HOLISTIC LANGUAGE PROCESSING:
JOINT MODELS OF LINGUISTIC STRUCTURE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Jenny Rose Finkel
September 2010

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Christopher D. Manning) Principal Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Dan Jurafsky (Linguistics))

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Daphne Koller)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Andrew Y. Ng)

Approved for the University Committee on Graduate Studies.

Abstract

Humans are much better than computers at understanding language. This is, in part, because humans naturally employ holistic language processing. They effortlessly keep track of many inter-related layers of low-level information, while simultaneously integrating in long-distance information from elsewhere in the conversation or document. This thesis is about joint models for natural language processing which also aim to capture the dependencies between different layers of information and between different parts of a document when making a decision. I address three aspects of holistic language processing.

First, I present an information extraction model that includes long-distance links in order to jointly make decisions about related words which may be far away from one another in a document. Most information extraction systems use sequence models, such as linear-chain conditional random fields, which only have access to a small, local, context when making decisions. I show how to add long-distance links between related words which can be arbitrarily far apart with the document. Experiments show that these long-distance links can be used to improve performance on multiple tasks.

I then move to jointly modeling different layers of information. First I present a sampling-based pipeline. In a typical linguistic annotation pipeline, different components are run one after another, and the best output from each is used as the input to the next stages. The pipeline I present is theoretically equivalent to passing the entire distribution from one stage to the next, instead of just the most likely output. Experimentally, this pipeline did outperform the typical, greedy pipeline, but did not outperform taking the k-best outputs at each stage. I follow this with a full joint model of parsing and named entity recognition. This joint model does not have the directionality constraints inherent in a

pipeline, and both levels of annotation can directly influence and constrain one another. Experiments show that this joint model can produce significant improvements on both tasks. I then show how to further improve the joint model using additional data which has been annotated with only one type of structure, unlike the jointly annotated data needed by the original joint model. The additional data is incorporated using a hierarchical prior, which links feature weights between models for the different tasks.

Lastly, I address the problem of multi-domain learning, where the goal is to jointly model different genres of text (annotated for the same task). This is once again done via a hierarchical prior which links the feature weights between the models for the different genres. Experiments show that this can technique can improve performance across all domains, though, not surprisingly, ones with smaller training corpora improve more.

Acknowledgements

First and foremost: thank you Chris. I really cannot imagine having had a better advisor. I think it has always been pretty clear that I really loved grad school, and you deserve a lot of the credit for that. I've always felt like you were on my side and cared about my success, and that you gave me the right combination of honest constructive criticism, while at the same time giving me confidence in myself and my abilities. You taught me everything I know about research, while also providing me with the level of support that I needed to thrive. I know that it is customary to enumerate all of the research-related skills that you have instilled in me, but I can't, and I won't even try. I will instead say that when I first joined the NLP group, I don't think I knew up from down, and now I feel capable of going out there and taking over the world. I expect to always keep you as my mentor, and as my drinking partner at conferences.

I'm also forever indebted to the other members of my committee. DanJ, you have been like a second advisor to me, and have helped me to always keep an eye on the big picture, and to look for what I can learn from other work, instead of looking for what's wrong with it. Daphne, I think I have learned far more from you than you will ever realize. CS228 was, without a doubt, the best class I have ever taken, and it has had a huge impact on how I approach and evaluate research problems. As I'm sure you know, you have a reputation for for being a no-nonsense kind of person. It took a little while to get used to, but I now think it's one of your best qualities. There was a line in Randy Pausch's last lecture about how it's a bad sign when people stop telling you what you're doing wrong, because it means they don't think you're good enough to be worth the effort. I hope you never stop telling me all the things I'm doing wrong. And, last but not least, is Andrew. You always manage to ask me the right questions, ones that leave me looking at my problem in a new way and

help me better get to the crux of whatever I'm currently working on. I am sad to be leaving all of you.

It is common knowledge that peer quality is one of the most important factors for academic success. The lone genius doesn't exist; in reality we all learn from one another, so having smart peers is very valuable. Thankfully, the NLP group is the best group in the CS department. I feel very lucky to have joined a group full of people who I could learn from, and who I enjoy hanging out with. Teg, you were my older-grad-student-mentor, and you helped me immensely my first few years. You also inspired me to take an active role in the lab, and I strived to serve a similar role for younger students. David Hall, I was always amazed that my favorite person for technical discussion was the lab undergrad, but it was. I can't wait to see what you accomplish during your PhD. Marie, you have always been a wonderful, calming, force for me, and I will miss you. I am excited for what the future holds for me, but doing research outside of the Stanford NLP group is going to be weird.

I wouldn't want to imagine what grad school would have been like without my friends. The Pauls: I'm glad you're both my best friends, and I love you both dearly. Caitlin and Jana, I don't know what I will do without the two of you around. I guess we'll just have to hang out in New York instead! And to the Shady Waffle crew: your wonderful insanity has helped keep me sane.

Finally, there's my family. You guys got the dedication, but I want to say a few more words than is allowed there. Mom and Dad, looking back, a lot of what you did for me when I was growing up makes a lot more sense now. I realize now how you strove to give me every possible opportunity, and to make me feel like I could do anything I put my mind to. It worked! Once I figured out what I wanted to do, you guys had already given me the tools, experiences, and encouragement that I needed to make it happen. And then there's my brother, Jon. We may not have been friends when we were growing up, but I am so glad that we are now. You're the best big brother in the world, and I can't wait to live near you again!

To my brother, Jon, and my parents, Mark and Clare.
I couldn't have done it without you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Why are humans so much better than computers at understanding language? In part, it is because humans naturally employ holistic language processing. They effortlessly keep track of many inter-related layers of *low-level* information, while simultaneously integrating in long-distance information from elsewhere in the conversation or document. In contrast, most NLP research focuses only on individual lower-level tasks, like parsing, named entity recognition, or part-of-speech tagging. Moreover, for the sake of efficiency, researchers modeling these phenomena make extremely strong independence assumptions, which completely decouple these tasks, and only look at local context when making decisions. When examining the output of these systems, it is easy to see places where these incorrect independence assumptions are harming performance, and producing inconsistent outputs. This thesis addresses the problems caused by many of these independence assumptions by presenting several different kinds of joint models for natural language processing: joint models of disparate parts of a document (long-distance modeling), joint models over multiple datasets (domain adaptation), and joint models over multiple types of structure.

The language understanding tasks that people really care about are high-level, semantically-oriented ones: question answering, machine translation, machine reading, speech interfaces for robots and machines, and others that we haven't even thought of yet. To do a good job on any of these high-level tasks, it is critical to start with a good analysis

of the document (or sentence or utterance) of interest. If you wish to do question answering, you will need to identify all the people, locations, and organizations mentioned (*named entity recognition*), and to decide which refer to the same real-world entities (*coreference resolution*). Uncovering the syntactic structure (*parsing*) is essential, since it provides information about how the entities interact with one another, and what actually happened. *Word senses* are also important: by *plant* did the writer mean a living thing that grows out of the ground, or a building where goods are manufactured? Without this kind of information, a question answering system may be able to answer a few simple questions based on substring matching and other heuristic tricks, but will never be able to integrate information from various sources, written by different authors with different writing styles, into a coherent, and correct, answer.

As a field, we have reached a point where models of low-level tasks have reasonably good performance on established datasets. Parsers on the Wall Street Journal portion of the PennTreebank (Marcus et al. 1993) score around 92% (Zhang et al. 2009). The best named entity recognition systems on the popular CoNLL 2003 named entity corpus (Sang and Meulder 2003) score above 90% (Ratinov and Roth 2009). Models of coreference resolution and semantic role labelling have also been steadily improving. So, given the strides that have been made in the field in the past decade, why are we still not very good at the high-level tasks which use these low-level systems as input? There are many answers to this question, and in this thesis I address several aspects of this question.

Firstly, we have the notion of *internal consistency*. When building a model which takes the outputs from several low-level systems as input, it is important that these outputs are coherent and consistent, both internally and in conjunction with one another. In the case of named entity recognition, if a name appears multiple times, and is incorrectly labelled differently in different instances, this will likely cause a high-level system to fail. Because most modern NLP models only look at local context when making decisions, long-distance information of this sort is often ignored entirely, and it is not uncommon to incorrectly label different instances of the same word or phrase differently in different contexts. One could apply the *one-sense-per-discourse* rule (Gale et al. 1992), which is common in word sense disambiguation, to named entity recognition, but oftentimes the wrong label (or "sense") will be picked, making the situation even worse. Additionally, there are often legitimate

reasons to have the same word labeled multiple ways in different contexts, and we do not want to remove that option. One well-known example from the CoNLL 2003 corpus is European sports team names, which are often identical to the name of the team's city. We want documents to be labelled consistently and correctly.

In addition to wanting documents to be labeled consistently for each indiviual type of annotation, we also want *inter-model consistency*, where all of the different levels of annotation are consistent with one another. For example, if a named entity recognition system labels two phrases as different entity types (e.g., one is a person and the other is an organization), but a coreference system claims that they are coreferent, this set of labellings is inconsistent and will also probably cause a high-level system to fail. Nonetheless, this is the most common technique for analyzing text as input to other systems. Researchers assemble components built by other researchers and downloaded from the Internet. Data is run through each component separately, and there is nothing is place to ensure consistency of the outputs.

One solution to the problems associated with this independent approach is to *pipeline* the components together. In this case, the output of one component is used as the input to the next component in the pipeline, and each system is forced to respect the decisions made by previous systems. This method ensures consistency, but has other potential problems. Errors will propagate, and the odds of finding the correct analysis significantly degrades with each new component added to the pipeline. As observed by Singh et al. (2009), if you pipeline together 6 components, each with 90% accuracy, then the final output will be wrong about half the time. This degradation can largely be attributed to the uni-directional flow of information. If the named entity component makes an error, there is no way for the coference component to go back and inform the named entity component that the input it received doesn't make sense. Instead, it will just to the best it can with what it has, and oftentimes the result is even more errors. One could pass the k-best outputs (instead of the 1-best output) as inputs to the next stage, and this will alleviate the problem somewhat, but usually it is not enough. In this dissertation I describe a *sampling pipeline*, which generates and passes samples at each stage, and has several nice theoretical properties. This model also outperformed using a 1-best output at each stage, but it did no better than the k-best pipeline.

This brings us to the solution advocated here: a full, joint model over multiple types of annotations. Like the pipeline approach, a joint model guarantees consistency between the different annotations. It has several other strengths as well, most notably the fact that the different types of information can influence one another. Continuing with our named entity and coreference example, it's not even clear what the proper order should be when using them in a pipeline. Knowing the output from a coreference system would be immensely helpful to a named entity recognizer, because it will know to label coreferent entities identically. Conversely, a coreference resolver would benefit from named entity knowledge, as this can both exclude some entities from being coreferent with one another, while also increasing the likelihood of coreference for other sets of entities. If one were to build a joint model of both of these phenomena, then they would be able to both influence one another in useful ways. This ability for the different tasks to influence each other is in fact one of the best avenues for improving performance on the individual components.

Lastly, because the researchers who work on high-level tasks often use components built by others, they may end up with serious degradations in performance due to *domain drift*. As stated above, the best parser for the PennTreebank scores around 92% when it is evaluated on data from the same source. However, we often want to work with other data which is not financial newswire. It is common to run these components on data downloaded from the Wikipedia and other webpages, blog posts, emails, and other domains, yet they are rarely trained on these genres of text. Indeed, recent work (McClosky et al. 2010) has shown that such a parser can drop 10% F_1 when used on to transcribed speech, and 15% when used on biomedical text. It is critical that we find ways of adapting our models so that they can perform well on many styles of text.

These are precisely the kinds of problems I have tried to solve in this dissertation. I have worked almost exclusively on low-level components, but always with an eye towards the fact that we only create these components to be used as input to higher-level systems. No one writes a named entity recognizer (NER) just so that they can have one running on their laptop: we can all pull out the names of entities when reading newspaper articles without any discernable effort. Named entity recognizers are written because high-level systems often need to know what entities are being mentioned in the text. One of the models presented in this thesis adds long-distance information to a standard NER system

which encourages, but does not force, consistent labelling of entities. I also present models over multiple types of structure – joint parsing and named entity recognition – and a domain adaptation model which works on both structured and unstructured outputs and which improves performance across all domains. Additionally, I give an improved method for building annotation pipelines, and a new model for a discriminative, conditional random field-based parser, which is a critical precursor for much of the work presented.

1.2 Contributions of this thesis

The work presented in this thesis represents a number of contributions to the field of natural language procesing.

- I present the first feature-rich, CRF-based parser which could scale beyond toy sentences. I also show how to use this parser for more than just simple constituency parsing: I use it to build a model of nested named entities, and to build a joint parse and named entity model.

- I present a joint model of parse structure and named entities, and experiments demonstrate that this joint model improves performance on both of the indidivual tasks. The joint model also produces a consistent output, which is more generally useful than the potentially inconsistent outputs produced by independent systems. The joint model, as initally presented, requires data which has been jointly annotated with both types of structure. I later show how to further improve this joint model using data which has been annotated for only a parsing or only named entity recognition.

- The techniques I have chosen to use have also proved influential. My work was amongst the first in NLP to use sampling-based Baysian inference, a tool which has now become commonplace in the community. I was also one of the first advocates for using a hierarchical prior to link related models in NLP. Other researchers have already begin to explore other scenarios in which a hierarchical prior can fruitfully link related models.

1.3 Structure of this thesis

Chapter 2 This chapter contains necessary background information for understanding the remainder of the dissertation. Specifically, it covers several prominent NLP tasks, along with datasets, evaluation. It also contains information on prior models, stochastic optimization, and word classes which are used throughout the document.

Chapter 3 This chapter addresses the problem of how to incorporate long-distance information into modern information extraction (IE) systems. Based on a linear-chain conditional random field (CRF) model, the current state-of-the-art approach to many IE tasks, I show how to add long-distance links while keeping inference tractable. I show that adding these links improves performance on two different tasks, and produces a more consistent output, where identical phrases are likely to be labeled identically, unless there is strong evidence to the contrary. The contents of this chapter are based on Finkel, Grenager, and Manning (2005).

Chapter 4 This chapter presents a method for improving linguistic annotation pipelines. Currently, most researchers who study high-level tasks crudely assemble 1-best pipelines from components written by other researchers and downloaded off of the Internet. This chapter presents a simple, sampling-based approach to annotation pipelines, which is equivalent to passing the entire distribution between different stages in the pipeline. I present results on two different high-level tasks, and compare to both greedy, 1-best pipelines and to *k*-best pipelines. The contents of this chapter are based on Finkel, Manning, and Ng (2006).

Chapter 5 This chapter gradually builds up to a joint model of parsing and named entity recognition. I first present a discriminative, feature-based constituency parser, and an analogous dependency parser. I then show how to use the constituency parser to build a named entity recognizer which can elegantly handle nested named entities. Finally, I provide the necessary modifications to build a joint model of parsing and named entity recognition, including experimental results which show that the joint model helps performance on both

tasks. The contents of this chapter are based on Finkel, Kleeman, and Manning (2008), Finkel and Manning (2009c), and Finkel and Manning (2009b).

Chapter 6 This chapter covers the use of a hierarchical prior for two very different purposes. Hierarchical priors allow the soft-tying of parameters in related tasks, and degree of information sharing can be specified by adjusting hyper-parameters. In this chapter, I use a hierarchical prior to do multi-domain learning (similar to domain adaptation but where performance across all domains is considered) for both named entity recognition and dependency parsing. I also use a hierarchical prior to improve the joint model presented in chapter 5 by linking it with single-task models for both of the individual tasks. The contents of this chapter are based on Finkel and Manning (2009a) and Finkel and Manning (2010)

Chapter 7 In the conclusion to this thesis, I summarize the work presented, and discuss several potential avenues for future work.

Chapter 2

Background

This chapter contains all of the background information necessary for understanding the rest of the material in this dissertation. Specifically, it includes information on parsing and named entity recognition, the two primary NLP tasks covered in this dissertation. It also includes descriptions of several datasets, as well as useful models and algorithms, and some information on repeatedly used word classes.

2.1 Tasks and their standard algorithms

In this section I give an overview of parsing (both constituency and dependency) and named entity recognition. A few other tasks appear in the document in isolated instances (semantic role labelling and recognizing textual entailment, in chapter 4), but they are described where they are used.

2.1.1 Named entity recognition

Named entity recognition (NER) is the task of identifying and classifying names in text. Here is an example sentence, along with its annotation:

> [Jenny Finkel]$_{\text{PER}}$ is a student at [Stanford University]$_{\text{ORG}}$ in [Palo Alto, CA]$_{\text{LOC}}$.

There is some variability in the types of entities extracted, but the three from the example, PERSON (or PER), LOCATION (or LOC), and ORGANIZATION (or ORG), are the three most prevalent. Other common entity types are NUMBER, DATE, TIME, PERCENT, GEO-POLITICAL ENTITY (or GPE) (this one is very similar to LOCATION), and MISC, which is often used as a catch-all for all names which don't fall under one of the categories for a given dataset. It is common practice to give all non-entity words the *background* label O, which stands for *outside* an entity.

Another common task is biomedical NER. BioCreAtIve, one of the first bio-NER shared tasks (Hirschman et al. 2005), had only two entity types, GENE and PROTEIN, but the million word GENIA corpus (discussed below) has many more entity types. Here is an example from GENIA:

> [Kappa B-specific DNA binding proteins]$_{\text{PROTEIN}}$: role in the regulation of [human interleukin-2 gene expression]$_{\text{DNA}}$.

Historically, most NER work has assumed a flat representation, like the examples above. However, the nesting of entities (such as *University of California*, a LOCATION inside an ORGANIZATION or *PEBP2 site*, a PROTEIN inside a DNA), is also covered in this dissertation (section 5.3).

Evaluation

Named entity recognition is typically evaluated on the entity level, not the token (word) level, using precision, recall, and F_1 score. These are computed using the *true positives* (the number of correct entities found by the model), *false positives* (the number of incorrect entities found by the model), and *false negatives* (the number of correct entities missed by the model). *Precision* conveys what percentage of the guessed entities are correct:

$$precision = \frac{true\_positives}{true\_positives + false\_positives} \tag{2.1}$$

Recall conveys what percentage of the true entities were found:

$$recall = \frac{true\_positives}{true\_positives + false\_negatives} \tag{2.2}$$

The F_1 *score* is the harmonic mean of the two:

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \tag{2.3}$$

It is common to report these scores on the individual entity types, and to then average them into an overall score. The background symbol does not count as one of these entity types. Correctly guessing the background symbol for a word counts as a *true negative*, and has no effect on the score. They can be *micro-averaged*, where each entity is given equal weight, or *macro-averaged*, where each category is given equal weight. Micro-averaging is more common, but macro-averaging can be useful when some entity types are rare but important. It is worth noting that there is no partial credit; an incorrect entity boundary is penalized as both a false positive and as a false negative.

Linear-chain conditional random fields (CRFs)

In this section, I will give an overview of linear-chain conditional random fields (CRFs) (Lafferty et al. (2001), or see Sutton and McCallum (2007) for a more recent tutorial), which are the current state-of-the-art for many sequence modeling tasks, including named entity recognition. A CRF is a conditional sequence model which represents the probability of a hidden state sequence given some observations (in our case, the words in a sentence or document). In named entity recognition, the possible entity types vary by dataset and task. These entity types form the set of possible states (or labels), and there is an additional state called O (which stands for OUTSIDE an entity, or not an entity). While the original CRF paper (Lafferty et al. 2001) describes inference in the model in terms of matrix multiplication, subsequent work has opted to describe it in a more general manner consistent with the Markov network literature (see Pearl (1988), Cowell et al. (1999), Wainwright and Jordan (2008) and Koller and Friedman (2009)). The original formulation only describes *first-order* CRFs, which means that labels are only conditioned on one label on either side. However, it is fairly straightforward to extend the model to the higher-order case. Because I only use first-order CRFs, for simplicity the equations I give are for that case, but by simply modifying the cliques to cover larger spans one can easily convert them to be higher-order.

Formally, we have a sequence of labels, represented by a vector $\mathbf{y}$ whose individual

elements y_i indicate the label at the ith position in the sequence. We have a corresponding sequence of observations $\mathbf{x}$. We also have a parameter vector θ of feature weights. Represented graphically, as in figure 3.1, the label sequence forms a *linear chain*; each label is conditionally independent of the other labels in the sequence, given the labels of its neighbors. We have pairwise *cliques* over adjacent nodes, and each clique has a *clique potential function*, also referred to as a *factor table*, which represents the interactions between the nodes in the clique.[1] The probability of a label sequence is given by the equation:

$$\mathrm{P}_{\mathrm{CRF}}(\mathbf{y}|\mathbf{x};\theta) = \frac{1}{Z_{\mathbf{x},\theta}} \prod_{i=1}^{|\mathbf{y}|} \phi(y_{i-1}, y_i|\mathbf{x};\theta) \tag{2.4}$$

where θ is the vector of feature weights, $\phi(y_{i-1}, y_i|x;\theta)$ is the clique potential at position i corresponding to states y_{i-1} and y_i,[2] and $Z_{\mathbf{x},\theta}$ is the *partition function*, which serves as a normalizer:

$$Z_{\mathbf{x},\theta} = \sum_{\mathbf{y}} \prod_{i=1}^{|\mathbf{y}|} \phi(y_{i-1}, y_i|\mathbf{x};\theta) \tag{2.5}$$

We have a feature function $\mathbf{f}(y_{i-1}, y_i, \mathbf{x})$ which computes features over adjacent labels, and can which utilize any and all information available in the observation sequences. We denote the value of feature f_i by $f_i(y_{i-1}, y_i, \mathbf{x})$, and we have a corresponding feature weight θ_i. The clique potential functions take an exponential form:

$$\phi(y_{i-1}, y_i|x;\theta) = \exp\{\mathbf{f}(y_{i-1}, y_i, \mathbf{x}) \cdot \theta\} \tag{2.6}$$

In practice, many features are over only one label (*node features*) and not neighboring pairs of labels (*edge features*). If $g(y_i, \mathbf{x})$ are the node features, and $h(y_{i-1}, y_i, \mathbf{x})$ are the edge features, then $\mathbf{f}(y_{i-1}, y_i, \mathbf{x}) = g(y_i, \mathbf{x}) + h(y_{i-1}, y_i, \mathbf{x})$.

Training a CRF consists of finding the feature weights which maximize the conditional probability of the data. The optimization is typically done on the log-likelihood, instead

[1] A factor table is similar in spirit to an unnormalized probability table, though it would be erroneous to normalize it and then assume that you have the correct marginal distributions for the nodes in the clique. CRFs are *globally normalized*, so information from the rest of the sequence must be taken into account when computing marginal probabilities.

[2] To handle the start condition properly, imagine also that we define a distinguished start state y_0.

of the likelihood itself, because both functions are convex and have the same maximum, and it is much simpler to compute the partial derivatives of the log-likelihood. It is also common practice to put a Gaussian prior[3] over the feature weights (this is often referred to as *regularization*) to encourage them to not get too large when sparse observations might make a feature an apparently strong, or even categorical, predictor. The regularized log-likelihood of a first-order, linear-chain, CRF is given by:

$$\mathscr{L}_{\text{CRF}}(\mathbf{y}, \theta|\mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} \mathbf{f}(y_{i-1}, y_i, \mathbf{x}) \cdot \theta - \sum_{\mathbf{y}'} \sum_{i=1}^{|\mathbf{y}'|} \mathbf{f}(y'_{i-1}, y'_i, \mathbf{x}) \cdot \theta - \frac{|\theta|^2}{2\sigma^2} \tag{2.7}$$

where σ is the variance of the prior, and determines the degree of penalization for feature weights deviating from zero. The partial derivatives of the log-likelihood, with respect to the feature weight parameters, are the actual feature counts in the gold data, minus the expected feature counts in the model using the current feature weights, along with a term for the prior over the feature weights:

$$\frac{\partial \mathscr{L}_{\text{CRF}}}{\partial \theta_j} = \sum_{i=1}^{|\mathbf{y}|} f_j(y_{i-1}, y_i, \mathbf{x}) - E_\theta[f_j|\mathbf{x}] - \frac{\theta_j}{\sigma^2} \tag{2.8}$$

Once we know how to compute the function value and partial derivatives, we can use any number of numerical optimization techniques, including L-BFGS (Liu and Nocedal 1989) and stochastic gradient descent (see section 2.3.2) to find the optimal parameter settings.

2.1.2 Constituency parsing

Parsing is the task of finding the underlying syntactic (or grammatical) structure of a sentence. The output of a parser is a *parse tree* (also sometimes referred to as simply a *parse* or a *tree*). In this thesis I cover two different types of parse trees, constituency trees, introduced in this section, and dependency trees, introduced in the following section. Refer to figure 2.1 for an example of a constituency tree.

[3] L_1 priors are also common, but I used a Gaussian prior for all work in this dissertation.

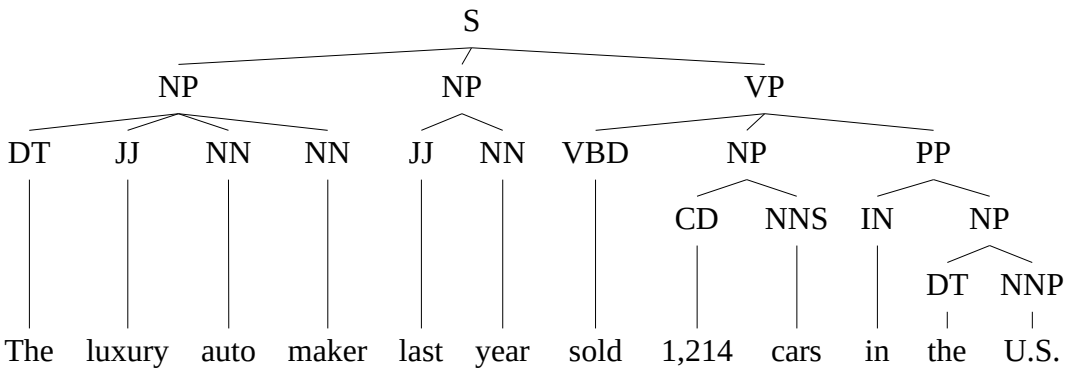


Figure 2.1: An example constituency tree from the PennTreebank.

Phrases The basic unit of a parse tree is a *phrase*, which has an associated *state* (or *label*). For example, *the luxury auto maker* is a *noun phrase* (or NP), and *in the U.S.* is a *prepositional phrase* (or PP). At the leaves of the tree are *parts of speech*, such as *common noun* (or NN) and *determiner* (or DT). Phrases are decomposed into smaller phrases and parts of speech. For instance, *in the U.S.* contains a *preposition* (or IN) and an NP. The *span* of a phrase refers to the set of words it *dominates*.

Lexicon Parsers will contain a *lexicon* which determines what parts of speech are allowed for a given word. There are *open* and *closed* classes of words. Determiners are a closed class – there are a small, known number of determiners, and new words cannot be added to the set. Nouns are an open class – new nouns are being created all the time. Many parsers have elaborate *unknown word models* which help determine the part of speech for a previously unseen word. It is common for the lexicon to return not only the set of possible tags for a word, but also associated probabilities specifying how likely each one is.

Grammar Parsers will contain a context free *grammar* (CFG) which contains rules which determine how phrases can decompose. For example, PP $\rightarrow$ IN NP, tells us that a prepositional phrase can be made by combining a preposition with a noun phrase. When building a *probabilistic context free grammar* (PCFG), each rule has an associated probability, and the probabilities of all rules with the same parent will sum to unity. Once a grammar has been determined, the goal of PCFG parsing is to find the most probable parse

for a given sentence. More broadly, the grammar and lexicon work together to determine the set of parse trees *licensed* by the grammar (the lexicon is often considered to be part of the grammar) for a given sentence.

Binarization Most parsing algorithms rely on dynamic programs which require the grammar to be *binarized*. This involves converting rules with more than two children into multiple, related rules. For instance, NP $\rightarrow$ DT JJ NP (JJ is an adjective) could be converted into NP $\rightarrow$ DT @NP-DT and @NP-DT $\rightarrow$ JJ NP In this case @NP is referred to as an *active state*, as it is only part of an NP, and it is being actively constructed. Including the -DT tells us that the previous child is a DT. The number of previous children included in the state is referred to as the level of *horizontal Markovization*, and is determined by the grammar designer. States can also be augmented with information about ancestors further up in the tree. This is called *vertical Markovization*, or, if only one level is included, *grandparent annotation*. The *split* refers to the point in between the two sub-phrases which compose the phrase.

Evaluation

Parse trees are usually evaluated using precision, recall and F_1 score of *labeled spans*. Scores for parts of speech are often reported separately as just tagging accuracy. It is also common to report numbers for percent of trees which were completely correct; the average number of *crossing brackets*, which are guessed spans which overlap with a correct span, but neither subsume it nor are subsumed by it; and the percent of guessed trees that have no crossing brackets with the gold tree. The *evalb* script[4] is typically used to compute these numbers, and it does some normalization of the input data with respect to punctuation.

2.1.3 Dependency parsing

A dependency grammar is an alternative representation to a phrase structure (constituency) grammar. It directly models the *head* of each phrase, and the other words in the phrase are *dependents* of the head. The head of a phrase is the most important word, and the other

[4] Available at http://nlp.cs.nyu.edu/evalb/.

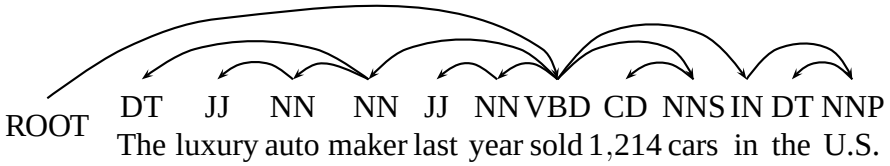


Figure 2.2: An example dependency tree, created by converting the constituency tree in figure 2.1.

words *modify* it. The head of a sentence is typically the main verb. Refer to figure 2.2 for an example of a dependency tree.

Dependency trees can be either *labeled* or *unlabeled*. In the unlabeled case, the tree structure represents which words modify each other, but not the nature of the relationship. The labels represent the type of dependency relation, and examples include *subject*, *apposition*, and *temporal modifier*. All of the dependency parsing work in this dissertation is unlabeled.

Projectivity The link structure connecting dependents to heads will form a tree, with the primary verb as the root. However, sometimes those links are allowed to cross one another, and sometimes they are not. When a tree is *projective* it means that none of the links in the tree cross one another. A *non-projective* tree has crossing links. Some models force projectivity (e.g. Eisner (1996)), and some do not (e.g., McDonald et al. (2005b)). The English language is mostly projective; there are very few instances where links should cross one another. One example of non-projectivity in English is *extraposition*, which is when a phrase is moved from its normal position to the end of the sentence (e.g., *My brother visited who's from New York and is a professional gambler*). It is also common to modify dependency trees to ensure projectivity, and most algorithms for converting constituency trees into dependency trees (discussed in section 2.2.3) do so in a way that forces projectivity. All of the dependency parsing results in this thesis are on English data and assume/enforce projectivity.

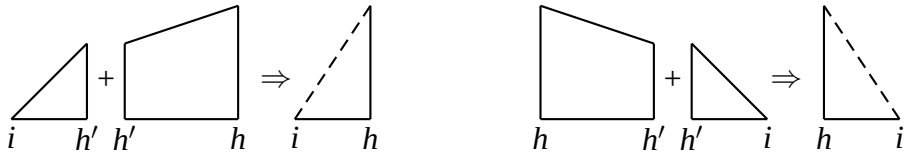
The Eisner dependency parsing algorithm

The discriminative dependency parser I will present in section 5.2.3 relies heavily on the Eisner dependency parsing algorithm (Eisner 1996), which I will briefly review here. The algorithm assumes that you have part-of-speech tags for all words in the sentence, and that you already have a grammar which specifies the probabilities of different parts of speech attaching to one another (*attachment probabilities*), and the probability that a word will stop taking dependents (the *stopping probability*). Given this grammar, and the parts of speech for an input sentence, it will find the most probable projective dependency tree.

At its heart, the algorithm relies on a set of three basic shapes. Different pairs of shapes can be combined to construct new shapes. Each shape has both a left and a right version, depending on whether it covers dependents to the right or to the left. Left and right dependents are conditionally independent of one another given the head word. The basic shapes are given below.

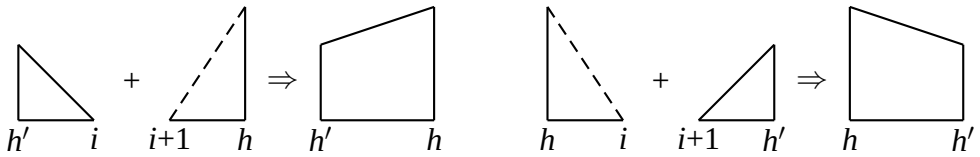
Incomplete left/right triangles These triangles have two indices associated with them: h, which is the index of the head word, and i, which tells us the other end of the span that this triangle covers. The head h is still accepting dependents on the left (right) for a left (right) triangle. It is a left (right) triangle if $i < h$ ($h < i$). Incomplete left triangles are formed by combining a complete left triangle and a left trapezoid. Similarly, incomplete right triangles are formed by combining a right trapezoid with a complete right triangle. The probability of the incomplete triangle is the product of the probabilities of the complete triangle and trapezoid of which it is made. See figure 2.3a for an illustration.

Complete left/right triangles These triangles have the same two indices associated with them as the incomplete triangles do: h, which is the index of the head word, and i, which indicates the other end of the span. The difference here is that we know that h is no longer accepting dependents on the left/right. Once again, it is a left triangle if $i < h$ and a right triangle if $h < i$. A complete left (right) triangle is made out of an incomplete left (right) triangle. The probability of an complete triangle is the product of the probabilities of the incomplete triangle of which is it made and a stopping probability which states how likely it is for the head h to stop collecting dependents on that side. See figure 2.3b for an



(a) Creation of incomplete left/right triangles

(b) Creation of complete left/right triangles



(c) Creation of left/right trapezoids

Figure 2.3: The basic shapes used in the Eisener dependency parsing algorithm, and how they are combined with one another to produce other shapes.

illustration.

Left/Right trapezoids These shapes also contain two indices: h and h', and h' becomes a dependent of h in the formation of the trapezoid. In a left trapezoid, $h' < h$, and h is a head word which can still take dependents on its left (though they must be further to the left than h'), while h' is no longer taking dependents on its right. (For a right trapezoid, invert everything.) A left trapezoid is formed by combining a complete right triangle with an incomplete left triangle. The probability of a trapezoid is the product of the probabilities of the triangles from which it is made, and the attachment probability for attaching h' to h. See figure 2.3c for an illustration.

Now that I have given an overview of the fundamental shapes, I can discuss the actual

algorithm. This is a dynamic programming algorithm based on a chart, much like chart-based PCFG parsing. The chart is initialized with incomplete left and right triangles over each word (so, $h = i$) We then incrementally build up the chart by increasing the span size that we are looking at. First we will attempt to build incomplete triangles and trapezoids by iterating through each possible split point for that span (each internal point of the span, possibly including the span edges as well), combining the appropriate shapes from the left and right portions, and seeing which split point gives the highest probability. The shapes and their probabilities are then added to the chart for that span. We then construct complete triangles for that span from the incomplete triangles, and add them to the chart, before moving to the next larger span.

The algorithm as just described gives the *max inside scores*. The max inside score for an incomplete right triangle tells you the probability of the most likely subtree structure which spans h to i and has h as the head. What if, instead, you wanted to know the total probability of all possible subtrees which span h to i and have head h? In that case, when constructing the chart, when iterating through the possible ways to construct a shape (the possible split points) you would add the probabilities for each option instead of taking the one with the highest probability. This is called the *summed inside score*. Why would you want the summed inside score? Because when combined with the *summed outside score* you can get the overall probability that i attaches to h, conditioned on the other words and parts of speech in the sentence. This probability will come in handy when I cover discriminative dependency parsing (section 5.2.3), because we will need to compute partial derivatives which require this number. Outside scores are computed after inside scores, and are done top down instead of bottom up. The outside score for a shape in the chart is constructed by looking at each shape it can be a child of (so, each shape that it helps to form by (potentially) combining with another shape). The outside score of the parent is multiplied by the inside score of the sibling (if one exists), along with the attachment/stopping probability (if one was used when constructing the parent during the inside pass). This is done for each possible parent (and sibling) and the probabilities are summed to get the outside score for the chart entry. PCFGs have an analogous inside/outside algorithm, which is discussed in more depth in Manning and Schütze (1999).

Evaluation

Dependency parsing is scored using attachment accuracy. Every word has a parent in both the correct and guessed trees,[5] and so the accuracy is simply the percent of predicted parents that are correct. In labeled dependency parsing, one can report both labeled and unlabeled attach accuracy, which (respectively) do and do not take the label into consideration when determining if an attachment is correct. Because this thesis only includes unlabeled dependency parsing, I only present unlabeled accuracy numbers.

2.2 Data

2.2.1 Named entity recognition

Because named entity recognition has been around for a long time, it is a well-supported task with a lot of available datasets. Many of them are used in this dissertation, and I briefly describe them below. Please see table 2.1 for some summary statistics on them.

CoNLL This dataset originates from the shared task at the 2003 meeting of the Conference on Computational Natural Language Learning (CoNLL). It contains British newswire, annotated with PERSON, LOCATION, ORGANIZATION, and MISC. This is probably the most popular NER dataset. For more information, please see Sang and Meulder (2003).

MUC-6 This dataset originated from the 1995 meeting of the Message Understanding Conference (MUC-6). It contains American newswire, and has been annotated with PERSON, LOCATION, ORGANIZATION, DATE, TIME, PERCENT, and NUMBER. Historically, this shared task jump-started research on named entity recognition (prior MUCs did not have NER shared tasks). For more information, please see Sundheim (1996).

MUC-7 This dataset originated from the 1998 meeting of the Message Understanding Conference (MUC-7). Like its predecessor, it contains American newswire, and is annotated with the same entity types as MUC-6. For more information, please see Chinchor

[5] The root verb has a "fake" parent that is ROOT.

(1998).

GENIA This biomedical named entity dataset is part of the larger GENIA project, which also contains resources for parsing, coreference resolution, part-of-speech tagging, and other types of annotations for biomedical data. There is also an associated ontology of terms and events. The data comes from 2000 Medline abstracts, and has been annotated with 36 different entity types. The data does contain nested entities, and is also annotated with parts of speech. The corpus does not have an official train/dev/test split, though the GENIA part-of-speech tagger (Tsuruoka et al. 2005) has been evaluated using the first 90% of the data for training, and the final 10% for testing. During development I further subdivided the training set into a devtrain and devtest set, but table 2.1 only contains information for this semi-official split. For more information, please see Ohta et al. (2002) and Kim et al. (2003).

AnCora This corpus contains Spanish and Catalan newspaper text, and a subset of the data was used in the SemEval 2007 Task 9 (Márquez et al. 2007b). The data contains nested entities, and is also annotated with part-of-speech tags, parse trees, semantic roles and word senses. The corpus annotators made a distinction between *strong* and *weak* entities. They define *strong* named entities as "a word, a number, a date, or a string of words that refer to a single individual entity in the real world." If a strong entity contains multiple words, it is collapsed into a single token. *Weak* named entities, "consist of a noun phrase, being it simple or complex" and must contain a *strong* entity.[6] Figure 2.4 shows an example from the corpus with both strong and weak entities. The entity types present are PERSON, LOCATION, ORGANIZATION, DATE, NUMBER, and OTHER. Weak entities are very prevalent; 47.1% of entities are embedded. This corpus was also not split into train and test sections, so I did so myself, and these are the numbers reported in table 2.1. For Spanish, files starting with 7–9 were the test set, 5–6 were the development test set, and the remainder were the development train set. For Catalan, files starting with 8–9 were the test set, 6–7 were the development test set, and the remainder were the development train set.

[6] Arguably, this represents a misunderstanding of the term "*named* entity", and weak named entities should just be termed *entities* or *referential expressions*.

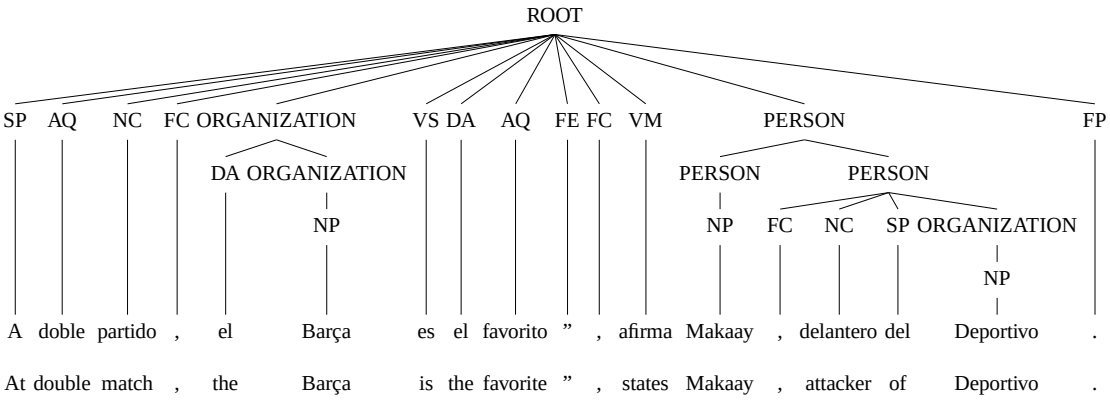


Figure 2.4: A named entity annotated sentence from the AnCora corpus, along with its English translation.

Corpus	Genre	# Entity Types	# Train Tokens	# Dev Tokens	# Test Tokens
CoNLL 2003	British newswire	4	203K	51K	46K
MUC-6	American newswire	7	165K	14K	15K
MUC-7	American newswire	7	90K	104K	65K
GENIA	biomedical abstracts	36	450K	–	50K
AnCora Spanish	Spansh newspaper	6	459K	–	57K
AnCora Catalan	Catalan newspaper	6	432K	–	49K

Table 2.1: Summary statistics for the named entity datasets used in this dissertation.

For both, the development train and test sets were combined to form the final train set. Due to the sub-splitting of the training set for development (meaning there is no additional data purely for development), I only report the train and test set sizes. For more information, please see Martí et al. (2007).

2.2.2 Constituency parsing

The most common constituency parsing dataset, by far, is the PennTreebank. It contains approximately one million words (forty thousand sentences) of parsed data from the Wall Street Journal. It is divided into sections, and sections $2-21$ are used for training, 22 is used for development, and 23 is used for testing. For more information, please see Marcus et al. (1993).

2.2.3 Dependency parsing

Data for dependency parsing comes from two sources: dependency tree datasets and conversion of constituency trees. In recent years the CoNLL shared task has focused on dependency parsing (Buchholz and Marsi 2006, Nivre et al. 2007), and several datasets have been created as a result. Alternatively, dependency trees are constructed out of constituency trees via a set of *head percolation rules*. In this case, when looking at a one level subtree within a tree (a parent and its children), the rules will determine which child is the head of the phrase, and will then percolate that child's head up to the parent. For instance, if you have S $\rightarrow$ NP VP (a sentence composed of a noun phrase a verb phrase) then the VP will be the head of the phrase, and its head will be percolated up. If that VP was constructed by a VP $\rightarrow$ VB NP (a verb phrase composed of a verb and a noun phrase), then the VB would have been the head of the VP, and would be percolated up to the S. The head of both NPs (the one under the S and the one under the VP) would both be dependents of the VB. The two commonly used sets of head percolation rules are Collins (2003) and Yamada and Matsumoto (2003). The two are quite similar, and in this dissertation I used Collins' rules.

2.2.4 OntoNotes

Many of the joint modeling experiments in this dissertation utilize the recently developed OntoNotes Corpus (Hovy et al. 2006). Its creation has been a joint effort by between BBN Technologies, the University of Colorado, the University of Pennsylvania, and the University of Southern California's Information Sciences Institute, and the project leaders describe it as "a large, multilingual richly-annotated corpus constructed at 90% inter-annotator agreement." The project is a work-in-progress and aims to fill many important gaps in the currently available set of annotated corpora. It contains data of a wide variety of genres in English, Chinese and Arabic, with a final goal of one million words each for English and Chinese and a half million words of Arabic.[7] The most exciting aspect of the corpus, in my opinion, is the fact that it has been annotated with multiple layers of information, including constituency trees, predicate structure, word senses, coreference, and named entities. Given the quantity and variety of data, combined with the many levels

[7] Most of this data is *not* in the form of parallel corpora, though a small portion is.

	Training		Testing	
	Range	# Sent.	Range	# Sent.
ABC	0–55	1195	56–69	199
MNB	0–17	509	18–25	245
NBC	0–29	589	30–39	149
PRI	0–89	1704	90–112	394
VOA	0–198	1508	199–264	385

Table 2.2: Training and test set sizes for the five OntoNotes datasets in sentences. The file ranges refer to the numbers within the names of the original OntoNotes files.

of annotation, OntoNotes really has the potential to have a large impact on NLP research. The PennTreebank catalyzed research in parsing technology, and OntoNotes gives us the ability to build high-quality models, trained on large quantities of data, over many of the key semantic structures necessary for high-level understanding.

The experiments in this dissertation use Release 3.0 of the data (with the exception of the dependency parsing work, which uses Release 2.0), though it is worth noting that Release 4.0 is now available. I also limited experiments to the English portion of the data, and used the ABC, CNN, MNB, NBC, PRI, and VOA sections, which represent a mix of speech and newswire data. Table 2.2 gives the number of training and test sentences in each of the sections. I used the parse and named entity annotations, and discarded the remaining levels, though I would like to include them in future models. While OntoNotes is an excellent and much-needed resource, it is not perfect. The different levels of annotation were done by different parties, and occasionally inconsistencies were found. In appendix A, I outline the modifications I made to the data before using it.

Named entity types

The data has been annotated with eighteen types of entities. Many of these entity types do not occur very often, and coupled with the relatively small amount of data, make it difficult to learn accurate entity models. Examples are WORK_OF_ART, PRODUCT, and LAW. Early experiments showed that it was difficult for even a standard named entity recognizer, based on a state-of-the-art CRF, to learn these types of entities.[8] As a result, I decided to merge

[8]The difficulties were compounded by somewhat inconsistent and occasionally questionable annotations. For example, the word *today* was usually labeled as a DATE, but about 10% of the time it was not labeled as

all but the three most dominant entity types into into one general entity type called MISC. The result was four distinct entity types: PERSON, ORGANIZATION, GPE (geo-political entity, such as a city or a country, and similar to LOCATION in other corpora), and MISC.

2.3 Optimization

Many of the models in this thesis, and in NLP in general, require doing an optimization in order to estimate parameters (usually feature weights).

2.3.1 Deterministic optimization methods

Many models for NLP tasks have convex objective functions. For a long time researchers used generalized iterative scaling (Darroch and Ratcliff 1972) (e.g., Ratnaparkhi (1997)) and conjugate gradient methods to do the optimization (e.g., Toutanova et al. (2003)). Nowadays it is common to use L-BFGS (Liu and Nocedal 1989), a limited-memory quasi-newton method. A lot of the experiments in this dissertation use L-BFGS, but this usually requires repeatedly doing inference on an entire corpus, often hundreds of times. For some models, inference is fast enough that iterating over the corpus this many times is not a problem. But for other models in this dissertation, doing inference hundreds of times is just not computationally feasible. In those cases, I turned to stochastic optimization techniques, described in the next section, since they require many fewer passes through the data, and can even find fairly decent parameter settings after passing through the data only once.

2.3.2 Stochastic optimization methods

Stochastic optimization methods have proven to be extremely efficient for the training of models involving computationally expensive objective functions (Vishwanathan et al. 2006), like several we will encounter in this dissertation. In fact, the on-line backpropagation learning used in the neural network parser of Henderson (2004) is a form of stochastic gradient descent. Standard deterministic optimization routines such as L-BFGS make little

anything. I also found several strange WORK OF ARTs, including the *Stanley Cup* and the *U.S.S. Cole*.

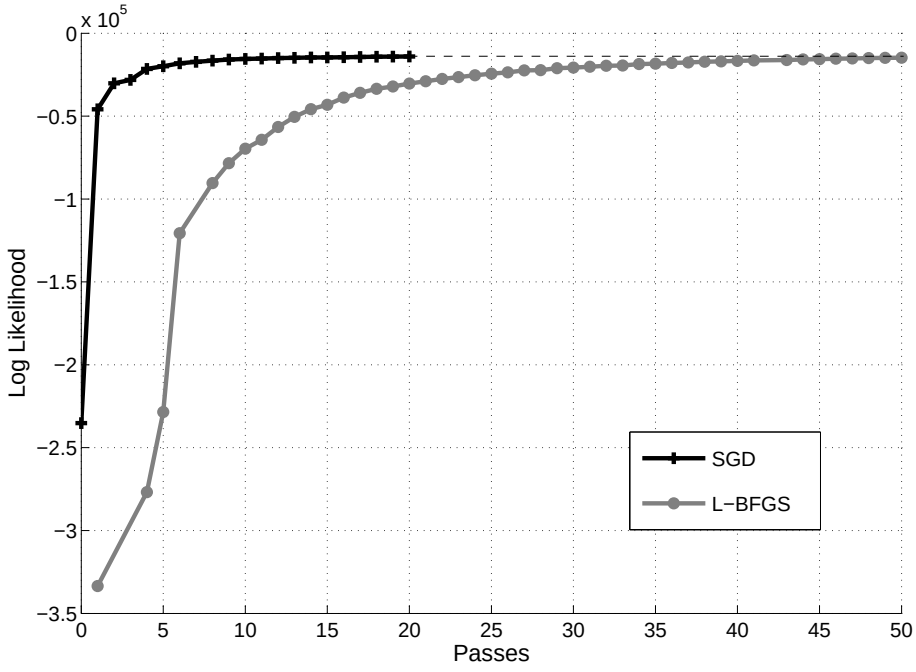


Figure 2.5: WSJ15 objective value for L-BFGS and SGD versus passes through the data. SGD ultimately converges to a lower objective value, but does equally well on test data.

progress in the initial iterations, often requiring several passes through the data in order to satisfy sufficient descent conditions placed on line searches. In our experiments stochastic gradient descent (SGD) converged to a worse objective function value than L-BFGS, however it required far fewer iterations (see figure 2.5) and achieved comparable test set performance to L-BFGS in a fraction of the time. One early experiment on WSJ15 showed a seven times speed up.

Stochastic function evaluation

Utilization of stochastic optimization routines requires the implementation of a stochastic objective function. This function, $\hat{\mathscr{L}}$ is designed to approximate the true function $\mathscr{L}$ based off a small subset of the training data represented by $\mathscr{D}_b$. Here b, the batch size, means that $\mathscr{D}_b$ is created by drawing b training examples, with replacement, from the training set $\mathscr{D}$.

With this notation we can express the stochastic evaluation of the function as $\hat{\mathscr{L}}(\mathscr{D}_b;\theta)$. This stochastic function must be designed to ensure that:

$$\mathrm{E}\left[\sum_i^n \hat{\mathscr{L}}(\mathscr{D}_b^{(i)};\theta)\right] = \mathscr{L}(\mathscr{D};\theta)$$

Note that this property is satisfied, without scaling, for objective functions that sum over the training data, as it is in our case, but any priors over the parameters must be scaled down by a factor of $b/|\mathscr{D}|$. The stochastic gradient, $\nabla\mathscr{L}(\mathscr{D}_b^{(i)};\theta)$, is then simply the derivative of the stochastic function value.

Stochastic gradient descent

SGD was implemented using the standard update:

$$\theta_{k+1} = \theta_k - \eta_k \nabla\mathscr{L}(\mathscr{D}_b^{(k)};\theta_k)$$

And employed a gain schedule in the form

$$\eta_k = \eta_0 \frac{\tau}{\tau + k}$$

where parameter τ was adjusted such that the gain is halved after five passes through the data.

2.4 Word classes

Most of the models in this thesis use discriminative parameter estimation and make extensive use of features. When creating these features I often augmented the words in the data with two additional pieces of information: a distributional similarity cluster, and a *word shape*, which encodes orthographic information. These two types of word classes are discussed in this section.

2.4.1 Distributional similarity clusters

Many of the models discussed in this dissertation make use of distributional similarity clusters. There are many popular methods for clustering words – the Brown algorithm (Brown et al. 1992) has been used extensively (Liang 2005, Koo et al. 2008, Turian et al. 2010), and clusters based on deep-learning (Collobert and Weston 2008) are also beginning to gain popularity (Turian et al. 2010). I chose to use clusters output from Alexander Clark's software which he distributes on his webpage,[9] and which is based on Clark (2003). I chose his model because the code is easy to use, and the default settings give good results.

The exact same clusters were used whenever I discuss the use of distributional similarity clusters, except where explicitly noted.[10] They were trained on approximately 275 million words. This included the 100 million word British National Corpus (BNC 2007), a random subset of the EnglishGigaword corpus (Graff et al. 2007), and the words in the CoNLL 2003 (Sang and Meulder 2003), MUC-6 (Sundheim 1996), and MUC-7 (Chinchor 1998) named entity corpora and the PennTreebank (PTB) (Marcus et al. 1993). When I first experimented with the clustering software, I initially trained the clusters using approximately 40 million words, and found the cluster quality to be much worse. I also tried both 40 clusters (since this is roughly the number of part-of-speech tags in the PTB) and 200 clusters, and found that using 200 clusters resulted in more useful clusters. When I added cluster-based features to the state-of-the-art Stanford named entity recognizer (Finkel et al. 2005), I found that, depending on the dataset, there was a 10%–25% reduction in error. This is comparable to the results in Miller et al. (2004), the first paper to my knowledge to leverage features based on distributional similarity tags.

I used these clusters for two very different purposes. The first use was the creation of cluster-based features, usually modeled after features which use part-of-speech tags as input. The second use was to restrict possible labels for a given word. Further details are given in the appropriate sections.

[9] http://www.cs.rhul.ac.uk/home/alexc/

[10] New clusters were needed for experiments using biomedical data and non-English data.

word		word shape
Jenny	⇒	Xxxx
beta-carotene	⇒	g-xxx
McDonald	⇒	XxXxxx
CD28-responsive	⇒	XXdd-xxx

Table 2.3: Examples of words and their corresponding word shapes..

2.4.2 Word shapes

The *word shape* reduces a word to a summary of its orthographic information, including information on letters, numbers, punctuation, etc. To convert a word into its word shape, the following rules are applied:

1. Spelled out versions of Greek letters are replaces with ‘g.’
2. Digits are replaced with ‘d.’
3. Uppercase letters are replaced with ‘X.’
4. Lowercase letters are replaced with ‘x.’
5. Punctuation remains.
6. Instances where the same character (g/d/x/X) is repeated more than three times in a row are truncated to only include the first three instances.

For several examples of words and their corresponding shapes, please see table 2.3.

Chapter 3

Using long-distance information efficiently

3.1 Introduction

Named entity recognition (introduced in section 2.1.1)is often modeled using a linear-chain CRF, which makes decisions using only a small window of local context. When deciding if a particular word is the name of an person, the model can condition on the labels of the surrounding words, but not other, potentially more informative, instances of that word (or related words) elsewhere in the document. Oftentimes this is sufficient; the neighboring words and labels contain enough information to properly identify the entity class for that word. Sometimes, however, words appear in locally ambiguous contexts. If you look at the example in figure 3.1, you will see that the word *Tanjug* appears twice. In the first case it is modified by the phrase *news agency*, which should be sufficient to confidently conclude that it is an organization. In the second case, it is followed by the word *said*, which is ambiguous, because both people and organizations can say things. The second occurrence of the token *Tanjug* is in fact mislabeled by my CRF-based NER system, because people are more likely to say things than organizations are to say things. However, the probability distribution on the label for that token is not peaked; the model is much less certain about this decision than it is about the decision to label the first instance an ORGANIZATION. This error should be correctable if we can incorporate information about the previous decision

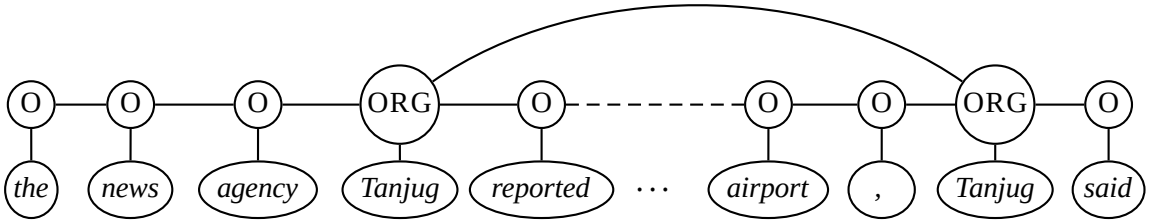


Figure 3.1: An example of the label consistency problem excerpted from a document in the CoNLL 2003 English dataset. Correct named entity tags are shown above the words: ORG indicates organization and O indicates a non-entity.

made in the less ambiguous context.

Despite the potential usefulness of this long-distance information, most statistical models currently used in natural language processing represent only local structure, because this allows them to be evaluated using exact inference techniques. Although this constraint is critical in enabling tractable model inference, it is a key limitation in many tasks, since natural language contains a great deal of non-local structure. A general method for solving this problem is to relax the requirement of exact inference, substituting approximate inference algorithms instead, thereby permitting tractable inference in models with non-local structure. One such technique is *Gibbs sampling*, a simple Markov chain Monte Carlo (MCMC) algorithm that can be used for appropriate inference in any factored probabilistic model, including sequence models (Geman and Geman 1984). By using Gibbs sampling, it is possible to add non-local structure to factored statistical models of language. Although Gibbs sampling is widely used elsewhere, until recently there has been extremely little use of it in natural language processing. The only uses in NLP, prior to the publication of the work described in this chapter, of which I am aware, are Kim et al. (1995), Della Pietra et al. (1997) and Abney (1997). In this chapter, I use it for inference after adding non-local dependencies to conditional random field (CRF)-based sequence models for two different information extraction tasks: named entity recognition (NER) and a template filling task.

The two different tasks benefited from modeling different types of non-local structure. For named entity recognition, I included a model of label consistency, illustrated in figure 3.1, and discussed above. The basic idea is that identical words and phrases should be labeled consistently. We do not necessarily want to enforce this as a hard constraint, since there are occasionally good reasons for the same word to be labeled in multiple ways within

the same document, but we would like to encourage it as a soft constraint. I also used the same technique, but with a different type of long-distance model, to improve performance in a template filling task. Here, the input is a seminar announcement, and the goal is to extract the name of the speaker, and the location, start, and end times of the seminar. It is common to use a sequence model for this type of task, but the problem that arises is that the model will often label multiple, different names which appear in the document as the speaker. For instance, it may also label the person hosting the speaker as the speaker, because the model primarily learns what peoples' names look like. In this instance, we want to add long-distance links which encourage (or even force) the model to pick only one answer for each of the slots. We can additionally add long-distance links which encode the fact that the start time should be before the end time, and that if only one time is listed it is likely to be the start time and not the end time.

Linear-chain CRFs (introduced in section 2.1.1) are a prominent approach to these types of information extraction tasks. These models (and their predecessors, hidden Markov models (HMMs) (Leek 1997, Freitag and McCallum 1999) and maximum entropy Markov models (MEMMs) (Borthwick 1999, McCallum et al. 2000)) encode the Markov property: decisions about the state at a particular position in the sequence can depend only on a small local window. It is this property which allows tractable computation, as both the Viterbi and forward-backward algorithms necessary for inference and training critically rely on this property.

In this chapter, I show how to efficiently incorporate constraints of these forms into a CRF model by using Gibbs sampling instead of the Viterbi algorithm as the inference procedure. At training time I build two separate models, the first is a linear-chain CRF, trained in the standard manner, and the second is a task-specific long-distance model. At test time the two models are combined as a product-of-experts (Hinton 2000). This is where I use Gibbs sampling for inference, as the combined model is a linear-chain with added long-distance links. I present experiments on both tasks, and demonstrate that this technique yields significant improvements.

3.2 Related work

Several authors have successfully incorporated a label consistency constraint into probabilistic named entity recognition systems. Mikheev et al. (1999), Finkel et al. (2004) and, more recently, Krishnan and Manning (2006) incorporate label consistency information by using ad-hoc multi-stage labeling procedures that are effective but special-purpose. Curran and Clark (1999) and Malouf (2002) use a different technique to incorporate label consistency into an MEMM. They condition the label of a token at a particular position on the label of the most recent previous instance of that same token in a prior sentence of the same document. Note that this violates the Markov property: the decision at a position is conditioned on a state that is arbitrarily far back in the sequence. To allow this, the authors slightly relax the requirement of exact inference. Instead of finding the maximum likelihood sequence over the entire document, they classify one sentence at a time, allowing them to condition on the maximum likelihood sequence of previous sentences. This approach is quite effective for enforcing label consistency in many NLP tasks, precisely because entities are generally most carefully described when they are first introduced in a text. Nevertheless, it permits a forward flow of information only, which is not sufficient for all cases of interest. Chieu and Ng (2002) and, more recently, Vilain et al. (2009) propose a solution to this problem. Though they use different base models (a multi-class logistic classifier which does not model sequence information, and a linear-chain CRF, respectively), the basic idea is the same for both. For each token, they define additional features taken from other occurrences of the same token in the document. This approach has the added advantage of allowing the training procedure to automatically learn good weightings for these "global" features relative to the local ones. However, this approach cannot easily be extended to incorporate other types of non-local structure, such as the case of template consistency that we discuss below.

Another highly relevant piece of prior work is that of Bunescu and Mooney (2004). They use a *relational Markov network* (RMN) (Taskar et al. 2002) to explicitly model long-distance dependencies, though they do not represent sequence information as in a linear-chain CRF. Unfortunately these dependencies must be defined in the model structure

before doing any inference, and so the authors are forced to pre-generate all possible candidate entities by using crude heuristic part-of-speech patterns, and then add dependencies between these text spans using *clique templates*. This generates a extremely large number of overlapping candidate entities, and dependencies between them, which then necessitates additional templates to enforce the constraint that text subsequences cannot both be different entities, something that is more naturally modeled by a CRF. The authors also used *loopy belief propagation* and a voted perceptron for approximate learning and inference, and noted that this led to problems with convergence.

Sutton and McCallum (2004) have the most similar prior work on long-distance dependencies. They introduce *skip-chain CRFs*, which maintain the underlying CRF sequence model (which Bunescu and Mooney (2004) lack) while adding *skip edges* between distant nodes, and use it to represent relations between entities in an information extraction task. The dependency arcs which they add to the CRF allow the long distance flow of information. The decision of which nodes to connect is also made heuristically, and because the authors chose to connect all pairs of identical capitalized words. They also utilize loopy belief propagation for approximate learning and inference.

While the technique I used is similar mathematically and in spirit to the above approaches, it differs in some important ways. Like Sutton and McCallum (2004), but unlike Bunescu and Mooney (2004), my method maintains a sequence model structure, while adding long-distance conditioning influences, and hence does not need to pre-generate candidates and then enforce no-overlap constraints. The model is implemented by adding additional constraints into the model at test time, and does not require the preprocessing step necessary in the two previously mentioned works. This allows for a broader class of long-distance dependencies, because one does not need to make any initial assumptions about which nodes should be connected, and is helpful when you wish to model relationships between nodes which are the same class, but may not be similar in any other way. For instance, in the CMU seminar announcements dataset, one of the labels is START_TIME. However, the announcement may also mention multiple times (for example, times when the speaker is available to meet with students) and these times may also be labeled by the CRF as a START_TIME. With the proposed technique, we can penalize the model is multiple, inconsistent times are all labeled as a START_TIME. This type of constraint cannot

easily be modeled in an RMN or a skip-chain CRF, because it requires the knowledge that both entities are given the same class label. But, by adding the constraint that all start times should be coreferent, it may be possible to fix this error.

My model also allows dependencies between multi-word phrases, and not just single words. Additionally, the model can be applied on top of a pre-existing trained sequence model. As such, it does not require complex training procedures, and can instead leverage all of the established methods for training high accuracy sequence models. It can indeed be used in conjunction with any statistical hidden state sequence model: HMMs, MEMMs, CRFs, or even heuristic models. Third, my technique employs Gibbs sampling for approximate inference, a simple and probabilistically well-founded algorithm. Gibbs sampling is generally better suited to estimating a probability distribution than to finding the most likely label sequence. However, through the use of simulated annealing, I did not find this to be a problem for our simple linear chain. As a consequence of these differences, my approach is easier to understand, implement, and adapt to new applications.

3.3 Approach

The model is a product of experts, where the two experts are a linear-chain CRF, which does not include any long-distance component, and a consistency model, which explicitly models long-distance relationships. CRFs were introduced in section 2.1.1. The CRFs used in this chapter were trained in the completely standard manner, as discussed when they were introduced. The CRF features that I used are outlined in table 3.1.

In this section, I will first discuss the general form of the long-distance consistency model. I will then discuss the use of Gibbs sampling for inference in a sequence model, and then how to apply the technique to our product-of-experts model.

3.3.1 Models of non-local structure

Our models of non-local structure are themselves just sequence models, defining a probability distribution over all possible state sequences. It is possible to flexibly model various forms of constraints in a way that is sensitive to the linguistic structure of the data (e.g., one

Feature	NER	TF
Current Word	✓	✓
Previous Word	✓	✓
Next Word	✓	✓
Current Word Character n-gram	all	length ≤ 6
Current POS Tag	✓	
Surrounding POS Tag Sequence	✓	
Current Word Shape	✓	✓
Surrounding Word Shape Sequence	✓	✓
Presence of Word in Left Window	size 4	size 9
Presence of Word in Right Window	size 4	size 9

Table 3.1: Features used by the CRF for the two tasks: named entity recognition (NER) and template filling (TF).

can go beyond imposing just exact identity conditions). I illustrate this by modeling two forms of non-local structure: *label consistency* in the named entity recognition task, and *template consistency* in the template filling task. One could imagine many ways of defining such models; for simplicity I use the form

$$\mathrm{P}_M(\mathbf{y}|\mathbf{x}) \propto \prod_{\lambda \in \Lambda} \theta_\lambda^{\#(\lambda,\mathbf{y},\mathbf{x})} \tag{3.1}$$

where the product is over a set of violation types Λ, and for each violation type λ we specify a penalty parameter θ_λ. The exponent $\#(\lambda, \mathbf{s}, \mathbf{o})$ is the count of the number of times that the violation λ occurs in the state sequence $\mathbf{s}$ with respect to the observation sequence $\mathbf{o}$. This has the effect of assigning sequences with more violations a lower probability. The particular violation types are defined specifically for each task, and are described in sections 3.4.1 and 3.5.2.

This model, as defined above, is not normalized, and clearly it would be expensive to do so. As we will see in the discussion of Gibbs sampling, this will not actually be a problem for us.

3.3.2 Gibbs sampling for inference in sequence models

In hidden state sequence models such as CRFs (and HMMS and MEMMs), it is standard to use the Viterbi algorithm, a dynamic programming algorithm, to infer the most likely hidden state sequence given the input and the (parameterized) model (see, e.g., Rabiner (1989)). Although this is the only tractable method for exact computation, there are other methods for computing an approximate solution. Monte Carlo methods are a simple and effective class of methods for approximate inference based on sampling. As an illustration of such methods, consider the following (extremely) naive method for performing inference. Imagine we have a hidden state sequence model which defines a probability distribution over state sequences conditioned on any given input. With such a model M we should be able to compute the conditional probability $\mathrm{P}_M(\mathbf{y}|\mathbf{x})$ of any state sequence $\mathbf{y} = \{y_0, \ldots, y_N\}$ given some observed input sequence $\mathbf{x} = \{x_0, \ldots, x_N\}$. To then find the sequence with the maximum conditional probability, we could sample a large number of random sequences from the uniform distribution, score them all with the distribution given by the model, and output the sequence with the highest score as our most likely state sequence. The obvious drawback of this approach is that the space of possible state sequences is large (exponential in the sequence length) and so one is unlikely to happen upon a sequence anywhere near the optimal one. Alternatively, one can sample sequences from the conditional distribution defined by the model. These samples are likely to be in high probability areas, increasing our chances of finding the maximum. The challenge is how to sample sequences efficiently from the conditional distribution defined by the model.

Gibbs sampling provides a solution (Geman and Geman 1984). Gibbs sampling defines a Markov chain in the space of possible variable assignments (in this case, hidden state sequences) such that the stationary distribution of the Markov chain *is* the joint distribution over the variables. Thus it is called a Markov chain Monte Carlo (MCMC) method; see Andrieu et al. (2003) for a good MCMC tutorial. In practical terms, this means that we can walk the Markov chain, occasionally outputting samples, and that these samples are guaranteed to be drawn from the target distribution. Furthermore, the chain is defined in very simple terms: from each state sequence $\mathbf{y}^{(t)}$ we can only transition to a new state sequence $\mathbf{y}^{(t+1)}$ obtained by changing the state at any one position i, and the distribution

over these possible transitions is just

$$\mathrm{P}_G(\mathbf{y}^{(t)}|\mathbf{y}^{(t-1)}) = \mathrm{P}_M(y_i^{(t)}|\mathbf{y}_{-i}^{(t-1)}, \mathbf{x}). \tag{3.2}$$

where $\mathbf{y}_{-i}$ is all states except y_i. In other words, the transition probability of the Markov chain is the conditional distribution of the label at the position, given the rest of the sequence. This quantity is easy to compute in any Markov sequence model. One easy way to walk the Markov chain is to randomly initialize the labels at each point, and then loop through the positions i from 1 to N, and for each one, to resample the hidden state at that position from the distribution given in equation 3.2.[1] By outputting complete sequences at regular intervals (such as after resampling all N positions), we can sample sequences from the conditional distribution defined by the model. However, before we begin to regularly output samples, we can allow the chain to *mix*. Because we started from an arbitrary point, the first samples will be of particularly bad quality. If we were trying to estimate the shape of the probability distribution, this would be problematic. However, because we can compute the model likelihood of each sample, and pick the sample with the highest probability (this is discussed in the following paragraph), there is no real harm in including these early samples. They are unlikely to have high likelihood, and correspondingly, they are unlikely to be the sample that gets selected, but if one of them does end up being the most likely sample, then there is no reason to not use it.

The process just described is still gravely inefficient. Random sampling may be a good way to estimate the shape of a probability distribution, but it is not an efficient way to do what we want: find the maximum. However, we cannot just transition greedily to higher probability sequences at each step, because the space is extremely non-convex. We can, however, borrow a technique from the study of non-convex optimization and use *simulated annealing* (Kirkpatrick et al. 1983). Geman and Geman (1984) show that it is easy to modify a Gibbs Markov chain to do annealing; at time t we replace the distribution in equation 3.2 with

$$\mathrm{P}_A(\mathbf{y}^{(t)}|\mathbf{y}^{(t-1)}) = \frac{\mathrm{P}_M(y_i^{(t)}|\mathbf{y}_{-i}^{(t-1)}, \mathbf{x})^{1/c_t}}{\sum_j \mathrm{P}_M(y_j^{(t)}|\mathbf{y}_{-j}^{(t-1)}, \mathbf{x})^{1/c_t}} \tag{3.3}$$

[1] Instead of looping through the positions, one could also repeatedly randomly sample positions.

Inference	CoNLL	Seminars
Viterbi	85.51	91.85
Gibbs	85.54	91.85
Sampling	85.51	91.85
	85.49	91.85
	85.51	91.85
	85.51	91.85
	85.51	91.85
	85.51	91.85
	85.51	91.85
	85.51	91.86
Mean	85.51	91.85
Std. Dev.	0.01	0.004

Table 3.2: An illustration of the effectiveness of Gibbs sampling, compared to Viterbi inference, for the CoNLL named entity recognition task, and the CMU seminar announcements information extraction task. This table shows 10 runs of Gibbs sampling in the same CRF model that was used for Viterbi. For each run the sampler was initialized to a random sequence, and used a linear annealing schedule that sampled the complete sequence 1000 times. CoNLL performance is measured as per-entity F_1, and CMU seminar announcements performance is measured as per-token F_1.

where $\mathbf{c} = \{c_0, \ldots, c_T\}$ defines a *cooling schedule*. At each step, we raise each value in the conditional distribution to an exponent and renormalize before sampling from it. Note that when $c = 1$ the distribution is unchanged, and as $c \to 0$ the distribution becomes sharper, and when $c = 0$ the distribution places all of its mass on the maximal outcome, having the effect that the Markov chain always climbs uphill. Thus if we gradually decrease c from 1 to 0, the Markov chain increasingly tends to go uphill. This annealing technique has been shown to be an effective technique for stochastic optimization (Laarhoven and Arts 1987).

To verify the effectiveness of Gibbs sampling and simulated annealing as an inference technique for hidden state sequence models, I compared Gibbs and Viterbi inference methods for a basic CRF, without the addition of any non-local model. For each of the ten trials, I generated 1000 samples, and used the one with the highest likelihood. The results, given in table 3.2, show that if the Gibbs sampler is run long enough, its accuracy is the same as a Viterbi decoder. Keep in mind that this is exactly the behavior that was expected, but it is useful to see how many samples were necessary to get there, and it is comforting to

observe the low variance between each of the ten runs.

3.3.3 Gibbs sampling for inference in the product-of-experts model

We've now covered how to to Gibbs sampling in a CRF, and extending it to the product of experts model is trivial. Our overall model is specified by the following equation:

$$P_{\text{PoE}}(\mathbf{y}|\mathbf{x};\theta) \propto \prod_{i=1}^{|\mathbf{y}|} \phi(y_{i-1}, y_i|x;\theta) \times \prod_{\lambda \in \Lambda} \theta_\lambda^{\#(\lambda,\mathbf{y},\mathbf{x})} \tag{3.4}$$

Like the equation for the long-distance model (equation 3.1), this equation is unnormalized. The partition function from the CRF portion of the equation (the first term) has also been removed, since it only serves as a normalizer, and the equation is already unnormalized due to the fact that we are multiplying two probability distributions together (plus, one of them is already unnormalized).

Thankfully this doesn't matter, because we only use the model for Gibbs sampling, and so only need to compute the conditional distribution at a single position i (as defined in equation 3.2). One (inefficient) way to compute this quantity is to enumerate all possible sequences differing only at position i (so, conditioned on the labels at all points other than i), compute the score assigned to each by the model, and renormalize. Although it seems expensive, this computation can be made very efficient with a straightforward memoization technique: at all times the model maintains data structures representing the relationship between entity labels and token sequences, from which one can quickly compute counts of different types of violations.

3.4 The CoNLL NER task

I tested the effectiveness of the technique on two established datasets: the CoNLL 2003 English named entity recognition dataset, discussed in this section, and the CMU seminar announcements information extraction dataset, which I will discuss in the next section.

The CoNLL NER corpus was discussed in section 2.2.1, but I will briefly review its most salient features. It is composed of British newswire, and is annotated with four entity

	PER	LOC	ORG	MISC
PER	3141	4	5	0
LOC		6436	188	3
ORG			2975	0
MISC				2030

Table 3.3: Counts of the number of times multiple occurrences of a token sequence is labeled as different entity types in the same document. Taken from the CoNLL training set.

	PER	LOC	ORG	MISC
PER	1941	5	2	3
LOC	0	167	6	63
ORG	22	328	819	191
MISC	14	224	7	365

Table 3.4: Counts of the number of times an entity sequence is labeled differently from an occurrence of a subsequence of it elsewhere in the document.Rows correspond to sequences, and columns to subsequences. Taken from the CoNLL training set.

types: PER, LOC, ORG, and MISC. The data is separated into a training set, a development set (testa), and a test set (testb). The training set contains 945 documents, and approximately 203,000 tokens. The development set has 216 documents and approximately 51,000 tokens, and the test set has 231 documents and approximately 46,000 tokens.

3.4.1 Consistency model

Label consistency structure derives from the fact that within a particular document, different occurrences of a particular token sequence are unlikely to be labeled as different entity types. A named entity recognition system modeling this structure would try to assign all the occurrences of the token sequence to the same entity type, thereby sharing evidence among them. Although any one occurrence may be ambiguous, it is unlikely that all instances are unclear when taken together. Thus, modeling this structure should (and does) lead to accuracy improvements.

The CoNLL training data empirically supports the strength of the label consistency constraint. Table 3.3 shows the counts of entity labels for each pair of identical token sequences within a document, where both are labeled as an entity. Note that inconsistent

labelings are very rare.[2] In addition, we also want to model subsequence constraints: having seen *Geoff Woods* earlier in a document as a person is a good indicator that a subsequent occurrence of *Woods* should also be labeled as a person. However, if we examine all cases of the labelings of other occurrences of subsequences of a labeled entity, we find that the consistency constraint does not hold nearly so strictly in this case. As an example, one document contains references to both *The China Daily*, a newspaper, and *China*, the country. The first should be labeled as an ORGANIZATION, and second as a LOCATION. Counts of subsequence labelings within a document are listed in table 3.4. Note that there are many off-diagonal entries: the *China Daily* case is the most common, occurring 328 times in the dataset.

The penalties used in the long distance constraint model for CoNLL are the empirical Bayes estimates taken directly from the data (tables 3.3 and 3.4), except that I changed counts of 0 to be 1, so that the distribution remains positive. So the estimate of a PER also being an ORG is $\frac{5}{3151}$; there were 5 instances of an entity being labeled as both, PER appeared 3150 times in the data, and we add 1 to this for smoothing, because PER-MISC never occurred. However, when we have a phrase labeled differently in two different places, continuing with the PER-ORG example, it is unclear if we should penalize it as PER that is also an ORG or an ORG that is also a PER. To deal with this, we multiply the square roots of each estimate (a PER mislabeled as an ORG, and an ORG mislabeled as a PER) together to form the penalty term. The penalty term is then multiplied in a number of times equal to the length of the offending entity; this is meant to "encourage" the entity to shrink.[3] For example, say we have a document with three entities, *Rotor Volgograd* twice, once labeled as PER and once as ORG, and *Rotor*, labeled as an ORG. The likelihood of a PER also being an ORG is $\frac{5}{3151}$, and of an ORG also being a PER is $\frac{5}{3169}$, so the penalty for this violation is $(\sqrt{\frac{5}{3151}} \times \sqrt{\frac{5}{3169}})^2$. The likelihood of an ORG being a subphrase of a PER is $\frac{2}{842}$. So the total penalty would be $\frac{5}{3151} \times \frac{5}{3169} \times \frac{2}{842}$.

[2] A notable exception is the labeling of the same text as both organization and location within the same document. This is a consequence of the large portion of European sports news in the CoNLL dataset, so that city names are often also team names.

[3] While there is no theoretical justification for this, I found it to work well in practice.

CoNLL					
Approach	LOC	ORG	MISC	PER	ALL
B&M LT-RMN	–	–	–	–	80.09
B&M GLT-RMN	–	–	–	–	82.30
Local+Viterbi	88.16	80.83	78.51	90.36	85.51
NonLoc+Gibbs	88.51	81.72	80.43	92.29	86.86

Table 3.5: F_1 scores of the local CRF and non-local models on the CoNLL 2003 named entity recognition dataset. I also provide the results from Bunescu and Mooney (2004) for comparison.

3.4.2 Experiments

In the experiments I compared the impact of adding the non-local models with Gibbs sampling to a baseline CRF implementation, and to prior work by Bunescu and Mooney (2004). I evaluated using entity-level, micro-averaged, precision, recall and F_1 score (see section 2.1.1), which was the same evaluation metric used in the shared task from which the data originated. The results are found in table 3.5. For all experiments involving Gibbs sampling, I used a linear cooling schedule. I initialized the chain to the Viterbi output from the baseline CRF, and then collected 200 samples per trial, and report the average of all trials. The trials had a low standard deviation – 0.083% – and a high minimum F_1 score – 86.72% – demonstrating the stability of the method. Improvements are significant with greater than 95% confidence using a standard t-test.

The non-local model increased the F_1 score by about 1.3% compared to the baseline CRF. Although such gains may appear modest, I was only targeting one particular type of error: consistency errors. Often, words mislabeled by the baseline system would be corrected by the addition of long-distance links, but occasionally the incorrectly labeled word would "win," and words labeled correctly by the baseline system would become wrong in the new model. Other classes of errors remained largely unchanged. Also, note that these gains are achieved relative to a near state-of-the-art NER system: the winner of the CoNLL English task reported an F_1 score of 88.76%. In contrast, the increases published by Bunescu and Mooney (2004) are relative to a baseline system which scores only 80.9% on the same task.

The biggest drawback to the model is the computational cost. Taking 200 samples

dramatically increases test time. Averaged over 3 runs on both Viterbi and Gibbs, CoNLL testing time increased from 55 to 1738 seconds. However, this increase in time is due largely to computing the long-distance penalties and not the conditional probabilities in the CRF. The 1000 sample runs (in table 3.2) which did not use a global model, averaged 328 seconds, indicating that with simpler, better optimized, global models, test time could be dramatically reduced.

3.5 The CMU seminar announcements task

3.5.1 Task description

This dataset was developed as part of Dayne Freitag's dissertation research (Freitag 1998).[4] It consists of 485 emails containing seminar announcements at Carnegie Mellon University. It is annotated for four fields: SPEAKER, LOCATION, START_TIME, and END_TIME. Sutton and McCallum (2004) used 5-fold cross validation on the entire corpus when evaluating on this dataset, so I obtained and used their data splits, so that results can be properly compared. Because the entire dataset is used for testing, there is no development set, and so I did all of my development using the CoNLL NER data. I also used their evaluation metric, which is slightly different from the method for the CoNLL data. Instead of evaluating precision and recall on a per-entity basis, they are evaluated on a per-token basis, and the overall score is computed by macro-averaging (as opposed to micro-averaging) the individual types.

3.5.2 Consistency model

Due to the lack of a development set, my consistency model for the CMU seminar announcements is much simpler than the CoNLL model; the numbers where selected according to my intuitions, and I did not spend much time hand optimizing the model. Specifically, I had three constraints. The first is that all entities labeled as START_TIME are normalized (e.g., *3:00 pm* and *3pm* are both turned into *0300*), and are penalized if they are

[4] Available at *http://nlp.shef.ac.uk/dot.kom/resources.html*.

CMU Seminar Announcements					
Approach	STIME	ETIME	SPEAK	LOC	ALL
S&M CRF	97.5	97.5	88.3	77.3	90.2
S&M Skip-CRF	96.7	97.2	88.1	80.4	90.6
Local+Viterbi	96.67	97.36	83.39	89.98	91.85
NonLoc+Gibbs	97.11	97.89	84.16	90.00	92.29

Table 3.6: F_1 scores of the local CRF and non-local models on the CMU seminar announcements dataset. I also provide the results from Sutton and McCallum (2004) for comparison.

inconsistent. The second is a corresponding constraint for END_TIME. The last constraint attempts to consistently label the SPEAKER. If a phrase is labeled as a SPEAKER, we assume that the last word is the speaker's last name, and we penalize for each occurrence of that word which is not also labeled SPEAKER. For START_TIME and END_TIME the penalty is multiplied in based on how many words are in the entity. For SPEAKER, the penalty is only multiplied in once. I used a hand selected penalty of $\exp\{-4.0\}$.

3.5.3 Experiments

My experiments on the CMU seminar announcements dataset parallel those on the CoNLL data. I once again compared the impact of adding the non-local models with Gibbs sampling to a baseline CRF implementation, and to prior work by Sutton and McCallum (2004). The results are found in table 3.6. For all experiments involving Gibbs sampling, I used a linear cooling schedule, and report the average over 100 trials. The trials once again had low standard deviation – 0.007% – and a high minimum F_1 score – 92.28%. Compared with prior work, my model had larger improvements compared to a stronger baseline. Improvements are significant with greater than 95% confidence using a standard t-test.

3.6 Summary

In this chapter, I presented a method for efficiently adding long-distance information into a linear-chain CRF. This is done by training two separate models: the linear-chain CRF and a long-distance model. At test time, the two models are multiplied together, and instead

of the standard Viterbi decoding algorithm we use Gibbs sampling to perform inference and find (an approximation to) the most likely labeling in combined model. While Gibbs sampling is designed more for estimating a distribution than for finding its max, I did not have any difficulties with this, as evidenced by the experimental results. I demonstrated the effectiveness of this technique on two separate tasks, and in both cases achieved larger improvements to stronger baselines when compared to previous work on the same data.

There are many other potential applications for the technique presented here. I combined a trained (sequence) model with a hand-tuned (long-distance) model, but clearly one could combine any number of trained and/or hand-tuned models. One could also try to learn the parameters for the long-distance model through max-margin (Taskar et al. 2003, Tsochantaridis et al. 2005) or perceptron (Freund and Schapire 1998, Collins 2002) learning. Because both of these techniques require only the argmax, and not partial derivatives, the sampling procedure could be used during training time to find the argmax. A similar technique could also be used to learn how to weight the different models; the model in this chapter gave them equal weight. Moving beyond sequence models, this technique can be applied to nearly any model for which it is known how to do MCMC sampling. Currently, for PCFG parsing there is no known MCMC procedure (though it is known how to generate samples from the correct distribution (Finkel et al. 2006, Johnson and Griffiths 2007)), but if one were developed then long-distance links could usefully be added in a number of places. They would be helpful for deciding prepositional phrase attachment, since in that case local information is often insufficient. Long-distance links would also be useful in places where there is likely to be parallelism, such as the internal structure of two phrases joined by a conjunction.

Chapter 4

Bayes-optimal inference to improve NLP pipelines

4.1 Introduction

In the last chapter we considered incorporating long-distance information at one level of linguistic analysis. In this chapter we turn our attention to the links between different levels of analysis. Almost any high-level system for natural language understanding must recover hidden linguistic structure at many different levels: part-of-speech tags, syntactic dependencies, named entities, etc. Consider the case of semantic role labeling (SRL). In this task (described more fully in section 4.5), the goal is to identify semantic roles, such as *subject*, *direct object*, and *temporal modifier* for a particular predicate, or verb. Modern semantic role labeling systems use the syntactic parse tree of the sentence. Question answering systems require question type classification, parsing, named entity recognition, semantic role labeling, and often other tasks, many of which are dependent on one another and are often pipelined together. Pipelined systems are ubiquitous in NLP: in addition to the above examples, commonly parsers and named entity recognizers use part-of-speech and chunking information, and also word segmentation for languages such as Chinese. Almost no NLP task is truly standalone.

Most current systems for higher-level, aggregate NLP tasks employ a simple 1-best feed forward architecture: they greedily take the best output at each stage in the pipeline

and pass it on to the next stage. This is the simplest architecture to build (particularly if reusing existing component systems), but errors are frequently made during this pipeline of annotations, and when a system is given incorrectly labeled input it is much harder for that system to do its task correctly. For example, when doing semantic role labeling, if no syntactic constituent of the parse actually corresponds to a given semantic role, then that semantic role will almost certainly be misidentified. So, while it is not surprising, it is disappointing that F-measures on SRL drop more than 10% when switching from gold parses to automatic parses (for instance, from 91.2 to 80.0 for the joint model of (Toutanova et al. 2005)).

A common improvement on this architecture is to pass *k*-best lists between processing stages, for example Sutton and McCallum (2005) and Wellner et al. (2004). Passing on a *k*-best list gives useful improvements (e.g., in Koomen et al. (2005)), but efficiently enumerating *k*-best lists often requires very substantial cognitive and engineering effort, e.g., in Huang and Chiang (2005) and Toutanova et al. (2005).

At the other extreme, one can maintain the entire space of representations (and their probabilities) at each level, and use this full distribution to calculate the full distribution at the next level. If restricting oneself to weighted finite state transducers (WFSTs), a framework applicable to a number of NLP applications (as outlined in Karttunen (2000)), a pipeline can be compressed down into a single WFST, giving outputs equivalent to propagating the entire distribution through the pipeline. In the worst case there is an exponential space cost, but in many relevant cases composition is in practice quite practical. Outside of WFSTs, maintaining entire probability distributions is usually infeasible in NLP, because for most intermediate tasks, such as parsing and named entity recognition, there is an exponential number of possible labelings. Nevertheless, for some models, such as most parsing models, these exponential labelings can be compactly represented in a packed form (e.g., Maxwell and Kaplan (1995) and Crouch (2005)) and subsequent stages can be re-engineered to work over these packed representations (Geman and Johnson 2002). However, doing this normally also involves a very high cognitive and engineering effort, and in practice this solution is infrequently adopted. Moreover, in some cases, a subsequent module is incompatible with the packed representation of a previous module and an exponential amount of work is nevertheless required within this architecture.

In this chapter, I explore an attractive middle ground between the standard, greedy linguistic pipelines and the full joint models discussed in subsequent chapters. Rather than only using the 1 or k most likely labelings at each stage, we would indeed like to take into account all possible labelings and their probabilities. Like the long-distance models of the previous chapter, this is achieved by use of sampling for approximate inference. The form of approximate inference I use is very simple: at each stage in the pipeline, we draw a sample from the distribution of labels, conditioned on the samples drawn at previous stages. We repeat this many times, and then use the samples from the last stage, which corresponds to the final, higher-level task, to form a majority vote classifier. As the number of samples increases, this method will approximate the complete distribution. Use of the method is normally a simple modification to an existing piece of code, and the method is general. It can be applied not only to all pipelines, but to any directed acyclic graph (DAG) structured multi-stage algorithms as well.

While both this chapter and the previous focus on the use of sampling for inference, there are some important conceptual differences. In the previous chapter, I used MCMC to walk around the probability space for a single (highly-structured, joint) distribution, because we were unable to do exact inference. In this chapter, we are generating independent samples instead of doing MCMC. Here, the samples are generated as a means of doing Bayesian inference – we only care about the output from the last stage in the pipeline, but instead of taking point estimates at each prior stage we would like to take the entire distribution into account, and this technique provides a means to do that.

I applied the method to two problems: semantic role labeling and recognizing textual entailment. For semantic role labeling I used a two stage pipeline which parses the input sentence and then performs semantic role labeling using the parse tree from the previous stage. For recognizing textual entailment I used a three stage pipeline which tags the sentence with named entities, and then parses it (forcing it to accept named entity boundaries) before passing it to the entailment decider, which uses both the named entity and parse information. The sampling pipeline performed better than the greedy 1-best pipeline, and performed comparably to a k-best pipeline.

4.2 Related work

While sampling pipelines have not been tried before, there is prior work which uses *k*-best lists. Sutton and McCallum (2005) used the output of a *k*-best list from a parser as input to a semantic role labelling system, but unfortunately they had a negative result and found the 1-best parser output to perform better. More recently, Zhao et al. (2009) used *k*-best parse information as input to their semantic role labelling system, and performed quite well in the 2009 CoNLL shared task on joint parsing and SRL. The top performing system however still completely decoupled the tasks.

Hollingshead and Roark (2007) also explore an alternative to standard pipelines, and present something called *pipeline iteration*. One way to view pipelines, is that the outputs from earlier stages constrain the possible output space for latter stages. In their work, they make multiple passes through the pipeline, and allow the outputs from later stages to constrain the earlier stages during the next iteration. They saw improvements when using pipeline iteration in a reranking parser.

The work in this chapter also makes extensive use of Ng and Jordan (2001). That paper focuses on the *voting Gibbs classifier*, and analyzes its use as an approximation to the Bayes optimal classifier. They were interested in the case of Bayesian classification, where the hyper-parameters for the prior are unknown. In the Bayes optimal classifier, these hyper-parameters would be integrated out, effectively utilizing the entire distribution of possible hyper-parameter values. The voted Gibbs classifier instead samples a value for the hyper-parameters, and then using that value, generates a sample from the classifier. This is repeated multiple times, and the outputs from the classifier are used to construct a majority vote classifier, the output of which is the final output. While I have a different problem setting, the fundamental idea is still the same. In our case, the Bayes optimal classifier is what you would get if you passed the entire distribution through at each stage in the pipeline (instead of just a point estimate, which is what a 1-best list passes). My sampling pipeline is a voting Gibbs classifier, because it passes samples along at each stage, and then uses them to form a majority-vote classifier in the last stage.

4.3 Approach

4.3.1 Overview

In order to do approximate inference, we model the entire pipeline as a Bayesian network. Each stage in the pipeline corresponds to a variable in the network. For example, the parser stage corresponds to a variable whose possible values are all possible parses of the sentence. The probabilities of the parse trees are conditioned on the parent variables, which may just be the words of the sentence, or may be the part-of-speech tags output by a part-of-speech tagger.

The simple linear structure of a typical linguistic annotation network permits exact inference that is quadratic in the number of possible labels at each stage, but unfortunately our annotation variables have a very large domain. Additionally, some networks may not even be linear; frequently one stage may require the output from multiple previous stages, or multiple earlier stages may be completely independent of one another. For example, a typical question answering system will do question type classification on the question, and from that extract keywords which are passed to the information retrieval part of the system. Meanwhile, the retrieved documents are parsed and tagged with named entities; the network rejoins those outputs with the question type classification to decide on the correct answer. My approach addresses these issues by using approximate inference instead of exact inference. The structure of the nodes in the network permits direct sampling based on a topological sort of the nodes. Samples are drawn from the conditional distributions of each node, conditioned on the samples drawn at earlier nodes in the topological sort.

4.3.2 Probability of a complete labeling

Before we can discuss how to sample from these Bayes nets, we will formalize how to move from an annotation pipeline to a Bayes net. Let $\mathbf{A}$ be the set of n annotators A_1, A_2, ..., A_n (e.g., part-of-speech tagger, named entity recognizer, parser). These are the variables in the network. For annotator A_i, we denote the set of other annotators whose input is directly needed as $Parents(A_i) \subset \mathbf{A}$ and a particular assignment to those variables is $parents(A_i)$. The possible values for a particular annotator A_i are a_i (i.e., a particular parse tree or named

entity tagging). We can now formulate the probability of a complete annotation (over all annotators) in the standard way for Bayes nets:

$$\mathrm{P_{BN}}(a_1, a_2, ..., a_n) = \prod_{i=1}^{n} P(a_i | parents(A_i)) \tag{4.1}$$

4.3.3 Approximate inference in Bayesian networks

This factorization of the joint probability distribution facilitates inference. However, exact inference is intractable because of the number of possible values for our variables. Parsing, part-of-speech tagging, and named entity tagging (to name a few) all have a number of possible labels that is exponential in the length of the sentence, so we use approximate inference. I chose Monte Carlo inference, in which samples drawn from the joint distribution are used to approximate a marginal distribution for a subset of variables in the distribution. First, the nodes are sorted in topological order. Then, samples are drawn for each variable, conditioned on the samples which have already been drawn. Many samples are drawn, and are used to estimate the joint distribution.

Importantly, for many language processing tasks our application only needs to provide the most likely value for a high-level linguistic annotation (e.g., the guessed semantic roles, or answer to a question), and other annotations such as parse trees are only present to assist in performing that task. The probability of the final annotation is given by:

$$\mathrm{P_{BN}}(a_n) = \sum_{a_1, a_2, ..., a_{n-1}} \mathrm{P_{BN}}(a_1, a_2, ..., a_n) \tag{4.2}$$

Because we are summing out all variables other than the final one, we effectively use only the samples drawn from the final stage, ignoring the labels of the internal variables, to estimate the marginal distribution over that variable. We then return the label which had the highest number of samples. For example, when trying to recognize textual entailment (RTE) (explained in section 4.6), we count how many times we sampled “yes, it is entailed” and how many times we sampled “no, it is not entailed” and return the answer with more samples.

When the outcome you are trying to predict is binary (as is the case with RTE) or *n*-ary for small *n*, the number of samples needed to obtain a good estimate of the posterior probability is very small. Somewhat counter-intuitively, this is true even if the spaces being sampled from during intermediate stages are exponentially large (such as the space of all parse trees). Ng and Jordan (2001) show that under mild assumptions, with only N samples the relative classification error will be at most $O(\frac{1}{N})$ higher than the error of the Bayes optimal classifier (in our case, the classifier which does exact inference). Even if the outcome space is not small, the sampling technique I present can still be very useful, as we will see later for the case of SRL.

4.4 Generating samples

The method I have outlined requires the ability to sample from the conditional distributions in the factored distribution of equation 4.1: in our case, the probability of a particular linguistic annotation, conditioned on other linguistic annotations. Note that this differs from the usual annotation task: taking the argmax. But for most algorithms the change is small and easy. I will now discuss how to obtain samples efficiently from a few different annotation models: probabilistic context free grammars (PCFGs), and linear-chain CRFs.

4.4.1 Sampling parses

In this section, I will show how to generate parse tree samples from a PCFG for a given sentence. Early work which used parse sampling (Bod 1995) used a different formalism: tree substitution grammars. That work presented a bottom-up algorithm for sampling parse *derivations*. The derivation is the binarized initial tree output from a parser, which often includes other, additional annotations on the nodes. This derivation is then debinarized, and the additional mark-up is removed, before the real parse tree is returned. One example of an additional annotation is augmenting NPs (noun phrases) to include information stating that they are temporal or locative. In this case, in the derivation, the state would be NP-TMP or NP-LOC. You could have two derivations which are identical except for one node which

is labeled NP-TMP in one, and NP-LOC in the other. The final parse trees would be identical, because the LOC/TMP would be removed. So, as you can see, multiple derivations can correspond to the same final parse tree. When finding the Viterbi (most likely) parse, the algorithm actually finds the most likely derivation, and there is some chance that this will not correspond to the most likely parse. However, the sampling procedure presented in Bod (1995) samples derivations, and sampling from the space of derivations is equivalent to sampling in the space of final parse trees. Goodman (1998) then presented a top-down version of this algorithm. Although I used a PCFG for parsing (instead of a tree substitution grammar, like the just described work), it is the grammar of Klein and Manning (2003), which uses extensive state-splitting, which results in additional mark-ups on the nodes in the tree. Once again there is again a many-to-one correspondence between derivations and parses, and I use an algorithm similar to Goodman's, described below.

PCFGs put probabilities on each rule, such as S $\rightarrow$ NP VP and NN $\rightarrow$ *'dog'*. The probability of a complete parse tree is the product of the probabilities of the rules used to construct the parse tree. A dynamic programing algorithm, the *inside algorithm*, can be used to find the probability of a sentence. The *inside probability* $\beta_k(p,q)$ is the probability that words p through q, inclusive, were produced by the non-terminal k. So the probability of the sentence *The boy pet the dog.* is equal to the inside probability $\beta_S(1,6)$, where the first word, w_1 is *The* and the sixth word, w_6, is *[period]*. This quantity is the sum of the probabilities of all parses of the sentence which have S as the root symbol. The probability can be defined recursively (Manning and Schütze 1999) as follows:

$$\beta_k(p,q) = \begin{cases} P(N^k \rightarrow w_p) & \text{if } p = q \\ \sum_{r,s} \sum_{d=p}^{q-1} P(N^k \rightarrow N^r N^s) \beta_r(p,d) \beta_s(d+1,q) & \text{otherwise} \end{cases} \tag{4.3}$$

where N^k, N^r and N^s are non-terminal symbols and w_p is the word at position p. These probabilities can be efficiently computed using a dynamic program, or memoization of each value as it is calculated.

function DRAWSAMPLE(N^k, r, s)
 if $r = s$
  $tree.label = N^k$
  $tree.child = word(r)$
  return $(tree)$
 for each `rule` $m \in \{m' : head(m') = N^k\}$
  $N^i \leftarrow lChild(m)$
  $N^j \leftarrow rChild(m)$
  for $q \leftarrow r$ **to** $s-1$
   $scores(m,q) \leftarrow P(m)\beta_i(r,q)\beta_j(q+1,s)$
 $(m,q) \leftarrow$ SAMPLEFROM$(scores)$
 $tree.label = head(m)$
 $tree.lChild =$ DRAWSAMPLE$(lChild(m), r, q)$
 $tree.rChild =$ DRAWSAMPLE$(rChild(m), q+1, s)$
 return $(tree)$

Figure 4.1: Pseudo-code for sampling parse trees from a PCFG. This is a recursive algorithm which starts at the root of the tree and expands each node by sampling from the distribution of possible rules and ways to split the span of words. Its arguments are a non-terminal and two integers corresponding to word indices, and it is initially called with arguments S, 1, and the length of the sentence. There is a call to *sampleFrom*, which takes an (unnormalized) probability distribution, normalizes it, draws a sample and then returns that sample.

Once all of the inside probabilities have been computed, they can be used to generate parses from the distribution of all parses of the sentence, using the algorithm in figure 4.1.This algorithm is called after all of the inside probabilities have been calculated and stored, and takes as parameters S, 1, and $length(sentence)$. It works by building the tree, starting from the root, and recursively generating children based on the posterior probabilities of applying each rule and each possible position on which to split the sentences. Intuitively, the algorithm is given a non-terminal symbol, such as *S* or *NP*, and a span of words and has to decide (a) what rule to apply to expand the non-terminal, and (b) where to split the span of words, so that each non-terminal resulting from applying the rule has an associated word span, and the process can repeat. The inside probabilities are calculated just once, and we can then generate many samples very quickly; *DrawSamples* is linear in the number of words and rules.

4.4.2 Sampling named entity taggings

To do named entity recognition, I used the same linear-chain conditional random field (CRF) model as presented earlier in section 2.1.1. Previously, we used Markov chain Monte Carlo (MCMC) as a means of dealing with the additional long-distance dependencies. Here, because we do not have these long-distance dependencies, and because we want samples from the distribution and not the most likely labelling, we can use a simpler sampling technique which repeatedly generates independent samples. To review briefly, when building a CRF we create a linear chain of *cliques*, each of which represents the probabilistic relationship between an adjacent pair of states using a *factor table* containing $|S|^2$ values. These factor tables are defined in terms of exponential models conditioned on features of the observation sequence, and must be instantiated for each new observation sequence. As stated earlier, these factor tables are *not* unnormalized probability tables. We need to first execute the forward-backward algorithm, a special case of a process called *clique tree calibration*, which involves passing *messages* between the cliques (see Koller and Friedman (2009) for a full treatment of this topic). This process propagates information throughout the entire sequence, and after it has completed, the factor tables can be

viewed as unnormalized probabilities, which can be used to compute conditional probabilities, $P_{CRF}(y_i|y_{i-n}\ldots y_{i-1},x)$. Once these probabilities have been calculated, generating samples is very simple. First, we draw a sample for the label at the first position,[1] and then, for each subsequent position, we draw a sample from the distribution for that position, conditioned on the label sampled at the previous position. This process results in a sample of a complete labeling of the sequence, drawn from the posterior distribution of complete named entity taggings.

Similarly to generating sample parses, the expensive part is calculating the probabilities; once we have them we can generate new samples very quickly.

4.4.3 *k*-Best lists

At first glance, *k*-best lists may seem like they should outperform sampling, because in effect they are the *k* best samples. However, there are several important reasons why one might prefer sampling. One reason is that the *k* best paths through a word lattice, or the *k* best derivations in parse forest do not necessarily correspond to the *k* best sentences or parse trees. In fact, there are no known sub-exponential algorithms for the best outputs in these models, when there are multiple ways to derive the same output (this was discussed in section 4.4.1). This is not just a theoretical concern – the Stanford parser (Klein and Manning 2003) uses such a grammar, and I found that when generating a 50-best derivation list that on average these derivations corresponded to about half as many unique parse trees. My approach circumvents this issue entirely, because the samples are generated from the actual output distribution.

Intuition also suggests that sampling should give more diversity at each stage, reducing the likelihood of not even considering the correct output.[2] Using the Brown portion of the SRL test set (discussed in section 4.5), and 50-samples/50-best, I found that on average the 50-samples system considered approximately 25% more potential SRL labelings than the 50-best system.

When pipelines have more than two stages, it is customary to do a beam search, with a

[1] Conditioned on the distinguished start states.

[2] The *i*-best answer will differ in only one position from one of the *j*-best answers, where $j < i$.

beam size of k. This means that at each stage in the pipeline, more and more of the probability mass gets "thrown away." Practically, this means that as pipeline length increases, there will be increasingly less diversity of labels from the earlier stages. In a degenerate 10-stage, k-best pipeline, where the last stage depends mainly on the first stage, it is probable that all but a few labelings from the first stage will have been pruned away, leaving something much smaller than a k-best sample, possibly even a 1-best sample, as input to the final stage. Using approximate inference to estimate the marginal distribution over the last stage in the pipeline, such as this sampling approach, the pipeline length does not have this negative impact or affect the number of samples needed. And unlike k-best beam searches, there is an entire research community, along with a large body of literature, which studies how to do approximate inference in Bayesian networks and can provide performance bounds based on the method and the number of samples generated.

One final issue with the k-best method arises when instead of a linear chain pipeline, one is using a general directed acyclic graph, where a node can have multiple parents. In this situation, doing the k-best calculation actually becomes exponential in the size of the largest in-degree of a node – for a node with p parents, you must try all k^p combinations of the values for the parent nodes. With sampling this is not an issue; each sample can be generated based on a topological sort of the graph.

4.5 Semantic role labeling

4.5.1 Task description

Given a sentence and a target verb (also called the *predicate*) the goal of semantic role labeling is to identify and label syntactic constituents of the parse tree with semantic roles of the predicate. Common roles are *agent*, which is the thing performing the action, *patient*, which is the thing on which the action is being performed, and *instrument*, which is the thing with which the action is being done. Additionally, there are *modifier arguments* which can specify the location, time, manner, etc. The following sentence provides an example of a predicate and its arguments:

[The luxury auto maker]$_{agent}$ [last year]$_{temp}$ [sold]$_{pred}$ [1,214 cars]$_{patient}$ in [the

Figure 4.2: The pipeline for semantic role labeling.

U.S]$_{location}$.

Semantic role labeling is a key component for systems that do question answering, summarization, and any other task which directly uses a semantic interpretation.

4.5.2 System description

I modified the system described in Haghighi et al. (2005) and Toutanova et al. (2005) to test my method. The system uses two kinds of models: local models, which score subtrees of the entire parse tree independently of the labels of other nodes not in that subtree, and joint models, which score the entire labeling of a tree with semantic roles (for a particular predicate).

First, the task is separated into two stages, and local models are learned for each. At the first stage, the *identification stage*, a classifier labels each node in the tree as either *ARG*, meaning that it is an argument (either core or modifier) to the predicate, or *NONE*, meaning that it is not an argument. At the second stage, the *classification stage*, the classifier is given a set of arguments for a predicate and must label each with its specific semantic role.

Next, a Viterbi-like dynamic algorithm is used to generate a list of the k-best joint (identification and classification) labelings according to the local models. The algorithm enforces the constraint that the roles should be non-overlapping. Finally, a joint model is constructed which scores a completely labeled tree, and it is used to re-rank the k-best list. The separation into local and joint models is necessary because there is an exponential number of ways to label the entire tree, and using the joint model alone would be intractable. I retained the k-best structure here, but ideally one would want to use approximate inference instead of a k-best list here as well. Importance sampling would be particularly well suited – instances could be sampled from the local model and then re-weighted using the joint model.

Because the SRL system outputs a *k*-best list, we already know exactly how likely it thinks each of the *k* outputs is, for a particular parse sample input. If we drew samples here, we would lose valuable information which we have easy access to. So, for each parse sample, the *k* outputs become weighted votes, where their weight is their likelihood in the *k*-best list. Multiple samples are passed through, and in the end voting is done with these weighted samples. The complete pipeline is shown graphically in figure 4.3.

4.5.3 Experiments

In 2004 and 2005 CoNLL had shared tasks on SRL (Carreras and Màrquez (2004) and Carreras and Màrquez (2005)). I used the CoNLL 2005 data and evaluation script. When evaluating semantic role labeling results, it is common to present numbers on both the core arguments (i.e., excluding the modifying arguments) and all arguments. I follow this convention and present both sets of numbers. I give precision, recall and F1 (see section 2.1.1), which are based on the number of arguments correctly identified. For an argument to be correct both the span and the classification must be correct; there is no partial credit.

To generate sampled parses, I used the Stanford parser (Klein and Manning 2003). The CoNLL data comes with parses from Charniak's parser (Charniak 2000), so I re-parsed the data and retrained the SRL system on these new parses, resulting in a lower baseline than previously presented work. I choose to use Stanford's parser because of the ease with which it could be modified to generate samples. Unfortunately, its performance is slightly below that of the other parsers.

The CoNLL data has two separate test sets; the first is section 23 of the PennTreebank (PTB) (Marcus et al. 1993), and the second is "fresh sentences" taken from the Brown corpus. For full results, please see table 4.1. On the PTB portion, compared to the standard greedy pipeline, I saw an absolute F-score improvement of 0.7% on both core and all arguments. On the Brown portion of the test set I saw an improvement of 1.25% on core and 1.16% on all arguments. In this context, a gain of over 1% is quite large: for instance, the scores for the top 4 systems on the Brown data at CoNLL 2005 were within 1% of each other. For both portions, I generated 50 samples, and did this 4 times, averaging the results. The better performance on the Brown portion compared to the PTB portion is likely to be

SRL Results – Penn Treebank Portion			
Core Args	Precision	Recall	F-measure
Greedy	79.31%	77.7%	78.50%
K-Best	80.05%	78.45%	79.24%
Sampling	80.13%	78.25%	79.18%
All Args	Precision	Recall	F-measure
Greedy	78.49%	74.77%	76.58%
K-Best	79.58%	74.90%	77.16%
Sampling	79.81%	74.85%	77.31%
SRL Results – Brown Portion			
Core Args	Precision	Recall	F-measure
Greedy	68.28%	67.72%	68.0%
K-Best	69.25%	69.02%	69.13%
Sampling	69.35%	68.93%	69.16%
All Args	Precision	Recall	F-measure
Greedy	66.6%	60.45%	63.38%
K-Best	68.82%	61.03%	64.69%
Sampling	68.6%	61.11%	64.64%

Table 4.1: Results for semantic role labeling task. The sampled numbers are averaged over several runs, as discussed.

because the parser was trained on the PennTreebank training data, so the most likely parses will be of higher quality for the PTB portion of the test data than for the Brown portion. I also ran the pipeline using a 50-best list, and found the two results to be comparable.

4.6 Recognizing textual entailment

4.6.1 Task description

In the task of recognizing textual entailment (RTE), also commonly referred to as robust textual inference, you are provided with two passages, a *text* and a *hypothesis*, and must decide whether the hypothesis can be inferred from the text. The term *robust* is used because the task is not meant to be domain specific. The term *inference* is used because this is not meant to be logical entailment, but rather what an intelligent, informed human would infer. Many NLP applications would benefit from the ability to do robust textual

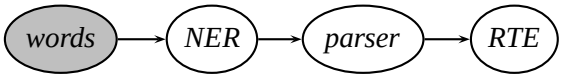


Figure 4.3: The pipeline for recognizing textual entailment.

entailment, including question answering, information retrieval and multi-document summarization. Starting in 2005, there has been an annual series of RTE workshops with an associated shared challenge (Dagan et al. 2005, Ido et al. 2006, Giampiccolo et al. 2007; 2008, Bentivogli et al. 2009). I used the data from the 2005 and 2006 workshops. In 2005 there were 576 text-hypothesis pairs in the development set, and 800 pairs in the test set (there is no training set). In 2006 there were 800 pairs in both the development and test sets[3]. Here is an example from the development set from the first RTE challenge:

Text: *Researchers at the Harvard School of Public Health say that people who drink coffee may be doing a lot more than keeping themselves awake – this kind of consumption apparently also can help reduce the risk of diseases.*

Hypothesis: *Coffee drinking has health benefits.*

The positive and negative examples are balanced, so the baseline of guessing either all *yes* or all *no* would score 50%. This is a hard task – at the first challenge no system scored over 60%.

4.6.2 System description

MacCartney et al. (2006) describe a system for doing robust textual inference. They divide the task into three stages – linguistic analysis, graph alignment, and entailment determination. The first of these stages, *linguistic analysis*, is itself a pipeline of parsing and named entity recognition. They use the syntactic parse to (deterministically) produce a typed dependency graph for each sentence. This pipeline is the one replaced. The second stage, *graph alignment*, consists of trying to find good alignments between the typed dependency

[3]The datasets and further information from the challenges can be downloaded from *http://www.pascal-network.org/Challenges/RTE2/Datasets/*.

graphs for the text and hypothesis. Each possible alignment has a score, and the alignment with the best score is propagated forward. The final stage, *entailment determination*, is where the decision is actually made. Using the score from the alignment, as well as other features, a logistic model is created to predict entailment. The parameters for this model are learned from development data.[4] While it would be preferable to sample possible alignments, their system for generating alignment scores is not probabilistic, and it is unclear how one could convert between alignment scores and probabilities in a meaningful way.

Our modified linguistic analysis pipeline (see figure 4.3) does NER tagging and parsing (in their system, the parse is dependent on the NER tagging because some types of entities are pre-chunked before parsing) and treats the remaining two sections of their pipeline, the alignment and determination stages, as one final stage. Because the entailment determination stage is based on a logistic model, a probability of entailment is given and sampling is straightforward.

4.6.3 Experiments

For the second PASCAL RTE challenge, two different types of performance measures were used to evaluate labels and confidence of the labels for the text-hypothesis pairs. The first measure is accuracy – the percentage of correct judgments. The second measure is *average precision*. Responses are sorted based on entailment confidence and then average precision is calculated by the following equation:

$$average\_precision = \frac{1}{R}\sum_{i=1}^{n} E(i)\frac{\#\text{ correct up to pair } i}{i} \tag{4.4}$$

where n is the size of the test set, R is the number of positive (entailed) examples, $E(i)$ is an indicator function whose value is 1 if the ith pair is entailed, and the is are sorted based on the entailment confidence. The intention of this measure is to evaluate how well calibrated a system is. Systems which are more confident in their correct answers and less confident in their incorrect answers will perform better on this measure.

[4]They report their results on the first PASCAL dataset, and use only the development set from the first challenge for learning weights. When I tested on the data from the second challenge, I used all data from the first challenge and the development data from the second challenge to learn these weights.

RTE Results		
	Accuracy	Average Precision
Greedy	59.13%	59.91%
Sampling	60.88%	61.99%

Table 4.2: Results for recognizing textual entailment.
The sampled numbers are averaged over several runs, as discussed.

Experimental results are presented in table 4.2. I generated 25 samples for each run, and repeated the process 7 times, averaging over runs. Accuracy was improved by 1.5% and average precision by 2%, when compared with the original 1-best pipeline. It does not come as a surprise that the average precision improvement was larger than the accuracy improvement, because our model explicitly estimates its own degree of confidence by estimating the posterior probability of the class label.

4.7 Summary

In this chapter, I have presented an attractive middle-ground between the standard greedy pipeline, and the fully joint models covered in subsequent chapters. Specifically, I have presented a method for handling language processing pipelines in which later stages of processing are conditioned on the results of earlier stages. It is still common practice to take the best labeling at each point in a linguistic analysis pipeline, but this method ignores information about alternate labelings and their likelihoods. My approach uses all of the information available, and has the added advantage of being extremely simple to implement. By modifying your subtasks to generate samples instead of the most likely labeling, the method can be used with very little additional overhead. And, as I have shown, such modifications are usually simple to make; further, with only a "small" (polynomial) number of samples k, under mild assumptions the classification error obtained by the sampling approximation approaches that of exact inference (Ng and Jordan 2001). In contrast, an algorithm that keeps track only of the k-best list enjoys no such theoretical guarantee, and can require an exponentially large value for k to approach comparable error. However, I found that experimentally, the two performed comparably. It would be interesting to see how they compare on longer pipelines, or DAG-structured, but non-linear, pipelines.

There are many possible future directions for this work. One drawback of the model I have presented is that information only flows forward. In some respects, that makes the model similar to an MEMM (Borthwick 1999, McCallum et al. 2000), and it may suffer from similar problems related to the direction of information. It would be preferable to instead model this type of pipeline after a linear-chain CRF (Lafferty et al. 2001), and allow information to flow in both directions. This could potentially be done by particle filtering (Gordon et al. 1993, Arulampalam et al. 2002), since the weight of each particle would be based on its likelihood at each stage in the pipeline. Another approach is to move to a full, joint model, which is what I do in the next chapter, though the joint model is over fewer tasks than the pipelines presented in this chapter.

The papers on which this chapter and the previous chapter are based helped begin the trend in the NLP community of using sampling-based methods for Bayesian inference, and since their publication there has been extensive work on the subject. MCMC-based Bayesian inference has been used for both unsupervised part-of-speech tagging (Goldwater and Griffiths 2007, Snyder et al. 2008; 2009) and semi-supervised part-of-speech tagging (Toutanova and Johnson 2008); for unsupervised coreference resolution (Haghighi and Klein 2007); for a wide variety of parsing models (Johnson and Griffiths 2007, Johnson et al. 2007, Finkel et al. 2007, Post and Gildea 2009, Cohn et al. 2009, Cohn and Blunsom 2010); and for several machine translation models (DeNero et al. 2008, Cohn and Blunsom 2009, Blunsom et al. 2009, Blunsom and Cohn 2010).

Chapter 5

Joint discriminative learning: parsing and named entity recognition

5.1 Introduction

As discussed in the previous chapter, no NLP task is truly standalone. In order to build high quality systems for complex NLP tasks, such as question answering and textual entailment, it is essential to first have high quality systems for lower level tasks. A good (deep analysis) question answering system requires the data to first be annotated with several types of information: parse trees, named entities, word sense disambiguation, etc. When building such a system, researchers typically follow one of two routes. The simplest, and most common, is to cobble together independent systems, often written by many other independent researchers, for the various types of annotation. While the simplicity is appealing, no information is shared between the different levels of annotation, there is no guarantee that their outputs will be consistent, and often suboptimal heuristic fixes are required. The next most common approach is to pipeline different systems for the different components together. Usually just the 1-best or k-best outputs are propagated at each stage, though more elaborate options are possible, such as the one covered in the previous chapter. While pipelining does guarantee consistency, it is not a complete solution. Errors propagate, and components further down the pipeline will get progressively worse quality inputs, and they have no way of communicating this information back to the earlier stages. Moreover, it's

not always clear what the correct ordering of components should be; ideally they should all be able to influence one another. This calls for a full joint model, which is the topic of this chapter.

In this chapter, I gradually build up to a full joint model of both parsing and named entity recognition. Joint modeling of multiple phenomena has been tried before, but often with limited success. For instance, it has proven very difficult to build a joint model of parsing and semantic role labeling, either with PCFG trees (Sutton and McCallum 2005) or with dependency trees. The CoNLL 2008 shared task (Surdeanu et al. 2008) was intended to be about joint dependency parsing and semantic role labeling, but the top performing systems decoupled the tasks and outperformed the systems which attempted to learn them jointly. Despite these earlier results, I found that combining parsing and named entity recognition modestly improved performance on both tasks, due to their ability to both constrain and influence one another, and in the next chapter I will show how to leverage singly-annotated data to further improve the joint model.

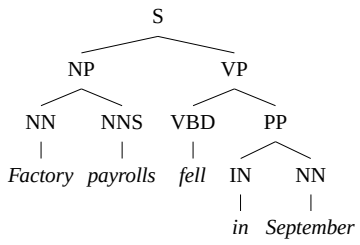
First, I will cover two feature-rich, discriminative, CRF-based parsers, one for constituency trees and one for dependency trees, as they are the backbone for much of the remaining work in this dissertation, including the joint parse and NER model which appears later in the chapter. Then I will show how to convert the discriminative constituency parsing model into a model for nested named entity recognition. With only a handful of exceptions, all NER work to date has focused on a flat structure. When entities are nested inside one another (e.g. *University of California*), common practice is to ignore all but the outermost entity. This strategy throws away a lot of perfectly useful information – information which is both valuable to the user and which should be helpful in identifying and classifying the outermost entities. By utilizing the parser, I explicitly model this nesting structure and the result is a much more useful NER system. The nested NER model also provides a nice segue to the full joint parse and NER model. The joint model operates by augmenting the parse tree with named entity information, and so it also views NER as a parsing problem. The data used for those experiments does not contain nested named entities (such data is difficult to come by), but could very naturally be set up to include nested entities. The full joint model produces an output which has consistent parse structure and named entity spans, and does a better job at both tasks than separate models with the same

features and training data.

5.2 Discriminative parsing

Over the past decade, feature-based discriminative models have become the tool of choice for many natural language processing tasks. Although they take much longer to train than generative models, they typically produce higher performing systems, in large part due to the ability to incorporate arbitrary, potentially overlapping features. However, constituency parsing remains an area dominated by generative methods, due to the computational complexity of the problem. A generative constituency parser can be trained nearly instantaneously, because training only requires taking counts off of a treebank. In contrast, training a discriminative parser typically requires re-parsing the treebank multiple times. Parsing a treebank can be computationally expensive, because it is $O(n^3)$ in sentence length, and there is a large grammar constant.

The work in this section provides a framework for training a feature-rich discriminative parser. I mostly focus on constituency parsing, but the technique can also be applied to dependency parsing, which is covered in section 5.2.3. Unlike prior work, experiments are not restricted to short sentences, but I do provide results both for training and testing on sentences of length ≤ 15 (WSJ15) and for training and testing on sentences of length ≤ 40, allowing previous WSJ15 results to be put in context with respect to most modern parsing literature. The model is a conditional random field-based model. For a rule application, arbitrary features can be defined over the rule categories, span and split point indices, and the words of the sentence. It is well known that constituent length influences parse probability, but PCFGs cannot easily take this information into account. My parser allows features over span length and placement, which can be used to model the strong right-branching tendency of English sentences. Another benefit of the feature-based model is that it effortlessly allows smoothing over previously unseen rules. While the rule may be novel, it will likely contain features which have been seen previously. Practicality comes from three sources. I made use of stochastic optimization methods which allow us to find optimal model parameters with very few passes through the data. On WSJ15, I found no difference in parser performance between using stochastic gradient descent (SGD), and the



(a) PCFG Structure

Phrasal rules

$r_1 = \mathrm{S}_{0,5} \rightarrow \mathrm{NP}_{0,2} \quad \mathrm{VP}_{2,5} \mid$ *Factory payrolls fell in September*

$r_3 = \mathrm{VP}_{2,5} \rightarrow \mathrm{VBD}_{2,3} \quad \mathrm{PP}_{3,5} \mid$ *Factory payrolls fell in September*

...

Lexicon rules

$r_5 = \mathrm{NN}_{0,1} \rightarrow$ *Factory* | *Factory payrolls fell in September*

$r_6 = \mathrm{NNS}_{1,2} \rightarrow$ *payrolls* | *Factory payrolls fell in September*

...

(b) Rules r

Figure 5.1: A parse tree and the corresponding rules over which potentials and features are defined.

more common, but significantly slower, L-BFGS (L-BFGS experiments were too slow to be performed on WSJ40). I also used limited parallelization, and pre-filtering of the chart to avoid the expensive feature computation for rules which cannot tile into complete parses of the sentence. This speed-up does not come with a performance cost; indeed this simpler, faster model attains an F-score of 90.9%, a 14% relative reduction in errors over previous discriminative parsing work evaluated on WSJ15.

5.2.1 A conditional random field context free grammar (CRF-CFG)

My parsing model is based on a conditional random field model, however, unlike previous TreeCRF work (e.g., Cohn and Blunsom (2005), Jousse et al. (2006)) (but like other discriminative CFG work), we do not assume a particular tree structure, and instead find the most likely structure *and* labeling. This is different from most work in graphical models, where the model structure is pre-defined.[1] This is similar to conventional probabilistic context-free grammar (PCFG) parsing, with two exceptions: (a) we maximize *conditional* likelihood of the parse tree, given the sentence, not *joint* likelihood of the tree and sentence; and (b) probabilities are normalized *globally* instead of *locally* – the graphical models depiction of our trees is undirected.

Formally, we have a context-free grammar (CFG) G, which consists of (Manning and

[1] This also differs from most structure-finding work, because in our case the structure is part of the label for a particular instance, and in the typical case the point of finding the structure is to determine the dependencies between the random variables in the model. It is possible to set up our model as a single structure with additional variables which encode the tree constraints, but such a structure would be more difficult to work with, less intuitively easy to understand, and would not offer many benefits.

Schütze 1999):

(i) a set of terminals $\{w^k\}, k = 1, \ldots, V$

(ii) a set of nonterminals $\{N^k\}, k = 1, \ldots, n$

(iii) a designated start symbol *ROOT*

(iv) a set of rules, $\{\rho = N^i \rightarrow \zeta^j\}$, where ζ^j is a sequence of terminals and nonterminals

A PCFG additionally assigns probabilities to each rule ρ such that $\forall i \sum_j P(N^i \rightarrow \zeta^j) = 1$. The conditional random field CFG (CRF-CFG) instead defines local clique potentials $\phi(r|s;\theta)$, where s is the sentence, and r contains a one-level subtree of a tree t, corresponding to a rule ρ, along with relevant information about the span of words which it encompasses, and, if applicable, the split position (see figure 5.1). These potentials are relative to the sentence, unlike a PCFG where rule scores do not have access to words at the leaves of the tree, or even how many words they dominate. To further emphasize this point: *in a PCFG, the score for a rule is always the same, regardless of context, but in a CRF-CFG, the score for the rule is context-dependent*. We then define a conditional probability distribution over entire trees, using the standard CRF distribution, shown in equation 5.1. There is, however, an important subtlety lurking in how we define the partition function. The partition function $Z_{s,\theta}$, which makes the probability of all possible parses sum to unity, is defined over all *structures* as well as all labelings of those structures. We define $\tau(s)$ to be the set of all possible parse trees for the given sentence licensed by the grammar G.

$$\mathrm{P}(t|s;\theta) = \frac{1}{Z_{s,\theta}} \prod_{r \in t} \phi(r|s;\theta) \tag{5.1}$$

where

$$Z_{s,\theta} = \sum_{t \in \tau(s)} \prod_{r \in t'} \phi(r|s;\theta)$$

The above model is not well-defined over all CFGs. Unary rules of the form $N^i \rightarrow N^j$ can form cycles, leading to infinite unary chains with infinite mass. However, it is standard in the parsing literature to transform grammars into a restricted class of CFGs so as to permit efficient parsing. Binarization of rules (Earley 1970) is necessary to obtain cubic

parsing time, and closure of unary chains is required for finding total probability mass (rather than just best parses) (Stolcke 1995). To address this issue, I define our model over a restricted class of CFGs which limits unary chains have no repeated states. This was done by collapsing all allowed unary chains to single unary rules, and disallowing multiple unary rule applications over the same span.[2] I give the details of my binarization scheme in section 5.2.2.

Computing the objective function

Our clique potentials take an exponential form. We have a feature function, represented by $\mathbf{f}(r,s)$, which returns a vector with the value for each feature. I denote the value of feature $featii$ by $f_i(r,s)$ and the model has a corresponding parameter θ_i for each feature. The clique potential function is then:

$$\phi(r|s;\theta) = \exp\{\mathbf{f}(r,s)\cdot\theta\} \tag{5.2}$$

Note that this function is not normalized, and can return any non-negative real number.

The log-likelihood of the training data $\mathscr{D}$, with an additional L_2 regularization term, is then:

$$\mathscr{L}(\mathscr{D};\theta) = \sum_{(t,s)\in\mathscr{D}} \left(\sum_{r\in t} (\mathbf{f}(r,s)\cdot\theta) - \log Z_{s,\theta} \right) - \sum_i \frac{\theta_i^2}{2\sigma^2} \tag{5.3}$$

And the partial derivatives of the log-likelihood, with respect to the model weights are, as usual, the difference between the empirical counts and the model expectations:

$$\frac{\partial\mathscr{L}}{\partial\theta_i} = \sum_{(t,s)\in\mathscr{D}} \left(\sum_{r\in t} f_i(r,s) - E_\theta[f_i|s] \right) - \frac{\theta_i}{\sigma^2} \tag{5.4}$$

The partition function $Z_{s,\theta}$ and the partial derivatives can be efficiently computed with the help of the inside-outside algorithm.[3] $Z_{s,\theta}$ is equal to the inside score of *ROOT* over the span of the entire sentence. To compute the partial derivatives, we walk through each

[2] In my implementation of the inside-outside algorithm, we then need to keep two inside and outside scores for each span: one from before and one from after the application of unary rules.

[3] In our case the values in the chart are the clique potentials which are non-negative numbers, but not probabilities.

rule, and span/split, and multiply the outside score of the parent, the inside score(s) of the child(ren), and the score for that rule and span/split. This value is divided by $Z_{s,\theta}$ to get the normalized probability of that rule in that position. Using the probabilities of each rule application, over each span/split, we can compute the expected feature values (the second term in equation 5.4), by multiplying this probability by the value of the feature corresponding to the weight for which we are computing the partial derivative. The process is analogous to the computation of partial derivatives in linear chain CRFs. The complexity of the algorithm is cubic in sentence length, since we loop over all possible start, end, and split points for phrases in the tree.

Features

As discussed in section 5.2.2, I performed experiments both on sentences of length ≤ 15 and length ≤ 40. All feature development was done on the length 15 corpus, due to the substantially faster train and test times. This has the unfortunate effect that the features are optimized for shorter sentences and less training data, but I found development on the longer sentences to be infeasible. Features are divided into two types: *lexicon features*, which are over words and tags, and *grammar features* which are over the local subtrees and corresponding span/split (both have access to the entire sentence). I ran two kinds of experiments: a discriminatively trained model, which used only the rules and no other grammar features, and a feature-based model which did make use of grammar features. Both models had access to the lexicon features. I viewed this as equivalent to the more elaborate, smoothed unknown word models that are common in many PCFG parsers, such as Klein and Manning (2003).

I preprocessed the words in the sentences to obtain two extra pieces of information: distributional similarity clusters (described in section 2.4.1) and an orthographic word shape (described in section 2.4.2). The full set of features, along with an explanation of notation, is listed in table 5.1.

Lexicon Features	**Grammar Features**	
t		Binary-specific features
$b(t)$	ρ	
$\langle t, w\rangle$	$\langle b(p(r_p)), ds(w_s)\rangle$	$\langle b(p(r_p)), ds(w_{s-1}, dsw_s)\rangle$
$\langle t, lc(w)\rangle$	$\langle b(p(r_p)), ds(w_e)\rangle$	**PP feature:**
$\langle b(t), w\rangle$	**unary?**	if right child is a PP then $\langle r, w_s\rangle$
$\langle b(t), lc(w)\rangle$	**simplified rule:**	**VP features:**
$\langle t, ds(w)\rangle$	base labels of states	if some child is a verb tag, then rule,
$\langle t, ds(w_{-1})\rangle$	**dist sim bigrams:**	with that child replaced by the word
$\langle t, ds(w_{+1})\rangle$	all dist. sim. bigrams below	
$\langle b(t), ds(w)\rangle$	rule, and base parent state	Unaries which span one word:
$\langle b(t), ds(w_{-1})\rangle$	**dist sim bigrams:**	
$\langle b(t), ds(w_{+1})\rangle$	same as above, but trigrams	$\langle r, w\rangle$
$\langle p(t), w\rangle$	**heavy feature:**	$\langle r, ds(w)\rangle$
$\langle t, unk(w)\rangle$	whether the constituent is "big"	$\langle b(p(r)), w\rangle$
$\langle b(t), unk(w)\rangle$	as described in Johnson (2001)	$\langle b(p(r)), ds(w)\rangle$

Table 5.1: Lexicon and grammar features for the CRF-CFG. w is the word and t the tag. r represents a particular rule along with span/split information; ρ is the rule itself, r_p is the parent of the rule; w_b, w_s, and w_e are the first, first after the split (for binary rules) and last word that a rule spans in a particular context. All states, including the POS tags, are annotated with parent information; $b(s)$ represents the base label for a state s and $p(s)$ represents the parent annotation on state s. $ds(w)$ represents the distributional similarity cluster, and $lc(w)$ the lower cased version of the word, and $unk(w)$ the unknown word class.

Parallelization

Unlike Taskar et al. (2004), this algorithm has the advantage of being easily parallelized (see footnote 7 in their paper). Because the computation of both the log-likelihood and the partial derivatives involves summing over each tree individually, the computation can be parallelized by having many clients which each do the computation for one tree, and one central server which aggregates the information to compute the relevant information for a set of trees. Because I used a stochastic optimization method, as discussed in section 2.3.2, we compute the objective for only a small portion of the training data at a time, typically between 15 and 30 sentences. In this case the gains from adding additional clients decrease rapidly, because the computation time is dominated by the longest sentences in the batch.

Chart pre-filtering

Training is also sped up by pre-filtering the chart. On the inside pass of the algorithm one will see many rules which cannot actually be tiled into complete parses. In standard PCFG parsing it is not worth figuring out which rules are viable at a particular chart position and which are not. In our case however this can make a big difference for several reasons. Firstly, we are not just looking up a score for the rule, but must compute all the features (which often requires computationally expensive string manipulations), and dot product them with the feature weights, which is far more time consuming. We also have to do an outside pass as well as an inside one, which is sped up considerably by not considering impossible rule applications. Lastly, we iterate through the data multiple times, so if we can compute this information just once, we will save time on all subsequent iterations on that sentence. I do this by doing an inside-outside pass which is simply boolean valued (and requires no feature computation) to determine which rules are possible at which positions in the chart. I simultaneously compute the features for the possible rules and then save the entire data structure to disk. For all but the shortest of sentences, the disk I/O was easily worth the time compared to re-computation. The first time we see a sentence this method is still about one third faster than if we did not do the pre-filtering, and on subsequent iterations the improvement is closer to tenfold.

5.2.2 Experiments

Data

For all experiments, I trained and tested on the PennTreebank (PTB) (see section 2.2.2). I used the standard splits, training on sections 2 to 21, testing on section 23 and doing development on section 22. Previous work on (non-reranking) discriminative parsing has given results on sentences of length ≤ 15, but most parsing literature gives results on either sentences of length ≤ 40, or all sentences. To properly situate this work with respect to both sets of literature I trained models on both length ≤ 15 (WSJ15) and length ≤ 40 (WSJ40), and we also tested on all sentences using the WSJ40 models. These results also provide a context for interpreting previous work which used WSJ15 and not WSJ40. WSJ15 has 9,753 training sentences, 421 development sentences and 603 test sentences.

Model	States	Binary Rules	Unary Rules
WSJ15	1,428	5,818	423
WSJ15 relaxed	1,428	22,376	613
WSJ40	7,613	28,240	823

Table 5.2: Grammar size for the CRF-CFG models.

WSJ40 has 36,765 training sentences and 2,245 test sentences (all development was done on the length 15 models, due to significantly faster training time). The WSJ40 models were also evaluated on all 2,416 sentences in section 23, with no length restrictions.

Grammar

I used a relatively simple grammar with few additional annotations. Starting with the grammar read off of the training set, I added parent annotations onto each state, including the POS tags, resulting in rules such as S-ROOT $\rightarrow$ NP-S VP-S. I also added head tag annotations to *VP*s, in the same manner as Klein and Manning (2003). Lastly, for the WSJ40 runs I used a simple, right branching binarization where each active state is annotated with its previous sibling and first child. This is equivalent to children of a state being produced by a second order Markov process. For the WSJ15 runs, each active state was annotated with only its first child, which is equivalent to a first order Markov process. See table 5.2 for the number of states and rules produced.

Grammar relaxation

I ran additional *grammar relaxation* experiments, where I added unseen rules to the grammar. One advantage of my model is that smoothing over unseen rules in the grammar happens automatically. For each potential rule application, the clique potential is a function of the dot product of the features and the weights. If no features are present, this equals $e^0 = 1$. The actual probability will depend on the parameters of the model, and the rest of the sentence, but it will not be zero. Moreover, due to how I chose to relax the grammar (discussed below) it will never be the case that an unseen rule has no previously seen features. This is because new rules are only constructed from previously seen states, and we

have features for the states contained in the rule. My motivation for relaxing the grammar was twofold. Adding new rules could make sentences that were previously unparseable become parseable (cf. Petrov et al. (2006)). Our early experiments showed that increasing the number of rules to discriminate between generally improved performance of the final model. To get a sense of what kinds of gains could be hoped for from relaxing the grammar, I did an oracle experiment where I trained the feature-based model, but additionally included in the grammar all the rules present in the development set. I then compared the performance of this model on the development data with the performance of the regular, non-relaxed, feature-based model. The relaxed model increased in performance by 1.5%. Motivated by this gain I set out to find ways to relax the grammar that would produce useful rules without expanding the grammar so much as to make training and testing intolerably slow.

To relax the grammar, I went through each state, and counted the number of rules in which it was present. Then I went through each state in the grammar, and looked at its parent annotation. I tried replacing it with every possible parent annotation, and if the created state was present in more than υ rules, I would find all rules for which the original state was the parent, and create identical copies, replacing the original parent with this alternate state. I experimented with several values of υ, and while the performance on the development set did not vary greatly for values between 5 and 15, I ultimately found 5 to have the best performance.

Experimental results

For both WSJ15 and WSJ40, I trained a generative model; a discriminative model, which used lexicon features, but no grammar features other than the rules themselves; and a feature-based model which had access to all features. For the length 15 data I also did experiments with the relaxed grammar. I used stochastic gradient descent (see section 2.3.2) for these experiments. Using development data, I found that an initial gain of $\eta_0 = 0.1$ worked well for our setting. The length 15 models had a batch size of 15 and I allowed twenty passes through the data.[4] The length 40 models had a batch size of 30 and I allowed

[4]Technically we did not make passes through the data, because we sampled with replacement to get our batches. By this I mean having seen as many sentences as are in the data, despite having seen some sentences

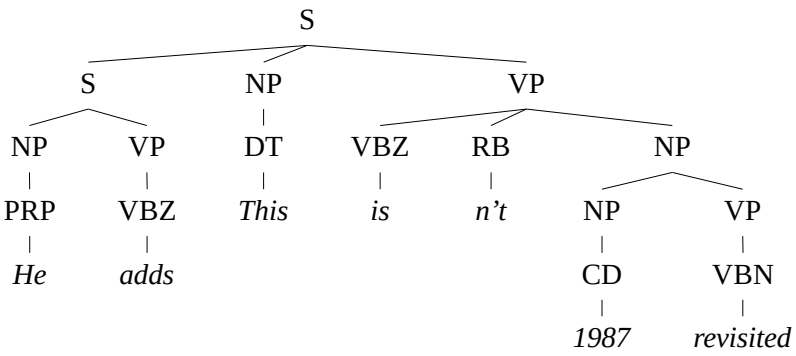
ten passes through the data. I used performance on the development data to decide when the models had converged. Additionally, I provide generative numbers for training on the entire PTB using the same grammar to give a sense of how much performance suffered from the reduced training data (*generative-all* in table 5.4).

The full results for WSJ15 are shown in table 5.3 and for WSJ40 are shown in table 5.4. The evaluation metrics are explained in section 2.1.2; **CB** stands for crossing brackets. The WSJ15 models were each trained on a single Dual-Core AMD Opteron™ using three gigabytes of RAM and no parallelization. The discriminatively trained generative model (*discriminative* in table 5.3) took approximately 12 minutes per pass through the data, while the feature-based model (*feature-based* in table 5.3) took 35 minutes per pass through the data. The feature-based model with the relaxed grammar (*relaxed* in table 5.3) took about four times as long as the regular feature-based model. The discriminatively trained generative WSJ40 model (*discriminative* in table 5.4) was trained using two of the same machines, with 16 gigabytes of RAM each for the clients.[5] It took about one day per pass through the data. The feature-based WSJ40 model (*feature-based* in table 5.4) was trained using four of these machines, also with 16 gigabytes of RAM each for the clients. It took about three days per pass through the data.

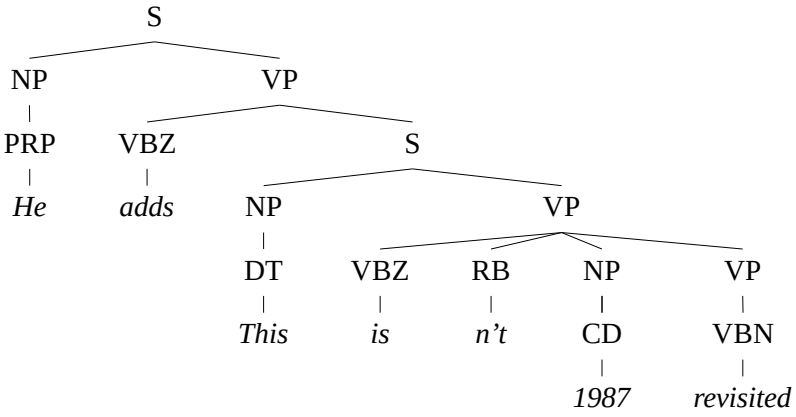
The results clearly show that gains came from both the switch from generative to discriminative training, and from the extensive use of features. In figure 5.2 I show for an example from section 22 the parse trees produced by the generative model and the feature-based discriminative model, and the correct parse. The parse from the feature-based model better exhibits the right branching tendencies of English. This is likely due to the heavy feature (see table 5.1), which encourages long constituents at the end of the sentence. It is difficult for a standard PCFG to learn this aspect of the English language, because the score it assigns to a rule does not take its span into account.

multiple times and some not at all.

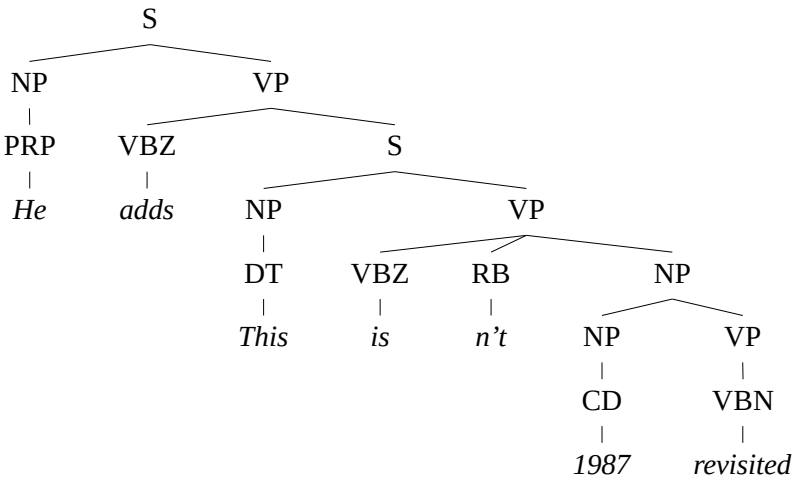
[5]The server does almost no computation.



(a) generative output



(b) feature-based discriminative output



(c) gold parse

Figure 5.2: Example output from our generative and feature-based discriminative models, along with the correct parse.

Model	P	R	F_1	Exact	Avg CB	0 CB
	development set – length ≤ 15					
Taskar 2004	89.7	90.2	90.0	–	–	–
generative	86.9	85.8	86.4	46.2	0.34	81.2
discriminative	89.1	88.6	88.9	55.5	0.26	85.5
feature-based	90.4	89.3	89.9	59.5	0.24	88.3
relaxed	91.2	90.3	90.7	62.1	0.24	88.1
relaxed-oracle	92.2	90.5	91.3	63.1	0.19	89.5
	test set – length ≤ 15					
Taskar 2004	89.1	89.1	89.1	–	–	–
Turian 2007	89.6	89.3	89.4	–	–	–
generative	87.6	85.8	86.7	49.2	0.33	81.9
discriminative	88.9	88.0	88.5	56.6	0.32	85.0
feature-based	91.1	90.2	90.6	61.3	0.24	86.8
relaxed	91.4	90.4	90.9	62.0	0.22	87.9

Table 5.3: Development and test set results, training and testing on sentences of length ≤ 15 from the PennTreebank (WSJ15).

5.2.3 CRF-based dependency parsing

In addition to the constituency parser just described, I also built a CRF-based dependency parsing model, optimizing the likelihood of the parse, conditioned on the words and part-of-speech tags of the sentence. At the heart of the model is the Eisner dependency grammar chart-parsing algorithm (Eisner 1996), which allows for efficient computation of inside and outside scores, and is described in section 2.1.3. The Eisner algorithm, originally designed for generative parsing, decomposes the probability of a dependency parse into the probabilities of each attachment of a dependent to its parent, and the probabilities of each parent stopping taking dependents. These probabilities can be conditioned on the child, parent, and direction of the dependency. I used a slight modification of the algorithm which allows each probability to also be conditioned on whether there is a previous dependent. While the unmodified version of the algorithm includes stopping probabilities, conditioned on the parent and direction, they have no impact on which parse for a particular sentence is most likely, because all words must eventually stop taking dependents. However, in the modified version, the stopping probability is also conditioned on whether or not there is a previous dependent, so this probability does make a difference.

Model	P	R	F_1	Exact	Avg CB	0 CB
	test set – length ≤ 40					
Petrov 2007	–	–	88.8	–	–	–
generative	83.5	82.0	82.8	25.5	1.57	53.4
generative-all	83.6	82.1	82.8	25.2	1.56	53.3
discriminative	85.1	84.5	84.8	29.7	1.41	55.8
feature-based	89.2	88.8	89.0	37.3	0.92	65.1
	test set – all sentences					
Petrov 2007	–	–	88.3	–	–	–
generative	82.8	81.2	82.0	23.8	1.83	50.4
discriminative	84.2	83.7	83.9	27.8	1.67	52.8
feature-based	88.2	87.8	88.0	35.1	1.15	62.3

Table 5.4: Test set results, training on sentences of length ≤ 40 from the PennTreebank. The *generative-all* results were trained on all sentences regardless of length

While the original Eisner algorithm computes locally normalized probabilities for each attachment decision, our model computes unnormalized scores. From a graphical models perspective, our parsing model is undirected, while the original model is directed.[6] The score for a particular tree decomposes the same way in our model as in the original Eisner model, but it is globally normalized instead of locally normalized. Using the inside and outside scores we can compute partial derivatives for the feature weights, as well as the value of the normalizing constant needed to determine the probability of a particular parse. This is done in a manner completely analogous to the constituency parser just described. Partial derivatives and the function value are all that is needed to find the optimal feature weights using L-BFGS.[7]

Features are computed over each attachment and stopping decision, and can be conditioned on the parent, dependent (or none, if it is a stopping decision), direction of attachment, whether there is a previous dependent in that direction, and the words and parts of speech of the sentence. I used the same features as McDonald et al. (2005b), augmented with information about whether or not a dependent is the first dependent (information they did not have). The unsupervised dependency parsing work of Klein and Manning (2004)

[6]The dependencies themselves are still *directed* in both cases, it is just the underlying graphical model used to compute the likelihood of a parse which changes from a directed model to an undirected model.

[7]It was computationally feasible to use L-BFGS in this case, because untyped dependency parsing is much faster than CRF-based parsing.

Dependency Parsing

	Training		Testing		
	Range	# Sent	Range	# Sent	ACCURACY
ABC	0–55	1195	56–69	199	83.32%
CNN	0–375	5092	376–437	1521	85.53%
MNB	0–17	509	18–25	245	77.06%
NBC	0–29	552	30–39	149	76.21%
PRI	0–89	1707	90–112	394	87.65%
VOA	0–198	1512	199–264	383	89.17%

Table 5.5: CRF-based dependency parsing results. Performance is measured as unlabeled attachment accuracy.

similarly conditioned on the presence or absence of a previous dependent.

Experimental results

For the dependency parsing experiments, I used OntoNotes (Release 2.0), as described in section 2.2.4. I converted the PCFG trees into dependency trees using the Collins head rules (Collins 2003). For each of the six domains (ABC, CNN, MNB, NBC, PRI, and VOA), I aimed for an 75/25 data split, but because I divided the data using the provided sections, this split was fairly rough. The number of training and test sentences for each domain are specified in the table 5.5, along with the results.

5.2.4 Related work

Previous work on discriminative constituency parsing falls under one of three approaches. One approach does discriminative reranking of the *k*-best list of a generative parser, still usually depending highly on the generative parser score as a feature (Collins 2000, Charniak and Johnson 2005). A second group of papers does parsing by a sequence of independent, discriminative decisions, either greedily or with use of a small beam (Ratnaparkhi 1997, Henderson 2004).

The present work falls under the third thread of work, where joint inference via dynamic programming algorithms is used to train models and to attempt to find the globally best parse. Prior to 2008, work in this context had mainly been limited to use of artificially

short sentences due to exorbitant training and inference times. The most similar prior work in this category is the discriminative constituency parser of Johnson (2001), who did discriminative training of a generative PCFG. The model was quite similar to this one, except that it did not incorporate any features and it required the parameters (which were just scores for rules) to be locally normalized, as with a generatively trained model. Due to training time, they used the ATIS treebank corpus of air travel reservations, which is much smaller than even WSJ15, with only 1,088 training sentences, 294 testing sentences, and an average sentence length of around 11. They found no significant difference in performance between their generatively and discriminatively trained parsers. There are two probable reasons for this result. The training set is very small, and it is a known fact that generative models tend to work better for small datasets and discriminative models tend to work better for larger datasets (Ng and Jordan 2002). Additionally, they made no use of features, one of the primary benefits of discriminative learning.

Taskar et al. (2004) took a large margin approach to discriminative learning. They only reported results on short sentences – they used the PennTreebank, but restricted the corpus to sentences of length 15 and less (WSJ15). More recent is the work of Turian and Melamed (2006) and Turian et al. (2007), who also only reported results on WSJ15, and which improved both the accuracy of Taskar et al. (2004). They define a simple linear model, use boosted decision trees to select feature conjunctions, and a line search to optimize the parameters. They use an agenda parser, and define their atomic features, from which the decision trees are constructed, over the entire state being considered. While they make extensive use of features, their setup is much more complex than mine and takes substantially longer to train – up to 5 days on WSJ15 – while achieving only small gains over Taskar et al. (2004).

There are two recent exceptions to the trend of only reporting discriminative parsing results on artificially short sentences. These were both developed at the same time as the work presented here. The first example is the work of Petrov and Klein (2008), who discriminatively train parameters for a grammar with latent variables, and do not restrict themselves to short sentences. Following up on their previous work on grammar splitting (Petrov et al. 2006), they do discriminative parsing with latent variables, which requires them to optimize a non-convex function. They iteratively refine their grammar, but when judging

refinements they train the intermediate models discriminatively instead of generatively. Instead of using a stochastic optimization technique, they use L-BFGS, but do coarse-to-fine pruning to approximate their gradients and log-likelihood. Because they were focusing on grammar splitting they did not employ any features, and only small gains from switching from generative to discriminative training. It has been shown on other NLP tasks that modeling improvements, such as the switch from generative training to discriminative training, usually provide much smaller performance gains than the gains possible from good feature engineering. For example, in Lafferty et al. (2001), when switching from a generatively trained hidden Markov model (HMM) to a discriminatively trained, linear-chain, conditional random field (CRF) for part-of-speech tagging, their error drops only slightly from 5.7% to 5.6%. When they add in only a small set of orthographic features, their CRF error rate drops considerably more to 4.3%, and their out-of-vocabulary error rate drops by more than half. This is further supported by Johnson (2001), who saw no parsing gains when switching from generative to discriminative training, and by Petrov and Klein (2008) who saw only small gains of around 0.7% for their final model when switching training methods. The second piece of work, presented in Carreras et al. (2008), is also a CRF-based discriminative parser, but with several crucial differences. The authors use a different formalism, a tree adjoining grammar (TAG), meaning that they decompose trees in a different manner, and this decomposition determines what pieces of substructure the features are defined over. They also use a dependency parsing model to efficiently prune down the search space to make training and inference computationally feasible.

There are several other high performing dependency parsers. McDonald et al. (2005b) used the MIRA algorithm to train a dependency parser using maximum spanning trees. The MALT parser (Nivre et al. 2006) is a history-based inductive dependency parser which has been shown to perform well on a variety of languages.

5.3 Nested named entity recognition

Named entity recognition, introduced in section 2.1.1, is the task of finding entities, such as people and organizations, in text. Frequently, entities are nested within each other, such as *Bank of China* and *University of Washington*, both ORGs with nested LOCs. Nested

entities are also common in biomedical data, where different biological entities of interest are often composed of one another. In the GENIA corpus (Ohta et al. 2002, Kim et al. 2003), which is labeled with entity types such as PROTEIN and DNA, roughly 17% of entities are embedded within another entity. In the AnCora corpus of Spanish and Catalan newspaper text (Martí et al. 2007), nearly half of the entities are embedded. However, work on named entity recognition (NER) has almost entirely ignored nested entities and instead chosen to focus on the outermost entities.

This omission of nested entities has largely been for practical, not ideological, reasons. Most corpus designers have chosen to skirt the issue entirely, and have annotated only the outermost entities. The widely used CoNLL 2003 (Sang and Meulder 2003), MUC-6 (Sundheim 1996), and MUC-7 (Chinchor 1998) NER corpora, composed of British and American newswire, are all flatly annotated. The GENIA corpus contains nested entities, but the JNLPBA 2004 shared task (Collier et al. 2004), which utilized the corpus, removed all embedded entities for the evaluation. To my knowledge, the only shared task which has included nested entities is the SemEval 2007 Task 9 (Márquez et al. 2007b), which used a subset of the AnCora corpus. However, in that task, all entities corresponded to a pre-determined set of part-of-speech tags or noun phrases in the provided syntactic structure, and no participant directly addressed the nested nature of the data.

Another reason for the lack of focus on nested NER is technological. The NER task arose in the context of the MUC workshops, as small chunks which could be identified by finite state models or gazetteers. This then led to the widespread use of sequence models, first hidden Markov models, then maximum entropy Markov models (Borthwick 1999, McCallum et al. 2000), and, more recently, linear chain CRFs (Lafferty et al. 2001). All of these models suffer from an inability to easily model nested entities.

This section presents a novel solution to the problem of nested named entity recognition, which directly utilizes the discriminative constituency parser just described. The model explicitly represents the nested structure, allowing entities to be influenced not just by the labels of the words surrounding them, as in a CRF, but also by the entities contained in them, and in which they are contained. Each sentence is represented as a parse tree, with the words as leaves, and with phrases corresponding to each entity (and a ROOT node which joins the entire sentence). The trees look just like syntactic constituency trees, such

as those in the PennTreebank (Marcus et al. 1993), but they tend to be much flatter. Part-of-speech tags can be included in the tree, and be jointly modeled with the named entities. Once sentences are converted into parse trees, they are used as input to train the discriminative constituency parser described in the previous section. I found that on top-level entities, the nested NER model does just as well as more conventional methods. When evaluating on *all* entities the model does very well, with F-scores ranging from slightly worse than performance on top-level only, to substantially better than top-level only.

5.3.1 Related work on nested named entity recognition

There is a large body of work on named entity recognition, but very little of it addresses nested entities. Early work on the GENIA corpus (Kazama et al. 2002, Tsuruoka and Tsujii 2003) only worked on the innermost entities. This was soon followed by several attempts at nested NER in GENIA which built hidden Markov models over the innermost named entities, and then used a rule-based post-processing step to identify the named entities containing the innermost entities (Shen et al. 2003, Zhang et al. 2004, Zhou et al. 2004). Zhou (2006) used a more elaborate model for the innermost entities, but then used the same rule-based post-processing method on the output to identify non-innermost entities. Gu (2006) focused only on proteins and DNA, by building separate binary SVM classifiers for innermost and outermost entities for those two classes.

Several techniques for nested NER in GENIA were presented in Alex et al. (2007). Their first approach was to layer CRFs, using the output of one as the input to the next. For inside-out layering, the first CRF would identify the innermost entities, the next layer would be over the words and the innermost entities to identify second-level entities, etc. For outside-in layering, the first CRF would identify outermost entities, and then successive CRFs would identify increasingly nested entities. They also tried a cascaded approach, with separate CRFs for each entity type. The CRFs would be applied in a specified order, and then each CRF could utilize features derived from the output of previously applied CRFs. This technique has the problem that it cannot identify nested entities of the same type; this happens frequently in the data, such as the nested PROTEINs at the beginning of the sentence in figure 5.3. They also tried a joint labeling approach, where they trained a

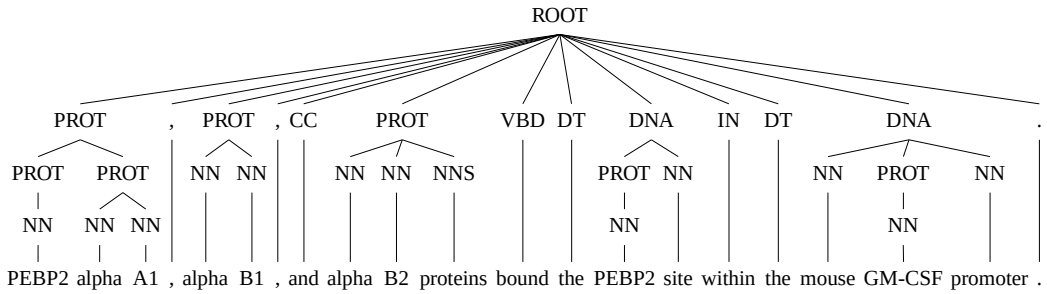


Figure 5.3: An example of a tree representation over nested named entities. The sentence is from the GENIA corpus. *PROT* is short for *PROTEIN*.

single CRF, but the label set was significantly expanded so that a single label would include all of the entities for a particular word. Their best results were from the cascaded approach.

Byrne (2007) took a different approach, which she used on historical archive text. She modified the data by concatenating adjacent tokens (up to length six) into potential entities, and then labeled each concatenated string using the C&C tagger (Curran and Clark 1999). When labeling a string, the "previous" string was the one-token-shorter string containing all but the last token of the current string. For single tokens the "previous" token was the longest concatenation starting one token earlier.

SemEval 2007 Task 9 (Márquez et al. 2007b) included a nested NER component, as well as noun sense disambiguation and semantic role labeling. However, the parts of speech and syntactic tree were given as part of the input, and named entities were specified as corresponding to noun phrases in the tree, or particular parts of speech. This restriction substantially changes the task. Two groups participated in the shared task, but only one Márquez et al. (2007a) worked on the named entity component. They used a multi-label AdaBoost.MH algorithm, over phrases in the parse tree which, based on their labels, could potentially be entities.

Finally, McDonald et al. (2005a) presented a technique for labeling potentially overlapping segments of text, based on a large margin, multi-label classification algorithm. Their method could be used for nested named entity recognition, but the experiments they performed were on joint (flat) NER and noun phrase chunking.

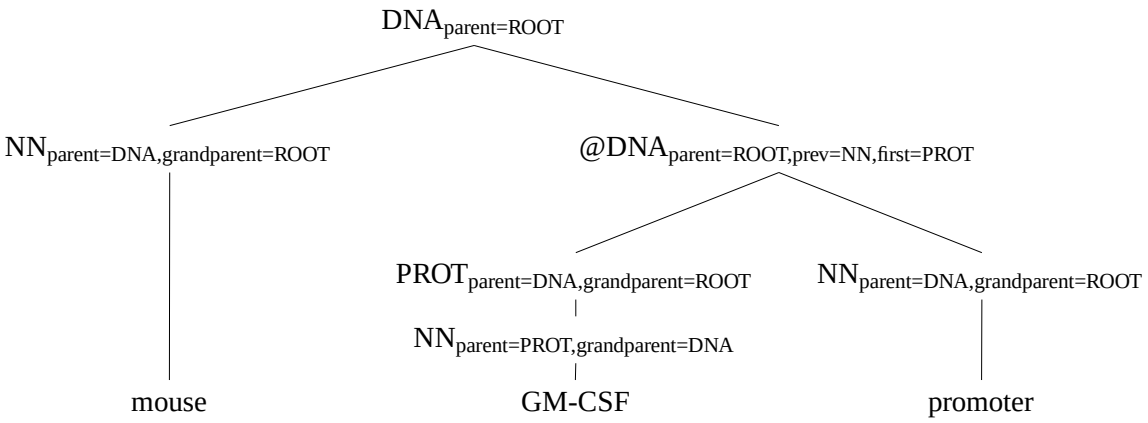


Figure 5.4: An example of a subtree after it has been annotated and binarized. Features are computed over this representation. An @ indicates a chart parser active state (incomplete constituent).

5.3.2 Nested named entity recognition as parsing

My model for nested NER is quite simple – I represent each sentence as a constituency tree, with each named entity corresponding to a phrase in the tree, along with a root node which connects the entire sentence. No additional syntactic structure is represented. I also model the parts of speech as preterminals, and the words themselves as the leaves. See figure 5.3 for an example of a named entity tree. Each node is then annotated with both its parent and grandparent labels, which allows the model to learn how entities nest. I binarize the trees in a right-branching manner, and then build features over the labels, unary rules, and binary rules. I also use first-order horizontal Markovization, which allows us to retain some information about the previous node in the binarized rule. See figure 5.4 for an example of an annotated and binarized subtree. Once each sentence has been converted into a tree, the trees are used to train a discriminative constituency parser.

It is worth noting that if you use the model on data that does not have any nested entities, then it is precisely equivalent to a semi-CRF (Sarawagi and Cohen 2004, Andrew 2006), but with no length restriction on entities. Like a semi-CRF, we are able to define features over entire entities of arbitrary length, instead of just over a small, fixed window of words like a regular linear chain CRF.

Part-of-speech tags are modeled jointly with the named entities, though the model also

works without them. Possible part-of-speech tags are determined based on distributional similarity clusters (described in section 2.4.1; cluster training data for the individual experiments are given in the appropriate sections). I allowed each word to be labeled with any part-of-speech tag seen in the data with any other word in the same cluster. Because the part-of-speech tags are annotated with the parent (and grandparent) labels, they restrict what, if any, entity types a word can be labeled with. Many words, such as verbs, cannot be labeled with any entities. I also limited the grammar based on the rules observed in the data. The rules whose children include part-of-speech tags restrict the possible pairs of adjacent tags. Interestingly, the restrictions imposed by this joint modeling (both observed word/tag pairs and observed rules) actually results in much faster inference (and therefore faster train and test times) than a model over named entities alone, because the space of possible parse trees has been significantly reduced. This is different from most work on joint modeling of multiple levels of annotation (including the work later in this chapter), which usually results in significantly slower inference.

The biggest drawback to this model is runtime. The algorithm is $O(n^3)$ in sentence length. Training on all of GENIA took approximately 23 hours for the nested model and 16 hours for the semi-CRF. A semi-CRF *with* an entity length restriction, or a regular CRF, would both have been faster. At runtime, the nested model for GENIA tagged about 38 words per second, while the semi-CRF tagged 45 words per second. For comparison, a first-order linear-chain CRF trained with similar features on the same data can tag about 4,000 words per second.

5.3.3 Features

When designing features, I first made ones similar to the features typically designed for a first-order CRF, and then added features which are not possible in a CRF, but are possible in the enhanced representation. This includes features over entire entities, features which directly model nested entities, and joint features over entities and parts of speech. When features are computed over each label, unary rule, and binary rule, the feature function is aware of the rule span and split.

Each word is labeled with its distributional similarity cluster (*distsim*), and a string

Local Features		**Pairwise Features**
label_i	distsim_i + distsim_{i-1} + label_i	label_{i-1} + label_i
word_i + label_i	shape_i + shape_{i+1} + label_i	word_i + label_{i-1} + label_i
word_{i-1} + label_i	shape_{i-1} + shape_i + label_i	word_{i-1} + label_{i-1} + label_i
word_{i+1} + label_i	word_{i-1} + shape_i + label_i	word_{i+1} + label_{i-1} + label_i
distsim_i + label_i	shape_i + word_{i+1} + label_i	distsim_i + label_{i-1} + label_i
distsim_{i-1} + label_i	words in a 5 word window	distsim_{i-1} + label_{i-1} + label_i
distsim_{i+1} + label_i	prefixes up to length 6	distsim_{i+1} + label_{i-1} + label_i
shape_i + label_i	suffixes up to length 6	distsim_{i-1} + distsim_i + label_{i-1} + label_i
shape_{i-1} + label_i		shape_i + label_{i-1} + label_i
shape_{i+1} + label_i		shape_{i-1} + label_{i-1} + label_i
		shape_{i+1} + label_{i-1} + label_i
		shape_{i-1} + shape_i + label_{i-1} + label_i
		shape_{i-1} + shape_{i+1} + label_{i-1} + label_i

Table 5.6: The local and pairwise NER features used in all of our experiments. Consult the text for a full description of all features, which includes feature classes not in this table.

indicating orthographic information (*shape*) (see section 2.4). Subscripts represent word position in the sentence. In addition to those below, we include features for each fully annotated label and rule.

Local named entity features. Local named entity features are over the label for a single word. They are equivalent to the local features in a linear chain CRF. However, unlike in a linear chain CRF, if a word belongs to multiple entities (e.g., a word which is both a LOCATION and ORGANIZATION) then the local features are computed for each entity. Local features are also computed for words not contained in any entity. Local features are in table 5.6.

Pairwise named entity features. Pairwise features are over the labels for adjacent words, and are equivalent to the edge features in a linear chain CRF. They can occur when pairs of words have the same label, or over entity boundaries where the words have different labels. Like with the local features, if a pair of words are contained in, or straddle the border of, multiple entities, then the features are repeated for each. The pairwise features I use are shown in table 5.6.

Embedded named entity features. Embedded named entity features occur in binary rules where one entity is the child of another entity. For our embedded features, we replicated the pairwise features, except that the embedded named entity was treated as one of the words, where the "word" (and other annotations) indicates the type of entity, and not the actual string that is the entity. For instance, in the subtree in figure 5.4, we would compute $word_i$+$label_{i-1}$+$label_i$ as *PROT-DNA-DNA* for $i = 18$ (the index of the word *GM-CSF*). The normal pairwise feature at the same position would be *GM-CSF-DNA-DNA.*

Whole entity features. I created four whole entity features: the entire phrase; the preceding and following word; the preceding and following distributional similarity tags; and the preceding distributional similarity tag with the following word.

Local part of speech features. I used the same part-of-speech features as in the discriminative constituency parser (see *lexicon features* in table 5.1).

Joint named entity and part of speech features. For the joint features I replicated the part-of-speech features, but included the parent of the part-of-speech, which either is the innermost entity type, or would indicate that the word is not in any entities.

5.3.4 GENIA experiments

I performed two sets of experiments to validate my model. The first set, covered in this section, is over biomedical data, and the second set, covered in the following section, is over Spanish and Catalan newspaper text. I designed the experiments to show that my model works just as well as standard models on outermost entities, the typical NER task, and also works well on nested entities.

Data

I performed experiments on the GENIA corpus, introduced in section 2.2.1. This corpus contains 2000 Medline abstracts (≈500k words), annotated with 36 different kinds of biological entities, and with part-of-speech tags. Previous NER work using this corpus

has employed 10-fold cross-validation for evaluation. I wanted to explore different model variations (e.g., level of Markovization, and different sets of distributional similarity clusterings) and feature sets, so I needed to set aside a development set. I split the data by putting the first 90% of sentences into the training set, and the remaining 10% into the test set. This is the exact same split used to evaluate part-of-speech tagging in Tsuruoka et al. (2005). For development I used the first half of the data to train, and the next quarter of the data to test.[8] I made the same modifications to the label set as the organizers of the JNLPBA 2004 shared task (Collier et al. 2004). They collapsed all DNA subtypes into DNA; all RNA subtypes into RNA; all PROTEIN subtypes into PROTEIN; kept CELL_LINE and CELL_TYPE; and removed all other entities. However, they also removed all embedded entities, while I kept them.

As discussed in section 5.3.2, I annotated each word with a distributional similarity cluster. I used 200 clusters, trained using 200 million words from PubMed abstracts. During development, I found that fewer clusters resulted in slower inference with no improvement in performance.

Experimental setup

I ran several sets of experiments, varying between all entities, or just top-level entities, for training and testing. As discussed in section 5.3.2, if we train on just top-level entities then the model is equivalent to a semi-CRF. Semi-CRFs are state-of-the-art and provide a good baseline for performance on just the top-level entities. Semi-CRFs are (theoretically) strictly better than regular, linear chain CRFs, because they can use all of the features and structure of a linear chain CRF, but also utilize whole-entity features (Andrew 2006). I also evaluated the semi-CRF model on all entities. This may seem like an unfair evaluation, because the semi-CRF has no way of recovering the nested entities, but I wanted to illustrate just how much information is lost when using a flat representation.

[8] This split may seem strange: I had originally intended a $50/25/25$ train/dev/test split, until I found the previously used $90/10$ split.

GENIA – Testing on All Entities

	# Test Entities	Nested NER Model (train on all entities) Prec	Recall	F_1	Semi-CRF Model (train on top-level entities) Prec	Recall	F_1
Protein	3034	79.04	69.22	73.80	78.63	64.04	70.59
DNA	1222	69.61	61.29	65.19	71.62	57.61	63.85
RNA	103	86.08	66.02	74.73	79.27	63.11	70.27
Cell Line	444	73.82	56.53	64.03	76.59	59.68	67.09
Cell Type	599	68.77	65.44	67.07	72.12	59.60	65.27
Overall	5402	75.39	**65.90**	**70.33**	**76.17**	61.72	68.19

Table 5.7: Named entity results on GENIA, evaluating on all entities.

GENIA – Testing on Top-level Entities Only

	# Test Entities	Nested NER Model (train on all entities) Precision	Recall	F_1	Semi-CRF Model (train on top-level entities) Precision	Recall	F_1
Protein	2592	78.24	72.42	75.22	76.16	72.61	74.34
DNA	1129	70.40	64.66	67.41	71.21	62.00	66.29
RNA	103	86.08	66.02	74.73	79.27	63.11	70.27
Cell Line	420	75.54	58.81	66.13	76.59	63.10	69.19
Cell Type	537	69.36	70.39	69.87	71.11	65.55	68.22
Overall	4781	**75.22**	**69.02**	**71.99**	74.57	68.27	71.28

Table 5.8: Named entity results on GENIA, evaluating on only top-level entities.

Results

Named entity results when evaluating on all entities are shown in table 5.7 and when evaluating on only top-level entities are shown in table 5.8. The nested model outperforms the flat semi-CRF on both top-level entities and all entities.

While not my main focus, I also evaluated the models on parts of speech. The model trained on just top level entities achieved POS accuracy of 97.37%, and the one trained on all entities achieved 97.25% accuracy. The GENIA tagger (Tsuruoka et al. 2005) achieves 98.49% accuracy using the same train/test split.

Additional JNLPBA 2004 experiments

Because I could not directly compare my results on the NER portion of the GENIA corpus with any other work, I also evaluated on the JNLPBA corpus. This corpus was used in a shared task for the BioNLP workshop at Coling in 2004 (Collier et al. 2004). They used the entire GENIA corpus for training, and modified the label set as discussed in section 5.3.4. They also removed all embedded entities, and kept only the top-level ones. They then annotated new data for the test set. This dataset has no nested entities, but because the training data is GENIA I can still train my model on the data annotated with nested entities, and then evaluate on their test data by ignoring all embedded entities found by my named entity recognizer. This experiment allows me to show that my named entity recognizer works well on top-level entities, by comparing it with prior work. The model also produces part-of-speech tags, but the test data is not annotated with part-of-speech tags, so I cannot give part-of-speech tagging results on this dataset.

One difficulty I had with the JNLPBA experiments was with tokenization. The version of GENIA distributed for the shared task is tokenized differently from the original GENIA corpus, but I needed to train on the original corpus as it is the only version with nested entities. I tried my best to retokenize the original corpus to match the distributed data, but did not have complete success. It is worth noting that the data is actually tokenized in a manner which allows a small amount of "cheating." Normally, hyphenated words, such as *LPS-induced*, are tokenized as one word. However, if the portion of the word before the hyphen is in an entity, and the part after is not, such as *BCR-induced*, then the word is split into two tokens: *BCR* and *-induced*. Therefore, when a word starts with a hyphen it is a very strong indicator that the prior word is the last word of an entity. Because the train and test data for the shared task do not contain nested entities, fewer words are split in this manner than in the original data. I did not intentionally exploit this fact in my feature design, but it is probable that some of the orthographic features "learned" this fact anyway. This anomaly probably harmed our results overall, because some hyphenated words, which straddled boundaries in nested entities, and would have been split in the original corpus (and were split in our training data), were not split in the test data, prohibiting our model from properly identifying them.

JNLPBA 2004 – Testing on Top-level Entities Only

	# Test Entities	Nested NER Model (train on all entities) Prec	Recall	F_1	Semi-CRF Model (train top-level entities) Prec	Recall	F_1	Zhou & Su (2004) Prec	Recall	F_1
Protein	4944	67.0	74.6	70.6	68.2	62.7	65.3	69.0	79.2	73.8
DNA	1030	63.0	66.5	64.7	65.5	52.2	58.1	66.8	73.1	69.8
RNA	115	63.1	60.9	62.0	64.6	61.7	63.1	64.7	63.6	64.1
Cell line	487	49.9	60.8	54.8	49.6	52.2	50.9	53.9	65.8	59.2
Cell type	1858	75.1	65.3	69.9	73.3	55.8	63.4	78.1	72.4	75.1
Overall	8434	66.8	70.6	68.6	67.5	59.3	63.1	**69.4**	**76.0**	**72.6**

Table 5.9: Named entity results on the JNLPBA 2004 shared task data. Zhou and Su (2004) was the best system at the shared task, and is still state-of-the-art on the dataset.

For this experiment, I retrained my model on the entire, retokenized, GENIA corpus. I also retrained the distributional similarity model on the retokenized data. Once again, I trained one model on the nested data, and one on just the top-level entities, so that I can compare performance of both models on the top-level entities. Full results are shown in table 5.9, along with the current state-of-the-art (Zhou and Su 2004). Besides the tokenization issues harming our performance, Zhou and Su (2004) also employed clever post-processing to improve their results.

5.3.5 AnCora experiments

Data

I performed experiments on the NER portion of AnCora (Martí et al. 2007), introduced in section 2.2.1. Recall that this corpus has Spanish and Catalan portions, and I evaluated on both. The data is also annotated with part-of-speech tags, parse trees, semantic roles and word senses. The corpus annotators made a distinction between *strong* and *weak* entities. They define *strong* named entities as "a word, a number, a date, or a string of words that refer to a single individual entity in the real world." If a strong NE contains multiple words, it is collapsed into a single token. *Weak* named entities, "consist of a noun phrase, being it

simple or complex" and must contain a *strong* entity.[9] Figure 2.4 shows an example from the corpus with both strong and weak entities. The entity types present are PERSON, LOCATION, ORGANIZATION, DATE, NUMBER, and OTHER. Weak entities are very prevalent; 47.1% of entities are embedded.

Experimental setup

The part-of-speech tags provided in the data include detailed morphological information, using a similar annotation scheme to the Prague TreeBank (Hana and Hanová 2002). There are around 250 possible tags, and experiments on the development data with the full tagset where unsuccessful. I removed all but the first two characters of each POS tag, resulting in a set of 57 tags which more closely resembles that of the PennTreebank (Marcus et al. 1993). All reported results use this modified version of the POS tag set.

The model took only the words as input, none of the extra annotations. For both languages I trained a 200 cluster distributional similarity model over the words in the corpus. I performed the same set of experiments on AnCora as I did on GENIA.

Results and discussion

The full results for Spanish when testing on all entities are shown in table 5.10, and for only top-level entities are shown in table 5.11. For part-of-speech tagging, the nested model achieved 95.93% accuracy, compared with 95.60% for the flatly trained model. The full results for Catalan when testing on all entities are shown in table 5.12, and for only top-level entities are shown in table 5.13. Part-of-speech tagging results were even closer on Catalan: 96.62% for the nested model, and 96.59% for the flat model.

It is not surprising that the models trained on all entities do significantly better than the flatly trained models when testing on all entities. The story is a little less clear when testing on just top-level entities. In this case, the nested model does 4.38% better than the flat model on the Spanish data, but 2.45% worse on the Catalan data. But, the overall picture is the same as for GENIA: modeling the nested entities does not, on average, reduce

[9] Arguably, this represents a misunderstanding of the term "*named* entity", and weak named entities should just be termed *entities* or *referential expressions*.

AnCora Spanish – Testing on All Entities

	# Test Entities	Nested NER Model (train on all entities) Precision	Recall	F_1	Semi-CRF Model (train on top-level entities) Precision	Recall	F_1
Person	1778	65.29	78.91	71.45	75.10	32.73	45.59
Organization	2137	86.43	56.90	68.62	47.02	26.20	33.65
Location	1050	78.66	46.00	58.05	84.94	13.43	23.19
Date	568	87.13	83.45	85.25	79.43	29.23	42.73
Number	991	81.51	80.52	81.02	66.27	28.15	39.52
Other	512	17.90	64.65	28.04	10.77	16.60	13.07
Overall	7036	**62.38**	**66.87**	**64.55**	51.06	25.77	34.25

Table 5.10: Named entity results on the Spanish portion of AnCora, evaluating on all entities.

AnCora Spanish – Testing on Top-level Entities Only

	# Test Entities	Nested NER Model (train on all entities) Precision	Recall	F_1	Semi-CRF Model (train on top-level entities) Precision	Recall	F_1
Person	1050	57.42	66.67	61.70	71.23	52.57	60.49
Organization	1060	77.38	40.66	53.31	44.33	49.81	46.91
Location	279	72.49	36.04	48.15	79.52	24.40	37.34
Date	290	72.29	57.59	64.11	71.77	51.72	60.12
Number	519	57.17	49.90	53.29	54.87	44.51	49.15
Other	541	11.30	38.35	17.46	9.51	26.88	14.04
Overall	3739	**50.57**	**49.72**	**50.14**	46.07	44.61	45.76

Table 5.11: Named entity results on the Spanish portion of AnCora, evaluating on only top-level entities.

AnCora Catalan – Testing on All Entities

	# Test	**Nested NER Model** (train all entities)			**Semi-CRF Model** (train top-level entities only)		
	Entities	Precision	Recall	F_1	Precision	Recall	F_1
Person	1303	89.01	50.35	64.31	70.08	46.20	55.69
Organization	1781	68.95	83.77	75.64	65.32	41.77	50.96
Location	1282	76.78	72.46	74.56	75.49	36.04	48.79
Date	606	84.27	81.35	82.79	70.87	38.94	50.27
Number	1128	86.55	83.87	85.19	75.74	38.74	51.26
Other	596	85.48	8.89	16.11	64.91	6.21	11.33
Overall	6696	**78.09**	**68.23**	**72.83**	70.39	37.60	49.02

Table 5.12: Named entity results on the Catalan portion of AnCora, evaluating on all entities.

AnCora Catalan – Testing on Top-level Entities Only

	# Test	**Nested NER Model** (train all entities)			**Semi-CRF Model** (train top-level entities only)		
	Entities	Precision	Recall	F_1	Precision	Recall	F_1
Person	801	67.44	47.32	55.61	62.63	67.17	64.82
Organization	899	52.21	74.86	61.52	57.68	73.08	64.47
Location	659	54.86	67.68	60.60	62.42	57.97	60.11
Date	296	62.54	66.55	64.48	59.46	66.89	62.96
Number	528	62.35	70.27	66.07	63.08	68.94	65.88
Other	342	49.12	8.19	14.04	45.61	7.60	13.03
Overall	3525	57.67	59.40	58.52	**60.53**	**61.42**	**60.97**

Table 5.13: Named entity results on the Catalan portion of AnCora, evaluating on only top-level entities.

performance on the top-level entities, but a nested entity model does substantially better when evaluated on all entities.

5.4 Joint parsing and named entity recognition

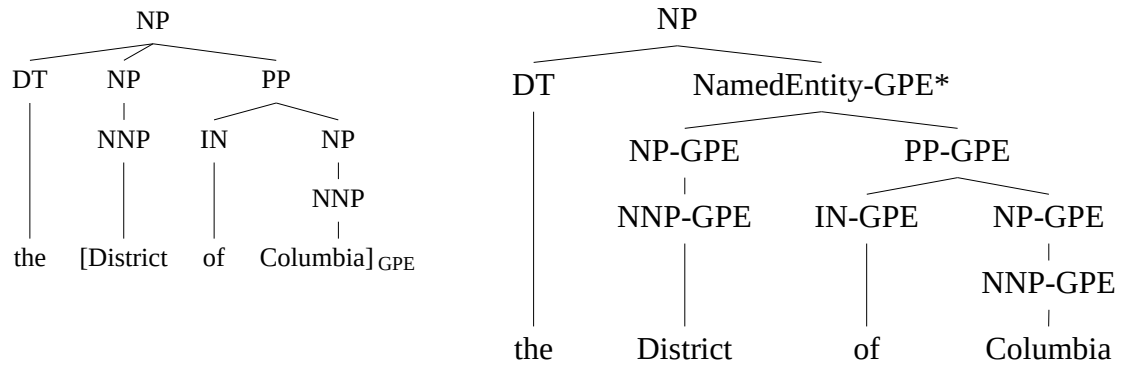
In this section, I will present a joint model of syntax and named entities. It will be based on the discriminative constituency parser earlier in this chapter, and will use the same named-entity-as-parsing representation from the previous section. As the field moves progressively towards higher-level tasks, like machine translation and question answering, joint modeling will become increasingly important, due to the need for complete, consistent, and coherent analysis of texts.

When constructing a joint model of parsing and named entity recognition, it makes sense to think about how the two distinct levels of annotation may help one another. Ideally, a named entity should correspond to a phrase in the constituency tree. However, parse trees will occasionally lack some explicit structure, such as with right branching noun phrases. In these cases, a named entity may correspond to a contiguous set of children within a subtree of the entire parse. The one thing that should never happen is for a named entity span to have crossing brackets with any spans in the parse tree.

For named entities, the joint model should help with boundaries. The internal structure of the named entity, and the structural context in which it appears, can also help with determining the type of entity. Finding the best parse for a sentence can be helped by the named entity information in similar ways. Because named entities *should* correspond to phrases, information about them should lead to better bracketing. Also, knowing that a phrase is a named entity, and the type of entity, may help in getting the structural context, and internal structure, of that entity correct.

5.4.1 Joint representation

After modifying the OntoNotes dataset to ensure consistency, which was discussed in section 2.2.4 and appendix A, I augment the parse tree with named entity information, for



(a) before **(b)** after

Figure 5.5: An example of a (sub)tree which is modified for input to our learning algorithm. Starting from the normalized tree (a) (discussed in section 2.2.4), a new NAMEDENTITY node (b) is added, so that the named entity corresponds to a single phrasal node. That node, and its descendents, have their labels augmented with the type of named entity. The * on the NAMEDENTITY node indicates that it is the root of the named entity.

input to the learning algorithm. In the cases where a named entity corresponds to multiple contiguous children of a subtree, I add a new NAMEDENTITY node, which is the new parent to those children. Now, all named entities correspond to a single phrasal node in the entire tree. The labels of the phrasal node and its descendants are then augmented with the type of named entity. I also distinguish between the root node of an entity, and the descendant nodes. See figure 5.5 for an illustration. This representation has several benefits, outlined below.

Nested entities

The OntoNotes data does not contain any nested entities. Consider the named entity portions of the rules seen in the training data. These will look, for instance, like NONE → NONE PERSON, and ORGANIZATION → ORGANIZATION ORGANIZATION. Because we only allow named entity derivations which we have seen in the data, and the data does not contain any nested entities, nested entities are impossible. However, as discussed in section 5.3, there is clear benefit in a representation allowing nested entities. For example, it

would be beneficial to recognize that the *United States Supreme Court* is a an ORGANIZATION, but that it also contains a nested GEO-POLITICAL ENTITY (GPE). Fortunately, if we encounter data which has been annotated with nested entities, this representation will be able to handle them in a natural way. In the given example, the derivation would include ORGANIZATION → GPE ORGANIZATION. This information will be helpful for correctly labeling previously unseen nested entities such as *New Jersey Supreme Court*, because the model will learn how nested entities tend to decompose. So, if the data were further augmented to include nested entity information, this representation would already work with it and be able to take advantage of this additional information.

Feature representation for named entities

As discussed in chapter 3, named entity recognizers are usually constructed using sequence models, with linear-chain CRFs being the most common. While it is possible for CRFs to have links that are longer distance than just between adjacent words, most of the benefit is from local features, over the words and labels themselves, and from features over adjacent pairs of words and labels. My joint representation allows us to port both types of features from such a named entity recognizer. In the nested NER work in the previous section, it is fairly straightforward to see how these features can be represented, since the named entity trees only contain nodes for named entities, and there are not syntax-based nodes to potentially complicate things. Fortunately, with one minor exception, the joint model can still represent all linear-chain CRF-style features. The local features can simply be computed at the same time as the features over parts of speech are computed. These are the leaves of the tree, when only the named entity for the current word is known.[10] The pairwise features, over adjacent labels, are computed at the same time as features over binary rules. Binarization of the tree is necessary for efficient computation, so the trees consist solely of unary and binary productions. Because of this, for all pairs of adjacent words within an entity, there will be a binary rule applied where one word will be under the left child and the other word will be under the right child, and so we will know if those two words have the same entity type, or if they straddle the boundary of an entity.

[10]Note that features can include information about other *words*, because the entire sentence is observed. The features *cannot* include information about the labels of those words.

Therefore, we compute features over adjacent words/labels when computing the features for the binary rule which joins them. The one exception is adjacent non-entity words. When evaluating a binary rule application, where neither child is an entity, this does not necessarily mean that the rightmost descendant of the left child (or the leftmost descendant of the right child) is not part of an entity. Thankfully, most of the beneficial pairwise features are over entity boundaries, and, to a lesser extent, adjacent words within an entity. From a practical perspective, losing boundary features over adjacent non-entities is not a problem.

5.4.2 Grammar smoothing

When building my discriminative constituency parser, the grammar (recall that this is the set of allowed rules) was determined by reading off the rules used in the training data, as is common practice. However, because of the addition of named entity annotations to grammar rules, if we use the grammar as read off the treebank we will encounter problems with sparseness which severely degrade performance. This degradation occurs because of CFG rules which only occur in the training data augmented with named entity information, and because of rules which only occur without the named entity information. To combat this problem, I added extra rules, unseen in the training data.

Augmenting the grammar

For every rule encountered in the training data which has been augmented with named entity information, extra copies of that rule are added to the grammar. I add one copy with all of the named entity information stripped away, and another copy for each other entity type, where the named entity augmentation has been changed to the other entity type.

These additions help, but they are not sufficient. Most entities correspond to noun phrases, so I took all rules which had an NP as a child, and made copies of that rule where the NP was augmented with each possible entity type. These grammar additions sufficed to improve overall performance.

Augmenting the lexicon

The lexicon is augmented in a similar manner to the rules. For every part-of-speech tag seen with a named entity annotation, I also add that tag with no named entity information, and a version which has been augmented with each type of named entity.

It would be computationally infeasible to allow any word to have any part-of-speech tag. This is because the grammar and lexicon together determine the set of possible parse trees for a given sentence. Allowing a word to have more possible parts of speech, means that more grammar rules can cover that word, which then means that more parse trees are possible over that sentence. Pruning the lexicon and grammar can have significant effects of parsing time. I therefore limit the allowed part-of-speech tags for common words based on the tags they have been observed with in the training data. I also augment each word with a distributional similarity tag, which I discuss in greater depth in section 2.4.1, and allow tags seen with other words which belong to the same distributional similarity cluster. When deciding what tags are allowed for each word, I initially ignore named entity information. Once allowed base tags are determined for a word, we also allow that tag, augmented with any type of named entity, if the augmented tag is present in the lexicon.

5.4.3 Features

Features are defined over both the parse rules and the named entities. Most of the features are over one or the other aspects of the structure, but not both.

Both the named entity and parsing features utilize the words of the sentence, as well as an orthographic *word shape* and a distributional similarity cluster, which were described in section 2.4.

For the named entity features, I used a fairly standard feature set, the same as those used in section 5.3 (see table 5.6). For parse features, I used the exact same features as those in section 5.2 (see table 5.1). When computing those features, all of the named entity information was removed from the rules, so that these features were just over the parse information and not at all over the named entity information.

Lastly, we have the joint features. I included as features each augmented rule and each augmented label. This allowed the model to learn that certain types of phrasal nodes, such

		Parse Labeled Bracketing			Named Entities			Training
		Precision	Recall	F_1	Precision	Recall	F_1	Time
ABC	Just Parse	70.2%	70.1%	**70.2%**		–		25m
	Just NER		–		76.8%	72.3%	74.5%	
	Joint Model	69.8%	70.2%	70.0%	77.7%	72.3%	**74.9%**	45m
CNN	Just Parse	76.9%	77.1%	77.0%		–		16.5h
	Just NER		–		75.6%	76.0%	75.8%	
	Joint Model	77.4%	78.0%	**77.7%**	78.7%	78.7%	**78.7%**	31.7h
MNB	Just Parse	64.0%	67.1%	65.5%		–		12m
	Just NER		–		72.3%	54.6%	62.2%	
	Joint Model	63.8$	67.5%	**65.6%**	71.4%	62.2%	**66.5%**	19m
NBC	Just Parse	59.7%	63.7%	61.6%		–		10m
	Just NER		–		67.5%	60.7%	63.9%	
	Joint Model	60.7%	65.4%	**62.9%**	71.4%	64.8%	**68.0%**	17m
PRI	Just Parse	76.2%	76.5%	76.4%		–		2.4h
	Just NER		–		82.1%	84.9%	83.4%	
	Joint Model	76.9%	78.0%	**77.4%**	86.1%	86.6%	**86.4%**	4.2h
VOA	Just Parse	76.6%	75.7%	76.2%		–		2.3h
	Just NER		–		82.8%	76.0%	79.2%	
	Joint Model	77.6%	77.5%	**77.5%**	88.4%	88.0%	**88.2%**	4.4h

Table 5.14: Full parse and NER results for the six datasets. Parse trees were evaluated using evalB, and named entities were scored using macro-averaged F-measure (conlleval).

as NPs are more likely to be named entities, and that certain entities were more likely to occur in certain contexts and have particular types of internal structure.

5.4.4 Experiments

I ran the joint model on the six OntoNotes datasets described in section 2.2.4, using sentences of length 40 and under (approximately 200,000 annotated English words, considerably smaller than the PennTreebank).

For comparison, I also trained the parser without the named entity information (and omitted the NAMEDENTITY nodes), and a linear chain CRF using just the named entity information. Both the baseline parser and CRF were trained using the exact same features as the joint model, and all were optimized using stochastic gradient descent (see section 2.3.2). The full results can be found in table 5.14. Parse trees were scored using *evalB* (the extra NAMEDENTITY nodes were ignored when computing evalB for the joint model), and

named entities were scored using entity F-measure (see section 2.1.1).[11]

While the main benefit of the joint model is the ability to get a consistent output over both types of annotations, I also found that modeling the parse and named entities jointly resulted in improved performance on both. When looking at these numbers, it is important to keep in mind that the sizes of the training and test sets are significantly smaller than the PennTreebank. The largest of the six datasets, CNN, has about one seventh the amount of training data as the PennTreebank, and the smallest, MNB, has around 500 sentences from which to train. Parse performance was improved by the joint model for five of the six datasets, by up to 1.6% F_1. Looking at the parsing improvements on a per-label basis, the largest gains came from improved identification of NML (*nominal*, used to represent left-branching noun phrases) constituents, from an F-score of 45.9% to 57.0% (on all the data combined, for a total of 420 NML constituents). This label was added in the new treebank annotation conventions, so as to identify internal left-branching structure inside previously flat NPs (*noun phrases*). To my surprise, performance on NPs only increased by 1%, though over $12,949$ constituents, for the largest improvement in absolute terms. The second largest gain was on PPs (*prepositional phrases*), which improved by 1.7% over $3,775$ constituents. I tested the significance of our results (on all the data combined) using Dan Bikel's randomized parsing evaluation comparator[12] and found that both the precision and recall gains were significant at $p \leq 0.01$.

Much greater improvements in performance were seen on named entity recognition, where most of the domains saw improvements in the range of $3-4\%$, with performance on the *VOA* data improving by nearly 9%, which is a 43% reduction in error. There was no clear trend in terms of precision versus recall, or the different entity types. The first place to look for improvements is with the boundaries for named entities. Once again looking at all of the data combined, in the baseline model there were 203 entities where part of the

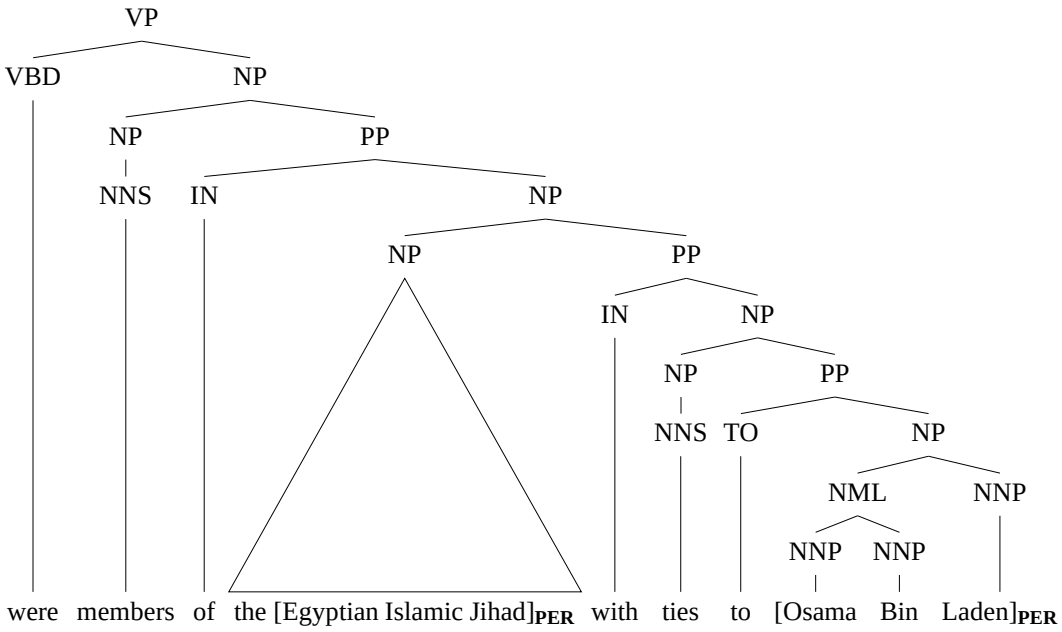
[11] Sometimes the parser would be unable to parse a sentence (less than 2% of sentences), due to restrictions in part of speech tags. Because the underlying grammar (ignoring the additional named entity information) was the same for both the joint and baseline parsers, it is the case that whenever a sentence is unparseable by either the baseline or joint parser, it is in fact unparseable by both of them, and would affect the parse scores of both models equally. However, the CRF is able to named entity tag any sentence, so these unparseable sentences had an effect on the named entity score. To combat this, I fell back on the baseline CRF model to get named entity tags for unparseable sentences.

[12] Available at http://www.cis.upenn.edu/ dbikel/software.html

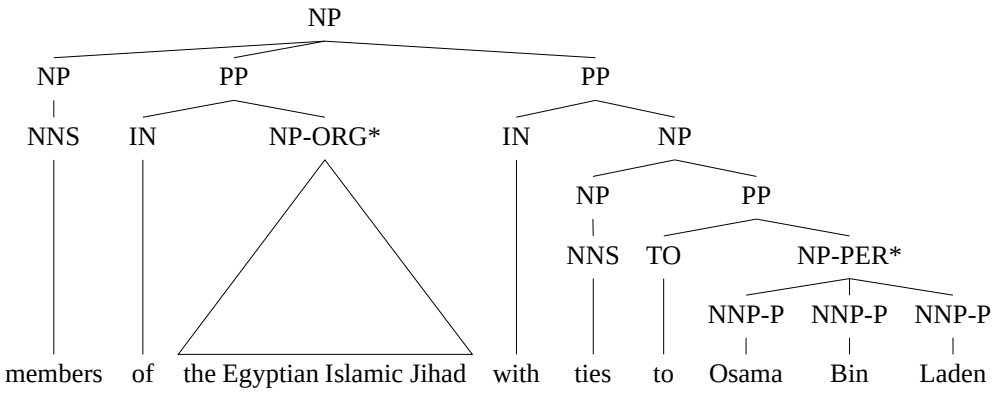
entity was found, but one or both boundaries were incorrectly identified. The joint model corrected 72 of those entities, while incorrectly identifying the boundaries of 37 entities which had previously been correctly identified. In the baseline NER model, there were 243 entities for which the boundaries were correctly identified, but the type of entity was incorrect. The joint model corrected 80 of them, while changing the labels of 39 entities which had previously been correctly identified. Additionally, 190 entities were found that the baseline model had missed entirely, and 68 entities were lost. I tested the statistical significance of the gains (of all the data combined) using the same sentence-level, stratified shuffling technique as Bikel's parse comparator and found that both precision and recall gains were significant at $p < 10^{-4}$.

An example from the data where the joint model helped improve both parse structure and named entity recognition is shown in figure 5.6. The output from the individual models is shown in part (a), with the output from the named entity recognizer shown in brackets on the words at leaves of the parse. The output from the joint model is shown in part (b), with the named entity information encoded within the parse. In this example, the named entity *Egyptian Islamic Jihad* helped the parser to get its surrounding context correct, because it is improbable to attach a PP headed by *with* to an *organization*. At the same time, the surrounding context helped the joint model correctly identify *Egyptian Islamic Jihad* as an *organization* and not a *person*. The baseline parser also incorrectly added an extra level of structure to the person name *Osama Bin Laden*, while the joint model found the correct structure.

One significant limitation of this method is the training time. The training times for the individual corpora are also given in table 5.14, and you can see that training the joint model took approximately twice as long as training the parse-only model, which was already quite expensive. While I would have liked to evaluate the full joint model on all of OntoNotes combined, it would have taken several months and so was not really feasible. Hopefully as computers get faster, this drawback will be mediated.



(a) Output from the single-task model



(b) Output from the joint parse and named entity model

Figure 5.6: An example for which the joint model helped with both parse structure and named entity recognition. The individual models (a) incorrectly attach the PP, label *Egyptian Islamic Jihad* as a *person*, and incorrectly add extra internal structure to *Osama Bin Laden*. The joint model (b) gets both the structure and the named entity correct.

5.4.5 Related work on joint modeling

A pioneering antecedent for this work is Miller et al. (2000), who trained a Collins-style generative parser (Collins 1997) over a syntactic structure augmented with the *template entity* and *template relations* annotations for the MUC-7 shared task. Their sentence augmentations were similar to ours, but they did not make use of features due to the generative nature of their model. This approach was not followed up on in other work, presumably because around this time nearly all the activity in named entity and relation extraction moved to the use of discriminative sequence models, which allowed the flexible specification of feature templates that are very useful for these tasks. The present model is able to bring together both these lines of work, by integrating the strengths of both approaches. The task was also a bit different, as template entity extraction consists of filling in a template, as opposed to named entity recognition, which requires marking the entities within the text.

There have been other attempts in NLP to jointly model multiple levels of structure, with varying degrees of success. Most work on joint parsing and semantic role labeling (SRL) has been disappointing, despite obvious connections between the two tasks. Sutton and McCallum (2005) attempted to jointly model PCFG parsing and SRL for the CoNLL 2005 shared task, but were unable to improve performance on either task. The CoNLL 2008 shared task (Surdeanu et al. 2008) was joint dependency parsing and SRL, but the top performing systems decoupled the tasks, rather than building joint models. Zhang and Clark (2008) successfully built a joint model of Chinese word segmentation and parts of speech using a single perceptron.

5.5 Summary

In this chapter, I presented two discriminative parsers, and then used them to build models of nested named entities and a joint model of parsing and named entity recognition. The discriminative constituency parser is a feature-rich CRF-based parser, and was the first such parser to scale up to non-toy sentences. I showed substantial improvements over a generative baseline, and found that about a third of the gains were a result of switching from generative to discriminative training, and two thirds were from the inclusion of features.

This discriminative constituency parser was then used to build a model of nested named entities. It is very natural to model nested entities using a parse tree, since once a root node is added, the nested structure is in fact a parse tree. The parser makes it easy to utilize all of the linear-chain CRF features which have been developed for NER over the past decade. I performed experiments on biomedical data, and on Spanish and Catalan newspaper text. The experiments showed that the nested model did not (on average) increase or decrease performance on the top-level entities (the standard NER task), while still doing quite well on the nested entities, which most prior work has ignored. One difficulty here is the lack of available data for nested NER; it would be nice to see some English newswire or web data which has been annotated to include nested entities and not just the top level ones.

Finally, I presented a full joint model of parsing and named entity recognition. This model also used the parse as its backbone, and represented named entities as nodes in the tree. I showed that by modeling these different types of information together, we can get modest parsing gains and large NER improvements.

Chapter 6

Hierarchical joint learning

6.1 Introduction

The previous chapter culminated in a joint model of parsing and named entity recognition, and the model presented required data which had been jointly annotated with both kinds of structure. However, it is common to have multiple, related tasks, each of which have their own separate annotated corpora. This setup is called *multi-task learning*, and in this chapter I show how to use a hierarchical prior to do multi-task learning for two very different use cases.

In the experimental results on joint modelling in the previous chapter, I showed that the resulting models produce more consistent outputs, and that performance improves across all aspects of the joint structure. However, one significant limitation for many joint models is the lack of jointly annotated data. The previously described joint model of parsing and named entity recognition had small gains on parse performance and moderate gains on named entity performance, when compared with single-task models trained on the same data. However, the performance of this model, trained using the OntoNotes corpus, fell short of separate parsing and named entity models trained on larger corpora, annotated with only one type of information. By using a hierarchical prior to link the feature weights for related tasks, we can learn high-quality joint models with smaller quantities of jointly-annotated data that has been augmented with larger amounts of single-task annotated data. To my knowledge this work is the first attempt at such a task. The hierarchical prior links

a joint model trained on jointly-annotated data with other single-task models trained on single-task annotated data. The key to making this work is for the joint model to share some features with each of the single-task models. Then, the singly-annotated data can be used to influence the feature weights for the shared features in the joint model. This is an important contribution, because it provides all the benefits of joint modeling, but without the high cost of jointly annotating large corpora. I applied the hierarchical joint model to parsing and named entity recognition, and it reduced errors by over 20% on both tasks when compared to a joint model trained on only the jointly annotated data.

The second use case for hierarchical multi-task learning is multi-domain learning, which is very similar to domain adaptation, an important NLP task. The only difference is that in multi-domain learning the focus is on improving performance across *all* domains, while in domain adaptation there is a distinction between *source* data and *target* data, and the goal is to improve performance on the target data. The word *domain* is used here somewhat loosely: it may refer to a topical domain or to distinctions that linguists might term mode (speech versus writing) or register (formal written prose versus SMS communications). For example, one may have a large amount of parsed newswire, and want to use it to augment a much smaller amount of parsed e-mail, to build a higher quality parser for e-mail data. I also consider the extension to the task where the annotation is not the same, but is consistent across domains (that is, some domains may be annotated with more information than others). With named entity recognition, it is not uncommon to successfully identify an entity, but mislabel the type of entity. Using data from another domain that has been labeled with additional entity types not present in the original data may help prevent misidentifying entities in the original data which are of a type only labeled in the other datasets.

This problem is important because it is omnipresent in real life natural language processing tasks. Annotated data is expensive to produce and limited in quantity. Typically, one may begin with a considerable amount of annotated newswire data, some annotated speech data, and a little annotated e-mail data. It would be most desirable if the aggregated training data could be used to improve the performance of a system on each of these domains. I apply this model to two previously discussed tasks, named entity recognition and dependency parsing, and in both cases found significant improvements when compared to

strong baselines.

In this chapter I will first cover the fundamentals of multi-task learning with a hierarchical prior. I will then ground the technique with respect to each individual task, and provide experimental results for both.

6.2 Related work

My domain adaptation model is a generalization of the model presented in (Daumé III 2007), which is discussed in more detail in section 6.3.4. Another similar piece of domain adaptation work is Chelba and Acero (2004), who also modify their prior. Their work is limited to two domains, a source and a target, and their algorithm has a two stage process: First, train a classifier on the source data, and then use the learned weights from that classifier as the mean for a Gaussian prior when training a new model on just the target data.

Daumé III and Marcu (2006) also took a Bayesian approach to domain adaptation, but structured their model in a very different way. In their model, it is assumed that each datum within a domain is either a domain-specific datum, or a general datum, and then domain-specific and general weights were learned. Whether each datum is domain-specific or general is not known, so they developed an EM-based algorithm for determining this information, while simultaneously learning the feature weights. Their model had good performance, but came with a 10 to 15 times slowdown at training time. My slowest dependency parser took four days to train, making this model close to infeasible for learning on that data.

Both tasks addressed in this chapter can be viewed as instances of *multi-task learning*, a machine learning paradigm in which the objective is to simultaneously solve multiple, related tasks for which you have separate labeled training data. There has not been much work on multi-task learning in the NLP community; in addition to the domain adaptation work of Daumé III (2007), described above, Ando and Zhang (2005) utilized a multi-task learner within their semi-supervised algorithm to learn feature representations which were useful across a large number of related tasks. Outside of the NLP community, Elidan et al. (2008) used an undirected Bayesian transfer hierarchy to several tasks, including jointly

modeling the shapes of multiple mammal species and a text classification task. Like the work presented here, they also used pointwise estimation on the parameters. Evgeniou et al. (2005) applied a hierarchical prior to modeling exam scores of students. Other instances of multi-task learning include (Baxter 1997, Caruana 1997, Yu et al. 2005, Xue et al. 2007). For a more general discussion of hierarchical models, we direct the reader to Chapter 5 of Gelman et al. (2003) and Chapter 12 of Gelman and Hill (2006).

6.3 Hierarchical priors for multi-task learning

In this section I will cover the general technique behind both of my hierarchical joint learning models, as the underlying technology is identical in both cases. The only requirement for using this technique is that you have multiple base models which share some subset of features with one another. In the first case, those base models are a joint parsing and NER model, and then single-task models for both parsing and NER. In the second case, those base models are identical named entity (or dependency parsing) models where each base model is trained on data from a different domain. I will discuss the details of both of these cases in subsequent sections. At times when it helps to have an example I will concretize the general model using the case of joint parsing and NER.

6.3.1 Intuitive overview

As discussed, we have multiple related base models (and their corresponding training data sets). The key to the hierarchical model is that each of the base models have some features in common with some of the other base models, though they can also have some features which are only present in one of the base models. Each model has its own set of parameters (feature weights). However, parameters for the features which are shared between the different base models are able to influence one another via a hierarchical prior. This prior encourages the learned weights for the different models to be similar to one another. After training has been completed, we can retain only the parameters about which we care – in the joint parsing and and NER case this is the joint model's parameters, whereas in the multi-domain learning case this corresponds to the parameters for all of the domains.

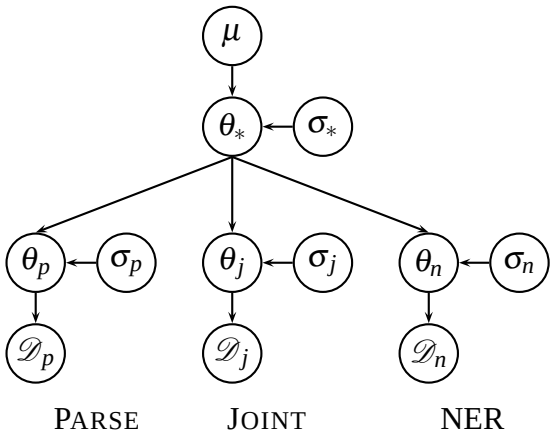


Figure 6.1: A graphical representation of my hierarchical joint model. There are separate base models for just parsing, just NER, and joint parsing and NER. The parameters for these models are linked via a hierarchical prior.

6.3.2 Formal model

We have a set $\mathscr{M}$ of base models which have corresponding log-likelihood functions $\mathscr{L}_m(\mathscr{D}_m; \theta_m)$ for each $m \in \mathscr{M}$, where $\mathscr{D}_m$ is the model-specific training data and θ_m is the model-specific parameter (feature weight) vectors for model m. These likelihood functions do *not* include priors over the θs. For representational simplicity, we assume that each of these vectors is the same size and corresponds to the same ordering of features. Features which don't apply to a particular model type (e.g., parse features in the named entity model) will always be zero, so their weights have no impact on that model's likelihood function. Conversely, allowing the presence of those features in models for which they do not apply will not influence their weights in the other models because there will be no evidence about them in the data. These base models are linked by a hierarchical prior, and their feature weight vectors are all drawn from this prior. The parameters θ_* for this prior have the same dimensionality as the model-specific parameters θ_m and are drawn from another, top-level prior. In our case, this top-level prior is a zero-mean Gaussian.[1]

The graphical representation of a hierarchical model with three base models is shown

[1]Though I used a zero-mean Gaussian prior, this top-level prior could take many forms, including an L_1 prior, or another hierarchical prior.

in figure 6.1. The log-likelihood of this model is

$$\mathscr{L}_{\textit{hier-joint}}(\mathscr{D};\theta) = \sum_{m\in\mathscr{M}} \left(\mathscr{L}_m(\mathscr{D}_m;\theta_m) - \sum_i \frac{(\theta_{m,i}-\theta_{*,i})^2}{2\sigma_m^2} \right) - \sum_i \frac{(\theta_{*,i}-\mu_i)^2}{2\sigma_*^2} \tag{6.1}$$

The first summation in this equation computes the log-likelihood of each model, using the data and parameters which correspond to that model, and the prior likelihood of that model's parameters, based on a Gaussian prior centered around the top-level, non-model-specific parameters θ_*, and with model-specific variance σ_m. The final summation in the equation computes the prior likelihood of the top-level parameters θ_* according to a Gaussian prior with variance σ_* and mean μ (typically zero). This formulation encourages each base model to have feature weights similar to the top-level parameters (and hence one another). I use pointwise estimation (Elidan et al. 2008) when learning the feature weights. By this I mean that I directly estimate the top-level parameters, θ_*, instead of being Bayesian and integrating them out.

The effects of the variances σ_m and σ_* warrant some discussion. σ_* has the familiar interpretation of dictating how much the model "cares" about feature weights diverging from zero (or μ). The model-specific variances, σ_m, have an entirely different interpretation. They dictate how strong the penalty is for the domain-specific parameters to diverge from one another (via their similarity to θ_*). When σ_m are very low, then the feature weights are encouraged to be very similar, and taken to the extreme this is equivalent to completely tying the parameters between the tasks. When σ_m are very high, then there is less encouragement for the parameters to be similar, and taken to the extreme this is equivalent to completely decoupling the tasks.

We need to compute partial derivatives in order to optimize the model parameters. The partial derivatives for the parameters for each base model m are given by:

$$\frac{\partial \mathscr{L}_{\textit{hier}}(\mathscr{D};\theta)}{\partial \theta_{m,i}} = \frac{\partial \mathscr{L}_m(\mathscr{D}_m,\theta_m)}{\partial \theta_{m,i}} - \frac{\theta_{m,i}-\theta_{*,i}}{\sigma_d^2} \tag{6.2}$$

where the first term is the partial derivative according to the base model, and the second term is the prior centered around the top-level parameters. The partial derivatives for the

top level parameters θ_* are:

$$\frac{\partial \mathscr{L}_{hier}(\mathscr{D};\theta)}{\partial \theta_{*,i}} = \left(\sum_{m \in \mathscr{M}} \frac{\theta_{*,i} - \theta_{m,i}}{\sigma_m^2} \right) - \frac{\theta_{*,i} - \mu_i}{\sigma_*^2} \tag{6.3}$$

where the first term relates to how far each model-specific weight vector is from the top-level parameter values, and the second term relates how far each top-level parameter is from zero.

When a model has strong evidence for a feature, effectively what happens is that it pulls the value of the top-level parameter for that feature closer to the model-specific value for it. When it has little or no evidence for a feature then it will be pulled in the direction of the top-level parameter for that feature, whose value was influenced by the models which have evidence for that feature.

6.3.3 Optimization with stochastic gradient descent

As we saw in the previous chapter, inference in joint models tends to be slow, and often requires the use of stochastic optimization in order for the optimization to be tractable. I used stochastic gradient descent (SGD) (see section 2.3.2) for the experiments in this section, and here I discuss how to use SGD when we have a hierarchical prior, since it is less straightforward than the more common case where we have an objective function which only requires summing over data.

L-BFGS and gradient descent, two frequently used (non-stochastic) numerical optimization algorithms, require computing the value and partial derivatives of the objective function using the entire training set. Instead, we use stochastic gradient descent. It requires a stochastic objective function, which is meant to be a low computational cost estimate of the real objective function. In most NLP models, such as logistic regression with a Gaussian prior, computing the stochastic objective function is fairly straightforward: you compute the model likelihood and partial derivatives for a randomly sampled subset of the training data. When computing the term for the prior, it must be rescaled by multiplying its value and derivatives by the proportion of the training data used. The stochastic objective

function, where $\widehat{\mathscr{D}} \subseteq \mathscr{D}$ is a randomly drawn subset of the full training set, is given by

$$\mathscr{L}_{stoch}(\mathscr{D};\theta) = \mathscr{L}_{orig}(\widehat{\mathscr{D}};\theta) - \frac{|\widehat{\mathscr{D}}|}{|\mathscr{D}|}\sum_i \frac{(\theta_{*,i})^2}{2\sigma_*^2} \tag{6.4}$$

This is a *stochastic* function, and multiple calls to it with the same $\mathscr{D}$ and θ will produce different values because $\widehat{\mathscr{D}}$ is re-sampled each time. When designing a stochastic objective function, the critical fact to keep in mind is that the summed values and partial derivatives for any partitioning of the data need to be equal to that of the full dataset. In practice, stochastic gradient descent only makes use of the partial derivatives and not the function value, so we will focus the remainder of the discussion on how to rescale the partial derivatives.

I will now describe the more complicated case of stochastic optimization with a hierarchical objective function. For the sake of simplicity, let us assume that we are using a batch size of one, meaning $|\widehat{\mathscr{D}}| = 1$ in the above equation. Note that in the hierarchical model, each datum (sentence) in each base model should be weighted equally, so whichever dataset is the largest should be proportionally more likely to have one of its data sampled. For the sampled datum d, we then compute the function value and partial derivatives with respect to the correct base model for that datum. When we rescale the model-specific prior, we rescale based on the number of data in that model's training set, *not* the total number of data in all the models combined. Having uniformly randomly drawn datum $d \in \bigcup_{m\in\mathscr{M}} \mathscr{D}_m$, let $m(d) \in \mathscr{M}$ indicate to which model's training data the datum belongs. The stochastic partial derivatives will equal zero for all model parameters θ_m such that $m \neq m(d)$, and for $\theta_{m(d)}$ it becomes:

$$\frac{\partial \mathscr{L}_{hier\text{-}stoch}(\mathscr{D};\theta)}{\partial \theta_{m(d),i}} = \frac{\partial \mathscr{L}_{m(d)}(\{d\};\theta_{m(d)})}{\partial \theta_{m(d),i}} - \frac{1}{|\mathscr{D}_{m(d)}|}\left(\frac{\theta_{m(d),i} - \theta_{*,i}}{\sigma_d^2}\right) \tag{6.5}$$

Now I will discuss the stochastic partial derivatives with respect to the top-level parameters θ_*, which requires modifying equation 6.3. The first term in that equation is a summation over all the models. In the stochastic derivative we only perform this computation for the datum's model $m(d)$, and then we rescale that value based on the number of data in that datum's model $|\mathscr{D}_{m(d)}|$. The second term in that equation is rescaled by the *total* number of

data in all models combined. The stochastic partial derivatives with respect to θ_* become:

$$\frac{\partial \mathscr{L}_{\textit{hier-stoch}}(\mathscr{D};\theta)}{\partial \theta_{*,i}} = \frac{1}{|\mathscr{D}_{m(d)}|}\left(\frac{\theta_{*,i}-\theta_{m(d),i}}{\sigma_m^2}\right) - \frac{1}{\sum_{m\in\mathscr{M}}|\mathscr{D}_m|}\left(\frac{\theta_{*,i}}{\sigma_*^2}\right) \tag{6.6}$$

where for conciseness we omit μ under the assumption that it equals zero.

An equally correct formulation for the partial derivative of θ_* is to simply rescale equation 6.3 by the *total* number of data in all models. Early experiments found that both versions gave similar performance, but the latter was significantly slower to compute because it required summing over the parameter vectors for all base models instead of just the vector for the datum's model.

When using a batch size larger than one, you compute the given functions for each datum in the batch and then add them together.

6.3.4 Formalization of prior feature-augmentation work

This technique for utilizing a hierarchical prior to link related tasks is equivalent to the "frustratingly easy" domain adaptation method presented in Daumé III (2007), and can be viewed as a formal version of his model.[2] In his presentation, the adaptation is done through feature augmentation. Specifically, for each feature in the original version, a new version is created for each domain, as well as a general, domain-independent version of the feature. For each datum, two versions of each original feature are present: the version for that datum's domain, and the domain independent one.

The equivalence between the two models can be shown with simple arithmetic. Recall that the log likelihood of our model is:

$$\sum_d \left(\mathscr{L}_{orig}(\mathscr{D}_d;\theta_d) - \sum_i \frac{(\theta_{d,i}-\theta_{*,i})^2}{2\sigma_d^2}\right) - \sum_i \frac{(\theta_{*,i})^2}{2\sigma_*^2}$$

[2]Many thanks to David Vickrey for pointing this out.

We now introduce a new variable $\psi_d = \theta_d - \theta_*$, and plug it into the equation for log likelihood:

$$\sum_d \left(\mathscr{L}_{orig}(\mathscr{D}_d; \psi_d + \theta_*) - \sum_i \frac{(\psi_{d,i})^2}{2\sigma_d^2} \right) - \sum_i \frac{(\theta_{*,i})^2}{2\sigma_*^2}$$

The result is the model of Daumé III (2007), where the ψ_d are the domain-specific feature weights, and θ_* are the domain-independent feature weights. In his formulation, the variances $\sigma_d^2 = \sigma_*^2$ for all domains d.

However, this formalization highlights the opportunity to set the different variances separately. This separation of the domain-specific and independent variances was critical to our improved performance. When using a Gaussian prior there are two parameters set by the user: the mean, μ (usually zero), and the variance, σ^2. Technically, each of these parameters is actually a vector, with an entry for each feature, but almost always the vectors are uniform and the same parameter is used for each feature (there are exceptions, e.g. Lee et al. (2007)). Because Daumé III (2007) views the adaptation as merely augmenting the feature space, each of his features has the same prior mean and variance, regardless of whether it is domain specific or independent. He could have set these parameters differently, but he did not.[3] In our presentation of the model, we explicitly represent different variances for each domain, as well as the top level parameters. We found that specifying different values for the domain specific versus domain independent variances significantly improved performance, though we found no gains from using different values for the different domain specific variances. The values were set based on development data.

6.4 Improving joint parsing and named entity recognition

I just described the general framework for hierarchical learning, and now I will discuss how to use it to improve a joint model using additional data annotated for only one task. Specifically, we have a joint model of parsing and named entity recognition, and two single-task models, one for parsing and one for named entity recognition. I will discuss each of these base models, though to varying degrees all have been discussed previously in this

[3] Although he alludes to the potential for something similar in the last section of his paper, when discussing the kernelization interpretation of his approach.

dissertation. I will also give experimental results, comparing the joint model trained inside of the hierarchical model with the joint model described in section 5.4.

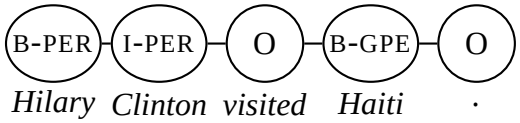
6.4.1 Base models

Named entity recognition For the named entity recognition-only model I used a semi-CRF (Sarawagi and Cohen 2004, Andrew 2006). Semi-CRFs (also mentioned briefly in section 5.3.2) are very similar to the more popular linear-chain CRFs, but they have several key advantages. Semi-CRFs *segment and label* the text simultaneously, whereas a linear-chain CRF will *only label* each word, and segmentation is implied by the labels assigned to the words. When doing named entity recognition, a semi-CRF will have one node for each entity, unlike a regular CRF which will have one node for each word.[4] Please refer to figure 6.2a-b for an example of a semi-CRF and a linear-chain CRF over the same sentence. Note that the entity *Hilary Clinton* has one node in the semi-CRF representation, but two nodes in the linear-chain CRF. Because different segmentations have different model structures in a semi-CRF, one has to consider all possible structures (segmentations) as well as all possible labelings. It is common practice to limit segment length in order to speed up inference, as this allows for the use of a modified version of the forward-backward algorithm. When segment length is not restricted, the inference procedure is the same as that used in parsing (we used this fact in section 5.3).[5] In this work I did not enforce a length restriction, and directly utilize the fact that the model can be transformed into a parsing model. Figure 6.2c shows a parse tree representation of a semi-CRF, and figure 6.2d shows the same tree after binarization. @ROOT-PER means that we are in the middle of building a ROOT (so this is an *active state*; see section 2.1.2), and the previous child was a PER.

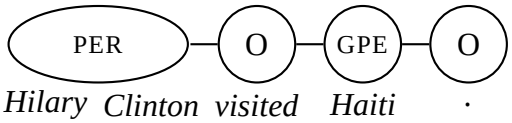
While a linear-chain CRF allows features over adjacent words, a semi-CRF allows them over adjacent segments. This means that a semi-CRF can utilize all features used by a linear-chain CRF, and can also utilize features over entire segments, such as *First National Bank of New York City*, instead of just adjacent words like *First National* and *Bank of*. Let $\mathbf{y}$ be a vector representing the labeling for an entire sentence. y_i encodes the label of the

[4] Both models will have one node per word for non-entity words.

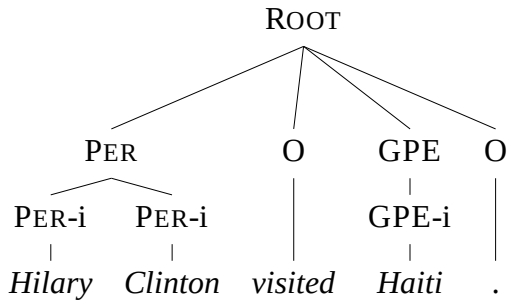
[5] While converting a semi-CRF into a parser results in much slower inference than a linear-chain CRF, it is still significantly faster than a treebank parser due to the reduced number of labels.



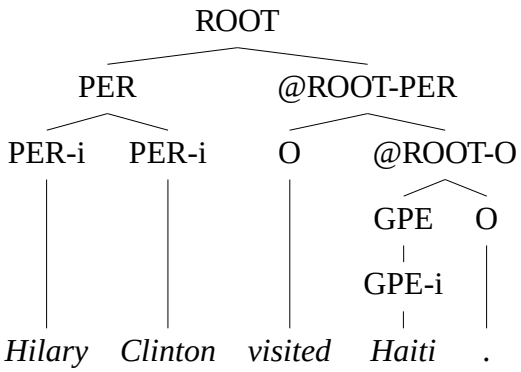
(a)



(b)



(c)



(d)

Figure 6.2: A linear-chain CRF **(a)** labels each word, whereas a semi-CRF **(b)** labels entire entities. A semi-CRF can be represented as a tree **(c)**, where **i** indicates an internal node for an entity. **(d)** shows the tree representation after binarization.

*i*th segment, along with the span of words the segment encompasses. Let θ be the feature weights, and $\mathbf{f}(s, y_i, y_{i-1})$ the feature function over adjacent segments y_i and y_{i-1} in sentence s.[6] The regularized log-likelihood of a semi-CRF for a single sentence s is given by:

$$\mathscr{L}(\mathbf{y}|s;\theta) = \sum_{i=1}^{|\mathbf{y}|} \exp\{\theta \cdot \mathbf{f}(s, y_i, y_{i-1})\} - Z_{s,\theta} - \frac{|\theta|^2}{2\sigma^2} \tag{6.7}$$

The partition function $Z_{s,\theta}$ serves as a normalizer. It requires summing over the set $\mathbf{y}_s$ of all possible segmentations and labelings for the sentence s:

$$Z_{s,\theta} = \sum_{\mathbf{y} \in \mathbf{y}_s} \sum_{i=1}^{|\mathbf{y}|} \exp\{\theta \cdot \mathbf{f}(s, y_i, y_{i-1})\} \tag{6.8}$$

Because we use a tree representation, it is easy to ensure that the features used in the NER model are identical to those in the joint parsing and named entity model, because the joint model is also based on a tree representation where each entity corresponds to a single node in the tree.

Parsing For the parsing-only model I used the discriminative CRF-based parser described previously in section 5.2.

Joint parsing and named entity recognition For the joint parsing and named entity recognition model I used the joint model described previously in section 5.4.

6.4.2 Experiments and discussion

I compared the hierarchical joint model to the regular (non-hierarchical) joint model presented in section 5.4, once again using the OntoNotes corpus (see section 2.2.4). I have also included the parse-only and NER-only baseline models from that section, for comparison. Table 6.1 has the complete set of results. For each section of the data (ABC, MNB, NBC,

[6] There can also be features over single entities, but these can be encoded in the feature function over adjacent entities, so for notational simplicity we do not include an additional term for them.

		Parse Labeled Bracketing			Named Entities		
		Precision	Recall	F_1	Precision	Recall	F_1
ABC	Just Parse	69.8%	69.9%	69.8%		–	
	Just NER		–		77.0%	75.1%	76.0%
	Baseline Joint	70.2%	70.5%	70.3%	79.2%	76.5%	77.8%
	Hierarchical Joint	75.5%	74.4%	**74.9%**	85.1%	82.7%	**83.9%**
MNB	Just Parse	61.7%	65.5%	63.6%		–	
	Just NER		–		69.6%	49.0%	57.5%
	Baseline Joint	61.7%	66.2%	63.9%	70.9%	63.5%	67.0%
	Hierarchical Joint	72.6%	70.2%	**71.4%**	74.4%	75.5%	**74.9%**
NBC	Just Parse	59.9%	63.9%	61.8%		–	
	Just NER		–		63.9%	60.9%	62.4%
	Baseline Joint	59.3%	64.2%	61.6%	68.9%	62.8%	65.7%
	Hierarchical Joint	70.4%	69.9%	**70.2%**	72.9%	74.0%	**73.4%**
PRI	Just Parse	78.6%	77.0%	76.9%		–	
	Just NER		–		81.3%	77.8%	79.5%
	Baseline Joint	78.0%	78.6%	78.3%	86.3%	86.0%	**86.2%**
	Hierarchical Joint	79.2%	78.5%	**78.8%**	84.2%	85.5%	84.8%
VOA	Just Parse	77.5%	76.5%	77.0%		–	
	Just NER		–		85.2%	80.3%	82.7%
	Baseline Joint	77.2%	77.8%	77.5%	87.5%	86.7%	87.1%
	Hierarchical Joint	79.8%	77.8%	**78.8%**	87.7%	88.9%	**88.3%**

Table 6.1: Full parse and NER results for the six datasets. Parse trees were evaluated using evalB, and named entities were scored using micro-averaged F-measure (conlleval).

PRI, VOA) I ran experiments training a linear-chain CRF on only the named entity information, a CRF-CFG parser on only the parse information, a joint parser and named entity recognizer, and our hierarchical model. For the hierarchical model, I used the CNN portion of the data (5093 sentences) for the extra named entity data (and ignored the parse trees) and the remaining portions combined for the extra parse data (and ignored the named entity annotations). I used $\sigma_* = 1.0$ and $\sigma_m = 0.1$, which were chosen based on early experiments on development data. Small changes to σ_m do not appear to have much influence, but larger changes do. I similarly decided how many iterations to run stochastic gradient descent for (20) based on early development data experiments. I did not run this experiment on the CNN portion of the data, because the CNN data was already being used as the extra NER data.

As table 6.1 shows, the hierarchical model did substantially better than the joint model

overall, which is not surprising given the extra data to which it had access. Looking at the smaller corpora (NBC and MNB) we see the largest gains, with both parse and NER performance improving by about 8% F1. ABC saw about a 6% gain on both tasks, and VOA saw a 1% gain on both. The one negative result is in the PRI portion: parsing improves slightly, but NER performance decreases by almost 2%. The same experiment on development data resulted in a performance increase, so I are not sure why we saw a decrease here. One general trend, which is not surprising, is that the hierarchical model helps the smaller datasets more than the large ones. The source of this is three-fold: lower baselines are generally easier to improve upon; the larger corpora had less singly-annotated data to provide improvements, because it was composed of the remaining, smaller, sections of OntoNotes; and transfer leaning of this sort is generally know to help less when you have more data. I found it interesting that the gains tended to be similar on both tasks for all datasets, and believe this fact is due to my use of roughly the same amount of singly-annotated data for both parsing and NER.

One possible conflating factor in these experiments is that of domain drift. While I tried to get the most similar annotated data available – data which was annotated by the same annotators, and all of which is broadcast news – these are still different domains. While this is likely to have a negative effect on results, I also believe this scenario to be a more realistic than if it were to also be data drawn from the exact same distribution.

6.5 Multi-domain learning

In this section I will cover two sets of experiments on domain adaptation using a hierarchical prior, one on named entity recognition and one on dependency parsing. In this case, unlike the previous section, the base models are all of the same underlying form and the only difference between them is the training data associated with each. The NER and parsing base models are both models we have seen previously.

	# Train Words	# Test Words
MUC-6	165,082	15,032
MUC-7	89,644	64,490
CoNLL	203,261	46,435

Table 6.2: Number of words in the training and test sets for each of the named entity recognition datasets.

6.5.1 Named entity recognition

For the NER experiments, I used a linear-chain CRF, as described in section 2.1.1, for the base models. I used three named entity datasets, from the CoNLL 2003, MUC-6 and MUC-7 shared tasks (see section 2.2.1 for a more complete description). CoNLL is British newswire, while MUC-6 and MUC-7 are both American newswire. Arguably MUC-6 and MUC-7 should not count as separate domains, but because they were annotated separately, for different shared tasks, I chose to treat them as such, and feel that the experimental results justify the distinction. I used the standard train and test sets for each domain, which for CoNLL corresponds to the (more difficult) testb set. For details about the number of training and test words in each dataset, please see table 6.2.

One interesting challenge in dealing with both CoNLL and MUC data is that the label sets differ. CoNLL has four classes: PERSON, ORGANIZATION, LOCATION, and MISC. MUC data has seven classes: PERSON, ORGANIZATION, LOCATION, PERCENT, DATE, TIME, and MONEY. They overlap in the three core classes (PERSON, ORGANIZATION, and LOCATION), but CoNLL has one additional class and MUC has four additional classes.

The differences in the label sets led me to perform two sets of experiments for the baseline and hierarchical Bayesian models. In the first set of experiments, at training time, the model allows any label from the union of the label sets, regardless of whether that label was legal for the domain. At test time, we would ignore guesses made by the model which were inconsistent with the allowed labels for that domain.[7] In the second set of experiments, I restricted the model at training time to only allow legal labels for each domain. At test time, the domain was specified, and the model was once again restricted so

[7] I treated them identically to if they had been labeled with the background symbol. So, for instance, labelling a word a *date* in the CoNLL data had no effect on the score.

that words would never be tagged with a label outside of that domain's label set.

In the experiments, I compared my model to several strong baselines, and the full set of results is in table 6.3. The models I used were:

TARGET ONLY. Trained and tested on only the data for that domain.

ALL DATA. Trained and tested on data from all domains, concatenated into one large dataset.

ALL DATA*. Same as ALL DATA, but restricted possible labels for each word based on domain.

DAUME07. Trained and tested using the same technique as Daumé III (2007). Note that they present results using per-token label accuracy, while we used the more standard entity precision, recall, and F score (as in the CoNLL 2003 shared task).

DAUME07*. Same as DAUME07, but restricted possible labels for each word based on domain.

HIER BAYES. My hierarchical domain adaptation model.

HIER BAYES*. Same as HIER BAYES, but restricted possible labels for each word based on the domain.

For all of the baseline models, and for the top level-parameters in the hierarchical Bayesian model, I used $\sigma_* = 1$. For the domain-specific parameters, I used $\sigma_d = 0.1$ for all domains.

The HIER BAYES model outperformed all baselines for both of the MUC datasets, and tied with the DAUME07 for CoNLL. The largest improvement was on MUC-6, where HIER BAYES outperformed DAUME07*, the second best model, by 1.36%. This improvement is greater than the improvement made by that model over the ALL DATA* baseline. To assess significance I used a document-level paired t-test (over all of the data combined), and found that HIER BAYES significantly outperformed all of the baselines (not including HIER BAYES*) with greater than 95% confidence.

For both the HIER BAYES and DAUME07 models, I found that performance was better for the variant which did not restrict possible labels based on the domain, while the ALL

Named Entity Recognition

Model	Precision	Recall	F1
	MUC-6		
TARGET ONLY	86.74	80.10	83.29
ALL DATA*	85.04	83.49	84.26
ALL DATA	86.00	82.71	84.32
DAUME07*	87.83	83.41	85.56
DAUME07	87.81	82.23	85.46
HIER BAYES*	88.59	84.97	86.74
HIER BAYES	**88.77**	**85.14**	**86.92**
	MUC-7		
TARGET ONLY	81.17	70.23	75.30
ALL DATA*	81.66	76.17	78.82
ALL DATA	82.20	70.91	76.14
DAUME07*	**83.33**	75.42	79.18
DAUME07	83.51	75.63	79.37
HIER BAYES*	82.90	76.95	79.82
HIER BAYES	83.17	**77.02**	**79.98**
	CoNLL		
TARGET ONLY	85.55	84.72	85.13
ALL DATA*	86.34	84.45	85.38
ALL DATA	86.58	83.90	85.22
DAUME07*	86.09	85.06	85.57
DAUME07	86.35	**85.26**	**85.80**
HIER BAYES*	86.33	85.06	85.69
HIER BAYES	**86.51**	85.13	**85.81**

Table 6.3: Named entity recognition results for each of the models. With the exception of the TARGET ONLY model, all three datasets were combined when training each of the models.

DATA model did benefit from the label restriction. For HIER BAYES and DAUME07, this result may be due to the structure of the models. Because both models have domain-specific features, the models likely learned that these labels were never actually allowed. However, when a feature does not occur in the data for a particular domain, then the domain-specific parameter for that feature will have positive weight due to evidence present in the other domains, which at test time can lead to assigning an illegal label to a word. This information that a word may be of some other (unknown to that domain) entity type may help prevent the model from mislabeling the word. For example, in CoNLL, nationalities, such as *Iraqi*

and *American*, are labeled as MISC. If a previously unseen nationality is encountered in the MUC testing data, the MUC model may be tempted to label is as a *location*, but this evidence from the CoNLL data may prevent that, by causing it to instead be labeled MISC, a label which will subsequently be ignored.

In most previous domain adaptation work, there is a distinction between *source* and *target* data. Results are only presented on the target data, and frequently the amount of training data in the target domain has been reduced, thus making it easier to show an improvement. The motivation for this reduction is usually to demonstrate that the user need only label a small amount of data and can then intelligently utilize the abundant source data. However, another common use case is that the user has a lot of labeled data, from various sources, and wishes to build the best possible model from that data. Our results show that, so long as the amount of data in each domain is not widely disparate, it is possible to achieve gains on all of the domains simultaneously.

6.5.2 Dependency parsing

I also tested the hierarchical domain adaptation model on an untyped dependency parsing task, to see how it performs on a more structurally complex task than sequence modeling. For these experiments, I used a CRF-based dependency parser, as described in section 5.2.3, for the base models.

For the dependency parsing experiments, I used OntoNotes Release 2.0 data (an earlier release of the data described in section 2.2.4). I once again converted the PCFG trees into dependency trees using the Collins head rules (Collins 2003).

I compared the same four domain adaptation models for dependency parsing as I did for the named entity experiments, once again setting $\sigma_* = 1.0$ and $\sigma_d = 0.1$. Unlike the named entity experiments however, there were no label set discrepancies between the domains, so only one version of each domain adaptation model was necessary, instead of the two versions in that section. The TARGET ONLY results are the same as in the dependency parsing experiments reported earlier in section 5.2.3.

.

The full dependency parsing results can be found in table 6.4. Firstly, I found that

Dependency Parsing

	Training		Testing		TARGET	ALL		HIER
	Range	# Sent	Range	# Sent	ONLY	DATA	DAUME07	BAYES
ABC	0–55	1195	56–69	199	83.32%	**88.97%**	87.30%	88.68%
CNN	0–375	5092	376–437	1521	85.53%	87.09%	86.41%	**87.26%**
MNB	0–17	509	18–25	245	77.06%	86.41%	84.70%	**86.71%**
NBC	0–29	552	30–39	149	76.21%	**85.82%**	85.01%	85.32%
PRI	0–89	1707	90–112	394	87.65%	90.28%	89.52%	**90.59%**
VOA	0–198	1512	199–264	383	89.17%	**92.11%**	90.67%	**92.09%**

Table 6.4: Dependency parsing results for each of the domain adaptation models. Performance is measured as unlabeled attachment accuracy.

DAUME07, which had outperformed the ALL DATA baseline for the sequence modeling task, performed worse than the baseline here, indicating that the transfer of information between domains in the more structurally complicated task is inherently more difficult. My model's gains over the ALL DATA baseline are quite small, but I tested their significance using a sentence-level paired t-test (over all of the data combined) and found them to be significant at $p < 10^{-5}$. I am unsure why some domains improved while others did not. It is not simply a consequence of training set size, but may be due to qualities of the domains themselves.

6.6 Summary

In this chapter, I used a hierarchical prior for multi-task learning in two very different scenarios. I first covered the general technique of using a hierarchical prior to link the feature weights for related tasks. I then used this to improve my joint parse and named entity recognition model, by linking that joint model with separate task-specific models for the two tasks. The task specific models were trained using data which had only been annotated for the single task. This resulted in gains on both tasks, thanks to the ability to incorporate information from the singly-annotated data.

I then used a hierarchical prior to do multi-domain learning. I linked three different NER corpora to one another, linking their features via a hierarchical prior, and showed

that this resulted in better performance for all three datasets. I also performed a comparable experiment on dependency parsing, this time linking all of the different sections of OntoNotes. In this case, the hierarchical model always beat the model trained on just a single domain, and was comparable to one trained on all the data combined.

There are several potentially future avenues for future work. In all the experiments in this chapter, I used a uniform variance for all the different features and domains. However, intuitively, it seems that some features should be more likely to be similar between tasks than others, and therefore we would like to be able to learn the sigmas as well.

There are also many potential uses for hierarchical priors in NLP. They may be useful for semi-supervised learning: you could use a pre-trained model to label large amounts of raw text, and then link a model trained on the automatically-annotated text with a model trained on the gold-standard text. In that case, the weights learned on the gold-standard text will be more influenced by the gold data than the non-gold data, but the non-gold data can still provide useful information about unseen or rarely seen features in the gold data. Quite recently, two papers (Berg-Kirkpatrick and Klein 2010, Iwata et al. 2010) used hierarchical priors to link parameters between different languages when doing unsupervised parsing.

Chapter 7

Conclusions

In this dissertation, I presented techniques for various kinds of joint modeling. In chapter 3, I addressed the case of modeling long-distance dependencies in information extraction systems; in this case, we were jointly modeling multiple, disparate parts of the same document. My proposed solution was to use a product-of-experts model to combine a state-of-the-art linear-chain CRF with a long-distance model. For named entity recognition, the long-distance model was designed to encourage identical words and phrases to be labeled consistently. For a template filling task, the long-distance model encouraged only one phrase to be labeled for each of the template slots. I used Gibbs sampling, a form of approximate inference, combined with simulated annealing, to find the best labeling in the product model.

I then turned to the task of jointly modeling different kinds of phenomena. Many higher-level NLP systems require input from multiple lower-level systems. Typically, this lower-level information is acquired by running data through a pipeline of annotators. In chapter 4, I proposed an alternative to the standard, greedy, 1-best pipeline. Instead of taking the most likely output at each stage, I generated samples instead, and passed them along to the next stage. This was done many times, and in the end the samples formed a majority vote classifier for the final stage. I found that this version of a pipeline outperformed a 1-best pipeline on two different tasks, and performed comparably to a k-best pipeline.

A major limitation of 1-best, k-best, and sampling pipelines is that information only

flows forward. Oftentimes you instead want a true joint model, where all levels of information can constrain and influence each other. In chapter 5, I gradually built up to a full joint model of parsing and named entity recognition. First, I presented a discriminative constituency parser. This parser was the first feature-rich CRF-based parser which could scale up beyond toy sentences. This parser also served as the backbone for several other models. I first used it to do nested named entity recognition. Most prior work on named entities has assumed a flat structure, and has ignored named entities which were nested inside of other named entities (e.g., *Charles Shaw Corp.*). The discriminative parser provided a natural framework for representing this nesting structure, while also allowing the utilization of previously developed features which had originally been designed for sequence (flat) models. Having shown how to use the parser for both parsing and named entity recognition separately, I then showed how to use it to model them jointly. By picking the proper representation, grammar, and features, I found that modeling these two phenomena together I could improve performance on both tasks.

One drawback to the joint model just described is the relatively small amount of training data, compared to the amount of data available which has been annotated with just a single task (chapter 6). With this in mind, I presented a hierarchical model which could leverage both jointly and singly labeled data to improve the joint model. This was done via a hierarchical prior, which loosely linked the feature weights for single-task and joint models, and allowed the weights for the joint model to be influenced by the single-task data.

Finally, I used the same hierarchical technique to jointly learn models for multiple domains or genres of text. In this case, the same type of model (a linear-chain CRF in one experiment, and a discriminative dependency parser in another) was used for each genre, but each had its own specific model, and the feature weights of the models were linked via a hierarchical prior.

There are several conclusions that can be drawn from the work presented here. Chapter 3 and chapter 4 both utilized sampling-based approaches to solve problems which could not be solved with exact inference methods. Regardless of the tasks being addressed, or the specific techiques being used, I think the fundamental idea here is important: language has very complex structure which cannot be modeled perfectly, but it is better to try to get the

model right, including many of the complex interactions, and accept that inference may not be perfect, than to have perfect inference in a severely handicapped model. Good inference techiques are important, but without properly capturing the complex structure of language, we can never hope to build systems which do true natural language understanding. That said, it is also not clear if a full joint model like the one presented in chapter 5 is the final answer. Adding new layers of annotations greatly increased the possible space of outputs, and as a result slowed inference down considerably. Part of the problem here may in fact have been my insistence on using an exact inference technique, and I do not know how well this would scale when even more types of information are added. I believe that future joint models will likely need to be set up in a manner that more naturally allows for high-quality approximate inference. Lastly, I think that hierarchical models have a bright future in NLP. They are an easy way to link related tasks. From an implementation persepctive, it is incredibly simple to add them to existing models, when using pointwise estimation. From a computational standpoint, they do not take much longer to optimize than the individual models from which they are constructed. And, from a modeling perspective, they provide an elegant way to effectively share information between related tasks and datasets.

There are many avenues for future work building off of the ideas in this thesis, some of which have already begun to be explored. Several of the models presented here utilized the OntoNotes corpus. I used this for the joint parse and named entity experiments, but the data is annotated with more than just these two kinds of information. It also contains semantic role labels, coreference information, and word senses. It is easy to see how these kinds of information should also be able to influence and constrain one another. If two entities are coreferent, then they should be the same named entity type. Semantic role labels directly utilize phrases in the tree, and also knowing what types of roles certain phrases tend to play should influence where they are placed in the parse tree. It would be wonderful to see a joint model of all these phenomena. One of the most basic ideas behind the joint model I presented was the fact that the features could decompose: some were over just the parse structure, some were over only the named entities, and some were over both. Future joint models will need to find even more ways to decompose the model for the sake of tractability. One potential way to do this would be *dual decomposition* (Rush et al. 2010, Koo et al. 2010). Dual decomposition allows us to do inference in complicated,

previously intractable, models, by factoring the model into two (or more) components, and then forcing the components to agree on the subset of variables which they share. For instance, it would be natural to jointly model named entities and coreference information. Such a model could be combined with the joint parse and named entity model from this thesis, and the two models would be forced to agree on the named entity annotations in the data.

Many of the techniques proposed in this thesis could be useful for tasks beyond joint learning. The sampling-based approaches presented in chapters 3 and 4 have already helped foster the widespread use of sampling for approximate inference in many tasks, as was discussed in section 4.7. The hierarchical models in chapter 6 similarly have the potential to be quite useful. Hierarchical priors have already begun to be used for soft parameter typing between languages for parsing models (see section 6.6).

Ultimately, to do real natural language understanding, we need to first to a good job at these kinds of low-level tasks. It is my hope that the ideas presented in this dissertation have brought us closer to that goal, both with the technologies used and by showing that is preferable to jointly model things (whether it be words, tasks, or genres) which are related. Computationally we want to decouple tasks, because it will usually make inference faster and simpler, but intellectually we know that these tasks are related, and that we will never be able to truly do natural language understanding while we keep them separate.

Appendix A

OntoNotes data inconsistencies

While other work has utilized the OntoNotes corpus (Pradhan et al. 2007, Yu et al. 2008), the papers on which this thesis is based are the first work to my knowledge to simultaneously model the multiple levels of annotation available. Because this is a new corpus, still under development, it is not surprising that there were places where the data was inconsistently annotated, namely with crossing brackets between named entity and tree annotations.

In the places where I found inconsistent annotation it was rarely the case that the different levels of annotation were inherently inconsistent, but rather the inconsistency resulted from somewhat arbitrary choices made by the annotators. For example, when the last word in a sentence ends with a period, such as *Corp.*, one period functions both to mark the abbreviation and the end of the sentence. The convention of the PennTreebank is to separate the final period and treat it as the end of sentence marker, but when the final word is also part of an entity, that final period was frequently included in the named entity annotation, resulting in the sentence terminating period being part of the entity, and the entity not corresponding to a single phrase. See figure A.1 for an illustration from the data. In this case, I removed the terminating period from the entity, to produce a consistent annotation.

Overall, I found that 656 entities, out of 55,665 total, could not be aligned to a phrase, or multiple contiguous children of a node. I identified and corrected the following sources of inconsistencies:

Periods and abbreviations. This is the problem described above with the *Corp.* example. I corrected it by removing the sentence terminating final period from the entity

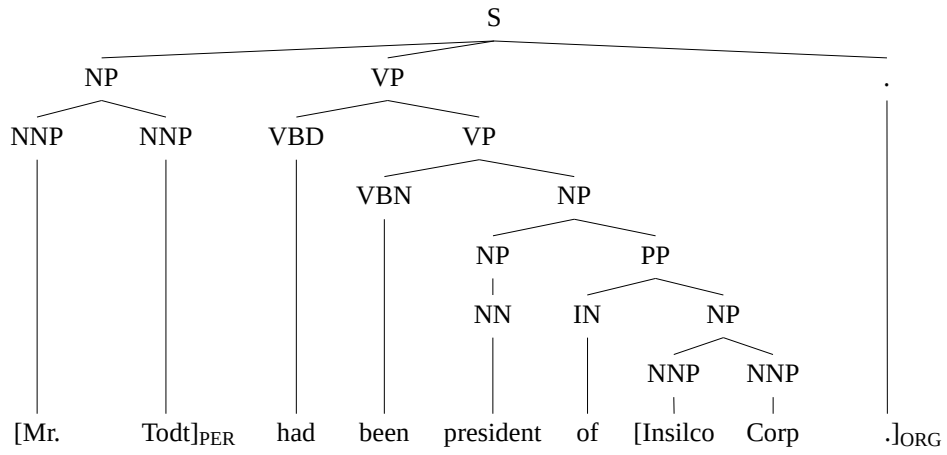


Figure A.1: An example from the OntoNotes data of inconsistently labeled named entity and parse structure.The inclusion of the final period in the named entity results in the named entity structure having crossing brackets with the parse structure.

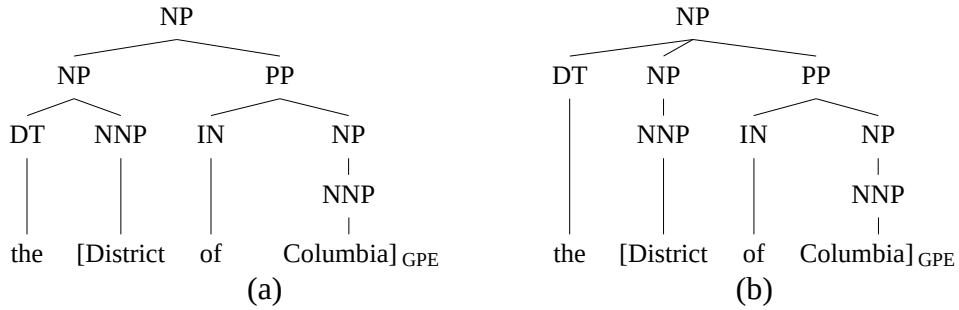


Figure A.2: (a) Another example from the data of inconsistently labeled named entity and parse structure. In this instance, we flatten the nested NP, resulting in (b), so that the named entity corresponds to a contiguous set of children of the top-level NP.

annotation.

Determiners and PPs. Noun phrases composed of a nested noun phrase and a prepositional phrase were problematic when they also consisted of a determiner followed by an entity. I dealt with this by flattening the nested NP, as illustrated in figure A.2.

Adjectives and PPs. This problem is similar to the previous problem, with the difference being that there are also adjectives preceding the entity. The solution is also similar to the solution to the previous problem. I moved the adjectives from the nested NP into the main NP.

These three modifications to the data solved most, but not all, of the inconsistencies. Another source of problems was conjunctions, such as *North and South Korea*, where *North and South* is a phrase, but *South Korea* is an entity. The rest of the errors seemed to be due to annotation errors and other random weirdnesses. I ended up unable to make 0.4% of the entities consistent with the parses, so I omitted those entities from the training and test data.

One more change made to the data was with respect to possessive NPs. When I encountered noun phrases which ended with *(POS 's)* or *(POS ')*, I modified the internal structure of the NP. Originally, these NPs were flat, but I introduced a new nested NP which contained the entire contents of the original NP except for the POS. The original NP label was then changed to POSSNP. This change is motivated by the status of *'s* as a phrasal affix or clitic: It is the NP preceding *'s* that is structurally equivalent to other NPs, not the larger unit that includes *'s*. This change has the additional benefit in this context that more named entities will correspond to a single phrase in the parse tree, rather than a contiguous set of phrases.

Bibliography

2007. The British National Corpus, version 3 (BNC XML Edition). Distributed by Oxford University Computing Services on behalf of the BNC Consortium. URL: http://www.natcorp.ox.ac.uk/.

Abney, S. 1997. Stochastic attribute-value grammars. *Computational Linguistics* 23:597–618.

Alex, Beatrice, Barry Haddow, and Claire Grover. 2007. Recognising nested named entities in biomedical text. In *BioNLP Workshop at ACL 2007*, pp. 65–72.

Ando, Rie Kubota, and Tong Zhang. 2005. A high-performance semi-supervised learning method for text chunking. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pp. 1–9. Association for Computational Linguistics. DOI: http://dx.doi.org/10.3115/1219840.1219841.

Andrew, Galen. 2006. A hybrid markov/semi-markov conditional random field for sequence segmentation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2006).*

Andrieu, C., N. d Freitas, A. Doucet, and M. I. Jordan. 2003. An introduction to MCMC for machine learning. *Machine Learning* 50:5–43.

Arulampalam, M. Sanjeev, Simon Maskell, and Neil Gordon. 2002. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* 50:174–188.

Baxter, J. 1997. A bayesian/information theoretic model of learning to learn via multiple task sampling. In *Machine Learning*, volume 28.

Bentivogli, L., Bernardo Magnini, Ido Dagan, H. T. Dang, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *Proceedings of the TAC-09 Text Analysis Conference*.

Berg-Kirkpatrick, Taylor, and Dan Klein. 2010. Phylogenetic grammar induction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 1288–1297. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/P10-1131`.

Blunsom, Phil, and Trevor Cohn. 2010. Inducing synchronous grammars with slice sampling. In *Proceedings of Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*.

Blunsom, Phil, Trevor Cohn, Chris Dyer, and Miles Osborne. 2009. A gibbs sampler for phrasal synchronous grammar induction. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pp. 782–790. URL: `http://www.aclweb.org/anthology/P/P09/P09-1088`.

Bod, Rens. 1995. The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars. In *Proceedings of EACL 1995*.

Borthwick, A. 1999. *A Maximum Entropy Approach to Named Entity Recognition.* PhD thesis, New York University.

Brown, P.F., P.V. Desouza, R.L. Mercer, V.J.D. Pietra, and J.C. Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18(4):467–479.

Buchholz, S., and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pp. 149–164. Association for Computational Linguistics.

Bunescu, R., and R. J. Mooney. 2004. Collective information extraction with relational Markov networks. In *Proceedings of the 42nd ACL*, pp. 439–446.

Byrne, Kate. 2007. Nested named entity recognition in historical archive text. In *ICSC '07: Proceedings of the International Conference on Semantic Computing*, pp. 589–596.

Carreras, X., M. Collins, and T. Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pp. 9–16.

Carreras, Xavier, and Lluís Màrquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL 2004*.

Carreras, Xavier, and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL 2005*.

Caruana, R. 1997. Multitask learning. In *Machine Learning*, volume 28.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *HLT-NAACL 2000*, pp. 132–139.

Charniak, Eugene, and Mark Johnson. 2005. Coarse-to-fine *n*-best parsing and maxent discriminative reranking. In *ACL 43*, pp. 173–180.

Chelba, Ciprian, and Alex Acero. 2004. Adaptation of a maximum entropy capitalizer: Little data can help a lot. In *EMNLP 2004*.

Chieu, H. L., and H. T. Ng. 2002. Named entity recognition: a maximum entropy approach using global information. In *Proceedings of the 19th Coling*, pp. 190–196.

Chinchor, N. 1998. Overview of MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

Clark, Alexander. 2003. Combining distributional and morphological information for part of speech induction. In *Proceedings of the tenth Annual Meeting of the European Association for Computational Linguistics (EACL)*, pp. 59–66.

Cohn, Trevor, and Philip Blunsom. 2005. Semantic role labelling with tree conditional random fields. In *CoNLL 2005*.

Cohn, Trevor, and Phil Blunsom. 2009. A Bayesian model of syntax-directed tree to string grammar induction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 352–361. URL: `http://www.aclweb.org/anthology/D/D09/D09-1037`.

Cohn, Trevor, and Phil Blunsom. 2010. Blocked inference in Bayesian tree substitution grammars. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, p. To Appear.

Cohn, Trevor, Sharon Goldwater, and Phil Blunsom. 2009. Inducing compact but accurate tree-substitution grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pp. 548–556. URL: `http://www.aclweb.org/anthology/N/N09/N09-1062`.

Collier, Nigel, J. Kim, Y. Tateisi, T. Ohta, and Y. Tsuruoka (eds.). 2004. *Proceedings of the International Joint Workshop on NLP in Biomedicine and its Applications*.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *ACL 1997*.

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *ICML 17*, pp. 175–182.

Collins, Michael. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pp. 1–8. Association for Computational Linguistics. DOI: http://dx.doi.org/10.3115/1118693.1118694.

Collins, M. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics* 29(4):589–637.

Collobert, R., and J. Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine learning*, pp. 160–167.

Cowell, R. G., A. Philip Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. 1999. *Probabilistic Networks and Expert Systems*. Springer-Verlag.

Crouch, Richard. 2005. Packed rewriting for mapping semantics to KR. In *Proceedings of the 6th International Workshop on Computational Semantics*.

Curran, J. R., and S. Clark. 1999. Language independent NER using a maximum entropy tagger. In *CoNLL 1999*, pp. 164–167.

Dagan, Ido, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognizing textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*.

Darroch, J.N., and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics* 43(5):1470–1480.

Daumé III, Hal. 2007. Frustratingly easy domain adaptation. In *Conference of the Association for Computational Linguistics (ACL)*.

Daumé III, Hal, and Daniel Marcu. 2006. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*.

Della Pietra, S., V. Della Pietra, and J. Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19:380–393.

DeNero, John, Alexandre Bouchard-Côté, and Dan Klein. 2008. Sampling alignment structure under a bayesian translation model. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 314–323. Association for Computational Linguistics.

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 6(8):451–455.

Eisner, Jason. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*.

Elidan, Gal, Benjamin Packer, Geremy Heitz, and Daphne Koller. 2008. Convex point estimation using undirected bayesian transfer hierarchies. In *UAI 2008*.

Evgeniou, T., C. Micchelli, and M. Pontil. 2005. Learning multiple tasks with kernel methods. In *Journal of Machine Learning Research*.

Finkel, J., S. Dingare, H. Nguyen, M. Nissim, and C. D. Manning. 2004. Exploiting context for biomedical entity recognition: from syntax to the web. In *Joint Workshop on Natural Language Processing in Biomedicine and Its Applications at Coling 2004*.

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL 2005*.

Finkel, Jenny Rose, Trond Grenager, and Christopher D. Manning. 2007. The infinite tree. In *In Association for Computational Linguistics (ACL)*.

Finkel, Jenny Rose, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based conditional random field parsing. In *ACL/HLT-2008*.

Finkel, Jenny Rose, and Christopher D. Manning. 2009a. Hierarchical bayesian domain adaptation. In *Proceedings of the North American Association of Computational Linguistics (NAACL 2009)*. URL: `pubs/hdba.pdf`.

Finkel, Jenny Rose, and Christopher D. Manning. 2009b. Joint parsing and named entity recognition. In *Proceedings of the North American Association of Computational Linguistics (NAACL 2009)*. URL: `pubs/joint-parse-ner.pdf`.

Finkel, Jenny Rose, and Christopher D. Manning. 2009c. Nested named entity recognition. In *Proceedings of EMNLP 2009*. URL: `pubs/nested-ner.pdf`.

Finkel, Jenny Rose, and Christopher D. Manning. 2010. Hierarchical joint learning: Improving joint parsing and named entity recognition with non-jointly labeled data. In *Proceedings of ACL 2010*. URL: `pubs/hier-joint.pdf`.

Finkel, Jenny Rose, Christopher D. Manning, and Andrew Y. Ng. 2006. Solving the problem of cascading errors: Approximate bayesian inference for linguistic annotation pipelines. In *EMNLP 2006.*

Freitag, D. 1998. *Machine learning for information extraction in informal domains*. PhD thesis, Carnegie Mellon University.

Freitag, D., and A. McCallum. 1999. Information extraction with HMMs and shrinkage. In *AAAI 1999 Workshop on Machine Learning for Information Extraction*.

Freund, Yoav, and Robert E. Schapire. 1998. Large margin classification using the perceptron algorithm. In *Machine Learning*, pp. 277–296.

Gale, W.A., K.W. Church, and D. Yarowsky. 1992. One sense per discourse. In *Proceedings of the workshop on Speech and Natural Language*, p. 237. Association for Computational Linguistics.

Gelman, A., J. B. Carlin, H. S. Stern, and Donald D. B. Rubin. 2003. *Bayesian Data Analysis*. Chapman & Hall.

Gelman, Andrew, and Jennifer Hill. 2006. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Geman, S., and D. Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transitions on Pattern Analysis and Machine Intelligence* 6:721–741.

Geman, Stuart, and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of ACL 2002*.

Giampiccolo, Danilo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

Giampiccolo, Danilo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2008. The fourth pascal recognizing textual entailment challenge. In *Proceedings of the TAC-08 Text Analysis Conference*.

Goldwater, Sharon, and Tom Griffiths. 2007. A fully bayesian approach to unsupervised part-of-speech tagging. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 744–751. Association for Computational Linguistics.

Goodman, Joshua. 1998. *Parsing Inside-Out*. PhD thesis, Harvard University.

Gordon, N.J., D.J. Salmond, and A.F.M. Smith. 1993. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings*, volume 140, pp. 107–113.

Graff, D., J. Kong, K. Chen, and K. Maeda. 2007. English gigaword third edition. *Linguistic Data Consortium*.

Gu, Baohua. 2006. Recognizing nested named entities in GENIA corpus. In *BioNLP Workshop at HLT-NAACL 2006*, pp. 112–113.

Haghighi, A., and D. Klein. 2007. Unsupervised coreference resolution in a nonparametric bayesian model. In *ACL 2007*.

Haghighi, Aria, Kristina Toutanova, and Christopher D. Manning. 2005. A joint model for semantic role labeling. In *Proceedings of CoNLL 2005*.

Hana, Jiří, and Hana Hanová. 2002. Manual for morphological annotation. Technical Report TR-2002-14, UK MFF CKL.

Henderson, James. 2004. Discriminative training of a neural network statistical parser. In *ACL 42*, pp. 96–103.

Hinton, Geoffrey. 2000. Training products of experts by minimizing contrastive divergence. *Neural Computation* 14:2002.

Hirschman, L., A. Yeh, C. Blaschke, and A. Valencia. 2005. Overview of BioCreAtIvE: critical assessment of information extraction for biology. *BMC bioinformatics* 6(Suppl 1):S1.

Hollingshead, Kristy, and Brian Roark. 2007. Pipeline iteration. In *ACL 2007*.

Hovy, Eduard, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: The 90% solution. In *HLT-NAACL 2006*.

Huang, Liang, and David Chiang. 2005. Better *k*-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies*.

Ido, Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Iwata, Tomoharu, Daichi Mochihashi, and Hiroshi Sawada. 2010. Learning common grammar from multilingual corpus. In *Proceedings of the ACL 2010 Conference Short Papers*, pp. 184–188. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/P10-2034`.

Johnson, Mark. 2001. Joint and conditional estimation of tagging and parsing models. In *Meeting of the Association for Computational Linguistics*, pp. 314–321.

Johnson, M., T. Griffiths, and S. Goldwater. 2007. Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models. In *NIPS 2007*.

Johnson, Mark, and Thomas L. Griffiths. 2007. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *In Proceedings of the North American Conference on Computational Linguistics (NAACL 07*.

Jousse, Florent, Rémi Gilleron, Isabelle Tellier, and Marc Tommasi. 2006. Conditional Random Fields for XML trees. In *ECML Workshop on Mining and Learning in Graphs*.

Karttunen, Lauri. 2000. Applications of finite-state transducers in natural-language processing. In *Proceesings of the Fifth International Conference on Implementation and Application of Automata*.

Kazama, Jun'ichi, Takaki Makino, Yoshihiro Ohta, and Jun'ichi Tsujii. 2002. Tuning support vector machines for biomedical named entity recognition. In *Proceedings of the Workshop on Natural Language Processing in the Biomedical Domain (ACL 2002)*.

Kim, Jin-Dong, Tomoko Ohta, Yuka Teteisi, and Jun'ichi Tsujii. 2003. Genia corpus – a semantically annotated corpus for bio-textmining. *Bioinformatics* 19(suppl. 1):i180–i182.

Kim, M., Y. S. Han, and K. Choi. 1995. Collocation map for overcoming data sparseness. In *Proceedings of the 7th EACL*, pp. 53–59.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220:671–680.

Klein, D., and C.D. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*.

Klein, Dan, and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the Association of Computational Linguistics (ACL)*.

Koller, D., and N. Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Koo, T., X. Carreras, and M. Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL*, volume 8, pp. 595–603.

Koo, Terry, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *EMNLP*.

Koomen, Peter, Vasin Punyakanok, Dan Roth, and Wen tau Yih. 2005. Generalized inference with multiple semantic role labeling systems. In *Proceedings of CoNLL 2005*, pp. 181–184.

Krishnan, Vijay, and Christopher D. Manning. 2006. An effective two-stage model for exploiting non-local dependencies in named entity recognition. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 1121–1128. Association for Computational Linguistics. DOI: http://dx.doi.org/10.3115/1220175.1220316.

Laarhoven, P. J. Van, and E. H. L. Arts. 1987. *Simulated Annealing: Theory and Applications*. Reidel Publishers.

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*, pp. 282–289. Morgan Kaufmann, San Francisco, CA.

Lee, Su-In, Vassil Chatalbashev, David Vickrey, and Daphne Koller. 2007. Learning a meta-level prior for feature relevance from multiple related tasks. In *ICML '07: Proceedings of the 24th international conference on Machine learning*.

Leek, T. R. 1997. Information extraction using hidden Markov models. Master's thesis, U.C. San Diego.

Liang, P. 2005. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology.

Liu, Dong C., and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Programming* 45(3, (Ser. B)):503–528.

MacCartney, Bill, Trond Grenager, Marie de Marneffe, Daniel Cer, and Christopher D. Manning. 2006. Learning to recognize features of valid textual entailments. In *Proceedings of NAACL-HTL 2006*.

Malouf, R. 2002. Markov models for language-independent named entity recognition. In *Proceedings of the 6th CoNLL*, pp. 187–190.

Manning, Christopher D., and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19 (2):313–330.

Márquez, L., L. Padrè, M. Surdeanu, and L. Villarejo. 2007a. UPC: Experiments with joint learning within semeval task 9. In *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*.

Márquez, L., L. Villarejo, M.A. Martí, and M. Taulè. 2007b. Semeval-2007 task 09: Multilevel semantic annotation of Catalan and Spanish. In *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*.

Martí, M.A., M. Taulè, M. Bertran, and L. Márquez. 2007. Ancora: Multilingual and multilevel annotated corpora. MS, Universitat de Barcelona. URL: `http://clic.ub.edu/ancora/ancora-corpus.pdf`.

Maxwell, III, John T., and Ronald M. Kaplan. 1995. A method for disjunctive constraint satisfaction. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen (eds.), *Formal Issues in Lexical-Functional Grammar*, number 47 in CSLI Lecture Notes Series, chapter 14, pp. 381–481. CSLI Publications.

McCallum, A., D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *ICML 2000*, pp. 591–598. Morgan Kaufmann, San Francisco, CA.

McClosky, David, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies 2010 Conference (NAACL-HLT'10), Main Conference*.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005a. Flexible text segmentation with structured multilabel classification. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 987–994.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005b. Online large-margin training of dependency parsers. In *ACL 2005*.

Mikheev, A., M. Moens, and C. Grover. 1999. Named entity recognition without gazetteers. In *EACL 1999*, pp. 1–8.

Miller, Scott, Heidi Fox, Lance Ramshaw, and Ralph Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *In 6th Applied Natural Language Processing Conference*, pp. 226–233.

Miller, S., J. Guinness, and A. Zamanian. 2004. Name tagging with word clusters and discriminative training. In *Proceedings of HLT-NAACL*, volume 4.

Ng, Andrew, and Michael Jordan. 2001. Convergence rates of the voting Gibbs classifier, with application to Bayesian feature selection. In *Proceedings of the Eighteenth International Conference on Machine Learning*.

Ng, Andrew, and Michael Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems (NIPS)*.

Nivre, J., J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Association for Computational Linguistics.

Nivre, J., J. Hall, and J. Nilsson. 2006. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6. Citeseer.

Ohta, Tomoko, Yuka Tateisi, and Jin-Dong Kim. 2002. The GENIA corpus: an annotated research abstract corpus in molecular biology domain. In *Proceedings of the second international conference on Human Language Technology Research*, pp. 82–86.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufman.

Petrov, Slav, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *ACL 44/COLING 21*, pp. 433–440.

Petrov, Slav, and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In *NIPS*.

Post, Matt, and Daniel Gildea. 2009. Bayesian learning of a tree substitution grammar. In *ACL-IJCNLP '09: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pp. 45–48. Association for Computational Linguistics.

Pradhan, Sameer S., Lance Ramshaw, Ralph Weischedel, Jessica MacBride, and Linnea Micciulla. 2007. Unrestricted coreference: Identifying entities and events

in ontonotes. *International Conference on Semantic Computing* 0:446–453. DOI: http://doi.ieeecomputersociety.org/10.1109/ICSC.2007.93.

Rabiner, L. R. 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

Ratinov, L., and D. Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pp. 147–155. Association for Computational Linguistics.

Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *EMNLP 2*, pp. 1–10.

Rush, Alexander M, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *EMNLP*.

Sang, Erik F. Tjong Kim, and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*.

Sarawagi, Sunita, and William W. Cohen. 2004. Semi-markov conditional random fields for information extraction. In *In Advances in Neural Information Processing Systems 17*, pp. 1185–1192.

Shen, Dan, Jie Zhang, Guodong Zhou, Jian Su, and Chew-Lim Tan. 2003. Effective adaptation of a hidden markov model-based named entity recognizer for biomedical domain. In *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine*. Association for Computational Linguistics (ACL 2003).

Singh, S., K. Schultz, and A. McCallum. 2009. Bi-directional joint inference for entity resolution and segmentation using imperatively-defined factor graphs. *Machine Learning and Knowledge Discovery in Databases* pp. 414–429.

Snyder, Benjamin, Tahira Naseem, Jacob Eisenstein, and Regina Barzilay. 2008. Unsupervised multilingual learning for POS tagging. In *EMNLP '08: Proceedings of the*

Conference on Empirical Methods in Natural Language Processing, pp. 1041–1050. Association for Computational Linguistics.

Snyder, Benjamin, Tahira Naseem, Jacob Eisenstein, and Regina Barzilay. 2009. Adding more languages improves unsupervised multilingual part-of-speech tagging: a bayesian non-parametric approach. In *NAACL '09: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 83–91. Association for Computational Linguistics.

Stolcke, Andreas. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics* 21:165–202.

Sundheim, B.M. 1996. Overview of results of the MUC-6 evaluation. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*.

Surdeanu, Mihai, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the 12th Conference on Computational Natural Language Learning (CoNLL)*.

Sutton, C., and A. McCallum. 2004. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning and Its connections to Other Fields*.

Sutton, Charles, and Andrew McCallum. 2005. Joint parsing and semantic role labeling. In *Conference on Natural Language Learning (CoNLL)*.

Sutton, Charles, and Andrew McCallum. 2007. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar (eds.), *Introduction to Statistical Relational Learning*. MIT Press.

Taskar, B., P. Abbeel, and D. Koller. 2002. Discriminative probabilistic models for relational data. In *Proceedings of the 18th Conference on Uncertianty in Artificial Intelligence (UAI-02)*, pp. 485–494.

Taskar, B., C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *NIPS*.

Taskar, Ben, Dan Klein, Michael Collins, Daphne Koller, and Christopher D. Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Toutanova, K., A. Haghighi, and C. D. Manning. 2005. Joint learning improves semantic role labeling. In *ACL 2005*.

Toutanova, Kristina, and Mark Johnson. 2008. A Bayesian LDA-based model for semi-supervised part-of-speech tagging. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis (eds.), *Advances in Neural Information Processing Systems 20*, pp. 1521–1528. MIT Press.

Toutanova, K., D. Klein, C. Manning, and Y. Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL 2003*.

Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)* 6:1453–1484.

Tsuruoka, Yoshimasa, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou, and Jun'ichi Tsujii. 2005. Developing a robust part-of-speech tagger for biomedical text. In *Advances in Informatics - 10th Panhellenic Conference on Informatics, LNCS 3746*, pp. 382–392.

Tsuruoka, Yoshimasa, and Jun'ichi Tsujii. 2003. Boosting precision and recall of dictionary-based protein name recognition. In *Proceedings of the ACL-03 Workshop on Natural Language Processing in Biomedicine*, pp. 41–48.

Turian, Joseph, and I. Dan Melamed. 2006. Advances in discriminative parsing. In *ACL 44*, pp. 873–880.

Turian, Joseph, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 384–394. Association

for Computational Linguistics. URL: http://www.aclweb.org/anthology/P10-1040.

Turian, Joseph, Ben Wellington, and I. Dan Melamed. 2007. Scalable discriminative learning for natural language parsing and translation. In *Advances in Neural Information Processing Systems 19*, pp. 1409–1416. MIT Press.

Vilain, Marc, Jonathan Huggins, and Ben Wellner. 2009. A simple feature-copying approach for long-distance dependencies. In *CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pp. 192–200. Association for Computational Linguistics.

Vishwanathan, S. V. N., Nichol N. Schraudolph, Mark W. Schmidt, and Kevin P. Murphy. 2006. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML 23*, pp. 969–976.

Wainwright, Martin J., and Michael I. Jordan. 2008. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc.

Wellner, Ben, Andrew McCallum, Fuchun Peng, and Michael Hay. 2004. An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence*.

Xue, Ya, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. 2007. Multi-task learning for classification with dirichlet process priors. *J. Mach. Learn. Res.* 8.

Yamada, H., and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, pp. 195–206.

Yu, Kai, Volker Tresp, and Anton Schwaighofer. 2005. Learning gaussian processes from multiple tasks. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*.

Yu, Liang-Chih, Chung-Hsien Wu, and Eduard Hovy. 2008. OntoNotes: Corpus cleanup of mistaken agreement using word sense disambiguation. In *Proceedings of the 22nd*

International Conference on Computational Linguistics (Coling 2008), pp. 1057–1064. URL: `http://www.aclweb.org/anthology/C08-1133`.

Zhang, H., M. Zhang, C.L. Tan, and H. Li. 2009. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pp. 1552–1560. Association for Computational Linguistics.

Zhang, Jie, Dan Shen, Guodong Zhou, Jian Su, and Chew-Lim Tan. 2004. Enhancing HMM-based biomedical named entity recognition by studying special phenomena. *Journal of Biomedical Informatics* 37(6):411–422.

Zhang, Yue, and Stephen Clark. 2008. Joint word segmentation and POS tagging using a single perceptron. In *ACL 2008*.

Zhao, Hai, Wenliang Chen, Jun'ichi Kazama, Kiyotaka Uchimoto, and Kentaro Torisawa. 2009. Multilingual dependency learning: Exploiting rich features for tagging syntactic and semantic dependencies. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pp. 61–66. Association for Computational Linguistics.

Zhou, Guodong. 2006. Recognizing names in biomedical texts using mutual information independence model and SVM plus sigmoid. *International Journal of Medical Informatics* 75:456–467.

Zhou, GuoDong, and Jian Su. 2004. Exploring deep knowledge resources in biomedical name recognition. In *Joint Workshop on Natural Language Processing in Biomedicine and Its Applications at Coling 2004*.

Zhou, Guodong, Jie Zhang, Jian Su, Dan Shen, and Chewlim Tan. 2004. Recognizing names in biomedical texts: a machine learning approach. *Bioinformatics* 20(7):1178–1190.