1997. *Iterative Methods for Solving Linear Systems.* Number 17 in Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM).

Jane Grimshaw. 1994. Lexical reconciliation. *Lingua*, 92:411–430.

Ralph Grishman, Catherine Macleod, and Adam Meyers. 1994. Comlex syntax: Building a computational lexicon. In *Proceedings of COLING*, pages 268–272.

Peter Grünwald. 1996. A minimum description length approach to grammar inference. In S. Wermter, E. Riloff, and G. Scheler, editors, *Symbolic, Connectionist and Statistical Approaches to Learning for Natural Language Processing*, number 1040 in Lecture Notes in Artificial Intelligence, pages 203–216. Springer Verlag, Berlin.

Marti Hearst. 1991. Noun homograph disambiguation using local context in large text corpora. In *Proceedings of the 7th Annual Conference of the UW Centre for the New OED and Text Research: Using Corpora*, Oxford.

David Heckerman. 1995. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research Advanced Technology Division, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, March. Revised November 1996.

Donald Hindle and Mats Rooth. 1993. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

J. Hobbs and R. Grishman. 1976. The automatic transformational analysis of English sentences: An implementation. *International Journal of Computer Mathematics, Section A*, 5:267–283.

Thomas Hofmann and Jan Puzicha. 1998. Statistical models for co-occurrence data. AI Memo 1625, Artificial Intelligence Laboratory, MIT, February. Also available as CBCL Memo 159 from the Center for Biological and Computational Learning, MIT.

J. J. Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, pages 2554–2558.

A. S. Hornby, editor. 1989. *Oxford Advanced Learner's Dictionary of Current English*. Oxford University Press, 4^{th} edition.

Norbert Hornstein and David Lightfoot. 1981. Introduction. In Norbert Hornstein and David Lightfoot, editors, *Explanation in Linguistics: The Logical Problem of Language Acquisition*, pages 9–31. Longman, London.

M. Iida, C. D. Manning, P. O'Neill, and I. A. Sag. 1994. The lexical integrity of Japanese causatives. In *68^{th} Annual Meeting of the Linguistic Society of America*, Boston.

M. Jamshidian and R. I. Jennrich. 1993. Conjugate gradient acceleration of the EM algorithm. *Journal of the American Statistical Association*, 88(412):221–228.

Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, North-Holland, May.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of ACL*.

Christopher R. Johnson, Charles J. Fillmore, Esther J. Wood, Josef Ruppenhofer, Margaret Urban, Miriam R. L. Petruck, and Collin F. Baker. 2001. The FrameNet project: Tools for lexicon building. Version 0.7 (available online), July.

Mark Johnson. 1999. PCFG models of linguistic tree representations. *Computational Linguistics*.

Mark A. Jones and Jason M. Eisner. 1992. A probabilistic parser applied to software testing documents. In *Proceedings of National Conference on Artificial Intelligence (AAAI-92)*, pages 322–328, San Jose.

Aravind K. Joshi and Yves Schabes. 1991. Tree-adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees*. Elsevier.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163, February.

R. E. Kalman. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering*, 82:35–45, March.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3):400–401, March.

Kevin Knight and Jonathan Graehl. 1997. Machine transliteration. In *Proceedings of the 35th ACL and 8th European ACL.*

Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization—step one: Sentence compression. In *Proceedings of National Conference on Artificial Intelligence (AAAI).*

Anna Korhonen, Genevieve Gorrell, and Diana McCarthy. 2000. Statistical filtering and subcategorization frame acquisition. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong.

Anna Korhonen. 1997. Acquiring subcategorization from textual corpora. M.phil. dissertation, University of Cambridge.

Anna Korhonen. 2000. Using semantically motivated estimates to help subcategorization acquisition. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong.

H. Kucera and W. N. Francis. 1967. *Computational analysis of present-day American English.* Brown University Press, Providence, RI.

John Lafferty, Daniel Sleator, and Davy Temperley. 1992. Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pages 89–97, Cambridge, MA, October.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning.*

Joachim Lambek. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170.

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. 1998. Results of the Abbadingo One DFA Learning Competition and a new evidence-driven state merging algorithm. In *Proceedings of ICGI-98.* Proceedings to be published by Springer-Verlag's series Lecture Notes in Computer Science.

Marie Lapata and Chris Brew. 1999. Using subcategorization to resolve verb class ambiguity. In P. Fung and J. Zhou, editors, *Joint SIGDAT Conference on Empirical Methods in NLP and Very Large Corpora*, pages 266–274, College Park, Maryland.

K. Lari and S. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Raymond Lau, Ronald Rosenfeld, and Salim Roukos. 1993. Adaptive language modelling using the maximum entropy principle. In *Proceedings ARPA Human Language Technologies Workshop*, pages 81–86, March.

Y. Le Cun. 1985. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva 85*, pages 599–604.

Lillian Lee. 1997. *Similarity-Based Approaches to Natural Language Processing.* Ph.D. thesis, Harvard University. Technical Report TR-11-97.

V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dokl.*, 6:707–710.

Beth Levin. 1993. *English Verb Classes and Alternations: A Preliminary Investigation.* University of Chicago Press, Chicago, IL.

Hang Li and Naoki Abe. 1996. Learning dependencies between case frame slots. In *Proceedings of COLING*, pages 10–15.

Hang Li and Naoki Abe. 1998. Generalizing case frames using a thesaurus and the MDL principle. *Computational Linguistics*, 24(2):217–244.

Wentian Li. 1992. Random texts exhibit zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6):1842–1845.

David MacKay and Linda Peto. 1995. A hierarchical Dirichlet language model. *Journal of Natural Language Engineering,*, 1(3):1–19.

David J. C. MacKay. 1996. Density networks and their application to protein modelling. In J. Skilling and S. Sibisi, editors, *Maximum Entropy and Bayesian Methods, Cambridge 1994*, pages 259–268, Dordrecht. Kluwer.

Christopher D. Manning. 1993. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st ACL*, pages 235–242.

Manolis Maragoudakis, Katia Lida Kermanidis, Nikos Fakotakis, and George Kokkinakis. 2000. Learning subcategorization frames from corpora: A case study for Modern Greek. In *Proceedings of the Workshop on Computational Lexicography and Multimedia Dictionaries*, Kato Achaia, Greece, September. Department of Electrical and Computer Engineering, University of Patras, Greece.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Mitchell P. Marcus. 1980. *A Theory of Syntactic Recognition for Natural Language.* MIT Press, Cambridge, MA.

N. M. C. Marques, J. G. P. Lopes, and C. A. Coelho. 2000. Mining subcategorization information by using multiple feature loglinear models. In Paola Monachesi, editor, *Computational Linguistics in the Netherlands 1999: selected papers from the Tenth CLIN Meeting*. Rodopi, Amsterdam-Atlanta.

S. McConnell-Ginet. 1982. Adverbs and logical form: A linguistically realistic theory. *Language*, 58(1):144–184.

David McNeill. 1970. *Acquisition of Language: The Study of Developmental Psycholinguistics*. Harper and Row, New York.

I. Dan Melamed. 1997. A word-to-word model of translational equivalence. In *Proceedings of the 35th ACL and 8th European ACL*, page 490.

I. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. 1990. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244.

P. H. Miller. 1992. *Clitics and Constituents in Phrase Structure Grammar*. Garland, New York.

David Milward. 1994. Dynamic dependency grammar. *Linguistics and Philosophy*, 17:561–605.

Takashi Miyata, Takehito Utsuro, and Yuji Matsumoto. 1997. Bayesian network models of subcategorization and their MDL-based learning from corpus. In *Proceedings of the 4th Natural Language Processing Pacific Rim Symposium*, pages 321–326, December.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2).

Mehryar Mohri. 2000. Generic epsilon-removal algorithm for weighted automata. In *Proceedings of the Fifth International Conference on Implementation and Application of Automata (CIAA'2000)*, London, Ontario, Canada, July.

Marcelo A. Montemurro. 2001. Beyond the Zipf-Mandelbrot law in quantitative linguistics. Available from the Condensed Matter archive at www.arXiv.org.

Glyn V. Morrill. 1994. *Type-Logical Grammar: Categorial Logical of Signs*. Kluwer Academic Publishers, Dordrecht.

Radford M. Neal and Geoffrey E. Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, chapter A View of the EM Algorithm That Justifies Incremental, Sparse and Other Variants, pages 355–368. Kluwer Academic Publishers, Dordrecht. Revision of an earlier draft from February 1993.

Allen Newell. 1990. *Unified Theories of Cognition.* Harvard University Press, Cambridge, MA.

Jeffrey D. Oldham. 1999. Combinatorial approximation algorithms for generalized flow problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'99).*

Miles Osborne and Ted Briscoe. 1997. Learning stochastic categorial grammars. In T. Mark Ellison, editor, *Proceedings of the Workshop on Computational Natural Language Learning (CoNLL97)*, pages 80–87, Madrid, July. ACL.

Miles Osborne. 2000. Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of COLING*, Saarbrücken, Germany.

Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California.

Fernando Pereira, Naftali Tishby, and Lillian Lee. 1993. Distributional clustering of English words. In *Proceedings of ACL*, Ohio State University. Long version at `http://www.cs.huji.ac.il/labs/learning/Papers/clustew.ps.gz`.

S. Della Pietra, V. Della Pietra, and J. Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4).

Steven Pinker. 1989. *Learnability and Cognition.* MIT Press, Cambridge, MA.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press and Stanford: CSLI Publications, Chicago.

Detlef Prescher, Stefan Riezler, and Mats Rooth. 2000. Using a probabilistic class-based lexicon for lexical ambiguity resolution. In *Proceedings of the 18^{th} International Conference on Computational Linguistics (COLING)*, Saarbrücken, Germany.

W. H. Press, R. P. Flamery, S. A. Tenkolsky, and W. T. Vetterling. 1992. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 2nd edition.

Paul Procter, editor. 1978. *Longman Dictionary of Contemporary English.* Longman, Burnt Mill, Harlow, Essex.

Adam Przepiórkowski. 1999. *Case Assignment and the Complement-Adjunct Dichotomy: A Non-Configurational Constraint-Based Approach.* Ph.D. thesis, Universität Tübingen, Germany.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38^{th} Annual Meeting of the ACL.*

Stefan Riezler. 1998. Statistical inference and probabilistic modeling for constraint-based nlp. In B. Schröder, W. Lenders, W. Hess, and T. Portele, editors, *Computers, Linguistics, and Phonetics between Language and Speech: Proceedings of the 4^{th} Conference on Natural Language Processing (KONVENS'98)*, pages 111–124, Bonn. Lang.

Stefan Riezler. 1999. *Probabilistic Constraint Logic Programming.* Ph.D. thesis, Universität Tübingen. Available as AIMS Report 5(1), 1999, from Universität Stuttgart.

Eric Sven Ristad and Peter N. Yianilos. 1996. Learning string edit distance. Technical Report CS-TR-532-96, Princeton University, Department of Computer Science, October. Revised October 1997.

Eric Sven Ristad and Peter N. Yianilos. 1998. A surficial pronunciation model. In Helmer Strik, editor, *ESCA Workshop on Modeling Pronunciation for Automatic Speech Recognition*, Kerkrade, The Netherlands, May.

Douglas Roland, Daniel Jurafsky, Lise Menn, Susanne Gahl, Elizabeth Elder, and Chris Riddoch. 2000. Verb subcategorization frequency differences between business-news and balanced corpora: The role of verb sense. In *Proceedings of the ACL Workshop on Comparing Corpora*.

Mats Rooth, Stefan Riezler, Detlef Prescher, Glenn Carroll, and Franz Beil. 1999. Inducing a semantically annotated lexicon via EM-based clustering. In *Proceedings of ACL*.

Mats Rooth. 1995. Two-dimensional clusters in grammatical relations. In *Proceedings of the IJCAI Lexicon Workshop*, Stanford.

D. E. Rumelhart and J. L. McClelland. 1986. On learning the past tenses of English verbs. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2, pages 216–271. MIT Press, Cambridge, MA.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–364. MIT Press, Cambridge, MA.

Anoop Sarkar and Daniel Zeman. 2000. Automatic extraction of subcategorization frames for Czech. In *Proceedings of COLING*, Saarbrücken, Germany.

Yves Schabes and Stuart Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Yves Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14^{th} International Conference on Computational Linguistics*, pages 426–432, Nantes, France, August.

Lenhart K. Schubert. 1984. On parsing preferences. In *Proceedings of COLING-84*, pages 247–250, Stanford, CA.

Sabine Schulte im Walde. 2000. Clustering verbs semantically according to their alternation behaviour. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING-00)*, pages 747–753, Saarbrücken, Germany, August.

Hinrich Schütze. 1994. A connectionist model of verb subcategorization. In A. Ram and K. Eiselt, editors, *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, pages 784–788, Hillsdale, NJ. Erlbaum.

O. Sehitoglu and C. Bozsahin. 1999. Lexical rules and lexical organization. In E. Viegas, editor, *Breadth and Depth of Semantic Lexicons*. Kluwer.

Upendra Shardanand and Pattie Maes. 1995. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM Conference on on Computer-Human Interaction.*

John M. Sinclair, editor. 1987. *Collins Cobuild English Language Dictionary.* Collins, London.

Jeffrey Mark Siskind. 1996. A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, 61(1–2):39–91, October/November.

Daniel Sleator and Davy Temperley. 1991. Parsing English with a link grammar. Computer Science Technical Report CMU-CS-91-196, Carnegie Mellon University, October.

Daniel Sleator and Davy Temperley. 1993. Parsing English with a link grammar. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, pages 277–291, August.

Danny Solomon and Mary McGee Wood. 1994. Learning a radically lexical grammar. In *Proceedings of The Balancing Act Workshop*, Las Cruces, New Mexico, June. Association for Computational Linguistics, MIT Press.

R. J. Solomonoff. 1964. A formal theory of inductive inference. *Information and Control*, 7:1–22, 224–254, March and June.

Richard Sproat. 1996. Multilingual text analysis for text-to-speech synthesis. In *ECAI Workshop on Extended Finite-State Models of Language.*

Edward Stabler. 1997. Derivational minimalism. In Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 68–95. Springer.

Mark Steedman. 1990. Gapping and constituent coordination. *Linguistics and Philosophy*, 13:207–264.

Mark Steedman. 1996. A very short introduction to CCG. Ms., University of Pennsylvania, November.

Jiri Stetina and Makoto Nagao. 1997. Corpus-based PP attachment: Ambiguity resolution with a semantic dictionary. In *Proceedings of the 5th Workshop on Very Large Corpora*, pages 66–80, Beijing.

Andreas Stolcke and Stephen M. Omohundro. 1994a. Best-first model merging for hidden Markov model induction. Technical Report ICSI TR-94-003, ICSI, Berkeley, CA.

Andreas Stolcke and Stephen M. Omohundro. 1994b. Inducing probabilistic grammars by Bayesian model merging. In *Proceedings of the Second International Colloquium on Grammatical Inference and Applications (ICGI-94)*, Lecture Notes in Artificial Intelligence. Springer-Verlag.

M. K. Tanenhaus, M. J. Spivey-Knowlton, K. M. Eberhard, and J. E. Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268:632–634.

Robert E. Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, June.

Lucien Tesnière. 1959. *Elements de Syntaxe Structural.* Klincksieck, Paris.

C. A. Thompson and R. J. Mooney. 1999. Automatic construction of semantic lexicons for learning natural language interfaces. In *Proceedings of AAAI*, Orlando, FL, July.

Jorg Tiedemann. 1999. Automatic construction of weighted string similarity measures. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora.*

A. Ushioda, D. Evans, T. Gibson, and A. Waibel. 1993. The automatic acquisition of frequencies of verb subcategorization frames from tagged corpora. In B. Boguraev and J. Pustejovsky, editors, *SIGLEX ACL Workshop on the Acquisition of Lexical Knowledge from Text*, pages 95–106, Columbus, Ohio.

H. Uszkoreit and S. J. Peters. 1985. On some formal properties of metarules. Report CSLI-85-43, Center for the Study of Language and Information, Stanford University.

Takehito Utsuro, Takashi Miyata, and Yuji Matsumoto. 1998. General-to-specific model selection for subcategorization preference. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics*, pages 1314–1320, August. Extended version available online.

Gertjan van Noord and Gosse Bouma. 1994. Adjuncts and the processing of lexical rules. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 250–256, Kyoto, Japan.

A. van Wijngaarden. 1969. Report on the algorithmic language ALGOL68. *Numerische Mathematik*, 14:79–218.

Menno van Zaanen. 2000. ABL: Alignment-based learning. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 961–967, Saarbrücken.

K. Vijay-Shanker and David Weir. 1990. Polynomial-time parsing of combinatory categorial grammars. In *Proceedings of the 28th ACL.*

E. Wanner and M. Maratsos. 1978. An ATN approach to comprehension. In M. Halle, J. Bresnan, and G. A. Miller, editors, *Linguistic Theory and Psychological Reality*, pages 119–161. MIT Press.

Mort Webster and Mitchell Marcus. 1989. Automatic acquisition of the lexical semantics of verbs from sentence frames. In *Proceedings of ACL*, pages 177–184, Vancouver.

P. Werbos. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Ronald J. Williams and David Zipser. 1995. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, *Back-Propagation: Theory, Architectures and Applications*. Erlbaum, Hillside, NJ.

Paul R. Wilson. in progress. An introduction to Scheme and its implementation. Book in progress; draft available online.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, September.

Fei Xia, Martha Palmer, and K. Vijay-Shanker. 1999. Toward semi-automating grammar development. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99*, Beijing, November.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of ACL.*

D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of ACL*, pages 189–196.

Jakub Zavrel and Walter Daelemans. 1997. Memory-based learning: Using similarity for smoothing. In *Proceedings of the 35th ACL and 8th European ACL.*

George K. Zipf. 1932. *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press, Cambridge, MA.

George K. Zipf. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading, MA.

A. M. Zwicky, J. Friedman, B. C. Hall, and D. E. Walker. 1965. The MITRE syntactic analysis procedure for transformational grammars. In *AFIPS Conference Proceedings: Fall Joint Computer Conference*, volume 27, pages 317–326.