Inducing Grammars from Linguistic Universals and Realistic Amounts of Supervision

by

Daniel Hunter Garrette, B.S.; M.S.Comp.Sci.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2015

Inducing Grammars from Linguistic Universals and Realistic Amounts of Supervision

Daniel Hunter Garrette, Ph.D.
The University of Texas at Austin, 2015

Supervisors: Jason Baldridge and Raymond J. Mooney

The best performing NLP models to date are learned from large volumes of manually-annotated data. For tasks like part-of-speech tagging and grammatical parsing, high performance can be achieved with plentiful supervised data. However, such resources are extremely costly to produce, making them an unlikely option for building NLP tools in under-resourced languages or domains.

This dissertation is concerned with reducing the annotation required to learn NLP models, with the goal of opening up the range of domains and languages to which NLP technologies may be applied. In this work, we explore the possibility of learning from a degree of supervision that is at or close to the amount that could reasonably be collected from annotators for a particular domain or language that currently has none. We show that just a small amount of annotation input — even that which can be collected in just a few hours — can provide enormous advantages if we have learning algorithms that can appropriately exploit it.

This work presents new algorithms, models, and approaches designed to learn grammatical information from weak supervision. In particular, we look at ways of intersecting a variety of different forms of supervision in complementary ways, thus lowering the overall annotation burden. Sources of information include tag dictionaries, morphological analyzers, constituent bracketings, and partial tree annotations, as well as unannotated corpora. For example, we present algorithms that are able to combine faster-to-obtain type-level annotation with unannotated text to remove the need for slower-to-obtain token-level annotation.

Much of this dissertation describes work on Combinatory Categorial Grammar (CCG), a grammatical formalism notable for its use of structured, logic-backed categories that describe how each word and constituent fits into the overall syntax of the sentence. This work shows how linguistic universals intrinsic to the CCG formalism itself can be encoded as Bayesian priors to improve learning.

The Dissertation Committee for Daniel Hunter Garrette certifies that this is the approved version of the following dissertation:

Inducing Grammars from Linguistic Universals and Realistic Amounts of Supervision

Committee:

Jason Baldridge, Supervisor

Raymond J. Mooney, Co-Supervisor

Pradeep Ravikumar

James G. Scott

Noah A. Smith

Acknowledgments

I would like to thank my co-authors: Jason Baldridge, Chris Dyer, and Noah Smith. I would also like to thank Ray Mooney, Katrin Erk, James Scott, Liang Sun, Jason Mielens, Yoav Goldberg, Slav Petrov, Oscar Täckström, Sujith Ravi, Kyle Jerro, Vijay John, Jim Evans, the members of the UT Natural Language Learning group, and the anonymous reviewers for their feedback.

This work was supported by: a National Defense Science and Engineering Graduate (NDSEG) Fellowship from the U.S. Department of Defense; grant number W911NF-10-1-0533 from the U.S. Army Research Office; NSF grant EIA-0303609 for the UTCS Mastodon Cluster; and a grant from the Texas Advanced Computing Center.

Table of Contents

Chapter 1
Introduction

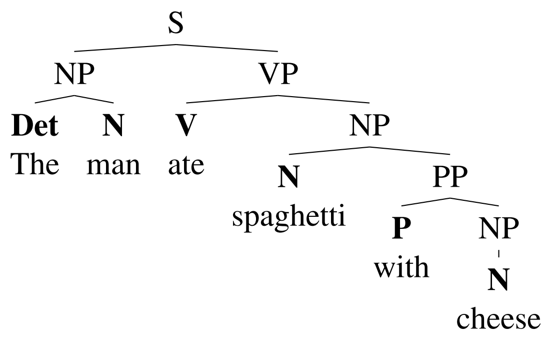
The field of natural language processing (NLP) has seen great success in the development of models built through the use of statistical machine learning techniques. However, the best of these models are trained using large amounts of fully-annotated data as supervision. For languages and domains for which we do not have access to large annotated corpora, it is necessary to use approaches that are able to learn from less input.

This dissertation is concerned with *weakly-supervised* learning for models of natural language syntax. In particular, we want to be able to learn from only small amounts of cheaply-obtained annotations from people — either expert linguists, or non-experts. We accomplish this goal by using the little available data, in conjunction with universal, cross-lingual properties common to natural language grammars, to construct *inductive biases* that can be used to learn better models. By combining our prior beliefs about how natural languages function with the statistics gleaned from larger amounts of raw text within a Bayesian setting, we are able to encourage our models to estimate better parameters.

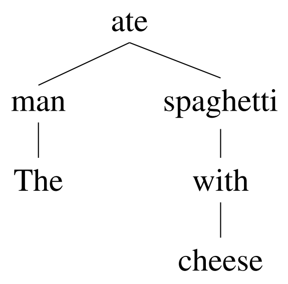
We explore a variety of syntactic tasks, including part-of-speech tagging, Combinatory Categorial Grammar (CCG) supertagging, and CCG parsing. Our experiments empirically demonstrate the benefit of these universal biases, and show that it is possible to train good models from very small amounts of human input.

1.1 Syntactic Analysis

Most contemporary NLP systems are designed around a *pipeline* architecture in which each stage in a chain of programs receives some input, performs a transformation or analysis on that input, and sends the output to the next stage of the pipeline. Such a pipeline usually begins with raw text (for example, text pulled from the web). The system may start by tokenizing (separating punctuation from



(a) A constituent parse for the sentence, with part of speech tags (bold) written above words.



(b) An unlabeled dependency parse.

Figure 1.1: Alternative parses for the sentence "The man ate spaghetti with cheese".

words), then segmenting the text into sentences.

At this point, it is typical for an NLP system to perform some form of *syntactic analysis* on the tokenized sentences. This often means tagging each token in the sentence with its part of speech (POS) before running a parser to produce a full syntax tree that describes the grammar of the sentence. The syntax tree breaks the sentence into *constituents*, and shows how those constituents assemble. An example constituent parse tree is shown in Figure 1.1a. Alternatively, one might convert this sequence of POS tags into a *dependency tree*, in which words are directly connected according to their relationships (see Figure 1.1b).

The ability to produce such analyses may be of intrinsic value to linguists wishing to conduct large-scale analyses of language usage; by parsing large corpora, one may be able to compare empirical observations about grammatical usage, or discover new, unexpected linguistic patterns (Sinclair, 1992; Wallis, 2007). Work in NLP, however, is generally concerned with using these analyses as inputs to higher-level language-processing tasks. Previous researchers have demonstrated the value of syntactic analyses in a wide range of applications including machine translation (Gildea, 2004; Chiang, 2005; Weese et al., 2012), information extraction (Ramshaw et al., 2001; Viola and Narasimhan, 2005), speech recognition (Chelba and Jelinek, 1998), text correction (Shieber and Tao, 2003), and semantic parsing (Zettlemoyer and Collins, 2005).

1.2 Statistical Approaches

Early NLP research was focused on the development of *symbolic* systems. This typically meant that taggers and parsers were built from hand-written rules describing the grammar of a language. This approach evolved naturally from what had long been the standard in theoretical linguistics, where the idea of constituent grammar dates back to Wundt (1900) and Bloomfield (1914, 1933). Chomsky (1956) later introduced the Context-Free Grammar (CFG), which has become the dominant mathematical model of natural language phrase structure. The CFG formalism has also had particular appeal for NLP researchers due to the abundance of efficient algorithms for parsing and learning. Symbolic approaches, however, have inherent drawbacks. In particular, they require enormous rulebanks that must be developed by expert linguists at great expense, and may be fairly inflexible, making it difficult to adapt them to new domains or changing patterns of language use.

Since the early 1990s, most NLP work has shifted to *statistical* approaches, the kinds of approaches we use in this dissertation. Instead of writing rules by hand, researchers began to use statistical *machine learning* techniques to automatically recognize patterns in tag sequences and parse trees, thus inducing probabilistic models of grammar simply by seeing example analyses (Lari and Young, 1990; Jelinek et al., 1990; Ney, 1991; Kupiec, 1992). Perhaps most importantly for our purposes, these empirically-driven approaches have the major advantage that they learn directly from the data itself, meaning that the parameters of the learned models will reflect the way the language is actually being used.

The most accurate statistical NLP tools to date are built using *supervised* machine learning approaches, in which the training data contains pairs of inputs and their corresponding expected outputs. For a task like POS-tagging, this means sentences paired with full tag sequences; for parsing, it might mean either bare sentences or POS-tagged sentences paired with full parse trees. Research on supervised learning has produced extremely effective taggers and parsers — for those languages for which annotated corpora are available. The best POS taggers for the Penn Treebank (Marcus et al., 1993), a million-word corpus of phrase-structure parse trees of English news text, have accuracies above 97% (Manning, 2011). The best constituent parsers on the same data exceed 91% F1 scores (Charniak and Johnson, 2005; Petrov and Klein, 2007).

These supervised approaches, however, necessarily require large amounts of annotated data in the target language or domain. Large treebanks are simply not a viable option for under-studied languages due to a lack of expert linguists who work on those languages, a lack of time available for those experts to spend on annotation efforts (since they are presumably busy with their own research), and especially a lack of funding to pay experts for their time and effort. As a result, there are currently not more than a few dozen languages with non-negligible amounts of machine-readable data (Maxwell and Hughes, 2006; Abney and Bird, 2010). For the thousands of languages and countless domains for which these resources are not available, alternative approaches must be explored.

The other end of the spectrum from supervised learning is appropriately called *unsupervised* learning. Under this paradigm, algorithms attempt to divine linguistic structure without the benefit of fully annotated example outputs (like parse trees). Early research in this area explored various ways that guidance could be given to the learning algorithms, without resorting to giving full examples; this often took the form of various constraints that restrict the parameter space that the algorithms must examine. For POS-tagger induction, this often meant, in addition to unannotated text, a restricted set of valid tags would be given for each word type so that, for example, the algorithm would not learn that "the" should be a verb (Merialdo, 1994). For parsing, this could mean restrictions on which words may be dependents of other words (Carroll and Charniak, 1992) or specification of which strings of tokens are constituents (Pereira and Schabes, 1992).

Ultimately, the most successful unsupervised approaches have tended to be those that define a generative probabilistic model and estimate its parameters using techniques such as maximum likelihood estimation with the Expectation-Maximization (EM) algorithm (Merialdo, 1994; Ney, 1991; Klein and Manning, 2002, 2004).

1.3 Weakly-Supervised Learning

This dissertation is concerned with learning models from so-called *weak supervision*. While this term could mean many things, we take it to broadly mean learning without abundant amounts of data, and without full, expert-produced annotations. We examine these issues for a variety of NLP tasks, and explore a variety of forms of supervision for each of those tasks, in order to help us (and other researchers) understand the trade-offs between various forms and amounts of annotation. The ultimate goal of this line of research is to make it possible to efficiently build NLP tools for languages and domains for which it is not feasible to acquire traditional forms of supervised training data. Dealing with this paucity of resources is what drives the questions that underlie this dissertation: Can we train tools for syntactic analysis from less supervision, or less-costly supervision, and how can we make better use of the data that we have available?

Broadly speaking, our approach is to use whatever information we have available to construct effective *inductive biases* (Mitchell, 1980) that will help guide our learning algorithms toward finding better parameters. These biases will encode our basic beliefs about how the models should look. In this work, we examine two complementary approaches to inducing inductive biases: generalizing small amounts of human-provided guidance, and encoding universal grammatical principles.

1.3.1 Minimal input from annotators

While it is certainly true that for most under-studied languages and domains, it is not possible to acquire the amount of annotation needed to train state-of-the-art NLP tools, it is almost always possible to acquire *some* information from a knowledgeable person. With a very small annotation budget, it becomes important to figure out how best to allocate our funds. The goal of a small annotation effort is to maximize the benefit (measured in percent-improvement of the final model) given the fixed budget (measured in money). This requires optimizing according to several variables.

We must first decide what annotation tasks should be completed. The tasks we choose should be fast to complete, making efficient use of the annotators' time, and they should produce data that is valuable to the model. We then need to decide who should perform the tasks. Expert linguists may be able to provide complex annotations with high quality, but there are not very many of them who are knowledgeable about any particular language, they are expensive to hire, and they have limited amounts of time that can be dedicated to annotation tasks. Thus, having experts perform simple tasks would be a poor allocation of budget. Finally, we must decide how time should be dedicated to each annotation task. It might be best to have a small amount of difficult, but highly valuable tasks that can be done only by experts, complemented with a larger amount of shallower annotation that can be done by non-experts at lower cost.

Once we have these annotations, we need to be able to extract as much information out of them as we can to achieve the best performance we can on our models. Likely this will involve analyzing them carefully with a procedure that generates an initial starting estimate for the parameter values based on whatever information is available. Additionally, it may require the use of procedures that automatically generalize from the small sample of annotator-provided data to a larger (possibly noisy) corpus of pseudo-annotated data.

1.3.2 Universal grammar

The concept of *universal grammar*, in the strictest sense, refers to a theory promoted by Chomsky (1965) which argues that all natural languages have a common underlying structure, and that the human brain is hard-wired with a small set of grammar rules that give us the ability to learn language by mapping what we hear onto that structure. For our purposes, we use a weaker definition of *universal*, in which we recognize that, cross-lingually, languages tend to share certain broad grammatical properties. Of particular value to us is the observation that natural languages seem to follow the principle that simpler syntactic structures tend to be more common than complex ones. With prior beliefs about how languages work, we can design statistical models that are biased toward simpler grammars, encouraging them to assign simpler syntactic analyses to sentences.

Chomskian theories assume that human language learning is highly constrained by the set of representations that can be supported by the architecture of the brain (Goldwater, 2007). Thus, computational approaches that have employed the concept of universal grammar in the past have tended to treat these universal characteristics as a set of hard constraints on the space of possible grammars that can be learned (Villavicencio, 2002). Since we are interested in a probabilistic modeling approach, we have the advantage that we can treat universal grammatical information as *soft* constraints (Goldwater, 2007). This soft approach is consistent with our views of language learning as statistical: the grammar of natural languages is known to be very flexible, but it has very strong tendencies, such as the tendency to prefer simpler syntactic structures when possible. Soft constraints are able to guide the learner toward more appropriate grammars, while allowing these preferences to be overridden when there is compelling contradictory evidence in the data.

This approach can be seen as a linguistic argument for the Minimum Description Length (MDL) principle (Rissanen, 1995), or "Occam's razor". The MDL principle states that when we are attempting to find the best hypothesis to describe a set of examples, we should prefer the hypothesis that is compact in length, but that describes the data with reasonable accuracy. For our task, we are attempting to find the grammar that describes a set of example sentences. Thus, the "length" in question is the grammar itself: word-tag relationships, grammatical production rules, etc. The MDL principle says that we should choose a grammar that minimizes the number of relationships and rules used, or alternatively, that maximizes the amount of rule reuse, as it seeks to describe all example sentences. The MDL principle ensures a balanced model: neither a model that overfits the data by choosing parameters are too closely tied to data, nor an over-simplified model that describes the data poorly (Villavicencio, 2002).

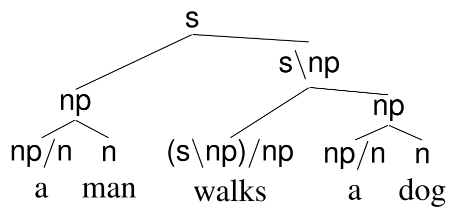
Given our desire to train NLP models in low-supervision scenarios, the possibility of constructing inductive biases out of universal properties of languages is enticing: if we can do this well, then it only needs to be done once, and can be applied to any language or domain without the need to collect additional new annotation data. One of the main driving factors behind the development of weakly-supervised learning techniques is that we want to be able to build NLP tools for new domains and languages quickly, so minimizing the amount of new data that must be collected is crucial.

Furthermore, such universal biases help to alleviate the issues of domain-dependence that cause problems for supervised approaches (Smith, 2006). If the training data is domain-specific, then models trained using it are unlikely to perform well when applied to other novel domains (Daumé III and Marcu, 2006). Universal biases are domain-independent, meaning a reduction in the amount of new-domain information that is required for adaptation.

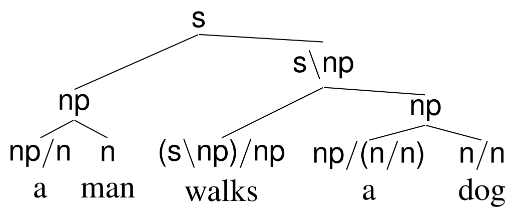
This work is similar in spirit to, for example, Collins (1999), who demonstrated the value of capturing constituent "heads"; Naseem et al. (2010), who showed that significant gains can be made in the area of unsupervised dependency parsing by starting with a small set of universal dependency rules that guide learning by specifying a few key head-argument relationships; or Cohn et al. (2010), who developed inductive biases in a Bayesian setting to encourage sparse grammars with shallow productions while inducing Tree Substitution Grammars.

We take a different path, working within the Combinatory Categorial Grammar (CCG) formalism. The CCG framework requires that each word token in a sentence be associated with a complex *category* that dictates its relationships to surrounding constituents — effectively dictating its role in the syntax of the sentence. Our strategy is to encourage our models to prefer simpler syntactic structures by developing Bayesian priors that encode the idea that we should universally prefer simpler, more cross-linguistically plausible categories. By grounding our priors in CCG, we are building our inductive biases directly into a true grammatical formalism.

Figure 1.2 shows a pair of CCG parse trees with alternate analyses for the same sentence: "a man walks a dog". The sentence contains two noun phrases: "a man" and "a dog". In Figure 1.2a, these noun phrases are analyzed in the same way, with the binary production '$\mathsf{np} \rightarrow \mathsf{np/n}\ \mathsf{n}$'. This repetition, and the fact that the word "a" receives the same category $\mathsf{np/n}$ in both cases, conforms to our MDL-inspired



(a) A simple parse in which both noun phrases are analyzed the same way, and the word "a" is assigned the same category on both occurrences.



(b) An overly-complex parse in which there are two different noun phrase analyses, the word "a" is assigned two different categories (one of which is more complex), and the word "dog" is given a more complex category.

Figure 1.2: Alternate CCG parse trees for the sentence "a man walks a dog".

belief that the optimal grammar will be one that does not contain more complexity than necessary. The analysis in Figure 1.2b uses *different* production rules for each of the noun phrases, and different categories for the word "a", thus increasing the length of the grammar. Furthermore, the categories chosen for words "a" and "dog" in the second noun phrase, np/(n/n) and n/n, are themselves more complex than the categories in the first noun phrase, np/n and n. As we will see beginning in Chapter 3, there is a positive interaction between biasing toward small rulesets and small categories: if we encourage the grammar to use simpler categories, then we will be further biasing the model toward reusing rules — specifically reusing the rules that contain those simpler categories.

While there has been much work in computational modeling of the interaction between universal grammar and observable data in the context of studying child language acquisition (e.g. Villavicencio (2002); Goldwater (2007); Pearl and Goldwater (In press)), we are interested in applying these principles to the design of models and learning procedures that result in better tagging and parsing tools.

1.3.3 Combining the universal with the empirical

In order to train the best weakly-supervised NLP tools we can, it is important that we make the best possible use of all language information we have available; this means both efficiently collecting the right kinds of annotations from people, and combining them with the right universal inductive biases. Thus, our work brings together the long-standing notion of universal grammar with contemporary data-driven statistical machine learning techniques.

Our work on CCG in performed using Bayesian techniques. A grammar can be thought of as the combination of structure and parameters; the CCG formalism and corresponding universal biases give us the structure, while the data-driven machine learning estimates the parameters. The Bayesian framework seems to be well-matched to this approach since our inductive biases — those derived from universal grammar principles, human-annotated data, and estimations based on unannotated data — can be encoded as *priors*, and we can use Markov chain Monte Carlo (MCMC) inference procedures to automatically blend these biases with the way language is seen in actual corpora. This allows us to conveniently incorporate the available linguistic information into proven statistical techniques, and results in learned models that perform better than that which could be learned from the text alone.

Given that NLP systems seem to be most successful when the kind of syntactic analysis performed is tailored to the specific end task, highly-adaptable learning methods seem particularly important. For approaches like the work of Chiang (2005) or Zettlemoyer and Collins (2005), where syntactic induction is performed *within* a larger task (phrase-based translation and semantic parsing, respectively), the ability to combine universal constraints with empirical data seems particularly valuable.

In Table 1.1, we summarize the sources of weak supervision discussed in this dissertation. Each column defines a broad genre of supervision: universal grammar constraints, word-tag association lexicons, or various kinds of annotation. Each of these supervision sources has its own benefits but also its own acquisition challenges, from universal information that only needs to be done once but must be done by an expert, to fast annotation that requires less expertise but must be done on many individual documents. Importantly, all of these sources of information are *complementary*, since they each provide their own view of the underlying grammatical structure.

	Grammar constraints	**Lexicons**	**Shallow annotations**
Examples	"prefer simpler analyses"	word-to-tagset mappings	brackets, dependencies
Scope	universal, cross-lingual	specific to the language or domain	can annotate many documents
Expertise required	expert	expert	non-expert
Ease to obtain	based on grammar formalism	linguistic knowledge, but can automatically expand a (very) small lexicon into a large one	can be fast, noisy, and incomplete

Table 1.1: Sources of weak supervision.

1.4 This Dissertation

In this work, we present new algorithms, models, and approaches designed to learn grammatical information from weak supervision. On the annotation front, we look at ways of intersecting a variety of different forms of supervision in complementary ways, thus lowering the overall annotation burden. Sources of information include tag dictionaries, morphological analyzers, and constituent bracketings, as well as unannotated corpora. In terms of algorithms, we present new approaches that are able to combine simple, fast-to-obtain annotation with unannotated text to remove the need for slower-to-obtain types of annotation. We also present new parsing models that are able to capture and exploit linguistic information for better learning.

In Chapter 2, we show that for the task of part-of-speech (POS) tagging, a simple building-block of grammatical analysis, it is possible to combine different forms of simple annotation collected from a linguist in just a few hours to train taggers that provide accuracies comparable to existing systems that use *years* worth of annotation to train. We accomplish this by introducing machine learning algorithms that are able to generalize from the tiny amount of initially-provided annotation data that covers only a small fraction of the language, to automatically induce some amount of information on every token in a previously-unannotated corpus.

The remainder of this dissertation describes work on Combinatory Categorial Grammar (CCG), a grammatical formalism notable for its use of structured, logic-backed categories that describe how each word and constituent fits into the overall syntax of a sentence. In Chapter 3, we move from POS-tagging to weakly-supervised learning of a sequence model that labels each token in a text with a one of these CCG categories, a task known as *supertagging*. These supertags are a valuable stage in grammatical analysis because they provide more grammatical information than simple POS tags without requiring the modeling of the full grammar of a sentence. While the task of supertagging is substantially more difficult than POS-tagging due to the very high number of potential tags for any given word, we are able to show how linguistic universals intrinsic to the CCG formalism itself — in particular, the knowledge that certain categories are more likely than others, and that certain pairs of categories are likely to be in close proximity — can be encoded as Bayesian priors to improve learning when only limited supervision is available. We further show that we can build on the techniques we developed for POS-tagging that allowed us to automatically extract valuable information from only weak supervision in order to combine corpus-specific information with these universals to learn even better supertagging models.

In Chapter 4, we extend these principles to full parsing models by using our prior on the relative likelihoods of categories throughout the parse tree, not just at the supertag level. We present a novel model in §4.3 that shows how the other CCG principle used in the supertagger — that certain pairs of categories are more likely to be in close proximity — can be incorporated into our parsing model to complement the bias toward simpler categories. In §4.4, we demonstrate that fast, noisy, incomplete, cheap-to-obtain *bracket* annotations, collected from human annotators in only a few hours, can provide positive benefit when training a parser. Finally, in §4.5, we develop a novel nonparametric parsing model that is able to use an infinite set of CCG categories to learn parsers in the face of incomplete information.

Chapter 2
Minimally-Supervised POS Tagging

Learning accurate part-of-speech (POS) taggers based on plentiful labeled training material is generally considered a solved problem. The best taggers obtain accuracies of over 97% for English newswire text in the Penn Treebank, which can be considered as an upper-bound that matches human performance on the same task (Manning, 2011). However, this story changes as soon as the amount or quality of annotation is reduced: learning a POS-tagger from less training data, or different kinds of training data, remains a difficult problem. In particular, there has been recent work on efforts to induce POS tags without labels (Christodoulopoulos et al., 2010); learn from POS-tag dictionaries (Merialdo, 1994; Smith and Eisner, 2005; Ravi et al., 2010b), incomplete dictionaries (Hasan and Ng, 2009) and human-constructed dictionaries (Goldberg et al., 2008); bootstrap taggers for a language based on knowledge about other languages (Das and Petrov, 2011); and create supervised taggers for new, challenging domains such as Twitter (Gimpel et al., 2011).

Learning from just a partial tag dictionary — a mapping from some small set of known words to their potential tags — and a raw, unlabeled corpus has particularly strong appeal since it seemingly requires substantially less annotation effort. This type of learning, often characterized as unsupervised or weakly-supervised training, but which we call *type-supervised* learning to distinguish it from training only from raw text, seems particularly relevant for developing taggers for languages or domains in which no annotated resources exist. While it may take substantial effort to produce a corpus of labeled sentences for *token-supervised* learning, pro-

The contents of this chapter are based on the following publications: Dan Garrette and Jason Baldridge. 2012. Type-supervised hidden Markov models for part-of-speech tagging with incomplete tag dictionaries. In Proc. of EMNLP; Dan Garrette and Jason Baldridge. 2013. Learning a part-of-speech tagger from two hours of annotation. In Proc. of NAACL; Dan Garrette, Jason Mielens, and Jason Baldridge. 2013. Real-world semi-supervised learning of POS-taggers for low-resource languages. In Proc. of ACL. My co-authors worked with me to develop the ideas underlying this work.

ducing a dictionary may be relatively cheap.

There is real need for the ability to learn taggers using very little data: only a tiny fraction of the world's languages have enough data for standard supervised models to work well (Abney and Bird, 2010). The collection or development of resources is a time-consuming and expensive process, which creates a significant barrier for the development of tools for under-studied languages where there are few experts and little funding. It is thus important to develop approaches that achieve good accuracy based on the amount of data that can be reasonably obtained, for example, in just a few hours by a linguist doing fieldwork on a language that he or she does not even speak natively.

The work presented in this chapter is threefold. First, we present our data annotation effort in which we asked linguists, in time-controlled sessions, to provide a variety of manual annotation on multiple languages, including truly low-resource languages that have not been studied in NLP and thus, for which there really are not resources. Second, we present a training procedure that is able to train a reasonably-accurate POS-tagger from the amount of data that can be collected from a linguist in only a few hours by automatically generalizing the annotations to cover the entire language. And third, we perform a series of experiments to evaluate the trade-offs of using various kinds and amounts of annotation, with the goal of gaining insights that will allow us to make recommendations to researchers wishing to train POS-taggers for languages with no existing resources (Garrette and Baldridge, 2012, 2013; Garrette et al., 2013).

2.1 Background

Type-supervised POS-tagger learning has a long history in NLP, with most work centered on using Expectation-Maximization (EM) to train Hidden Markov Model (HMM) taggers (Kupiec, 1992; Merialdo, 1994). Early research appeared to show that learning from types works nearly as well as learning from tokens, with researchers in the 1990s obtaining accuracies up to 96% on English (e.g. Kupiec (1992)). However, the tag dictionaries in these cases were obtained by automatic extraction from corpora with *token*-level annotations, thus bending the parameters of the scenario that they were purporting to solve. While replicating earlier experiments, Banko and Moore (2004) discovered that performance was highly dependent on cleaning tag dictionaries using statistics gleaned from the token-level frequencies. Removing low-frequency entries from the dictionary greatly simplifies the job of a type-supervised HMM since it no longer needs to entertain entries for uncommon word-tag pairs (or mistaken pairs due to annotation errors), which otherwise stand on equal footing with the common ones and have a strong tendency to derail learning. When a full, noisy tag dictionary is employed, Banko and Moore found accuracies drop from 96% to 77%, while Smith and Eisner (2005) saw smaller drops, but with a more sophisticated *contrastive estimation* model.

Following on Banko and Moore's findings, some subsequent researchers have sought to improve performance in the face of full, noisy dictionaries. However, most of this work still relies on unrealistic assumptions about the tag dictionaries they use as input (Toutanova and Johnson, 2008; Ravi and Knight, 2009; Hasan and Ng, 2009). For example, tag dictionaries are typically constructed from every word-tag pair in the corpus, including the data that is to be used as raw, unlabeled text during training. This is troublesome because it means that every word encountered during training will have an entry in the tag dictionary, substantially lowering the degree of ambiguity that the learning algorithm must contend with. But in real-world scenarios, perfect corpus knowledge is never possible, and designing algorithms around this assumption is highly problematic because the scenarios for which type-supervised learning makes the most sense — the low-resource scenarios — are precisely those scenarios in which one must learn using a corpus populated primarily with unknown words. Even more egregiously, most work constructs the tag dictionary additionally using the *test corpus* word-tag pairs, meaning that the evaluation of these taggers does not measure how they perform on sentences that contain unseen words or unseen word-tag pairs — a certainty in any real use of a trained tagger — which allows their work to side-step the most difficult cases for the tagger to handle. Worse, for rare words there will almost certainly be exactly one possible tag listed in the dictionary, meaning that the tagger will achieve perfect performance on the words that should be maximally ambiguous since the model has no information about them!

The justification for the use of labeled-corpus-extracted tag dictionaries is that they are a proxy for dictionaries produced by linguists. But even if we do away with the assumption of completeness exercised by most previous work, this setup still overstates the effectiveness of dictionary-based training. In addition to pruning based on should-be-unavailable token-level annotations, dictionaries extracted from corpora are unrealistically biased towards including only the most likely tag for each word type, and especially only the tag most likely in the given *domain*, resulting in a cleaner dictionary than one would find in real scenario. Further, tag dictionaries extracted from annotated tokens benefit from the annotation process of labeling *and* review *and* refinement over an extended collaboration period by teams of trained expert annotators.

2.2 Setting up a Realistic Training Scenario

We are interested in developing tools for low-resource languages, and, thus, must directly confront the issue of how to bootstrap a tagger when no annotation is available. Rather than attempting to "simulate" low-resource scenarios but relying on large resources, we chose to collect our own annotations in a scenario that resembles what might be possible for a linguist working in the field. For this, we set up time-constrained annotation sessions and had linguists produce annotations for truly low-resource languages with which they were familiar, but for which they were not native speakers. As professionals studying and documenting those languages, they were the ideal annotators for the experimental data since they are precisely the type of individuals that this line of research is designed to help.

The annotations produced under our conditions differ in several ways from the labeled data used in previous work. Most obviously, there is less of it. Instead of using hundreds of thousands of labeled tokens to construct a tag dictionary (and hundreds of thousands more as unlabeled (raw) data for training), we only use the 1-2k labeled tokens or types provided by our annotators within the timeframe.

Our training data is also much noisier than the data from a typical corpus: the annotations were produced by a single non-native-speaker working alone for only a few hours. Therefore, dealing with the size and quality of training data were core challenges to our task. We are particularly interested in being able to assess the trade-offs between the amount of time spent on various annotation tasks, and the gains in tagger accuracy produced by the additional annotation. Therefore, we had the annotators work in specific time increments, and annotate data in multiple ways. We specifically look at the effect of four types of data collection:

1. Time spent annotating full sentences (token supervision)
2. Time spent creating tag dictionary (type supervision)
3. Time spent constructing a morphology finite state transducer (FST)
4. Amount of raw data available for training

To learn a POS-tagger from so little labeled data, we developed an approach that starts by generalizing the initial annotations to cover the entire raw corpus. Our approach uses label propagation (LP) (Talukdar and Crammer, 2009), which allows us to connect annotated information to unannotated text in a graph and push labels between them. The LP algorithm is used to infer a distribution over POS tags for every token in the raw corpus, essentially producing a soft tagging of every previously-unannotated sentence. We then apply a novel *weighted* variant of the model minimization procedure originally developed by Ravi and Knight (2009) to find an approximately-minimal set of tag bigrams needed to explain the data. This procedure allows us to induce a hard tagging of every sentence from the soft one. The hard tagging contains significantly less noise than the LP output, and, crucially, allows us to estimate sequence and word-tag frequency information even when the only annotations provided were at the type level. Using this hard tagging of a larger corpus, we can effectively estimate initial transition and emission distributions to use as a starting point for learning of an HMM using EM, which far outperforms just using EM on the raw annotations themselves.

We explore these strategies in the context of POS-tagging for Kinyarwanda and Malagasy. We also include experiments for English, pretending it is a low-resource language. The overwhelming take-away from our results is that type supervision — when backed by an effective semi-supervised learning approach that is able to induce sequence and frequency information — is the most important source of linguistic information. Also, morphological analyzers help for morphologically-rich languages when there are few labeled types or tokens (and, it never hurts to use them). Finally, performance improves with more raw data, though we see diminishing returns past 400k tokens. With just four hours of type annotation, our system obtains good accuracy across the three languages: 89.8% on English, 81.9% on Kinyarwanda, and 81.2% on Malagasy.

In order to ensure the validity of our experimental scenarios, all results presented in this work adhere to clear standards regarding the use of data. In every experiment, data is always divided into four disjoint sets: data used to construct the tag dictionary, unlabeled data to be used during training, an annotated development evaluation corpus for hyperparameter tuning, and an annotated test evaluation corpus for measuring final results.

Our results compare favorably with previous work despite using considerably less supervision and a finer-grained set of tags. For example, Li et al. (2012) use the entirety of English Wiktionary directly as a tag dictionary to obtain 87.1% accuracy on English, below our result. Täckström et al. (2013) average 88.8% across eight major languages, but for Turkish, a morphologically rich language, they achieve only 65.2%, significantly below our 81.9% for morphologically-rich Kinyarwanda.

2.3 HMM Learning

Hidden Markov models (HMMs) are well-known generative probabilistic sequence models commonly used for POS-tagging. An HMM is defined by a set of *transition* parameters $\pi_t(u)$, the probability of transitioning from tag t to tag u, and a set of *emission* parameters $\phi_t(w)$, the probability of emitting word w for a token tagged as t. The probability of a tag sequence $\mathbf{t}$ for a word sequence $\mathbf{w}$ is determined from the product of emission and transition probabilities (where $\langle \textsc{s} \rangle$ and $\langle \textsc{e} \rangle$

are special *start* and *end* tags, respectively):

$$P(\mathbf{w}, \mathbf{t}) = \pi_{\langle \text{S} \rangle}(t_1) \cdot \left(\prod_{i=1}^{N} \phi_{t_i}(w_i) \cdot \pi_{t_{i-1}}(t_i) \right) \cdot \pi_{t_N}(\langle \text{E} \rangle)$$

HMMs can be trained directly from fully-supervised, token-labeled data by calculating maximum likelihood estimates or from type-supervision (in the form of tag dictionaries that constrain the set of tags that each word may take) using the Expectation Maximization (EM) algorithm (Dempster et al., 1977).

2.3.1 Token-supervised training

When token-level tag annotations are given, an HMM can be trained simply by taking Maximum Likelihood Estimates (MLE). Because the MLE calculation will result in zero-probabilities for any emissions or transitions that were absent form the training corpus, we employ simple add-δ smoothing to assume non-zero counts for any value licensed by the tag dictionary. While more complex smoothing methods exist, this simple scheme is quite effective: an HMM trained with δ=0.1 for both transitions and emissions on a tag dictionary built from sections 00-15 of the Penn Treebank (PTB) corpus, using sections 16-18 as supervised training data, and evaluated on sections 19-21, achieves 94.0% accuracy.

2.3.2 Type-supervised training

Token-level supervision is, however, not often available. For those contexts, we must learn from some form of weaker supervision. This has traditionally been done using a combination of unlabeled text and tag dictionary. As is standard, we use Expectation-Maximization (EM) to iteratively estimate token-label counts from the unlabeled data.

The EM algorithm has two alternating phases. In the *Expectation* phase, we use the Forward-Backward algorithm (Baum, 1972) to compute the expected counts for every possible transition and emission in the raw corpus. In the *Maximization* step, we sum those expected counts and compute the MLE.

In order to begin the iterative process, the algorithm requires initial estimated transition and emission distributions so that it can compute the first round of expected counts. The simplest way to estimate these distributions would be to just assume that all distributions are uniform. However, given any available information, it is usually advantageous to attempt to pick a better initialization. Relevant information might include any token-level annotations that are available, or even estimates carefully derived from the tag dictionary and raw corpus.

When the EM algorithm terminates, the result is a final set of expected transition and emission counts from which we can compute the MLE. These counts (which may be fractional) will be non-zero for every word-tag pair allowed by the tag dictionary, and will cover every token in the raw corpus. However, there will clearly not be counts for words that do not appear in the raw corpus. But since novel words may appear in the test data, it is critical that our HMM not assume zero probabilities for these words. To avoid this scenario, we smooth the final computed expected counts from EM as we would with normal token-label counts in the fully-supervised case: by assuming they have some small fractional count δ (and adding δ to the count of every known word as well).

2.4 Data

In past POS-tagging work, tag dictionaries have typically been automatically extracted from large annotated corpora. Additionally, in most cases researchers have taken the additional step of removing tag dictionary entries that occur with low frequencies (see Banko and Moore (2004) for discussion). However, conclusions about how well these experimental scenarios, given their reliance on large labeled corpora, often do not generalize to low-resource situations.

In order to determine how our tagger learning methods would perform in true low-resource scenarios — exactly those languages for which weakly-supervised learning is needed — we designed and carried out an annotation effort in which we had linguists annotate data that we could use to train our models. Our experiments use Kinyarwanda (KIN) and Malagasy (MLG), both true low-resource languages.

KIN is a Niger-Congo language spoken in Rwanda while MLG is an Austronesian language spoken in Madagascar. Additionally, KIN is morphologically-rich, meaning that are many morphemes in the language that can be concatenated to make a wide variety of words, resulting in a higher word-type to word-token ratio than is seen in a language like English; this presents additional challenges in learning. Annotations for KIN and MLG were performed by linguistics graduate students who were familiar with the languages, but not native speakers. Our experiments are thus relevant to the reasonable context in which one has access to a linguist who is familiar with the target language and a given set of POS tags. We also used English (ENG) for experiments comparing multiple annotators on the same data, and to compare with previous work.

2.4.1 Data sources

The KIN texts are transcripts of testimonies by survivors of the Rwandan genocide provided by the Kigali Genocide Memorial Center. The MLG texts are articles from the websites[1] *Lakroa* and *La Gazette* and Malagasy Global Voices,[2] a citizen journalism site.[3] For each language, the word tokens were divided into four sets: training data to be labeled by annotators, raw training data, development data, and test data. The KIN and MLG data have 12 and 23 distinct POS tags, respectively.

The Penn Treebank (PTB) (Marcus et al., 1993) is used as ENG data. Section 01 was used for token-supervised annotation, sections 02-14 were used as raw data, 15-18 for development of the FST, 19-21 as a dev set and 22-24 as a test set. The PTB uses 45 distinct POS tags.

2.4.2 Annotation-collection tasks

We are interested in studying how different kinds of annotation affect model performance so that we might be able to make concrete suggestions to field linguists

[1]wwww.lakroa.mg and wwww.lagazette-dgi.com

[2]www.ark.cs.cmu.edu/global-voices/; wmg.globalvoicesonline.org/

[3]The public-domain data is available at github.com/dhgarrette/low-resource-pos-tagging-2013

who are collecting data in real scenarios. Therefore, we asked the annotators to perform two different time-constrained annotation tasks, which we explain below.

In the first task, *type-supervision*, the annotator was given a list of the words in the target language (ranked from most to least frequent), and they annotated each word type with its potential POS tags. The annotators were allowed to give multiple tags for each word if necessary, attempting to identify all possible tags relevant to the word. The annotators were also permitted to skip words if they were unsure of the correct tags; these words were simply ignored for training, along with all the words that were not reached due to time constraints. The word types and frequencies used for this task were taken from the raw training data and did not include the test sets.

In the second task, *token-supervision*, full sentences were annotated with POS tags. Sentences were taken from the raw corpora and presented, in order, to the annotators.[4] Again, annotators were permitted to skip words if they felt it necessary; in these (rare) cases, the skipped tokens were simply excluded from the final transition and emission counts, along with any sentences that were not reached in time.

Having both kinds of annotations allows us to investigate the relative value of each with respect to training taggers. Token-supervision provides valuable frequency and tag context (transition) information, but type-supervision produces larger dictionaries, giving higher word-type coverage. This can be seen in Table 2.1, where the dictionary size column in the table gives the number of unique word/tag pairs derived from the data.

It was important that we be able to measure tagger performance directly against the time required to produce the inputs to learning so that we could see how learning ability progressed over the course of the annotation task. To accomplish this, we had our annotators work in 30-minute increments, which allowed us to run experiments measuring the incremental benefit of an additional batch of anno-

[4]Providing the sentences in corpus-order was a poor choice. Consecutive sentences tend to be on the same topic, and thus reuse the same words, leading to fewer distinct words being annotated. A better ordering of sentences, even simply *shuffling* the order of sentences, would almost certainly perform better.

	sent.	tok.	dict.
KIN human token-level A	90	1537	750
KIN human type-level A			1798
MLG human token-level B	92	1805	666
MLG human type-level B			1067
ENG human token-level A	86	1897	903
ENG human type-level A			1644
ENG human token-level B	107	2650	959
ENG human type-level B			1090

Table 2.1: Statistics for Kinyarwanda, Malagasy, and English data annotated by two equally-experienced annotators (A and B). Annotations were provided separately at the token level and type level.

tations. Being able to measure accuracy over the course of the annotation process can help us to understand where the gains begin to diminish. Additionally, having increments for each type of annotation allows us to see how different kinds of annotations might be combined within a fixed annotation budget.

Baldridge and Palmer (2009) found that annotator expertise greatly influences effectiveness of active learning for morpheme glossing, a related task. To see how differences in annotator speed and quality impact our task, we had two different annotators complete the same two tasks for English: one who had performed that task before (the "experienced" annotator), and one who had not (the "novice"). As can be seen in Table 2.1, there are clear differences between the two annotators. Most notably, the experienced annotator worked much more quickly and, thus, provided much more data.

In the error analysis of some of our previous work we found that annotator mistakes on high-frequency tokens, such as punctuation and the word "to", can have a large negative impact on overall accuracy (Garrette and Baldridge, 2013). We sought to lessen these problems in our second data collection — the data used for the results presented here — by providing clear instructions to our annotators about the correct uses of these low-entropy tags. We felt that this step was justified because, while it is not possible to provide annotation instructions for all word types

	KIN		MLG		ENG - Exp.		ENG - Nov.	
time	type	token	type	token	type	token	type	token
0:30	362	316	329	262	225	296	96	197
1:00	801	559	660	422	910	522	210	308
1:30	1352	794	981	581	1735	782	409	469
2:00	1814	948	1363	785	2660	1036	631	646
2:30	2190	1137	1698	927	3632	1201	984	797
3:00	2539	1324	2043	1082	4561	1314	1350	953
3:30	3028	1477	2410	1221	5587	1508	1725	1091
4:00	3682	1651	2773	1378	6598	1697	2185	1220

Table 2.2: Annotations for each language and annotator (experienced and novice) as time increases. Shows the number of tag dictionary entries from type annotation vs. token.

(or even all punctuation), it is reasonable that, when designing a set of POS tags, one would have a good idea of how the punctuation and closed-class tags are intended to be used. These decisions can be trivially conveyed to the annotator while explaining the task, resulting in cleaner annotations and leaving the annotators' attention focused on the lower-frequency and open-class words.

Ngai and Yarowsky (2000) investigated the effectiveness of rule-writing versus annotation (using active learning) for chunking, and found the latter to be far more effective. While we do not explore a rule-writing approach to POS-tagging, we were interested in the impact of rule-based morphological analyzers as a component in our semi-supervised POS-tagging system.

2.4.3 Annotated data

Table 2.2 gives statistics for all languages and annotators showing progress during the 4-hour tasks. With token-annotation, tag dictionary growth is slower because high-frequency words are repeatedly annotated, producing only additional frequency and sequence information. In contrast, every type-annotation label is a new tag dictionary entry. For types, growth increases over time, reflecting the fact that high-frequency words (which are addressed first) tend to be more ambiguous

	ENG - Exp.		ENG - Nov.	
	type	tok	type	tok
0:30	0.01	0.02	0.01	0.01
1:00	0.05	0.03	0.01	0.02
1:30	0.10	0.04	0.02	0.02
2:00	0.15	0.05	0.03	0.03
2:30	0.19	0.06	0.05	0.04
3:00	0.24	0.06	0.07	0.05
3:30	0.28	0.07	0.09	0.05
4:00	0.32	0.08	0.11	0.06

Table 2.3: Tag dictionary recall against the test set for ENG annotators on type and token annotations.

and thus require more careful thought than later words. For ENG, we can compare the tagging speed of the experienced annotator with the novice: 50% more tokens and three times as many types. The token-tagging speed stayed fairly constant for the experienced annotator, but the novice increased his rate, showing the benefit of practice.

Checking the annotators' output against the gold tags in the PTB shows that both had good tagging accuracy on tokens: 94-95%. Comparing the tag dictionary entries versus the test data, precision starts in the high 80%s and falls to the mid-70%s in all cases. However, the differences in recall, shown in Table 2.3, are more interesting. On types, the experienced annotator maxed out at 32%, but the novice only reaches 11%. Moreover, the maximum recall for token annotations is much lower due to high repeat-annotation. The discrepancies between experienced and novice, and between type and token recall, explain a great deal of the performance disparity seen in the experiments.

2.5 Morphological finite-state transducers

Finite-state transducers (FSTs) can be constructed easily using regular expressions, which makes them quite useful for phonology, morphology and limited areas of syntax (Karttunen, 2001). Past work has used FSTs for direct POS-tagging

(Roche and Schabes, 1995), but this requires tight coupling between the FST and target tagset. We use FSTs for morphological analysis: the FST accepts a word type and produces a set of morphological features. If there are multiple possible analyses for a given word type, the FST returns them all. For instance the Kinyarwanda verb *sibatarazuka* "he is not yet resurrected" is analyzed in several ways:

(a) *negation, class 2, 1st-person plural, verb, "arazuk", imperative mood*
(b) *negation, class 2, NOT YET, "razuk", imperative mood*
(c) *negation, class 2, NOT YET, present tense, "zuk", imperative mood*

Class 2 refers to the category reprsenting *people* in the noun class marking system in Kinyarwanda/Bantu. The imperative mood usually indicates a command or warning.

FSTs are particularly valuable for their ability to analyze out-of-vocabulary items. By looking for known affixes, FSTs can guess the stem of a word and produce an analysis despite not having knowledge of that stem. For morphologically complex languages like KIN, this ability is especially useful. Other factors, such as a large number of morphologically-conditioned phonological changes (seen in MLG) make out-of-vocabulary guessing more challenging because of the large number of potential stems (high ambiguity).

Development of the FSTs for all three languages was done by iteratively adding rules and lexical items with the goal of increasing coverage on a raw dataset. To accomplish this on a fixed time budget, the most frequently occurring unanalyzed tokens were examined, and their stems plus any observable morphological or phonological patterns were added to the transducer. Additionally, developers searched for known morphological alternations to locate instances of phonological change for inclusion. Coverage was checked against a raw dataset which did not include the test data used for the POS experiments.

The KIN and MLG FSTs were created by English-speaking linguists who were familiar with their respective language. They also used dictionaries and grammars. Each FST was developed in 10 hours. To evaluate the benefits of more development time, a version of the English FST was saved every 30 minutes (Table 2.4).

elapsed time	tokens count	tokens pct	types count	types pct
2:00	130k	61%	2.1k	12%
4:00	159k	75%	4.1k	24%
6:00	170k	80%	6.7k	39%
8:00	182k	86%	7.7k	44%
10:00	192k	91%	10.7k	62%

Table 2.4: Coverage of the English morphological FST during development. For brevity, showing 2-hour increments instead of 30-minute segments.

	tokens		types	
	cov.	ambig.	cov.	ambig.
KIN	86%	2.62	82%	5.31
MLG	78%	2.98	37%	1.13
ENG	91%	1.19	62%	1.97

Table 2.5: Coverage and ambiguity (analyses per word type) of the final FST for each language.

2.6 Generalizing the Annotations

In order to learn from the extremely small amounts of annotation available to our system, we have designed the following training scheme. First, we generalize the initial annotations to cover the entire raw corpus. Next, we automatically clean up noise from the generalization. Then, we use the cleaned expanded annotations to initialize EM learning. Finally, we smooth the output expected counts from EM to produce an HMM, which we use to tag the raw corpus and train an Maximum Entropy Markov Model (MEMM).

In a low-resource setting, most word types will not be found in the initial tag dictionary. EM-HMM training uses the tag dictionary to limit ambiguity by restricting the set of tags with which a word may be associated, so a sparse tag dictionary is problematic because it does not sufficiently confine the parameter space. Small dictionaries also interact poorly with the model minimization of Ravi et al. (2010b): if there are too many unknown words, and every tag must be considered for them, then the minimal model will simply be the one that assumes that they all have the same tag.

For these reasons, we must automatically *expand* the initial small set of annotations into one that has coverage for most of the vocabulary appearing in the raw corpus.

2.6.1 Annotation expansion with label propagation

For the initial annotation expansion, we use label propagation (LP) — specifically, the Modified Adsorption (MAD) algorithm (Talukdar and Crammer, 2009)[5] — which is a graph-based technique for spreading labels between related items in a graph. The MAD algorithm uses the concept of an iterative graph random-walk to assign probability distributions over labels for each node in the graph, given some seed distributions on a subset of nodes. It constructs an objective based on three characteristics that the output should be close to any given seed labels and nodes that are close in the graph should have similar labelings.

Our technique for setting up the LP graph is simple: we create a node for every token in the raw corpus, a node for every annotated type and token, and then connect them all via various kinds of feature nodes. We seed the "annotated" nodes with the labels from the annotations and use the MAD algorithm to push labels from annotated nodes to unannotated tokens.

The result of the LP procedure is a tag distribution over every raw corpus token. In other words, we take a raw corpus for which we have no information about most of its vocabulary and contexts, and we induce noisy, soft annotations for every token in that corpus.

Defining the LP graph

Our LP graph has several types of nodes, as shown in Figure 2.1. The graph contains a TOK node for each token of the labeled corpus (when available) and raw

[5]The MAD implementation we used is provided through the open-source software package Junto: *github.com/scalanlp/junto*

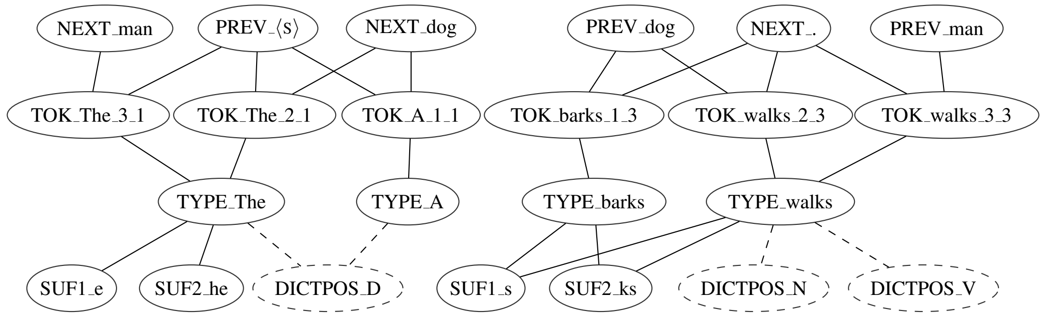


Figure 2.1: Subsets of the LP graph showing regions of connected nodes. Graph represents the sentences "A dog barks .", "The dog walks .", and "The man walks .". Nodes derived from external dictionaries are optional, and only used for some extension experiments.

corpus. Each word type has one TYPE node that is connected to its TOK nodes. Both kinds of nodes are connected with *feature* nodes.

Seeding the graph is straightforward. With token-supervision, labels for tokens are injected into the corresponding TOK nodes with a weight of 1.0. In the type-supervised case, any TYPE node that appears in the tag dictionary is injected with a uniform distribution over the tags in its tag dictionary entry.

Bigram Features The PREV_x and NEXT_x nodes represent the features of a token being preceded by or followed by word type x in the corpus. These *bigram* features capture extremely simple syntactic information.

Affix Features To capture shallow morphological relatedness, we use *prefix* and *suffix* nodes that connect word types that share prefix or suffix character sequences up to length 5. For each node-feature pair, the connecting edge is weighted as $1/N$ where N is the number of nodes connected to the particular feature.

(Optional) External Dictionary For some experiments, we also explored the effectiveness of using an optional external dictionary in the graph since this is one of the few available sources of information for many low-resource languages. Though a standard dictionary probably will not use the same POS tag set that we are targeting, it nevertheless provides information about the relatedness of various word types. Thus, we use nodes DICTPOS_p that indicate that a particular word type

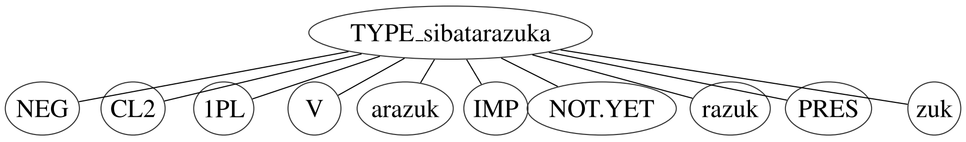


Figure 2.2: LP subgraph showing connections between a word type and its morphological features.

is listed as having POS p in the dictionary. Crucially, *these tags bear no particular connection to the tags we are predicting*: we still target the tags defined by the linguist who annotated the types or tokens used, which may be more or less granular than those provided in the dictionary. As external dictionaries, we use English Wiktionary (614k entries), `malagasyworld.org` (78k entries), and `kinyarwanda.net` (3.7k entries).[6]

Morphological Features The character n-gram affix-as-morphology approach produces many features, but only a fraction of them represent actual morphemes. Incorrect features end up pushing noise around the graph, so affixes can lead to more false labels that drown out the true labels. While affixes may be sufficient for languages with limited morphology, their effectiveness diminishes for morphology-rich languages, which have much higher type-to-token ratios. More types means sparser word frequency statistics and more out-of-vocabulary items, and thus problems for EM. Here, we modify the LP graph by supplementing or replacing generic affix features with a focused set of morphological features produced by an FST, as introduced in §2.5. These targeted morphological features are effective during LP because words that share them are much more likely to actually share POS tags.

FSTs produce multiple analyses, which is actually advantageous for LP. Ambiguities need not be resolved since we just take the union of all morphological features for all analyses and use them as features in the graph. Note that each FST produces its own POS-tags as features, but these do *not* correspond to the target POS tagset used by the tagger. This is important because it decouples FST development and the final POS task. Thus, any FST for the language, regardless of its provenance,

[6]Wiktionary (`wiktionary.org`) has only 3,365 entries for Malagasy and 9 for Kinyarwanda.

can be used with any target POS tagset.

Extracting a result from LP

LP assigns a label distribution to every node. Importantly, each individual TOK gets its own distribution instead of having all tokens of the same word type share a single tag distribution. From this graph, we extract a new version of the raw corpus that contains a distribution over tags for each token. This provides the input for model minimization.

We seek a small set of likely tags for each token, but LP gives each token a distribution over the entire set of tags. Most of the tags are simply noise, some of which we remove by normalizing the weights and excluding tags with probability less than some threshold (we use 0.1). After applying this cutoff, the weights of the remaining tags are re-normalized. We stress that this tag dictionary cutoff is not like those used in past research by Merialdo (1994) and others, which were done with respect to frequencies obtained from labeled tokens: we use either no word-tag frequency information (type-supervision) or very small amounts of word-tag frequency information indirectly through LP (token-supervision). See Banko and Moore (2004) for further discussion of these issues.

Some tokens might not have any associated tag labels after LP. This occurs when there is no path from a TOK node to any seeded nodes or when all tags for the TOK node have weights less than the threshold. Since we require a distribution for every token, we use a *default* distribution for such cases. Specifically, we use a novel unsupervised emission probability distribution that captures both the estimated *frequency* of a tag and its *openness* using only a small tag dictionary and unlabeled text. This exact procedure is explained in §3.2.4.

Finally, we ensure that tokens of a word in the original tag dictionary are only assigned tags from its entry. With this filter, LP does not add new tags to known words (without it, we found performance drops). If there is no overlap between the small tag dictionary's entry and the token's resulting distribution from LP (after thresholding), we fall back to the filtered and renormalized *default* distribution for that token's type.

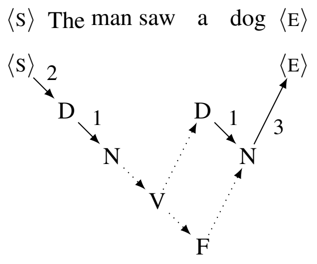
The result of this process is a sequence of (initially raw) tokens, each associated with a distribution over a subset of tags. From this we can extract an *expanded* tag dictionary for use in subsequent stages that, crucially, provides tag information for words not covered by the human-supplied tag dictionary. This expansion is simple: an word type is assigned the set of tags that is the union of all tags assigned to its tokens by the LP procedure. Additionally, we add the full entries of word types given in the original tag dictionary.

2.6.2 Removing annotation noise with model minimization

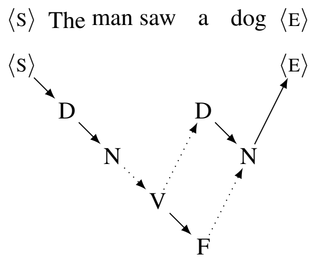
As was discussed above, one of the major problems for type-supervised POS-tagger training with EM is a tag dictionary with low-frequency entries such as the word "a" being associated with the *foreign word* tag when nearly all of its instances are tagged as *determiner*. To avoid the need for manually pruning the tag dictionary, Ravi and Knight (2009) proposed that low-probability tags might be automatically filtered from the tag dictionary through a model minimization procedure applied to the raw text and constrained by the full tag dictionary. The procedure works by finding a minimal set of tag bigrams needed to explain the sentences in the raw corpus. Ravi et al. (2010b) develop a faster approach for model minimization using a greedy algorithm that they call MIN-GREEDY. It is this algorithm that we extend.

Because the LP procedure is not very discriminating in the way that it pushes labels between nodes, the resulting automatically-induced annotations contain quite a bit of noise. Is it for this reason that we chose to apply model minimization. However, unlike the original model minimization work, our input comes from LP in the form of distributions over tags for every token in the corpus. Therefore, we have developed a novel variant of MIN-GREEDY that is specifically designed to make use of this *weighted* information to find a better minimized model. Our version is also innovative in its attempts to break ties in desirable ways. The model minimization procedure outputs a corpus of tagged sentences, which are used as a good starting point for EM training of an HMM.

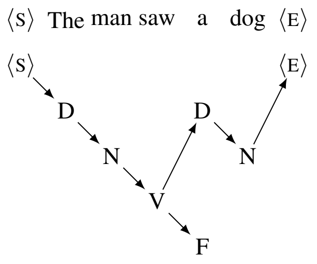
Model minimization is also a natural fit for our system since our low-resource



(a) MIN-GREEDY graph showing a state in the first phase. Numbered, solid arrows: order of chosen bigrams; dotted: potential choices.



(b) End of the first MIN-GREEDY phase; every word is touched by an edge.



(c) Potential MIN-GREEDY conclusion; gaps have been filled so that there is a complete path through the sentence.

Figure 2.3: Demonstration of the original MIN-GREEDY algorithm (Ravi et al., 2010b).

scenario means that we start with little or no frequency information. By finding a decent tagging of the corpus, model minimization helps us to be able to estimate that sequence and frequency information that we lack. This, in turn, allows us to estimate a much better starting point for EM than we would be able to with only the small set of provided annotations.

The original MIN-GREEDY algorithm

The MIN-GREEDY algorithm starts by initializing a graph with a vertex for each possible tag of each token in the raw data. The set of possible tags for each token is the set of tags associated with that word in the tag dictionary. Special *sentence boundary* vertices are added to the graph for each sentence to mark its beginning and end. The algorithm works in three phases: Greedy Set Cover, Greedy Path Completion, and Iterative Model-Fitting. In the first two phases, the algorithm chooses tag bigrams that form the edges of the graph. The goal of these phases is to select a set of edges that is sufficient to allow a path through every sentence in the raw corpus. The algorithm greedily selects these edges in an attempt to quickly approximate the minimal set of tag bigrams needed to accomplish this goal. In the final phase, the algorithm runs several iterations of EM in order to fit the bigram set to the raw data.

In the first phase, Greedy Set Cover, the algorithm selects tag bigrams in an effort to *cover* all of the word tokens. A word token is considered *covered* if there is at least one tag bigram edge connected to at least one of its vertices. At each iteration, the algorithm examines the entire graph, across all sentences, to find the tag bigram that, if added, would maximize the number of newly covered words.

Consider the graph in Figure 2.3a. Assume, for example, that this sentence comprises the entire raw corpus. At the start of the first phase, no tag bigrams are selected. On the first iteration, the algorithm chooses the tag bigram D$\rightarrow$N because this tag bigram describes two edges for a total of four words newly covered: *The*, *boy*, *a*, and *dog*. On the second iteration, there are only three word tokens left uncovered: the start, *saw*, and the end. At this point, as the figure shows, there are six tag bigrams that would each result in covering one additional token. Since there are no tag bigrams whose choosing would result in covering more than one additional token, the algorithm randomly chooses one of these six. The algorithm iterates like this until all words are covered, as in, for example, Figure 2.3b.

The second phase of the MIN-GREEDY algorithm, Greedy Path Completion, seeks to fill *holes* in the tag paths found in the graph. A *hole* is a potential edge that, if added, would connect two existing edges. At each iteration, the algorithm finds the tag bigram that, if selected, would maximize the number of holes that would be filled across all raw sentences.

The example graph in Figure 2.3b shows a potential start of the second phase. At this point, there are three tag bigrams that each fill one hole if selected, and the algorithm randomly selects one. Iteration continues until there is at least one complete tag path through each sentence in the raw corpus. One potential resolution for the example is given in Figure 2.3c.

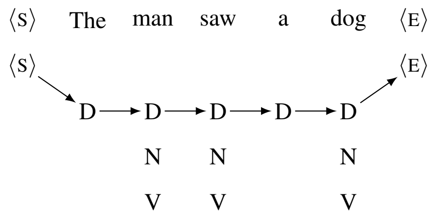
Once a set of tag bigrams has been generated that allows for a complete tag path through every sentence of the raw corpus, MIN-GREEDY begins its final phase: Iterative Model-Fitting. In this phase, the algorithm trains a succession of type-supervised HMM models. Each iteration trains an HMM using EM and then uses it to tag the raw corpus, the result of which is used to prepare inputs for the next iteration.

Problems with the original MIN-GREEDY algorithm

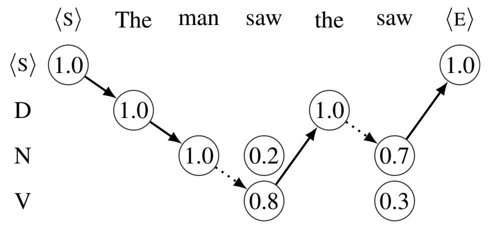
The MIN-GREEDY algorithm, as originally presented, has several problems both in terms of the algorithm design itself, and in terms of the experimental assumptions required for the algorithm to work properly.

The first issue is that the algorithm's heuristics for choosing the next tag bigram frequently result in many-way ties. In the first two phases of MIN-GREEDY, the greedy procedure looks for the tag bigram that will have the most positive impact. In the Greedy Set Cover phase this means choosing the tag bigram that would cover the most new tokens, and in the Greedy Path Completion phase this means choosing the tag bigram that would fill the most holes. However, it is frequently the case that there are many distinct tag bigrams that would cover the most new tokens or fill the most holes, leaving the MIN-GREEDY algorithm with no choice but to randomly select from these options. Since there are frequently cases of having many dozens of options, it is clear that some of those choices *must* be better than others, even though MIN-GREEDY does not make a distinction and considers them all to be equally good choices.

Consider the example in Figure 2.3a representing a possible state of the minimization graph. To have reached this stage, tag bigram D$\rightarrow$N would have been chosen since it covered the highest number of tokens: four. Additionally, $\langle$S$\rangle\rightarrow$D and N$\rightarrow\langle$E$\rangle$ could have been chosen as the second and third tag bigrams since they tied for the most new tokens covered: one. For the state shown in this figure, there is only one uncovered token, *sees*, but three tag bigrams that cover it. Since each of these tag bigrams covers exactly one new word, they are all considered by MIN-GREEDY to be equally good choices as the next tag bigram for inclusion, and the algorithm will choose one at random. However, it should be clear that the V$\rightarrow$F tag bigram is wrong while the other two would lead to a correct answer. As such, we would like for the algorithm to avoid choosing V$\rightarrow$F, and to pick one of the others.



(a) A degenerate optimal solution to MIN-GREEDY that requires only three tag bigrams. This occurs when words *man*, *saw*, and *dog* are are not found in the tag dictionary, meaning that they must be assumed to take any tag. Such degenerate bigram choices are highly likely as tag dictionaries become more incomplete.



(b) Weighted, greedy model minimization graph showing a potential state between the stages of the tag bigram choosing algorithm. Solid edges: selected bigrams. Dotted edges: holes in the path.

Figure 2.4: Demonstrations of model minimization behaviors.

Another, more serious, issue is that the MIN-GREEDY algorithm is very reliant on a fairly complete tag dictionary, meaning that it requires that most words appearing in the corpus be associated with a subset of tags. The reason for this is that if a token's word is not found in the dictionary, then we must assume that it can have any possible tag. If there are many such words, then the algorithm will end up choosing tag bigrams that cover large numbers of words but that are not discriminating enough about which tags are chosen. An example of this behavior can be seen in Figure 2.4a. In our low-resource scenario, where the tag dictionary provided by an annotator is extremely sparse, this aspect of MIN-GREEDY is likely to be extremely problematic.

Our novel *weighted* variant of MIN-GREEDY, presented below, addresses both of these drawbacks when employed in our semi-supervised training pipeline, as well as improving the general ability of model minimization to find a good tagging of the corpus.

Weighted model minimization

The output of the tag dictionary generalization procedure via LP is a corpus of sentences where each token is associated with a distribution over tags. Rather than discard this distribution information, reducing this data to a simple (expanded) tag dictionary to use the MIN-GREEDY as it was originally presented, we developed a novel extension of the algorithm that makes explicit use of this weighted information. As shown in Figure 2.4b, each node in the graph is assigned a weight corresponding to the probability assigned to that label on that token by the LP procedure. Our general strategy is to find a minimal model that is biased toward keeping tag bigrams that have consistently high weights across the entire corpus.

The original MIN-GREEDY algorithm selects tag bigrams that cover the most new words (stage 1) or fill the most holes in the tag paths (stage 2). Our version modifies the criteria to use the tag weights on each token: a tag bigram b is chosen by summing up the node weights of any not-yet covered words touched by the tag bigram and choosing the bigram that maximizes this value.[7] Summing node weights captures the intuition of Ravi et al. (2010b) that good bigrams are those which have high coverage of new words: each newly covered node contributes additional (partial) counts. However, by using the weights instead of full counts, we also account for the confidence assigned by LP. Considering only the weights of not-yet-included tokens keeps the algorithm focused on bigrams pushing toward full coverage of the corpus.

At the start of model minimization, there are no selected tag bigrams, and thus no valid path through any sentence in the corpus. As bigrams are selected, we can begin to cover sub-sequences and eventually full sentences. There may be multiple valid taggings for a sentence, so after each new bigram is selected, we run the Viterbi algorithm over the raw corpus using the set of selected tag bigrams as a hard constraint on the allowable transitions. This efficiently identifies the highest-weight path through each sentence, if one exists. If such a path is found, we remove

[7] In the case of token-supervision, we simply include the sentences and declare that each token's distribution is defined such that the given tag receives probability 1.

the sentence from the corpus and store the tags from the Viterbi tagging. The algorithm terminates when a path is found for every raw corpus sentence. The result of weighted model minimization is this set of tag paths. Since each path represents a valid tagging of the sentence, we use this output as a noisily labeled corpus for initializing EM in the next stage.

Removing sentences from consideration as soon as a tag path though the sentence is found is a novel contribution of our approach that has several advantages over the original MIN-GREEDY formulation. First, if all sentences were left in place, then as more bigrams are selected, we could end up with multiple paths through sentences. However, by storing the sentence's tagging as soon as the first tagging is found, we are sure to be storing the tags that our greedy algorithm determined to be best. Second, by removing the sentences, we no longer include their count statistics when trying to choose bigrams. This is useful because if already-completed sentences are driving the selection, then we are likely to end up picking tags that are less useful for helping us to tag the remaining sentences, meaning that we will likely end up choosing more tag bigrams than we want. Finally, removing sentences has the side effect of making the bigram-choosing iterations faster because the algorithm has less data that it must consider for each decision. The end result of this improvement to the algorithm is that we find cleaner minimized tag bigram sets and cleaner corpus taggings.

The use of weights also has the advantage that it virtually eliminates the possibility of ties among bigram choices because it is very unlikely that two bigrams would have exactly the same real-valued scores. This addresses one of the issues with the original MIN-GREEDY algorithm without needing to resort to the tie-breaking strategy introduced by Garrette and Baldridge (2012) which complicates the bigram-choice procedure by trying to find the bigram that introduces the smallest possible number of new *word/tag pairs* into the paths.

2.7 Tagger Training

The approach just described for generalizing the initial annotations produces two outputs: a large tag dictionary that contains an entry for every word in the raw data, and a noisy tagging of every sentence in that raw data. With these two sources of information, we can run EM to estimate good parameters for an HMM. To initialize EM, we simply calculate the smoothed MLE based on the noisy tagging that came from model minimization. We use this as the starting set of parameters, and run EM to iteratively estimate new parameters, constraining the tag choices with the generalized tag dictionary. If human-tagged sentences are available as training data, then we use their counts to supplement the noisy labeled text for initialization and we add their counts into every iteration's result.

To obtain a still-better tagger, we take the additional step of using the EM-trained HMM to find the Viterbi tagging of the entire raw corpus and then using that "auto-supervised" corpus, concatenated with any token-supervised corpus sentences that might be available, to train a Maximum Entropy Markov Model (MEMM) tagger.[8] The MEMM exploits subword features and generally produces 1-2% better results than an HMM trained on the same material. While training an HMM before the MEMM is not strictly necessary, our tests have shown that this generative-then-discriminative combination generally results in around 3% accuracy improvement.

A tagger trained using our semi-supervised pipeline performs better than one trained simply on the provided annotations for two major reasons. The first is that we use our procedure to estimate a good initialization for EM. If EM were run directly on the annotated data, it would likely lead to a poor local optimum, far from what the corpus actually needs. The second reason is that by inducing a large tag dictionary that contains an entry for every raw corpus word, we dramatically constrain the search space of EM because it no longer has to assume that those unknown words can choose from any of the 45 possible tags. This constraint helps keep EM on track in learning a good HMM.

[8]We use OpenNLP: `opennlp.apache.org`.

	0. No EM			1. EM only			2. With LP			3. LP+min		
Initial data	A	K	U	A	K	U	A	K	U	A	K	U
KIN tokens	**72**	90	58	55	82	32	71	86	58	71	86	58
KIN types				63	77	32	78	83	69	**79**	83	70
MLG tokens	74	89	49	68	87	39	**74**	89	49	**74**	89	49
MLG types				71	87	46	72	81	57	**74**	86	56
ENG tokens A	63	83	38	62	83	37	**72**	85	55	**72**	85	55
ENG types A				66	76	37	75	81	56	**76**	83	56
ENG tokens B	70	87	44	70	87	43	**78**	90	60	**78**	90	60
ENG types B				69	83	38	75	82	61	**78**	85	61

Table 2.6: Results from the NAACL two-hour experiments. Three languages are shown: Kinyarwanda (KIN), Malagasy (MLG), and English (ENG). The letters A and B refer to the annotator. Five experiments were executed: (0) supervised training directly from the human-provided labeled sentences without EM; (1) EM without LP or model minimization; (2) EM with LP, but not model minimization; (3) EM with LP and our weighted model minimization procedure; and (4) EM with LP that includes nodes from an external dictionary and weighted model minimization. Each result given as percentage of tokens accurately labeled for All word types (A), Known word types (K), which appear in the original tag dictionary, and Unknown word types (U), which do not.

2.8 Experiments[9]

To better understand the effect that each type of supervision has on tagger accuracy, we perform a series of experiments, with KIN and MLG as true low-resource languages. English experiments, for which we had multiple annotators, allow for further exploration into issues concerning data collection and preparation.[10]

During the course of this research, we ran two different batches of experiments. In the first batch, we gave annotators two hours to annotate types and two hours to annotate tokens. We did not provide them with much guidance beyond the basic parameters of the task, which led to a relatively high degree of annotator error.

[9]Code available at `github.com/dhgarrette/low-resource-pos-tagging-2013`

[10]We are unable to provide the KIN or ENG data for download due to licensing restrictions. However, ENG data may be shared with those holding a license for the Penn Treebank and KIN data may be shared on a case-by-case basis.

The results this batch of experiments is discussed in §2.8.1 and in the error analysis of §2.8.8.

Based on our findings in the first experiments, we ran new experiments by collecting new data under different guidelines. For this second batch of experiments, we gave our annotators four hours to complete each task, and asked them to indicate 30-minute breaks so that we could analyze learning curves. For English, we again had two annotators, though this time we specifically chose an experienced annotator who had performed the task for the previous experiments, and a novice who had not performed the task before. We also gave the annotators stronger guidance, which yielded cleaner annotations. This data allows us to provide the more in-depth analyses below.

The overall best accuracies achieved by language are 81.9% for KIN using four hours of type annotations, 81.2% for MLG using two hours each of type and token annotations, and 89.8% for ENG using all types and the maximal amount of raw data. All of these best values were achieved using both FST and affix LP features.

All results described in this section are averaged over five folds of raw data.

2.8.1 Evaluating the pipeline

Experimental results are shown in Table 2.6 that evaluate at each stage of our semi-supervised pipeline. Each experiment starts with an initial data set provided by an annotator. Experiment (1) uses EM with the initial small tag dictionary to learn a tagger from the raw corpus. (2) uses LP to infer an expanded tag dictionary and tag distributions over raw corpus tokens, but then takes the highest-weighted tag from each token for use as noisily-labeled training data to initialize EM. (3) performs weighted model-minimization on the LP output to derive that noisily-labeled corpus. Finally, (4) is the same as (3), but additionally uses external dictionary nodes in the LP graph.

The results show that performance improves with our LP and minimization techniques compared to basic EM-HMM training. LP gives large across-the-board improvements over EM training with only the original tag dictionary (compare columns 1 & 2). Weighted model minimization further improves results for type-supervision settings, but not for token supervision (compare 2 & 3).

Using an external dictionary in the LP graph has little effect for KIN, probably due to the available dictionary's very small size. However, MLG with its larger dictionary obtains an improvement in both scenarios. Results on ENG are mixed; this may be because the PTB tagset has 45 tags (far more than the dictionary) so the external dictionary nodes in the LP graph may consequently serve to collapse distinctions (e.g. singular and plural) in the larger set.

Our results show differences between token- and type-supervised annotations. Tag dictionary expansion is helpful no matter what the annotations look like: in both cases, the initial dictionary is too small for effective EM learning, so expansion is necessary. However, model minimization only benefits the type-supervised scenarios, leaving token-supervised performance unchanged. This suggests that minimization is working as intended: it induces frequency information when none is provided. With token-supervision, the annotator provides some information about which tag transitions are best and which emissions are most likely. This is missing with type-supervision, so model minimization is needed to bootstrap word/tag frequency guesses.

This leads to perhaps our most interesting result: in a time-critical annotation scenario, it seems better to collect a simple tag dictionary than tagged sentences. While the tagged sentences certainly contain useful information regarding tag frequencies, our techniques can learn this missing information automatically. Thus, having wider coverage of word type information, and having that information be focused on the most frequent words, is more important.

Our experiments also allow us to compare how the data from different annotators affects the quality of taggers learned. Looking at the direct comparison on English data, annotator B was able to tag more sentences than A, but A produced more tag dictionary entries in the type-supervision scenario. However, it appears, based on the EM-only training, that the annotations provided by B were of higher quality and produced more accurate taggers in both scenarios. Regardless, our full training procedure is able to substantially improve results in all scenarios.

Tag Dictionary Source	R	P
(1) human-annotated TD	18.42	29.33
(2) LP output	35.55	2.62
(3) model min output	30.49	4.63

Table 2.7: Recall (R) and precision (P) for tag dictionaries versus the test data in a "MLG types" run.

Table 2.7 gives the recall and precision of the tag dictionaries for MLG for settings 1, 2 and 3. The initial, human-provided tag dictionary unsurprisingly has the highest precision and lowest recall. LP expands that data to greatly improve recall with a large drop in precision. Minimization culls many entries and improves precision with a small relative loss in recall. Of course, this is only a rough indicator of the quality of the tag dictionaries since the word/tag pairs of the test set only partially overlap with the raw training data and annotations.

Because gold-standard annotations are available for the English sentences, we also ran *oracle* experiments using labels from the PTB corpus (essentially, the kind of data used in previous work). We selected the same amount of labeled tokens or word/tag pairs as were obtained by the annotators. We found similar patterns of improved performance by using LP expansion and model minimization, and all accuracies are improved compared to their human-annotator equivalents (about 2-6%). Overall accuracy for both type and token supervision comes to 78-80%.

2.8.2 Weighted vs. unweighted minimization

To determine whether that *weighted* model minimization is beneficial, we ran experiments using the *unweighted* strategy. On average, the weighted version yields an increase of 0.76%. However, the average increase is 1.27% for type-supervision, so the weighted version is particularly useful when the human annotation is a tag dictionary without frequency information.

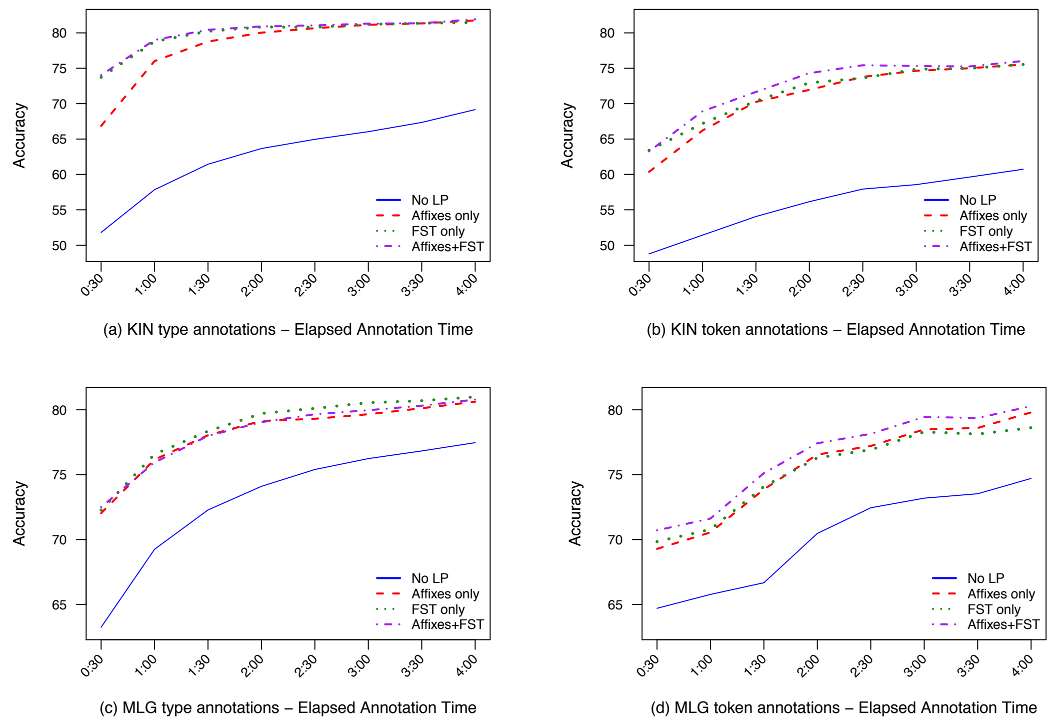
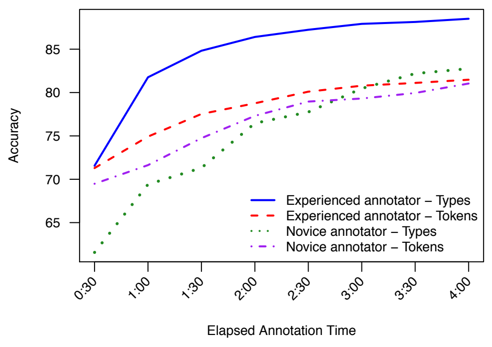


Figure 2.5: Annotation time vs. tagger accuracy for type-only and token-only annotations.

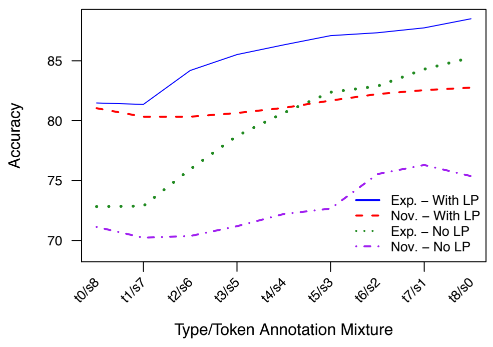
2.8.3 Types versus tokens

Our primary question was the relationship between annotation type and time. Annotation must be done by someone familiar with the target language, linguistics, and the target POS tagset. For many low-resource languages, such people, and the time they have to spend, are likely to be in short supply. To make the best use of their time, we need to know which annotations are most useful so that efforts can be concentrated there. Additionally, it is useful to identify when returns on annotation effort diminish so that annotators do not spend time doing work that is unlikely to add much value.

The annotators produced four hours each of type and token annotations, each in 30-minute increments. To assess the effects of annotation time, we trained taggers cumulatively on each increment and determine the value of each additional



(a) Annotation time vs. tagger accuracy for ENG type-only and token-only annotations with affix and FST LP features.



(b) Annotation mixture vs. tagger accuracy on ENG using affix and FST LP features for experienced (Exp.) and novice (Nov.) annotators.

Figure 2.6: English experimental results. In (b), x-axis labels give annotation proportions, e.g. "t2/s6" indicates 2/8 of the time (1 hour) was spent annotating types and 6/8 (3 hours), full sentences.

half-hour of effort. Results are shown for KIN and MLG in Figure 2.5 and ENG in Figure 2.6a. In all scenarios, the use of LP (with model minimization) delivers huge performance gains. Additionally, the use of FST features, usually along with affixes, yielded better results than without. This indicates the LP procedure makes effective use of the morphological features produced by the FST and that the affix features are able to capture missing information without adding too much noise to the LP graph.

Furthermore, performance is considerably better when type annotations are used than with only tokens. Type annotations plateau much faster, so less time is required for annotating if types are used rather than token annotations. For KIN it takes approximately 1.5 hours to reach near-maximum accuracy for types, but 2.5 hours for tokens. This difference is due to the fact that the type annotations started with the most frequent words whereas the token annotations were on random sentences. Thus, type annotations quickly cover a significant portion of the language's tokens. With annotations directly on tokens, some of the highest frequency types are covered, but annotation time is also ineffectively used on low-frequency types that happen to appear in those sentences.

Finally, the use of FST features yields larger gains for KIN than other languages, but only when small amounts of annotation are available. This makes sense: KIN is a morphologically rich language, so sparsity is greater and crude affixes capture less actual morphology. With little annotated data, LP relies heavily on morphological features to make clean links between words. But, with more annotations, the gains of the FST over affix features alone diminish: the affix features eventually capture enough of the morphology to make up the difference.

Figure 2.6a shows the dramatic differences between the experienced and novice ENG annotators.[11] For the former, results using types and tokens were similar after 30 minutes, but type annotations proved much more useful beyond that. In contrast, the novice annotated types much more slowly, so early on there were not enough annotated types for the training to be as effective. Even so, after three hours of annotation, type annotations still win with the novice, and even beat the experienced annotator labeling tokens.

2.8.4 Mixing type and token annotations

Because type and token annotations are each better at providing different information — a tag dictionary of high-frequency words vs. sequence and frequency information — it is reasonable to expect that a combination of the two might yield higher performance by each contributing different but complementary information during training. This matters in low-resource settings because type or token annotations will likely be produced by the same people, so there is a tradeoff between spending resources on one form of annotation over the other. Understanding the best mixture of annotations can inform us on how to maximize the benefit of a set annotation budget. To this end, we ran experiments fixing the annotation time to four hours while varying the mix of type and token annotations. Results are shown for KIN and MLG in Figure 2.7 and ENG in Figure 2.6b.

[11]The ENG graph omits "No LP" results since they followed patterns similar to KIN and MLG. Additionally, the results without FST features are not shown because they were nearly identical (though slightly lower) than with the FST.

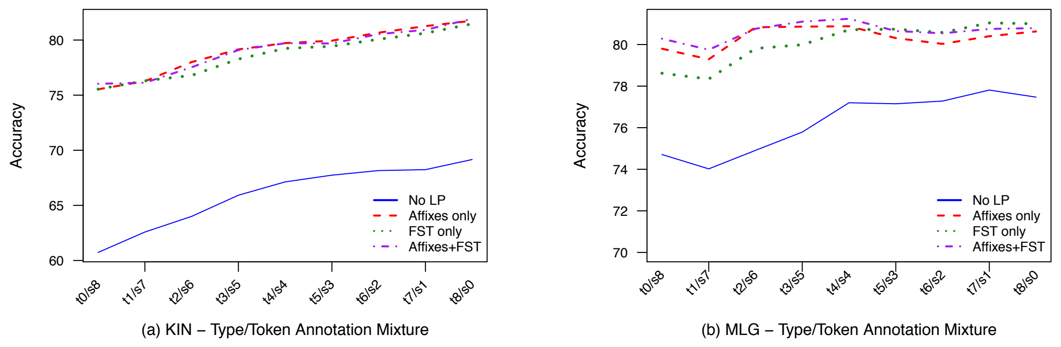


Figure 2.7: Annotation mixture vs. tagger accuracy. X-axis labels give annotation proportions, e.g. "t2/s6" indicates 2/8 of the time (1 hour) was spent annotating types and 6/8 (3 hours), full sentences.

Types clearly win for ENG. The experienced annotator was much faster at annotating types and the speed difference was less pronounced for tokens, so accuracy is most similar when only token annotations are used. The performance disparity grows with increasing the type proportion.

Täckström et al. (2013) explore the use of mixed type and token annotations in which a tagger is learned by projecting information via parallel text. In their experiments, they—like us—found that type information is more valuable than token information. However, they were able to see gains from the complementary effects of mixing type and token annotations. It is likely that this difference in our results is due to the amount of annotated data used. It seems that the amount of type information collected in four hours is not sufficient to saturate the system, meaning that switching to annotating tokens tends to hurt performance.

2.8.5 FST development

The third set of experiments evaluate how the amount of time spent developing an FST affects the performance of trained tagger. To do this, we had our ENG FST developer save progress after each hour (for ten hours). The results show that, for ENG, the FST provided no value, regardless of how much time was spent on its development. Moreover, since large gains in accuracy can be achieved by spending a small amount of time just annotating word types with POS tags, we are led to

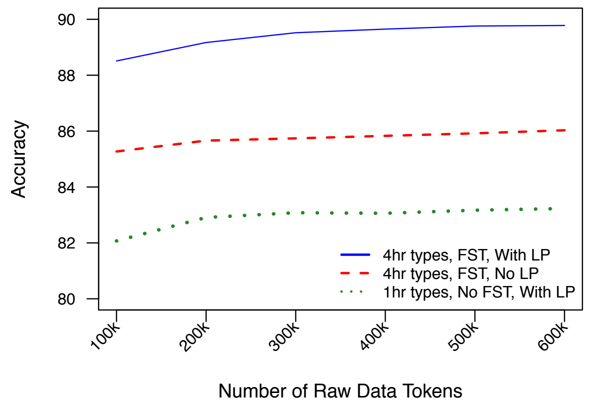


Figure 2.8: Amount of raw data vs. tagger accuracy for ENG using high vs. low amounts of annotation and using LP vs. no LP., for experienced annotator (novice results were similar).

conclude that time should be spent annotating types or tokens instead of developing an FST. While it is likely that FST development time would have a greater impact for morphologically rich languages, we suspect that greater gains can still be obtained by instead annotating types. Nonetheless, using FSTs never seems to *hurt* performance, so if one is readily available, it should be used.[12]

2.8.6 The effect of more raw data

In addition to annotations, semi-supervised tagger training requires a corpus of raw text. Raw data can be easier to acquire since it does not need the attention of a linguist. Even so, for many low-resource languages, the amount of digitized text, such as transcripts or websites, is very limited and may, in fact, require substantial effort to accumulate, even with assistance from computational tools (Bird, 2011). Therefore, the collection of raw data can be considered another time-sensitive task for which the tradeoffs with previously-discussed annotation efforts must contend.

It could be the case that more raw data for training could make up for additional annotation and FST development effort or make the LP procedure unneces-

[12]The KIN and MLG experts were no longer available to re-develop FSTs so that time increments could be measured.

sary. Figure 2.8 shows that that increased raw data does provide increasing gains, but they diminish after 200k tokens. The best performance is achieved by using more annotation and LP. Most importantly, however, removing either annotations or LP results in a significant decline in accuracy, such that even with 600k training tokens, we are unable to achieve the results of high annotation and LP using only 100k tokens.

2.8.7 Correcting existing annotations

For all of the ENG experiments, we also ran "oracle" experiments using gold tags for the same sentences or a tag dictionary containing the same number of type/tag entries as the annotator produced, but containing only the most frequent entries as determined by the gold-labeled corpus. Using this simulated "perfect annotator" data shows we lose accuracy due to annotator mistakes: for our experienced annotator and maximal FST, using 4 hours of types, the oracle accuracy is 90.5 vs. 88.5, while using only tokens we see 83.9 vs. 81.5. This indicates that there are gains to be made by correcting mistakes in the annotations. This is true even after the point of diminishing returns on the learning curve, meaning that even when adding *more* annotations no longer improves performance, progress can still be made by correcting errors, so it may be reasonable to ask annotators to attempt to correct errors in their past annotations. Automated techniques for facilitating error identification can be employed for this (Dickinson and Meurers, 2003).

2.8.8 Error analysis

One potential source of errors comes from the annotators. Though our approach is designed to be robust to annotation errors, it cannot correct all mistakes. In the first round of experiments (two-hour annotation time), we gave less guidance to the annotators, which resulted in more annotation errors. For example, for the "ENG types B" experiment, the annotator listed IN (preposition) as the only tag for word type "to". However, the PTB test set only ever assigns tag TO for this word. This single error accounts for a 2.3% loss in overall tagging accuracy (Table 2.8).

#Errors	Gold	Model
11k	TO	IN
6k	NNP	NN
5k	NN	JJ
4k	JJ	NN
3k	NNP	JJ

Table 2.8: Top errors from an "ENG types B" run.

for	*IN	*RP	JJ	NN	CD
(1) EM	1,221	2764		9	5
(2) LP	4,003				
(3) min	4,004		1		
gold	3,999	5			

, (comma)	*,	*:	JJS	PTD	VBP
(1) EM	24,708		4	3	3
(2) LP	15,505	9226			1
(3) min	24,730				
gold	24,732				

opposition	NN	JJ	DT	NNS	VBP
(1) EM	24	4	1	4	4
(2) LP	41	4			
(3) min	45				
gold	45				

Table 2.9: Tag assignments in different scenarios for three representitive words. Star (*) indicates an entry in the human-provided TD.

In many situations, however, we are able to automatically remove improbable tag dictionary entries, as shown in Table 2.9. Consider the word type "for". The annotator has listed RP (particle) as a potential tag, but only five out of 4k tokens have this tag. With RP included, EM becomes confused and labels a majority of the tokens as RP when nearly all should be labeled IN. We are able to eliminate RP as a possibility, giving excellent overall accuracy for the type. Likewise for the comma type, the annotator has incorrectly given ":" as a valid tag, and LP, which uses the tag dictionary, pushes this label to many tokens with high confidence. However, minimization is able to correct the problem.

Finally, the word type "opposition" provides an example of the expected behavior for unknown words. The type is not in the tag dictionary, so EM assumes all tags are valid and uses many labels. LP expands the starting dictionary to cover the type, limiting it to only two tags. Minimization then determines that NN is the best tag for each token.

2.9 Related Work

As previously discussed, many researchers have explored a variety of approaches for weakly-supervised POS-tagging. Some previous work has focused on learning from existing linguistic resources. Cucerzan and Yarowsky (2002) learn a POS-tagger from a dictionary and a reference grammar while Li et al. (2012) train an HMM using EM using a dictionary extracted directly from Wiktionary. Goldberg et al. (2008) trained a tagger for Hebrew using a high-coverage, high-quality lexicon manually created by trained lexicographers.

Other work has focused on the task of expanding small resources into larger ones. Haghighi and Klein (2006) develop a model in which a POS-tagger is learned from a list of POS tags and just three "prototype" word types for each tag, using a vector space to compute the distributional similarity between prototypes and other word types in the corpus. Toutanova and Johnson (2008) use a simple method for predicting possible tags for unknown words: a set of 100 most common suffixes are extracted and then models of $P(\textit{tag} \mid \textit{suffix})$ are built and applied to unknown words. Ding (2011) constructed an LP graph for learning POS tags on Chinese text by propagating labels from an initial tag dictionary to a larger set of data. This LP graph contained Wiktionary word/POS relationships as features as well as Chinese-English word alignment information and used it to directly estimate emission probabilities to initialize an EM training of an HMM.

Finally, there has been more recent work on adapting tag information for use in weakly-supervised scenarios. Subramanya et al. (2010) apply LP to the problem of tagging for domain adaptation, constructing a graph that connects tokens in low- and high-resource domains, and propagate labels from high to low. Das and Petrov (2011) learn taggers for languages in which there are no POS-annotated resources, but for which parallel texts are available between that language and a high-resource language. Täckström et al. (2013) further evaluate the use of mixed type and token constraints generated by projecting information from a high-resource language to a low-resource language via this parallel corpus.

Our results compare favorably with previous work despite using considerably less supervision and a more difficult set of tags. For example, Li et al. (2012) use the entirety of English Wiktionary directly as a tag dictionary to obtain 87.1% accuracy on English, below our result. Täckström et al. (2013) average 88.8% across 8 major languages, but for Turkish, a morphologically rich language, they achieve only 65.2%, significantly below our 81.9% for morphologically-rich Kinyarwanda.

2.10 Conclusions and Future Work

Care must be taken when drawing conclusions from small-scale annotation studies such as those presented in this chapter. Nonetheless, we have explored realistic annotation scenarios for POS-tagging for low-resource languages and found several consistent patterns. Most importantly, it is clear that type annotations are the most useful input one can obtain from a linguist in a low-resource scenario — provided a semi-supervised algorithm for projecting that information reliably onto raw tokens is available. In a sense, this result validates the research trajectory of efforts over the past two decades put into learning taggers from tag dictionaries: type-supervised learning is indeed a valuable tool, as we have been able to show in experiments that approximate realistic low-resource conditions.

The result of most immediate practical value is that we show it is possible to train effective POS-taggers on actual low-resource languages given only a relatively small amount of unlabeled text and a few hours of annotation by a non-native linguist. Instead of having annotators label full sentences as one might expect the natural choice would be, it is much more effective to simply extract a list of the most frequent word types in the language and concentrate efforts on annotating these types with their potential parts of speech. We note, however, that techniques such as *active learning* may be able to choose better sentences or phrases, boosting the utility of token supervision. Furthermore, for languages with rich morphology, a morphological transducer can yield significant performance gains when large amounts of other annotated resources are unavailable. (And it never hurts performance.)

Finally, additional raw text does improve performance. However, using substantial amounts of raw text is unlikely to produce gains larger than only a few hours spent annotating types. Thus, when deciding whether to spend time locating volumes of digitized text or to spend time annotating types, choose types.

Future work might dig deeper into what kinds of annotations should be used. Petrov et al. (2012) proposed an alternative set of "universal" POS tags that greatly simplify the distinctions found in the PTB data. For example, it has just one verb tag instead of the six types of verbs in the PTB set. Having annotators label data with this coarse tagset would dramatically increase the speed with which they could label data since they would no long need to spend time trying to decide which types of, e.g., verb a particular word type represents. However, as Petrov et al. note, this coarse data comes at the expense of input detail, providing less specific information than the fine-grained set, though we suspect that the increased volume of annotation achievable due to the decreased effort would likely have a much higher impact.

Chapter 3
Grammar-Informed CCG Supertagging

Supertagging, broadly defined, is the task of assigning complex categories, known as *supertags*, to each lexical item in a text (Bangalore and Joshi, 1999). These supertags are used to localize information about linguistic structure and provide constraints on local contexts. Supertagging approaches have been applied to Lexicalized Tree-Adjoining Grammar (Joshi and Srinivas, 1994) and Combinatory Categorial Grammar (Clark, 2002). The goal of supertagging is to encode enough information at the lexical level that reassembling the full structure — generally a parse tree — can be done with relative ease. In this sense, the task of supertagging can be thought of as "almost parsing". By reducing the parsing problem to one of supertagging, we are turning a complex structure-prediction problem into one that can be undertaken with simpler models that focus on local decisions, even as these local decisions have goal ramifications. This is particularly desirable to us because we are interested in learning models of language using only weak forms of supervision. With limited input, simpler models are appealing.

We approach supertagging as a sequence modeling task in the vein of weakly-supervised part-of-speech (POS) induction, a much more widely studied problem (see Chapter 2). Many proposed solutions to POS-tagger learning are based on Hidden Markov models (HMMs), with various improvements obtainable through: inductive bias in the form of tag dictionaries (Kupiec, 1992; Merialdo, 1994), sparsity constraints (Lee et al., 2010), careful initialization of parameters (Goldberg et al., 2008), feature based representations (Berg-Kirkpatrick et al., 2010; Smith and Eisner, 2005), and priors on model parameters (Johnson, 2007; Goldwater and Griffiths, 2007; Blunsom and Cohn, 2011, *inter alia*).

The contents of this chapter are based on the following publication: Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. 2014. Weakly-supervised Bayesian learning of a CCG supertagger. In Proc. of CoNLL. My co-authors worked with me to develop the ideas underlying this work.

When tag dictionaries are available, a situation we called *type-supervision* in Chapter 2, POS induction from unlabeled corpora can be relatively successful; however, as the number of possible tags increases, performance drops (Ravi and Knight, 2009). In such cases, there are a large number of possible labels for each token, so picking the right one simply by chance is unlikely; the parameter space tends to be large, and devising good initial parameters is difficult. Therefore, it is unsurprising that the weakly-supervised learning of a supertagger for Combinatory Categorial Grammar (CCG), for which there is a very large (possibly unbounded) number of structured categories, is a considerable challenge.

Despite the apparent complexity of the task, supertag sequences have regularities due to universal properties of the CCG formalism (§3.1) that can be used to reduce the complexity of the problem (Garrette et al., 2014). Previous work showed promising results by using these regularities to initialize an HMM that is then refined with EM (Baldridge, 2008). Here, we exploit CCG's category structure to motivate a novel prior over HMM parameters for use in Bayesian learning (§3.2). This prior encourages (i) cross-linguistically common tag types, (ii) tag bigrams that can combine using CCG's combinators, and (iii) sparse transition distributions. We also go beyond the use of these universals to show how additional, *corpus-specific* information can be automatically extracted from a combination of the tag dictionary and raw data, and how that information can be combined with the universal knowledge for integration into the model to improve the prior.

We use a blocked sampling algorithm to sample supertag sequences for the sentences in the training data, proportional to their posterior probability (§3.3). We experimentally verify that our Bayesian formulation is effective and substantially outperforms the state-of-the-art baseline initialization/EM strategy in several languages (§3.4).

3.1 CCG and Supertagging

CCG (Steedman, 2000; Steedman and Baldridge, 2011) is a grammar formalism in which each lexical token is associated with a structured category, often referred to as a *supertag*. CCG categories are defined by the following recursive definition:

$$C \rightarrow \{\mathsf{s}, \mathsf{s}_{\mathsf{dcl}}, \mathsf{s}_{\mathsf{adj}}, \mathsf{s}_{\mathsf{b}}, \mathsf{n}, \mathsf{np}, \mathsf{np}_{\mathsf{nb}}, \mathsf{pp}, ...\}$$
$$C \rightarrow \{(C/C), (C\backslash C)\}$$

A CCG category can either be an *atomic* category indicating a particular type of basic grammatical phrase (s for a sentence, n for a noun, np for a noun phrase, etc), or a *complex* category formed from the combination of two categories — possibly complex themselves — by one of two *slash* operators. In CCG, complex categories indicate a grammatical relationship between the two operands. For example, the category $(\mathsf{s}\backslash\mathsf{np})/\mathsf{np}$ might describe a transitive verb, looking first to its right (indicated by $/$) for an object, then to its left ($\backslash$) for a subject, to produce a sentence. Further, atomic categories may be augmented with *features*, such as $\mathsf{s}_{\mathsf{dcl}}$, to restrict the set of atoms with which they may unify. The task of assigning a category to each word in a text is called *supertagging* (Bangalore and Joshi, 1999).

Because they are recursively defined, there are an infinite number of potential CCG categories (though in practice it is limited by the number of actual grammatical contexts). As a result, the number of supertags appearing in a corpus far exceeds the number of POS tags (see Table 3.2). Since supertags specify the grammatical context of a token, and high frequency words appear in many contexts, CCG grammars tend to have very high lexical ambiguity, with frequent word types associating with a large number of categories. This ambiguity has made type-supervised supertagger learning very difficult because the typical approaches to initializing parameters for EM become much less effective (Baldridge, 2008; Ravi et al., 2010a).

3.1.1 Grammar-informed supertagger learning

Baldridge (2008) was successful in extending the standard type-supervised tagger learning to the task of CCG supertagging by setting the initial parameters for EM training of an HMM using two intrinsic properties of the CCG formalism: the tendency for adjacent tags to combine, and the tendency to use less complex tags.

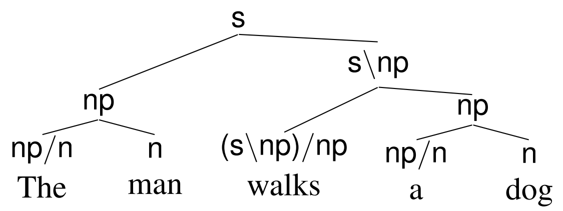


Figure 3.1: CCG parse for "The man walks a dog"

Forward Application:	X/Y	Y	$\Rightarrow$	X	$(>)$
Backward Application:	Y	$X\backslash Y$	$\Rightarrow$	X	$(<)$
Forward Harmonic Comp.:	X/Y	Y/Z	$\Rightarrow$	X/Z	$(>\mathbf{B})$
Forward Harmonic Comp. 2:	X/Y	$(Y/Z)\vert_i W$	$\Rightarrow$	$(X/Z)\vert_i W$	$(>\mathbf{B}^2)$
Backward Harmonic Comp.:	$Y\backslash Z$	$X\backslash Y$	$\Rightarrow$	$X\backslash Z$	$(<\mathbf{B})$
Backward Crossed Comp.:	Y/Z	$X\backslash Y$	$\Rightarrow$	X/Z	$(<\mathbf{B}_\times)$
Backward Crossed Comp. 2	$(Y/Z)\vert_i W$	$X\backslash Y$	$\Rightarrow$	$(X/Z)\vert_i W$	$(<\mathbf{B}^2_\times)$

Table 3.1: Combination rules used by our supertagger. In generalized composition rules, $|_i$ may be either / or \, but co-indexed operators must be the same. Backward composition rules are blocked where Y is n or np.

These properties are explained in detail in the original work, but we restate the ideas briefly throughout this chapter for completeness.

Tag combinability

A CCG parse of a sentence is derived by recursively combining the categories of sub-phrases. Category combination is performed using only a small set of generic rules (see Table 3.1). In the tree in Figure 3.1, we can see that *a* and *dog* can combine via Forward Application (>), with np/n and n combining to produce np. The associativity engendered by CCG's composition rules means that most adjacent lexical categories may be combined. In the Figure 3.1 tree, we can see that instead of combining (walks·(a·dog)), we could have combined ((walks·a)·dog) since (s\np)/np and np/n can combine using $>\mathbf{B}$.

Note, however, that this does not mean that two adjacent tags are *required* to be combinable. For any bigram of tags A·B, it is always possible that some constituent C after the bigram may combine with B without involving A: A·(B·C). Likewise with constituents to the left of A. Thus, any bias toward supertag combinability we implement must be *soft*, allowing for non-combinations as well.

3.2 Model

In this section we define the generative process of the Bayesian HMM (Goldwater and Griffiths, 2007) that we use to model a corpus of sentences. We begin by generating the model parameters: for each supertag type $\mathbf{t}$ in the tag set $\mathcal{T}$, the transition probabilities to the next state ($\pi_{\mathbf{t}}$) and the emission probabilities ($\phi_{\mathbf{t}}$) are generated by draws from Dirichlet distributions parameterized with per-tag mean distributions ($\pi^0_{\mathbf{t}}$ and $\phi^0_{\mathbf{t}}$, respectively) and concentration parameters (α_π and α_ϕ). By setting α_π close to zero, we can encode our prior expectation that transition distributions should be relatively peaked (i.e., that each tag type should be followed by relatively few tag types). The prior means, discussed below, encode both linguistic intuitions about expected tag-tag transition behavior and automatically-extracted corpus information. Given these parameters, we next generate the sentences of the corpus. This process is summarized as follows:

Parameters:

$$\phi_{\mathbf{t}} \sim \text{Dirichlet}(\alpha_\phi, \phi^0_{\mathbf{t}}) \quad \forall \mathbf{t} \in \mathcal{T}$$

$$\pi_{\mathbf{t}} \sim \text{Dirichlet}(\alpha_\pi, \pi^0_{\mathbf{t}}) \quad \forall \mathbf{t} \in \mathcal{T}$$

Sentence:

$$y_1 \sim \text{Categorical}(\pi_{\langle \text{S} \rangle})$$

$$\text{for } i \in \{1, 2, \ldots\}, \text{until } y_i = \langle \text{E} \rangle$$

$$x_i \mid y_i \sim \text{Categorical}(\phi_{y_i})$$

$$y_{i+1} \mid y_i \sim \text{Categorical}(\pi_{y_i})$$

We next discuss how the prior distributions are constructed to build in additional inductive bias.

3.2.1 Transition prior means ($\pi^0_{\mathbf{t}}$)

We use the prior mean for each tag's transition distribution to build in two kinds of bias. First, we want to favor linguistically probable tags. Second, we want to favor transitions that result in a tag pair that combines according to CCG's combinators. Thus, we will define $\pi^0_{\mathbf{t}}$ as a mixture of two components, the first, $P_{\text{CAT}}(\mathbf{u})$ is an (unconditional) distribution over category types $\mathbf{u}$ that favors cross-linguistically probable categories. The second component, $P_{\text{COMB}}(\mathbf{u} \mid \mathbf{t})$, conditions on the previous tag type, $\mathbf{t}$, and assigns higher probability to pairs of tags that can be combined. That is, the probability of transitioning from $\mathbf{t}$ to $\mathbf{u}$ in the Dirichlet mean distribution is given by[1]

$$\pi^0_{\mathbf{t}}(\mathbf{u}) = \lambda \cdot P_{\text{CAT}}(\mathbf{u}) + (1-\lambda) \cdot P_{\text{COMB}}(\mathbf{u} \mid \mathbf{t}).$$

We discuss the two mixture components in turn.

3.2.2 Unigram category generator ($P_{\text{CAT}}(\mathbf{u})$)

In this section, we define a CCG category generator that generates cross-linguistically likely category types. Baldridge's approach estimated the likelihood of a category using the inverse number of sub-categories: $P_{\text{CPLX}}(\mathbf{u}) \propto 1/\textit{complexity}(\mathbf{u})$. We propose an improvement, P_{CAT}, expressed as a probabilistic grammar over the infinite set of CCG categories $\mathcal{T}$.[2] For readability, we use the notation $\overline{p} = (1-p)$.

$C \rightarrow z$		$p_{\textit{punc}} \cdot p_{\textit{puncdist}}(z)$
$C \rightarrow a$		$\overline{p}_{\textit{punc}} \cdot p_{\textit{term}} \cdot p_{\textit{atom}}(a)$
$C \rightarrow A/A$		$\overline{p}_{\textit{punc}} \cdot \overline{p}_{\textit{term}} \cdot p_{\textit{fwd}} \cdot (p_{\textit{mod}} \cdot P_{\text{CAT}}(A) + \overline{p}_{\textit{mod}} \cdot P_{\text{CAT}}(A)^2)$
$C \rightarrow A/B$	$A \neq B$	$\overline{p}_{\textit{punc}} \cdot \overline{p}_{\textit{term}} \cdot p_{\textit{fwd}} \cdot \overline{p}_{\textit{mod}} \cdot P_{\text{CAT}}(A) \cdot P_{\text{CAT}}(B)$
$C \rightarrow A\backslash A$		$\overline{p}_{\textit{punc}} \cdot \overline{p}_{\textit{term}} \cdot \overline{p}_{\textit{fwd}} \cdot (p_{\textit{mod}} \cdot P_{\text{CAT}}(A) + \overline{p}_{\textit{mod}} \cdot P_{\text{CAT}}(A)^2)$
$C \rightarrow A\backslash B$	$A \neq B$	$\overline{p}_{\textit{punc}} \cdot \overline{p}_{\textit{term}} \cdot \overline{p}_{\textit{fwd}} \cdot \overline{p}_{\textit{mod}} \cdot P_{\text{CAT}}(A) \cdot P_{\text{CAT}}(B)$

[1]Following Baldridge (2008), we fix $\lambda = 0.5$ for our experiments.

[2]This is an updated version of the model first presented in (Garrette et al., 2014) to accommodate usage by the parsing models in later chapters.

where A, B, C are categories, z is a punctuation category (and terminal), neither A nor B is a punctuation category, and a is an atomic category (and terminal): $a \in \{\mathsf{s}, \mathsf{n}, \mathsf{np}, ...\}$.[3] Note that the probability of a modifier category (A/A or $A\backslash A$) is the sum of both the probability of being a modifier *and* the probability of being a A/B (or $A\backslash B$), since every A/A is also a A/B; this ensures that probability mass is not lost, making P_{CAT} a valid probability distribution. Also note that punctuation may not be combined with a slash operator as part of a complex category.

We have designed this grammar to capture several important CCG characteristics. In particular we encode four main ideas, each captured through a different parameter of the grammar and discussed in greater detail below:

1. Simpler categories are more likely: e.g. $\mathsf{n/n}$ is *a priori* more likely than $\mathsf{(n/n)/(n/n)}$.
2. Some atoms are more likely than others: e.g. np is more likely than s, and much more than $\mathsf{np_{expl}}$.
3. Modifiers are more likely: e.g. $\mathsf{(s\backslash np)/(s\backslash np)}$ is more likely than non-modifiers with otherwise similar complexity, such as $\mathsf{(s\backslash np)/(np\backslash np)}$.
4. Slash operators may occur with different frequencies.

The first idea subsumes the complexity measure used by Baldridge, but accomplishes the goal naturally by letting the probabilities decrease as the category grows. This ensures that simple categories are *a priori* more likely than complex categories, a trend that can be corroborated by examining the data: for example, the transitive verb "buy" appears with supertag $\mathsf{(s_b\backslash np)/np}$ 342 times in CCGbank, but just once with $\mathsf{(((s_b\backslash np)/pp)/pp)/np}$. The rate of decay is governed by the p_{term} parameter: the marginal probability of generating a terminal (atomic) category in each expansion. A higher p_{term} means a stronger emphasis on simplicity. The probability distribution over categories is guaranteed to be proper so long as $p_{term} > \frac{1}{2}$

[3]While very similar to standard probabilistic context-free grammars seen in NLP work, this grammar is not context-free because modifier categories must have matching operands. However, this is not a problem for our approach since the grammar is unambiguous, defines a proper probability distribution, and is only used for modeling the relative likelihoods of categories (not parsing categories).

since the probability of the depth of a tree will decrease geometrically (Chi, 1999).

The second idea is a natural extension of the complexity concept and is particularly relevant when features are used. The original complexity measure treated all atoms uniformly, but e.g. we would expect $\mathsf{np_{expl}/n}$ to be less likely than $\mathsf{np/n}$ since it contains the more specialized, and thus rarer, atom $\mathsf{np_{expl}}$. We define the distribution $p_{atom}(a)$ as the prior over atomic categories.

Due to our weak, type-only supervision, we have to estimate p_{atom} from just the tag dictionary and raw corpus, without frequency data. Our goal is to estimate the number of each atom in the supertags that should appear on the raw corpus tokens. Since we don't know what the correct supertags are, we first estimate counts of supertags, from which we can extract estimated atom counts. Our strategy is to uniformly distribute each raw corpus token's counts over all of its possible supertags, as specified in the tag dictionary. Word types not appearing in the tag dictionary are ignored for the purposes of these estimates. Assuming that $C(w)$ is the number of times that word type w is seen in the raw corpus, $atoms(a, \mathbf{t})$ is the number of times atom a appears in $\mathbf{t}$, $\textsc{td}(w)$ is the set of tags associated with w, and $\textsc{td}(\mathbf{t})$ is the set of word types associated with $\mathbf{t}$:

$$
\begin{aligned}
C_{supertag}(\mathbf{t}) &= \sum_{w \in \mathrm{TD}(\mathbf{t})} C(w) \;/\; |\textsc{td}(w)| \\
C_{atom}(a) &= \sum_{\mathbf{t} \in \mathcal{T}} atoms(a, \mathbf{t}) \cdot C_{supertag}(\mathbf{t}) \\
p_{atom}(a) &\propto C_{atom}(a) + \delta
\end{aligned}
$$

Adding δ smooths the estimates.

Using the raw corpus and tag dictionary data to set p_{atom} allows us to move beyond Baldridge's work in another direction: it provides us with a natural way to combine CCG's universal assumptions with corpus-specific data.

The third and fourth ideas pertain only to complex categories. If the category is complex, then we consider two additional parameters. The parameter p_{fwd} is the marginal probability that the complex category's operator specifies a forward argument. The parameter p_{mod} gives the amount of marginal probability mass that is specially reserved for modifier categories to be added to their normal ($\overline{p}_{mod}$) probability mass. This addresses the fact that, for example, a category containing six atoms may, in general, be very unlikely, but a six-atom category that is merely a modifier of a three-atom category (like an adverb modifying a transitive verb) would be fairly common.

3.2.3 Bigram category generator ($P_{\text{COMB}}(\mathbf{u} \mid \mathbf{t})$)

While the above processes encode important properties of the distribution over categories, the internal structure of categories is not the full story: cross-linguistically, the categories of adjacent tokens are much more likely to be combinable via some CCG rule. This is the second component of our mixture model.

Baldridge derives this bias by allocating the majority of the transition probability mass from each tag $\mathbf{t}$ to tags that can follow $\mathbf{t}$ according to some combination rule. Let $\kappa(\mathbf{t}, \mathbf{u})$ be an indicator of whether $\mathbf{t}$ combines with $\mathbf{u}$ (in that order); for $\sigma \in [0, 1]$:[4]

$$P_\kappa(\mathbf{u} \mid \mathbf{t}) = \begin{cases} \sigma \cdot \text{uniform}(\mathbf{u}) & \text{if } \kappa(\mathbf{t}, \mathbf{u}) \\ (1-\sigma) \cdot \text{uniform}(\mathbf{u}) & \text{otherwise} \end{cases}$$

There are a few additional considerations that must be made in defining κ, however. In assuming the special tags $\langle\text{S}\rangle$ and $\langle\text{E}\rangle$ for the start and end of the sentence, respectively, we can define $\kappa(\langle\text{S}\rangle, \mathbf{u}) = 1$ when $\mathbf{u}$ seeks no left-side arguments (since there are no tags to the left with which to combine) and $\kappa(\mathbf{t}, \langle\text{E}\rangle) = 1$ when $\mathbf{t}$ seeks no right-side arguments. So $\kappa(\langle\text{S}\rangle, \mathsf{np/n}) = 1$, but $\kappa(\langle\text{S}\rangle, \mathsf{s\backslash np}) = 0$. If atoms have *features* associated, then the atoms are allowed to unify if the features match, or if at least one of them does not have a feature. So $\kappa(\mathsf{np_{nb}}, \mathsf{s\backslash np}) = 1$, but $\kappa(\mathsf{np_{nb}}, \mathsf{s\backslash np_{conj}}) = 0$. In defining κ, it is also important to ignore possible arguments on the wrong side of the combination since they can be consumed without affecting the connection between the two. To achieve this for $\kappa(\mathbf{t}, \mathbf{u})$, it is assumed that it is possible to consume all preceding arguments of $\mathbf{t}$ and all following arguments of $\mathbf{u}$. So $\kappa(\mathsf{np}, \mathsf{(s\backslash np)/np}) = 1$. This helps to ensure the associativity dis-

[4]Again, following Baldridge (2008), we fix $\sigma = 0.95$ for our experiments.

cussed earlier. Finally, the atom np is allowed to unify with n if n is the argument. So $\kappa(\mathsf{n}, \mathsf{s}\backslash\mathsf{np}) = 1$, but $\kappa(\mathsf{np/n}, \mathsf{np}) = 0$. This is due to the fact that CCGBank assumes that n can be rewritten as np.

Type-supervised initialization

As above, we want to improve upon Baldridge's ideas by encoding not just universal CCG knowledge, but also automatically-induced corpus-specific information where possible. To that end, we can define a conditional distribution $P_{tr}(\mathbf{u} \mid \mathbf{t})$ based on statistics from the raw corpus and tag dictionary. We use the same approach as we did above for setting p_{atom} (and the definition of $\phi^0_{\mathbf{t}}$ below): we estimate by evenly distributing raw corpus counts over the tag dictionary entries. Assume that $C(w_1, w_2)$ is the (δ-smoothed) count of times word type w_1 was directly followed by w_2 in the raw corpus, and ignoring any words not found in the tag dictionary:

$$C(\mathbf{t}, \mathbf{u}) = \sum_{w_1 \in \mathrm{TD}(\mathbf{t}),\ w_2 \in \mathrm{TD}(\mathbf{u})} \frac{C(w_1, w_2)}{|\mathrm{TD}(w_1)| \cdot |\mathrm{TD}(w_2)|}$$

$$P_{tr}(\mathbf{u} \mid \mathbf{t}) = \frac{C(\mathbf{t}, \mathbf{u})}{\sum_{\mathbf{u}' \in \mathcal{T}} C(\mathbf{t}, \mathbf{u}')}$$

Then the alternative definition of the compatibility distribution is as follows:

$$P^{tr}_{\kappa}(\mathbf{u} \mid \mathbf{t}) = \begin{cases} \sigma \cdot P_{tr}(\mathbf{u} \mid \mathbf{t}) & \text{if } \kappa(\mathbf{t}, \mathbf{u}) \\ (1-\sigma) \cdot P_{tr}(\mathbf{u} \mid \mathbf{t}) & \text{otherwise} \end{cases}$$

Our experiments compare performance when $\pi^0_{\mathbf{t}}$ is set using $P_{\text{CAT}}(\mathbf{u}) = P_{\text{CPLX}}$ (experiment 3) versus our category grammar P_{CAT} (experiments 4–6), and using $P_{\text{COMB}}(\mathbf{u} \mid \mathbf{t}) = P_{\kappa}$ as the compatibility distribution (experiments 3–4) versus P^{tr}_{κ} (experiments 5–6).

3.2.4 Emission prior means ($\phi^0_{\mathbf{t}}$)

For each supertag type $\mathbf{t}$, $\phi^0_{\mathbf{t}}$ is the mean distribution over words it emits. While Baldridge's approach used a uniform emission initialization, treating all words as equally likely, we can, again, induce token-level corpus-specific information.[5] To set $\phi^0_{\mathbf{t}}$, we use a variant and simplification of the procedure introduced in (Garrette and Baldridge, 2012) that takes advantage of our prior over categories P_{CAT}.

Assuming that $C(w)$ is the count of word type w in the raw corpus, $\text{TD}(w)$ is the set of supertags associated with word type w in the tag dictionary, and $\text{TD}(\mathbf{t})$ is the set of *known* word types associated with supertag $\mathbf{t}$, the count of word/tag pairs for known words (words appearing in the tag dictionary) is estimated by uniformly distributing a word's (δ-smoothed) raw counts over its tag dictionary entries:

$$C_{\textit{known}}(\mathbf{t}, w) = \begin{cases} (C(w) + \delta)/|\text{TD}(w)| & \text{if } \mathbf{t} \in \text{TD}(w) \\ 0 & \text{otherwise} \end{cases}$$

For *unknown* words, we first use the idea of tag "openness" to estimate the likelihood of a particular tag t applying to an unknown word: if a tag applies to many word types, it is likely to apply to some new word type.

$$P(\text{unk} \mid \mathbf{t}) \propto |\text{known words } w \text{ s.t. } \mathbf{t} \in \text{TD}(w)|$$

Then, we apply Bayes' rule to get $P(\mathbf{t} \mid \text{unk})$, and use that to estimate word/tag counts for unknown words:

$$P(\mathbf{t} \mid \text{unk}) \propto P(\text{unk} \mid \mathbf{t}) \cdot P_{\text{CAT}}(\mathbf{t})$$
$$C_{\text{unk}}(\mathbf{t}, w) = C(w) \cdot P(\mathbf{t} \mid \text{unk})$$

[5]Again, without gold tag frequencies.

Thus, with the estimated counts for all words:

$$P_{em}(w \mid \mathtt{t}) = \frac{C_{known}(\mathtt{t}, w) + C_{unk}(\mathtt{t}, w)}{\sum_{w'} C_{known}(\mathtt{t}, w') + C_{unk}(\mathtt{t}, w')}$$

Our experiments compare when $\phi^0_{\mathtt{t}} = P_{em}$ (experiment 6) versus a uniform prior (experiments 3–5).

3.3 Posterior Inference

We wish to find the most likely supertag of each word, given the model we just described and a corpus of training data. Since there is exact inference with these models is intractable, we resort to Gibbs sampling to find an approximate solution. At a high level, we alternate between resampling model parameters $(\phi_{\mathtt{t}}, \pi_{\mathtt{t}})$ given the current tag sequence and resampling tag sequences given the current model parameters and observed word sequences. It is possible to sample a new tagging from the posterior distribution over tag sequences for a sentence, given the sentence and the HMM parameters using the forward-filter backward-sample (FFBS) algorithm (Carter and Kohn, 1996). To efficiently sample new HMM parameters, we exploit Dirichlet-multinomial conjugacy. By repeating these alternating steps and accumulating the number of times each supertag is used in each position, we obtain an approximation of the required posterior quantities.

Our inference procedure takes as input the transition prior means $\pi^0_{\mathtt{t}}$, the emission prior means $\phi^0_{\mathtt{t}}$, and concentration parameters α_π and α_ϕ, along with the raw corpus and tag dictionary. The set of supertags associated with a word w will be known as $\text{TD}(w)$. We will refer to the set of word types included in the tag dictionary as "known" words and others as "unknown" words. For simplicity, we will assume that $\text{TD}(w)$, for any unknown word w, is the full set of CCG categories. During sampling, we always restrict the possible tag choices for a word w to the categories found in $\text{TD}(w)$. We refer to the sequence of word tokens as **x** and tags as **y**.

We initialize the sampler by setting $\pi_{\mathtt{t}} = \pi^0_{\mathtt{t}}$ and $\phi_{\mathtt{t}} = \phi^0_{\mathtt{t}}$ and then sampling tagging sequences using FFBS.

To sample a tagging for a sentence $\mathbf{x}$, the strategy is to inductively compute, for each token x_i starting with $i = 0$ and going "forward", the probability of generating $x_0, x_1, \ldots, x_i$ via any tag sequence that ends with $y_i = \mathbf{u}$. This can be done efficiently using the *forward algorithm*, as is typically done for EM on an HMM:

$$p(y_i = \mathbf{u} \mid x_{0:i}) = \phi_{\mathbf{u}}(x_i) \cdot \sum_{\mathbf{t} \in \mathcal{T}} \pi_{\mathbf{t}}(\mathbf{u}) \cdot p(y_{i-1} = \mathbf{t} \mid x_{0:i-1})$$

We then pass through the sequence again, this time "backward" starting at $i = |\mathbf{x}|$ and sampling

$$y_i \mid y_{i+1} \sim p(y_i = \mathbf{t} \mid x_{0:i}) \cdot \pi_{\mathbf{t}}(y_{i+1}).$$

The block-sampling approach of choosing new tags for a sentence all at once is particularly beneficial given the sequential nature of the model of the HMM. In an HMM, a token's adjacent tags tend to hold onto its current tag due to the relationships between the three. Resampling all tags at once allows for more drastic changes at each iteration, providing better opportunities for mixing during inference. The FFBS approach has the additional advantage that, by resampling the distributions only once per iteration, we are able to resample all sentences in parallel. This is not strictly true of all HMM problems with FFBS, but because our data is divided by sentence, and each sentence has a known start and end tag, the tags chosen during the sampling of one sentence cannot affect the sampling of another sentence in the same iteration.

Once we have sampled tags for the entire corpus, we resample π and ϕ. The newly-sampled tags $\mathbf{y}$ are used to compute $C(w, \mathbf{t})$, the count of tokens with word type w and tag $\mathbf{t}$, and $C(\mathbf{t}, \mathbf{u})$, the number of times tag $\mathbf{t}$ is directly followed by tag $\mathbf{u}$. We then sample, for each $\mathbf{t} \in \mathcal{T}$ where $\mathcal{T}$ is the full set of valid CCG categories:

$$\pi_{\mathbf{t}} \sim \text{Dir}\left(\langle \alpha_\pi \cdot \pi^0_{\mathbf{t}}(\mathbf{u}) + C(\mathbf{t}, \mathbf{u}) \rangle_{\mathbf{u} \in \mathcal{T}}\right)$$
$$\phi_{\mathbf{t}} \sim \text{Dir}\left(\langle \alpha_\phi \cdot \phi^0_{\mathbf{t}}(w) + C(w, \mathbf{t}) \rangle_{w \in V}\right)$$

It is important to note that this method of resampling allows the draws to

Corpus		num. tags	raw tokens	TD tokens	TD entries	ambiguity type	ambiguity token
English	CCGBank POS	50	158k	735k	45k	3.75	13.11
	CCGBank	1,171			65k	56.98	296.18
Chinese	CTB-CCG	829	99k	439k	60k	96.58	323.37
Italian	CCG-TUT	955	6k	27k	9k	178.88	426.13

Table 3.2: Supertag statistics for the various corpora used. CCGBank is English, CCG-CTB is Chinese, and TUT is Italian. The number of tags includes only those tags found in the tag dictionary (TD). Ambiguity rates are the average number of entries in the unpruned tag dictionary for each word in the raw corpus. English POS statistics are shown only for comparison; only CCG experiments were run.

incorporate both the data, in the form of counts, and the prior mean, which includes all of our carefully-constructed biases derived from both the intrinsic, universal CCG properties as well as the information we induced from the raw corpus and tag dictionary.

With the distributions resampled, we can continue the procedure by resampling tags as above, and then resampling distributions again, until a maximum number of iterations is reached.

3.4 Experiments[6]

To evaluate our approach, we used CCGBank (Hockenmaier and Steedman, 2007), which is a transformation of the English Penn Treebank (Marcus et al., 1993); the CTB-CCG (Tse and Curran, 2010) transformation of the Penn Chinese Treebank (Xue et al., 2005); and the CCG-TUT corpus (Bos et al., 2009), built from the TUT corpus of Italian text (Bosco et al., 2000). Statistics on the size and ambiguity of these datasets are shown in Table 3.2.

For CCGBank, sections 00–15 were used for extracting the tag dictionary, 16–18 for the raw corpus, 19–21 for development data, and 22–24 for test data. For TUT, the first 150 sentences of each of the CIVIL_LAW and NEWSPAPER sec-

[6]All code and experimental scripts are available at: `www.github.com/dhgarrette/2014-ccg-supertagging`

tions were used for raw data, the next sentences 150–249 of each was used for development, and the sentences 250–349 were used for test; the remaining data, 457 sentences from CIVIL_LAW and 548 from NEWSPAPER, plus the much smaller 132-sentence JRC_ACQUIS data, was used for the tag dictionary. For CTB-CCG, sections 00–11 were used for the tag dictionary, 20–24 for raw, 25–27 for dev, and 28–31 for test.

Because we are interested in showing the relative gains that our ideas provide over Baldridge (2008), we reimplemented the initialization procedure from that paper, allowing us to evaluate all approaches consistently. For each dataset, we ran a series of experiments in which we made further changes from the original work. We first ran a baseline experiment with uniform transition and emission initialization of EM (indicated as “1.” in Table 3.3) followed by our reimplementation of the initialization procedure by Baldridge (2). We then experimented with the Bayesian formulation, first using the same information used by Baldridge, and then adding our enhancements: using our category grammar in P_{CAT}, using P_{κ}^{tr} as the transition compatibility distribution, and using P_{em} as $\phi_{\mathbf{t}}^{0}(w)$.

For each dataset, we ran experiments using four different levels of tag dictionary pruning. Pruning is the process of artificially removing noise from the tag dictionary by using token-level annotation counts to discard low-probability tags; for each word, for cutoff x, any tag with probability less than x is excluded. Tag dictionary pruning is a standard procedure in type-supervised training, but because it requires information that does not truly conform to the type-supervised scenario, we felt that it was critical to demonstrate the performance of our approach under situations of less pruning, *including* no artificial pruning at all.

We emphasize that unlike in most previous work, we use *incomplete* tag dictionaries. Most previous work makes the unrealistic assumption that the tag dictionary contains an entry for every word that appears in either the training *or testing* data. This is a poor approximation of a real tagging system, which will never have complete lexical knowledge about the test data. Even work that only assumes complete knowledge of the tagging possibilities for the lexical items in the training corpus is problematic (Baldridge, 2008; Ravi et al., 2010a). This still makes learning unrealistically easy since it dramatically reduces the ambiguity of words that would have been unseen, and, in the case of CCG, introduces additional tags that would not have otherwise been known. To ensure that our experiments are more realistic, we draw our tag dictionary entries from data that is totally disjoint from both the raw and test corpora. During learning, any unknown words (words not appearing in the tag dictionary) are unconstrained so that they may take *any* tag that has been seen with any word in the tag dictionary, and are, thus, maximally ambiguous.

We only performed minimal parameter tuning, choosing instead to stay consistent with Baldridge (2008) and simply pick reasonable-seeming values for any additional parameters. Any tuning that was performed was done with simple hill-climbing on the development data of English CCGBank. All parameters were held consistent across experiments, including across languages. For EM, we used 50 iterations; for FFBS we used 100 burn-in iterations and 200 sampling iterations.[7] For all experiments, we used $\sigma = 0.95$ for $P_{\kappa}^{(tr)}$ and $\lambda = 0.5$ for $\pi_{\mathbf{t}}^{0}$ to be consistent with previous work, $\alpha_{\pi} = 3000$, $\alpha_{\phi} = 7000$, $p_{term} = 0.6$, $p_{fwd} = 0.5$, $p_{mod} = 0.8$, and $\delta = 1000$ for p_{atom}. These hyperparameters were tuned via a coarse grid-search, using the development set to evaluate a few values for each hyperparameter. Test data was run only once, for the final figures.

The final results reported were achieved by using the following training sequence: initialize parameters according to the scenario, train an HMM using EM or FFBS starting with that set of parameters, tag the raw corpus with the trained HMM, add-0.1 smooth counts from the now-tagged raw corpus, and train a maximum entropy Markov model (MEMM) from this "auto-supervised" data.[8]

Results are shown in Table 3.3. Most notably, the contributions described in this chapter improve results in nearly every experimental scenario. We can see immediate, often sizable, gains in most cases simply by using the Bayesian formulation. Further gains are seen from adding each of the other various contributions of

[7]Final counts are averaged across the sampling iterations.

[8]Auto-supervised training of an MEMM increases accuracy by 1–3% on average (Garrette and Baldridge, 2013). We use the OpenNLP MEMM implementation with its standard set of features: `http://opennlp.apache.org`

Corpus		English	Chinese	Italian
1. uniform	EM	38	26	30
2. init (Baldridge)	EM	41	28	32
3. init	Bayes	42	37	40
4. P_{CAT}	Bayes	42	36	40
5. P_{CAT}, P^{tr}_{κ}	Bayes	50	44	43
6. P_{CAT}, P^{tr}_{κ}, P_{em}	Bayes	**51**	**49**	**46**

(a) Main results.

Corpus		English			Chinese			Italian		
TD cutoff		0.1	0.01	0.001	0.1	0.01	0.001	0.1	0.01	0.001
1. uniform	EM	77	62	47	64	39	30	51	32	30
2. init (Baldridge)	EM	78	67	55	66	43	33	**54**	36	33
3. init	Bayes	74	68	56	65	56	47	52	46	40
4. P_{CAT}	Bayes	74	70	59	64	57	47	52	40	39
5. P_{CAT}, P^{tr}_{κ}	Bayes	75	72	61	66	58	49	52	44	41
6. P_{CAT}, P^{tr}_{κ}, P_{em}	Bayes	**80**	**80**	**73**	**69**	**62**	**56**	53	**47**	**45**

(b) Results varying the degree of tag dictionary pruning cutoff, see text.

Table 3.3: Experimental results: test-set per-token supertag accuracies. (1) is uniform EM initialization. (2) is a reimplementation of (Baldridge, 2008). (3) is Bayesian formulation using only the ideas from Baldridge: P_{CPLX}, P_{κ}, and uniform emissions. (4–6) are our enhancements to the prior: using our category grammar in P_{CAT} instead of P_{CPLX}, using P^{tr}_{κ} instead of P_{κ}, and using P_{em} instead of uniform.

this chapter. Perhaps most interestingly, the gains are only minimal with maximum pruning, but the gains increase as the pruning becomes less aggressive — as the scenarios become more realistic. This indicates that our improvements make the overall procedure more robust.

3.4.1 Error analysis

Like POS-taggers, the learned supertagger frequently confuses nouns (n) and their modifiers ($\mathsf{n/n}$), but the most frequent error made by the English (6) experiment was ($\mathsf{((s\backslash np)\backslash(s\backslash np))/n}$) instead of ($\mathsf{np_{nb}/n}$). However, these are both determiner types, indicating an interesting problem for the supertagger: it often predicts an object type-raised determiner instead of the vanilla $\mathsf{np/n}$, but in many contexts, both categories are equally valid. (In fact, for parsers that use type-raising as a rule, this distinction in lexical categories does not exist.)

3.5 Related Work

Ravi et al. (2010a) also improved upon the work by Baldridge (2008) by using integer linear programming to find a minimal model of supertag transitions, thereby generating a better starting point for EM than the grammatical constraints alone could provide. This approach is complementary to the work presented here, and because we have shown that our work yields gains under tag dictionaries of various levels of cleanliness, it is probable that employing minimization to set the base distribution for sampling could lead to still higher gains.

On the Bayesian side, Van Gael et al. (2009) used a nonparametric, *infinite* HMM for truly unsupervised POS-tagger learning (Van Gael et al., 2008; Beal et al., 2001): their model is not restricted to the standard set of POS tags, and may learn a more fine-grained set of labels. (We will return to this comparison in §4.5 with our discussion of nonparametric CCG models.)

Finally, we find the task of weakly-supervised supertagger learning to be particularly relevant given the recent surge in popularity of CCG. An array of NLP applications have begun using CCG, including semantic parsing (Zettlemoyer and

Collins, 2005) and machine translation (Weese et al., 2012). As CCG finds more applications, and as these applications move to lower-resource domains and languages, there will be increased need for the ability to learn without full supervision.

3.6 Conclusion and Future Work

Standard strategies for type-supervised HMM estimation are less effective as the number of categories increases. In contrast to POS tag sets, CCG supertags, while quite numerous, have structural clues that can simplify the learning problem. Baldridge (2008) used this formalism-specific structure to inform an initialization procedure for EM. In this work, we have shown that CCG structure can instead be used to motivate an effective *prior distribution* over the parameters of an HMM supertagging model, allowing our work to outperform Baldridge's approach, and to do so in a principled manner that lends itself better to future extensions such as incorporation in more complex models.

This work also improves on Baldridge's simple "complexity" measure, developing instead a probabilistic category grammar over supertags that allows our prior to capture a wider variety of interesting and useful properties of the CCG formalism. This well-defined distribution over the infinite space of CCG categories also positions our work well for extensions, as we will see with our model in §4.5.

Finally, we were able to achieve further gains by augmenting the universal CCG knowledge with corpus-specific information that could be automatically extracted from the weak supervision that is available: the raw corpus and the tag dictionary. This allows us to combine the cross-linguistic properties of the CCG formalism with corpus- or language-specific information in the data into a single, unified Bayesian prior.

Our model uses a relatively large number of parameters, e.g., p_{term}, p_{fwd}, p_{mod}, p_{atom}, in the prior. Here, we fixed each to a single value (i.e., a "fully Bayesian" approach). Future work might explore sensitivity to these choices, or empirical Bayesian or maximum *a posteriori* inference for their values (Johnson and Goldwater, 2009).

In this work, as in most type-supervised work, the tag dictionary was automatically extracted from an existing tagged corpus. However, a tag dictionary could instead be automatically induced via multi-lingual transfer (Das and Petrov, 2011) or generalized from human-provided information as we did in Chapter 2. Again, since the approach presented here has been shown to be somewhat robust to tag dictionary noise, it is likely that the model would perform well even when using an automatically-induced tag dictionary.

Chapter 4
Grammar-Informed CCG Parsing

Supervised learning of natural language parsers of various types (context-free grammars, dependency grammars, categorial grammars, and the like) is by now a well-understood task with plenty of high-performing models—when training data is abundant. Learning from sparse, incomplete information is, naturally, a greater challenge. To build parsers for domains and languages where resources are scarce, we need techniques that take advantage of very limited kinds and amounts of supervision.

In this chapter, we extend the intuitions developed for Combinatory Categorial Grammar (CCG) supertagging in Chapter 3. For the supertagger, we developed a probability distribution over CCG categories (§3.2.2) that captured some of the intrinsic, cross-lingual properties of the CCG formalism to use as a prior on the categories chosen by the sequence model. While this method was successful in improving supertagging performance in weak-supervision scenarios, supertags are just the start of a full syntactic analysis of a sentence, and—for all but very short sentences—an HMM is very unlikely to produce a sequence of supertags that can actually be combined into a complete tree.

In this chapter, we present models for CCG parsing that are designed to take advantage of the universal properties of CCG that we used to improve supertagging. We begin by showing how the prior over categories (§3.2.2) can be incorporated into a standard PCFG parsing model and used to bias the categories of all constituents, at all levels of the tree, toward simpler, *a priori* more likely categories for type-supervised parsing (§4.2) (Garrette et al., 2015). Then we present a novel model that goes further by extending the standard PCFG to include *supertag contexts* to the left and right side of each constituent, allowing the model to exploit the natural

Section §4.2 is based on the following publication: Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. 2015. Weakly-supervised grammar-informed Bayesian CCG parser learning. In Proc. of AAAI. My co-authors worked with me to develop the ideas underlying this work.

associativity of CCG, and allowing us to incorporate the supertagger's connectivity bias (§3.2.3), pushing the model toward constituent categories that "fit" with their contexts (§4.3). Finally, we discuss ways of moving beyond type-level supervision, evaluating the utility of *bracket* annotations, a lightweight form of supervision that provides complementary information (§4.4). At the end of this chapter, we include a discussion of how Bayesian nonparametric techniques might be employed in type-supervised parser learning in order to overcome the limitations of a fixed, finite tag dictionary, by allowing our model explore the full, infinite space of CCG categories to introduce new categories as they are needed (§4.5).

4.1 Combinatory Categorial Grammar[1]

Because of their structured nature, CCG categories (unlike the part-of-speech tags and non-terminals of a standard PCFG) contain information that gives evidence of their frequencies. In §3.2.2, we developed a category generator that defined a probability distribution P_{CAT} over the infinite set of CCG categories and used it as a prior that encouraged more cross-linguistically likely categories. Supertagging accuracy was improved further by complementing the language-universal knowledge from CCG with corpus-specific information extracted automatically from the tag dictionary and raw corpus. Counts were estimated from the data then used to empirically set the parameters of the priors.

There has been much work on statistical CCG parsing since its development, including early work on learning stochastic grammars by Osborne and Briscoe (1997), among others, but the development of CCGBank, a corpus of CCG-parsed newswire sentences (Hockenmaier and Steedman, 2007), has made a wide array of machine learning approaches possible. Notable examples include generative models developed by Hockenmaier and Steedman (2002) and log-linear parsing by Clark and Curran (2007). More recent approaches by Bisk and Hockenmaier (2013) have included learning simplified CCG representations from POS tags but without explicit CCG tree supervision.

[1]We refer the reader to §3.1 for a more general discussion of CCG.

Beyond the properties that make it desirable for constructing syntactic parsers, CCG's tight connection between syntax and semantics has made it particularly popular as a tool for *semantic parsing* (Zettlemoyer and Collins, 2005, *inter alia*). In CCG, it is possible to pair each syntactic category with a lambda-calculus expression specifying the semantic content of the particular constituent. As categories are combined to construct a syntactic derivation, the corresponding meaning expressions follow the same paths of combination, finally deriving a logical form representation that describes the whole sentence. We have also seen CCG used for machine translation (Weese et al., 2012) and semantic role labeling (Gildea and Hockenmaier, 2003), among other applications.

4.2 CCG Parsing with Likely Categories

In Chapter 3, we introduced two universal properties of the CCG formalism that we used to design priors for a CCG supertagger. The first of these was that some CCG categories are intrinsically more likely than others. We showed that it is possible to define a prior over the *a priori* likelihood of a particular category using a probabilistic grammar that is biased toward "simple" categories, as seen in §3.2.2, and that this prior allows us to learn better better supertaggers by biasing the model toward more likely categories.

In this section, we define a parsing model, and extend the principle of preferring simpler categories to higher-level constituents in the tree, defining priors over grammar productions that bias the model toward *a priori* more likely categories throughout the tree.

Our inputs are unannotated sentences and an incomplete tag dictionary mapping some words to their potential categories, and we model CCG trees with a probabilistic context-free grammar (PCFG). The parameters of the PCFG are estimated using a blocked sampling algorithm based on the Markov chain Monte Carlo approach of Johnson et al. (2007). This allows us to efficiently sample parse trees for sentences in an unlabeled training corpus according to their posterior probabilities as influenced by the linguistically-informed priors. This approach yields improvements over a baseline PCFG that uses only uninformative priors. Further, as a demonstration of the universality of our approach in capturing valuable grammatical biases, we evaluate on three diverse languages: English, Italian, and Chinese (Garrette et al., 2015).

4.2.1 Generative Model

Our CCG parsing model assumes the following generative process. First, the parameters that define our PCFG are drawn. We generate a distribution σ over root categories, a conditional distribution $\theta_{\mathbf{t}}$ over binary branching non-terminal productions given each category **t**, a conditional distribution $\pi_{\mathbf{t}}$ over unary non-terminal productions given each category **t**, and a conditional distribution $\mu_{\mathbf{t}}$ over terminal (word) productions given each category **t**. Each of these parameters is drawn from a Dirichlet distribution parameterized by a concentration parameter $(\alpha_\sigma, \alpha_\theta, \alpha_\pi, \alpha_\mu)$ and a prior mean distribution $(\sigma^0, \theta^0, \pi^0, \mu^0_{\mathbf{t}})$. By setting each α close to zero, we can bias learning toward relatively peaked distributions. The prior means, explained in detail below, are used to encode both universal linguistic knowledge as well as information automatically extracted from the weak supervision.

Note that unlike a standard phrase-structure grammar where the sets of terminal and non-terminal labels are non-overlapping (part-of-speech tags vs. internal nodes), a CCG category may appear at any level the tree and, thus, may yield binary, unary, or terminal word productions. Therefore, we also generate a distribution $\lambda_{\mathbf{t}}$ for every category $\mathbf{t}$ that defines the mixture over production types (binary, unary, terminal) yielded by $\mathbf{t}$. These parameters are generated by draws from a Dirichlet distribution parameterized by concentration parameter α_λ and prior mean distribution λ^0.

Next, the process generates each sentence in the corpus. This begins by generating a root category **s** and then recursively generating subtrees. For each subtree rooted by a category **t**, with probability determined by $\lambda_{\mathbf{t}}$, we generate either a binary ($\langle \mathbf{u}, \mathbf{v} \rangle$), unary ($\langle \mathbf{u} \rangle$), or terminal ($w$) production from **t**; for binary and unary productions, we generate child categories and recursively generate subtrees. A tree is complete when all branches end in terminal words.

```
Parameters:
  $\sigma \sim \text{Dirichlet}(\alpha_\sigma, \sigma^0)$                                      root categories
  $\theta_{\mathbf{t}} \sim \text{Dirichlet}(\alpha_\theta, \theta^0)$                   $\forall \mathbf{t} \in \mathcal{T}$   binary productions
  $\pi_{\mathbf{t}} \sim \text{Dirichlet}(\alpha_\pi, \pi^0)$                   $\forall \mathbf{t} \in \mathcal{T}$   unary productions
  $\mu_{\mathbf{t}} \sim \text{Dirichlet}(\alpha_\mu, \mu^0_{\mathbf{t}})$                   $\forall \mathbf{t} \in \mathcal{T}$   terminal productions
  $\lambda_{\mathbf{t}} \sim \text{Dirichlet}(\alpha_\lambda, \langle \lambda^0(1), \lambda^0(2), \lambda^0(3) \rangle)$   $\forall \mathbf{t} \in \mathcal{T}$   production mixture

Sentence:
  $\mathbf{s} \sim \text{Categorical}(\sigma)$
  generate($\mathbf{s}$)
where
  function generate($\mathbf{t}$) :
    $z \sim \text{Categorical}(\lambda_{\mathbf{t}})$
    if $z = 1$ : $\langle \mathbf{u}, \mathbf{v} \rangle \mid \mathbf{t} \sim \text{Categorical}(\theta_{\mathbf{t}})$
              Tree($\mathbf{t}$, generate($\mathbf{u}$), generate($\mathbf{v}$))
    if $z = 2$ : $\langle \mathbf{u} \rangle \mid \mathbf{t} \sim \text{Categorical}(\pi_{\mathbf{t}})$
              Tree($\mathbf{t}$, generate($\mathbf{u}$)))
    if $z = 3$ : $w \mid \mathbf{t} \sim \text{Categorical}(\mu_{\mathbf{t}})$
              Leaf($\mathbf{t}$, $w$)
```

Root prior mean (σ^0)

Since σ is a distribution over root categories, we can use P_{CAT}, the probability of a category as defined above in terms of the category grammar, as its prior mean, biasing our model toward simpler root categories. Thus, $\sigma^0(\mathbf{t}) = P_{\text{CAT}}(\mathbf{t})$.

Non-terminal production prior means (θ^0 and π^0)

Our model includes two types of non-terminal productions: binary productions of the form "$\mathbf{t} \rightarrow \langle \mathbf{u}, \mathbf{v} \rangle$", and unary productions of the form "$\mathbf{t} \rightarrow \langle \mathbf{u} \rangle$". As with the root distribution prior, we would like our model to prefer productions that yield high-likelihood categories. To provide this bias, we again use P_{CAT}:

$$\theta^0(\langle \mathbf{u}, \mathbf{v} \rangle) = P_{\text{CAT}}(\mathbf{u}) \cdot P_{\text{CAT}}(\mathbf{v})$$
$$\pi^0(\langle \mathbf{u} \rangle) = P_{\text{CAT}}(\mathbf{u})$$

Terminal production prior means ($\mu^0_{\mathbf{t}}$)

Because we model terminal productions separately, we are able to borrow directly from the supertagger model to define the terminal production prior mean $\mu^0_{\mathbf{t}}$ in a way that exploits the dictionary and unlabeled corpus to estimate the distribution over words for each supertag. Terminal productions in our grammar are defined as "word given supertag," which is exactly the relationship of the emission distribution in an HMM supertagger. Thus, we simply use the supertagger's emission prior mean, as defined in §3.2.4, for our terminal productions:

$$\mu^0_{\mathbf{t}}(w) = P_{em}(w \mid \mathbf{t})$$

Production type mixture prior means ($\lambda^0_{\mathbf{t}}$)

For our experiments, we use an uninformative prior over production types: $\lambda^0 = \langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \rangle$ and $\alpha_\lambda = 3$.

Decoding

In order to parse with our model, we seek the highest-probability parse tree for a given sentence $\mathbf{w}$:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{w}).$$

This can be computed efficiently using the well-known probabilistic CKY algorithm.

4.2.2 Posterior Inference

Since inference about the parameters of our model using a corpus of unlabeled training data is intractable, we resort to Gibbs sampling to find an approximate solution. Our strategy is based on that of Johnson et al. (2007), using a block sampling approach. We initialize our parameters by setting each distribution to its prior mean ($\sigma = \sigma^0$, $\theta_{\mathbf{t}} = \theta^0$, etc.) We then alternate between sampling trees given the current model parameters and observed word sequences, and sampling model parameters $(\sigma, \theta, \pi, \mu, \lambda)$ given the current set of parse trees. To efficiently sample new model parameters, we exploit Dirichlet-multinomial conjugacy. We accumulate all parse trees sampled across all sampling iterations and use them to approximate the posterior quantities.

Our inference procedure takes as input each of the distribution prior means $(\sigma^0, \theta^0, \pi^0, \mu^0)$, along with the raw corpus and tag dictionary. During sampling, we always restrict the possible supertag choices for a word w to the categories found in the tag dictionary entry for that w: $\text{TD}(w)$. Since real-world learning scenarios will always lack complete knowledge of the lexicon, we, too, want to allow for unknown words. Thus, we use *incomplete* tag dictionaries in our experiments, meaning that for a word w not present in the dictionary, we assign $\text{TD}(w)$ to be the full set of known categories, indicating maximal ambiguity. It is also possible that the "correct" supertag for a given word is not present in the tag dictionary, though in these scenarios we hope that the parse will succeed through a different route.

Our Gibbs sampler, based on the one proposed by Goodman (1998) and used by Johnson et al. (2007), uses a block sampling approach to sample an entire parse tree at once. The procedure is similar in principle to the Forward-Filter Backward-Sampler algorithm used in §3.3 for the HMM supertagger, but sampling trees instead of sequences. To sample a tree for a sentence $\mathbf{w}$, the strategy is to use the Inside algorithm (Lari and Young, 1990) to inductively compute, for each potential non-terminal position (i, j) (spanning words w_i through w_{j-1}) and category $\mathbf{t}$, going "up" the tree, the probability of generating $w_i, \ldots, w_{j-1}$ via any arrangement of productions that is rooted by $y_{ij} = \mathbf{t}$:

$$\begin{aligned}
p(w_i \mid y_{i,i+1} = \mathbf{t}) &= \lambda_{\mathbf{t}}(3) \cdot \mu_{\mathbf{t}}(w_i) \\
&\quad + \sum_{\mathbf{t} \to \mathbf{u}} \lambda_{\mathbf{t}}(2) \cdot \pi_{\mathbf{t}}(\langle \mathbf{u} \rangle) \cdot p(w_{i:j-1} \mid y_{ij} = \mathbf{u}) \\
p(w_{i:j-1} \mid y_{ij} = \mathbf{t}) &= \\
&\quad \sum_{\mathbf{t} \to \mathbf{u}} \lambda_{\mathbf{t}}(2) \cdot \pi_{\mathbf{t}}(\langle \mathbf{u} \rangle) \cdot p(w_{i:j-1} \mid y_{ij} = \mathbf{u}) \\
&\quad + \sum_{\mathbf{t} \to \mathbf{u}\ \mathbf{v}} \sum_{i<k<j} \lambda_{\mathbf{t}}(1) \cdot \theta_{\mathbf{t}}(\langle \mathbf{u}, \mathbf{v} \rangle) \cdot p(w_{i:k-1} \mid y_{ik} = \mathbf{u}) \cdot p(w_{k:j-1} \mid y_{kj} = \mathbf{v})
\end{aligned}$$

We then pass through the chart again, this time "downward" starting at the root (y_{0n}) and sampling productions until we reach a terminal word on all branches:

$$\begin{aligned}
y_{0n} &\sim \sigma_{\mathbf{t}} \cdot p(w_{0:n-1} \mid y_{0n} = \mathbf{t}) \\
x \mid y_{ij} &\sim \Big\langle \theta_{y_{ij}}(\langle \mathbf{u}, \mathbf{v} \rangle) \cdot p(w_{i:k-1} \mid y_{ik} = \mathbf{u}) \cdot p(w_{k:j-1} \mid y_{kj} = \mathbf{v}) \\
&\qquad\qquad\qquad\qquad \forall\, y_{ik}, y_{kj} \ \text{ when } j > i+1, \\
&\qquad \pi_{y_{ij}}(\langle \mathbf{u} \rangle) \cdot p(w_{i:j-1} \mid y'_{ij} = \mathbf{u}) \qquad \forall\, y'_{ij}, \\
&\qquad \mu_{y_{ij}}(w_i) \qquad\qquad \text{when } j = i+1 \Big\rangle
\end{aligned}$$

where x is either a split point k and pair of categories y_{ik}, y_{kj} resulting from a binary rewrite rule, a single category y'_{ij} resulting from a unary rule, or a word w resulting from a terminal rule.

Resampling the parameters uses the just-sampled parse trees $\mathbf{y}$ to compute $C_{root}(\mathbf{t})$, the count of trees in which the root category is $\mathbf{t}$, $C(\mathbf{t} \to \langle \mathbf{u}, \mathbf{v} \rangle)$, the count of binary non-terminal productions whose category is $\mathbf{t}$ that are producing the pair of categories $\langle \mathbf{u}, \mathbf{v} \rangle$, the count of unary non-terminal productions $C(\mathbf{t} \to \langle \mathbf{u} \rangle)$, and the count of terminal productions $C(\mathbf{t} \to w)$. We then sample, for each $\mathbf{t} \in \mathcal{T}$ where $\mathcal{T}$ is the full set of valid CCG categories (and V is the full vocabulary of known words):

$$
\begin{aligned}
\sigma &\sim \text{Dirichlet}(\langle\ \alpha_\sigma \cdot \sigma^0(\mathbf{t}) + C_{\mathit{root}}(\mathbf{t})\ \rangle_{\mathbf{t}\in\mathcal{T}}) \\
\theta_{\mathbf{t}} &\sim \text{Dirichlet}(\langle\ \alpha_\theta \cdot \theta^0(\langle\mathbf{u},\mathbf{v}\rangle) + C(\mathbf{t}\rightarrow\langle\mathbf{u},\mathbf{v}\rangle)\ \rangle_{\mathbf{u},\mathbf{v}\in\mathcal{T}}) \\
\pi_{\mathbf{t}} &\sim \text{Dirichlet}(\langle\ \alpha_\pi \cdot \pi^0(\langle\mathbf{u}\rangle) + C(\mathbf{t}\rightarrow\langle\mathbf{u}\rangle)\ \rangle_{\mathbf{u}\in\mathcal{T}}) \\
\mu_{\mathbf{t}} &\sim \text{Dirichlet}(\langle\ \alpha_\mu \cdot \mu^0_{\mathbf{t}}(w) + C(\mathbf{t} \rightarrow w)\ \rangle_{w\in V}) \\
\lambda_{\mathbf{t}} &\sim \text{Dirichlet}(\langle\ \alpha_\lambda \cdot \lambda^0(1) + \textstyle\sum_{\mathbf{u},\mathbf{v}\in\mathcal{T}} C(\mathbf{t}\rightarrow\langle\mathbf{u},\mathbf{v}\rangle), \\
&\qquad\qquad\qquad \alpha_\lambda \cdot \lambda^0(2) + \textstyle\sum_{\mathbf{u}\in\mathcal{T}} C(\mathbf{t}\rightarrow\langle\mathbf{u}\rangle), \\
&\qquad\qquad\qquad \alpha_\lambda \cdot \lambda^0(3) + \textstyle\sum_{w\in V} C(\mathbf{t}\rightarrow w) \qquad \rangle)
\end{aligned}
$$

These distributions are derived from the conjugacy of the Dirichlet prior to the multinomial; note that the result selects parameters based on both the data (counts) and the bias encoded in the prior.

After all sampling iterations have completed, the parameters are estimated as the maximum likelihood estimate of the pool of trees resulting from all sampling iterations. Dramatic time savings can be obtained by generating and reusing a chart that compactly stores all possible parses for all possible sentences. This allows avoiding calculation for subtrees and productions that never participate in a complete parse.

As a further optimization, we enforce the use of punctuation as phrasal-boundary indicators, a technique used previously by Ponvert et al. (2011) and Spitkovsky et al. (2011). This means that when we attempt to parse a sentence (or sample a parse tree for a sentence), we do not allow constituents that cross punctuation without covering the whole inter-punctuation phrase. For example, the sentence "On Sunday, he walked.", the constituent "Sunday , he" would be disallowed. Since punctuation does regularly mark a phrasal boundary, this choice has negligible effect on accuracy while reducing runtime and memory use. In cases where a punctuation-as-boundary requirement (along with the tag dictionary) renders a sentence unparseable according to the CCG rules, we lift the punctuation requirement for that sentence.

4.2.3 Experiments

We evaluated our approach on the three available CCG corpora: English CCGBank (Hockenmaier and Steedman, 2007), Chinese Treebank CCG (Tse and Curran, 2010), and the Italian CCG-TUT corpus (Bos et al., 2009). Each corpus was split into four non-overlapping datasets: a portion for constructing the tag dictionary, sentences for the unlabeled training data, development trees (used for tuning α, p_{term}, p_{mod}, and p_{fwd} hyperparameters), and test trees. We used the same splits as Garrette et al. (2014). Since these treebanks use special representations for conjunctions, we chose to rewrite the trees to use conjunction categories of the form $(X\backslash X)/X$ so that additional special rules would not need to be introduced.

We ran our sampler for 50 iterations.[2] For the category grammar, we used p_{term}=0.7, p_{mod}=0.1, p_{fwd}=0.5. For the priors, we use α_σ=1, α_θ=100, α_π=10,000, α_μ=10,000.[3] We trained on 1,000 sentences for English and 750 for Chinese, but only 150 for Italian since it is a much smaller corpus.

Unlike many CCG parsers, we do not use supertagging as a preprocessing step during either inference or parsing (Clark and Curran, 2007; Lewis and Steedman, 2014). This means that for any word token, there may be a very large number of potential supertag choices—with as many as 1,300 for words that do not appear at all in the tag dictionary. As a result, the potential for spurious ambiguity—different CCG parses that result in the same dependencies—is very high. To limit the amount of spurious ambiguity, we follow Lewis and Steedman (2014) in allowing only a small set of generic, linguistically-plausible grammar rules, and adopt their set of 13 unary category-rewriting rules. The set of rules allowed by our parser is given in Table 4.1. For binary combinations, we allow for forward and backward application, as well as rules for combining with punctuation to the left and right. We further allow for a *merge* rule since this is seen frequently in the corpora (Clark and

[2]We experimented with higher numbers of iterations but found that accuracy was not improved past 50 iterations.

[3]In order to ensure that these concentration parameters, while high, were not dominating the posterior distributions, we ran experiments in which they were set much higher (including using the prior alone), and found that accuracies plummeted in those cases, demonstrating that there is a good balance with the prior.

rule	**t**	$\rightarrow$	$\langle$ **u** ,	**v** $\rangle$
forward application	X		X/Y	Y
backward application	X		Y	X\Y
right punctuation	X		X	P
left punctuation	X		P	X
merge	X		X	X

(a) Binary rules used by the parser. X and Y represent any CCG category; P, any punctuation category.

usage	**t**	$\rightarrow$ $\langle$ **u** $\rangle$
bare NP	np	n
type raising	s/(s\np)	np
	(s\np)/((s\np)/np)	np
	(s\np)/((s\np)/pp)	pp
reduced relative clauses	np\np	$\mathsf{s}_{\mathsf{pss}}$\np
	np\np	s_{ng}\np
	np\np	$\mathsf{s}_{\mathsf{adj}}$\np
	np\np	s_{to}\np
	n\n	s_{to}\np
	np\np	$\mathsf{s}_{\mathsf{dcl}}$/np
VP sentence modifiers	s/s	$\mathsf{s}_{\mathsf{pss}}$\np
	s/s	s_{ng}\np
	s/s	s_{to}\np

(b) Unary rules used by the parser. From Lewis and Steedman (2014).

Table 4.1: All rules used by the parser.

Curran, 2007). CCG composition rules are rarely necessary to parse a sentence, but do increase the overall number of parses, many of which represent the same underlying grammatical structures. This choice drastically reduces the time and space requirements for learning, without sacrifices in accuracy. Allowing the *backward crossed composition* rule—the third most-frequent rule in CCGBank—not only dramatically increases the time and memory requirements, but also tends to lower the accuracy of the resulting parser by 1% or more, likely because it increases ambiguity.

CCG parsers are typically evaluated on the *dependencies* they produce instead of their CCG derivations directly (Bisk and Hockenmaier, 2012; Lewis and Steedman, 2014). There can be many different CCG parse trees that all represent the same dependency relationships (spurious ambiguity), and CCG-to-dependency conversion can collapse those differences. To convert a CCG tree into a dependency tree, we traverse the parse tree, dictating at every branching node which words will be the dependents of which. For binary branching nodes of *forward* rules, the right side—the argument side—is the dependent, unless the left side is a modifier (X/X) of the right, in which case the left is the dependent. The opposite is true for *backward* rules. For *punctuation* rules, the punctuation is always the dependent. For *merge* rules, the right side is always made the parent. The results presented in this chapter are dependency accuracy scores: the proportion of words that were assigned the correct parent (or "root" for the root of a tree).

During training, we only use sentences for which we are able to find at least one parse since we cannot sample a parse tree for a sentence that has no available parses. However, for testing, we must make extra efforts to find valid parses. To that end, if we encounter a test sentence that cannot be parsed given the tag dictionary and CCG rules (either with or without enforcement of a punctuation-as-boundary requirement), then we fall back to a plan in which additional supertag options are added for each token. For a word w_i, we add the set of tags $X\backslash X$ for all $X \in$ TD(w_{i-1}), and Y/Y for all $Y \in$ TD(w_{i+1}). Since $X\backslash X$ and Y/Y represent *modifier* categories, the result of these categories is to provide the parser the option of simply making w_i a modifier of one of its immediate neighbors, resulting in w_i simply

	English	Chinese	Italian
1. uniform	53.38	35.94	58.16
2. P_{CAT}	54.75	40.08	59.21
3. P_{CAT}, P_{em}	**55.69**	**42.00**	**60.04**

(a) Main results.

	English			Chinese			Italian		
	0.001	0.01	0.1	0.001	0.01	0.1	0.001	0.01	0.1
1.	56.62	61.30	56.46	34.88	41.24	47.42	**60.29**	59.91	**54.72**
2.	57.79	61.93	56.69	41.59	42.86	47.74	58.58	**60.76**	53.31
3.	**60.12**	**62.11**	**57.20**	**43.42**	**43.84**	**49.40**	59.60	60.02	53.39

(b) Increasing degrees of artificial tag dictionary pruning (see text). The pruning cutoff is given at the top of each column.

Table 4.2: Experimental results: test-set dependency accuracies. (1) uses uniform priors on all distributions. (2) uses the category prior P_{CAT}. (3) uses the tag dictionary and raw corpus to automatically estimate category prior and word production information. Table (a) gives the results obtained when no artificial tag dictionary pruning was performed; table (b) shows performance as the artificial pruning is increased.

being assigned as a dependent of that neighboring word. This effectively allows the parser to ignore inconvenient tokens as it searches for the optimal tree. This is similar to the "deletion" strategy employed by Zettlemoyer and Collins (2007) in which words can be skipped.

Baseline

As a baseline, we trained our model with uniform prior mean distributions ($\sigma^0, \theta^0, \pi^0, \mu^0_{\mathfrak{t}}$). The uniform priors do not make any distinction among the relative likelihoods of different CCG categories or words, and thus do not take advantage of either the universal properties of the CCG formalism (P_{CAT}), or the initialization information that can be automatically estimated from the type-supervised data (P_{em}).

Results

The results of our experiments are given in Table 4.2a. We find that the use of a well-designed category prior (P_{CAT}) achieves performance gains over the baseline across all three languages. Our results also show that still further gains can be achieved by using the available weak supervision—the tag dictionary and unlabeled text—to estimate corpus counts that can be used to influence the priors on terminal productions (P_{em}).

The largest gains are in the Chinese data, though the accuracies are lower on Chinese overall, indicating the difficulty of the Chinese parsing task.

Error analysis

Supertag accuracy degrades roughly 2% for each language from the uniform prior to the full prior. Inspection of the errors shows us that this is due in part to the category prior encouraging simpler categories, e.g., categories like $((\mathsf{s}\backslash\mathsf{np})/(\mathsf{s}\backslash\mathsf{np}))/\mathsf{np}$ being learned as pp/np. It is counter-intuitive that supertagging accuracy decreases while parsing performance improves, but note that it may be easier for the parser to recover correct dependencies, using the *merge* rule, when an incorrect supertag is simpler.

The most frequent errors under uniform priors involve very complex categories, like $((\mathsf{s}_{\mathsf{dcl}}\backslash\mathsf{np})/(\mathsf{s}_{\mathsf{dcl}}\backslash\mathsf{np}))/\mathsf{np}$ in Chinese. When the category prior is introduced, these complex categories vanish from the errors; the most complex common category error with the category prior in Chinese is $(\mathsf{s}_{\mathsf{dcl}}\backslash\mathsf{np})/\mathsf{np}$. After bringing the data-based prior in, we again see more complex categories, plus others that have high arity, like modifiers of modifiers $(\mathsf{np}/\mathsf{np})/(\mathsf{np}/\mathsf{np})$. This suggests that good performance relies on priors that blend theoretical constraints with empirical guidance.

We also trained the parser on gold-standard trees for an upper-bound on performance for the given training sentences, obtaining 66%, 48%, and 65% for English, Chinese, and Italian, respectively.

Supertag dictionary pruning

Tag dictionaries used in experimental setups are typically extracted from labeled corpora by finding all word/tag pairs in some set of annotated sentences, as we saw in Chapter 3. As dictionaries, however, the distinctions between high- and low-frequency tags are lost, and all tags in the dictionary entry appear equally valid for a given word. Unfortunately, the inclusion of low-probability tags in this way tends to cause problems during training by over-representing the likelihoods of tags that are only rarely applicable, or even tags that are the result of annotation errors and should not have been included in the dictionary at all.

Traditionally, researchers have avoided this problem by using tag frequency information to automatically *prune* the tag dictionary of its low-frequency tags (Merialdo, 1994; Kupiec, 1992), leading Banko and Moore (2004), among others, to argue that early successes in type-supervised learning were due, in large part, to the use of that frequency information that is not available from unlabeled data alone, undercutting the promises of weakly-supervised learning.

Since it is the goal of this research to develop techniques that can be applied without artificial data cleaning, we desire models that are robust to noise in the training data. To see how this noise affects our model, we executed a series of experiments in which varying degrees of noise were artificially removed. For different cutoff levels (0.001, 0.01, 0.1), we computed the tag dictionary entry for word w, as the supertags $\mathbf{t}$ where:

$$\textsc{td}(w) = \left\{ \mathbf{t} \;\middle|\; \frac{\mathit{freq}(w,\mathbf{t})}{\sum_{\mathbf{t}' \in \mathcal{T}} \mathit{freq}(w,\mathbf{t}')} \geq \mathit{cutoff} \right\}$$

Results under pruned conditions are given in Table 4.2b. Table 4.2a can be interpreted as results when $\mathit{cutoff} = 0$.

From the results, we can see that in most scenarios, grammar-informed priors still provide benefits to the model. More notable, however, is that these priors provide *more* value in cases where there is less artificial pruning. This tells us that our constructed priors are most helpful in the noisier, more difficult, and *more realistic* learning scenarios.

When only uniform priors are used, the model is not able to differentiate *a priori* between probable and improbable categories. This results in poor performance when artificial assistance is not given. However, our category prior, with its knowledge of the intrinsic properties of the CCG formalism, is able to overcome this problem, allowing the model to differentiate between likely and unlikely categories and biasing the model toward better categories even though category frequency information is not available. Importantly, it also does this without eliminating tags that (though infrequent) are useful for parsing.

These results support our hypothesis that when supervised data is scarce, it becomes more important to take advantage of linguistic knowledge.

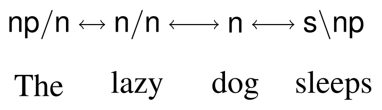
4.2.4 Conclusions

We have presented a Bayesian approach to CCG parser parameter estimation that can be trained given only a lexicon and raw text. It flexibly incorporates linguistically-informed prior distributions and naturally accommodates the deviations from pure CCG grammars that have been employed in annotations for existing CCG corpora, especially CCGBank. The model enhances a standard PCFG by factoring in prior distributions over categories into both non-terminal and terminal productions; those priors can be derived from a universal prior distribution, from a distribution built by combining a tag dictionary with raw text, or both. Our results show that using both sources for defining these priors leads to better performing CCG parsers in low-resource scenarios.

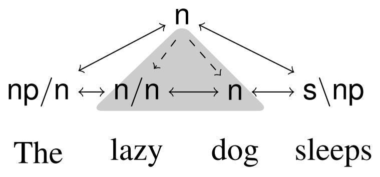
In the next section, we present a novel model that is able to capture priors not just on CCG's preference for simple categories, but al CCG's preference for adjacent categories that are combinable under some CCG rule.

4.3 Supertag-Context Parsing

In §4.2, we introduced a CCG parsing model with priors that encourage the use of categories throughout the tree that are cross-linguistically more plausible. In doing so, we were able to incorporate one of the two principles that were the



(a) An HMM supertagger uses a strong prior on combinability between adjacent supertags.



(b) Higher-level category n subsumes the categories of its constituents. Thus, n should have a strong prior on combinability with its adjacent supertags np/n and s\np.

Figure 4.1: An example of how the prior on connectivity extends to relationships between higher-level non-terminal categories and their neighboring supertags.

basis for the supertagger presented in Chapter 3: that some CCG categories are intrinsically more likely than others. However, the other principle, that, given the natural associativity of CCG, adjacent categories are likely to be combinable under some CCG rule, does not have a straightforward analog in the PCFG model. The reason for this is that the PCFG generates categories from parent categories, not siblings, and thus does not have probability distributions governing the relationships of adjacent tags for which we might be able to develop linguistically-informed priors. Note that while binary productions do model a subset of the adjacent category pairs, these tags are, according to the grammar, *required* to be combinable by some CCG rule, rendering the concept of a prior here moot. Given that our supertagging experiments showed strong performance gains from these inclusion of these priors (§3.4), it seems desirable that we would be able to incorporate these ideas into our parsing model.

In this chapter, we introduce a novel parsing model that adds *context productions* to the PCFG model from §4.2 in order to capture this adjacent-category associativity property of CCG (§4.3.1). The intuition behind this enhancement is that since adjacent categories combine with each other to form new categories that describe larger constituents, if we believe *a priori* that adjacent supertags are likely to combine, then it should follow that the categories of these higher-level constituents are likely to combine with *their* adjacent supertags as well.

As an example, consider the simple sentence “The lazy dog sleeps”, as shown in Figure 4.1. As with the simple supertagging case, we want our prior to bias toward connections between each pair of words: The↔lazy↔dog↔sleeps. Here we can see that the supertags of adjacent words *lazy* and *dog* can combine using forward application to produce the category n, which describes the entire constituent span “lazy dog”. Since we have produced a new category that subsumes that entire span, a valid parse must next combine that n with one of the remaining supertags, producing either (The·(lazy·dog))·sleeps or The·((lazy·dog)·sleeps). Because we know that one (or both) of these combinations must be valid, we will similarly want a strong prior on the connectivity between lazy·dog and its supertag context: The↔(lazy·dog)↔sleeps.

This supertag-context approach has similarities to the constituent context model (CCM) of Klein and Manning (2002), which captures part-of-speech tag context information surrounding a grammatical constituent. With this model, they were able to show that context information can be beneficial for parser learning. However, unlike CCM, we do not assume a fixed set of unambiguous pre-terminal-level categories to serve as context anchors; our model must deal with highly ambiguous lexicons in which any given token may be associated with as many as 1,300 possible labels, meaning that there is a very high number of potential context categories for every constituent.

The addition of these context parameters allows us to provide priors that bias the model toward adjacent categories throughout the tree that combine with their contexts, just as we were able to bias our trees to choose cross-linguistically likely categories at higher levels, thus extending the full set of intuitions from Chapter 3 to a parsing model. In order to estimate the parameters of our model, we also develop a blocked, Metropolis-Hastings sampler to efficiently sample parse trees for sentences in the training corpus according to their posterior probabilities (§4.3.2).

4.3.1 Generative Model

As shown in Chapter 3, it is possible to formulate effective priors on sequences of CCG supertags based on their internal structure and the intuition that they should be simple, in conjunction with the tendency for adjacent categories to

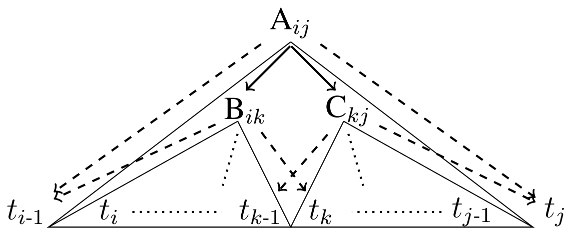


Figure 4.2: The generative process starting with non-terminal A_{ij}, where t_x is the supertag for w_x, the word at position x, and "A $\rightarrow$ B C" is a valid production in the grammar. We can see that non-terminal A_{ij} generates nonterminals B_{ik} and C_{kj} (solid arrow) as well as generating left context $t_{i\text{-}1}$ and right context t_j (dashed arrows); likewise for B_{ik} and C_{kj}. The triangle under a non-terminal indicates the complete subtree rooted by the node.

be combinable. In §4.2, we showed that the prior on simple categories can applied throughout the tree to bias all constituent labels toward simplicity. Here, we develop a model that continues in this vein by extending the bias toward adjacent combinability upward to find derivation trees to capture *a priori* expected regularities between constituent labels and the (pre-terminal) contexts in which they occur.

To capture combinability beliefs, we recognize that connectivity between a constituent root and its surroundings is similar to the connectivity bias transitions in an HMM supertagger since the same combinatory rules apply uniformly. When adjacent categories do combine with each other, they form a new category that represents the merged constituents. To complete the a CCG derivation, the new category must then combine with one of its adjacent constituent categories. Therefore, just as we expect supertags to be combinable with their adjacent supertags, we also expect higher-level constituent categories to be combinable with adjacent supertags. While these categories will not necessarily need to actually combine with the adjacent supertags since they may instead combine with higher-level derived categories, we still expect them to be combinable in the same was as was observed by Baldridge, given the natural associativity of CCG.

In order to take advantage of this behavior in a CCG parse, we propose a novel model that extends a standard PCFG to consider the relationship between each non-terminal category and the supertags directly to the left and right of the constituent's span. The generative story of our model is shown in Figure 4.3.[4]

The generative process begins by sampling the parameters: a distribution σ over root categories, a conditional distribution $\theta_{\mathbf{t}}$ over non-terminal binary branching productions given category $\mathbf{t}$, a conditional distribution $\pi_{\mathbf{t}}$ over non-terminal unary branching productions given category $\mathbf{t}$, a conditional distribution $\mu_{\mathbf{t}}$ over terminal (word) productions given category $\mathbf{t}$, a conditional distribution $\psi_{\mathbf{t}}$ over left contexts given category $\mathbf{t}$, and a conditional distribution $\xi_{\mathbf{t}}$ over right contexts given category $\mathbf{t}$. We also sample a parameter $\lambda_{\mathbf{t}}$ for every category $\mathbf{t}$ that the probabilities of $\mathbf{t}$ producing a binary branch, unary rewrite, or terminal word. Next we sample a sentence. This begins by sampling first a root category $\mathbf{s}$ and then recursively sampling subtrees. For each subtree rooted by a category $\mathbf{t}$, we generate a left context supertag $\boldsymbol{\ell}$ and a right context supertag $\mathbf{r}$. Then, we sample a production type z corresponding to either a (1) binary, (2) unary, or (3) terminal production. Depending on z, we then sample either a binary production $\langle \mathbf{u}, \mathbf{v} \rangle$ and recurse, a unary production $\langle \mathbf{u} \rangle$ and recurse, or a terminal word w and end that branch. A tree is complete when all branches end in terminal words. See Figure 4.2 for a graphical depiction of the generative behavior of the process. Finally, since it is possible to generate a supertag context category that does not match the actual category generated by the neighboring constituent, we must allow our process to reject such invalid trees and re-attempt to sample.

One additional complication that must be addressed is that left-frontier non-terminal categories — those whose subtree span includes the first word of the sentence — do not have a left-side supertag to use as context. For these cases, we invent a special, unique sentence-start symbol $\langle \text{S} \rangle$ to serve as context. Similarly, we invent a sentence-end symbol $\langle \text{E} \rangle$ for the right-side context of right-frontier non-terminals. These correspond directly to the start and end tags used in the supertagger, and their "combinability" is treated the same way as well (§3.2.3).

This model can be understood as a context-aware variant of a Bayesian PCFG

[4]Note that this model is *deficient* in the sense that the same supertags are generated multiple times, and sentences with conflicting supertags are not possible. As demonstrated by Klein and Manning (2002), this fact does not affect the correctness of the model. (Though Smith and Eisner (2004) showed that correcting deficiency might be a good idea.)

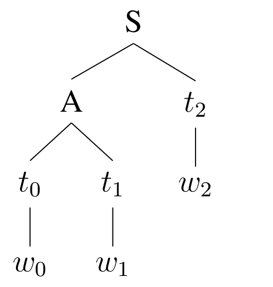
Parameters:

$\sigma \sim \text{Dirichlet}(\alpha_\sigma, \sigma^0)$		root categories
$\theta_\mathbf{t} \sim \text{Dirichlet}(\alpha_\theta, \theta^0)$	$\forall \mathbf{t} \in \mathcal{T}$	binary productions
$\pi_\mathbf{t} \sim \text{Dirichlet}(\alpha_\pi, \pi^0)$	$\forall \mathbf{t} \in \mathcal{T}$	unary productions
$\mu_\mathbf{t} \sim \text{Dirichlet}(\alpha_\mu, \mu^0_\mathbf{t})$	$\forall \mathbf{t} \in \mathcal{T}$	terminal productions
$\lambda_\mathbf{t} \sim \text{Dirichlet}(\alpha_\lambda, \langle \lambda^0(1), \lambda^0(2), \lambda^0(3) \rangle)$	$\forall \mathbf{t} \in \mathcal{T}$	production type mixture
$\psi_\mathbf{t} \sim \text{Dirichlet}(\alpha_\psi, \psi^0_\mathbf{t})$	$\forall \mathbf{t} \in \mathcal{T}$	left context productions
$\xi_\mathbf{t} \sim \text{Dirichlet}(\alpha_\xi, \xi^0_\mathbf{t})$	$\forall \mathbf{t} \in \mathcal{T}$	right context productions

Sentence:

```
  **do**
    $\mathbf{s} \sim \text{Categorical}(\sigma)$
    *generate*$(\mathbf{s})$
  **until** *the generated tree is valid*
where
  **function** *generate*$(\mathbf{t})$ :
    $\boldsymbol{\ell} \mid \mathbf{t} \sim \text{Categorical}(\psi_\mathbf{t})$
    $\mathbf{r} \mid \mathbf{t} \sim \text{Categorical}(\xi_\mathbf{t})$
    $z \sim \text{Categorical}(\lambda_\mathbf{t})$
    **if** $z = 1 : \langle \mathbf{u}, \mathbf{v} \rangle \mid \mathbf{t} \sim \text{Categorical}(\theta_\mathbf{t})$
              $\text{BinaryTree}(\mathbf{t}, \boldsymbol{\ell}, \textit{generate}(\mathbf{u}), \textit{generate}(\mathbf{v}), \mathbf{r})$
    **if** $z = 2 : \langle \mathbf{u} \rangle \mid \mathbf{t} \sim \text{Categorical}(\pi_\mathbf{t})$
              $\text{UnaryTree}(\mathbf{t}, \boldsymbol{\ell}, \textit{generate}(\mathbf{u})), \mathbf{r})$
    **if** $z = 3 : w \mid \mathbf{t} \sim \text{Categorical}(\mu_\mathbf{t})$
              $\text{Leaf}(\mathbf{t}, \boldsymbol{\ell}, w, \mathbf{r})$
```

Figure 4.3: The generative story of the supertag-context parsing model.


(a) A standard PCFG tree.

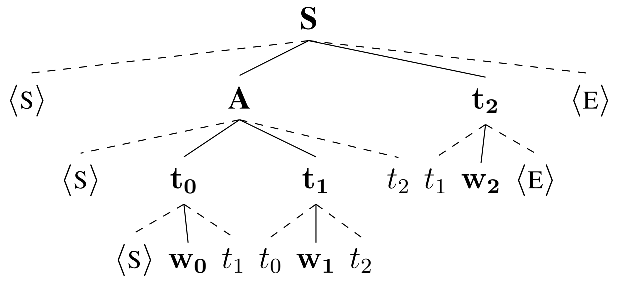

(b) The corresponding supertag-context tree.

Figure 4.4: Expansion of a PCFG tree to include left and right contexts.

(Johnson et al., 2007). We next discuss how the prior distributions are constructed to encode desirable biases, using universal CCG properties.

PCFG parameter prior means ($\sigma^0, \theta^0, \pi^0, \mu^0_{\mathbf{t}}, \lambda^0$)

For the PCFG parameters, we set the prior means exactly as we did previously in §4.2.1, capturing the desires to find cross-linguistically-plausible categories as well as the terminal/emission priors initialized by analysis of the tag dictionary and raw corpus:

$$\begin{aligned}
\sigma^0(\mathbf{t}) &= P_{\text{CAT}}(\mathbf{t}) && \text{for categories } \mathbf{t} \\
\theta^0(\langle \mathbf{u}, \mathbf{v} \rangle) &= P_{\text{CAT}}(\mathbf{u}) \cdot P_{\text{CAT}}(\mathbf{v}) && \text{for category pairs } \langle \mathbf{u}, \mathbf{v} \rangle \\
\pi^0(\langle \mathbf{u} \rangle) &= P_{\text{CAT}}(\mathbf{u}) && \text{for categories } \langle \mathbf{u} \rangle \\
\mu^0_{\mathbf{t}}(w) &= P_{em}(w \mid \mathbf{t}) && \text{for terminal productions ``}\mathbf{t} \rightarrow w\text{''} \\
\lambda^0 &= \langle \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3} \rangle
\end{aligned}$$

The root, binary, and unary productions are specified using the category prior P_{CAT} defined in §3.2.2, while the terminal production prior means $P_{em}(w \mid \mathbf{t})$ are set using the procedure from §3.2.4.

Context parameter prior means ($\psi^0_{\mathbf{t}}$ and $\xi^0_{\mathbf{t}}$)

In order to encourage our model to choose trees in which the constituent labels "fit" into their supertag contexts, we want to bias our context parameters toward contexts categories that *combine* with the constituent label. The right-side context of a non-terminal category — the probability of generating a category to the right of the current constituent's category — corresponds directly to the category transitions used for the HMM supertagger, as seen in §3.2.1. Thus, the right-side context prior mean $\xi^0_{\mathbf{t}}$ can use the same notions of combinability as the supertagger to similarly bias our model toward context supertags that connect to the constituent category.

The prior mean of producing a right-context supertag $\mathbf{r}$ from a constituent category $\mathbf{t}$, $P^{right}(\mathbf{r} \mid \mathbf{t})$, is defined so that combinable pairs are given a higher probability than non-combinable pairs. We further experimented with a prior that biases toward both combinability *and* category likelihood by replacing the uniform treatment of categories with our prior over categories, yielding $P^{right}_{\text{CAT}}(\mathbf{r} \mid \mathbf{t})$. If $\mathcal{T}$ is the full set of known CCG categories:

$$P^{right}(\mathbf{r} \mid \mathbf{t}) = \begin{cases} a \cdot 1/|\mathcal{T}| & \kappa(\mathbf{t}, \mathbf{r}) \\ 1/|\mathcal{T}| & \text{otherwise} \end{cases}$$

$$P^{right}_{\text{CAT}}(\mathbf{r} \mid \mathbf{t}) = \begin{cases} a \cdot P_{\text{CAT}}(\mathbf{r}) & \text{if } \kappa(\mathbf{t}, \mathbf{r}) \\ P_{\text{CAT}}(\mathbf{r}) & \text{otherwise} \end{cases}$$

where $a > 1$.

The left-side context represents the equivalent of a "backward" transition. Distributions $P^{left}(\boldsymbol{\ell} \mid \mathbf{t})$ and $P^{left}_{\text{CAT}}(\boldsymbol{\ell} \mid \mathbf{t})$ are defined in the same way, but with the combinability direction flipped, $\kappa(\boldsymbol{\ell}, \mathbf{t})$, since the left context supertag precedes the constituent category.

While the design of the left-context prior can, in principle, be set differently from the right-context, we leave them identical (though reversed) for our experiments.

4.3.2 Posterior Inference

We wish to find the most likely CCG parse, given the model we just described, for each sentence in a corpus of training data. Since there is no way to analytically compute these modes, we resort to Gibbs sampling to find an approximate solution. Our strategy is based on the approach taken by Johnson et al. (2007). At a high level, we alternate between resampling model parameters $(\sigma, \theta, \pi, \mu, \lambda, \psi, \xi)$ given the current set of parse trees and resampling those trees given the current model parameters and observed word sequences. To efficiently sample new model parameters, we exploit Dirichlet-multinomial conjugacy. By repeating these alternating steps and accumulating the number of times each non-terminal category is used in each position, we obtain an approximation of the required posterior quantities.

Our inference procedure takes as input each of the distribution prior mean $(\sigma^0, \theta^0, \pi^0, \mu^0, \psi^0, \xi^0)$, along with the raw corpus and tag dictionary. We refer to the set of supertags associated with a word w as $\text{TD}(w)$. During sampling, we always restrict the possible tag choices for a word w to the categories found in $\text{TD}(w)$. We refer to the sequence of word tokens as **w** and a non-terminal category covering the span i through $j-1$ as y_{ij}.

While it is technically possible to sample directly from our context-sensitive model, the high number of potential supertags available for both contexts means that computing the inside chart for this model is intractable for most sentences. In order to overcome this limitation, we employ an accept/reject Metropolis-Hastings (MH) step, the basic idea being that we sample parse trees according to a simpler *proposal* distribution that approximates the full distribution and for which direct sampling is tractable (in our case, the one defined by the PCFG parameters $(\sigma, \theta, \pi, \mu, \lambda)$), and then choose to accept or reject those trees based on the true distribution (defined by the full set of parameters).

If we refer to the proposal (PCFG-only) distribution as Q, and the true distribution as P. In our case, the true distribution defines the probability of a tree **y** given the full set of model parameters $(\sigma, \theta, \pi, \mu, \lambda, \psi, \xi)$, while the proposal distribution defines the probability of **y** given only the PCFG parameters $(\sigma, \theta, \pi, \mu, \lambda)$. The *acceptance* distribution $A(\mathbf{y}' \mid \mathbf{y})$ gives the probability of accepting a new tree $\mathbf{y}'$ given that the previous tree was $\mathbf{y}$, and is defined standardly as follows:

$$A(\mathbf{y}' \mid \mathbf{y}) = \min\left(1, \frac{P(\mathbf{y}')}{P(\mathbf{y})}\frac{Q(\mathbf{y})}{Q(\mathbf{y}')}\right)$$

When the parameters of the relevant distributions are spelled out, and the denominators reversed, we have:

$$A(\mathbf{y}' \mid \mathbf{y}) = \min\left(1, \frac{p(\mathbf{w}, \mathbf{y}' \mid \sigma, \theta, \pi, \mu, \lambda, \psi, \xi)}{p(\mathbf{w}, \mathbf{y}' \mid \sigma, \theta, \pi, \mu, \lambda)} \cdot \frac{p(\mathbf{w}, \mathbf{y} \mid \sigma, \theta, \pi, \mu, \lambda)}{p(\mathbf{w}, \mathbf{y} \mid \sigma, \theta, \pi, \mu, \lambda, \psi, \xi)}\right)$$

At this point, it should be clear given the way the two models are defined—the new model in §4.3.1 and the PCFG model in §4.2.1—that the PCFG model is a subset of the new model, and that the probability of a tree **y** under this model is equivalent to the product of the probability of **y** under the PCFG and the probability of **y** given only the context parameters:

$$p(\mathbf{w}, \mathbf{y} \mid \sigma, \theta, \pi, \mu, \lambda, \psi, \xi) = p(\mathbf{w}, \mathbf{y} \mid \sigma, \theta, \pi, \mu, \lambda) \cdot p(\mathbf{w}, \mathbf{y} \mid \psi, \xi)$$

Thus, we have:

$$A(\mathbf{y}' \mid \mathbf{y}) = \min\left(1, \frac{\cancel{P(\mathbf{y}')} \cdot p(\mathbf{w}, \mathbf{y}' \mid \psi, \xi)}{\cancel{P(\mathbf{y}')}} \cdot \frac{\cancel{P(\mathbf{y})}}{\cancel{P(\mathbf{y})} \cdot p(\mathbf{w}, \mathbf{y} \mid \psi, \xi)}\right)$$

where $p(\mathbf{w}, \mathbf{y} \mid \psi, \xi)$ is defined by:[5]

$$p(\mathbf{w}, \mathbf{y} \mid \psi, \xi) = \prod_{0 \le i < j \le n} \psi(y_{i-1,i} \mid y_{ij}) \cdot \xi(y_{j,j+1} \mid y_{ij})$$

Finally, the PCFG parameters cancel in the equation, leaving us with an acceptance

[5]Note that there may actually be multiple y_{ij} due to unary rules that "loop back" to the same cell; all of these must be included in the product.

distribution defined purely in terms of context parameters:

$$A(\mathbf{y}' \mid \mathbf{y}) = \min\left(1, \frac{p(\mathbf{w}, \mathbf{y}' \mid \psi, \xi)}{p(\mathbf{w}, \mathbf{y} \mid \psi, \xi)}\right)$$

So in our MH sampler, we draw trees from the PCFG distribution (an operation we know can be done efficiently, as it was in §4.2.2), and then decide whether to accept or reject them based on the contexts.

We initialize the sampler by setting each distribution to its prior mean ($\sigma = \sigma^0$, $\theta_{\mathbf{t}} = \theta^0$, etc). We then parse the raw corpus (with probabilistic CKY) using the initial PCFG parameters to get a full bank of trees for use in the first MH step.

The sampler alternates sampling parse trees for the entire corpus of sentences using the procedure from §4.2.2, then resampling the parameters, then trees again, and so on. Re-sampling the parameters uses the just-sampled parse trees **y** to compute $C_{\mathit{root}}(\mathbf{t})$, the count of trees in which the root category is **t**, $C(\mathbf{t} \rightarrow \langle \mathbf{u}, \mathbf{v} \rangle)$, the count of binary productions whose category is **t** that are producing the pair of categories $\langle \mathbf{u}, \mathbf{v} \rangle$, $C(\mathbf{t} \rightarrow \langle \mathbf{u} \rangle)$, the count of unary productions whose category is **t** that are producing categories $\langle \mathbf{u} \rangle$, $C(\mathbf{t} \rightarrow w)$, the count of terminal productions whose category is **t** that are producing word w, $C_{\mathit{right}}(\mathbf{t}, \mathbf{r})$, the count of categories **t** that have right context supertag **r**, and $C_{\mathit{left}}(\mathbf{t}, \boldsymbol{\ell})$, the count of categories **t** that have left context supertag $\boldsymbol{\ell}$. These counts are taken from the trees resulting from the previous round of sampling: new trees that have been "accepted" by the MH step, as well as existing trees for sentences in which the newly-sampled tree was rejected. We then sample, for each $\mathbf{t} \in \mathcal{T}$ where $\mathcal{T}$ is the full set of valid CCG categories:

$$
\begin{aligned}
\sigma &\sim \text{Dirichlet}(\langle \alpha_\sigma \cdot \sigma^0(\mathbf{t}) + C_{\textit{root}}(\mathbf{t}) \,\rangle_{\mathbf{t}\in\mathcal{T}}) \\
\theta_{\mathbf{t}} &\sim \text{Dirichlet}(\langle \alpha_\theta \cdot \theta^0(\langle \mathbf{u},\mathbf{v}\rangle) + C(\mathbf{t}\to\langle \mathbf{u},\mathbf{v}\rangle) \,\rangle_{\mathbf{u},\mathbf{v}\in\mathcal{T}}) \\
\pi_{\mathbf{t}} &\sim \text{Dirichlet}(\langle \alpha_\pi \cdot \pi^0(\langle \mathbf{u}\rangle) + C(\mathbf{t}\to\langle \mathbf{u}\rangle) \,\rangle_{\mathbf{u}\in\mathcal{T}}) \\
\mu_{\mathbf{t}} &\sim \text{Dirichlet}(\langle \alpha_\mu \cdot \mu^0_{\mathbf{t}}(w) + C(\mathbf{t}\to w) \,\rangle_{w\in V}) \\
\lambda_{\mathbf{t}} &\sim \text{Dirichlet}(\langle \alpha_\lambda \cdot \lambda^0(1) + \textstyle\sum_{\mathbf{u},\mathbf{v}\in\mathcal{T}} C(\mathbf{t}\to\langle \mathbf{u},\mathbf{v}\rangle), \\
&\qquad\qquad\qquad \alpha_\lambda \cdot \lambda^0(2) + \textstyle\sum_{\mathbf{u}\in\mathcal{T}} C(\mathbf{t}\to\langle \mathbf{u}\rangle), \\
&\qquad\qquad\qquad \alpha_\lambda \cdot \lambda^0(3) + \textstyle\sum_{w\in V} C(\mathbf{t}\to w) \qquad \rangle) \\
\psi_{\mathbf{t}} &\sim \text{Dirichlet}(\langle \alpha_\psi \cdot \psi^0_{\mathbf{t}}(\boldsymbol{\ell}) + C_{\textit{left}}(\mathbf{t},\boldsymbol{\ell})\rangle_{\boldsymbol{\ell}\in\mathcal{T}}) \\
\xi_{\mathbf{t}} &\sim \text{Dirichlet}(\langle \alpha_\xi \cdot \xi^0_{\mathbf{t}}(\mathbf{r}) + C_{\textit{right}}(\mathbf{t},\mathbf{r})\rangle_{\mathbf{r}\in\mathcal{T}})
\end{aligned}
$$

It is important to note that this method of resampling allows the draws to incorporate both the data, in the form of counts, and the prior mean, which includes all of our carefully-constructed biases derived from both the intrinsic, universal CCG properties as well as the information we induced from the raw corpus and tag dictionary.

After all sampling iterations have completed, the final model is estimated by taking the maximum likelihood estimate of the average over all trees resulting from each sampling iteration, including trees accepted by the MH steps as well as the duplicated trees retained due to rejections.

4.3.3 Experiments

In our evaluation we compared our supertag-context approach to the simpler parsing model presented in §4.2. We evaluated using the same setup. We used English, Chinese, and Italian corpora using the same data splits.In order to increase the amount of raw data available to the sampler, we supplemented the English data with raw, unannotated newswire sentences from the NYT Gigaword 5 corpus (Parker et al., 2011) and supplemented Italian with the out-of-domain WaCky corpus (Baroni et al., 1999). For English and Italian, this allows us to use 100k raw tokens for training (Chinese uses 62k). For Chinese and Italian, for training efficiency, we used only raw sentences that were 50 words or fewer (note that we did *not* drop tag dictionary set or test set sentences).

For each language, we executed four experiments. The no-context baseline used the best model from §4.2, using only the PCFG parameters $(\sigma, \theta, \pi, \mu, \lambda)$ along with the category prior P_{CAT} to bias toward more likely categories throughout the tree, and the terminal production prior P_{em} estimated from the tag dictionary and raw corpus. We then evaluated our new supertag-context model, but used uniform priors for the context parameters (ψ, ξ). Then, we evaluated the supertag-context model using context parameter priors that bias toward categories that combine with their contexts (see §4.3.1). Finally, we evaluated the supertag-context model using context parameter priors that bias toward combinability *and* bias toward *a priori* more likely categories, based on the category grammar.

The development sets were used to tune hyperparameters using grid search. We used the same hyperparameters across all three languages. For the category grammar, we used $p_{\textit{punc}}$=0.1, $p_{\textit{term}}$=0.7, $p_{\textit{mod}}$=0.2, $p_{\textit{fwd}}$=0.5. For the priors, we use α_σ=1, α_θ=100, α_π=100, α_μ=10^4, α_λ=3, α_ψ=α_ξ=10^3.[6] For the context prior, we used a=10^5. We ran our sampler for 50 burn-in and 50 sampling iterations.[7]

For each language and level of supervision, we executed four experiments. The no-context baseline used (a reimplementation of) the best model from §4.2: using only the non-context parameters $(\sigma, \theta, \pi, \mu, \lambda)$ along with the category prior P_{CAT} to bias toward more likely categories throughout the tree, and the terminal production prior $\mu^0_{\mathbf{t}}(w)$ estimated from the tag dictionary and raw corpus. We then added the supertag-context parameters (ψ, ξ), but used uniform priors for those (still using P_{CAT} for the rest). Then, we evaluated the supertag-context model using context parameter priors that bias toward categories that combine with their contexts: $P^{\textit{left}}$ and $P^{\textit{right}}$ (see §4.3.1). Finally, we evaluated the supertag-context model using context parameter priors that bias toward combinability *and* bias toward *a priori*

[6]In order to ensure that these concentration parameters, while high, were not dominating the posterior distributions, we ran experiments in which they were set much higher (including using the prior alone), and found that accuracies plummeted in those cases, demonstrating that there is a good balance with the prior.

[7]We experimented with higher numbers of iterations but found that accuracy was not improved past 50 iterations.

TD corpus size (tokens):		**250k**	**200k**	**150k**	**100k**	**50k**	**25k**
EN	no context	60.43	61.22	59.69	58.61	56.26	54.70
	context (uniform)	64.02	**63.89**	62.58	61.80	59.44	57.08
	$+P^{left}$ / P^{right}	**65.44**	63.26	**64.28**	**62.90**	**59.63**	**57.86**
	$+P^{left}_{\text{CAT}}$ / P^{right}_{CAT}	59.34	59.89	59.32	58.47	57.85	55.77
CH	no context				32.70	32.07	28.99
	context (uniform)				**36.02**	33.84	32.55
	$+P^{left}$ / P^{right}				35.34	33.04	31.48
	$+P^{left}_{\text{CAT}}$ / P^{right}_{CAT}				35.15	**34.04**	**33.53**
IT	no context						51.54
	context (uniform)						**53.57**
	$+P^{left}$ / P^{right}						52.54
	$+P^{left}_{\text{CAT}}$ / P^{right}_{CAT}						53.29

Table 4.3: Experimental results in three languages, English (EN), Chinese (CH), and Italian (IT), under varying amounts of *tag dictionary* input data. Scores are dependency accuracies: percentage of words labeled with the correct parent word (or "root"). For each language, four experiments were executed: (1) a no-context model baseline, directly from §4.2; (2) our supertag-context model, with uniform priors on context parameters; (3) supertag-context model with priors that prefer combinability; (4) supertag-context model with priors that prefer combinability *and* simpler categories. Results are shown for six different levels of supervision, as determined by the size of the corpus used to extract a tag dictionary.

more likely categories, based on the category grammar ($P^{\textit{left}}_{\text{CAT}}$ and $P^{\textit{right}}_{\text{CAT}}$).

Because we are interested in understanding how our models perform under varying amounts of supervision, we executed sequences of experiments in which we reduced the size of the corpus from which the tag dictionary is drawn, thus reducing the amount of information provided to the model. As this information is reduced, so is the size of the full inventory of known CCG categories that can be used as supertags. Additionally, a smaller tag dictionary means that there will be vastly more unknown words; since our model must assume that these words may take any supertag from the full set of known labels, the model must contend with a greatly increased level of ambiguity.

The results of our experiments are given in Table 4.3. We find that the incorporation of supertag-context parameters into a CCG model improves performance in every scenario we tested; we see gains between 2–5% across the board. Adding context parameters never hurts, and in most cases, using well-constructed priors based on intrinsic, cross-lingual aspects of the CCG formalism to bias those parameters provides further gains. In particular, biasing the model toward trees in which constituent labels are combinable with their adjacent supertags frequently helps the model.

However, for English, we found that additionally biasing context priors toward simpler categories using P_{CAT} degraded performance. This is likely due to the fact that the priors on production parameters (σ, θ, π) are already biasing the model toward likely categories, and that having the context parameters do the same ends up over-emphasizing the need for simple categories, preventing the model from choosing more complex categories when they are needed. On the other hand, this bias seemed to help in Chinese and Italian.

4.3.4 Conclusions

Because of the structured nature of CCG categories and the logical framework in which they must assemble to form valid parse trees, the CCG formalism offers multiple opportunities to bias model learning based on universal, intrinsic properties of the grammar. In Chapter 3 we showed that good CCG supertaggers can be learned from weak supervision by constructing Bayesian priors that encourage the use of simpler supertags, as well as supertags that connect with their surrounding tags via some CCG rule. In §4.2 we designed a parsing model that biases toward simpler categories throughout the tree. In this chapter, we have presented a novel parsing model that is able to incorporate all of these ideas. By augmenting our previous parsing model with supertag-context information, we are able to define priors that bias our model toward trees in which categories logically "fit" into their contexts.

4.4 Learning with Constituent Constraints

Learning natural language parsers from full supervision is one of the better-understood and developed NLP tasks (Smith, 2011). However, we still have much to learn about which types of annotations are most useful in low-data situations. Creating large-scale treebanks is prohibitively expensive for most languages and domains, so it is important to explore alternative forms of weaker supervision. Therefore, understanding how different forms of supervision can be used, how effective they are, and how difficult they are to obtain are all integral when preparing to train a parser on a new language. Furthermore, different forms of supervision may be *complementary*, providing multiple views of the data in ways that benefit each other, and understanding the trade-offs among sources of supervision may lead to lower amounts of total annotation effort.

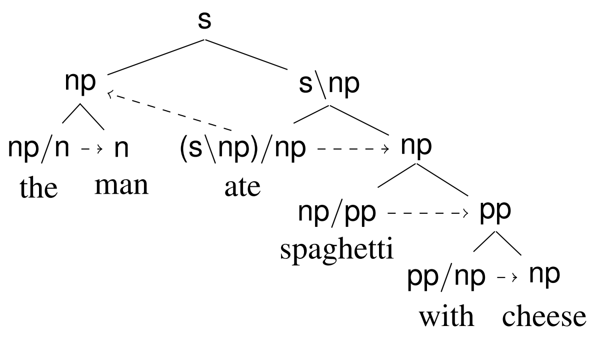
Human annotation is one potentially valuable source of supervision, but choices related to the ways in which human knowledge is collected can be complex. An annotation task is ideally straightforward for the annotator, efficiently performed, and produces high value for the end task. Instead of focusing on simply reducing the amount of fully-annotated data (Hwa, 2000; Steedman et al., 2003) or using a simplified grammar (Bisk and Hockenmaier, 2013), we consider one of the simplest annotation types: constituent bracketings, an idea proposed by Pereira and Schabes (1992). Unlike annotating full parse trees, bracketing can be done quickly and does not require an expert linguist, meaning lower overall cost and/or higher availability of annotation.

Bracket annotations are advantageous to us because they can be trivially incorporated into our existing parsing model, presented in §4.2 (as well as the models that will be presented in §4.3 and §4.5). Moreover, those models are trained using type-level supervision in the form of (incomplete) tag dictionaries. Tag dictionary information is highly complementary to bracketing information, the former providing information on word types, and the latter providing information at the phrase or sentence level, making it an excellent fit for our work.

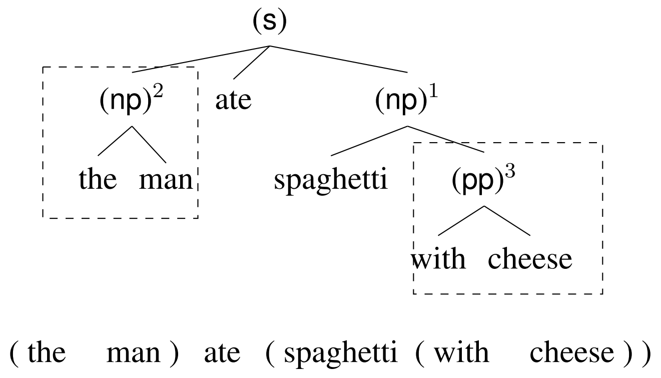
Previous work learning parsers from brackets focused on learning context-free grammar parsers from gold part-of-speech tags (Pereira and Schabes, 1992; Hwa, 1999). Here, we instead test their effectiveness in learning Combinatory Categorial Grammar (CCG) parsers from raw, untagged data (Steedman, 2000; Steedman and Baldridge, 2011). We build on the approach of §4.2 for learning CCG parsers from a partial, type-level lexicon and raw sentences. By adding human-labeled brackets, we can constrain the search space of their algorithm to focus on better analyses more quickly. We show, through annotation simulation experiments, that complete bracketings are unnecessary for good performance and that high-level brackets (e.g. for sentences) are slightly more useful than low-level brackets (e.g. for base noun phrases). Finally, we find that 400 human-annotated sentences will improve parser performance on unlabeled dependencies from 57.8% to 61.6%. The use of the semi-supervised CCG parser with these annotations successfully brings together three forms of supervision: linguistically-informed CCG-based priors, incomplete categorial lexicons (type-level annotation), and partial bracketings (token-level annotation).

4.4.1 Bracket Annotations and CCG Parsing

In §4.2, we presented a Bayesian model for learning CCG parsers from only type-level supervision, in the form of tag dictionaries, plus raw sentences. The dictionary provides constraints on the categories that words may take, but says nothing about how individual tokens should be allowed to interact with neighboring words. Bracketings, on the other hand, provide information on token relationships, mak-



(a) Example derivation with dependencies. In the standard CCG transformation to dependency parse, each combination produces a dependency, and the argument of the combination is treated as the dependent (except when the function is a *modifier*).



(b) Hierarchical bracketing for the derivation in (a).

Figure 4.5: Example CCG derivation and corresponding dependency tree. In (b), each bracket is associated with a CCG category for the constituent that it covers. Base-level constituents are boxed; superscripts show the ordering from high-level to low-level (see details in §4.4.2).

ing the two forms of supervision complementary. Additionally, one of the main challenging of CCG parser learning is overcoming the high degree of ambiguity that arises from the lexicon. For instance, the English CCGBank contains over 1,300 lexical categories (Hockenmaier and Steedman, 2007); each token must be assigned to one of these categories, and unknown words (out-of-vocabulary types) are allowed to take any of the 1,300. Typically, the learning algorithm must deal with an extremely large space of possible trees, but bracketing annotations directly limit that space, making learning both easier and more efficient.

Because the CCG parser can be trained with no brackets, it can also use any subset of valid bracketings. This allows annotators to provide constraints to certain aspects of the tree, while leaving others underspecified. For example, they can skip difficult items and let the learning algorithm handle the complexity, or they can skip easy brackets that will likely not be very informative for the learner (but which are numerous and thus take time to annotate). Hwa (1999) explored this question in detail for a tree insertion grammar parser and found that it was more effective to label high-level constituents than low-level ones. We replicate this finding, and we furthermore find differences in the value of giving brackets for different types of constituents might be more valuable to bracket: noun phrases, or prepositional phrases, or s-type clauses. Finally, we show that brackets produced by humans lead to performance consistent with those obtained from gold trees, demonstrating that the task is indeed easy enough for humans to perform well.

4.4.2 Experiments

We use the setup from §4.2 for our experiments, starting with the best configuration and swapping in bracket-labeled sentences for the raw sentences we used before. Everything else is held fixed. The application of bracketing information is straightforward: if the annotation has a bracket covering words i through j, then there may be no constituent that includes words $i-1$ and i without j, or j and $j+1$ without i. Eliminating nodes in the parse chart is both simple and efficient: when the sampler computes the Inside table, it does not need to consider any eliminated cells.

We evaluate using data from English CCGBank, with the same data splits as §4.2. We follow the standard CCG parsing evaluation of reporting dependency accuracy based on the dependency transformation of the model's output CCG parse tree. All experiments use the same 1,000 sentences for training. Any annotated data came from those sentences, meaning that if n sentences contained annotations, then $1000 - n$ sentences would be used in their unannotated form. Using no annotated sentences is equivalent to the setup in §4.2.

Since our approach combines both bracketing and tag dictionary information as constraints, it is possible for these to conflict and make the sentence unparseable. If an annotation is given for a particular training sentence, and the system is not able to find a parse that adheres to the annotation, then the annotation is automatically rejected, and the sentence is used in raw form. This way, the addition of annotations never results in fewer sentences being used for sampling; it can only reduce the number of potential parses of those sentences. Note that we still count rejected sentences toward the number of annotated since they still take time to collect. Unuseable annotations are a reality in this endeavor, and removing them and replacing them with good annotations would artificially inflate the results and obscure the amount of effort that was actually expended.

Simulation experiments

We first explore the value of different kinds of bracket annotations by extracting different sets of brackets from gold-standard trees.

Amount of annotated data

We first varied the number of fully-bracketed sentences. Table 4.4a shows that, as expected, adding more annotated sentences improves results. We also report the number of annotated sentences that were rejected when a parse could not be found that was compatible with the tag dictionary and brackets. These numbers are low (under 9% at each level), which is reflective of the fact that the brackets, being from gold parses, should be good. However, they are not zero because the tag

sentences annotated	accuracy	anno. rejected
0	57.78	0
200	59.36	9
400	59.96	30
600	60.12	51
800	61.57	70
1000	63.11	89

(a) Accuracy as the number of gold fully-bracketed sentences is increased. Annotations were rejected, and their unannotated forms used, if no conforming parse could be found for them.

bracket coverage	accuracy	anno. rejected
0%	57.78	0
20%	59.01	22
40%	61.90	46
60%	62.04	63
80%	62.31	78
100%	63.11	89

(b) Accuracy as the number of gold bracket annotations per sentences is increased. In each case, all 1,000 sentences are annotated, but brackets are only specified for a percentage of constituents.

Table 4.4: Parser accuracy under varying annotation conditions. (Note that the first and last rows of (a) and (b) are identical since they both represent either *none* or *all* of the annotations.)

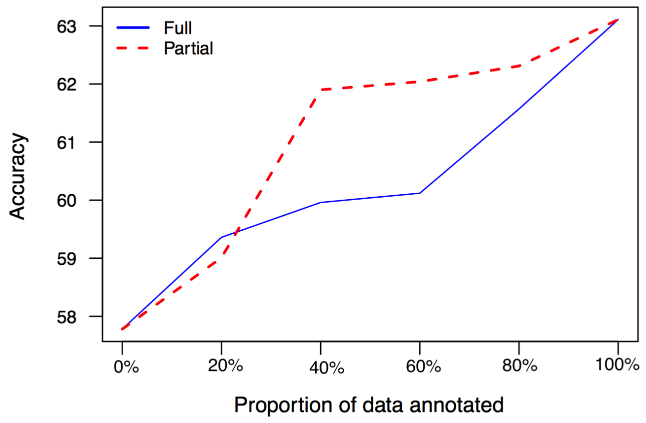


Figure 4.6: Completeness versus ease: full annotation on x% of the training sentences, or x% of the brackets across all of the training sentences. Note that producing "complete" annotations requirest considerably more effort, meaning that the success of the "partial" annotation should be considered even more striking.

bracket type	accuracy	# occ.	anno. rejected
none	57.78	—	0
s	60.65	1,738	8
np	60.90	11,349	42
pp	58.80	1,177	4

(a) Accuracy when only specific types of gold constituents (as determined by their CCG categories) are bracketed, along with their corpus frequencies.

bracket placement	accuracy	anno. rejected
none	57.78	0
base-level	60.78	32
highest-level	61.83	42

(b) Accuracy when either only gold base-level brackets are used, or the corresponding number of gold highest-level brackets.

Table 4.5: Parser accuracy under various bracket data conditions.

dictionary is pulled from a different set of sentences as the test data, and, therefore not all dictionary entries required for parsing are represented.

Annotation coverage per sentence

An appeal of lightweight bracketing is the ability to create underspecified annotations that leave many substrings unbracketed. Table 4.4b shows the effect of using only a given percentage of brackets for each sentence. Note that this strategy is generally more effective than the full-bracket strategy. E.g., it obtains 62.04% accuracy compared to 60.12% when using the same proportion of the available annotations (60%).

This shows that the sampler benefits a lot from having *some* constraint on a sentence. Despite the fact that more annotations are being rejected with partial brackets, the learner is able to fill in the remaining part of the sentence with more success than it otherwise would. This result is very encouraging because it is very difficult to provide a complete bracketing for many sentences. Since partial annotations are *more* effective, we can advise annotators to annotate what comes easily and move on, rather than struggling with thorny corner-cases.

Bracket types

We also want to understand the value of different kinds of brackets. To this end, we ran experiments in which all training data sentences are partially annotated, but only with brackets for constituents of particular CCG categories. We evaluated using only (and all of the) clause (s) brackets, only noun phrase (np) brackets, and prepositional phrase (pp) brackets.

Table 4.5a clearly shows that s-type brackets provide the highest value. They are relatively infrequent, but using them has a similar benefit as annotating ten times as many np brackets. Clauses tend to be fairly high-level, so the sampler is presumably aided by having the sentence broken into large pieces and then just needing to fill in the smaller-scale structures within those pieces (as well as connecting those larger pieces to form the full sentence). Prepositional phrases are relatively infrequent and have a relatively small benefit. It is unsurprising that np's have the highest absolute effect since they comprise more than one-third of all brackets.

High- vs. low-level annotation

We also compared performance when only the lowest-level or highest-level brackets were given as annotation. We extract the low-level brackets from the base of the gold parse tree, where a *base* node is one which has no grandchildren. The dotted boxes in Figure 4.5b are base-level brackets. Highest-level brackets are pulled from high-level splits in the tree via a right-to-left breadth-first traversal of the gold parse tree. This approximates an annotator who recursively breaks a sentence into large constituent pieces. We search right-to-left because English tends to be right-branching, so larger brackets tend to be found on the right side of the tree. The traversal ordering of the high-level bracket search can be seen in the superscripts of the nodes in Figure 4.5b.

There are vastly more higher-level nodes than base-level nodes, so using all non-base-level brackets would be an unfair comparison against only the relatively small number of base-level brackets. We thus extract exactly as many high-level brackets as there are base-level brackets for each sentence.

sentences annotated	accuracy	annotations rejected
0	57.78	0
200	59.08	15
400	61.57	52

Table 4.6: Accuracy as the number of *human-annotated* sentences is increased.

Our results show that both forms of bracketing are useful, but using the high-level brackets has a greater impact, 61.83, than base-level brackets, 60.78, a finding that is consistent with the results in Table 4.5a.

Human-annotated data

Our ultimate goal is to assess performance using human annotations. For these, we obtained annotations from NLP-knowledgeable graduate students. Each was given a description of the task and a set of ten (held-out) bracketed sentences as a reference. They were instructed to use their own judgment regarding what brackets to annotate. Because we found that having partial annotations on many sentences was more valuable than full annotation of fewer sentences, they were told to focus on brackets they were confident about and pass on anything they found tricky. They were further advised that problematic annotations might ultimately be automatically rejected, so if they were unsure about a bracket, it would be preferred to leave it out. The annotators bracketed 400 sentences at a rate of roughly fifty per hour.

Table 4.6 shows that the human annotators produced slightly more unparsable annotations than the simulated gold ones; nonetheless, they are *more* valuable for parser learning. With 200 sentences, we obtain results only slightly lower than an equivalent number of gold-extracted annotations. With 400 annotations, the parser using human annotations exceeds the performance of a gold-trained parser.

The strong performance under human-annotation is likely do to the fact that human annotators, when instructed to avoid difficult constructions, naturally tend to annotate only the most certain constituents. This means the sampler receives strong guidance on high-certainty constructs, while being free to explore various bracketings where there is more ambiguity, which is desirable when the syntax is unclear.

4.4.3 Conclusion

Like Hwa (1999), we find that brackets are a cheap and effective form of annotation. Like her, we find that high-level brackets are more valuable than low-level ones, and that partial brackets are more cost-effective than complete annotations. Importantly, this relieves annotators of the burden of providing complete annotations. We additionally show, through experiments performed with human subjects, that human-provided annotations can obtain a 4% absolute improvement with just eight hours of effort.

In order to gain a better understanding of the costs, benefits, and trade-offs of various forms of parser supervision, much more research remains to be done. Perhaps most concretely, we should investigate the trade-offs between the size of the tag dictionary and the amount of bracketing information.

A more significant effort would be to go beyond simple bracketing into additional, deeper forms of parser supervision. In particular, the Graph Fragment Language (GFL) framework makes it fast and easy for annotators to provide not only partial constituent information, but also partial dependency information (Schneider et al., 2013). Given the strong relationship between CCG and dependencies, it would be interesting to explore the complementarity of CCG type-level category information, bracketings, and dependencies. The primary advantage to the GFL scheme is that it makes it very simple for annotators to provide *partial* annotations. In the context of a low-resource learning scenario, the ability to provide annotations at a flexible degree of specificity is extremely desirable. When very little time is available for data collection, it is critical to get the most out of the amount of time that annotators may devote to these tasks.

4.5 A Future Direction: Infinite CCG Parsing

In Chapter 4, we presented a Bayesian approach to parser learning for CCG. One key characteristic of this approach is that it was *type-supervised*, meaning that it learned from a tag dictionary that maps word types to potential categories. This approach has a very clear downside: the tag dictionary will inevitably be incomplete. Incompleteness comes in two forms: words may not be associated with all the categories that they should, and there may be categories that are associated with none of the words at all. Our previous models have nevertheless restricted themselves to the lexicon of the initial tag dictionary: if word w is associated with supertags $\text{TD}(w)$, then w will never be assigned a tag outside of $\text{TD}(w)$, and no word will ever receive a label outside of the full set of known supertags $\bigcup_w \text{TD}(w)$.

To overcome this limitation, we present here a preliminary exploration of a novel Bayesian nonparametric model that allows for an infinite set of CCG categories, providing a principled way for our model grow and shrink the lexicon by "inventing" new CCG categories as needed during sampling, and to lift restrictions on word types, letting them take any category as a label. Given the nature of CCG parsing, an infinite category set provides an elegant way to allow for an infinite set of production rules, making it possible for our model to adapt to novel grammatical constructions as it explores the data.

We extend the parsing model presented in Chapter 4 by adapting the techniques of Van Gael et al. (2008)'s beam sampler for the infinite hidden Markov model (iHMM). The iHMM is a nonparametric variation of a hidden Markov model that allows for an infinite set of tags (Beal et al., 2001). As a sequence model, the iHMM is appropriate for sequence-based tasks, such as part-of-speech tagging (Van Gael et al., 2009). In this work, we present a new infinite CCG model in which we model an infinite category set for the nodes within a probabilistic context-free grammar (PCFG) tree model instead of a sequence model. In order to perform efficient sampling over an infinite space of CCG derivations, we adapt the beam sampler developed by Van Gael et al. (2008) to work for PCFG trees.

Unlike Van Gael et al.'s models, which tagged words with opaque, atomic labels, our model remains firmly grounded in the CCG formalism. Each of these labels, while infinite in number, is a specific CCG category that must interact with other categories in specific and pre-defined ways, ultimately allowing us to manage the complexity of the model, without sacrificing the infinite characteristics. Moreover, since we know *a priori* what these labels mean, we can apply the CCG-specific priors designed in Chapter 3 that use universal knowledge of how the CCG formalism works to bias the model toward categories that are cross-linguistically more likely.

4.5.1 Infinite CCG model

In §4.2.1, we presented a Bayesian PCFG model for CCG parsing. While it had clear benefits, such as the ability to incorporate universal linguistic knowledge about the CCG formalism as priors, the model required a fixed, finite set of CCG categories, making it overly dependent on the tag dictionary provided as weak supervision.

In this section, we present a novel, *infinite CCG model*. This new model takes a Bayesian nonparametric approach, allowing it to have an infinite category set, and thus, an infinite set of production rules. The nonparametric model is directly based off the finite version, and its generative story is identical to the one presented in §4.2.1, with the single exception that binary productions are now drawn from a nonparametric *Dirichlet process* instead of a finite Dirichlet distribution to enable sampling from an infinite space of possible binary productions:

$$\theta_{\mathtt{t}} \sim \mathrm{DP}(\alpha_\theta, \theta^0)$$

In principle, it might also be possible to allow for an infinite set of unary production rules based on our category generator's ability to invent new categories to rewrite to. However, we have chosen to continue to use only the fixed set of 13 linguistically-plausible unary rules proposed by Lewis and Steedman (2014) (see Table 4.1b). Doing so considerably cuts down the potentially complexity of the model.

We define the prior means for the model the same way as §4.2.1:

$$\begin{aligned}
\sigma^0(\mathbf{t}) &= P_{\text{CAT}}(\mathbf{t}) \\
\theta^0(\langle \mathbf{u}, \mathbf{v} \rangle) &= P_{\text{CAT}}(\mathbf{u}) \cdot P_{\text{CAT}}(\mathbf{v}) \\
\pi^0(\langle \mathbf{u} \rangle) &= P_{\text{CAT}}(\mathbf{u}) \\
\mu^0_{\mathbf{t}} &= P_{em} \\
\lambda^0 &= \langle \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3} \rangle
\end{aligned}$$

Critically, note that the prior for binary productions, θ^0, is defined for the cross-product of infinite category sets. It does this by making use of P_{CAT}, the generative model presented in §3.2.2 that defines a probability distribution over the infinite set of CCG categories $\mathcal{T}$. This consideration ensures that the binary production prior will be applicable as the model grows and invents new categories and new production rules.

4.5.2 Background: iHMM beam sampler

Van Gael et al. (2008) presented an inference algorithm for the iHMM that they call *beam sampling* that is able to efficiently sample tag sequences from the *true posterior* even when the tagset is infinite. The beam sampler for the iHMM can be thought of as an nonparametric variation of the forward-filter backward-sample (FFBS) algorithm (Carter and Kohn, 1996), the algorithm we used in Chapter 3 for our supertagger. It works by sampling auxiliary variables that "slice" the space of possible tag sequences, allowing the forward-filter operation to considering only a finite space of tag sequences (Neal, 2003).

Given $\pi_t(u)$, the probability of transitioning from tag t to tag u, and $\phi_t(w)$, the probability of emitting word w from tag t, and tagset $\mathcal{T}$, the FFBS algorithm samples a tagging for a sentence $\mathbf{w}$ using a dynamic program to inductively compute "forward," for each token w_i starting with $i = 0$, the probability of generating

$w_0, w_1, \ldots, w_i$ via any tag sequence that ends with $y_i = u$:[8]

$$p(y_i = u \mid w_{0:i}) = \phi_u(w_i) \cdot \sum_{t \in \mathcal{T}} \pi_t(u) \cdot p(y_{i-1} = t \mid w_{0:i-1})$$

The FFBS algorithm then passes "backward" through the sentence starting at $i = |\mathbf{w}|$ and sampling:

$$y_i \mid y_{i+1} \sim p(y_i = t \mid w_{0:i}) \cdot \pi_t(y_{i+1})$$

Since the iHMM allows for an infinite tagset $\mathcal{T}$, it is not possible to simply sum over all tags to compute $\sum_{t \in \mathcal{T}} \pi_t(u)$. Instead, the beam sampler separates the forward computation into two phases. First, for each word w_i, an auxiliary transition threshold q_i is sampled:

$$q_i \sim \text{uniform}(0, \pi_{t_{i-1}}(t_i))$$

where t_{i-1} and t_i are the tags assigned to w_{i-1} and w_i in the *previously*-sampled tag sequence. This allows us to compute, for each tag u at each position i, the forward values summing over only the previous tags t such that the probability of transitioning from t to u, $\pi_t(u)$, is greater than the threshold q_i. Assuming that $\mathcal{T}_{i-1}(u)$ is the set of tags at position $i-1$ that can possibly transition to u at position i, and $\mathcal{T}_i$ as the set of tags at position i that can possibly be reached by a transition from *some* tag at position $i-1$,[9]

$$\begin{aligned} p(y_i = u \mid w_{0:i}) = \phi_u(w_i) \ &\cdot \sum_{t \in \mathcal{T}_{i-1}(u)} p(y_{i-1} = t \mid w_{0:i-1}) \\ &\text{for each } u \in \mathcal{T}_i \\ &\text{where } \mathcal{T}_{i-1}(u) = \{t \mid q_i < \pi_t(u)\} \end{aligned}$$

Notice that the transition probability term $\pi_t(u)$ is no longer included in the summation; it only appears in the comparison to q_i.

[8]This is the same as the forward calulation in the Forward-Backward algorithm, as used for EM (Baum, 1972).

[9]$\mathcal{T}_i$ can be thought of as the complete (finite) set of tags that may be used to label token i.

This process effectively divides the sampling into two phases. The first phase, sampling q_i, incorporates the transition probabilities from π, while the second phase, forward-backward, incorporates the emission probabilities ϕ; in contrast, FFBS takes both probability distributions into account directly in the forward-backward procedure.

Given the above equation, for each u, assuming that $\mathcal{T}_{i-1}(u)$ is finite, we can compute $p(y_i = u \mid w_{0:i})$. But since there are a potentially infinite number of tags u, we must ensure that we only need to handle a finite subset of them, $\mathcal{T}_i$. However, we know that for any t at $i-1$, there are finitely many u such that $q_i < \pi_t(u)$, since if there were infinitely many such u, then $\sum_u \pi_t(u)$ would be greater than 1 (it would actually be infinite), making π_t an invalid probability distribution. This means that for any t, there are a finite number of u that are reachable via a non-zero transition. Given that the sentence must start with $y_0 = \langle \text{S} \rangle$, there will necessarily be finitely many possibilities for y_1, making $\mathcal{T}_1$ finite; and from each of those, there are finitely many possibilities for y_2, making $\mathcal{T}_2$ finite since it is the union of a finite set of finite sets; and so on, demonstrating that for any y_i, there will be finitely many choices, so every $\mathcal{T}_i$ will be finite. Likewise, we know that every set $\mathcal{T}_{i-1}(u)$ will be finite since it will necessarily be a subset of $\mathcal{T}_{i-1}$ (only those tags that may transition to u at i), and $\mathcal{T}_{i-1}$ is guaranteed to be finite.

Sets $\mathcal{T}_i$ and $\mathcal{T}_{i-1}(u)$ can be computed inductively, starting with $\mathcal{T}_0 = \{\langle \text{S} \rangle\}$:

$$\begin{aligned} \textit{PotentialTransitions}(i-1,i) &= \textstyle\bigcup_{t \in \mathcal{T}_{i-1}} \{\langle t,u \rangle \mid q_i < \pi_t(u)\} \\ \mathcal{T}_i &= \{u \mid \exists t.[\langle t,u \rangle \in \textit{PotentialTransitions}(i-1,i)]\} \end{aligned}$$

and

$$\mathcal{T}_{i-1}(u) = \{t \mid \langle t,u \rangle \in \textit{PotentialTransitions}(i-1,i)\}$$

Finally, recall that $\pi_{t_{i-1}}(t_i)$ is the probability of the $i-1$ to i transition in the *current* sequence (the one sampled during the previous iteration). Since every q_i is necessarily less than $\pi_{t_{i-1}}(t_i)$, it will always be the case that every transition in the current sequence will exceed the full set of q thresholds. This ensures that the full current tag sequence will always be a candidate for resampling, even after thresholding. Note that this also entails that despite the presence of thresholds and cutoffs, there will always be at least one full tag sequence that exceeds all thresholds: the current one.

4.5.3 Beam sampling for the infinite CCG

By analogy to the beam sampler presented by Van Gael et al. (2008) for the iHMM, in which the FFBS algorithm was augmented with auxiliary variables to slice through an infinite model space, we designed a beam sampler for the infinite CCG model. Our sampler is based on the one we presented in §4.2.2 for our finite CCG model, but it introduces auxiliary slicing variables to handle the infinite space of binary productions.

The inference procedure still iterates other two alternating phases: given an initial set of trees, we first sample the model parameters, then sample a new tree for each sentence based on these new model parameters, then resample the parameters, and so on.

Sampling parameters

Given prior means and a set of CCG trees, we can resample the model parameters. We do so in the same manner as we did with the finite model (§4.2.2): count of occurrences of the various productions, combine them with the priors, and sample new parameters from Dirichlet distributions. The one exception is that the binary production parameters are drawn from a Dirichlet process to allow for an infinite set of potential binary rules:

$$\theta_{\mathbf{t}} \sim \mathrm{DP}(\langle \alpha_\theta \cdot \theta^0(\langle \mathbf{u}, \mathbf{v} \rangle) + C(\mathbf{t} \rightarrow \langle \mathbf{u}, \mathbf{v} \rangle) \rangle_{\mathbf{u},\mathbf{v} \in \mathcal{T}})$$

Note that the result selects parameters based on both the data (counts) and the biases encoded in the priors.

To efficiently sample infinite distributions, we borrow from the techniques of Van Gael et al. (2008), adapting them to be applicable to a tree model and to take advantage of the nature of CCG categories. For each infinite distribution $\theta_{\mathbf{t}}$, we distinguish between *observed* and *unobserved* productions. Observed productions are those that have counts from the sampled trees, while unobserved productions are those that are possible but do not appear in the current data.

We reduce the sampling of an infinite distribution to sampling a finite distribution over observed productions, leaving space for all unobserved productions. Since a Dirichlet distribution needs a finite set of parameters, we will have one parameter for each observed production, plus one additional parameter that accounts for *all* of the unobserved productions. Thus, for a set of observed productions $\mathcal{O}$, we will have a $|\mathcal{O}| + 1$ Dirichlet parameters. So if $\mathcal{O}^{\theta}_{\mathbf{t}}$ is the set of category pairs $\langle \mathbf{u}, \mathbf{v} \rangle$ that have been observed as productions from category $\mathbf{t}$, and $\mathcal{U}^{\theta}_{\mathbf{t}}$ is the set of productions that are unobserved, then each of the Dirichlet parameters will look very much like they did previously: the concentration parameter α_{θ}, times the prior mean θ^0, plus the count observed in the previous iteration's sampled result. The last parameter to the Dirichlet will be of a similar form, but its prior production probability will be the *sum* of all the prior probabilities for all unobserved productions (i.e., all of the productions not handled by the previous $|\mathcal{O}|$ parameters), meaning that it is the prior probability of seeing *any* unobserved production.

$$\begin{aligned}\theta^{\text{OBSV}}_{\mathbf{t}} \sim \text{Dir}\Big(&\alpha_{\theta} \cdot \theta^0(\mathbf{o}_1) + C(\mathbf{t} \rightarrow \mathbf{o}_1), \\ &\alpha_{\theta} \cdot \theta^0(\mathbf{o}_2) + C(\mathbf{t} \rightarrow \mathbf{o}_2), \\ &\ldots, \\ &\alpha_{\theta} \cdot \theta^0(\mathbf{o}_{|\mathcal{O}^{\theta}_{\mathbf{t}}|}) + C(\mathbf{t} \rightarrow \mathbf{o}_{|\mathcal{O}^{\theta}_{\mathbf{t}}|}), \\ &\alpha_{\theta} \cdot \sum_{\mathbf{u}' \in \mathcal{U}^{\theta}_{\mathbf{t}}} \theta^0(\mathbf{u}')\Big)\end{aligned}$$

for $\mathbf{o}_i \in \mathcal{O}^{\theta}_{\mathbf{t}}$. Note that no counts are included in the last sum since unobserved elements have, by definition, zero counts. Of course, it is not possible to compute the sum over the full infinite set of unobserved productions $\mathcal{U}^{\theta}_{\mathbf{t}}$, but this value can be easily computed as $1 - \sum_{\mathbf{o}' \in \mathcal{O}^{\theta}_{\mathbf{t}}} \theta^0(\mathbf{o}')$, and we know that $\mathcal{O}^{\theta}_{\mathbf{t}}$ will always be finite since we can only have seen a finite number of productions. We invent the pseudo-production z to be the "any-unobserved production" described by this final parameter, so that $\theta_{\mathbf{t}}^{\text{OBSV}}(z)$ is the probability of producing *any* unobserved pair from $\mathbf{t}$.

For observed productions, having sampled $\theta_{\mathbf{t}}^{\text{OBSV}}(\mathbf{o})$ is sufficient. But for unobserved productions, we must be able to determine the sampled probability $\theta_{\mathbf{t}}(\mathbf{u})$. To compute these values efficiently, we make use of the fact that we have defined our model so that we have a complete prior probability distribution over all possible binary productions (based on the use of the infinite category prior P_{CAT}). We can use this to extract the probability of a *particular* production $\mathbf{u}$ given the sum of the probabilities of all unobserved productions $\theta_{\mathbf{t}}^{\text{OBSV}}(z) = \sum_{\mathbf{u}' \in \mathcal{U}_{\mathbf{t}}^{\theta}} \theta_{\mathbf{t}}^{\text{OBSV}}(\mathbf{u}') = 1 - \sum_{\mathbf{o}' \in \mathcal{O}_{\mathbf{t}}^{\theta}} \theta_{\mathbf{t}}^{\text{OBSV}}(\mathbf{o}')$.[10] Using this information, we can define the sampled probability of an unobserved production $\mathbf{u}$ to be the fraction of $\theta_{\mathbf{t}}^{\text{OBSV}}(z)$ that is proportional to the fraction of the prior distribution's unobserved production mass devoted to $\mathbf{u}$:

$$\begin{aligned}\theta_{\mathbf{t}}^{\text{UNOB}}(\mathbf{u}) &= \theta_{\mathbf{t}}^{\text{OBSV}}(z)\frac{\theta^0(\mathbf{u})}{\sum_{\mathbf{u}' \in \mathcal{U}_{\mathbf{t}}^{\theta}} \theta^0(\mathbf{u}')}, \qquad \text{for } \mathbf{u} \in \mathcal{U}_{\mathbf{t}}^{\theta} \\ &= \theta_{\mathbf{t}}^{\text{OBSV}}(z)\frac{\theta^0(\mathbf{u})}{1 - \sum_{\mathbf{o}' \in \mathcal{O}_{\mathbf{t}}^{\theta}} \theta^0(\mathbf{o}')}\end{aligned}$$

Thus, for the full sampled distribution, we have:

$$\theta_{\mathbf{t}}(\mathbf{x}) = \begin{cases} \theta_{\mathbf{t}}^{\text{OBSV}}(\mathbf{x}) & \text{if } \mathbf{x} \in \mathcal{O}_{\mathbf{t}}^{\theta} \\ \theta_{\mathbf{t}}^{\text{UNOB}}(\mathbf{x}) & \text{otherwise} \end{cases}$$

Note that for any unseen $\mathbf{t}$, $\mathcal{O}_{\mathbf{t}}^{\theta} = \emptyset$, and so it follows from the above that $\theta_{\mathbf{t}}(\mathbf{x}) = \theta_{\mathbf{t}}^{\text{UNOB}}(\mathbf{x}) = \theta^0(\mathbf{x})$, for all $\mathbf{x}$. We likewise assume that in this case, all $\mathbf{t}$-conditioned distributions default to their prior means ($\pi_{\mathbf{t}} = \pi^0$, $\mu_{\mathbf{t}} = \mu^0$, ...).

One major advantage of this approach of representing a Dirichlet process as a finite vector whose unobserved tail is divided deterministically is that it allows sentence sampling to be performed in parallel, further increasing the potential for computational efficiency. Since all of the observed productions are known ahead of

[10]This is similar to Algorithm 8 of Neal (2000).

time, their probabilities can be sampled up front, before the sentences are resampled. But since we will not know which unobserved productions are needed until we resample the sentences, sampling these probabilities must wait. If a traditional approach to Dirichlet process sampling was used, we would have to guarantee that if an unobserved production's probability was sampled on one sentence, that the same probability would be used for that production on all other sentences, thus creating inter-sentence dependencies that would prevent parallelization. The amount of probability mass devoted to each unobserved-production tail is known as well, so in our case we are able to use our fixed category prior to divide the unobserved-production tail without worrying about interaction with other sentences, knowing that those sentences are guaranteed to produce the same probabilities.

Sampling threshold variables (q)

By analogy with the iHMM beam sampler, we seek to restrict the sets of productions that must be examined during sampling to be finite. For the iHMM, this meant a finite set of potential transitions from each observation. For the parsing model, as a tree, it will mean a finite set of productions from each cell in the parse chart.

But if we allow for an infinite category set, then for any category $\mathbf{t}$, we can find an infinite number of category pairs $\langle \mathbf{u}, \mathbf{v} \rangle$ that can combine to form $\mathbf{t}$. For example, the category s can be created, among other ways, by *forward application* on the pairs $\langle \mathsf{s/s}, \mathsf{s} \rangle$, $\langle \mathsf{s/(s/s)}, \mathsf{(s/s)} \rangle$, $\langle \mathsf{s/(s/(s/s))}, \mathsf{(s/(s/s))} \rangle$, etc. Since computing an inside probability for $\mathbf{t}$ requires summing across all potential subtrees that might form $\mathbf{t}$, it would be necessary to sum over an infinite set.

Just like the iHMM, we employ a strategy of sampling threshold values q and limiting valid productions to only those that exceed the threshold. To determine the finite subset of productions valid at each potential node in the tree, we present a procedure that passes top-down through the tree sampled on the previous iteration to sample q values and feed downward finite sets of categories.

We note here that while it may be possible to consider unary productions to also be an infinite set since unary productions produce categories and there are an infinite number of categories, we follow Lewis and Steedman (2014) and others in allowing only a small, fixed, finite set of linguistically-motivated unary rules, listed in Table 4.1b. We will refer to this set as $\mathcal{R}^{\pi}$, and the set of unary productions $\langle \mathbf{u} \rangle$ reachable from **t** as $\mathcal{R}^{\pi}_{\mathbf{t}}$. As a result, our sampler does not need to worry about thresholds on unary productions; it can always simply sum across the small set of relevant unary rules.

To compute finite sets $\mathcal{R}^{\theta}_{ij,\mathbf{t}}$, the set of productions possible from **t** in cell (i, j), we execute the following algorithm. Following Van Gael et al. (2008), we sample threshold values q based on the productions in the tree y sampled on the previous iteration. However, unlike the iHMM, there is not a one-to-one correspondence between parse chart cells and nodes in y since a parse tree will not have a node for every span. This is similar to the problem encountered by Blunsom and Cohn (2010a) for sampling synchronous grammars; we adopt their two-part strategy of sampling based on y when (i, j) is a span in y, and sampling from a Beta distribution when it is not.[11] Assuming that S_y is the set of spans in y, y_{ij} is y's category at node index (i, j), and $\mathcal{T}_{ij}$ is the set of allowed categories for that node based on the sampled threshold, we present the procedure in Algorithm 1.[12]

It is important to note that there may be multiple nodes in the (i,j) position of the tree y due to the fact that unary rules "loop back" to the same cell. But for the purposes of sampling q thresholds in Algorithm 1, we are only concerned with nodes that are part of a binary production. Thus, when we write y_{ij}, we only want the category that emits a binary production in the tree. Likewise, we only want the y_{ik} and y_{kj} that result from that binary production.

Of course, it is not actually possible to traverse $\mathcal{T}$ in order to iterate for **u** and **v** since $\mathcal{T}$ is infinite. We are required to traverse the set of *observed* binary productions, since their probabilities are sampled directly from the Dirichlet. But for the infinite tail of unobserved productions we instead rely on the fact that the generative model of P_{CAT} gives us a recipe for computing the probabilities. We can use the definition of P_{CAT} to generate a list of categories **u**, ordered highest- to

[11]Following Blunsom and Cohn (2010a), $a < 1$ and $b = 1$.

[12]Note that $\theta_{\mathbf{t}}(\langle \mathbf{u}, \mathbf{v} \rangle) = 0$ if there is no CCG rule that can combine **u** and **v** to form **t**.

Algorithm 1 Construct a finite parse chart.

```
1: $\mathcal{T}_{0n} = \{\mathsf{s_{dcl}}, \mathsf{np}\}$
2: for $span \leftarrow n$ downto $1$ do
3:     for $i \leftarrow 0$ to $n - span$ do
4:         $j = i + span$
5:         $\mathcal{T}_{visited} = \emptyset$
6:         for $\mathbf{t} \leftarrow \mathcal{T}_{ij} - \mathcal{T}_{visited}$ do
7:             $\mathcal{T}_{visited}$ += $\mathbf{t}$
8:             // follow any relevant unary rules
9:             $\mathcal{T}_{ij} = \mathcal{T}_{ij} \cup \mathcal{R}^{\pi}_{\mathbf{t}}$
10:            // if non-terminal (binary branch point)
11:            if $span > 1$ then
12:                // sample a threshold $q$
13:                if $(i, j) \in S_y$ then
14:                    $q \sim$ uniform$(0, \lambda_{y_{ij}}(1) \cdot \theta_{y_{ij}}(\langle y_{ik}, y_{kj} \rangle))$
15:                else
16:                    $q \sim$ Beta$(a, b)$
17:                // follow above-q binary productions
18:                for $\mathbf{u} \in \mathcal{T}, \mathbf{v} \in \mathcal{T}$ do
19:                    if $\lambda_{\mathbf{t}}(1) \cdot \theta_{\mathbf{t}}(\langle \mathbf{u}, \mathbf{v} \rangle) > q$ then
20:                        for $k \leftarrow i + 1$ to $j - 1$ do
21:                            $\mathcal{R}^{\theta}_{ij,\mathbf{t}}$ += $\langle k, \langle \mathbf{u}, \mathbf{v} \rangle \rangle$
22:                            $\mathcal{T}_{ik}$ += $\mathbf{u}$
23:                            $\mathcal{T}_{kj}$ += $\mathbf{v}$
```

lowest-probability, allowing us to traverse only as far as necessary until the production probability will drop below q. One could also do the same thing for **v**, listing all high-probability categories and finding the pairs $\langle \mathbf{u}, \mathbf{v} \rangle$ that can be combined using some combination rule $c \in \mathcal{C}$ to form **t** with total probability above q, a process would take $O(|\mathcal{T}|^2 \times |\mathcal{C}|)$ time. However, because of the nature of CCG, we can reduce the time complexity by recognizing that for any parent category **t** and combination rule c, we can infer either child category from the other. For example, if $\mathbf{t} = \mathsf{np}$ and $\mathbf{u} = \mathsf{np/n}$, then **v** would be n under Forward Application, or $\mathsf{np\backslash(np/n)}$ under Backward Application, and so on. This means that we need only traverse, for each high-probability **u**, the list of combination rules to directly compute each **v**, yielding a time complexity of $O(|\mathcal{T}| \times |\mathcal{C}|)$.

This algorithm, since it starts at the root, will only generate productions that can actually be used in complete parse trees; this ensures that the inside algorithm will not perform unnecessary computation, without requiring an additional pruning stage such as that used in §4.2.

As we discussed with the iHMM, a finite set $\mathcal{T}_{ij}$ guarantees that both $\mathcal{T}_{ik}$ and $\mathcal{T}_{kj}$ will be finite. But since the root node has no higher-level node to guarantee its finiteness, we simply set $\mathcal{T}_{0n} = \{\mathsf{s}_{\mathsf{dcl}}, \mathsf{np}\}$, so that there will only be a finite number of productions from the root.

Like the iHMM beam sampler, this procedure has an additional important guarantee: there will always be at least one tree available to be sampled after thresholding. Specifically, the previous tree y will always be available. We know this to be the case because the threshold q sampled for any span (i, j) in S_y will always be less than the probability of the existing production in y, $\lambda_{\mathsf{t}}(1) \cdot \theta_{y_{ij}}(\langle y_{ik}, y_{kj}\rangle)$. For spans not in S_y, we sample from a Beta distribution, but it is not necessary that q be less than any production probability since that span is not required for y. Finally, all root, unary, and terminal productions used in y will be available since these production sets are finite, and thus not thresholded.

Sampling trees

In §4.2.2, we presented a sampler for a CCG model with a finite category set. We extend that work here to design a novel beam sampler for our nonparametric model. To do so, we make use of the finite $\mathcal{R}^{\theta}$ sets computed above.

Like Johnson et al. (2007), our sampler makes two passes over the tree: an "upward" pass computing the inside chart, and a "downward" pass to sample a tree from that chart. The distinction is that, much like the iHMM sampler, our procedure needs to have only a finite set of productions over which to sum during the inside computation. By sampling auxiliary threshold variables q, and using them to constrain the $\mathcal{R}^{\theta}$ sets as shown above, we are able to define our new inside algorithm, a modification of the algorithm presented in §4.2.2:

$$\begin{aligned}
p(w_i \mid y_{i,i+1} = \mathbf{t}) &= \lambda_{\mathbf{t}}(3) \cdot \mu_{\mathbf{t}}(w_i) \\
&\quad + \textstyle\sum_{\langle \mathbf{u} \rangle \in \mathcal{R}^{\pi}_{\mathbf{t}}} \lambda_{\mathbf{t}}(2) \cdot \pi_{\mathbf{t}}(\langle \mathbf{u} \rangle) \cdot p(w_{i:i} \mid y_{i,i+1} = \mathbf{u}) \\
p(w_{i:j-1} \mid y_{ij} = \mathbf{t}) &= \\
&\quad \textstyle\sum_{\langle \mathbf{u} \rangle \in \mathcal{R}^{\pi}_{\mathbf{t}}} \lambda_{\mathbf{t}}(2) \cdot \pi_{\mathbf{t}}(\langle \mathbf{u} \rangle) \cdot p(w_{i:j-1} \mid y_{ij} = \mathbf{u}) \\
&\quad + \textstyle\sum_{\langle k, \langle \mathbf{u}, \mathbf{v} \rangle \rangle \in \mathcal{R}^{\theta}_{ij,\mathbf{t}}} \; p(w_{i:k-1} \mid y_{ik} = \mathbf{u}) \cdot p(w_{k:j-1} \mid y_{kj} = \mathbf{v})
\end{aligned}$$

$\mathcal{R}^{\theta}_{ij,\mathbf{t}}$ represents the set of above-the-threshold binary production rules $\mathbf{t} \rightarrow \langle \mathbf{u}, \mathbf{v} \rangle$ where $\mathbf{t}$ is in cell (i, j), $\mathbf{u}$ is in cell (i, k), $\mathbf{v}$ is in cell (k, j). Notice that the $\lambda_{\mathbf{t}}(1) \cdot \theta_{\mathbf{t}}(\langle \mathbf{u}, \mathbf{v} \rangle)$ component is not present in the inside calculation (as it was with our finite-CCG model); this is because, as we saw with the iHMM, the probability of the binary production has already been incorporated into the sampling of the threshold q used to select $\mathcal{R}^{\theta}_{ij,\mathbf{t}}$.

From a completed (now-finite) inside chart, we can sample a tree in the downward direction, respecting the sampled thresholds by following $\mathcal{R}^{\theta}_{ij,\mathbf{t}}$:

$$\begin{aligned}
y_{0n} &\sim \quad \sigma_{\mathbf{t}} \cdot p(w_{0:n-1} \mid y_{0n} = \mathbf{t}) \\
x \mid y_{ij} &\sim \Big\langle \theta_{y_{ij}}(\langle \mathbf{u}, \mathbf{v} \rangle) \cdot p(w_{i:k-1} \mid y_{ik} = \mathbf{u}) \cdot p(w_{k:j-1} \mid y_{kj} = \mathbf{v}) \\
&\qquad\qquad\qquad\qquad \forall \langle y_{ik}, y_{kj} \rangle \in \mathcal{R}^{\theta}_{ij,y_{ij}} \text{ when } j > i{+}1, \\
&\qquad \pi_{y_{ij}}(\langle \mathbf{u} \rangle) \cdot p(w_{i:j-1} \mid y'_{ij} = \mathbf{u}) \qquad \forall\, y'_{ij}, \\
&\qquad \mu_{y_{ij}}(w_i) \qquad\qquad\qquad\qquad \text{when } j = i + 1 \;\Big\rangle
\end{aligned}$$

where x is either a split point k and pair of categories y_{ik}, y_{kj} resulting from a binary rewrite rule, a single category y'_{ij} resulting from a unary rule, or a word w resulting from a terminal rule.

Decoding

We parse using probabilistic CKY (P-CKY). Since it is necessary to limit the set of supertags for each token, but we don't want to use the initial tag dictionary since it doesn't support any of the new categories that were created during sampling, we extract a new tag dictionary from the pool of trees resulting from all sampling iterations.

4.5.4 Performing inference

Since an initial tree for each training sentence is needed in order to sample q values, we start the inference procedure with the max probability parse of each sentence in the training corpus, computed using P-CKY with the prior means as parameters (σ^0, θ^0, ...). We use an initial tag dictionary to constrain the supertag set of each token for P-CKY. Once sampling begins, we drop the tag dictionary, allowing a fully-infinite category set for all words; this allows our lexicon to grow beyond the categories found in the initial tag dictionary.

After all sampling iterations have completed, the parameters are estimated as the maximum likelihood estimate of the pool of trees resulting from all sampling iterations.

4.5.5 Conclusions

In this section, we have presented a novel nonparametric CCG parsing model that allows for an infinite set of CCG categories, and thus, an infinite set of production rules. For this model, we have also designed a novel tree-based *beam sampling* inference procedure that efficiently samples from an infinite space of potential CCG derivations. The flexibility of an infinite category set within the CCG framework provides a principled way for our model to adapt to new data and new grammatical constructions, inventing new categories and production rules as needed.

Moreover, our category and production sets, while still infinite, are firmly grounded in the framework of CCG. This means that our models will always produce valid derivations that adhere to the rules of CCG. More importantly, it means that we are able to use the priors defined in Chapter 3 to bias our model toward the use of categories that are more plausible given the intrinsic, universal, cross-linguistic properties of the CCG formalism.

Finally, though we have described it in the context of CCG, this approach is generalizable to generic PCFG models. However, doing so would lose many of the serious efficiency advantages that come from our use of CCG, such as the fact that when examining potential productions, we only have to iterate over categories for one side of a branch since we are able to infer its sibling given the small, finite set of CCG rules.

4.6 Conclusions and Future Work

In this chapter, we have presented work on weakly-supervised CCG parsing that is able to take advantage of universal properties of the CCG formalism to estimate better model parameters under weak supervision in a Bayesian setting. We have further showed how additional careful exploitation of any available supervision can lead to further gains.

Because of the structured nature of CCG categories and the logical framework in which they must assemble to form valid parse trees, the CCG formalism offers multiple opportunities to bias model learning based on universal, intrinsic properties of the grammar. In §4.2 we saw how priors on root, unary, and binary productions in a parsing model could be biased toward *a priori* more likely categories, as defined by the probabilistic category grammar presented in §3.2.2 that encourages simpler categories and modifier categories. Subsequently, in §4.3, we showed how CCG's preference for adjacent categories that are combinable could be captured by a novel *supertag-context* parsing model, incorporating the sequential relationships that were successfully exploited by the CCG supertagger in Chapter 3. This allowed us to bias the model toward trees whose categories logically "fit" into their contexts. For both models, we showed that using the tag dictionary and raw corpus to estimate relationships between words and supertags to use as priors on terminal distributions leads to further improvements.

Finally, we showed in §4.4 that exploiting bracket annotations can help guide a sampler to estimate better parameters in a parsing model. We used oracle simulation experiments to determine what kinds of brackets lead to the best results under constrained time scenarios, and then used the lessons learned to give guidance to human annoators, who participated in timed experiments to prepare data. Ultimately, we found that the brackets produced by human annotators are able to compete with gold bracket information extracted from the supervised corpus. This demonstrates that collecting bracket annotations is a viable form of supervision.

The idea of incorporating linguistic knowledge into a model via priors is very appealing when supervised data is scarce. We have shown how knowledge about the structure of CCG categories can be used, but a more sophisticated model may be able to additionally make use of knowledge about the relative likelihoods of various CCG rules. For example, we know that *application* rules are always preferred, and that *composition* rules should only be used when necessary, and in specific scenarios (Baldridge, 2002). Further, rules like *merge* should only be used as a last resort.

Lastly, while a relatively large tag dictionary was used for the experiments presented here, it may be possible to learn from even less supervision by generalizing a small dictionary into a large one, as we have shown in possible for type-supervised part-of-speech tagging in Chapter 2.

We conclude with a final section, §4.5 below, in which we present a novel, *nonparametric*, Bayesian CCG parsing model. One of the obvious limitations to the type-supervised learning shown so far is that any tag dictionary used will necessarily be incomplete, missing not only some of the words, but also some of the necessary categories associated with known words. However, because the category generator we designed (§3.2.2) is itself a generative model, we are able to generate new categories at will while sampling in order to introduce novel categories. This allows us to assocate new categories with known words, introduce categories that have never been seen with any word, and introduce finite sets of likely categories for words unknown words. We also present a novel, efficient sampler for our nonparametric model based on the *beam sampler* introduced by Van Gael et al. (2008) for sampling in an *infinite hidden Markov model*. We show that this sampler can be designed to advantage of properties of the CCG formalism to further reduce the time complexity.

Chapter 5
Conclusion

As natural language processing applications move toward new languages and domains, where resources for training models are less plentiful, the need for methods that learn models from less supervision becomes more important. In this dissertation, we have presented work on learning syntactic tagging and parsing models from various forms of weak supervision. Our work is able to use available resources to construct *inductive biases*, and we have demonstrated that these biases are valuable when training our models.

Our work has drawn on two broad categories of information to construct our inductive biases. The first is information collected from human annotators. In Chapter 2, we showed how it is possible to train good part-of-speech (POS) taggers on supervision that was collected under extremely short time restrictions. Our methods were able to achieve, with just four hours of guidance, results comparable to those of systems announced around the same time that required volumes of data that took years to collect. To accomplish this, we developed techniques for estimating probability distributions from partial tag dictionaries (mappings from word types to sets of potential tags) and raw text — but without fully-annotated text — that could be used for parameter initialization or as a prior on the word-tag relationships in our models. Further, we designed an approach for automatically generalizing an extremely small set of annotations into a much larger set of pseudo-annotations, allowing for better even better estimates. We evaluated these approaches with live experiments on truly low-resource languages in which we had human subjects annotate data under a variety of scenarios, allowing us to compare the relative value of different kinds of supervision, while being sure to make our comparisons on the basis of actual annotation effort expended. These studies have made it possible for us to make specific recommendations to researchers who wish to collect corpora for new languages.

The other major source of inductive bias is derived from the universal properties of grammar itself. Here, we recognize that, cross-lingually, languages tend share certain underlying properties. Relevant to us is the tendency for languages to prefer simpler syntactic analyses when possible. To account for this tendency, we invoked the *minimum description length* (MDL) principle, stating that we wanted our models to learn compact grammars that minimize the number of rules and word-tag relationships required, or, alternatively, that maximize rule reuse across sentences.

In order to be able to train tools for syntactic analysis that follow these principles, we selected a grammatical formalism and learning framework that would be able to capture and exploit our prior beliefs about the universal properties of grammar. We chose to work within the framework of Bayesian statistical methods due to its compatibility with our problem. Parameter inference in a Bayesian model can be thought of as the combination of two elements: the (unannotated) data, which the algorithm attempts to describe through the model and its parameter settings, and the *prior*, which captures our underlying beliefs about how we think the model should look. We showed that our problem aligns well to the Bayesian context since unannotated text is relatively voluminous and easy to obtain, and we can encode our universal properties of grammar (as well as additional information extracted from other available annotations) as priors to guide the inference algorithm toward simpler, more compact grammars. These priors are, therefore, treated as *soft* constraints on learning, meaning that they bias the learner toward simpler models, but that these biases may be overridden when enough contradictory evidence is found in the data.

For our grammatical representation, we chose to use the Combinatory Categorial Grammar (CCG) formalism because we were able to devise a way to encode the intrinsic, universal, cross-lingual idea of "preferring simpler grammars" using the structure of CCG categories. In Chapter 3, we showed that it was possible to design a generative probabilistic grammar over CCG categories, to be used as a prior, that assigns high probability to simple tags, and geometrically decreases in probability as categories grow more complex. Further, we were able to design a method for estimating the likelihood of seeing two categories on adjacent words based on their structures alone. We showed that by exploiting both of these properties for priors in the Bayesian learning of a CCG supertagger from only incomplete, type-level (tag dictionary) supervision, we were able to learn models that performed significantly better than those trained without the universal biases. We also showed that it is possible to achieve further large gains by carefully exploiting the weak supervision, extracting as much information as possible from the tag dictionary and raw, unannotated text, as we did previously with POS-tagging.

Since we are ultimately interested in full syntactic parsing, not simply tagging, we next turned our attention to CCG parsing models. In Chapter 4 we showed that the same prior on CCG categories that encourages categories that are simpler and more cross-linguistically plausible can be used to encourage simpler categories at all levels of a parse tree, and experimentally verified that when learning from incomplete type-level supervision, using this bias toward simplicity allows us to outperform a model trained without this explicit bias.

In §4.3 we went further, designing a novel model that adds additional parameters to our previous CCG parsing model for the specific purpose of being able to incorporate, as an inductive bias, our *a priori* knowledge that adjacent categories are likely to be combinable. These additional parameters capture left- and right-side *supertag contexts*, making it additionally possible to incorporate priors that encourage the use of constituent labels that "fit" into their contexts. Our experiments demonstrate that the addition of these context parameters improves the inference algorithm's ability to find good model parameters under incomplete type-level supervision scenarios, and that using a prior on these parameters that biases them toward combinability improves results further.

Thinking again about the task of weakly-supervised learning more broadly, we believe that when supervision is scarce, it is likely that using a variety of kinds of supervision may prove more valuable by providing different, and complementary, perspectives on the data. We saw this, for example, in our POS-tagging work when, for the morphologically-rich language Kinyarwanda, in extremely low annotation scenarios, the incorporation of morphological information, extracted automatically by a finite-state transducer, improved our ability to train a model. In §4.4

we explored this idea for parsing, and showed that CCG parsing can be improved by complementing the existing type-level supervision (a tag dictionary) with weak, partial constituent bracketing information. We executed a series of simulation experiments to assess the value of different bracket annotation collection strategies, giving us insights into the kinds of guidance that we should give to annotators in order to build the best parsers with the least effort. Among other results, we found, interestingly, that partial information on a large number of sentence proved more valuable to learning than obtaining complete bracketing information on a smaller number of sentences. We then put those insights to work by having human experimental subjects produce fast, noisy, incomplete bracket annotations for sentences that were used to train parsers, ultimately demonstrating that human-provided annotations provided in a short amount of time can be quite valuable, and provide results that can compete with bracketing information automatically extracted from an expertly-built treebank.

5.1 Future Directions

Future work can be classified into three broad categories: work on collecting better annotations and extracting better information from them, work on designing better models to make the most of weak annotation or allow for additional inductive biases, and work on the applications of weakly supervised models.

The first category of future work is that which continues on the path of lowering the amount of human input, or the cost of input, required to train these models. This may take the form of developing methods for using new kinds of supervision such as dependency annotations or semantic roles. For example, other researchers have found that information collected following the lightweight Graph Fragment Language dependency annotation scheme (Schneider et al., 2013) can be valuable for learning with limited time and funds for annotation (Kong et al., 2014; Mielens et al., 2015). It may also take the form of work that attempts to wring more value out of existing annotations, perhaps by developing better generalization techniques as we saw in particular with our POS work, or by identifying and exploiting addition universal properties of language. Additionally, the Grammar Matrix of Bender et al. (2002) is a framework for facilitating the efficient documentation of grammatical information about a language, which could in turn be used to inform parser learning approaches such as ours by providing grammar-level (as opposed to type- or token-level) information biases. Most importantly, there is much work to be done in discovering how different kinds of supervision may interact. By knowing which sources of data complement each other, which are redundant, how much of each is needed, and how difficult each is to procure, we can provide clues on how to optimize data collection to bootstrap NLP models in new domains.

The second category of future work is on the models used to lean from weak supervision. We have shown in this dissertation that simple, long-standing models such as hidden Markov models and probabilistic context-free grammars are indeed able to make good use of weak supervision — as long as it is encoded properly as a prior and an inference algorithm is used that takes advantage of it. However, we have also demonstrated that more complex models, such as our supertag-context model, are able to improve results by capturing additional inductive biases that simple models may have no way of encoding. In §4.5 we presented ideas for how to build a nonparametric parsing model that has the unique advantage of being able to model an infinite space of CCG syntax trees. By allowing for an infinite set of categories, and thus, an infinite set of production rules, this model is able to grow and shrink its lexicon, inventing new categories in order to adapt to new data. To learn the parameters for this model, we develop a new beam sampler that can sample from the posterior by slicing the infinite space of trees. Importantly, despite this high degree of flexibility, our model remains grounded in the CCG framework, which allows us to provide CCG-based priors that keep our model biased toward probable categories, and thus, simpler grammars. Further work in this direction may find ways of extending these or other models to be able to capture additional valuable linguistic biases.

Finally, the third broad category of future work concerns the applications of these weakly-supervised models. Part-of-speech tags and CCG grammars are not typically ends in themselves, but are building blocks for more complex NLP tasks. Many high-level NLP applications, as they attempt to move into new languages or new domains, may benefit from better-performing syntax-learning strategies. For example, machine translation and semantic parsing are both challenging tasks where grammar is an intermediate representation. To address machine translation for low-resource languages, or semantic parsing on new domains, it may be beneficial to be able to learn syntactic parsing models given only minimal amounts of inexpensive supervision.

References

Steven Abney and Steven Bird. 2010. The Human Language Project: Building a universal corpus of the world's languages. In *Proc. of ACL*.

Jason Baldridge and Alexis Palmer. 2009. How well does active learning actually work? Time-based evaluation of cost-reduction strategies for language documentation. In *Proc. of EMNLP*.

Jason Baldridge. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Jason Baldridge. 2008. Weakly supervised supertagging with grammar-informed initialization. In *Proc. of COLING*.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25.

Michele Banko and Robert C. Moore. 2004. Part-of-speech tagging in context. In *Proc. of COLING*.

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. 1999. The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3).

Leonard E. Baum. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Oved Shisha, editor, *Inequalities III: Proc. of the Third Symposium on Inequalities*, pages 1–8. New York: Academic Press.

Matthew J. Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. 2001. The infinite hidden Markov model. In *Proc. of NIPS*.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2002. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proc. of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*.

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Proc. of NAACL*.

Steven Bird. 2011. Bootstrapping the language archive: New prospects for natural language processing in preserving linguistic heritage. *Linguistic Issues in Language Technology*, 6.

Yonatan Bisk and Julia Hockenmaier. 2012. Simple robust grammar induction with combinatory categorial grammar. In *Proc. of AAAI*.

Yonatan Bisk and Julia Hockenmaier. 2013. An HDP model for inducing combinatory categorial grammars. *Transactions of the Association for Computational Linguistics*, 1.

Leonard Bloomfield. 1914. *An Introduction to the Study of Language*. Henry Holt and Company.

Leonard Bloomfield. 1933. *Language*. University of Chicago Press.

Phil Blunsom and Trevor Cohn. 2010a. Inducing synchronous grammars with slice sampling. In *Proc. of NAACL*.

Phil Blunsom and Trevor Cohn. 2010b. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proc. of EMNLP*.

Phil Blunsom and Trevor Cohn. 2011. A hierarchical Pitman-Yor process HMM for unsupervised part of speech induction. In *Proc. of ACL*.

Johan Bos, Cristina Bosco, and Alessandro Mazzei. 2009. Converting a dependency treebank to a categorial grammar treebank for Italian. In M. Passarotti, Adam Przepiórkowski, S. Raynaud, and Frank Van Eynde, editors, *Proc. of the Eighth International Workshop on Treebanks and Linguistic Theories (TLT8)*.

Cristina Bosco, Vincenzo Lombardo, Daniela Vassallo, and Leonardo Lesmo. 2000. Building a treebank for Italian: a data-driven annotation schema. In *Proc. of LREC*.

Glenn Carroll and Eugene Charniak. 1992. Two experiments on learning probabilistic dependency grammars from corpora. Technical report, Brown University.

Christopher K. Carter and Robert Kohn. 1996. On Gibbs sampling for state space models. *Biometrika*, 81(3):341–553.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. of ACL*.

Ciprian Chelba and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proc. of ACL*.

Zhiyi Chi. 1999. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proc. of ACL*.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT Press.

Christos Christodoulopoulos, Sharon Goldwater, and Mark Steedman. 2010. Two decades of unsupervised POS induction: How far have we come? In *Proc. of EMNLP*.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33.

Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proc. of TAG+6*.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *Journal of Machine Learning Research*.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Silviu Cucerzan and David Yarowsky. 2002. Bootstrapping a multilingual part-of-speech tagger in one person-day. In *Proc. of CoNLL*.

Dipanjan Das and Slav Petrov. 2011. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proc. of ACL-HLT*.

Hal Daumé III and Daniel Marcu. 2006. Domain adaptation for statistical classifiers. *JAIR*, 26.

Arthur P. Dempster, Nan M. Laird, and Donald. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39.

Markus Dickinson and W. Detmar Meurers. 2003. Detecting errors in part-of-speech annotation. In *Proc. of EACL*.

Weiwei Ding. 2011. Weakly supervised part-of-speech tagging for Chinese using label propagation. Master's thesis, University of Texas at Austin.

Dan Garrette and Jason Baldridge. 2012. Type-supervised hidden Markov models for part-of-speech tagging with incomplete tag dictionaries. In *Proc. of EMNLP*.

Dan Garrette and Jason Baldridge. 2013. Learning a part-of-speech tagger from two hours of annotation. In *Proc. of NAACL*.

Dan Garrette, Jason Mielens, and Jason Baldridge. 2013. Real-world semi-supervised learning of POS-taggers for low-resource languages. In *Proc. of ACL*.

Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. 2014. Weakly-supervised Bayesian learning of a CCG supertagger. In *Proc. of CoNLL*.

Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. 2015. Weakly-supervised grammar-informed Bayesian CCG parser learning. In *Proc. of AAAI*.

Daniel Gildea and Julia Hockenmaier. 2003. Identifying semantic roles using combinatory categorial grammar. In *Proc. of EMNLP*.

Daniel Gildea. 2004. Dependencies vs. constituents for tree-based alignment. In *Proc. of EMNLP*.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proc. of ACL-HLT*.

Yoav Goldberg, Meni Adler, and Michael Elhadad. 2008. EM can find pretty good HMM POS-taggers (when given a good start). In *Proc. of ACL*.

Sharon Goldwater and Thomas L. Griffiths. 2007. A fully Bayesian approach to unsupervised part-of-speech tagging. In *Proc. of ACL*.

Sharon Goldwater. 2007. *Nonparametric Bayesian Models of Lexical Acquisition*. Ph.D. thesis, Brown University.

Joshua Goodman. 1998. *Parsing inside-out*. Ph.D. thesis, Harvard University.

Aria Haghighi and Dan Klein. 2006. Prototype-driven learning for sequence models. In *Proceedings NAACL*.

Kazi Saidul Hasan and Vincent Ng. 2009. Weakly supervised part-of-speech tagging for morphologically-rich, resource-scarce languages. In *Proc. of EACL*.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *Proc. of ACL*.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3).

Rebecca Hwa. 1999. Supervised grammar induction using training data with limited constituent information. In *Proc. of ACL*.

Rebecca Hwa. 2000. Sample selection for statistical grammar induction. In *Proc. of EMNLP*.

Frederick Jelinek, John D. Lafferty, , and Robert L. Mercer. 1990. Basic methods of probabilistic context-free grammars. Technical report, IBM, Yorktown Heights, New York.

Mark Johnson and Sharon Goldwater. 2009. Improving nonparameteric Bayesian inference: Experiments on unsupervised word segmentation with adaptor grammars. In *Proc. of NAACL*.

Mark Johnson, Thomas Griffiths, and Sharon Goldwater. 2007. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Proc. of NAACL*.

Mark Johnson. 2007. Why doesn't EM find good HMM POS-taggers? In *Proc. of EMNLP-CoNLL*.

Aravind K. Joshi and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proc. of COLING*.

Lauri Karttunen. 2001. Applications of finite-state transducers in natural language processing. *Lecture Notes in Computer Science*, 2088.

Dan Klein and Christopher D. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proc. of ACL*.

Dan Klein and Christopher D. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proc. of ACL*.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A. Smith. 2014. A dependency parser for tweets. In *Proc. of EMNLP*.

Julian Kupiec. 1992. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech & Language*, 6(3).

Karim Lari and Steve J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Yoong Keok Lee, Aria Haghighi, and Regina Barzilay. 2010. Simple type-level unsupervised pos tagging. In *Proc. of EMNLP*.

Mike Lewis and Mark Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Proc. of EMNLP*.

Shen Li, João Graça, and Ben Taskar. 2012. Wiki-ly supervised part-of-speech tagging. In *Proc. of EMNLP*.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proc. of CICLing*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2).

Mike Maxwell and Baden Hughes. 2006. Frontiers in linguistic annotation for lower-density languages. In *Proc. of the Workshop on Frontiers in Linguistically Annotated Corpora*.

Bernard Merialdo. 1994. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2).

Jason Mielens, Liang Sun, and Jason Baldridge. 2015. Parse imputation for dependency annotations. In *Proc. of ACL*.

Tom M. Mitchell. 1980. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, New Jersey.

Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. 2010. Using universal linguistic knowledge to guide grammar induction. In *Proc. of EMNLP*.

Radford M. Neal. 2000. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265.

Radford M. Neal. 2003. Slice sampling. *Annals of Statistics*, 31(3):705–767.

Hermann Ney. 1991. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2):336–340.

Grace Ngai and David Yarowsky. 2000. Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking. In *Proc. of ACL*.

Miles Osborne and Ted Briscoe. 1997. Learning stochastic categorial grammars. In *Proc. of CoNLL*.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English Gigaword Fifth Edition LDC2011T07. Linguistic Data Consortium.

Lisa Pearl and Sharon Goldwater. In press. Statistical learning, inductive bias, and Bayesian inference in language acquisition. In Jeffrey Lidz, William Snyder, and Joseph Pater, editors, *Oxford Handbook of Developmental Linguistics*. Oxford University Press.

Fernando Pereira and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proc. of ACL*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. of NAACL*.

Slav Petrov, Dipanjan Das, and Ryan T. McDonald. 2012. A universal part-of-speech tagset. In *Proc. of LREC*.

Elias Ponvert, Jason Baldridge, and Katrin Erk. 2011. Simple unsupervised grammar induction from raw text with cascaded finite state models. In *Proc. of ACL-HLT*.

Lance Ramshaw, Elizabeth Boschee, Sergey Bratus, Scott Miller, Rebecca Stone, Ralph Weischedel, and Alex Zamanian. 2001. Experiments in multi-modal automatic content extraction. In *Proc. of HLT*.

Sujith Ravi and Kevin Knight. 2009. Minimized models for unsupervised part-of-speech tagging. In *Proc. of ACL-AFNLP*.

Sujith Ravi, Jason Baldridge, and Kevin Knight. 2010a. Minimized models and grammar-informed initialization for supertagging with highly ambiguous lexicons. In *Proc. of ACL*.

Sujith Ravi, Ashish Vaswani, Kevin Knight, and David Chiang. 2010b. Fast, greedy model minimization for unsupervised tagging. In *Proc. of COLING*.

Jorma Rissanen. 1995. Modeling by shortest data description. *Automatica*, 14(5):465–658.

Emmanuel Roche and Yves Schabes. 1995. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2).

Nathan Schneider, Brendan O'Connor, Naomi Saphra, David Bamman, Manaal Faruqui, Noah A. Smith, Chris Dyer, and Jason Baldridge. 2013. A framework for (under)specifying dependency syntax without overloading annotators. In *Proc. of the 7th Linguistic Annotation Workshop & Interoperability with Discourse*.

Stuart M. Shieber and Xiaopeng Tao. 2003. Comma restoration using constituency information. In *Proc. of NAACL*.

John Sinclair. 1992. The automatic analysis of corpora. In Jan Svartvik, editor, *Directions in Corpus Linguistics (Proceedings of Nobel Symposium 82)*. Mouton de Gruyter, Berlin.

Noah A. Smith and Jason Eisner. 2004. Annealing techniques for unsupervised statistical language learning. In *Proc. of ACL*.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proc. of ACL*.

Noah A. Smith. 2006. *Novel Estimation Methods for Unsupervised Discovery of Latent Structure in Natural Language Text*. Ph.D. thesis, Johns Hopkins University.

Noah A. Smith. 2011. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2011. Punctuation: Making a point in unsupervised dependency parsing. In *Proc. of CoNLL*.

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. In Robert Borsley and Kersti Borjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.

Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proc. of EACL*.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press.

Amarnag Subramanya, Slav Petrov, and Fernando Pereira. 2010. Efficient graph-based semi-supervised learning of structured tagging models. In *Proc. of EMNLP*.

Oscar Täckström, Dipanjan Das, Slav Petrov, Ryan McDonald, and Joakim Nivre. 2013. Token and type constraints for cross-lingual part-of-speech tagging. In *Transactions of the ACL*.

Partha Pratim Talukdar and Koby Crammer. 2009. New regularized algorithms for transductive learning. In *Proc. of ECML-PKDD*.

Kristina Toutanova and Mark Johnson. 2008. A Bayesian LDA-based model for semi-supervised part-of-speech tagging. In *Proc. of NIPS*.

Daniel Tse and James R. Curran. 2010. Chinese CCGbank: Extracting CCG derivations from the Penn Chinese Treebank. In *Proc. of COLING*.

Jurgen Van Gael, Yunus Saatci, Yee Whye Teh, and Zoubin Ghahramani. 2008. Beam sampling for the infinite hidden Markov model. In *Proc. of ICML*.

Jurgen Van Gael, Andreas Vlachos, and Zoubin Ghahramani. 2009. The infinite HMM for unsupervised PoS tagging. In *Proc. of EMNLP*.

Aline Villavicencio. 2002. *The acquisition of a unification-based generalised categorial grammar*. Ph.D. thesis, University of Cambridge.

Paul Viola and Mukund Narasimhan. 2005. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proc. of SIGIR*.

Sean Wallis. 2007. Annotation, retrieval and experimentation. In A. Meurman-Solin and A.A. Nurmi, editors, *Annotating Variation and Change*. Varieng, University of Helsinki, Helsinki.

Jonathan Weese, Chris Callison-Burch, and Adam Lopez. 2012. Using categorial grammar to label translation rules. In *Proc. of WMT*.

Wilhelm Wundt. 1900. *Völkerpsychologie: eine Untersuchung der Entwicklungsgesetze von Sprache, Mythus und Sitte*. Band II: Die Sprache, Zweiter Teil. W. Engelmann, Leipzig.

Nianwen Xue, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of UAI*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proc. of EMNLP*.