LEARNING WITH JOINT INFERENCE AND LATENT LINGUISTIC STRUCTURE IN GRAPHICAL MODELS

A Dissertation Presented

by

JASON NARADOWSKY

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2014

Typeset in LaTeX 2_ε.

Acknowledgements

They say calm seas never made a skilled sailor. I take this as some consolation: throughout the course of my dissertation I often felt like a vessel trapped in a tumultuous sea, rammed with unexpected challenges, and pushed to many far-flung ports. Even as far as PhDs go, the last six years have felt like both an oddity and an odyssey, and it has been an absolutely transformative experience. During the course of my graduate studies I have lived on four continents and have been a member of just as many research labs. As such I owe a tremendous debt of gratitude to the many people who have helped me along the way.

First and foremost, I am forever indebted to my primary advisor, David Smith. David has been a constant source of intellectual inspiration, and it is no surprise that many of the ideas David toyed with during his thesis are also present throughout my own. Through the many less-than-ideal situations we've found ourselves in, I've always been thankful for having the the freedom to try to make the best of them in my own way.

I must also thank the incomparable Mark Johnson. Mark is generous with his time, precise in every endeavor, and a scientist through-and-through. While Mark's assistance in shaping the dissertation's narrative was indispensable, I'm confident that a bigger debt of gratitude will be owed for countless snippets of great advice, and the lead-by-example lessons I'll be incorporating into my research life for years to come.

Though we did not pursue much research together, Andrew McCallum played a pivotal role in bringing me to UMass and introducing me to many of the concepts and themes that would later serve as the foundation of this thesis. And while much of my time at UMass was spent in the classroom, I have enjoyed pub nights, casual chats, and an occasional lunch with IESL peers: Sameer Singh, Greg Druck, Limin Yao, David Mimno, Kedar Bellare, Adam Saunders, and Sebastian Riedel. Leeanne Leclerc and Kate Moruzzi have been incredibly helpful in guiding me through the exorbitant amount of additional paperwork and bureaucratic hoops my research path has required, and it has been *greatly* appreciated. Additional thanks to Joe Pater, Lyn Frazier, and other members of the UMass Linguistics faculty for helping maintain the charade of being a competent linguist.

I would never have embarked on this long educational journey if not for the guidance and special attention given to me by a few exemplary teachers. A special thanks to my SUNY Oswego professors, Craig Graci and David Vampola, without whom I may never have discovered this field, and my serious interest in AI and NLP may have never been sparked.

Similarly, I have been blessed to have started my research career under the auspices of Sharon Goldwater, and I'm thankful for her role in nudging me from student to researcher. I have been similarly fortunate to have worked early in my PhD career with Kristina Toutanova, and I'm thankful for her role in the nudge from researcher to researcher-who-wants-an-industry-job.

I'm sincerely indebted to Yuji Matsumoto and Kevin Duh for hosting me in Japan on more than one occasion, and to an innumerable number of the nicest graduate students at the Nara Institute of Science and Technology (NAIST). If the reader of this acknowledgement is a PhD student, then I highly recommend the EAPSI fellowship program (and specifically through collaboration with Japanese Society for the Promotion of Science). But in that case I also recommend skipping the rest of the front matter and reading a more substantive section of the dissertation!

And perhaps most importantly, to all the close friends who have helped to shaped not just my PhD experience, but my aspirations, perspective, and life throughout the past decade of graduate school. To Mallory Earnshaw, Van Dang, Hanna Wallach, Tim Vieira, Bevan K. Jones, Benjamin Börschinger, and Maria Chan, thank you.

Abstract

A human listener, charged with the difficult task of mapping language to meaning, must infer a rich hierarchy of linguistic structures, beginning with an utterance and culminating in an understanding of what was spoken. Much in the same manner, developing complete natural language processing systems requires the processing of many different layers of linguistic information in order to solve complex tasks, like answering a query or translating a document.

Historically the community has largely adopted a "divide and conquer" strategy, choosing to split up such complex tasks into smaller fragments which can be tackled independently, with the hope that these smaller contributions will also yield benefits to NLP systems as a whole. These individual components can be laid out in a pipeline and processed in turn, one system's output becoming input for the next. This approach poses two problems. First, errors propagate, and, much like the childhood game of "telephone", combining systems in this manner can lead to unintelligible outcomes. Second, each component task requires annotated training data to act as supervision for training the model. These annotations are often expensive and time-consuming to produce, may differ from each other in genre and style, and may not match the intended application.

In this dissertation we pursue novel extensions of joint inference techniques for natural language processing. We present a framework that offers a general method for constructing and performing inference using graphical model formulations of typical NLP problems. Models are composed using weighted Boolean logic constraints, inference is performed using belief propagation. The systems we develop are composed of two parts: one a representation of syntax, the other a desired end task (part-of-speech tagging, semantic role labeling, named entity recognition, or relation extraction). By modeling these problems jointly, both models are trained in a single, integrated process, with uncertainty propagated between them. This mitigates the accumulation of errors typical of pipelined approaches. We further advance previous methods for performing efficient inference on graphical model representations of combinatorial structure, like dependency syntax, extending it to various forms of phrase structure parsing.

Finding appropriate training data is a crucial problem for joint inference models. We observe that in many circumstances, the output of earlier components of the pipeline is often irrelevant – only the end task output is important. Yet we often have strong a priori assumptions regarding what this structure might look like: for phrase structure syntax the model should represent a valid tree, for dependency syntax it should represent a directed graph. We propose a novel marginalization-based training method in which the error signal from end task annotations is used to guide the induction of a constrained latent syntactic representation. This allows training in the absence of syntactic training data, where the latent syntactic structure is instead optimized to best support the end task predictions. We find that across many NLP tasks this training method offers performance comparable to fully supervised training of each individual component, and in some instances improves upon it by learning latent structures which are more appropriate for the task.

Contents

List of Figures

List of Tables

1 Introduction

When humans interpret language they infer a rich hierarchy of linguistic structures prior to constructing a semantic and pragmatic representation. Many of the tasks commonly explored in natural language processing (NLP) are approached in the same way: first, the system recovers the relevant supporting linguistic structure (a parse tree, a tag sequence, etc.), which it then utilizes to improve performance on the desired task. However, the working assumption underlying most NLP research is that linguistic processing can be modularized (often into lexical, morphological, syntactic, semantic, and pragmatic information), and that these modules can be modeled and trained independently. The resulting models can be composed in a serial fashion, feeding the output of one component forward to serve as input to the next, in a pipeline, in order to perform complex NLP tasks.

A typical pipeline might comprise a tagger, a parser, and an **end task** - the goal of the system. While the notion of an end task varies, we consider tasks like named entity recognition, relation extraction, and semantic role labeling to be sufficiently close to a user's goal to be considered end tasks. Beginning with a sentence of raw text, a tagging model might pair each word with its most probable part-of-speech tag, before passing this information forward to the parser. The parser, leveraging the additional information provided by the tag sequence, assigns the most probable tree, and so forth. This provides a great deal of structured information from which to make end task predictions, but each model pushes forward only a fraction of the information it captures, and information never flows backward.

Linguistic evidence points to a decidedly different view of the human language processor, one in which information can flow more freely between components, helping to resolve ambiguity *jointly* [1–4]. Similar strategies have been pursued in the NLP community [5–8], yet so-called *joint inference* techniques have yet to become the standard approach to model composition. We hypothesize that this may be in part due to the lack of a flexible, general framework for constructing models for joint inference, with the ability to efficiently represent highly-structured models (such as models of syntactic structure). Additionally, we observe that *jointly annotated data* – a single

data set annotated for multiple tasks, sufficient for training all the models of a pipeline system – is scarce, and in most instances the outputs of intermediary models are never evaluated nor utilized other than to improve performance on a singular end task.

To this end we present a general approach for constructing **joint models**: models which unite more than one task-specific model, which we refer to as a **component model**, and where information is shared freely between them during inference. Unlike the pipeline approach, information can flow in both directions. In theory this allows a joint model to recover from errors made by components which appear early in the pipeline, preventing further errors from accumulating downstream.

We argue that factor graphs [9] are an ideal formalism for defining joint models. Factor graphs are a type of graphical model which are capable of representing many common NLP models, and provide many characteristics which are beneficial for joint modeling in NLP:

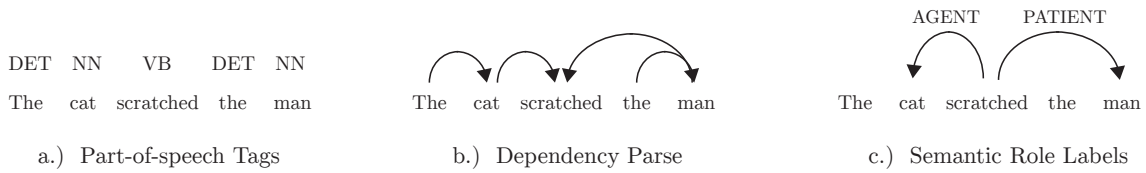
- In a factor graph there exists a natural semantics for modeling dependent components, and this provides a principled method for connecting two component models, using probability theory as the language of coordination. Thus factor graphs provide a means of constructing joint models from common single-task models.

- In a joint model, dependencies between component models often create cyclic dependencies (loops). In a factor graph, there are inference algorithms (belief propagation, [10]) which provide approximate solutions for arbitrary graph structures, even those with cyclic dependencies. Therefore, factor graphs have a mechanism for reasoning with joint models.

- One of the most common components of an NLP pipeline is a parser, which produces a syntactic analysis for each sentence. Due to the combinatorial nature of syntactic structure, incorporating a model of syntax can lead to difficult, if not intractable, inference. Following the recent work of [11], this problem can be circumvented in factor graphs through the use of specialized combinatorial factors.

Having addressed the problem of constructing and reasoning with joint models, we investigate new methods for training them. In order to train an NLP pipeline in the typical fully-supervised case, each component requires its own training data. For a joint model, the data requirements are significantly more demanding, as a single data set must be annotated with the labels for each task. Annotated data is already exceedingly scarce for most tasks in most languages, making this a very burdensome requirement.

To counter this we propose a paradigm shift away from the traditional motivation for joint inference, in which two or more tasks benefit mutually from fully-supervised joint training [6], and focus instead on the scenario where joint inference is applied solely towards improving performance of a single task. In this scenario, we can reduce the labeled data requirements of joint modeling by treating all supporting models (component models which are not the end task) as *latent*, or unobserved during training, requiring no training data of their own. For these models training supervision comes only indirectly, through the annotations of the end task, and is propagated through model dependencies during inference. In this scenario joint inference offers something new: a principled approach to learning with a combination of supervised and unsupervised components in a task-directed manner. We show, across a variety of NLP tasks, that this method provides performance comparable to fully-supervised training while requiring far less training data, and in some circumstances even improving over identical models using gold or parser-produced annotations.



a.) Part-of-speech Tags b.) Dependency Parse c.) Semantic Role Labels

FIGURE 1.1: Annotations for three common NLP tasks.

1.1 Overview

1.1.1 Model Combination: Maintaining Uncertainty

The natural language processing community works largely under the assumption that language can be effectively partitioned into largely independent modules – phonology, morphology, syntax, and semantics – each of which can be tackled independently. Each model assumes that all prerequisite steps have already been solved, and the input annotated accordingly. For example, a parser typically assumes the input has been part-of-speech tagged, and a semantic role labeling system typically assumes the input has been parsed.

But this strategy is only optimal when the tasks are largely independent of one another, i.e., that the solution to the latter problem does not influence the former. In the pipeline approach, errors made by earlier components force the following components to base their predictions on erroneous information, making it more likely for additional errors to accrue. If the models are dependent, information from later components could influence these early decisions. Methods of **joint decoding** aim to solve this problem. Starting from pre-trained models for individual tasks, joint decoding methods aim to find the optimal solution by considering information from all component models jointly. This may be the best solution for the entire pipeline (i.e., the best *global* solution), or it could be the optimal solution for only a subset of these components, conditioned on others.

In order to provide some context in which to discuss the problem of model combination, let us consider the task of semantic role labeling. Semantic role labeling (SRL) is the task of identifying basic semantic relationships in text. Given a sentence, the first step is to identify key verbs, known as *predicates*, and group them with their associated arguments. Each argument is given a label to describe the nature of its relationship with the predicate, known as its semantic role (i.e., agent, patient, instrument, etc..) For the example sentence "*The cat scratched the man*", the correct SRL analysis identifies *scratched* as a predicate, *cat* and *man* as its arguments, and labels them AGENT and PATIENT respectively. These annotations are shown in Fig.1.1c.

Current state-of-the-art SRL systems require a parsing component. In turn, because part-of-speech tags are required to train the parsing model (Fig.1.1a), the system will require yet another model to tag words with their parts-of-speech. We can define a set of models that reflect this this hierarchy:

$$\begin{aligned} Model_{Tags} &= P(Tags|Words, \Theta_{Tags}) \\ Model_{Tree} &= P(Tree|Tags, Words, \Theta_{Tree}) \\ Model_{SRL} &= P(SRL|Tree, Tags, Words, \Theta_{SRL}) \end{aligned} \tag{1.1}$$

where $Model_{Tags}$, $Model_{Tree}$, and $Model_{SRL}$ are models for part-of-speech tagging, parsing, and SRL, $Tags$, $Tree$, and SRL are the predicted variables, $\Theta_{Tags}, \Theta_{Tree}, \Theta_{SRL}$ are the corresponding model parameters, and $Words$ is a variable representing a sequence of words provided to each model as input. During training a parameter estimation method is used to optimize each model's parameters. In a supervised pipeline approach (Fig. 1.2) each model is trained independently, and requires its own annotated data set. During testing these models are utilized to annotate raw text with SRL annotations. Models for tagging, parsing and SRL will be defined in full detail in Chapters 2, 3 and 6, but for now let us consider how to *decode* (i.e., utilize the model's predictions to assign structure to raw text) using these models on an arbitrary sentence.

Our goal is to find the most probable SRL analysis, but, due to the hierarchical nature of the dependencies between these three models, finding this solution requires us to first consider the values of the $Tree$ and $Tags$ variables. However, in this scenario $Tree$ and Tag are *latent variables*: we have models specifying distributions over these variables, but their actual values are unknown. Here we present and contrast two approaches to dealing with these latent variables. The first decoding strategy truncates the latent variable distributions to a set of their most probable elements. A second decoding strategy relaxes this approximation, considering each possible analysis of latent structure and contributing its probability mass toward the SRL solutions that are most compatible with it.

N-Best Pipelines

Perhaps the most straightforward approach to decoding a set of dependent models is to process each model in turn, taking the output from each model and using it as input to the next. Here the model dependencies impose a natural order, beginning with the Tag prediction, proceeding to $Tree$ prediction, before finally reaching the goal, SRL prediction. This is aptly referred to as an ***n*-best pipeline**, where n is the number of analyses put forth by each component model (Figure 1.2). When the output from each model is its single best hypothesis, $n = 1$, this strategy can be summarized by the following equation:

$$\begin{aligned} GOAL_{1-best} = & \arg\max_{s \in \mathcal{S}} P(SRL = s | Tree = t, Tags = p, Words = \mathbf{w}, \Theta_{SRL}) \\ & \max_{t \in \mathcal{T}} P(Tree = t | Tags = p, Words = \mathbf{w}, \Theta_{Trees}) \\ & \max_{p \in \mathcal{P}} P(Tags = p | Words = \mathbf{w}, \Theta_{Tags}) \end{aligned} \tag{1.2}$$

where $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{P}$ are the sets of all possible SRL analyses, trees and tag sequences, respectively, and $GOAL_{1-best}$ is the single-best SRL analysis, conditioned on the single-best tree and tag sequence.

This is a very practical approach, but when viewing this decoding strategy as a maximization over each model in turn, it becomes clear that this is merely an approximation of a more exhaustive search. The approximation used here, known as the Viterbi approximation (in reference to the Viterbi decoding algorithm for hidden Markov models [12]), replaces each distribution with the most probable element of that distribution. This could also be viewed as three applications of *maximum a posteriori* (MAP) decoding, one for each of the three models, or as a case of n-best decoding, where $n = 1$.

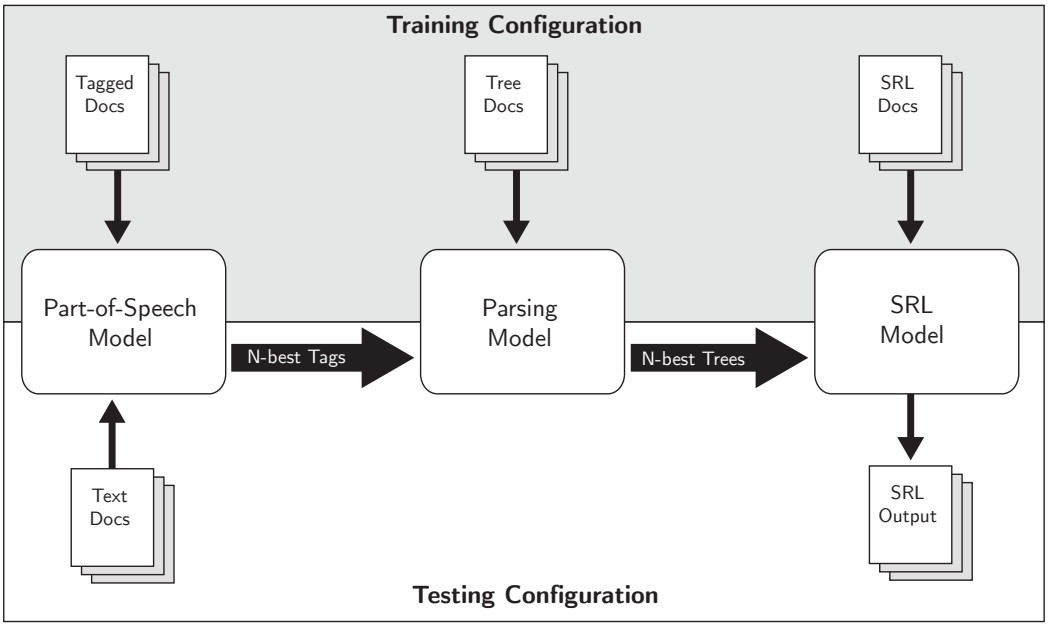


FIGURE 1.2: A pipeline approach to model combination. Individual models are trained independently from their own task-specific annotations, i.e. tagged documents are annotated with part-of-speech tags, tree documents with syntactic trees, and SRL documents with predicate-argument pairs. These data sets may come from difference sources (*top*). During testing, where the goal is to use the system to make end task predictions for new data, the only input to the system is raw text. Each successive model applies an additional layer of annotation, propagating it forward to be used as input to the next (*bottom*). This is often the single most probable analysis, but components can also propagate forward larger lists of analyses (known as an n-best list, of size n) in order to give later components some chance at recovering from a poor decision early in the pipeline.

While attractive from a practical perspective, pipeline approaches – both the Viterbi approximation and the more general n-best list decoding strategy – are severely handicapped when it comes to finding globally optimal solutions in NLP. In general, if the distribution of a preceding component is peaked (i.e., concentrated primarily on a small percentage of the analyses), the Viterbi approximation has a greater chance of selecting the correct hypothesis. If the distribution is flatter, the model is less certain, the most probable analysis inherently a poorer approximation to the full distribution.

To understand how this affects model combination in practice, consider the example outlined in Figure 1.3. Here we depict three syntactic trees and their probabilities as specified by $Model_{Tree}$, paired with the correct SRL analysis. Let us assume that a syntax tree provides evidence for an SRL analysis if there exists a corresponding syntactic dependency for each SRL dependency. When using the Viterbi approximation, ideally the most probable parse tree provides the maximum amount of evidence in support of the correct SRL analysis. However, here we find that the most probable parse tree, t_1 with probability 0.3, is incorrect, and only partially supports the correct SRL analysis. The most probable SRL analysis under the distribution $P(SRL|Tree = t_1, Tags = p, Words = \mathbf{w})$, conditioning on incorrect information, is also likely to be incorrect.

In this case, broadening the approximation to include all three parse trees would allow the

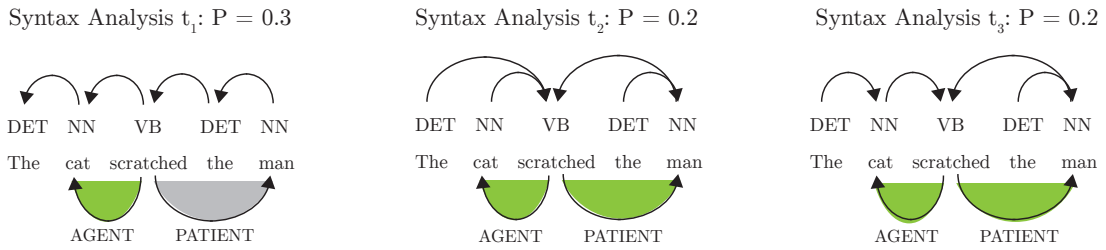


FIGURE 1.3: Failure of the Viterbi approximation. The top three trees of the parsing model's distribution are shown, along with the correct SRL analysis below. SRL dependencies that are supported by the syntactic analysis are shaded in green, and those which do not are shaded in grey. The solution chosen by the viterbi approximation (*leftmost tree*) does not maximally support the correct SRL analysis (shown below each tree), as it shares only one of the SRL solution's two arcs. In comparison, if the entire $Model_{Tree}$ distribution is utilized the correct SRL analysis would receive additional (and greater) support from trees t_2 and t_3.

model to include additional information from tree t_2 and t_3 that would be beneficial to SRL prediction. However, in practice this strategy is flawed: the space of possible parse trees for a sentence is often too large to tractably pass forward from one component to another. We now present an alternative method to decoding in which all syntactic analyses contribute in part to the decoding of the SRL model.

Marginalization-based Decoding

In a 1-best decoding strategy only a model's most probable analysis will affect the distributions of later models. All other information about the model distribution is discarded at each step. An alternative approach, which we will refer to as **marginalization-based decoding**, generalizes over the n-best list approach, and provides a method for including more of the information captured by the latent distributions. It does this by *marginalizing* (summing over) all possible values for the latent variables ($Tree$ and $Tags$):

$$\begin{aligned} GOAL_{Marg} = \arg\max_{s \in SRL} \sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}} & P(SRL = s | Tree = t, Tags = p, Words = \mathbf{w}, \Theta_{SRL}) \\ & P(Tree = t | Tags = p, Words = \mathbf{w}, \Theta_{Trees}) \\ & P(Tags = p | Words = \mathbf{w}, \Theta_{Tags}) \end{aligned} \tag{1.3}$$

where $GOAL_{Marg}$ is once again the single most probable SRL analysis, but this time conditioning on all possible values of latent trees and tag sequences, weighted by their probabilities. This makes it more robust to the effects of flatter model distributions.

Returning to the example analyzing the effect of parser decoding on the SRL model (Fig. 1.3), recall that the Viterbi decoding strategy selects the most probable parse tree, t_1, and $Model_{SRL}$ conditions upon it when finding the most probable SRL solution. Tree t_1 provides only partial support for the SRL analysis, with one of the two correct SRL dependencies having a corresponding syntactic dependency. Using the marginalization-based decoding strategy, summing over trees will provide nearly half ($P(t_2|Tags = p, Words = \mathbf{w}) + P(t_3|Tags = p, Words = \mathbf{w}) = 0.2 + 0.2 = 0.4$)

of the latent distribution's probability mass to work in favor of the correct SRL solution. The sum of support for the correct analysis now outweighs the support for the incorrect analysis, 0.4 to 0.3. For illustrative purposes our example here contains just three hypotheses. In practice there are thousands of possible trees for reasonably-sized sentences, and this mass must be distributed across many more trees, making the distribution inherently flatter and the Viterbi approach a poorer approximation of the full distribution.

We conclude with a clarification. The n-best and marginalization-based approach to decoding are useful for illustrating the difference between how model combination is typically done in practice, as an approximation, and the more general marginalization-based approach. However, the models we propose are joint. Joint models unite the advantages discussed here (maintaining uncertainty at each step) with bidirectional information flow. This allows the beliefs of the later models to influence other models which would appear earlier in the pipeline. For instance, bidirectional information flow allows the beliefs of the SRL model to influence the parsing model, during both decoding and training. Instead of taking a set of pre-trained models as input to decoding, a joint model provides just one (individual component models are not factored out).

1.1.2 Marginalization-based Approaches to Training

We have presented an initial set of motivations for preferring a marginalization-based or joint decoding strategy over alternatives like n-best lists or Viterbi decoding. By operating on full distributions marginalization-based decoding avoids some of the potential pitfalls of approximate solutions to the global problem. But if this is our ideal decoding strategy, then what is the corresponding ideal method for training? Here we describe an analog to training we refer to as a **marginalization-based training**, which extends to optimization the philosophy of working with full distributions.

Let us start by discussing the pipeline approach to training. During training in this scenario, there are no latent variables. In the case of our SRL example, the SRL model is given the true values for SRL, $Tree$, and $Tags$. The Tree model would observe the true analyses for $Tree$ and $Tags$, and so forth. These true analyses are derived not from the output of a previous model, as it is during testing, but are provided in annotated data. This decoupling allows the models to be trained independently of each other, as illustrated in Fig. 1.2 (top). Optimizing the set of models can then be done as follows:

$$\begin{aligned}
\Theta_{ML-SRL} &= \arg\max_{\Theta_{SRL}} \prod_{i=1}^{n} P(SRL_i | Tree_i, Tag_i, Words_i, \Theta_{SRL}) \\
\Theta_{ML-Tree} &= \arg\max_{\Theta_{Tree}} \prod_{i=1}^{n} P(Tree_i | Tag_i, Words_i, \Theta_{Tree}) \\
\Theta_{ML-Tags} &= \arg\max_{\Theta_{Tags}} \prod_{i=1}^{n} P(Tags_i | Words_i, \Theta_{Tags})
\end{aligned}$$

where i corresponds to the ith item from the training data, of which there are n in total. This type of optimization is known as **maximum likelihood estimation** (MLE). In MLE the goal is to

optimize the likelihood of the parameters given the data, $L(\Theta|x)$. This is equivalent to maximizing $P(x|\Theta)$, and thus the right hand side of each equation is a likelihood. Each instance of the training data is treated as independently generated, thus the likelihood of generating the entire data set is the product of all individual likelihoods.

There are some practical advantages to this approach. It is easier to find a separate data set for training each model than it is to find a single data set annotated for all three tasks, and it is also easier to train each model independently than to jointly optimize all models simultaneously. However, the disadvantage to training each model off an unrelated data set is that there will be some degree of domain or genre mismatch between them. [13] shows that a parser trained on a corpus of newswire performs notably worse when tested on a mixed-domain data set. A similar problem exists when training systems off of several task-specific data sets. For instance, both the SRL and parsing models utilize parse trees during training, each from their own respective data sets. During decoding, the trees used by the SRL model are obtained from the parsing model. If these trees are not representative of the trees the SRL model was trained on, the setting is identical to testing on out of domain data, and may be subject to similar performance loss.

In this thesis we propose a different strategy for training dependent models. Instead of training each model toward its own goal (i.e. a tagging model which attempts to best reproduce the gold tags from a corpus, a parsing model which attempts to reproduce the gold trees from another), we train all models toward one goal: improving performance on the end task. In the example case, the end task is SRL. Once we treat component models as parts of one larger, end-task directed model, there is no need to require task-specific annotations for each component model.

In theory this allows us to perform a complex end task like SRL, without requiring any training data for supporting tasks, like part-of-speech tagging or parsing. The resulting optimization on a joint model has the following form:

$$\Theta_{ML-Marg} = \underset{\Theta_{SRL},\Theta_{Tree},\Theta_{Tags}}{\arg\max} \prod_i \sum_{t\in\mathcal{T},p\in\mathcal{P}} P(SRL_i, Tree_i = t, Tags_i = p|Words_i, \Theta_{SRL}, \Theta_{Tree}, \Theta_{Tags}) \quad (1.4)$$

where SRL_i is the correct SRL structure for the ith training example, and we once again sum over the latent Tag and $Tree$ variables. But this is during training – if supervision for these models is not required, what constitutes a probable tree or tag sequence? What these models will learn to represent is something more abstract, with supervision coming only indirectly, through the end task annotations. The fact that this is a joint model is what allows the error signal, originating with end task annotations, to propagate out and influence these latent structures. Simultaneously, we provide general constraints on the kinds of latent structures that are permissible. For syntax, these constraints force the latent structure to be a directed graph or a tree. The learned latent structure will be one that optimizes Eq. 1.4 while adhering to these general structural constraints.

This discussion has provided a high level view of the approach pursued throughout this dissertation, leaving open the question of how we construct models, and how this method compares to other approaches. In the next section we introduce graphical models, a framework which we argue is convenient for constructing joint models, and performing joint inference.

1.2 Joint Inference in Graphical Models

We have previously described our marginalization-based approach to both model training and decoding. This goal requires information to be shared between component models, but we have not yet discussed *how* this communication is possible. The answer lies in **inference**, the process of drawing conclusions from data by using a statistical model. In this section we introduce graphical models, a powerful framework for representing and reasoning with probabilistic models. Here inference will be used to answer the question, for a variable x and a value q, "*Given values for certain variables, what is the probability that variable x has value q?*". The answer provides a key component, marginals, required for training these models when using a gradient-based optimization method.

1.2.1 Graphical Models

Graphical models are a family of formalisms that use graph theory to express probabilistic relationships, namely conditional independence assumptions, which are useful for defining efficient inference algorithms. Here we provide a brief overview of graphical models, before discussing them at depth in Section 2.2 (pg. 23).

A probabilistic model encodes a problem domain using a set of random variables, the value of each variable capturing the state of particular aspect of the world. In NLP, a random variable may represent a word's part-of-speech tag, a syntactic dependency between two words, or named entity over a span of words. A probabilistic model defines a distribution over these variables, specifying the probability for any possible assignment of values to the variables in the model.

But a distribution over a set of n variables, each with m values, yields $O(m^n)$ configurations, growing exponentially with the number of variables in the model. The models presented in this dissertation will often contain thousands, if not tens of thousands of variables, and thus working with the distribution directly would be intractable.

Graphical models are a type of probabilistic model which use graph structure to represent a factorization of this distribution. Variables correspond to nodes in the graph, and the graph's edges encode dependencies between variables. More precisely, the *absence* of edges in the graph specifies the conditional independence properties of the model (this is described in detail in Section 2.2 on pg. 23). This allows graphical models to factor the distribution as the product of many simpler functions, each of which deals with a much smaller subset of variables. The sparser the connections are in the graph, the more compactly the distribution can be represented using a graphical model.

This is the key to tractable representations of probabilistic models, but it is also a convenient factorization: a sparsely connected graph provides a more compact representation of the distribution which helps to support efficient inference, but it simultaneously complements the modeler's goal of applying their expert knowledge to define the problem in a simple, intuitive manner.

1.2.2 Factor Graphs

In this dissertation we focus on a type of graphical model known as a **factor graph**. Factor graphs make the dependencies between variables more explicit using a second set of nodes known as

factors. A factor scores any assignment of values to its neighboring variables. A factor which neighbors just a single Boolean variable, whose values range over {true, false}, specifies a score for the variable being true, and a score for the variable being false. The model's score for a configuration of variables is simply the product of all the corresponding factor scores for that configuration.

The factorization provided by a graphical model only yields a significant reduction in complexity when edges in the graph are sparse. When graphs are densely connected, the factorization is not as efficient and inference can become intractable. In NLP, one such case is that of syntax trees. Two leading syntactic formalisms, phrase structure and dependency structure, can be represented in a factor graph using $O(n^2)$ variables. However, to ensure that all configurations of variables correspond only to valid trees, a large (sometimes exponential) number of factors is required.

Factor graphs provide a solution to this problem. Instead of constraining variable configurations using many factors over small sets of variables, a single globally-connected special-purpose *combinatorial* factor can enforce this constraint. Such a factor encapsulates a function, different from standard inference procedure, to update the values of all syntactic variables simultaneously. This can be done in a much more efficient manner by leveraging dynamic programming algorithms developed earlier in the parsing literature, requiring only $O(n^3)$ time. We make use of this strategy throughout this thesis to efficiently represent structured models, primarily syntactic models, and present a novel factor for efficiently representing phrase structure syntax (Sec. 3.1.2).

Special-purpose factors are also useful in constructing joint models. We define Boolean logic factors to coordinate between corresponding variables from separate component models. For instance, if an SRL dependency commonly corresponds to a syntactic dependency, we can model a correlation between the two corresponding variables by connecting them with a Boolean logic factor. These factors contribute a score when the Boolean logic is violated. This score can be derived from a set of feature weights, so the model can learn how strongly these variables correlate. While there may be many such factors used in modeling the connections between two models, each connecting factor is associated with different features and will contribute a context-sensitive score to the model.

1.2.3 Inference

During training, maximum likelihood estimation is used to find the parameters for which the model most accurately predicts the data. In order to efficiently search the vast space of all real-valued parameters, we use gradient-based optimization, which in turn requires computing the gradient of the model parameters. This gradient is a difference between expectations (see pg. 48, Algorithm 2, line 11) – one expectation taken deterministically from the labeled training data, a second expectation calculable via *inference*. Inference in a factor graph produces factor **marginals**, sometimes called beliefs. A factor's marginal defines a distribution over the values of its neighboring variables, and is calculated by summing over them, the same marginalization process discussed earlier in Section 1.1.2 (pg. 7).

It follows that in order to compute the marginal for a single factor, other factors must be involved in the computation. To compute the marginal for a second factor, nearly all of the same factors will be used in an identical manner. A more attractive solution is offered by message passing algorithms, a family of inference algorithms for graphical models that operate by distributing

and collecting information across the graph. The key insight of message passing inference is that these otherwise redundant calculations can be cached locally, allowing for exact inference in just two passes through chain or tree-shaped graphs. For general graphs inference is intractable, but it can be approximated using loopy belief propagation, an inference method used repeatedly in this thesis.

Wrapping inference inside a gradient-based learning scheme is a common approach to supervised training, but to realize the goal of training in the presence of latent variables, as described in Eq. 1.4, we only need to slightly modify this procedure. Instead of acquiring the first expectation entirely from labeled training data, this too can be calculated (or approximated, in cyclic graphs) via inference. For this calculation the subset of observed variables are set to their true values prior to inference. Inference and gradient calculations can then be performed normally, with the resulting update reflecting a marginalization over latent variables whose true values were provided by the training data. By utilizing this two-step inference procedure inside gradient-based optimization we are able to train in the presence of latent variables, optimizing all model parameters toward the goal of improving end task performance.

1.3 A Comparison of Approaches to Joint NLP

The NLP community has made great strides in improving the performance of NLP models on single tasks, but the real-world success of these models will hinge on how well such models work together in the context of an end-to-end NLP system. In response to this need, model combination has received a great deal of attention in recent years. We divide these contributions into two groups along the conceptual axis of whether the models are trained independently and combined only during testing (joint decoding), or if they are treated jointly throughout (joint training). Common to all combination methods is the need to communicate or negotiate between model components. However, how models exchange information is unique to each approach.

1.3.1 Joint Decoding

Perhaps the simplest method of combining models is to train them independently and aggregate their output in a manner that solves the global problem. We refer to this class of approaches as *joint decoding* methods.

Comparisons to Pipeline Decoding with n-best Lists

As discussed previously, one of the most straightforward methods of model combination is the n-best pipeline, in which each model is processed in sequence, the output of one model – a set containing the n most probable analyses – becoming the input to the next. In Section 1.1.1 we provided a high-level overview of why an marginalization-based decoding scheme may be preferable alternative, but having now introduced the machinery by which we implement and train our models, we discuss the remaining advantages of our approach: (1) unidirectional vs. multidirectional information flow, and (2) passing complete analyses vs. beliefs over *parts* of analyses.

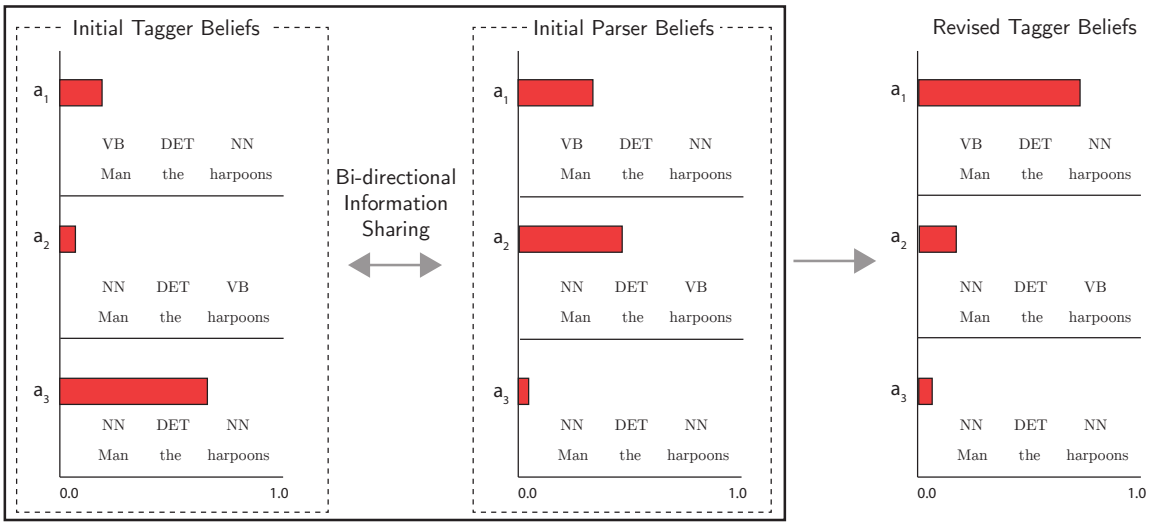


FIGURE 1.4: Bidirectional information flow via belief propagation. The figure depicts three analyses over tag sequences (a_1, a_2, and a_3) as scored by models prior to global inference (*left and middle*) and after (*right*). In a traditional NLP pipeline information flows in only one direction: forward. When a component suggests a poor analysis early in the pipeline, as is the case when taking the 1-best analysis of the tagger (*left*), downstream models (the parser, *middle*) are more likely to make poorer predictions, and errors multiply. A joint model (depicted within the bounding box) can circumvent confident, erroneous modeling components by allowing information to flow backwards in the pipeline, and influence earlier models. The resulting tagger beliefs favor the analysis most compatible with both models' beliefs (*right*).

Unidirectional vs. Bidirectional Information Flow

Recall that the pipeline approach fails when the optimal local solution falls outside of the n-best list. Passing forward a poor set of analyses provides a faulty foundation from which to make the next stage of predictions, leading to increasingly poorer predictions as information moves through the pipeline. One solution to this problem is to increase the size of n, making it more likely that the optimal local solution is contained within the larger list. In practice this strategy is rarely helpful, as even increasing the list capacity to hold thousands of elements rarely makes a significant difference when the list items are heavily structure objects with an enormous number of possible structures, like trees [14]. The larger problem is that information in a pipeline only flows in one direction.

This is a handicap not shared by other approaches. For instance, a joint model can circumvent these issues because each component model's contribution accounts for only part of the global model's final solution. In this sense, joint inference acts as a discourse in which all models participate. Like a jury slowly nudging a dissenter toward a unified consensus, inference has the effect of mitigating the weaker sources of conflicting information.

Let us consider how a joint model might recover from a poor local analysis in the context of a joint part-of-speech tagging and dependency parsing model. The sentence "*Man the harpoons.*" is lexically ambiguous: both *man* and *harpoons* can be either nouns or verbs. Consider a simple tagging model, a unigram model, which contains no dependencies between adjacent tags. Each word receives its most frequent tag. The word "*man*" is not frequently a verb, so the model would incorrectly assign the most probability to analyses where it is labeled a noun (Fig. 1.4, *left*).

However, a parser scores trees based on a separate set of criteria. For instance, a parser may assign a very low score to a tree that contains no verb, as verbs play an important syntactic role and trees without them are uncommon (Fig. 1.4, *middle*). Thus an analysis that was initially considered very probably under the tagger becomes very unlikely when placed in the larger context. If these components were connected in a 1-best pipeline, the system would already be committed to the incorrect tag sequence and leave successive components little chance to recover. In the Fig. 1.4 example, the resulting parse, a_3, would be highly unlikely.

A joint model circumvents this by allowing information to flow in both directions. During inference, the beliefs that a factor has over its neighbors propagate throughout the graph, influencing the beliefs of even distantly connected factors. In a joint model of tagging and parsing, the model's beliefs over syntactic variables can help guide its beliefs over part-of-speech variables, and vice versa. In essence, joint inference entirely discards the notion of a serialized order that was inherent in the pipeline. When inference ends and the model beliefs have converged to a globally-informed optimum, the resulting solution may differ from either the tagger or parser's single most probable independent analysis (Fig. 1.4, *right*), but will reflect the pooled information of all component models.

Units of Exchange

A second point of divergence is scope of the information that is passed between models. In an n-best pipeline the model components pass forward lists of complete structures. If the model is a tagger, a list of complete tag sequences is pushed forward. If the model is a parser, a list of trees. In doing so, n-best lists force the system to multiply out ambiguities.

A joint factor graph model can differ in this respect, as it can coordinate component models on a much finer level of granularity. In the method we propose, it is not models that share information, as much as it *parts of models* that share information. Assume there is a model of syntax and a model of semantic role labeling. Coordination between these models occurs at the level of corresponding variables. If we believe that a syntactic span over a pair of indices is correlated with an SRL relationship over those indices, the variables representing these structures can be connected via a factor.

During inference, the information shared between component models takes the form of variable-to-variable interactions. A variable from one model, the source, sends a message to the coordinating factor. This message represents the variable's belief about its values in light of all the evidence it has received from all of its other neighbors. The coordinating factor receives this message and multiplies in its own score before relaying it to the destination variable in the opposing model. This score either reflects the modeler's intuition regarding the nature of the coordinated variables, and we consider the special case where this intuition is captured by a Boolean logic relationship, or it could alternatively be learned entirely from data. The process of coordinating between component models in factor graphs is described in Section 2.4.3 (pg. 43).

While an n-best list might also pass forward additional information to better describe the model's distribution, fine-grained information is still lost. For example, in a pipeline a parser may push forward a list of trees *and their associated probabilities*, but the weights of substructures within each tree are lost. This would be similar to classifier stacking, where each model outputs a posterior distribution instead of a finite subset of possible analyses. If knowledge of a particular

substructure is useful to the end task prediction, much of this information may be lost by the n-best approximation.

For instance, in SRL, in determining whether a semantic dependency exists between two words it is useful to know if there is a corresponding syntactic dependency between them. When considering the model's beliefs over all trees, it may be very likely that this dependency exists in the tree, but this is no guarantee that it will be present in the trees selected for the n-best list. This is not true of our joint approach, where the corresponding syntax and SRL variables can be directly connected in the graph. During inference, the information sent from the syntactic variable to the SRL variable represents precisely what is desired: the syntax model's total belief that the syntactic dependency is present.

This also affects the scalability of these models. For a combinatorial structure, like a syntax tree, the number of possible trees for a given sentence length can be exceedingly large. This is often an intractable number to pass forward, and motivates approximations where only a subset of the total number of trees, the n-best, are exchanged. It's the fine-grained nature of connecting model variables rather than models, that allows our approach to remain tractable without such approximations[1]

Other Approaches to Joint Decoding

Our approach is not the first method to solve the joint decoding problem. In this section we discuss two alternatives: Integer Linear Programming, and Dual Decomposition.

Integer Linear Programming

Integer linear programming (ILP) is a form of mathematical optimization which can be used to perform global inference across a set of component models. ILP maximizes a linear objective function, and requires each model (and the relationships between them) be expressed as a set of linear constraints. ILP is NP-hard but tractable for many problems, and when solved ILP provides an exact solution.

When a model is intractable one strategy to circumventing computational inefficiency is to decompose the problem into many subproblems which may be easier to solve individually. ILP provides a method for finding the global solution from a set of solvers, each solving part of the problem. This strategy has been applied to dependency parsing [15], SRL [16], and natural language generation [17], among others. When these subproblems are component models, ILP becomes a method of model combination, and an alternative to the pipeline approach. [18] show that a global decoding strategy with ILP outperforms a pipeline approach when applied to named-entity recognition and relation extraction. In SRL, a task we pursue in Chapter 6, many joint parsing and ILP systems have been proposed and are included in some of the state-of-the-art systems for this task [19, 20].

Not all problems are easily expressible in terms of linear constraints. [15] notes that non-projective dependency parsing requires an exponential number of constraints, making ILP intractable. A similar problem occurs when expressing non-projective dependency parsing with

[1] Note that we do at times employ approximate inference techniques, but always refrain from the hard decisions found in n-best lists.

factor graphs. [15] avoid this problem by incrementally introducing constraints into the model, attempting to limit the number of necessary constraints to a tractable size. In our own approach with factor graphs, we adopt the method of [11] using combinatorial factors. This method is discussed further in Section 2.4 (pg. 36).

Dual Decomposition

A second and related approach is dual decomposition (DD, also known more generally as *lagrangian relaxation*[2]). DD has the same theoretical underpinnings as ILP, and is another technique borrowed from the mathematical optimization literature. In DD, models are coerced into agreement by adjusting Lagrange multipliers, iteratively tuning these parameters until the solutions of individual components models are consistent with one another. If the exact solution is found, it also returns a guarantee of optimality.

DD has been applied to a large number of joint tasks ranging from combining dependency parsing and constituency parsing [21], language models with syntactic machine translation [22], with phrase-based translation models [23], CCG super tagging and parsing [24], and MT alignment models [25]. As a decoding strategy, DD is an attractive framework. It's fundamental shortcoming as it pertains to our task-directed learning framework is that it is currently unclear how to apply it during training, in the presence of latent variables.

There are numerous differences between these alternative joint decoding method and our proposed method. Both ILP and DD have the benefit of exactness: ILP returns an exact solution if computation terminates, and DD has a high rate of discovering and proving exact solutions in a number of NLP tasks [21]. But exactness comes at a cost. ILP is NP-Hard and while the solution is guaranteed to be exact, the ILP solver may not return one in a practical amount of time. DD is also somewhat limited, requiring that subproblems have efficient solutions, and non-overlapping features (such as first order logic features and features that violate Markov assumptions) . Overcoming some of these difficulties has been addressed in previous work [26], but remains an open area of investigation.

In contrast, while loopy belief propagation only finds approximate solutions, this approximation compares favorably to DD in practice [24]. Graphical model representations have already been developed for many NLP problems, alleviating the need to find novel ways of representing the model in a particular framework. Models can be overlapping, and dependencies between models can be specified arbitrarily between any sets of variables.

However, the most distinguishing and most pertinent difference between these approaches and the joint factor graph approach is that our method can extend beyond test time, coupling models during training as well. Joint inference during training is a crucial aspect of the framework, ultimately allowing us to achieve our goal of optimizing latent structure towards a specific end task. Neither of these differences is entirely unexplored, but both come up only rarely in the literature. DD has been used during training previously [**?**], since any model that can be decoded can also have its parameters updated based upon these decoded structures (as an example, we perform such updates in Section 3.1.3, pg. 85, in order to learn sets of grammatical rules on top of our factor graph model's predictions). But this is not common and differs strongly from the message-passing

[2]Dual decomposition refers to a special case of Lagrangian relaxation methods in which two or more combinatorial algorithms are used.

view of joint inference by making hard decisions, where the message we propose works entirely with expectations. Additionally, DD has been used in the presence of latent variables (see [**?**] for learning latent variable CRFs using max-margin criteria), but these methods have not been applied to highly structured latent variables or NLP models.

1.3.2 Joint Training

In Section 2.5.2 (pg. 47) we presented a simple approach to decoding a set of dependent models, the n-best pipeline, and discussed scenarios where it is unlikely to find the global maximum. Joint decoding methods like ILP and DD circumvent these shortcomings, but still assume the models have already been trained, often independently from one another and using a separate data set for each. In this scenario there may be domain mismatch between data sets, and models which appear earlier in the pipeline ignore useful information from later models. In contrast, joint training optimizes all models toward a single objective.

In this section we provide an overview of two methods of joint training, and contrast them with our own.

Combining Predictions Instead of Models

One way in which to perform joint training is not to combine *models* per se, but rather to combine the structures they predict. The problem then shifts from learning how to coordinate model components, to how to train the more complicated model. Consider the sentence *John saw Mary*. A possible parse tree for this sentence is:

(S (NP (NN JOHN)) (VP (VB SAW) (NN MARY))).

And a corresponding named-entity span to indicate that John and Mary are named entities:

(PER John) saw (PER Mary)

If one wished to pursue a joint approach to named-entity recognition and parsing, one needs only to modify the grammar used by the parser to incorporate the named-entity information. The following tree reflects the predictions of both tasks:

(S (NP (NN-PER JOHN)) (VP (VB SAW) (NN-PER MARY))).

Using standard parsing methods with this grammar produce what can be thought of as a joint analysis, with one problem cleverly nested inside of another [6]. But this is precisely one of the limitations of the approach: only a problem that can be conveniently described inside of another can be pursued in this manner. For NER this is an intuitive method of coupling - a named entity *is* a specialized noun phrase, but trying to combine two problems where dependent structures are likely to cross would not be possible with this method. For instance, prosodic information is commonly characterized as a series of breaks, or pauses, in a spoken utterance. While the prosody and syntax are correlated, [27] notes that prosodic breaks do not always correspond to syntactic boundaries. A parse tree's syntactic spans cannot cross, and therefore there is no way to jointly model prosody and syntax in this manner.

A second disadvantage is that when combining a task with a parser, as described above, the grammar would grow multiplicatively, as a copy of each nonterminal must be made for each of the named entity labels. While many cubic time parsing algorithms exist, n as it pertains to parsing is often very small (< 40), and in practice the complexity of parsing is largely determined by the size of the grammar. Thus, from an efficiency standpoint, multiplicative increases to the size of the grammar affect the parsing algorithm at an especially susceptible point. This issue is explored in greater detail in Sec. 4.2.

A third disadvantage of this approach is one common to many joint models: data. In order to train the joint syntax and NER model there must be a data set jointly annotated with the structures for both tasks, as joint training hinges on learning the correlation between two tasks. In proportion to the amount of unstructured data in the world, annotated data is scarce, and jointly annotated data is still unobtainable in many languages for many pairs of tasks. [28] show how a joint model can benefit from additional singly-annotated data. In addition to the joint model, a standalone NER model is trained on data containing only NER annotations, and a standalone parser is trained on data containing only syntax trees. During training, a prior can be used to promote agreement between the standalone and joint models. This may reduce the amount of jointly annotated data required for comparable performance, but it does not remove the need for it.

Learning over Constrained Latent Representations

A second approach to joint training, and perhaps the work most closely resembling our own, is the Learning over Constrained Latent Representations (LCLR) framework of [29]. Both LCLR and our own approach share similar goals: both aim to learn the optimal latent structure for a particular end task, where only end task annotations are observed. This removes the need for jointly annotated data. However LCLR does not define latent structure to the extent that it can be thought of as a series of intermediate models – there is only input, a latent representation, and an output. As such there are no analogous supervised correlates for this algorithm (like the pipeline approach). Our method is more general and could in theory couple more than two models, yet for the scope of this thesis the setup is identical to our own.

In LCLR the end task annotations are used to guide the induction of the latent structure. Though LCLR is an attempt to improve upon the "two stage" learning method, in which a supporting task is decoded, features are extracted from these structures, and the end task model is trained to optimize these features (essentially a two-component n-best pipeline), it can also be thought of in two stages. In the first stage the algorithm proposes a set of "feasible" latent representations from a given example and its end task labels. These are problem-specific, and like the systems we present these could theoretically be syntax. Previous work has focused primarily on alignments [29] and chunking [30]. Current proposals for using LCLR rely on ILP for this step, requiring that these structures are capable of being phrased as a set of linear constraints.

Features are used to score intermediary structures, and the ultimate end task predictions. Learning in LCLR attempts to optimize the parameter weights that provide high scores to intermediary representations, while minimizing loss on the end task. Each latent structure decomposes, as in our framework, into many substructures. A sentence-level alignment decomposes into an alignment between words, a parse tree (in dependency syntax) decomposes into many pairwise relationships, and so forth. An optimal latent structure is one whose substructures correspond most directly to the correct substructures of the end task (each substructure scored via a function outputting $\{0, 1\}$

with respect to the end task), while providing the greatest benefit to classification accuracy. Previously SVMs have been used for learning, and thus the best latent structure is the one which is the most capable of pushing otherwise incorrect end task predictions across the decision boundary.

A key difference between LCLR and our marginalization-based approach to joint training is that LCLR is more explicit in the representation of latent structure. Like n-best lists, LCLR proposes sets of complete latent structures during the first step. In the case of structures with combinatorial characteristics, and therefore the potential for an intractably large number of possible latent structures, this step requires heuristic pruning to carefully tune which structures are proposed in this restricted set. And importantly, the features used for the end task classification are constructed solely from the single best latent structure. Thus the most significant difference between these methods may lie in how latent structure contributes to the end task predictions. In our method the set of all possible latent structures are marginalized over, whereas in LCLR this summation is a maximization.

Other Approaches: Task-driven Grammar Induction

A related line of research also seeks to use end task annotations to guide the induction of a task-specific parser. The major distinction between this and the aforementioned work is the focus on evaluating the latent structure and not the downstream task. [31] improved unsupervised dependency parsing accuracy by using HTML markup, viewed as an easily-obtainable noisy semantic annotation. [32] present a generative model which utilizes a small corpus annotated with semantic relations (of the sort found in the relation extraction task, Chapter 5) to train a joint model of syntax and relation extraction, with the aim of producing a better parser. The model puts several constraints on the relationship between syntax and semantics, noting that the different types of semantic relations are expressed syntactically in different ways, that semantic arguments of a word often form a syntactic constituent span, and that function words do not define semantic relations. Inference is done via Gibbs sampling, first sampling the latent tree, then the semantic structure. This method outperforms other leading grammar induction approaches, including the HTML-driven training of [31], and the previous state-of-the-art in purely unsupervised approaches [33].

[34], extend the augmented loss framework of [35] to include training in the absence of supporting (syntactic) training data. They posit that parsing errors dealing with coordination (*Google and MSR*) or apposition (*tech giant Google*) are due to a lack of real-world knowledge in the parser, and use facts from Freebase, a publicly available knowledge base, during training. Augmented loss training jointly optimizes two loss functions: an extrinsic loss function for the downstream task (here, relation extraction) and an intrinsic loss function for the supporting task (parsing). However, because parse trees are not available during training, the intrinsic loss function is replaced with a sampling function which, for a given candidate tree and the gold standard relation structure, is guaranteed to return a tree no worse than the candidate. Within this sampling function, a classifier, trained from jointly annotated data, is used to correct candidate trees. It is similar to LCLR, with both approaches proposing a set of candidate latent structures, and then selecting the single latent structure which best supports the downstream classification. Experiments on parse accuracy show significant improvement in the classification of coordination and apposition phrases.

These approaches hinge on finding the single parse tree which is most beneficial to the end task, training jointly but attempting to improve the performance of 1-best decoding in a task-specific

way. Our approach differs in that we marginalize over latent syntax trees, learning the *distribution* over latent trees which best supports the end task. The technical approach is similar to previous work in the computer vision [**?**], where missing values are marginalized over when performing scene interpretation in the presence of partially-labeled data (discussed further in Section 2.5.2, pg. 47).

1.4 Contributions of Dissertation

This thesis contains several contributions to our understanding of natural language processing and machine learning. First, we introduce new combinatorial factors for representing unlabeled phrase structure grammar and semi-Markov constraints analogous to previous work [11]. We utilize the former to construct the first factor graph representation of constituent parsing, show that it performs comparably by some metrics to state-of-the-art parsers, and extend it to incorporate grammatical rule learning. We then utilize Boolean soft factors to construct joint models. Similar factors have been used previously in translation [172], but not to coordinate across different tasks. Using this approach we construct factor graph models for named entity recognition, relation extraction, and semantic role labeling, all of which are novel (and the named entity model is notable as it is span-based – a significant departure from the typical linear-chain and semi-Markov models used for the task). Here we show that joint inference with syntax frequently outperforms syntactically-unaware baselines.

The main contribution of the thesis is how we train these models in the presence of latent variables, marginalizing over the variables of syntactic models for which we do not have labeled data. As noted before, this technique is not novel [**?**], but its application to NLP (namely syntax), and its use in conjunction with strongly constrained latent structures, is new. We show that across a diverse set of NLP tasks that this architecture and semi-supervised training approach consistently performs comparably to fully-supervised joint models, while requiring no syntactic training data. This allows for the development of statistical NLP models for the vast majority of languages and domains where syntactic training data is often not available.

1.5 Outline of Dissertation

- **Background** (Chapter 2), formally introduces factor graphs, the graphical model formalism used exclusively throughout the dissertation, discussing common factor graph architectures and inference techniques. We show how factors can be used to encapsulate algorithms that would be inefficient or intractable to represent explicitly in the graph structure, providing a method for representing syntactic formalisms and useful combinatorial logic. Gradient-based methods for training are described, and marginalization-based training is shown in greater detail.

- **Syntactic Representations** (Chapter 3), introduces the model structure used for representing constituency and dependency syntax in factor graph models. These representations function throughout the dissertation as latent structure for supporting a variety of end task predictions. We review the variables and factors used for representing first and second order dependency

parsing (a contribution of previous work [11]), and introduce those used for labeled and unlabeled constituent parsing. We discuss the necessary related issues, like binarization, and show unique extensions, like how grammatical rules may be represented as factors, and how these rule factors can be incrementally applied to the model to fix erroneous predictions. We demonstrate the effectiveness of these parsing models by comparing their performance to standard state-of-the-art parsers in a fully supervised parsing task.

- **Named-Entity Recognition** (Chapter 4), focus shifts to joint models, and we present a joint model coupling constituency syntax to a model named-entity recognition. We again utilize a combinatorial factor, this time for applying a semi-CRF constraint to the named-entity model, which allows the named-entity variables to represent spans instead of boundaries. With this design there is a one-to-one correspondence between syntactic variables and named-entity variables, which can be connected and coordinated via factors expressing logical relationships. We evaluate on the OntoNotes data set, showing state-of-the-art performance and an improvement even over other joint approaches to this task.

- **Relation Extraction** (Chapter 5), we develop a model for joint syntax (both dependency and constituency formalisms) and couple it with simple relation prediction and labeling. We compare the performance of training with parser-provided tree annotations (from state-of-the-art parsers) to our marginalization-based training method, and to a baseline consisting of only relation extraction predictions.

- **Semantic Role Labeling** (Chapter 6), we extend upon the relation extraction model of Chapter 5 to the similar, but more structured predictions of an SRL task. We perform a similar series of evaluations, but also compare against models observing gold-standard parse trees and find that our approach performs comparably, and sometimes even improves upon it by learning a more appropriate latent representation for the task.

- **Conclusion** (Chapter 7), summarizes the modeling and algorithmic contributions of this thesis and describes its implications for future work.

2

Factor Graphs, Belief Propagation, and Combinatorial Constraints

In this dissertation we present a framework for solving NLP problems using joint inference, with special attention paid to joint models in which one component model is a representation of syntax (either phrase structure or dependency structure), the other is a representation of the end task, and the two models are coordinated via soft Boolean logic constraints. We also describe how to train these models in the absence of syntactic annotation, inducing a task-specific syntactic representation to best support the model's predictions on the end task. We integrate several different techniques from machine learning to achieve this goal.

This chapter provides a discussion of the requisite material necessary for understanding the joint models, inference, and training methods used throughout the remainder of the dissertation.

- Section 2.2 introduces graphical models, a family of models which unify probability and graph theory to compactly represent distributions over larger numbers of variables. Here we describe factor graphs, our formalism of choice. We also review common factor graph models used for NLP tasks, and detail how these models can be extended to contain rich dependencies in arbitrary graph structures.

- Section 2.3 provides a discussion of belief propagation inference, a method for reasoning in graphical models. We discuss the requirements for exact inference, loopy belief propagation for approximate inference in cyclic graphs, and the effects of order on convergence.

- Section 2.4 provides a detailed analysis of combinatorial factors: specialized factors for efficiently representing structured constraints, like those found in syntax. We also provide an inventory of all special-purpose factors used throughout the remainder of the dissertation.

- Section 2.5 describes parameter estimation using stochastic gradient descent. It illustrates how training with latent variables can be naturally accommodated by the SGD algorithm,

provided the model supports the inference strategies necessary for computing the gradient (feature expectations). We also discuss more practical issues in optimization, like regularization.

- Section 2.6 is an application of the theory discussed in this chapter, and the conclusion of the running example used throughout. Here we present a joint model of parsing and tagging, which incorporates long-distance dependencies to supplement the local sequential dependencies typically used in tagging models.

Many of these concepts may be familiar to a machine learning researcher. We recommend that such a reader review Section 2.4, as this material is unique to our approach and will be used repeatedly throughout the dissertation.

2.1 A Running Example

Before we delve into the technical details which constitute this chapter, let us consider an example application of our proposed marginalization-based training method. Here we motivate a joint model of part-of-speech tagging and syntax, treating the syntax as latent structure and utilizing it to capture long-range dependencies that would be obscured in a purely sequential tagging model.

The problem of assigning labels to an ordered sequence of observations, otherwise known as *tagging*, is an important task in many fields, including bioinformatics [36], speech processing [37], and our current topic, computational linguistics [38]. Even information extraction tasks have been phrased as tagging problems [39]. Consider the task of part-of-speech tagging, where we observe a sequence of words $W = \{w\}_1^N$, and must predict the most likely part-of-speech tag (or word class) sequence, $T = \{t\}_1^N$:

pos-tags:	D	N	V	D	N
words:	The	cat	scratches	the	man

The pervasive ambiguity of natural language can make this a challenging task. For instance, the word *scratches* can be either a noun (N, as in "*He was covered in scratches.*") or a verb (V, as shown above). Humans accomplish this disambiguation through our knowledge of syntax and semantics, and sometimes with the help of additional linguistic cues. In order to tag a word without comparable deep linguistic knowledge, computational models must rely on the word's local context within the sentence.

A typical part-of-speech tagger models sequential dependencies. For instance, a hidden Markov model (HMM) conditions the choice of tag t_i on the previous tag, t_{i-1}. With respect to other dependencies that could be modeled (for instance, a dependency between t_i and t_0 or between t_i and t_N), these dependencies are very local, and yet using only local dependencies such models achieve greater than 95% accuracy on standard English data sets [40]. The discriminative analogs of HMMs, linear chain conditional random fields (CRFs), can attain over 97% accuracy [41], rivaling human performance and convincing many that supervised part-of-speech tagging is a solved problem [42].

However, long-range dependencies may help improve tagging performance. Consider Latin, where words in a syntactic relationship often agree in many morphological attributes. Here morphological tagging would benefit from knowledge of the sentence's syntactic structure [7]. This may seem antichronististic, as tagging is often pursued as a prerequisite step for parsing, but if syntax is treated as a *latent* structure, this structure can be optimized to best suit the tagging task without requiring any training data of its own.

In this chapter we introduce a joint model of parsing and tagging as a pedagogical example, using it to explain how we can define joint models, perform inference (even in cyclic factor graph models), and utilize marginalization during training to induce latent structures specific to the end task.

2.2 Probabilistic Graphical Models

Natural language is full of ambiguity. Words with identical sounds or identical spellings may have different parts of speech and entirely different meanings. A sequence of words may have many syntactic analyses, and yet each may be equally grammatical. Even identical sentences, with identical syntactic analyses, may mean different things in different (pragmatic) contexts. How can we model natural language phenomena when uncertainty is so pervasive, touching nearly all aspects of language?

Graphical models offer a solution. Graphical models are a family of models, encompassing Bayesian networks and Markov random fields, which couple probability and graph theory. As a probabilistic model, a graphical model is well suited to capturing the uncertainty that abounds in language. By relying on graph theory, graphical models are capable of compactly representing models with rich sets of dependencies. It is our hypothesis that when placed in an appropriate context – the context offered by modeling NLP tasks jointly – a better global solution can be found. Efforts to model this richer context benefit from the compact representation, and for these reasons we choose graphical models as the modeling formalism of choice throughout this dissertation. We now discuss what it means to be a probabilistic model.

A *probabilistic model* encodes a problem in terms of *random variables*, where each variable corresponds to one piece of information from the world. In NLP, a random variable may represent a part-of-speech tag, the presence of a syntactic dependency between two words, the location of a morpheme boundary, etc. Variables can be continuous and real-valued, or categorical and discrete. In this dissertation we will be concerned only with discrete-valued variables, thus each variable has a finite number of distinct values (a *multinomial variable*). If a multinomial variable has just two values, true or false, it is a *Boolean variable*.

Following the excellent presentation of graphical models in [43], we define a set of variables V, and make an important distinction between observed variables X, and predicted variables Y. We introduce the set of latent variables Z, such that $X \cup Y \cup Z = V$. The values of observed variables are given to the model, both during training and testing. Determining the values of predicted variables is the purpose of the model, thus their values are provided during training but not during testing. Hidden variables are neither observed nor predicted – their values must be inferred. Members of these sets are denoted by lowercase variables, i.e., $x \in X$, $y \in Y$, and $z \in Z$, often indexed with subscripts as $\{x_0, x_1, ..., x_n\} = X$. The subset $A \subseteq X$ is denoted as x_A. Let **x** denote an assignment of values to the variables in X. We denote a **configuration** (an

assignment of values to all variables) to the subset $A \subseteq X$ as $\mathbf{x}_A$, or explicitly state a variable's value, $x =$ "cat".

The purpose of a probabilistic model is to specify a distribution over the configurations of variables in the model. We refer to this distribution as the **model distribution**. If the model aims to capture the probability of randomly generating the observed data, $\mathbf{x}$, the model distribution will be the joint probability distribution, $P(\mathbf{x}, \mathbf{y})$.[1] If the aim of the model is to predict a set of variables given a set of observations, the model distribution will be the conditional distribution, $P(\mathbf{y}|\mathbf{x})$. Models of the latter type are known as discriminative (or conditional) models, and are the focus of this dissertation.

Such a distribution can be represented explicitly as a table, with each table entry corresponding to a single distinct configuration of variables (known as either a joint or conditional probability table). This requires $m^{|V|}$ cells, where $|V|$ is the number of variables in V, and m is the number of values per variable. Models with thousands of variables are not uncommon, and operating on the model distribution directly quickly becomes intractable. Fortunately, with respect to the fully-specified model distribution, the dependencies in our models will tend to be relatively sparse. Graphical models leverage this characteristic to factorize the model distribution into the product of localized functions, each ideally interacting with only a small number of variables.

2.2.1 Sequence Tagging with Probabilistic Models

Let us define a simple part-of-speech tagging model. Given a sequence of words $W = \{w\}_1^N$, we model the choice of part-of-speech tag for word w_i using a multinomial variable, y_i. Each tag variable y_i ranges over L values, where $L = \{\text{D, N, V}\}$, the set of part-of-speech tags. The the model specifies the probability of any configuration of these variables, and tagging the sentence amounts to finding the set of variable values which maximize this probability. For a generative model:

$$GOAL = \arg\max_{\mathbf{y}} P(\mathbf{x}, \mathbf{y})$$

where $\mathbf{x}$ is the set of observed variables, held fixed for a given sentence. For the sentence "*The cat scratches the man*", the observed variables are:

$\mathbf{x} = \{x_1 =$"the", $x_2 =$"cat", $x_3 =$"$scratches$", $x_4 =$"the", $x_5 =$"man" $\}$.

Consider one configuration of the predicted variables,

$\mathbf{y} = \{y_1 = \text{D}, y_2 = \text{N}, y_3 = \text{V}, y_4 = \text{D}, y_5 = \text{N}\}$.

A model could assign a probability by storing a table which maps any possible configuration of variables to its probability. For example, the model may score this configuration:

[1] A model which specifies the joint probability distribution, $P(\mathbf{x}, \mathbf{y})$ can also be used to optimize the conditional probability, $P(\mathbf{y}|\mathbf{x})$, through the application of Bayes rule. It follows from Bayes rule that $P(\mathbf{x}, \mathbf{y})$ is equal to $P(\mathbf{x})P(\mathbf{y}|\mathbf{x})$. In a classification task the observed variables are held fixed, and the remaining quantity is the conditional probability.

$$\begin{aligned} P(\mathbf{x}, \mathbf{y}) &= P(y_1 = D, y_2 = N, y_3 = V, y_4 = D, y_5 = N, \\ &\qquad x_1 = \text{“}the\text{”}, x_2 = \text{“}cat\text{”}, x_3 = \text{“}scratches\text{”}, x_4 = \text{“}the\text{”}, x_5 = \text{“}man\text{”}) \\ &= 0.7 \end{aligned}$$

Any such table represents a valid distribution provided it contains an entry for every configuration of variables, and that it normalizes as follows:

$$1.0 = \sum_{\mathbf{x},\mathbf{y}} P(\mathbf{x}, \mathbf{y}) \tag{2.1}$$

However, explicitly representing the model distribution with a table is not very efficient, as it does not exploit the dependencies, or the lack thereof, inherent between the variables. Here naively representing the distribution for a fixed sentence length of n requires a table of size of $|L|^n$ for the y variables, times an additional $|V|^n$ to account for the x variables. In this example where there are five words, three possible tags, and four possible words in the vocabulary, the table would be of size $3^5 \times 4^5 = 248832$.

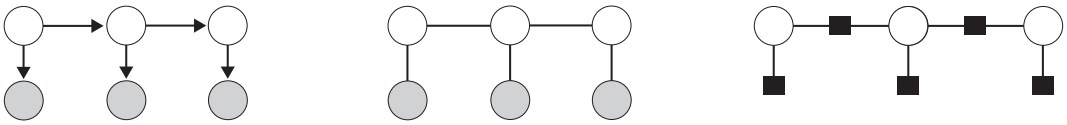
A graphical model exploits the intuition that actual dependencies between variables will be relatively sparse, and that variables will not strongly depend on a large number other variables. In English, the part-of-speech of word w_i is often highly indicative of the part-of-speech of the following word, w_{i+1}. For instance, if a word is a determiner (*the*), the following word will likely be a noun or adjective. But knowing this information is rarely useful when predicting the part-of-speech tag of word w_{i+5}, five words away. We can reflect this characteristic in the model by removing dependencies between distant tag variables.

Given sets of variables A, B, and C, we say A is *conditionally independent* of B given C if knowledge about B provides no additional information regarding A, in light of C. Formally, A is conditionally independent of B given C if $P(A, B|C) = P(A|C)P(B|C)$. In the tagging model we assume that y_i is conditionally independent of y_{i-2}, given y_{i-1}. This is known as making an *independence assumption*. The resulting model factorizes as follows:

$$P(\mathbf{x}, \mathbf{y}) = P(\mathbf{x}_1|\mathbf{y}_1) \prod_{i=2}^{n} P(\mathbf{x}_i|\mathbf{y}_i) P(\mathbf{y}_i|\mathbf{y}_{i-1}) \tag{2.2}$$

In this factorization the probability of any variable configuration can be expressed as the product of multiple tables, requiring one table with 12 entries for the distribution $P(\mathbf{x}_i|\mathbf{y}_i)$ and 9 entries for the distribution $P(\mathbf{y}_i|\mathbf{y}_{i-1})$, for a total of $12 \times 9 = 108$ entries. This is already a significant computational savings, but in practice it is often orders of magnitude larger. For instance, in this thesis we will define models which contain thousands of variables for a typical sentence, and here graphical models make what is often otherwise an intractable problem, tractable in many cases.

Most graphical models factorize the model distribution, but precisely how the distribution is formally defined is a property of the type of graphical model chosen. We now turn to discussing a particular class of graphical models, factor graphs, which we will focus on throughout the dissertation.



a.) Bayesian Network (HMM) b.) Markov Network (CRF) c.) Factor Graph

FIGURE 2.1: Representations of graphical models. Variable nodes are represented as circles, and factor nodes by boxes. Observed variables are shaded. In the factor graph representation the observed variables may be absorbed as features into factors, and are depicted as such throughout the dissertation.

2.2.2 Factor Graphs and Log-Linear Models

A **factor graph** [9] is a class of graphical model in which dependencies between variables are modeled using a set of additional nodes known as *factors*. Formally, a factor graph is an undirected bipartite graph $G = \{V, F, E\}$ connecting variable nodes $v \in V$ and factor nodes $f \in F$, with edges $e \in E \subseteq V \times F$. For any sets of variables $A, B, C \subseteq V$, A is separated from B if there is no path from $a \in A$ to $b \in B$ that does not pass through a variable in C. In a factor graph, if A is separated from B by C, A is conditionally independent of B given C (see the Hammersley-Clifford theorem [44] for more details).

A factor f has a corresponding potential function, Ψ_f, which assigns a real-valued weight to any configuration of its neighboring variables. Likely configurations of variables receive a high weight, unlikely configurations receive a low weight. These weights are known as *potentials*. Factor potentials play a pivotal role in how the model distribution is factorized in a factor graph model:

$$P(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{f \in F} \Psi_f(\mathbf{x}_A, \mathbf{y}_A) \tag{2.3}$$

where A is the subset of variables which neighbor f, and where Z, the partition function, ensures that the model normalizes to produce a proper probability distribution:

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{f \in F} \Psi_f(\mathbf{x}_A, \mathbf{y}_A) \tag{2.4}$$

In many cases computing this partition function will be intractable, but efficient approximations exist.

Graphically, factor graphs are depicted using circle nodes for variables and boxed nodes for factors, as illustrated in Figure 2.1c. Observed variables are shaded.

2.2.3 Factor Graph Sequence Models

Returning to our discussion of part-of-speech tagging, recall that we wish to leverage the properties of graphical models to efficiently factorize the model distribution. This can be accomplished by making an independence assumption, treating some variables as conditionally independent of others. A common independence assumption made in sequence models is to make y_t conditionally

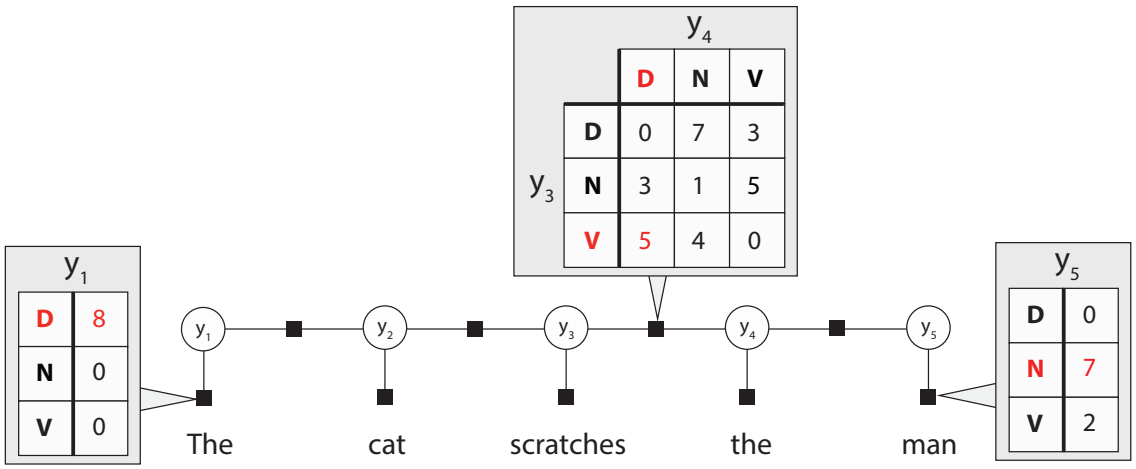


FIGURE 2.2: A linear chain CRF represented as a factor graph. A factor's potential function scores a configuration of variables, which may be represented by a table whose dimensionality corresponds to the number of neighboring variables. Each tag variable y_i is connected to a factor, u_i, whose potential table contains a potential for each tag value. The b_i factors, which connect y_{i-1} and y_i model the interactions between adjacent tag variables. We illustrate a subset of the model's potential tables, shown for factors u_1, u_5 and b_4.

independent of all other previous tag variables given y_{t-1}, otherwise known as a Markov assumption. When represented as a factor graph, this yields a first-order linear-chain conditional random field, as illustrated in Figure 2.2. For the sentence "*The cat scratches the man*", the model factorizes its distribution as follows:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_{i=1}^{n}\Psi_{u_i}(\mathbf{y}_i) \times \prod_{i=2}^{n}\Psi_{b_i}(\mathbf{y}_{i-1}, \mathbf{y}_i) \tag{2.5}$$

where u_i is the factor which neighbors y_i, and b_i is the factor which neighbors y_i and y_{i+1}.

What has happened to the observed variables, $\mathbf{x}$? A factor graph is a conditional model, and in conditional likelihood training we are interested in finding the model parameters which maximize the likelihood of the data, assuming the data is generated according to a conditional distribution. Because $\mathbf{x}$ variables are observed, their values are held fixed. Due to their role only as conditioning context, these variables can be incorporated into the factor functions themselves, as features, instead of being modeled explicitly. This is reflected in graphical depictions of factor graph models, as shown in the transition between Fig. 2.1b and c.

In this model the u_i factors each score how likely each tag is for the corresponding y_i tag variable. For u_5, $\Psi_{u_5}(y_5 = N) = 7$, significantly higher than other tags, reflecting the model's preference for variable y_5 to have the value N. The b_i factors score transitions from one part-of-speech tag to the next. For a common tag sequence, like D N, this score may be comparatively high, while sequences the model deems as unlikely, like D D, will be comparatively low. The highest scoring configuration is $\Psi_{b_4}(y_3 = D, y_4 = N) = 7$. However, individual factors may contribute conflicting information. For instance, it is unlikely that the best solution will contain $y_3 = D$ and $y_4 = N$ despite being the highest tag transition score, due to conflicting evidence from

u_3: *scratches* does not function as a determiner so a data driven estimate of $\Psi_{u_3}(y_3 = D)$ will be extremely low.

Finding the global consensus of which variables should have which values is the subject of **inference**, and is discussed in Section 2.3 (pg. 28).

2.2.4 Log-linear Models

We have described how a factor graph model uses factor potentials to factorize the joint distribution, but where do these potentials come from and how does learning occur? Many factors, which we refer to as **soft factors**, output a non-negative real-valued potential derived from feature weights. For the remainder of this section we can assume that each soft factor potential function has the general form of a log-linear model with a K-dimensional parameter vector, θ:

$$\Psi_f(\mathbf{x}_A, \mathbf{y}_A) = \exp\left(\sum_{k \in K} \theta_k \cdot \xi_{f_k}(\mathbf{x}_A, \mathbf{y}_A)\right) \tag{2.6}$$

where ξ_f is a feature function of factor f over a configuration of its neighboring variables $\mathbf{y}_A$, and conditioned on observed variables $\mathbf{x}_A$. ξ_{f_k} returns the count of the feature associated with k, an index into the model parameters, for the given configuration. For an indicator feature, which is either present or not, $\xi_k = \{0, 1\}$. Under this formulation the model views the data only through factors, which in turn view the data only through sets of associated features. The weight of each feature is a parameter in the model. During learning a parameter estimation method is used to fit the model distribution to the empirical distribution through the adjustment of feature weights.

This concludes the preliminary discussion of factor graph models. However, one of the advantages of modeling with factor graphs lies in being much more adventurous in the definition of potential functions. By defining specialized factors, logical or combinatorial constraints can be imposed in the model. Section 2.4 describes this approach and Section 2.4.2 lists a complete set of factors used throughout the dissertation. These can be thought of as the building blocks of factor graph models – each factor a modular, reusable component – which in sum are capable of representing many sophisticated models, with little problem-specific implementation. This general approach is largely due to message passing inference, the subject of the next section, which provides a method for reasoning regardless of graph structure.

2.3 Inference via Belief Propagation

Inference on factor graph models can be performed using efficient, well-studied message-passing algorithms, even on large graphs with many interdependent variables. In this section we describe the standard sum-product algorithm which provides an exact solution to inference when the graph is a chain or tree. When the graph contains cycles, a variant of this algorithm known as "loopy" belief propagation provides a method for performing approximate inference and has been shown to perform well in practical applications. We present belief propagation inference in the following section, and discuss the role of inference in decoding in Section 2.3.5 (pg. 34).

2.3.1 What Inference Accomplishes

When training a model using maximum likelihood estimation, model parameters are updated to maximize the likelihood of the training data. If the optimization method is gradient-based, as it is throughout this thesis, the likelihood is maximized by first calculating the gradient of the likelihood function, and then updating the parameters proportionally. The key to computing this gradient lies in inference. We will see in Section 2.5.4, (pg. 49) how the gradient decomposes into a difference between two expectations over model parameters. Inference produces **marginal distributions**, which can in turn be used to compute the expectations necessary for optimization.

A marginal distribution (also referred to simply as a marginal, or as a *belief*) is a probability distribution over a set of variables without any reference to the values of variables outside of the set. This differs from a conditional probability where the probability of a set of variables x_A is conditioned on an additional set of variables (however, a marginal may still be conditioned on observed variables in a factor graph, since they can be incorporated into the model as features).

Let $\pi(x_A)$ be the marginal probability of a set of variables, x_A. A marginal probability can be calculated by summing over (*marginalizing*) all values of the remaining variables, y_B:

$$\pi(x_A) = P(x_A) = \sum_{\mathbf{y}} P(x_A, \mathbf{y}) = \sum_{\mathbf{y}} P(x_A|\mathbf{y})P(\mathbf{y}) \tag{2.7}$$

Consider a marginal computation in the simple linear-chain CRF (of the sort shown in Fig. 2.1c) with the factorization over three variables: y_1, y_2 and y_3. Let $b_{1,2}$ be the factor that coordinates between y_1 and y_2, and let $b_{2,3}$ be the factor that coordinates between y_2 and y_3:

$$P(y_1, y_2, y_3) = \frac{1}{Z}\Psi_{b_{1,2}}(y_1, y_2)\Psi_{b_{2,3}}(y_2, y_3) \tag{2.8}$$

We omit the factors over single variables for brevity. For the purpose of training, we will require a marginal for each variable. To compute the marginal $P(y_2)$ we sum over the other variables:

$$P(y_2) = \sum_{y_1}\sum_{y_3} \frac{1}{Z}\Psi_{u_1}(y_1, y_2)\Psi_{u_2}(y_2, y_3) \tag{2.9}$$

This can be simplified using the distributive law [45]:

$$P(y_2) = \sum_{y_1}\sum_{y_3} \frac{1}{Z}\Psi_{u_1}(y_1, y_2)\Psi_{u_2}(y_2, y_3) \tag{2.10}$$

$$= \sum_{y_1} \frac{1}{Z}\Psi_{u_1}(y_1, y_2) \sum_{y_3} \Psi_{u_2}(y_2, y_3) \tag{2.11}$$

$$= \sum_{y_1} \frac{1}{Z}\Psi_{u_1}(y_1, y_2) m_3 \tag{2.12}$$

$$= \frac{1}{Z} m_1 m_3 \tag{2.13}$$

substituting temporary m factors in for the marginalized terms. Assuming each variable has r values, this factorization reduces the complexity from r^3 in Eq. 2.9 to r^2 in Eq. 2.10, as noted by a reduction in the maximum number of variables within each summation.

Now consider the marginal computation for y_1. Calculating $\pi(y_1)$ requires the same $\sum_{y_3} \Psi_{b_{2,3}}(y_2, y_3)$ term needed for the the computation of $\pi(y_2)$. For training we will require marginals for every variable (or factor) in the model, and repeatedly computing the same intermediate terms should be avoided. This is an advantage of **belief propagation** inference, which provides an efficient method for simultaneously computing all marginals in the graph.

The logic lies in the observation that an ideal elimination order for each variable will marginalize out its neighbors before marginalizing out itself. To compute the marginal for y_1 we must first eliminate y_3, then y_2. To compute the marginal for y_3 we must first eliminate y_1, then y_2. If as we eliminated these terms we were to cache in y_2 the temporary factors produced, when all of its neighbors have been marginalized y_2 will contain all the necessary quantities to compute its marginal. This holds for all intermediate nodes in the chain. So the quantities required to produce all marginals of a chain or tree can be computed by one pass through the chain (which is sufficient for calculating the marginal of the chain's end, y_t, as it only has one neighbor), and one pass back. Because child nodes (and their successors) are conditionally independent given a shared root node, and a tree is merely a set of chains with a shared root, this algorithm also generalizes to trees.

It may be more convenient to drop the notion of elimination, and instead view the operation of recursive elimination and caching as distributing messages across the graph. Variables which have received messages from all their neighbors have stored the necessary information to compute their marginal. In Eq. 2.10, the temporary m factors *are* precisely these messages. Methods of this sort are known as *message-passing* algorithms, and can be thought of as a dynamic programming solution to efficient inference, dating back to [10]. Pearl first described belief propagation as a message-passing algorithm for inference in directed graphical models, though message-passing algorithms subsume or parallel other model-specific inference algorithms later developed in the NLP community: the forward-backward algorithm for inference in HMMs [46], and the inside-outside algorithm for PCFGs [47].

In comparison to Gibbs sampling, belief propagation makes use of expectations rather than discrete choices. As we now turn to the sum-product algorithm, we note that modified versions of this algorithm (Section 2.4, pg. 36) allow for the efficient computation of highly-structured models that are slow to mix with a sampling-based approach.

2.3.2 Belief Propagation with the Sum-Product Algorithm

The sum-product algorithm is a message-passing inference algorithm for factor graphs and the standard inference method used throughout this dissertation. We described the general calculation of marginal messages in Sec. 2.3.1. For factor graphs, which contains two different types of nodes, there are two different types of messages.

First, messages from variables to factors. Paraphrasing from [48], the desired outgoing message from a variable v to its neighboring factors conveys the information, "My *other* neighbors jointly suggest I have the posterior distribution $m_{v \to u}(v)$ over my values." We treat incoming messages as independent, so it follows that the way in which neighboring factors *jointly* suggest a

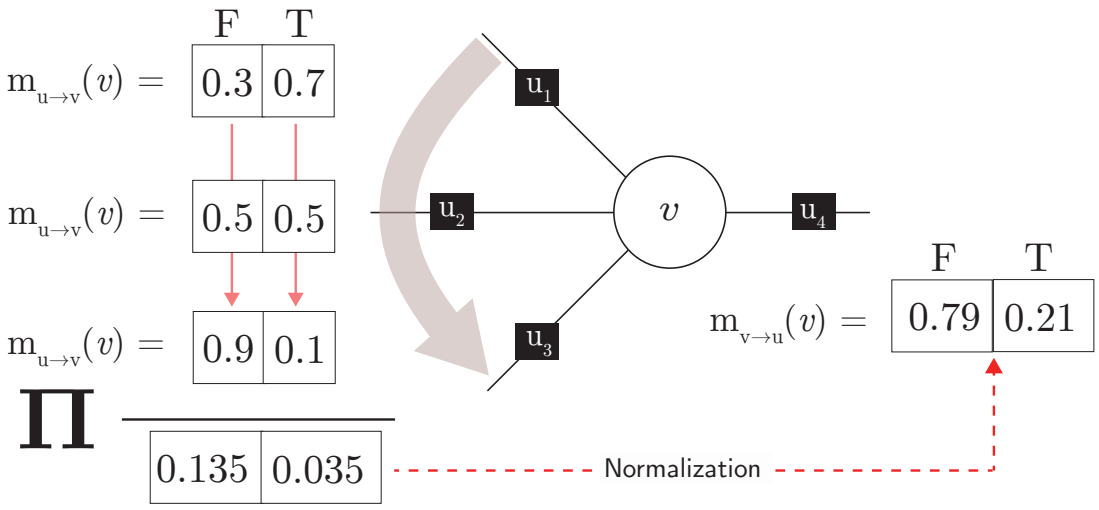


FIGURE 2.3: BP message calculations: variable to factor. To calculate the outgoing message from variable v to factor u_4 we collect messages from all neighboring factors, excluding u_4, and multiply corresponding elements from all incoming messages, i.e., all message components corresponding to the value true are multiplied together, and all corresponding to false are multiplied together.

variable's posterior is through the multiplication of their messages. When dealing with the multiplication of messages and potentials we are referring to element-wise multiplication: all message components corresponding to the variable's value being true are multiplied together, and all message components corresponding to the variable's value being false are multiplied together, and so forth for all values in the case of multinomial variables.

- Messages from a variable node v to a factor node u:

$$m_{v\to u}(v) = \prod_{u^* \in \mathcal{N}(v)\setminus\{u\}} m_{u^*\to v}(v) \tag{2.14}$$

where $\mathcal{N}(v)$ is the set of factors which neighbor variable v, and where $\setminus\{u\}$ omits factor u from this set. An example variable message calculation is provided in Figure 2.3.

Computing messages from factors to variables is slightly more involved. Here the desired message from a factor u to a variable v conveys "Together with my potential function and the messages from my *other* neighbors, I suggest you have the posterior distribution $m_{u\to v}(v)$ over your values." Again we would like to combine the messages from all neighbors $N(u)$ which are not the destination variable v, but in this case we must combine those messages with the factor's own potential function, and marginalize over those beliefs to produce the desired output message:

- Messages from a factor node u to a variable node v:

$$m_{u\to v}(v) = \sum_{\sim v} \left(\prod_{v^* \in \mathcal{N}(u)\setminus\{v\}} \Psi_u \cdot m_{v\to u}(v^*) \right) \tag{2.15}$$

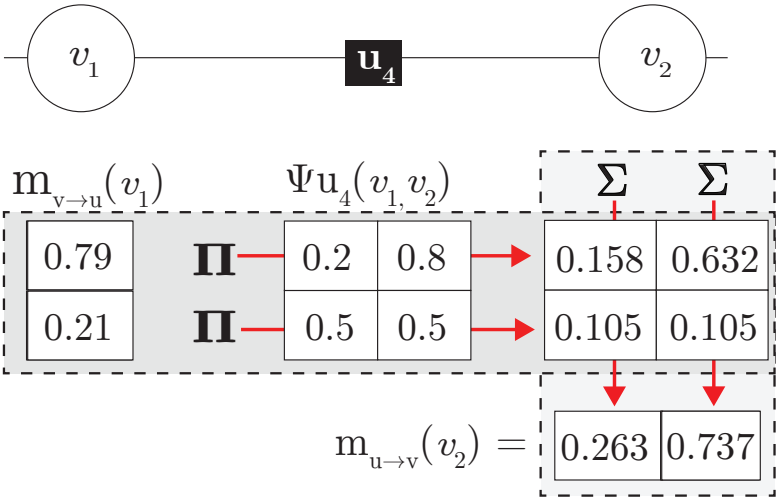


FIGURE 2.4: BP message calculations: factor to variable. To calculate the outgoing message from factor u_4 to variable v_2. Factors maintain distributions (in the form of potential tables in the discrete case) over neighboring variables, so when computing the message to variable v_1, all incoming messages are multiplied into the potential table using element-wise multiplication, excluding the destination variable v_2. It should be noted that each incoming message aligns to a different dimension of the potential table (i.e., here the reverse message from u_4 to v_1 would multiply *down* Ψ_{u_4}). The outgoing message is computed by marginalizing over the dimensions associated with the incoming message, yielding the factor's belief of v_2's values as informed by all other neighboring variables. This is clear in the case of a two-dimensional potential table: an incoming message may multiply across rows, making the marginalization operation a sum down columns.

Here we use Ψ_u to directly reference u's potential table, a matrix with $|\mathcal{N}|$ dimensions. Each dimension of Psi_u corresponds to one neighboring variable, with the size of the matrix in that dimension being equal to the number of values the corresponding variable can take. Thus the term within parenthesis returns a matrix consisting of the u's original potential table, multiplied with the incoming messages from each neighboring variable (excluding v).

Outside of the parenthesis, the summation over $\sim v$ indicates that the matrix is summed over, element-wise, in the direction of v. This produces a vector whose length is the number of possible values v can take (i.e., if v is Boolean, the resulting vector will have two elements). This is u's outgoing message to v.

These operations are illustrated visually in the case of a two-dimensional potential table (Figure 2.3.2). In the example calculation the single incoming message multiplies across the potential table and the marginalization is a summation downward, producing the desired binary-valued output message. For more detailed discussion of the basic theory underlying message-passing inference, we refer the reader to [45].

Computing Beliefs

Once message passing has terminated the marginals can be computed as follows:

$$P(v) = \frac{1}{Z} \prod_{u \in N(v)} m_{u \to v}(v) \tag{2.16}$$

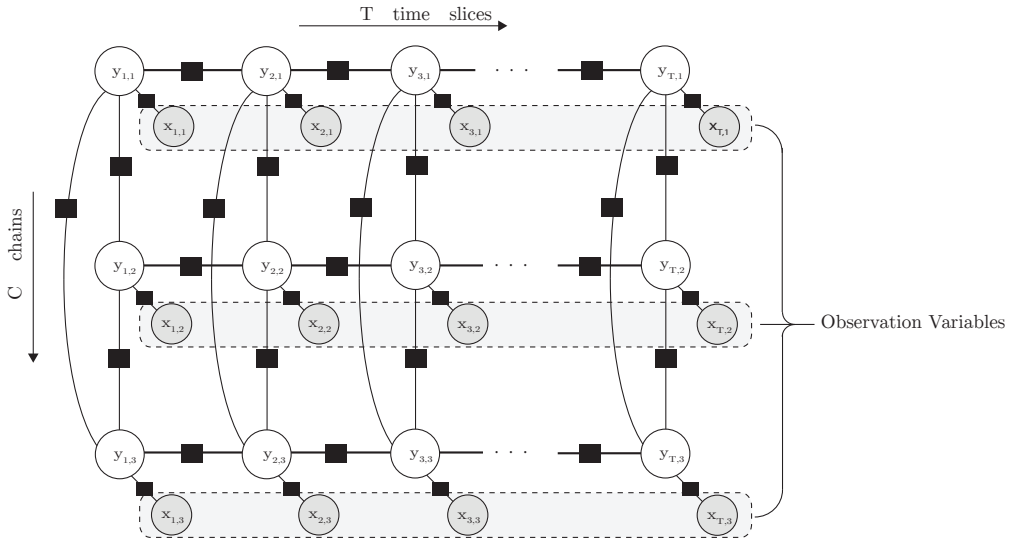


FIGURE 2.5: A factorial CRF. A model whose graph contains many cycles, but where approximate inference has proven suitable for NLP models [43]. The models presented in this thesis frequently exhibit a similar mix of cyclic dependencies and repeated substructures which can be used in defining a specialized inference order. The unary factors which attach to each label variable have been omitted for clarity.

$$P(u) = \frac{1}{Z} \prod_{v \in N(u)} \Psi_u \cdot m_{v \to u}(v) \tag{2.17}$$

Marginals are calculated for each type of node in a manner analogous to the messages they pass, but aggregate the messages from *all* neighbors, and normalize. Here Z refers to normalization within this calculation and not the global model normalization. Our earlier discussion cited marginals as the quantities of interest, necessary for gradient-based optimization. While this is true, it is in fact the *factor marginals* that are utilized in our work, as factors serve to connect the graph structure to features.

2.3.3 Loopy Belief Propagation

Belief propagation is only guaranteed to provide an exact solution when the graph is a chain or tree, but often times our goal will be to perform inference on a graph that contains cycles. Consider the factorial CRF, which models both sequential dependencies *and* dependencies between each chain (Fig. 2.5). While the sequential chain structure of the model is apparent, dependencies between chains create loops, preventing the graph from containing a single non-repeating order in which to perform message passing. In this situation one may choose to ignore this fact and use the sum-product algorithm regardless in what is known as **loopy belief propagation** (LBP).

Loopy inference differs from standard sum-product in that there is no longer a well-defined order in which to send messages, and thus there is no guarantee that inference will converge to the exact solution. Instead, the algorithm is applied for some set number of iterations, typically with some threshold to signify convergence (i.e., the maximum change in a single variable's beliefs in a single iteration, the sum change of variable beliefs in the graph, etc.). If the maximum deviation in

marginal beliefs from one iteration to the next is less than this threshold, the procedure terminates. Aside from these differences, the algorithm is identical.

This may sound unprincipled. Indeed, the behavior of belief propagation in loopy graphs is still not well understood, and BP may not be convergent at all: it can also cycle or diverge from the correct solution. It has been shown that a graph containing only a single cycle is still guaranteed to converge [49], and some sufficient conditions for convergence have been identified [50]. Convergence in loopy graphs has been linked to the Gibbs measure [51], such that LBP is capable of reaching a state of convergence whenever the Gibbs measure is unique. There have also been bounds placed on the propagation of message errors [52]. When the degree of error goes to zero, convergence can be assumed. However, these contributions remain largely theoretical.

Despite these concerns, LBP has proven exceptionally useful in many tasks. LBP is equivalent to algorithms developed for efficient turbo decoding and free energy approximation [53], and in practice has been used repeatedly in NLP (for sequence models [43], parsing [11]). It also compares favorably to other approaches to model combination, such as dual decomposition [24].

The models presented in this dissertation are significantly more complex than the models used for performing turbo code decoding and other problems that are exemplary applications for LBP, and a natural question to ask may be, "how good is the approximation provided by loopy belief propagation for work in the NLP setting, for the types of models presented in the following chapters?" We do not attempt to answer this question directly, and show only that from a practical perspective these models are capable of state-of-the-art performance in many tasks using this inference strategy.

2.3.4 Message Order

Belief propagation in guaranteed to converge in chains and trees, but the efficient two-iteration convergence associated with the forward-backward computations is only possible when messages traverse the graph from end to end. With LBP the ideal message order is less clear, but can still be an important factor for the performance of inference. For instance, a factorial CRF is a cyclic model, but it has strong linear constraints, and previous work has explored the benefits of specialized message queues to exploit this structure [54].

Message ordering as an important consideration, but defer to future sections where these details are provided on a model-specific basis. For some models we use an essentially random message-passing order, and in others we try to exploit substructures where a more efficient ordering can be employed. In all models we prioritize messages from unary factors, computing these messages first, as these factors have only one neighboring variable and need not be updated until their marginal values are queried at the end of inference. Repeatedly processing them only leads to redundant computations.

2.3.5 Minimum Bayes-Risk

In Section 1.1.1 (pg. 3) we discussed a joint decoding strategy which made use of full model posteriors instead of approximations in which each model contributes only a list of its most probable solutions, often containing just the single most probable analysis under the model. Message-passing inference techniques can also be thought of as a generalization of these special-purpose decoding

algorithms.

A common decoding strategy consists of calculating the *maximum a posteriori* (MAP) solution, which minimizes the expected loss for an entire structure, yielding the 1-best solution:

$$\mathbf{y}_{MAP} = \arg\min_{\mathbf{y}} \mathbb{E}_{P(\hat{\mathbf{y}}|\mathbf{x})}[\delta(\mathbf{y}, \hat{\mathbf{y}})] \tag{2.18}$$

where $\delta(\mathbf{y}, \hat{\mathbf{y}})$ is a loss function ranging over $\{0, 1\}$ (knowns as 0-1 loss):

$$\delta(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} 0, \text{ if } \mathbf{y} = \hat{\mathbf{y}} \\ 1, \text{ otherwise} \end{cases} \tag{2.19}$$

and $\mathbf{y}$ is the set of predicted variables which collectively specify a complete structure, and $\hat{\mathbf{y}}$ is the corresponding gold structure. If the model is a parser, then MAP decoding seeks to minimize the expected whole-tree loss, with $\delta(\mathbf{y}, \hat{\mathbf{y}})$ returning 1 if the *entire* tree is accurate, and 0 otherwise. If the model is a tagger, $\delta(\mathbf{y}, \hat{\mathbf{y}})$ returns 1 if the *entire* tag sequence is accurate, and 0 otherwise. This solution can be computed – either exactly or approximately – using posteriors produced via max-product inference.[2]

However, there are many reasons to consider an alternate decoding method. The standard evaluation measures for many structured prediction tasks rarely score entire structures with a 0-1 loss, opting instead to decompose evaluation metrics over smaller substructures. For instance, the standard parsing evaluation – precision, recall, and F1 – are calculated over individual arcs or constituents, not entire trees. In cases like these, a loss function that more directly reflects the evaluation measure has shown to improve performance. Maximizing expected recall has been suggested as an alternative loss function for parsing, both for CCG [55] and PCFG [56] parsing.

It is also worth considering how suitable the goal of MAP decoding is for joint models with hidden structure. If the model is joint, and one or more component models represent a supporting task which is of no consequence, then a decoding method which includes these variables in the maximization may not be ideal. An alternate decoding method might incorporate the beliefs of the supporting models, but maximize only over the posteriors of the variables associated with the end task. This could again be thought of as choosing a decoding strategy which more accurately reflects the evaluation measure.

This is precisely the advantage offered by **Minimum Bayes-risk** decoding (MBR), which generalizes Eq. 2.18 to an arbitrary loss function, l:

$$\mathbf{y}_{GOAL} = \arg\min_{\mathbf{y}} \mathbb{E}_{P(\hat{\mathbf{y}}|\mathbf{x})}[l(\mathbf{y}, \hat{\mathbf{y}})] \tag{2.20}$$

Here the loss function l can be chosen to match the evaluation measure. For instance, the most common evaluation loss function for a part-of-speech tagger is the number of tag errors. This can be integrated into an MBR decoding method:

[2]Max-product inference is a message-passing inference procedure analogous to sum-product, but for which the summation in the factor-to-variable message computation (Eq. 2.3.2, pg. 31) is replaced with a maximization.

$$\hat{\mathbf{y}} = \arg\min_{\mathbf{y}} \mathbb{E}_{P(\hat{\mathbf{y}}|\mathbf{x})} \left(\sum_{i=1}^{n} l(\mathbf{y}_i, \hat{\mathbf{y}}_i) \right) \tag{2.21}$$

$$= \arg\max_{\mathbf{y}} \sum_{i=1}^{n} P(\mathbf{y}_i = \hat{\mathbf{y}}_i|\mathbf{x}) \tag{2.22}$$

The tag variable posteriors required for Eq. 2.22 can be calculated for the model using the sum-product algorithm. In the case of joint models, these posteriors inherently reflect the beliefs of any hidden structure variables, and any hard constraints we may place over them. Ultimately this is a question of whether hidden structure variables should be maximized together with query variables, or summed over. Max-product inference provides the maximization solution (and the MAP decoding strategy), and sum-product inference provides the summation solution and posteriors compatible with MBR decoding. When l is a 0-1 loss over entire structures, and there are no hidden variables, these decoding strategies are identical.

2.4 Structured Models and Combinatorial Constraints

Belief propagation provides an efficient method for performing marginal inference, but there are situations where this efficiency breaks down. While factor graphs can represent many NLP models using only basic components, representing combinatorial structures such as syntactic trees requires densely-connected graphs to rule out invalid structures. In a graphical model a **clique** is a subset of variables in which every two variables is connected in the graph (via factors, in the case of factor graphs). Complexity of inference in graphical models is a function of the largest clique size, often making inference in combinatorial structures intractable. A recent innovation allows us to sidestep some of these complexity problems by incorporating efficient enumerations and dynamic programming algorithms into factor message computations. We refer to such factors as **combinatorial** factors.

In Section 2.4.1 we introduce the general method for implementing combinatorial factors by describing the message computation for the ISATMOST1 combinatorial factor. In Section 2.4.2 (pg. 40) we list all the special-purpose factors used in the thesis. Once defined, these factors can be thought of as a set of modular building blocks for factor graph models. It is straightforward to construct new models simply by connecting subsets of variables with the appropriate factors, and inference and learning can remain largely unchanged. Section 2.4.3 (pg. 43) describes how Boolean logic factors can be used to define an interface between two component models, producing a joint model where inference can be performed jointly over both tasks.

2.4.1 Message Computation for the IsAtMost1

In this section we introduce the ISATMOST1, a combinatorial factor originally described in [11], and describe how the outgoing messages are computed for this factor. This serves as an overview of the general methodology underlying all combinatorial factors presented in this thesis.

The ISATMOST1 factor is useful for representing, with Boolean variables, a labeled structure which may or may not be present. Such structures occur frequently throughout this thesis,

Algorithm 1 Computing outgoing messages for an IsAtMost1 factor

procedure PROPAGATEISATMOST1($v_I, \{d_1, ..., d_m\}$) ▷ indicator, label variables
  $\neg\overline{q}_{v_I} \leftarrow v_I(\textsf{false})/v_I(\textsf{true})$ ▷ Collect log-odds
  for $v \leftarrow \{d_1, ..., d_m\}$ **do**
    $\overline{q}_v \leftarrow v(\textsf{true})/v(\textsf{false})$
  end for
  $Z \leftarrow \neg\overline{q}_{v_I}$ ▷ Sum Z
  for $v \leftarrow \{d_1, ..., d_m\}$ **do**
    $Z \overset{\oplus}{\leftarrow} \overline{q}_v$
  end for
  $m_{u \to v_I}(v_I) \leftarrow [1.0, Z - \neg\overline{q}_{v_I}]$
  for $v \leftarrow \{d_1, ..., d_m\}$ **do** ▷ Assign messages to edges
    $m_{u \to v}(v) \leftarrow [Z - \overline{q}_v, 1.0]$
  end for
end procedure

FIGURE 2.6: Outgoing message computation for the ISATMOST1 factor. The notation $v(\textsf{true})$ references variable v's belief that it is true, obtained from the incoming message from v. First the incoming messages are collected and converted to an odds ratio. Second, Z is computed, using the terms in Eq. 2.25. Finally, outgoing messages are computed as a ratio with respect to the variable being true or false. A plussed arrow ($\overset{\oplus}{\leftarrow}$) indicates incremental assignment.

and include syntactic spans (Chapter 3), named entities (Chapter 4), relations (Chapter 5) and semantic roles (Chapter 6). Let v_I be the Boolean variable indicating the presence of a structure, and $\{d_1, ..., d_m\}$ be a set of label variables. An ISATMOST1 factor constrains the set of Boolean variables using the following potential function:

$$\Psi_{\text{ISATMOST1}}(v_I, d_1, .., d_m) = \begin{cases} 0, & \text{if } \begin{cases} v_I = \textsf{false} \wedge \exists d_i \in \{d_1, .., d_m\} : d_i = \textsf{true} \\ v_I = \textsf{true} \wedge \forall d_i \in \{d_1, .., d_m\} : d_i = \textsf{false} \\ v_I = \textsf{true} \wedge \exists d_i, d_j : d_i = \textsf{true} \wedge d_j = \textsf{true} \wedge\ i \neq j \end{cases} \\ 1, & \text{otherwise} \end{cases} \tag{2.23}$$

If the indicator variable is false ($v_I = \textsf{false}$) there is no structure to label, and the factor imposes a hard constraint preventing any of the label set variables from being true. If the indicator variable is true ($v_I = \textsf{true}$) exactly one variable from the label set must also be true. Permissible configurations correspond to an output of 1, and impermissible configurations correspond to an output of 0. In this sense the factor and its neighboring variables function like a multinomial variable where one of the multinomial's values is the null value, to indicate the lack of a structure.

Though this factor imposes a hard constraint, the outgoing messages computed during inference adjust variable beliefs *toward* one of these outcomes (i.e., the model does not force all neighboring variables to adhere to this constraint). In order to compute this gradient, we must compute a normalization function. The following describes this process, and the intuition which underlies a similar computation in all the combinatorial factors we present in this dissertation.

What distinguishes combinatorial factors from standard factors is that a combinatorial factor does not utilize a potential table to compute outgoing messages. Doing so would be inefficient in most cases, and intractable in others. Consider the ISATMOST1 factor. All variables which neighbor an ISATMOST1 factor are Boolean, yielding 2^n possible assignments, but only $n+1$ of these configurations do not violate the constraint. Clearly it would be burdensome to enumerate all possible 2^n assignments, multiplying messages and marginalizing as defined in Eq. 2.3.2, especially in situations where a model requires a large number of labels or ISATMOST1 factors.

A more efficient strategy is to compute the normalization function, Z, and the outgoing messages directly. Assume an ISATMOST1 factor coordinates between four variables: an indicator variable v_I and a set of three label variables ($\{d_1, d_2, d_3\}$). The total score of non-violating assignments under this constraint, Z, can be expressed as:

$$\begin{aligned} Z = &(1 - v_I)(1 - d_1)(1 - d_2)(1 - d_3) \\ &+ v_I \cdot d_1(1 - d_2)(1 - d_3) \\ &+ v_I \cdot d_2(1 - d_1)(1 - d_3) \\ &+ v_I \cdot d_3(1 - d_1)(1 - d_2) \end{aligned}$$

The first line is a score pertaining to the satisfying configuration where all variables are false. The remaining three lines each represent the score for when the indicator variable is true, and exactly one of the three remaining variables is true.

A factor potential table could be implemented to compute outgoing messages in accordance with Eq. 2.3.2, where $\Psi_{\text{IsAtMost1}}(\mathbf{x}, \mathbf{y}) = 0$ for all but the above four cases, in which case the output of the function should be 1. A combinatorial factor differs in that it has no explicit potential table, and computes Z directly. In the case of ISATMOST1, note that if we divide out $v_I\ (1 - d_1)\ (1 - d_2)\ (1 - d_3)$, then Z is simply the sum of these four quantities:

$$Z = \frac{1 - v_I}{v_I} + \frac{d_1}{1 - d_1} + \frac{d_2}{1 - d_2} + \frac{d_3}{1 - d_3} \tag{2.24}$$

with each term having the form of an *odds ratio*. We denote the odds ratio $\overline{q}_v = \frac{v(\text{true})}{v(\text{false})}$ and $\neg\overline{q}_v = \frac{v(\text{false})}{v(\text{true})}$.This greatly simplifies the computation. As shown in Fig. 2.6, like many of the combinatorial factor computations, the computation of outgoing messages for ISATMOST1 can now be partitioned into three steps: (1) gather incoming messages and convert to odds ratios, (2) calculate the normalizing constant Z, (3) compute and propagate each outgoing message by subtracting the corresponding incoming component from the partition function. The calculation hinges primarily on the calculation of Z, which is nearly trivial in this case, but may rely on more complex dynamic programming algorithms to express structure constraints (see the description of the CKY-TREE factor for phrase structure syntax, Section 3.1.2, pg. 72). Here we only substitute the odds ratios terms, $\overline{q}_v$ for variable v, into Eq. 2.24:

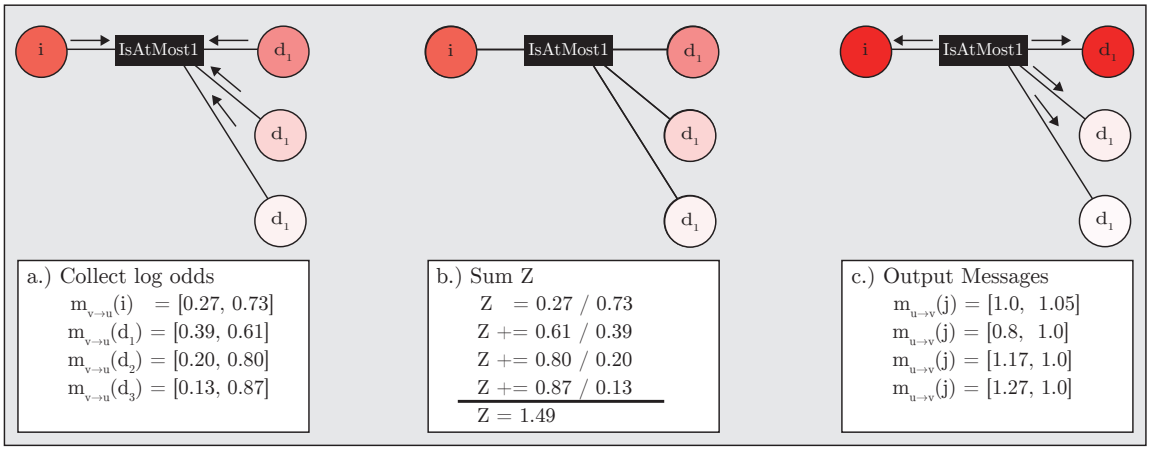


FIGURE 2.7: Message Computations for ISATMOST1 Factor. The indicator variable variable begins with a strong lean toward a value of true at 0.73, while the three dependent variables beliefs are in varying degrees of false. After summing Z (note the reciprocal fraction to compute i's contribution) and computing output messages (note that only one value is computed for each variable as a ratio with respect to 1.0) both i and d_1 become more likely, while d_2 and d_3 strengthen their false beliefs.

$$Z = \neg\overline{q}_{v_I} + \overline{q}_{d_1} + \overline{q}_{d_2} + \overline{q}_{d_3} \tag{2.25}$$

The outgoing messages are computed as ratios with respect to the variable being true or false. For the indicator variable the outgoing message is:

$$m_{u \to v_I}(v_I = \mathsf{true}) = Z - \frac{v_I(\mathsf{false})}{v_I(\mathsf{true})} \tag{2.26}$$

$$m_{u \to v_I}(v_I = \mathsf{false}) = 1.0 \tag{2.27}$$

For the variables representing the label set, the outgoing messages are as follows:

$$m_{u \to d_I}(d_i = \mathsf{true}) = 1.0 \tag{2.28}$$

$$m_{u \to d_I}(d_i = \mathsf{false}) = Z - \frac{d_i(\mathsf{false})}{d_i(\mathsf{true})} \tag{2.29}$$

Figure 2.7 illustrates the calculation of these outgoing messages for a simple model.

This concludes the discussion of how combinatorial factors differ from standard feature-based soft factors with log-linear potentials. In future chapters we will delve into the implementations behind many other factors while highlighting the contexts in which they are useful. An overview of all factors contained in this dissertation is presented in the following section.

2.4.2 Constraint Inventory

The following sections introduce the various special-purpose factors that serve as building blocks for the models described throughout this dissertation. Here we categorize factors along three major distinctions:

- **Soft Factors**

  A soft factor f is a factor whose possible outputs cover the entire range of $\mathbb{R}^+$, and whose potential function is learned as a function of parameters in the model. For the purposes of this dissertation, soft potential functions are always a log-linear combination of feature functions and model parameters:

$$\Psi_f(\mathbf{x}_A, \mathbf{y}_A) = \exp\left(\sum_k \theta_k \cdot \xi_{f_k}(\mathbf{x}_A, \mathbf{y}_A)\right) \tag{2.30}$$

  where θ_k is the kth model parameter, and ξ_f is the feature function for factor f. The feature function returns the count of each feature that factor f associates with the configuration of variables, $(\mathbf{x}_A, \mathbf{y}_A)$, with $\vec{\xi}_{f_k}(\mathbf{x}_A, \mathbf{y}_A)$ returning the count of the feature corresponding to parameter θ_k.

  Soft *Boolean logic* factors, a special case of soft factors, learn a potential only for configurations which violate a given Boolean logic. Following the conventions outlined in [48], soft Boolean logic factors are named for the case in which the Boolean logic returns false. For instance, a NAND factor is defined such that its soft potential applies to the case where both variables are true:

$$\text{NAND}_{\Psi_A(\mathbf{x}_1,\mathbf{x}_2)} = \begin{cases} \exp\sum_k \theta_k \cdot \xi_{f_k}(\mathbf{x}_A, \mathbf{y}_A), & \text{if } x_1 = \textsf{true} \wedge x_2 = \textsf{true} \\ 1, & \text{otherwise} \end{cases} \tag{2.31}$$

  Instead of imposing a hard constraint which prohibits configuration of variables that violate the Boolean logic, a soft Boolean logic factor can learn a real-valued weight to penalize or promote this configuration. These factors are utilized at the interface between joint models, as described in Section 2.4.3 (pg. 43).

- **Hard Factors**

  A hard factor f discerns between only two scenarios: a configuration of neighboring variables is valid, or it is not valid. In the former case the factor does not alter the model score, multiplying in a potential of 1. In the latter case, it multiplies in a potential of 0, prohibiting the configuration. These factors are not associated with a set of features.

- **Combinatorial Factors**

  A combinatorial factor is a special-purpose factor for efficiently enforcing a combinatorial structure constraint on a set of variables. While not a category outside of the hard/soft

distinction – combinatorial factors are a type of hard factor – they are used in two very distinct ways.

A combinatorial factor is more global in nature (earlier work even goes so far as to refer to such factors as *global factors* [11]), connecting to all the variables used to represent the structures of a particular model (span variables in phrase structure parsing or named entity recognition, and dependency relation variables in dependency parsing). Therefore it is rare that any one component model would have more than one combinatorial factor, while a component model may have many non-combinatorial hard factors, primarily to aid in representing the interface between models with Boolean variables.

The algorithms implemented by combinatorial factors are also very different in nature and scope, their origins firmly nested in the NLP literature. In contrast, the other hard factors implement simple algorithms to constrain configurations of variables in ways expressible by a logical function, with no sophisticated in-domain knowledge. In short and in practice, a combinatorial factor is used to efficiently encode a single useful syntactic or otherwise structural constraint, and the factors we refer to as hard factors are used to repeatedly and locally infuse simple bits of logic into the model.

Hard Factors

- **EXACTLY1** is a hard factor which coordinates a set of Boolean variables to ensure that exactly one variable of the set is true. This is useful for situations where the behavior of a multinomial distribution is required, but Boolean variables are preferred.

$$\text{EXACTLY1}_{\Psi(x_1,..,x_n)} = \begin{cases} 0, & \text{if either } \begin{cases} \forall x, y : x = \mathsf{true} \wedge y = \mathsf{true} \wedge \ x \neq y \\ \forall x : x = \mathsf{false} \end{cases} \\ 1, & \text{otherwise} \end{cases} \tag{2.32}$$

- **ISATMOST1** is a hard factor as described in Section 2.4.1 (pg. 36). It constrains a set of Boolean variables, one an indicator variable, with the following logic: if the indicator variable is false, all other variables are false. If the indicator variable is true, exactly one of the remaining variables must be true. This factor is useful for predicting a labeled structure which may or may not be present.

$$\Psi_{\text{ISATMOST1}}(v_I, d_1, .., d_m) = \begin{cases} 0, & \text{if } \begin{cases} v_I = \mathsf{false} \wedge \exists d_i \in \{d_1, .., d_m\} : d_i = \mathsf{true} \\ v_I = \mathsf{true} \wedge \forall d_i \in \{d_1, .., d_m\} : d_i = \mathsf{false} \\ v_I = \mathsf{true} \wedge \exists d_i, d_j : d_i = \mathsf{true} \wedge d_j = \mathsf{true} \wedge \ i \neq j \end{cases} \\ 1, & \text{otherwise} \end{cases} \tag{2.33}$$

Why not simply model labeled structures using multinomial variables? The primary motivation of decomposing to Boolean variables is one of convenience. In this thesis joint models are constructed by connecting Boolean variables from individual component models using soft Boolean logic factors (Sec. 2.4.3, pg. 43), but our intuition is that it is the structures of these models, and not the labels, that are highly correlated. For instance, in Chapter 4 we describe a joint model of phrase structure parsing and named entity recognition. Both the constituent spans of phrase structure trees and named entity spans are labeled structures: for a given pair of indices (i, j), a structure may or may not exist, and if it exists it should be labeled.

It is therefore more efficient to separate structure and label prediction, so that structure variables from opposing models can be directly coupled without the inclusion of the corresponding label variables. In principle it is just as easy to couple two multinomial variables, however it requires larger potential tables. There are also scenarios, as shown in Section 3.1.3, pg. 85), where this decomposition aids in coarse-to-fine strategies. We are able to efficiently assess the model's belief of particular labels, and attach higher-order factors to select sets of these variables without the need to include any extraneous ones.

- **EPLURIBUSUNUM** is a hard factor used to apply a constraint similar to ATMOST1, but in the situation where the indicator variable is Boolean and the set of dependent variables is represented by a single multinomial, which includes a null value. When the indicator variable is false, the factor constrains the multinomial to the null value, i.e., an EPLURIBUSUNUM factor contributes a potential of 0 when the indicator variable is false and the multinomial variable has any non-null value. It also contributes this potential when the indicator variable is true and the multinomial is null. It otherwise contributes a potential of 1.

$$\text{EPU}_{\Psi_A(v_I,x)} = \begin{cases} 0, & \text{if } \begin{cases} v_I = \textsf{false} \wedge x \neq \text{null} \\ v_I = \textsf{true} \wedge x = \text{null} \end{cases} \\ 1, & \text{otherwise} \end{cases} \tag{2.34}$$

Soft Factors

- **NAND** factors are soft factors used to learn a correlation between two or more variables. This factor fires when all of the tethered variables are true, contributing a potential derived from its feature function. It otherwise multiplies in a potential of 1.0 to the model score.

$$\text{NAND}_{\Psi_A(x_1,x_2)} = \begin{cases} \exp \sum_k \theta_k \cdot \vec{\xi}_{f_k}(\mathbf{x}_1, \mathbf{x}_2), & \text{if } x_1 = \textsf{true} \wedge x_2 = \textsf{true} \\ 1, & \text{otherwise} \end{cases} \tag{2.35}$$

- **IMPLIES** factors are a similar to NAND factors, but implement the logic of an implies, firing when the indicator variable (antecedent) is true while the remaining variables (consequents) are false.

$$\text{IMPLIES}_{\Psi_A(x_1,x_2)} = \begin{cases} \exp \sum_k \theta_k \cdot \vec{\xi}_{f_k}(\mathbf{x}_A, \mathbf{y}_A), & \text{if } x_1 = \mathsf{true} \wedge x_2 = \mathsf{false} \\ 1, & \text{otherwise} \end{cases} \tag{2.36}$$

Combinatorial Factors

Here we present a series of combinatorial syntax factors which efficiently constrain a set of Boolean variables representing trees in the chosen syntactic formalism. While some structures, like dependency trees and constituency trees, are theoretically interchangeable, within the modeling framework it is often preferable to choose one over another as some representations are inherently better suited for pairing with certain problems and their corresponding factor graph representations.

- **CKY-TREE** is a global combinatorial factor for imposing a tree-structured constraint for phrase structure grammar, computing outgoing messages in $O(n^3)$ time. It is discussed in Section 3.1.2 (pg. 72).
- **DEP-TREE** is a global combinatorial factor, as presented in [11], used to represent a syntactic analysis in dependency grammar. Like other syntactic combinatorial constraints, it is $O(n^3)$. Dependency grammar constraints are discussed in Section 3.2.2 (pg. 99).
- **PROJ-TREE** is a global combinatorial factor and a variant of DEP-TREE, which additionally requires that the graph be projective. A graph is projective if its edges do not cross.
- **SEMI-CRF** is a global combinatorial factor which imposes a 0th-order semi-Markov constraint [57] over a set of Boolean span variables. It is similar to the CKY-TREE factor in that its primary purpose is to prohibit spans from crossing, but it differs in that spans do not nest, and there is a maximum span width. These are useful properties for modeling named entity recognition. It is discussed in more detail in Section 4.3.1 (pg. 110).

Many of these factors are introduced in the work of [11], where they were applied to the task of dependency parsing. The novel factors are the CKY-TREE and SEMI-CRF factors, but it is primarily *how* we combine these components and utilize structured constraints over latent variables that is novel to this work. We now describe the construction of joint models using soft Boolean logical constraints, a key aspect of our approach.

2.4.3 Constructing Joint Models

Joint models couple component models for two or more tasks, allowing inference to be performed jointly. This is done in the hope that by sharing uncertainty between component models, the joint model will find a better global solution. In Chapter 1 we discussed many of the methods used for coupling two or more models: dual decomposition and ILP for joint decoding, and constrained latent representations and grammar augmentation for joint inference during training. Alternatively, factor graphs provide an attractive framework for constructing joint models. In a factor graph, variables connected via a factor are dependent. Therefore, a joint factor graph model can be constructed simply by connecting dependent variables from separate component models with a factor

that expresses the nature of their dependency. The question remains: what *kind* of factor should be used to connect these variables?

We argue that it is natural to conceptualize and express many general relationships between NLP tasks in terms of Boolean logic. In Chapter 4 we examine the relationship between named entities and phrase structure syntax, where both the entities and syntactic predictions consist of spans of words. In this domain we assume that the presence of a named entity from word w_i to w_j *implies* the presence of a corresponding syntactic span, composed of words w_i to w_j. In Chapter 6 we construct a joint model of semantic role labeling and dependency syntax. Similarly, we assume that a semantic relationship from word w_i to word w_j *implies* the presence of a syntactic relationship between these words. Due to the convenience of these types of constraints, we propose the use of Boolean logic to coordinate between component models. This logic can be easily implemented in a factor, provided that the variables which sit at the interface between component models are Boolean.

Natural language is full of ambiguity, and these constraints do not hold universally in any of the data sets we examine, but this is not a problem. Instead of restricting the potential function's range to $\{0, 1\}$, as in traditional Boolean logic, the factor's potential can range over the entire set of non-negative real numbers. The potentials of these *soft* Boolean logic factors are derived from an exponentiated sum of feature weights, allowing the model to learn a context-specific weight of coordination. In situations where variables should be coordinated (both true), the factor contributes a higher potential. A low potential can be contributed in situations which prove to be exceptions to the rule. This allows constraints to be conceptually simple, expressing the logic we believe reflects the interaction between these tasks a priori, while simultaneously being robust to noise. This is important not just for handling true linguistic exceptions, but also mismatches in annotation. In previous work [6], manual pre-processing of the data was required to eliminate instances where an NER span did not align with a syntactic span, an unnecessary requirement in our approach to joint modeling.

Coordinating Between Tagging and Parsing Models

Returning to the tagging model, our aim in constructing a joint model is to allow for the beliefs of tag variables t_i and t_j to influence and be influenced by the beliefs of the corresponding syntactic dependency variable. For instance, if the model's beliefs indicate tha there may be a syntactic dependency between two words, w_i and w_j, the model may assign a higher score to the configuration $t_i = V$ and $t_j = N$ than to $t_i = N, t_j = N$, as it is more likely for a verb and noun to exist in a syntactic dependency than a noun and a noun.

To model this phenomenon, assume a set of Boolean variables b_{i,j,T_i,T_j} each representing the presence of a tag combination T_i, T_j such that $t_i = T_i$ and $t_j = T_j$ when b_{i,j,T_i,T_j} = true. There is a bigram variable for every tag combination between every pair of words. Let $l_{i,j}$ be a set of Boolean variables representing the presence a syntactic dependency between word w_i and w_j when true (details regarding the dependency syntax formalism and its representation in a factor graph are provided in Section 3.2.2, pg. 99).

We can coordinate variables across component models using a set of soft Boolean logic factors. Let C_{i,j,T_i,T_j} be a set of soft NAND factors which connect each $l_{i,j}$ variable to all b_{i,j,T_i,T_j} variables. Though we do not know the precise nature of this relationship, the "soft" aspect of the coordinating factor provides the freedom to be rather noncommittal in the choice of coordinating logic by

learning a factor potential to capture the degree of coordination. A potential > 1.0 in a NAND factor will raise the true beliefs of connected variables, a potential < 1.0 will lower them. Let us examine how these coordinating factors affect the neighboring variable beliefs during inference, and the resulting marginal of the coordinating factor.

Assume two variables at the interface send the following (normalized) messages to a shared $\text{C}_{3,5,V,N}$ factor:

$$m_{l_{3,5} \to c_{3,5,V,N}}(l_{3,5} = \textsf{true}) = 0.7$$
$$m_{b_{3,5,V,N} \to c_{3,5,V,N}}(b_{3,5} = \textsf{true}) = 0.2$$

Thus the syntactic component of the model believes that a syntactic dependency exists from word w_3 to w_5, and the tagging component initially has a very low belief that word w_3 is a verb and w_5 is a noun.

Assuming that verbs and nouns are very likely to exist in a syntactic dependency, this intuition will enter the model in the form of a learned potential in coordinating factors. Assume coordinating factor $\text{C}_{3,5,V,N}$ is a NAND factor with a potential $\phi = 2.5$. In this situation the coordinating factor will drive the beliefs of both variables higher, as illustrated in Figure 2.8. Here the use of joint inference increases the marginal (true) belief of $b_{3,5,V,N}$ from 0.2 to 0.33 (normalizing from the resulting beliefs, $b_{3,5,V,N}(\textsf{true}) = 0.4$ and $b_{3,5,V,N}(\textsf{false}) = 0.8$). While this may not seem like a significant change, this is simply the result of one round of inference on a single portion of a highly connected model, and even the addition of this one factor may influence the individual tag variables, which connect to the $\text{B}_{3,5,V,N}$ factor, to more accurate predictions.

Other logics behave in different ways. If the connection between models was implemented as an IMPLIES factor, all model beliefs and parameters remaining constant, inference would push the two variable beliefs *apart* in order to adhere to the factor's logic.

How these feature weights are learned falls under the topic of optimization, and is discussed in the next section.

2.5 Parameter Estimation

In previous sections we discussed constructing graphical models of natural language processing problems, and how to perform inference over these models (exact inference in the case of trees or chains, and approximate loopy BP inference in graphs with loops). We also introduced a category of factors with log-linear feature functions known as *soft factors*, which in sum comprise the model's learnable parameters. But how does inference interact with training, and how do we optimize these feature weights to fit a data set? In this section we turn to optimization and describe how marginal beliefs can be utilized to learn a model from data. We will give special attention to learning in the presence of hidden structure.

2.5.1 Conditional Maximum Likelihood

The first step towards constructing an optimization method is to choose an objective function to optimize. Here we turn to maximum-likelihood estimation, which defines an objective function in

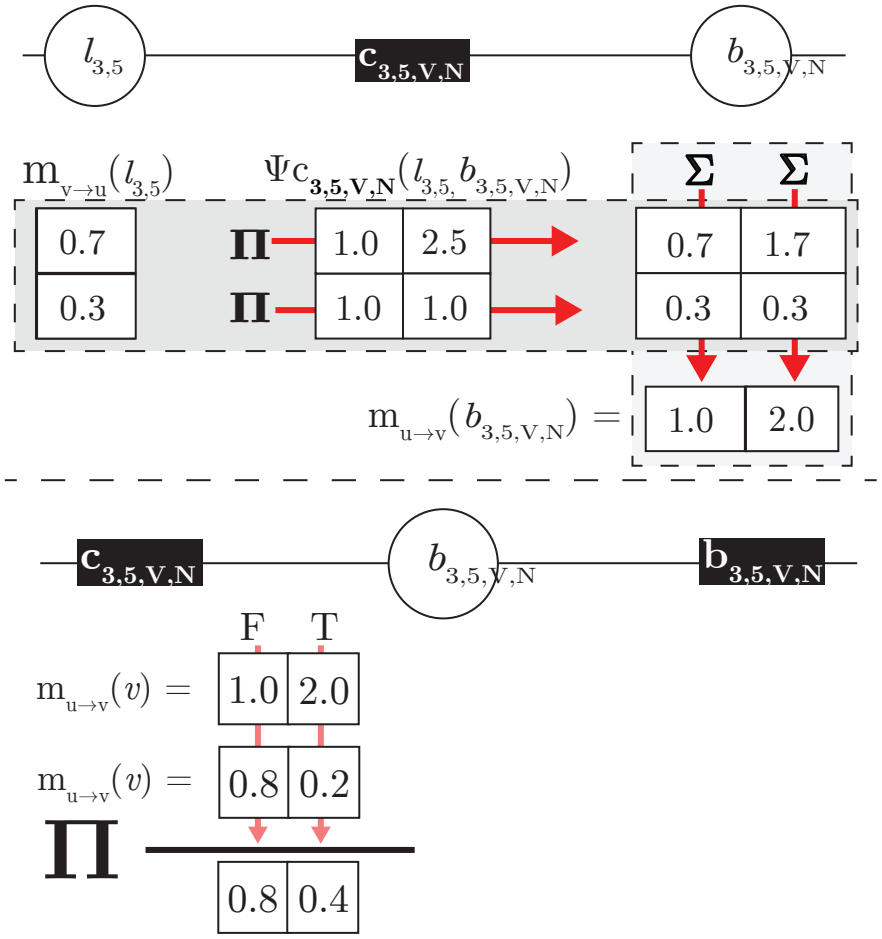


FIGURE 2.8: Coordinating models with soft NAND factors.

terms of the model parameters θ and the training data, $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$. Each $\mathbf{x}^{(i)}$ is a set of observations, and each $\mathbf{y}^{(i)}$ a set of labels. The goal of optimization is to find the parameter values which maximize the likelihood of the data, defined as:

$$\mathcal{L}(\theta|\mathcal{D}) = P(\mathcal{D}|\theta) \tag{2.37}$$

With the maximum likelihood estimate we seek to find the values of the model parameters which maximize the likelihood function L, maximizing the fit of the model – as defined by $\mathcal{L}$ – to the observed data. The maximum likelihood estimate provides a general framework for discussing estimation, and is well-defined for many families of distributions. Because the models we present are conditional, we will define the likelihood in terms of conditional probability. Assuming the observation data is independent and identically distributed, we can express the conditional likelihood:

$$\mathcal{L}(\theta|\mathcal{D}) \propto \arg\max_{\theta} \prod_{i} P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \theta) \tag{2.38}$$

In practice it is often more convenient to phrase this maximization in terms of *log likelihood*, ℓ,

replacing the product form with sums. Because the logarithmic function is monotonic, this yields an equivalent maximizer:

$$\ell(\theta|\mathcal{D}) = \arg\max_{\theta} \sum_{i} \log P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}, \theta) \tag{2.39}$$

Replacing the general conditional probability with the exponentiated conditional probability factorization for factor graphs (Eq. 2.6) allows the log and exponent to cancel, yielding the following conditional likelihood:

$$\ell(\theta|\mathcal{D}) = \arg\max_{\theta} \sum_{i} \sum_{f} \sum_{A} \sum_{k} \theta_k \cdot \xi_{f_k}(\mathbf{x}_A^{(i)}, \mathbf{y}_A^{(i)}) - \log Z(\mathbf{x}^{(i)}) \tag{2.40}$$

where $Z(\mathbf{x}^{(i)})$ is the global normalization term for example $\mathbf{x}^{(i)}$ (from Eq. 2.4, pg. 26). Maximizing the likelihood of the data is identical to finding the model parameters which maximize the probability of the model on that data. Here the model comprises a set of weighted feature functions over subsets of variables A.

2.5.2 Optimizing with hidden variables

The likelihood in Eq. 2.40 is only suitable when the set of model variables is fully observed in the training data. Many of the models we present treat syntactic structure as a latent variable, and thus we require a likelihood which incorporates a set of latent variables $Z \subset V$ (though we will henceforth reference latent variables only in assignments to them, $\mathbf{z}$, avoiding confusion with the partition function). However, it is straightforward to augment the previous likelihood to account for latent variables:

$$\mathcal{L}(\theta|\mathcal{D}) = \arg\max_{\theta} \prod_{i} \sum_{\mathbf{z}} P(\mathbf{y}^{(i)}, \mathbf{z}^{(i)}|\mathbf{x}^{(i)}, \theta) \tag{2.41}$$

where $\mathbf{z}^{(i)}$ is a configuration of latent variables compatible with the input, $\mathbf{x}^{(i)}$. Latent variables are neither observed, nor the target of supervised training. For this reason they are sometimes referred to as *nuisance variables*. We incorporate latent variables into the likelihood by *marginalizing* (summing over) all assignments to these variables, $\mathbf{z}$. Thus maximizing this likelihood searches for the best distribution over latent variables, where the worth of a latent structure is measured by how much it improves the probability of generating the correct target variables, $\mathbf{y}^{(i)}$.

Similar approaches have been used previously in NLP. For instance, [58] incorporate latent word senses into a model of parse reranking. Assignments to these latent variables are marginalized out to obtain a distribution over parse trees.

2.5.3 Stochastic Gradient Descent

Having defined an objective function for models trained in either a fully-supervised manner or in the presence of hidden variables, the remaining decision is how best to optimize this objective. For log-linear models, of the form we present, a common approach is the class of gradient-based methods. In these approaches the optimizer navigates the search space by computing the gradient of the objective function, and descending in this direction until a minimum is found. For convex

Algorithm 2 Stochastic Gradient Descent

function SGD($\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$) ▷ data pairs of observed variables $\mathbf{x}$, labels $\mathbf{y}$
  $\theta \leftarrow \mathbf{0}$ ▷ Initialize parameters
  $t \leftarrow 1$ ▷ blah
  repeat
    for $((\mathbf{x}, \mathbf{y}) \in \mathcal{D})$ **do**
      $G \leftarrow$ construct graph from $\mathbf{x}$, initializing beliefs w.r.t. θ
      $A^* \leftarrow$ variable assignment specified by observed labels $\mathbf{y}$.
      BP(G) ▷ Inference
      $\mathbb{E}_p[\xi_l(\mathbf{x}, \mathbf{y},)] \leftarrow \sum_k \log f_k(A^*)$ ▷ Computed from true labels
      $\mathbb{E}_p[\xi_l(\mathbf{x}, ,)] \leftarrow \sum_{f'} \sum_A m_{f_{k'} \rightarrow}(A) \nabla_\theta f_{k'}(A)$ ▷ Computed from marginals
      $\nabla_\theta L \leftarrow \mathbb{E}_p[\xi_{f_l}(x, y)] - \mathbb{E}_p[\xi_{f_l}(x, ,)]$ ▷ Gradient as specified in Eq. 2.51
      $\theta \leftarrow \theta - \eta^t \cdot \nabla_\theta L$ ▷ SGD update, Eq. 2.42
      $t \leftarrow t + 1$
    end for
  until max iterations exceeded or converged
  return w
end function

spaces, like those defined in the fully-supervised training via conditional likelihood, these methods are guaranteed to find the global optimum. When the model contains latent variables the space is non-convex, and gradient-based approaches may become stuck in local minima.

Gradient methods can be classified into batch and online algorithms. Batch algorithms, like the standard gradient descent, LBFGS [59], and quasi-newton methods process the entire data set before computing the gradient and performing a parameter update. Online algorithms, like stochastic gradient descent or the structured perceptron algorithm [60, 61], approximate the gradient using only some subset of the data, typically updating the model parameters after viewing each example.

Here we focus on stochastic gradient decent (SGD). Most of the data sets used for the tasks presented in this dissertation are large, and most of the joint models presented are complex enough that updating after each full pass through the data is quite cumbersome. Online learning methods will often make tens of thousands of updates per iteration, and reach good solutions after just a few full passes through the data [62].

Each SGD parameter update takes the form:

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} \nabla L_{\Theta}^{(t)} \tag{2.42}$$

where $\nabla L_{\Theta}^{(t)}$ is the approximation of the gradient produced from training example t, and η^t is the learning rate for example t. The learning rate can be reduced over the course of learning, as a function of t, to reduce the odds of over-stepping a minimum in the search space. SGD is guaranteed to converge to a single global minimum in convex spaces when paired with an appropriate and decreasing rate. When training with hidden variables, the parameter space is non-convex and SGD may find only a local minimum.

Fig. 2.5.3 describes the SGD algorithm in pseudocode. Placing the SGD update within the

larger algorithm highlights how inference is incorporated into training: inference yields factor marginals, and these quantities are in turn used to produce feature expectations, which specify the gradient.

2.5.4 Computing the Gradient

We now derive the gradient updates for the hidden conditional likelihood shown in Eq. 2.41. For any specific parameter θ_l in a log-linear term, the gradient of the log likelihood with respect to θ_l is:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \sum_i \frac{\partial}{\partial \theta_l} \log \sum_A \sum_f \sum_{\mathbf{z}} \exp(\theta_l \cdot \xi_{f_l}(\mathbf{x}_A^{(i)}, \mathbf{y}_A^{(i)}, \mathbf{z}_A^{(i)})) - \frac{\partial}{\partial \theta_l} \log Z(\mathbf{x}^{(i)}) \tag{2.43}$$

The gradient for any maximum likelihood training method is a ratio between two terms, each expressible as an expectation. For the numerator (the left term in the log-likelihood in Eq. 2.43):

$$\begin{aligned}
\frac{\partial}{\partial \theta_l} \log \sum_f \sum_A \sum_k \sum_{\mathbf{z}} \exp(\theta_l \cdot \xi_{f_l}(\mathbf{x}_A, \mathbf{y}_A, \mathbf{z}_A)) =& \frac{1}{\sum_{\mathbf{z}} P(\mathbf{y}, \mathbf{z}|\mathbf{x})} \sum_{\mathbf{z}} \frac{\partial}{\partial_{\theta_l}} P(\mathbf{y}, \mathbf{z}|\mathbf{x}) && (2.44)\\
=& \sum_{\mathbf{z}} P(\mathbf{y}, \mathbf{z}|\mathbf{x}) \log \frac{\partial}{\partial_{\theta_l}} P(\mathbf{y}, \mathbf{z}|\mathbf{x}) && (2.45)\\
=& \mathbb{E}_p[\xi_{f_l}(\mathbf{x}, \mathbf{y},)] && (2.46)
\end{aligned}$$

This follows through an application of the chain rule, and the useful identity $\frac{\partial f}{\partial \lambda} = f(\lambda) \frac{\partial \log f}{\partial \lambda}$ [54]. The resulting expectation is of parameter θ_l under the empirical distribution. When there are no latent variables this expectation can be deterministically computed from the data (1 if the predicted structure is present in the gold analysis, 0 if not.).

Differentiating the denominator (the right term in the log-likelihood in Eq. 2.43):

$$\begin{aligned}
\frac{\log Z(\mathbf{x})}{\partial \theta_l} =& \frac{\partial}{\partial \theta_l} \log \sum_{\mathbf{y},\mathbf{z}} \exp(\theta_l \cdot \xi_{f_l}(\mathbf{x}, \mathbf{y}, \mathbf{z})) && (2.47)\\
=& \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{x},\mathbf{y}} \exp(\theta_l \cdot \xi_{f_l}(\mathbf{x}, \mathbf{y}, \mathbf{z})) \frac{\partial \theta_l \cdot \xi_{f_l}(\mathbf{x}, \mathbf{y}, \mathbf{z})}{\partial \theta_l} && (2.48)\\
=& \sum_{\mathbf{y},\mathbf{z}} p(\mathbf{y}, \mathbf{z}|\mathbf{x}) \xi_{f_l}(\mathbf{x}, \mathbf{y}, \mathbf{z}) && (2.49)\\
=& \mathbb{E}_p[\xi_{f_l}(\mathbf{x}, ,)] && (2.50)
\end{aligned}$$

Importantly, this factors into the difference between two expectations:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \mathbb{E}_p[\xi_{f_l}(\mathbf{x}, \mathbf{y})] - \mathbb{E}_p[\xi_{f_l}(\mathbf{x}, ,)] \tag{2.51}$$

The first, an expectation of the empirical distribution under the model, the second, the model's expectation. This is noteworthy for two reasons. First, when the feature expectations of the model

perfectly match the distribution of the data (as viewed through the features) these expectations cancel and there is no parameter update. This is the optimal solution the model can reach with respect to the training data. Second, both expectations are computable with belief propagation [63]. In the case where training is fully supervised the first term reduces to $\xi_{f_l}(x_i, y_i^*)$, where the feature counts of the data are set deterministically from the provided labels.

2.5.5 Training with Latent Variables

We have now shown how to construct joint models and discussed how to train them in the fully supervised case. However, this requires jointly annotated training data, with annotations for all tasks – an unlikely scenario. Here we present a method for training factor graphs not just in the presence of predicted variables, but in the presence of *nuisance* variables: variables which help support end task predictions but are themselves inconsequential. In this training procedure we observe only the annotations associated with the end task, and the goal is to the propagate the error signal (from end task annotations) to help guide the induction of useful distributions over latent structures. Parameters for latent variables are optimized towards improving the model's performance on end task predictions.

Despite the ambitious nature of these goals, training in this fashion can be done as a very straightforward extension to the previously-discussed inference and training methods. In Sec. 2.51 (pg. 49) we described the SGD update as being expressible as the difference between two expectations. In the fully supervised case we observed that the first expectation, $\mathbb{E}_p[\xi_{f_l}(x_i, y_i^*,)]$, could be defined solely by the data and the associated true labels, without the need for inference. This was possible because this is an expectation over features when variables are clamped to their true value, and we observe the true value for *all* variables. In the case of hidden variable training we must rely on inference to compute this value.

This makes training a two-step process. In the first step we perform standard sum-product inference on the model, producing the marginals necessary to compute the denominator of the gradient. Instead of relying on data to compute the numerator, a second round of inference is performed, beginning by setting observed variables to their true values. This produces the marginalization described in Eq.2.41, and the denominator of the gradient.

Returning to the tagging example, instead of observing the true value for the now unobserved coordinating factor, $c_{3,5,V,N}$, we substitute the factor marginal computed via inference on the clamped graph. The marginal for a coordinating factor will be influenced by one variable held fixed to its true value, the end task, and one latent variable. Assuming its marginal is 0.7, the partial derivative is:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = 0.7 - 0.8 \tag{2.52}$$

Leading to the following parameter update:

$$\theta^{t+1}_{f(\text{`}VB+NN')} = \theta^{t}_{f(\text{`}VB+NN')} - \eta^t * -0.1 \tag{2.53}$$

moving the parameter in the opposite direction as the supervised update. The resulting feature weights for syntax and coordinating factor features may be significantly different, as they are here,

than when trained from the true error signal. However, the model learned in this manner has still honed a distribution over latent variables which supports the end task.

This approach has been previously utilized in scene detection for computer vision [**?**] to train in the presence of missing data labels. Variables whose labels are missing are marginalized over so that the model can be trained to maximize the log-likelihood of existing ones. There are many similarities between this work and our own: the desire to incorporate both local and global information, the use of loopy belief propagation (two times) to compute marginal beliefs, and the use of SGD for parameter optimization. Our work is in a different domain - NLP - and differs most notably in that we have strong priors over what kinds of latent structures are permissible, and constrain these variables using combinatorial factors.

2.5.6 Learning to Coordinate Between Models

Let us examine the process of computing the gradient and updating a set of parameters associated with the factors which coordinate between models. Recall that a joint model of tagging and parsing could be constructed such that the interface between models involves sets of bigram tag dependencies and syntactic dependencies. A tag transition from T_k to T_l between words w_i and w_j is captured by Boolean variable b_{i,j,T_k,T_l}, and a syntactic dependency between word w_i and w_j is captured by Boolean variable $l_{i,j}$. These variables can be connected via a soft Boolean logic factor, in our case a NAND factor, $\text{C}_{3,5,V,N}$, which learns a weight of coordination between variables from the two models via a set of features.

Ideally the model would learn that a verb is commonly the syntactic head of a noun. In the sentence "*The cat scratches the man*", both the word *scratches* and the word *man* may be either a noun or a verb. In this case *scratches* is the syntactic head of *man*, and this additional evidence could help the model correctly disambiguate the part-of-speech of both words. From the model standpoint, this occurs when the $\text{C}_{3,5,V,N}$ factors' *'VB+NN'* feature weight is sufficiently high, and the feature weight for *'NN+NN'* is sufficiently low.

After inference converges, the model can be queried to collect factor marginals. The marginal for the coordinating factor, $\pi(c_{3,5,V,N})$, can be computed as follows:

$$\begin{aligned}\pi(c_{3,5,V,N}) = \phi(c_{3,5,V,N}) \\ * m_{l_{3,5} \to c_{3,5,V,N}}(l_{3,5} = \textsf{true}) \\ * m_{b_{3,5,V,N} \to c_{3,5,V,N}}(b_{3,5,V,N} = \textsf{true})\end{aligned}$$

We will assume the following factor belief, continuing from discussion of joint inference in the joint tagging and parsing model (Section 2.4.3, pg. 44):

$$\pi(c_{3,5,V,N}) = 0.33$$

Training attempts to minimize the divergence between the model's expectation and the empirical expectation, shown in Eq. 2.51. Because this factor's potential over a configuration of variables that is observed in the training data, empirically it's marginal should be 1.0.

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = 1.0 - 0.33 \tag{2.54}$$

The update to the parameter vector, with learning rate η_t:

$$\theta^{t+1}_{f(VB+NN)} = \theta^{t}_{f(VB+NN)} + \eta^t * 0.67 \tag{2.55}$$

Everything else remaining constant, during the next iteration of training the model the new feature weights will result in a new potential function for factor $c_{3,5,V,N}$, the resulting marginal for this factor will become higher, and the update smaller, indicating that the model distribution has become more like the empirical distribution of the training data.

2.5.7 Regularization

Perfectly matching the empirical distribution is a theoretically sound goal when the training data accurately reflects the test data, but in practice this assumption is rarely met, making the objective less than ideal. In general the models presented here will have an abundance of features: in addition to the standard set of features required for training each component model independently, the joint model will require an extra set of features for learning the interactions between these components. To complicate matters, when learning with latent variables it is difficult to gather useful feature statistics prior to training, making it difficult to use common feature pruning techniques. As a result, some of the models presented in later sections will have hundreds of millions of features. Given the large number of parameters overfitting becomes a concern. As the training procedure overfits the model to the data, parameters can be driven towards infinity, creating numerical issues.

We can incorporate an additional constraint into the objective function to penalize weights that are too far from zero, applying a greater penalty the further the deviation, and keeping many values close to zero:

$$F = \sum_i \log \sum_{\mathbf{z}} P(\mathbf{y}^{(i)}, \mathbf{z}^{(i)} | \mathbf{x}^{(i)}) + \frac{1}{2\sigma^2} \sum_l \theta_l^2 \tag{2.56}$$

where $\mathbf{y}^{(i)}$ is the correct predicted structure for the ith training example, $\mathbf{z}^{(i)}$ is the set of hidden structures compatible with that structure, and σ^2 is a variance parameter. This particular form of regularization is known as L_2 regularization. Regularized optimization has connections to Bayesian MAP estimation [64], and an L_2 regularization can be considered as a Gaussian prior over weights: when the variance is high and the distribution flat the parameter weights are given more freedom to wander. When the variance is low the values will tend to cluster close to the mean of zero. For stochastic gradient descent this equates to multiplicatively applying a constant weight decay over the entirety of the parameter vector upon each update.

As noted in the literature, when the number of model parameters greatly outnumbers the number of training examples, as will often be the case in our work, there are theoretical motivations to instead pursue L_1 regularization, as the sample complexity (a measure of the number of training examples required for satisfactory learning) scales at an asymptotically slower rate in comparison to L_2 [65]. While L_1 regularization more directly enforces sparsity by forcing many parameters to zero, in practice L_2 is often sufficient to prevent overfitting and numerical problems.

2.6 A Hidden Syntax Approach to Tagging

In this chapter we have presented the necessary theoretical background required for constructing, reasoning with, and training joint factor graph models for NLP tasks. Each topic is grounded in a running example, and discussed in the context of a joint model of parsing and part-of-speech tagging. In this section we revisit this example, presenting the complete tagging model and results on a pair of standard data sets.

2.6.1 The Role of Syntax in Tagging

While the relationship between syntax and morphology (which includes part-of-speech) is often over-simplified in the NLP community, where they are commonly treated as separate tasks, the two are inherently linked. Morphological information is useful for parsing, but morphological attributes are simultaneously constrained by these same syntactic dependencies. [66] points out that these two information sources exist in a competitive relationship, each vying for roles in the process of linguistic expression. Languages with relatively free word-order often have rich systems of morphology that facilitate capturing long-range dependencies between distant words, and languages with relatively fixed word order can forgo rich morphology by having more predictable local dependencies. [67] show that for English, a language which is considered to have a more fixed word order, a majority of syntactic dependencies exist between adjacent words.

We find empirical evidence of this trade-off in a study of conditional tag probabilities in English and Latin text. The heatmaps in Figure 2.6.1 depict the conditional probability of an arbitrary tag t_i, conditioned on either the preceding tag, t_{i-1} (*top*), or the tag of the syntactic head word, $t_{head(i)}$ (*bottom*). For English (*left*) the tag of interest is part-of-speech. For Latin (*right*) we focus on morphological case, which is often described as being syntactically-assigned. Cells that are more heavily shaded indicate high conditional probabilities, and instances where the conditioning tag is highly informative of current tag.

For English, knowing the tag of the preceding word, t_{i-1}, often narrows the choice of t_i down to just one or two candidates. In Latin these adjacent tag statistics are not as informative, as a single tag, the null case marking (—) is the most likely tag in any context. Ignoring this tag, the conditional distributions are less concentrated, implying that knowledge of the previous tag is less informative for this task in Latin than in English.

Conditioning on syntactic dependencies tells a very different story. In English, the distributions remain about equally as concentrated, providing more confident information regarding some tag combinations, and less about others. However, in Latin a statistical correlation that was subtly present before is now strikingly clear: given the tag of the syntactic head, the most probable tag will almost always be the identical tag. A morphological phenomenon, noun-modifier agreement, is the underlying cause. In Latin, adjectives must agree with the nouns they modify in case, gender, and number, making syntactic information especially useful to the tagging task. [7] have shown previously that a joint model of these tasks can improve the predictions of morphological attributes (and, to a lesser extent, parse accuracy), substantiating this claim.

If syntax is a useful cue for tagging, why is it not used more often? There are two likely justifications for eschewing models of syntax when performing any morphological tagging. First, a disproportionate amount of the research community's attention has focused on part-of-speech

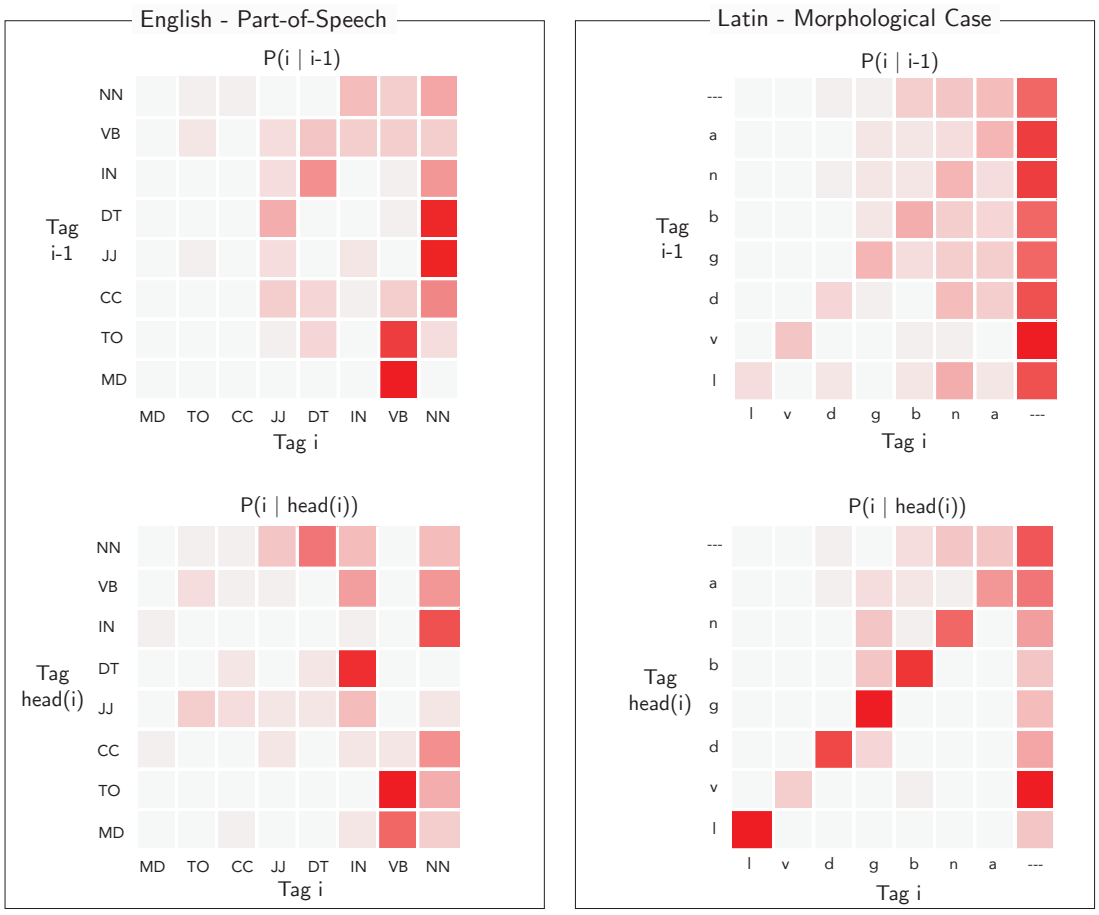


FIGURE 2.9: Heatmaps of conditional tag probabilities. Each cell represents the probability of the x-axis tag conditioned on the y-axis tag (rows normalize), with red indicating a high probability. Heatmaps at the top of the figure condition on the tag of the previous word, heatmaps at the bottom of the figure condition on the tag of the syntactic head.

tagging, and subsequent tasks like parsing, *for English*, where such dependencies are not as useful. Second, part-of-speech tagging is often considered to be a preprocessing step for tasks like parsing, and so in the most common scenario syntactic information is unavailable to the tagger. Even when inducing a model of syntax in a fully unsupervised manner, part-of-speech tags are usually assumed as input to the model, or are learned through distributional cues in an independent induction process [68].

2.6.2 A Joint Model of Tagging and Syntax

In previous sections we introduced this model piecewise. We now present the complete joint model, described by the components below where T is the tagging label set, and n is the length of the sentence. As in future sections, variables and factors are named to help distinguish their roles in the graph. Variable names are shown in $italics$, factors in SMALL CAPITALS:

Tagging Components

- Let $Unigram(i, T_k), k \in |T|$ be $O(n|T|)$ Boolean unigram tag variables, with corresponding unary factors. $Unigram(i, T_k) =$ true if and only if word w_i has the tag T_k.
- Let UNIGRAM$(i, T_k) k \in |T|$ be $O(n|T|)$ factors, each connected to the corresponding $Unigram(i, T_k)$ variable. Each UNIGRAM(i, T_k) factor has a pair of features: the word and tag combination, w_i + T_k, and the tag, T_k.
- Let EXACTLY1(i) be $O(n)$ combinatorial logic factors which constrain the set of $Unigram(i, T_i)$ variables at each i, forcing exactly one variable to have a true value. Each word receives one tag.
- Let $Bigram(i, j, T_k, T_l)$ be $O(n^2 \times |T|^2)$ Boolean bigram tag variables each representing whether words w_i and w_j have tags T_k and T_l respectively. There are no corresponding unary factors for these variables.
- Let BIGRAM(i, j, T_k, T_l) be $O(n^2 \times |T|^2)$ soft NAND factors which coordinate between $Unigram(i, T_k)$, $Unigram(j, T_l)$, and $Bigram(i, j, T_k, T_l)$ variables. Each BIGRAM(i, j, T_k, T_l) factor has a pair of features: the combination of words and tags, w_i + w_j + T_k + T_l, and the pair of tags, T_k + T_l.

Syntactic Components

- Let $Link(i, j), 0 <= i <= n; 1 <= j <= n$ be $O(n^2)$ syntactic dependency variables. $Link(i, j) =$ true iff there is a syntactic dependency from word w_i to w_j.
- Let LINK$(i, j), 0 <= i <= n; 1 <= j <= n$ be $O(n^2)$ factors which connect to corresponding $Link(i, j)$ variables. These factors are associated with many features, which are listed in [69].
- Let DEP-TREE be a single global combinatorial factor connecting to $O(n^2)$ syntactic $Link(i, j)$ variables and enforcing a hard constraint, requiring the connected variables to form a valid syntactic structure. In this instance a structure is valid if it is a non-cyclic directed graph. This syntactic formalism is discussed in detail in Section 3.2.2 (pg. 99).

Connecting Components

- Let CONNECT(i, j, T_k, T_l) be $O(n^2 \times |T|^2)$ logical factors implementing a soft NAND logic, and connecting $Bigram(i, j, T_k, T_l)$ and $Link(i, j)$. These factors fire when the values of both variables are true, contributing a non-negative potential, ϕ, to the model score. The variables are more likely to be true as ϕ increases beyond 1, and other configurations more likely as ϕ moves closer to 0.

The foundation of this model is a typical linear-chain CRF tagging model, with one exception: in a tagging model the choice of tag at each i is commonly represented with a multinomial. Here each multinomial variable has been decomposed into a set of Boolean variables. This necessitates some additional logic, and EXACTLY1 factors are utilized to constrain each set of Boolean variables such that *exactly one* variable is true.

This strategy extends to bigram dependencies. In this model, bigram dependencies between pairs of tag variables are modeled with a set of Boolean variables, and are coordinated with the

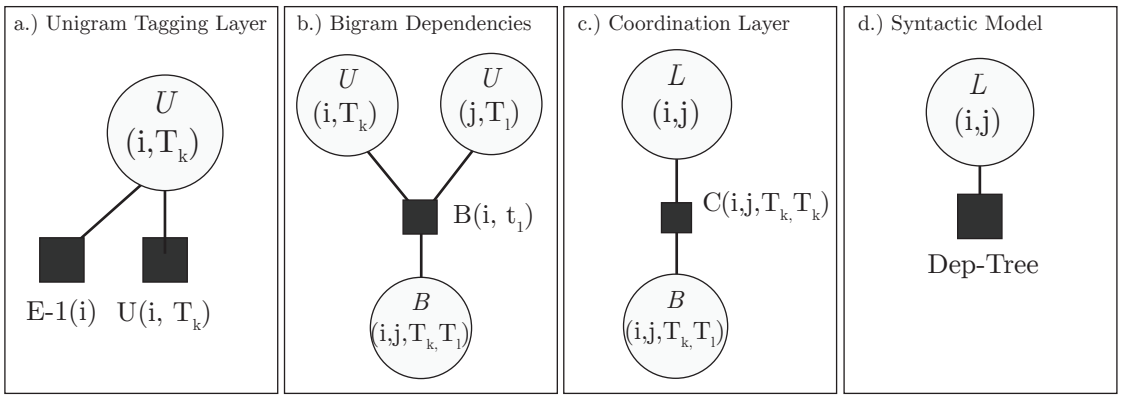


FIGURE 2.10: A joint model of tagging and parsing. The unigram (a.) and bigram (b.) sections constitute a typical bigram tagging model, albeit expressed entirely using Boolean variables. The syntactic representation is constrained by a combinatorial DEP-TREE factor, and syntactic variables coordinate with bigram tag variables via a set of soft logic factors (c.). Names have been abbreviated.

corresponding unigram tag variables with soft NAND factors. Here the motivation for decomposing the model may become clear. By representing bigram dependencies as Boolean variables, the interface between the two models consists solely of Boolean variables, allowing corresponding sets of these variables to be coordinated via soft Boolean logic factors.

In places where syntactic variables are true, the Bigram variables for likely syntactic tag combinations will be reinforced. A verb is frequently the syntactic head of a noun, so the (true) belief of $Bigram(i, j, V, N)$ will be increased when $Link(i, j)$ is true. The beliefs of bigram variables propagate to unigram tag variables, which are queried at decoding time to produce the predicted output.

2.6.3 Experimental Design and Results

Data

Models are evaluated on two languages: English and Latin. The English data is derived from a subset of the standard Penn Treebank Wall Street Journal corpus [70], converted to dependency syntax, as outline by [71] for use in the 2007 CoNLL shared task [72]. For Latin we make use of the Latin portion of the Ancient Greek and Latin Dependency Treebanks [73]. This corpus is an ongoing effort, and consists of roughly 3500 Latin sentences at the time of processing. We reproduce the training and test splits proposed in previous work [7].

Model Configurations

We evaluate five separate models:

- **Unigram**
  The unigram model consists only of $Unigram(i, T_k)$ variables, a corresponding set of unary factors, and a set of EXACTLY1 factors to force one tag variable to be true for each position $i \in n$.

- **Bigram**
  The bigram model extends upon the unigram model with bigram variables and coordinating factors.

- **Joint**
  In the joint model connects bigram variables to corresponding syntactic dependency variables using Boolean logic (NAND) factors. The correct trees are observed during training, but not testing time. [7] present a similar joint model, but one which uses the logic factors to explicitly model agreement phenomena between words.

- **Oracle**
  The oracle model relies on gold, or *oracle* syntax. The full joint model, as described in Figure 2.10, is constructed, but during inference the syntactic variables are clamped to their true values. This is done during both training and testing. It is meant to serve as a ceiling for the performance of the joint model if the joint model could perfectly predict trees at test time, though there is no guarantee that these trees are ideal for the task.

- **Hidden**
  In this scenario the full joint model is constructed, but all syntactic components (and consequently all coordinating factors) are treated as unobserved. The marginalization-based training method (Section 2.5.5, 50) induces a distribution over latent syntactic variables to improve the conditional maximum likelihood on the data.

Design

Each model was trained using 500 iterations of gradient descent, with learning rate $\eta = 0.1$, and using L2 regularization as described in Section 2.5.7 (pg. 52). We impose a 50 iteration maximum of belief propagation inference, which is exact in the UNIGRAM and BIGRAM models and approximate in all others. In models with syntactic components, we alternate between forward-backward sweeps along the chain of tag variables, and inference across the syntactic and connection components. The model is decoded by taking the tag T_k of the $Unigram(i, T_k)$ with the highest true belief, for all $i \in n$.

The features associated with each UNIGRAM(i, T_k) factor are simply the tag (T_k) and the word concatenated with the tag ($w_i + T_k$). The only feature for the BIGRAM(i, j, T_k, T_l) factors is a concatenation of the tags involved ($T_k + T_l$). This type of feature is also used in the CONNECT factors. In this case there is no need to use a feature which specifies an i, j, tag at position i, tag at position j, *and* whether the syntactic arc is present, because the only potential in the CONNECT factor that is derived from features is for when both variables are true. However we do differentiate between features for CONNECT and BIGRAM factors, i.e., features are mapped to different parameters in the model.

Results

The results of these experiments are shown in Table 2.1. We find that the BIGRAM model provides significant gains over the UNIGRAM tagging model in both languages. In English, where sequence models have a long and successfully history, we find that bigram dependencies between adjacent

	English	Latin
Unigram	91.5	95.3
Bigram	96.9	96.2
Joint†	96.4	96.6
Oracle††	96.7	96.7
Hidden	94.3	96.1

TABLE 2.1: Tagging accuracy for English and Latin. When applied to the English data, the model which uses local (adjacent) dependencies achieves the highest accuracy. For morphological case prediction in Latin, syntactic information produces superior results. Models which observe syntactic annotations during training are marked with a † symbol, and models which observe syntactic annotations during both training and testing with a †† symbol.

words are more useful than dependencies between words in syntactic relationships – both when using gold standard syntax, and when predicting syntactic variables. However, for morphological case prediction in Latin we find that the JOINT and ORACLE, which are trained from gold syntactic dependencies, provide more useful information than local bigram dependencies alone. This supports the general claim that syntactic information is a potentially useful source for information for tagging tasks in morphologically rich languages.

In this example we find that the HIDDEN model is not able to induce a latent syntactic representation that improves tagging accuracy over the BIGRAM model. This may be attributed to many potential causes (increased model complexity, a non-convex parameter space, under-training, or the lack of a sufficiently strong signal from the tagging annotations), but we will show in the remaining sections that for many NLP tasks an induced syntax can provide a significant improvement over models which do not utilize syntactic information. In many cases these latent syntax joint models often rival the performance of systems which make use of gold trees or trees produced by a state-of-the-art parser.

2.7 Conclusions

Our hypothesis is that coupling a model of syntax to a downstream NLP task and training the models jointly will improve performance on that task. Further, we believe that in many instances where data limitations would prevent fully supervised training of joint models, useful distributions over latent syntactic structure can be induced from the limited annotations that are available.

In this chapter we introduced the basic theory and techniques used repeatedly throughout the dissertation to accomplish these goals. In Section 2.2 we introduced factor graphs, and described the set of factors that serve as the building blocks for the complex, joint models presented in the following chapters. In Section 2.3 we described the message-passing inference techniques that underly exact inference strategies for many common NLP models (HMMs, PCFGs, etc.). The same inference strategy, when performed repeatedly, provides a useful approximate method for inference in cyclic graphs. This is particularly important when modeling two tasks jointly, as the dependencies between component models often create cycles in the graph.

However, structured constraints – like those necessary for representing syntax –can introduce

an exceedingly large number of factors into the graph, creating large cliques of densely connected variables, and hindering inference. We discuss combinatorial factors (Section 2.4) as a way of introducing special-purpose algorithms into the inference procedure to circumvent these issues.

In Section 2.5 we discuss how to optimize model parameters on a set of training data using gradient-based methods, in both the fully supervised case and when there is no syntactic training data available (i.e., in the presence of latent variables). We conclude by illustrating the use of these techniques to create and train a joint model of tagging and parsing (Section 2.6), inducing a latent syntactic representation to influence tagging performance.

In the following chapter we delve in greater detail into syntax, and the combinatorial factors used to model syntactic constraints. We introduce two syntactic formalisms: phrase structure grammar and dependency grammar. For phrase structure grammar, in which a syntactic analysis is represented by a tree, we provide a novel combinatorial factor for efficiently prohibiting invalid structures in the model. We then show how such a factor can be utilized at the core of a standalone *grammarless* phrase structure parsing model, and show performance comparable to well-established parsers. For dependency grammar we review the work of [11] which introduces a similar combinatorial factor for constraining syntactic variables to form directed graphs. These factors are the principle components of the latent syntactic representations we use throughout the thesis.

3

Factor Graph Representations of Syntax

Syntactic structure is essential to the understanding of natural language. Consider the sentence, "*The cat scratched the man with claws*". There are at least two equally valid syntactic analyses for this sentence (Fig. 3.1), with each analysis corresponding to a different interpretation of the events[1]. If the prepositional phrase attaches high, to the verb *scratched*, the sentence has a semantics akin to *The cat scratched the man with its claws*. If it attaches low, to the NP headed by *man*, it should be interpreted as *The cat scratched the man who has claws*. Syntactic analysis removes some of the ambiguity inherent in language and is therefore crucial for capturing deep linguistic understanding. Knowledge of the data's syntactic structure has long been considered a prerequisite for NLP tasks like semantic role labeling [19] and relation extraction [75], and methods which incorporate syntactic features or syntactic structure are becoming increasingly more common [6].

In this thesis we explore the use of syntax as a form of latent scaffolding: structure that serves little purpose in and of itself, but provides a useful platform from which end task predictions can be more effectively made. In order to achieve this goal we rely on a marginalization-based approach to training (Section 2.5.2, pg. 47) to induce a syntactic structure specific to the end task. This removes the need for syntactic annotations, but requires that both the syntactic and end task models be represented within the same framework. Here the chosen framework, factor graphs, are capable of representing a wide variety of models commonly used in NLP, but are not well-suited for efficiently representing the structured constraints found in syntax.

In this chapter we discuss graphical model approaches to representing two major syntactic formalisms: *phrase structure grammar*, in which a syntactic analysis must form a tree, and *dependency grammar*, in which a syntactic analysis must form a directed acyclic graph (DAG). Naive approaches to representing either can lead to a troublesome, and sometimes exponential growth in the number of factors necessary to rule out invalid structures. To circumvent this we introduce to

[1] A syntactic theory does not determine how a sentence should be interpreted, but many compositional theories of semantics have a strong correlation with syntactic structure (see *rule-to-rule hypothesis*, [74, pg. 5, 26]).

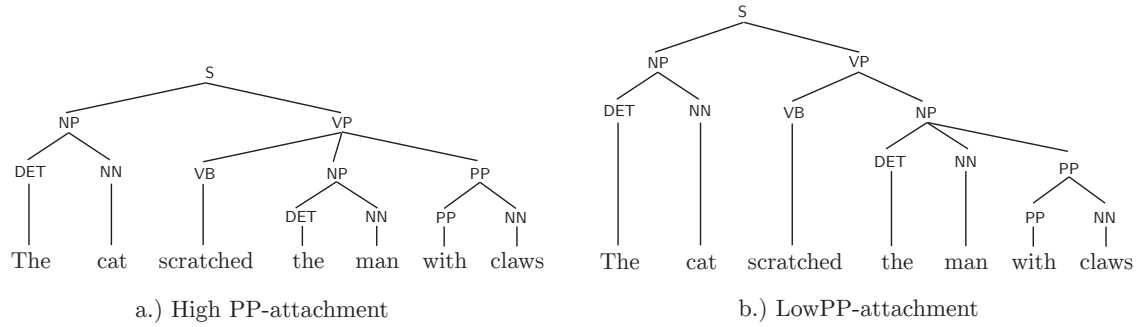


FIGURE 3.1: A phrase structure tree. High PP-attachment (*left*) vs. low PP-attachment (*right*).

combinatorial factors which override the default message passing algorithm and calculate the outgoing factor messages in a different and often more efficient manner. A single globally-connected factor can efficiently ensure, through dynamic programming algorithms, that the syntactic variables represent only valid parses. Thus the syntactic structure induced by these models obey general syntactic structure constraints, but may not correspond well to linguists' notion of syntax nor capture the subtleties of phenomena like the previously-discussed PP-attachment. However, it may nonetheless resolve ambiguity and constrain the problem space in a manner befitting the end task.

The remainder of this chapter is organized as follows. In Section 3.1 we discuss the context-free grammar models that constitute the dominant approach in the field, and the Inside-Outside algorithm for parameter estimation in these models. Much of the theory introduced in this section underlies our own work. In Section 3.1.2 we present a factor graph representation of phrase structure trees. Efficient inference requires the introduction of a combinatorial tree constraint factor, CKY-TREE, which can compute outgoing messages in $O(n^3)$ time (Section 3.1.2). It is possible to use these components as an unlabeled parser, producing a tree structure with no tree labels. In Section 3.1.2 we present a set of modeling extensions for performing labeled phrase structure parsing. In Section 3.1.3 we propose a method for introducing grammatical rules into the model, coupled with a perceptron-based learning method for estimating rule weights. We evaluate the performance of these models in Section 3.1.4,where we show that they perform comparably to more established state-of-the-art parsers while remaining asymptotically faster. In the following chapters we utilize this phrase structure representation to construct joint models for named entity recognition (Chapter 4) and relation extraction (Chapter 5)

Section 3.2 summarizes the contributions of [11], who present a combinatorial factor for efficient dependency parsing within a factor graph model. We rely on this approach in later chapters when constructing joint models of relation extraction (Chapter 5) and semantic role labeling (Chapter 6).

3.1 Phrase Structure Parsing

Phrase structure grammar, a syntactic formalism perhaps most commonly associated with [76], is built around the notion of a *constituent*: a contiguous span of words that functions as a single syntactic unit within a hierarchical structure. Constituents nest within one another, and the resulting structure is known as a phrase structure tree. A pair of phrase structure trees is shown in Fig. 3.1.

The spans of words which constitute a constituent often align closely to the desired predictions in NLP tasks. For instance, it is exceedingly common for named entities (Chapter 4) to correspond to noun phrases, and the words at constituent boundaries play an important part in constructing a semantic role labeling analysis (Chapter 6). Therefore in order to perform many NLP tasks at a state-of-the-art level, it is necessary to first produce a phrase structure tree using a statistical parsing model.

Statistical methods revolutionized the field of natural language processing, leading to significant and rapid advances in the accuracy of probabilistic phrase structure parsers. Aided by the development of large, tree-annotated corpora [70], computers began to produce high-quality parses of natural language for the first time. A majority of these approaches assume that a sentence's syntactic structure can be described by a set of weighted production rules collectively known as a probabilistic context-free grammar (PCFG). We will now discuss how probability distributions are defined over trees using PCFGs, and how the parameters of these models can be efficiently learned. While our own approach to phrase structure parsing is unique in that it does not require a grammar, it nonetheless shares many of the same underlying concepts.

A probabilistic parser defines a distribution over trees and parses a sentence s by finding the tree t which maximizes this distribution. If the parser is a generative model, this distribution will be the joint distribution, $P(t, s)$. If the parser is a conditional model, this distribution will be the conditional distribution, $P(t|s)$. As [77] notes, maximizing the joint distribution is equivalent to maximizing the conditional distribution:

$$t_{Best} = \arg\max_t \frac{P(t, s)}{P(s)} = \arg\max_t P(t|s) \tag{3.1}$$

The question then becomes one of how to define $P(t|s)$. For a majority of statistical parser research, the answer is to specify this distribution using a PCFG. Formally a PCFG G is a 5-tuple:

$G = \{V, \Sigma, R, SD\}$

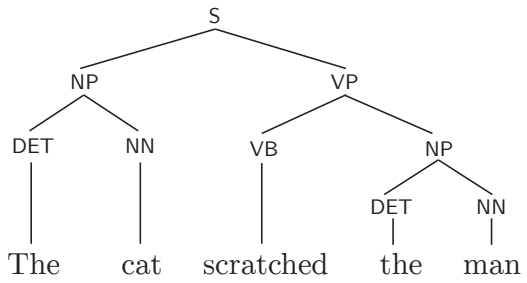
1. V is a finite set of nonterminal symbols. In the context of natural language these represent syntactic categories (i.e., noun phrase, verb phrase, etc.). A nonterminal is characterized by its ability to be rewritten in accordance with a set of production rules, R.

2. Σ is a finite set of terminal symbols, disjoint from V. In the domain of natural language, the terminal set typically comprises the words of the language, though they may also refer to part-of-speech tags if tags are observed. Terminal symbols cannot be rewritten.

3. R is a finite set of rules of the form $V \rightarrow \{V \cup \Sigma\}^+$, mapping a single nonterminal symbol to a sequence of nonterminal and/or terminals. As it pertains to trees, V corresponds to a parent node, and $\{V \cup \Sigma\}^+$ a sequence of children. The Kleene plus ($^+$) requires that at least one nonterminal be present in the right-hand side of the rule.

4. S is a start symbol, $S \in V$, which is used to initially represent the sentence. It is transformed during parsing by the application of rules in R.

5. ϕ, is a function of the form $R \rightarrow [0, 1]$, assigning a probability to each rule in the grammar.



a.) A Phrase Structure Tree

$V = \{\text{S, NP, VP, DET, NN, VB}\}$

$\Sigma = \{\text{The, cat, scratched, the, man}\}$

$S = \text{S}$

$R =$

1.0	S	⟶	NP VP
1.0	NP	⟶	DET NN
0.6	VP	⟶	VB NP
0.4	VP	⟶	VP PP
0.3	DET	⟶	The
0.2	NN	⟶	cat
1.0	VB	⟶	scratched
0.7	DET	⟶	the
0.8	NN	⟶	man

b.) A Probabilistic Context-Free Grammar

Each rule r has the form $X \rightarrow Y$, where X is the parent nonterminal, and Y is a sequence of terminal or nonterminal children. For a PCFG, $\phi(r) = P(X \rightarrow Y | X \rightarrow \_\,)$, i.e., the probability of rule r is the conditional probability of the parent X expanding to Y, given that X has already been chosen as the parent. Thus in a PCFG, for all rules which share the same parent, the sum of their corresponding rule weights must equal 1:

$$1 = \sum_{Y \in \{V \cup \Sigma\}^+} \phi(X \rightarrow Y) \tag{3.2}$$

With the addition of rule weights we can calculate the probability of any parse, from a potentially infinite set of trees. We denote the derivation of tree t, the sequence of rule applications that transforms the start symbol Σ into the tree t, with leaf nodes corresponding to the words in s, as $\delta(t)$. Using the chain rule, we define the probability of tree t and sentence s as the product of all rule weights used in its derivation:

$$P(t, s) = \prod_{r \in \delta(t)} \phi(r) \tag{3.3}$$

Figure 3.1 presents a PCFG and phrase structure tree t for the sentence "*the cat scratched the man*". Using this PCFG we can calculate the probability of the tree t as follows:

$$\begin{aligned}
P(t, s) =& \phi(\text{S} \rightarrow \text{NP VP}) \times \phi(\text{NP} \rightarrow \text{DET NN}) \\
&\times \phi(\text{VP} \rightarrow \text{VB NP}) \times \phi(\text{NP} \rightarrow \text{DET NN}) \\
&\times \phi(\text{NN} \rightarrow cat) \times \phi(\text{VB} \rightarrow scratched) \\
&\times \phi(\text{DET} \rightarrow The) \times \phi(\text{DET} \rightarrow the) \\
&\times \phi(\text{NN} \rightarrow man) \\
P(t, s) =& 1.0 \times 1.0 \times 1.0 \times 0.6 \times 0.3 \times 0.2 \times 1.0 \times 0.7 \times 0.8 \\
P(t, s) =& 0.02016
\end{aligned}$$

In summation, PCFGs define the joint probability of any parse tree and sentence as the product of the rule weights (conditional probability of children given their head) used in its derivation.

Finding the tree that maximizes this distribution is often accomplished through chart-based decoding, and we refer the interested reader to the excellent presentation of the CKY algorithm found in [78] for further details.

Extensions and alternatives to PCFGs

PCFGs were common in many of the early successful statistical parsing systems.[2] A *lexicalized* PCFG is a simple extension to standard PCFGs which embellishes the grammar with lexical (head) information. A **headword** is the word which directly determines the type of the encompassing phrase. For instance, "*cat*" is the head of the noun phrase "*the cat*", and "*scratched*" is the head of the verb phrase "*scratched the man*" (See also X' theory [81, 82]). The work of [83] modeled bigram dependencies between headwords, and outperformed previous state-of-the-art decision-tree approaches [80]. They have since become a staple in the field. [84] further extended this model to include a probabilistic treatment of subcategorization frames for headwords (i.e., the number and type of children a parent constituent is likely to have), improving upon previous results by 2.3% on average.

PCFGs are still a component in many state-of-the-art parsing systems. PCFGs can be used to provide an initial set of parse trees and their posteriors under the model, which can be further refined using discriminative re-ranking [58, 85]. Lexicalized re-ranking parsers currently provide the best parse accuracy on the WallStreet Journal corpus. Another state-of-the-art parsing approach which uses PCFGs are latent variable grammars, in which a fine-grained grammar is induced automatically from the data [86, 87]. These methods use the treebank annotation as an initialization, but iteratively refine the grammar nonterminals to maximize the likelihood of the data, capturing useful patterns.

Approaches to parsing which do not rely on PCFGs include log-linear grammars, which derive rule weights from the sum of exponentiated feature weights [88, 89]. Unlike a PCFG, whose use of conditional probabilities as rule weights guarantees that the product of those weights yields a probability, a log-linear grammar must explicitly normalize tree scores to ensure the sum of all scores for all possible trees is 1.0. It does this through the use of the normalizing constant, Z:

$$P(t|s) = \frac{1}{Z} \prod_{r \in \delta(t)} \phi(r) \tag{3.4}$$

This is known as a globally normalized model. The method we propose is similar to previous work in log-linear parsing, but with an important distinction: our approach is *grammarless*. We train a log-linear model to predict the presence and type of constituent directly, *without rules*, and in this respect the parser functions as a set of log-linear classifiers, one for each span. However, a globally-connected combinatorial factor constrains the constituent variables, prohibiting configurations which do not represent valid trees. It is this interaction between local and global information, facilitated through belief propagation, which makes accurate parsing without a grammar possible. In the next section we discuss parameter estimation in grammar-based models and an important algorithm for inference in tree structures, before discussing analogs in the combinatorial factor we propose (Sec. 3.1.2, pg. 72).

[2] Notable exceptions are [79] and [80], which use decision trees to estimate $P(t|s)$.

3.1.1 The Inside-Outside Algorithm

How are rule weights estimated? For standard PCFG models trained from a tree-annotated corpus, the maximum likelihood estimate can be computed simply by counting:

$$\phi_{MLE}(X \to Y) = \frac{\text{count}(X \to Y)}{\text{count}(X)} \tag{3.5}$$

where count($X \to Y$) is the number of times the rule with parent X and children Y occurs in the training data, and count(X) is the number of times a rule with parent X occurs in the training data.

However, an EM-based (expectational maximization) approach provides a more general solution for estimating these parameters when there are rules which are not observed in the data. This scenario is common in grammar induction, discriminative parsing, semi-supervised training for parsers, and the factor graph representation presented in this chapter. The EM algorithm alternates between two procedures: an E-step, which computes the expectation of the log-likelihood calculated under the model using the current parameters $\Theta^{(t)}$, and an M-step, which updates the model parameters to maximize the E-step's log-likelihood.

In this section we are interested only in the calculation of the E-step for grammars, as it is also the foundation of our approach to grammarless parsing. We refer the reader interested in the M-step or any of the wider applications of the EM algorithm to one of the many in-depth tutorials on the topic, such as [90].

In the context of parsing, the EM algorithm takes as input a set of sentences **s**, of size m, with the ith sentence denoted by s_i. In the E-step we must compute the expected counts, $\mathbb{E}[\text{count}(r)]$, for each rule r:

$$\mathbb{E}[\text{count}(r)] = \sum_{i=1}^{m} \sum_{t \in T_i} P(t|s_i, \Theta^{t-1}) \, \text{count}(r, t) \tag{3.6}$$

where T_i represents all possible parse trees for sentence s_i, and count(r, t) is the number of times rule r occurs in the derivation of tree t. Explicitly enumerating these trees is not tractable, but this problem can be efficiently solved using a dynamic programming algorithm known as the **Inside-Outside algorithm** [47, 91], a generalization of the Forward-Backward algorithm for exact inference in tree-structured models.[3]

The input to the algorithm is:

- A sentence $s = (w_0, ..., w_{n-1})$ of n words. The substring $w_i, ..., w_{j-1}$ is denoted by the pair of indices (i, j), where $0 <= i <= j <= n$.
- A CFG (V, Σ, R, S) in Chomsky normal form. A grammar is said to be in Chomsky normal form if all rules are of the form $(X \to Y\ Z)$ or $(X \to x)$, where $X, Y, Z \in V$ and $x \in \Sigma$. The conversion to binary-branching trees is discussed further in Section 3.1.2 (pg. 77).
- A potential function $\phi(r)$ for each $r \in R$. In a standard PCFG the rule weights will normalize locally (Eq. 3.5), but here we define a more general potential for a tree t as $\phi(t) = \prod_{r \in \delta(t)} \phi(r)$. This is not guaranteed to normalize and must be divided by Z.

[3] Belief propagation (Sec. 2.3.2, pg. 30) is a generalization of both algorithms, and performs exact inference for models with chain or tree-structured graphs, or approximate inference in arbitrary graph structures.

When the algorithm terminates we are left with three important quantities:

1. $\mu(X \rightarrow Y\ Z, i, k, j)$, the sum of all potentials $\phi(t)$ for all $t \in T$ which contain the rule $X \rightarrow YZ$ over span indices i, j, and k. Let a span (i, j) denote a constituent which dominates words $w_i, ..., w_{j-1}$. Here the rule states that child Y spans (i, j), and child X spans (j, k).

2. $\mu(X \rightarrow x, i, k)$, the analogous sum of potentials for terminal rules, which expand to words in the sentence.

3. Z, the total probability of all parse trees T:

$$Z = \sum_{t \in T} \phi(t) \tag{3.7}$$

From these quantities we can calculate the expected counts of both nonterminal and terminal rules, and use these counts for computing the gradient necessary for maximizing the log-likelihood (the M-step). These three desired quantities are expressible in terms of two intermediate ones: the inside score, $\alpha(i, k, v)$, and the outside score, $\beta(i, k, v)$, $v \in V$. Both the inside and outside scores contain the span (i, j) rooted at a nonterminal v (which we refer to as $Span(i, k, v)$. The inside score is the sum of potentials of all trees rooted at this span, generating the words within it. The outside score is the sum of the potentials for all trees rooted at S which generate words $w_1, .., w_{i-1}$ and $w_{k+1}, .., w_n$ and ultimately a context for $Span(i, k, v)$. Fig. 3.2 depicts these quantities. We now discuss the computation of these terms via the Inside-Outside algorithm.

Inside Computation

To define $\alpha(i, k, v)$ formally, let $\tau(i, k, v)$ be the set of all trees which begin with the nonterminal v and span (i, k), generating all words $w_i, .., w_k$ beneath it. Figure 3.1 (pg. 62) depicts two phrase structure trees. Both trees contain span $(2, 7$,VP$)$, but differ in the internal structure within these spans. As such, both subtrees over the span $(2, 7)$ represent two distinct elements of the set $\tau(2, 7$,VP$)$. Using $\tau(i, k, v)$ we can define $\alpha(i, k, v)$ as follows:

$$\alpha(i, k, v) = \sum_{t \in \tau(i,k,v)} \phi(t) \tag{3.8}$$

Enumerating all trees in set $\tau(i, k, v)$, for all i, k, would be intractable. Instead, note that the calculation of $\alpha(i, k, v)$ depends on $\alpha(i, j, X)$, where $i < j < k$ and X is any nonterminal such that there exists a rule of the form $v \rightarrow X$. A similar relationship exists between $\alpha(i, k, v)$ and $\alpha(j, k, X)$. For instance, the computation of $\alpha(2, 5, VP)$, using the grammar provided in Fig. 3.1, decomposes as follows:

$$\begin{aligned}
\alpha(2, 5, VP) = &\ \phi(VP \rightarrow VB\ NP) \times \alpha(2, 3, VB) \times \alpha(3, 5, NP) \\
&+ \phi(VP \rightarrow VB\ NP) \times \alpha(2, 4, VB) \times \alpha(4, 5, NP) \\
&+ \phi(VP \rightarrow VP\ PP) \times \alpha(2, 3, VP) \times \alpha(3, 5, PP) \\
&+ \phi(VP \rightarrow VP\ PP) \times \alpha(2, 4, VP) \times \alpha(4, 5, PP)
\end{aligned}$$

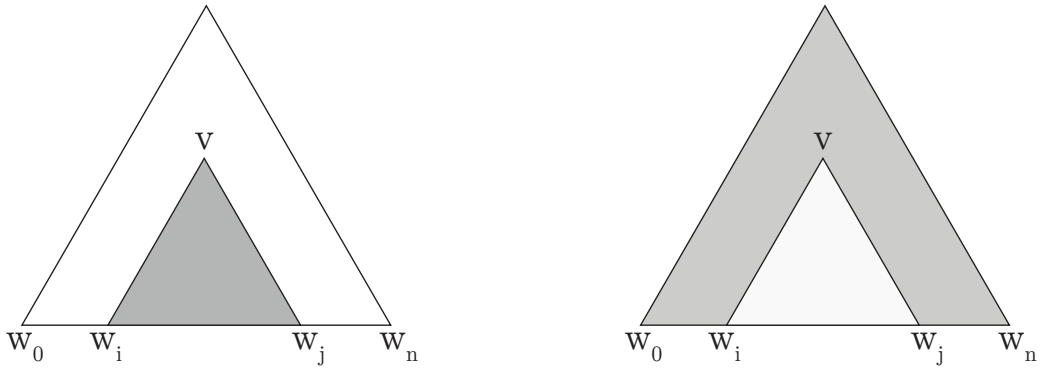


FIGURE 3.2: Inside & Outside trees. Shaded areas depict the space of trees which are summed over for each quantity.

The binary-branching structure of the input trees and grammar eliminates the need to consider decompositions over the three or more spans (i.e., a single decomposition to $(2,3),(3,4)$, and $(4,5)$ in the above example).

Thus, the computation of $\alpha(i,k,v)$ for any i,k,v can be defined recursively, thus avoiding the inefficiency of brute force enumeration. The algorithm begins by calculating $\alpha(i,k,v)$ for the smallest spans first, and proceeds to increasingly larger spans. Each $\alpha(i,k,v)$ is cached to eliminate redundant calculations. This strategy allows the Inside-Outside algorithm to compute updated expected counts for rules in $O(n^3)$ time.

Outside Computation

Similarly, to formally define the corresponding outside score we introduce the notion of an *outside tree*. An outside tree $\mathrm{O}(i,j,v)$ is a tree which a.) is rooted at the start symbol, S, b.) generates $Span(i,k,v)$, and c.) generates the words that precede (w_0 to w_i) and follow it (w_{k+1} to w_n). In other words, an outside tree generates a context for span $Span(i,k,v)$. We define the outside term $\beta(i,j,v)$ as the sum of the potentials for trees $t \in \mathrm{O}(i,j,v)$:

$$\beta(i,j,v) = \sum_{t \in \mathrm{O}(i,j,v)} \phi(t) \tag{3.9}$$

The outside scores can be calculated in a similar manner, however the algorithm begins by computing the outside score for the top-level span, and recursively calculating the β values for progressively smaller spans. However, note that to compute the β values one must rely on α values calculated in the previous step. To calculate $\beta(i,k,v)$ we must examine the outside score for span $(0,n)$ and the inside score for the remainder, $(0,i)$ and (k,n). We momentarily forgo the details of this calculation, and discuss instead only the outside calculation as it pertains to the grammarless factor graph parsing in Section 3.1.2 (pg. 72).

Computing the Gradient

Once the inside and outside scores have been computed, the desired quantities can be expressed in

terms of these quantities. Z, the partition function, is the total probability of all parses for a given sentence x of length n. It is required to normalize rule potentials to produce expected counts. Note that this is equal to the inside probability of the top-level span, beginning with the start symbol S:

$$Z = \alpha(0, n, S) \tag{3.10}$$

To compute $\mu(X \to Y\ Z, i, k, v)$ we combine the outside score of the span (i, k) and the inside score for the two analyses within the span, (i, j) and (j, k):

$$\begin{aligned}\mu(X \to Y\ Z, i, k, j) = \phi(X \to Y\ Z, i, k, j) \\ \times \beta(i, j, X) \\ \times \alpha(i, k, Y) \times \alpha(k, j, Z)\end{aligned} \tag{3.11}$$

Summing over the span indices and normalizing yields the expected counts for the rule, and the necessary quantity for computing the parameter update:

$$\mathbb{E}[\text{count}(X \to Y\ Z)] = \sum_{i,j,k} \frac{\mu(X \to Y\ Z, i, k, j)}{Z} \tag{3.12}$$

Rules of the form $X \to x$ are handled analogously. These expected counts can then be utilized in the M-step to compute the gradient and update the model parameters.

The Inside-Outside algorithm is used in a wide variety of applications. [92] use the Inside-Outside algorithm to infer the parameters of a PCFG from a partially-annotated treebank, containing a mix of full parsed, partially-parsed, and unparsed sentences. [93] utilize the algorithm in an attempt to adapt hand-written parse rules to new corpora. Conditional random field parsing [89], and latent variable grammars [86, 94] also rely on variants of this algorithm for parameter estimation.

This concludes our discussion of the Inside-Outside algorithm and the theory underlying efficient computation of rule expectations. In the following section we present a factor graph representation of phrase structure trees and show how the principles described in this section can be used to construct a new variant of the Inside-Outside algorithm which does not compute rule expectations, but rather the expectation that a given (unlabeled) constituent span is present in the tree. From within a globally-connected combinatorial factor, this algorithm can compute outgoing messages during inference which guide variable beliefs towards representing a valid phrase structure tree.

3.1.2 Representing Phrase Structure in Factor Graphs

In the previous section we discussed PCFG-based parsing and parameter estimation for discriminative or latent variable parsing models (including unsupervised models and training from partially-annotated corpora) using EM and the Inside-Outside algorithm. In this section we now contrast these methods with our own approach to parsing. Here we demonstrate how to represent phrase structure syntax in a factor graph, and how to perform efficient inference using a combinatorial factor. This produces an unlabeled tree structure which may be well-suited for use as a syntactic component in joint models. We then detail extensions necessary for competitive performance in parsing tasks, which includes labeled parsing and unary chain prediction, before discussing more practical considerations, like pruning heuristics. We begin with the factor graph representation.

For a sentence with n words there are $O(n^2)$ possible constituent spans. Recall that if a constituent spans (i, j) its leaf nodes are words w_i to w_{j-1}, and thus there is one possible constituent span for each $0 \leq i < j \leq n$. In order to represent a model of phrase structure syntax as a factor graph, we use $O(n^2)$ Boolean variables, each representing the presence of a constituent span when true, and the absence of a constituent span when false. This provides the a *representation* for any possible unlabeled phrase structure tree, but the suitability of each variable is scored locally by potentials, calculated via features, in corresponding unary factors. Formally, we describe this structure as follows:

- Let $\{Span(i, j) : 0 \leq i < j \leq n\}$ be $O(n^2)$ Boolean variables such that $Span(i, j) = \text{true}$ iff there is a span beginning at index i and ending at index j in the constituent structure.[4]
- Let $\{\text{SPAN}(i, j) : 0 \leq i < j \leq n\}$ be $O(n^2)$ unary factors, each attached to the corresponding $Span(i, j)$ variable. These factors score, via a set of exponentiated feature weights, the suitability of each span to appear in an unlabeled phrase structure tree.

This collection of variables is capable of representing all possible unlabeled phrase structure trees, provided that these trees do not contain multiple spans over the same pair of indices (i.e., a unary chain). However, it is also capable of representing many invalid structures. For instance, two variables representing overlapping spans $(Span(i, j), Span(k, l) : i < k < j < l)$ cannot be in a valid phrase structure tree. We now discuss the formal properties of a restricted class of trees, and a combinatorial factor which constrains configurations of $Span$ variables to form valid members of this set.

For our purposes of defining a phrase structure representation for joint inference, the usefulness of syntax lies purely in its structure (i.e., whether or not a span is in the tree) and not in its ability to reproduce treebank-style annotation. Therefore we do not aim to predict unary chains (though a method for reintroducing unary chains is discussed in Section 3.1.2, pg. 81). Similarly, an unlabeled constituent spanning a single word is not informative to other tasks, as we assume their presence (part-of-speech tags and words are observed). Therefore only spans of width two or greater are necessary. Our inference methods share the same underlying principles as previous parsing work, and similarly require binary-branching structure. Thus the class of phrase structure trees we are interested in modeling are unlabeled, binary-branching trees with constituents of width two or greater, and no more than one constituent per pair of indices.

Formally a tree drawn from this restricted set of phrase structure trees is any set of spans $Span(i, j)$ where $0 <= i <= j <= n$, and [5]:

1. $Span(0, n) = 1$
   All trees must contain a root constituent spanning the entire sequence of terminal nodes.
2. $\forall(i < j < k < l) : \neg(Span(i, k) \wedge Span(j, l))$
   Spans are not permitted to cross.
3. $\forall(i < j < k; k - i > 3) : Span(i, k) \Longrightarrow \exists j : Span(i, j) \wedge Span(j, k)$
   All spans of width 4 or greater must have exactly two children.

[4] In practice, we do not need to include variables for spans of width 1 or n, since they will always be true.

[5] These are variants of the more extensive set of axioms described in [95]

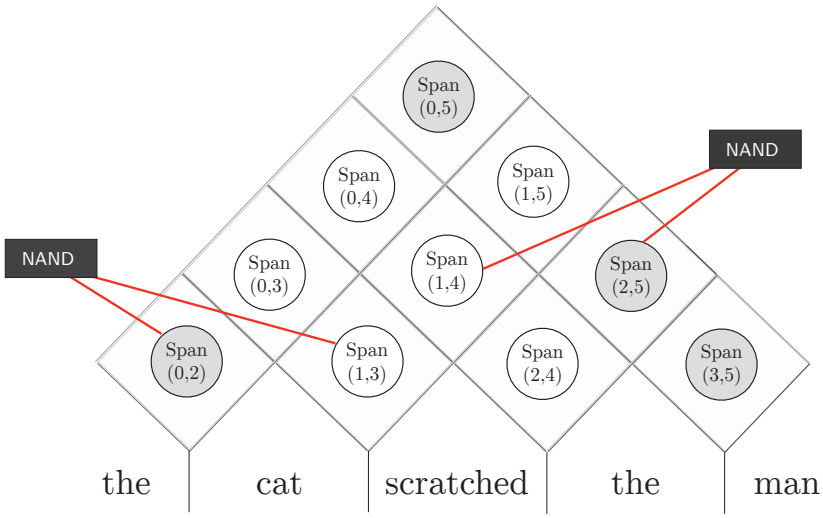


FIGURE 3.3: Variables for phrase structure syntax. The variables here represent the constituent tree for the sentence "*The cat scratched the man*", adapted from the example in Fig. 3.1. Shaded variables indicating a true value and the presence of a constituent span in the tree. The grid may aid readers who are familiar with chart parsing to grasp the correspondence between the factor graph representation and the traditional parsing literature. Hard NAND factors can be used to prohibit two overlapping span variables from being true simultaneously, but a quartic number of such factors would be required.

4. $\forall(i < j < k; k - i = 3) : Span(i, k) \implies k - i = 3 \wedge (\exists j : Span(i, j) \vee Span(j, k)$
   All spans of width 3 will have exactly one child, aligning either to the left or right side of the encompassing span. Spans of width 2 act as terminal nodes, in that they have no children.

The top level span variable, $Span(0, n)$, can be explicitly set to true, satisfying axiom $\#1$. The Inside-Outside algorithm operates on binary-branching trees, and this binarization, which we do to all trees prior to training and is assumed throughout the entire algorithm, guarantees that axiom $\#3$ and $\#4$ is satisfied (single word spans do not need to be represented in the model, so 2-word spans become the smallest in the tree). However, there is no mechanism for preventing variable configurations in which two variables representing overlapping spans are simultaneously true. This would violate axiom $\#2$, which requires that spans properly nest. Thus the crux of expressing a constituent tree as a factor graph lies in finding an efficient way to provide this constraint, preventing violations to axiom $\#2$, and guaranteeing that the collection of *Span* variables describes a valid tree.

One approach to solving this problem is to tackle it piecewise, attempting to capture a global constraint through the interactions of many simple local constraints. This approach is illustrated in Fig. 3.3. An invalid tree is formed when two overlapping spans are present, and therefore factors may be inserted between each pair of overlapping $Span$ variables to prohibit these configurations. Such a factor would be implement by a logical NAND, contributing a score of 0 when both variables are true, penalizing the configuration, and 1 otherwise. But note that the number of such factors necessary to accomplish this grows at a rate quartic, $O(n^4)$, in the length of the sentence, as a factor is required for all pairs of $Span(i, j)$ and $Span(k, l)$ variables such that $i < k \leq j < l$. Not only does this drastically increase the number of factors in the graph, but it creates a highly cyclic

graph which may significantly hinder inference. This makes it an unfavorable option.

In the same way we propose MRFs as a common language for joint models, [96] propose case-factor diagrams (CFDs) as a common language for structured models, including PCFGs. CFDs subsume MRFs, can represent a PCFG using $O(|G|n^3)$ Boolean variables, and can calculate variable marginals (or a corresponding Viterbi solution) in time linear in the number of variables. A CFD decomposes structured models into Boolean variables, and uses factors to coordinate between these variables in order to express the compositionality of constituent spans in the tree. To represent a PCFG it requires $|V|n^2$ variables of the kind presented above to represent a nonterminal $v \in V$ spanning from i to j for all $0 \leq i < j \leq n$ (we propose an identical set of variables for performing labeled parsing in Sec. 3.1.2, pg. 79), and a branch variable "$X_{i,k} \rightarrow Y_{i,j}, Z_{j,k}$" for each production rule in the grammar. The key insight of CFDs is that by decomposing the PCFG representation out into Boolean variables, inference in the graph can function in a zero-suppressed manner, meaning that the hierarchical structure of the tree can be exploited and portions of the graph which represent invalid configurations do not need to be explicitly searched. This makes them well-suited to sparse problems of the sort common in NLP. Therefore CFDs provide another method of providing a tree constraint, but require specialized inference.

Yet another approach for providing this constraint is to use a single globally-connected factor. [11] previously used this approach to efficiently represent dependency syntax in factor graphs, where variables must be constrained to prohibit cycles. In comparison to modeling the constraint through many pairwise factors, this has the benefit of not producing such a densely connected graph. In fact, inference is exact. Unlike CFDs, the constraint logic is encoded in a single factor, and not distributed throughout the graph. With the exception of this one factor, standard inference algorithms can be used. In the next section we present our application of this design to constituent parsing, the CKY-TREE factor.

Description of the CKY-TREE Factor

Given a set of Boolean variables, each representing the presence of a constituent span, a single global factor can be used to constrain these variables to form valid phrase structure trees. We introduce this factor formally as follows:

- Let CKY-TREE be a global combinatorial factor attached to all the $Span(i, j)$ variables. This factor contributes a potential of 1 to the model's score if the span variables collectively form a legal, binary-branching tree (as specified in the axioms in Section 3.1.2, pg. 69)and a potential of 0 otherwise. It enforces, therefore, a hard constraint on the variables. All outgoing messages from this factor are computed simultaneously by a variant of the Inside-Outside algorithm, described in Fig. 3.4.

In order to provide the necessary constraint, the desired factor needs to collect the beliefs of all $Span$ variables, and efficiently produce for each variable $Span(i, j)$ an outgoing message $m_{\text{CKY-TREE}\rightarrow Span(i,j)}$ which reflects (1) the beliefs of all other $Span$ variables, and (2) their role within the hierarchical structure. This is an expectation of the $Span$ variable's value taking into account the beliefs of all other $Span$ variables. Note that this goal is similar to that of general marginal inference (Section 2.3, pg. 28). Here we rely on a variant of the Inside-Outside algorithm (originally discussed in Section 3.1.1, pg. 66).

Algorithm 3 Bracket inside algorithm

function INSIDE(u, n)
  for $w \leftarrow 2, ..., n$ **do**
    for $i \leftarrow 0, ..., (n - w)$ **do**
      $k \leftarrow i + w$
      $s \leftarrow \mathbf{0}$
      for $j \leftarrow (i + 1)..(k - 1)$ **do**
        $s \overset{\oplus}{\leftarrow} \alpha(i, j) \otimes \alpha(j, k)$
      end for
      $\alpha(i, k) \leftarrow s + u(i, k)$
    end for
  end for
  return α
end function

Algorithm 4 Bracket outside algorithm

function OUTSIDE(u, α, n)
  $\beta(0, n) \leftarrow \alpha(0, n)$
  for $w \leftarrow n, ..., 2$ **do**
    for $i \leftarrow 0, ..., (n - w)$ **do**
      $k \leftarrow i + w$
      for $j \leftarrow (i + 1)..(k - 1)$ **do**
        $\beta(i, j) \overset{\oplus}{\leftarrow} \beta(i, k) \otimes \alpha(j, k) \otimes u(i, k)$
        $\beta(j, k) \overset{\oplus}{\leftarrow} \beta(i, k) \otimes \alpha(i, j) \otimes u(i, k)$
        $\mu(i, k) \overset{\oplus}{\leftarrow} \beta(i, k) \otimes \alpha(i, j) \otimes \alpha(j, k)$
      end for
    end for
  end for
  return μ
end function

FIGURE 3.4: Pseudocode for CKY-TREE. The inside algorithm is responsible for computing inside scores $\alpha(i, k)$ from input odds ratios $u(i, k)$. The outside algorithm similarly computes outside scores $\beta(i, k)$, and uses both inside and outside scores to compute the gradient $\mu(i, k)$. The $\overset{\oplus}{\leftarrow}$ denotes incremental assignment.

First, because we aim to encapsulate this algorithm inside a factor, the algorithm needs to function over the type of information captured by the variables in the model. Thus the input to the algorithm is a message from each $Span$ variable, i.e., the expectation of its value given the beliefs of its other neighbors. In this initial parsing model $Span$ variables have no neighboring variables, and their beliefs are informed solely by unary SPAN factors. As in earlier work in dependency parsing [11] and using a strategy detailed in Section 2.4 (pg. 36), the algorithm begins by gathering all incoming messages, $m_{Span(i,k) \rightarrow \text{CKY-TREE}}$, and converting each to an odds ratios, $u(i, k)$, to allow for efficient computation of the partition function:

$$u(i, k) = \frac{m_{Span(i,k) \rightarrow \text{CKY-TREE}}(\mathsf{true})}{m_{Span(i,k) \rightarrow \text{CKY-TREE}}(\mathsf{false})} \tag{3.13}$$

As described in Section 3.1.1, the Inside-Outside algorithm decomposes the complete computation into two parts. The inside pass computes $\alpha(i, k)$, for all $i, k : 0 \leq i < k \leq n$, the score for the span at (i, k) generating the words $w_i, ..., w_k$. The outside pass computes $\beta(i, k)$, for all $i, k : 0 \leq i < k \leq n$, the score for generating the words $w_0, ..., w_{i-1}$ and $w_{k+1}, ..., w_n$ immediately outside of the span at (i, k), from the top-level span at $(0, n)$. To reiterate, in other words, for a given span (i, k) the inside score refers to the total score of all analyses capable of generating the words within the span, while the outside score refers to the total score of all analyses which generate the words outside of the span. The desired outcome, gradient $\mu(i, k)$, is the normalized product of $\beta(i, k)$ and $\alpha(i, k)$.

After running the Inside-Outside algorithm to compute the gradient $\mu(i,k)$ in $O(n^3)$ time, we calculate the $O(n^2)$ outgoing messages from CKY-TREE as:

$$m_{\text{CKY-TREE}\to Span(i,j)}(\mathsf{true}) = \mu(i,j)$$
$$m_{\text{CKY-TREE}\to Span(i,j)}(\mathsf{false}) = 1 - u(i,j)\cdot\mu(i,j)$$

The resulting variable beliefs reflect the beliefs of all other $Span(i,j)$ variables and their roles within the hierarchical tree structure.

Proof

How do we know that the algorithm presented in Fig. 3.4 computes the constraint described in Section 3.1.2? First, we show that the outgoing messages can be derived from the outside scores.

As shown in the ISATMOST1 factor discussion of Section 2.4.1, efficient computation of outgoing messages in a combinatorial factor relies on expressing the partition function, Z, in terms of incoming variable messages. This avoids explicitly summing over entries in a potential table. In the ISATMOST1 factor message computation it was straightforward to define Z in terms of odds ratios from incoming messages. Here we can apply the same technique, but over a hierarchical structure.

We once again turn to the notion of an inside score, which we define recursively as:

$$\alpha(i,k) = \begin{cases} \text{if } k-i=2: & u(i,k) \\ \text{if } k-i>2: & u(i,k)\sum_{j:i<j<k}\alpha(i,j)\times\alpha(j,k) \end{cases} \tag{3.14}$$

We can then define Z in terms of inside scores:

$$Z = \alpha(0,n) \tag{3.15}$$

where $\alpha(0,n)$ is the sum score of all possible trees for a sentence of length n.

As described in Eq. 2.3.2 (pg. 31), in order to calculate outgoing messages to a variable v we must marginalize over the contributions of neighboring variables, excluding those from v. In the CKY-TREE factor, we exploit the hierarchical nature of the dependencies to more efficiently perform this same calculation. As shown in Eq. 3.14, the inside score computation for a span (i,k) marginalizes over the contributions of potential subspans. The marginalization over messages from spans larger than (i,k) occurs in the outside computation:

$$\beta(i,k) = \sum_{s_1:0\le s_1<i}\beta(s_1,k)\times\alpha(s_2,i)\times u(s_1,k) + \sum_{s_2:k<s_2\le n}\beta(i,s_2)\times\alpha(k,s_2)\times u(i,s_2)$$

where s_1 and s_2 range across all possible indices for constructing a left and right context respectively. As in the example exact marginal computation in Eq. 2.10 (pg. 29), multiplying all incoming marginalized messages and normalizing produces the variable's marginal. Here multiplying these two quantities yields the *expectation* of the variable's marginal, the desired quantity, gradient μ:

$$\mu(i,k) = \alpha(i,k)\times\beta(i,k)$$

Thus the desired quantities can be expressed in terms of α and β scores, computed by incoming variable messages.

Second, we show that the gradient, $\mu(i,k)$, for any span can be computed via a dynamic programming transformation, provided the algorithm exhibits the dynamic program property of optimal substructure, i.e., that the solution the entire problem can be computed efficiently from the solutions to its subproblems. It is straightforward to show that the Inside variant algorithm presented in Fig. 3.4 exhibits this property, using the definitions provided in the previous section.

Any inside score $\alpha(i,k)$ reflects the sum score for all possible trees within (i,k). By the definition in Eq. 3.14 we see that this can be computed efficiently by summing the scores for all possible child trees, using a single split point $j : i < j < k$ to divide (i,k) into (i,j) and (j,k). Each analysis of subtrees contributes $u(i,k) \times \alpha(i,k) \times \alpha(j,k)$ to the sum score, and summing over all such analysis can be shown to be an exhaustive search, as we assume trees to be binary-branching.

Therefore we have shown that this problem adheres to the optimal substructure property of dynamic programs. We can therefore apply a mechanical transformation which transforms the Inside algorithm (Algorithm 3 of Fig. 3.4) into the Outside algorithm (Algorithm 4 of Fig. 3.4), which produces the desired messages. We refer the reader to [see 97, pg. 150-161] for further discussion of this method.

CKY-TREE: An Example

Let us examine the CKY-TREE computation on an example sentence, "*The cat scratched the man*", with the correct binarized tree shown in Fig. 3.6, (*left*). We focus on the behavior of three spans: $Span(0,2)$, $Span(2,5)$, and $Span(1,5)$, covering *the cat*, *scratched the man*, and *cat scratch the man*. The following table contains the quantities computed during the algorithm, and the rows corresponding to the spans of interest are highlighted:

Span	$\pi(Span(i,k))$ $before$	$u(i,k)$	$\alpha(i,k)$	$\beta(i,k)$	$\mu(i,k)$	$m_{(Span(i,k)}$ (true)	$\pi(Span(i,k))$ $after$
0,2	0.80	4.00	4.00	0.18	0.18	0.37	0.70
1,3	0.20	0.25	0.25	0.25	0.25	0.21	0.06
2,4	0.20	0.25	0.25	0.23	0.23	0.19	0.06
3,5	0.80	4.00	4.00	0.23	0.23	0.80	0.94
0,3	0.10	0.11	0.47	0.15	0.65	0.41	0.07
1,4	0.10	0.11	0.06	0.06	0.03	0.03	0.00
2,5	0.50	1.00	4.25	0.20	0.87	0.87	0.87
0,4	0.10	0.11	0.17	0.04	0.06	0.05	0.01
1,5	0.60	1.50	7.96	0.04	0.20	0.22	0.29

Each of the three $Span$ variables begins with either a neutral belief or a strong belief of its own value being true, shown in the second column, but not all of these spans can be present in the predicted tree: $Span(1,5)$ conflicts with $Span(0,2)$. How does the CKY-TREE constraint alter the variable beliefs in this case?

The algorithm begins by converting variable beliefs to odds ratios, as shown in Eq. 3.13 [6] Note that the top-level span is omitted from this calculation, as it must always be true. The inside scores, $\alpha(i,j)$, are computed as the product of the odds ratio, $u(i,j)$, and the inside scores of any properly

[6] In practice it is prudent to convert to log-space prior to computing the odds ratio to avoid underflow errors.

nesting spans within it. $Span(0, 2)$ is the smallest representable in the model, and therefore cannot be decomposed further. Its inside score is simply its odds ratio:

$$\begin{aligned}\alpha(0, 2) &= u(0, 2) \\ &= 4.0\end{aligned}$$

For span $(2, 5)$ there are two possible split points ($j = 3$, and 4)[7]:

$$\begin{aligned}\alpha(2, 5) &= u(2, 5) \times ((\alpha(2, 3) \times \alpha(3, 5)) + (\alpha(2, 4) \times \alpha(4, 5))) \\ &= 1.0 \times ((1.0 \times 4.0) + (0.25 \times 1.0)) \\ &= 4.25\end{aligned}$$

For span $(1, 5)$ there are three possible split points ($j = 2, 3$, and 4):

$$\begin{aligned}\alpha(1, 5) &= u(1, 5) \times ((\alpha(1, 4) \times \alpha(4, 5)) + (\alpha(1, 3) \times \alpha(3, 5)) + (\alpha(1, 2) \times \alpha(2, 5))) \\ &= 1.5 \times ((0.06 \times 1.0) + (0.25 \times 4.00) + (1.0 \times 4.25)) \\ &= 7.96\end{aligned}$$

Each group of inner parentheses indicate the contributions from a different analysis of the tree structure under (i, k), i.e. calculations using a different split point j. In cases where the split point j produces a single width span (i.e., $j - i = 1$ or $k - j = 1$), it does not contribute to inside term $\alpha(i, k)$. In contrast to the Inside-Outside algorithm as presented for grammar-based parsing in Sec. 3.1.1 (pg. 66), note that the odds ratio substitutes for the rule weight: we no longer have any rules or beliefs pertaining to rules, but we do have a belief regarding each span's presence in the tree.

We calculate outside scores $\beta(i, j)$ similarly, working from larger to smaller spans:

$$\begin{aligned}\beta(0, 2) &= (u(0, 5) \times \beta(0, 5) \times \alpha(2, 5)) \\ &\quad + (u(0, 4) \times \beta(0, 4) \times \alpha(2, 4)) \\ &\quad + (u(0, 3) \times \beta(0, 3) \times \alpha(2, 3)) \\ &= 0.18 \\ \beta(2, 5) &= (u(0, 5) \times \beta(0, 5) \times \alpha(0, 2)) \\ &\quad + (u(1, 5) \times \beta(1, 5) \times \alpha(1, 2)) \\ &= 0.2 \\ \beta(1, 5) &= (u(0, 5) \times \beta(0, 5) \times \alpha(0, 1)) \\ &= 0.04\end{aligned}$$

[7] Intermediary terms are rounded to the nearest hundredth.

The computation of the gradient:

$$\begin{aligned}
\mu(0,2) &= (\alpha(0,1) \times \alpha(1,2) \times \beta(0,2) \\
&= 0.18 \\
\mu(2,5) &= (\alpha(2,3) \times \alpha(3,5) \times \beta(2,5)) \\
&\quad + (\alpha(2,4) \times \alpha(4,5) \times \beta(2,5)) \\
&= 0.87 \\
\mu(1,5) &= (\alpha(1,3) \times \alpha(3,5) \times \beta(1,5)) \\
&\quad + (\alpha(1,4) \times \alpha(4,5) \times \beta(1,5)) \\
&= 0.2
\end{aligned}$$

And the outgoing messages:

$$\begin{aligned}
m_{\text{CKY-TREE}\to Span(0,2)}(\textsf{true}) &= \mu(0,2) = 0.18 \\
m_{\text{CKY-TREE}\to Span(0,2)}(\textsf{false}) &= 1 - u(0,2) \times \mu(2,5) = 0.3 \\
m_{\text{CKY-TREE}\to Span(2,5)}(\textsf{true}) &= \mu(2,5) = 0.87 \\
m_{\text{CKY-TREE}\to Span(2,5)}(\textsf{false}) &= 1 - u(2,5) \times \mu(2,5) = 0.13 \\
m_{\text{CKY-TREE}\to Span(1,5)}(\textsf{true}) &= \mu(1,5) = 0.20 \\
m_{\text{CKY-TREE}\to Span(1,5)}(\textsf{false}) &= 1 - u(1,5) \times \mu(1,5) = 0.71
\end{aligned}$$

Normalizing produces the final outgoing messages. Examining the posteriors of the three variables reveals the influence of the CKY-TREE constraint: $Span(1,5)$ is heavily penalized, altering it's beliefs across the threshold from true to false. This is not surprising given that the span conflicts not only with $Span(0,2)$, but other variables with strong true beliefs as well. But this conflict also influenced the beliefs of $Span(0,2)$, and its true belief has been reduced to 0.7. The remaining span, $Span(2,5)$, did not conflict with any span with a strong true belief, and its own true belief increased significantly to 0.87. Thus the CKY-TREE factor took as input a set of variable beliefs which did not represent a valid tree structure, and adjusted these beliefs using global information. Here the resulting beliefs, when decoded, would produce the desired gold tree.

Binarization

When computing inside and outside scores in the CKY-TREE factor, there is an assumption made regarding the structure of the tree. By searching for just a single split point j within the bounds of span (i,k), we assume that the tree has a binary-branching structure. This is a general assumption prevalent in CKY parsing, necessary for achieving an $O(n^3)$ complexity, and thus we must transform all trees in the training data to be binary-branching.

However, there is not a consensus on precisely the best method for converting n-ary branching trees to their binary-branching counterparts. The choice of binarization strategy has shown to be important for efficiency considerations in parsing [98], and with synchronous grammars [99], but [100] find only small differences in performance across different binarization schemes. We adopt a right-branching 0-Markov binarization strategy (pseudocode for this method is provided in Fig.

Algorithm 5 Binarization algorithm

```
1: function BINARIZE(t)
2:     if |children(t)| > 2 then
3:         l ← if (labelOf(l)[0, 1] = "@″) then labelOf(l) else "@″+labelOf(l)
4:         return (labelOf(t),
               (binarize(head(children(t))),                          ▷ Left Child
               (l, tail(children(t)).map{c → binarize(c)}))           ▷ Right Child
5:     else
6:         return (labelOf(t), children(t).map{c → binarize(c)})
7:     end if
8: end function
```

FIGURE 3.5: Pseudocode of a right-branching binarization. The algorithm assumes that a tree is a tuple consisting of a label and a list of child trees.

3.5), and convert all trees prior to training. English has a predominantly right-branching structure, potentially making it a more appropriate choice than a simple left-branching scheme, as it improves the rate of agreement between syntactic spans and spans for other NLP tasks, like named entity recognition (4.1.1, pg. 105)

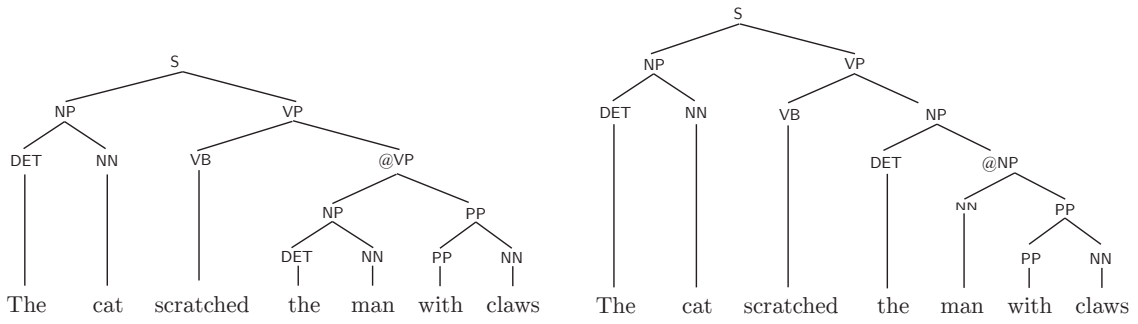
Figure 3.6 illustrates the trees produced by right-branching binarization, as applied to the n-ary branching trees shown in Fig. 3.1. To reiterate how these trees would be represented within the model, the true-valued variables which would produce each tree are as follows:

High PP-attachment	Low PP-attachment
$Span(0, 2)$	$Span(0, 2)$
$Span(3, 5)$	$Span(5, 7)$
$Span(5, 7)$	$Span(4, 7)$
$Span(3, 7)$	$Span(3, 7)$
$Span(2, 7)$	$Span(2, 7)$
$Span(0, 7)$	$Span(0, 7)$

All binary-branching trees over an identical number of leaf nodes will have an identical number of spans.

Decoding

The CKY-TREE factor prohibits configurations of $Span$ variables which do not represent members of the restricted class of trees presented in Section 3.1.2. However, it does not accomplish this goal immediately; it only adjusts $Span$ variable beliefs *toward* a valid configuration. To ensure that the model predicts valid trees it is necessary to decode using a special procedure, not unlike the decoding methods of traditional chart-parsing systems. While the CKY-TREE factor can be thought of as a variant of the sum-product algorithm for marginal inference which adheres to the axioms Section 3.1.2 (pg. 69), our decoding strategy can be thought of as the analogous max-product algorithm.



a.) High PP-attachment b.) LowPP-attachment

FIGURE 3.6: Binarized trees. The above trees are produced using right-branching binarization on the sentences shown in Fig. 3.1. Nonterminal labels with the @ prefix have been introduced in the binarization process.

Figure 3.7 provides pseudocode for the decoding procedure. The first step is to find the set of split points which yields the highest weighted subtrees, using the procedure described in the BRACK-DECODE function. Here we rely once again on the inside scores, $\alpha(i, k)$. For a span over (i, k) we examine all possible inner structures, split points j where $i < j < k$. We define the score of each decomposition, $s(i, j, k)$, as the product of $\alpha(i, j)$ and $\alpha(j, k)$. Note that for any two split points j and j', if $s(i, j, k) > s(i, j', k)$, and if the most probable tree contains a span over (i, k), then it must contain $s(i, j, k)$. Thus by processing spans from smallest to largest and finding for each span the split point that yields the highest score, we ensure that the maximum split point over $(0, n)$ is the max tree.

Once the maximum split points have been calculated, it is only a matter of backtracking through these points to determine which spans should be present in the maximum tree. The BACK-TRACE function performs a recursive search beginning at the top-level span and exploring progressively smaller spans, taking the maximum split point at each step. Thus all spans visited in this search are present in the final tree. The observed words and part-of-speech tags are then inserted into the tree, functioning as leaf nodes.

Labeled Parsing in Factor Graphs

The previous section discussed a factor graph approach for unlabeled phrase structure parsing. This structure can be useful in the context of a joint model, but the parsing evaluation measures labeled accuracy: a constituent is not considered correct unless the gold tree contains a span over the same indices, *and* both constituents share the same label. We now propose a model of labeled phrase structure parsing, which we evaluate against previous state-of-the-art parsers in Section 3.1.4 (pg. 89).

Extending the unlabeled parsing model to predict constituent types requires: a.) a set of variables to represent the constituent label set L for each span, and b.) a set of factors to coordinate between corresponding span and label variables. We formally introduce these components below:

- Let $\{Label(i, j, \lambda) : \lambda \in L, \text{ and } 0 \leq i < j \leq n\}$ be $O(|L|n^2)$ Boolean variables such that $Label(i, j, \lambda) = \mathsf{true}$ iff there is a span from i to j with constituent label λ.

Algorithm 6 Bracket Parser Decoding

```
1:  function BRACK-DECODE(n)
2:      for w ← 2..n do
3:          for i ← 0..(n − w) do
4:              k ← i + w
5:              for j ← (i + 1)..(k − 1) do
6:                  s ← α(i, j) ⊗ α(j, k)
7:                  if s > inside then
8:                      best-inside ← s
9:                      best-split ← j
10:                 end if
11:             end for
12:             α(i, k) ← best-inside +P(Span(i, k) = true)
13:             splits(i, k) ← best-split
14:         end for
15:     end for
16:     best-inside ← 0
17:     for j ← 1..n do
18:         s ← α(0, j) ⊗ α(j, n)
19:         if s > inside then
20:             best-inside ← s
21:             best-split ← j
22:         end if
23:     end for
24:     splits(0,n) ← best-split
25:     return BACK-TRACE(0, n, splits, bracks)
26: end function
```

Algorithm 7 Finding the Best Path

```
1:  function BACK-TRACE(i, k, splits, bracks)
2:      j ← splits(i, k)
3:      if j > i + 1 then
4:          bracks(i, j) = true
5:          BACK-TRACE(i, j, splits, bracks)
6:      end if
7:      if j < k − 1 then
8:          bracks(j, k) = true
9:          BACK-TRACE(j, k, splits, bracks)
10:     end if
11: end function
```

FIGURE 3.7: Pseudocode for unlabeled parser decoding

- Let $\{\text{ISATMOST1}(i, j) : 0 \leq i < j \leq n\}$ be $O(n^2)$ factors, each coordinating between a single $Span(i, j)$ variable and the set of corresponding $Label(i, j, \lambda)$ variables, as illustrated in Fig. 3.8. This factor constrains these variables such that iff $Span(i, j)$ = true, a single $Label(i, j, \lambda)$ variable will also be true. Otherwise all variables are false.

It may seem unnecessary to represent each label as its own Boolean variable, when a single multinomial would suffice. This representation has the disadvantage that it also requires an additional combinatorial logic factor, ISATMOST1, to coordinate between corresponding $Span$ and $Label$ variables. However, representing each label as a separate Boolean variable allows simple Boolean logic factors to have individual access to each label variable. This will prove to be convenient when implementing certain extensions, such as incorporating grammatical rule factors into the model (Section 3.1.3, pg. 85), or constructing joint models which use logical factors to coordinate between an NLP task and a set of $Label$ variables.

It is important to understand that incorporating a set of label variables does not merely assign a label to the unlabeled tree structure. The addition of label variables is more significant, influencing and potentially altering the *unlabeled structure* predicted by the model via the bidirectional information flow during inference. Corresponding $Label$ and $Span$ variables are coordinated using an ISATMOST1 factor, implementing a hard constraint: when $Span$ is true, a single corresponding

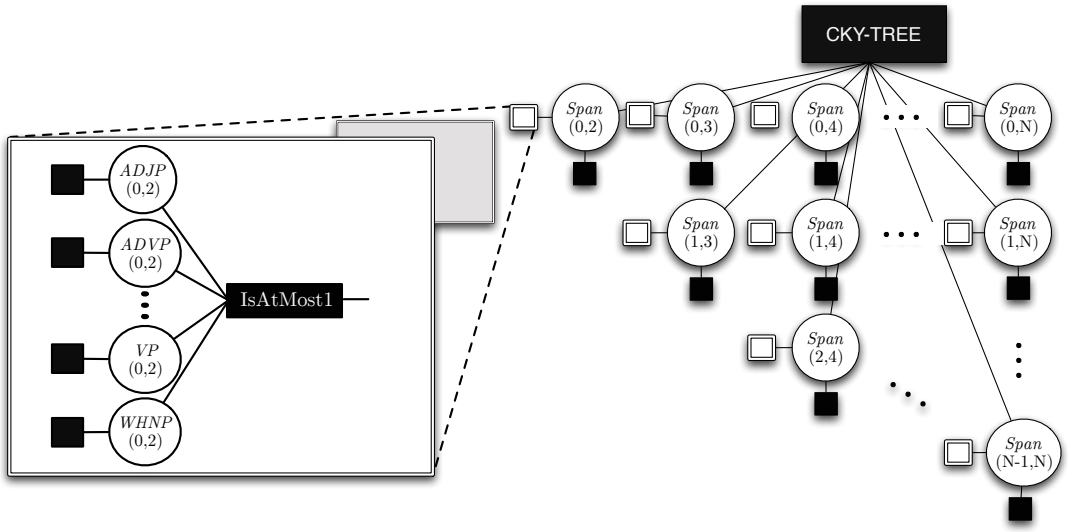


FIGURE 3.8: Graphical depiction of the labeled parsing model. Double-boxed rectangles refer to the labeling structure shown in the cut out (*left*). For the sake of clarity, label variable names are replaced with their corresponding label value.

$Label$ variable must also be true. Otherwise, all variables are false. Consequently, if the model strongly believes that a particular $Label$ variable must be true, this may sway an unlikely $Span$ variable's beliefs to become true in order to satisfy the ISATMOST1 constraint.

Unary Chains

While our primary motivation for introducing factor graph parsing models is for their use as a latent component in a larger joint model, in this chapter we also aim to show that this model, when treated as a standalone parser, is competitive with state-of-the-art parsers. However, the standard evaluation for phrase structure trees scores spans that are present in treebank annotation but not represented in our model. These spans, known as unary chains, occur when there are multiple spans for a given pair of indices (i, j). A example of a common unary chain in English is NP $\rightarrow$ NN, due to the noun playing two roles in the tree: a lexical role as a leaf node, and an NP at a higher point in the tree. Evaluating our parser without modeling at least some of these spans would significantly hinder our efforts to be competitive with state-of-the-art models on standard evaluations.

In grammar-based approaches the prediction of unary chains falls out naturally from the recursive definition of the model, but in our span-factored formulation they must be handled by a separate prediction. Similarly, [101] predict leaf-level unary chains separately from the main model, albeit for the purpose of improving decoding speed. Focusing on leaf-level unary chains is a good compromise between not predicting any, as it only requires predicting an extra n possible spans, instead of the n^2 that may appear elsewhere, with a majority of unary chains occur at leaf-level (only 11% to 15% of all unary chains found in the newswire sections of the OntoNotes occur at non-leaf spans). We adopt this approach and forfeit the performance gains that might come from successfully predicting unary chains in the remainder of the tree.

To accomplish this we train two models. For the label parser, described above, all unary rewrites have been removed from the data. In cases where a tree contains a unary chain, it is collapsed into its top-most node (e.g., NP $\rightarrow$ NN $\rightarrow$ "cat" becomes NP $\rightarrow$ "cat"). A second model is trained on the original data, and is used solely for predicting leaf-level unary chains. The goal of this model is to predict ($NN \rightarrow$ "cat", which would then be added into the parser-produced tree). This model is described below:

- Let $\{Unary\text{-}Span(i) : 0 \leq i < n\}$ be $O(n)$ Boolean variables such that $Unary\text{-}Span(i) =$ true iff there is a unary chain immediately above the part-of-speech tag at word w_i.
- Let $\{Unary\text{-}Label(i, \lambda) : \lambda \in L$, and $0 \leq i < n\}$ be $O(|L|n)$ Boolean variables such that $Label(i, j, \lambda) =$ true iff there is a unary chain span around the part-of-speech preterminal at word i with constituent label λ.
- Let $\{$UNARY-ATMOST1$(i) : 0 \leq i < n\}$ be $O(n)$ logical factors, each coordinating $Unary\text{-}Span(i)$ with the corresponding set of label variables, $Unary\text{-}Label(i, \lambda)$. When $Unary\text{-}Span(i)$ = true, a single $Unary\text{-}Label(i, \lambda)$ is also true. Otherwise all variables are false.

Unlike the $Span$ variables of the unlabeled parser, the beliefs of the $Unary\text{-}Span$ variables do not influence each other. In this sense they function as independent log-linear classifiers for predicting the presence and label of each leaf-level unary chain.

Decoding

When decoding the labeled parsing model, decoding begins with the unlabeled parser decoding (Sec. 3.1.2, pg. 78) to yield an unlabeled tree. For each span in the tree the corresponding $Label$ variable with the maximum belief is taken as the span label. The unary classifier then predicts the presence (or absence) of a unary chain above each terminal node for all words in the sentence. If a unary rewrite is predicted, it is labeled by taking the $Unary\text{-}Label$ variable with the maximal belief as its label.

Unlike the unlabeled parsing model, constituents introduced through binarization can be identified and removed from the final tree. Spans of this sort are identified by the "@" prefix attached to their labels, and easily removed.

Features

In this section we discuss the features used to compute potentials for many of the factors in the model. Features are required for SPAN, LABEL, UNARY-SPAN, and UNARY-LABEL factors.

Features for Parsing

Experiments on development data indicate that the information most useful for determining the presence of a constituent span is also an effective indicator of the type (label) of that span. Therefore we present one set of features for both the identification and labeling of constituent spans. When these features are used for calculating a potential for a LABEL factor, the label is attached as a prefix to the string representation of the feature, i.e., a feature for identification is "$width - 5$",

Unigram Constituent Span Features	
Feature String	Comment
[width]-3	Width of the span.
[span-start]-2	Index of the start of the span.
[span-end]-5	Index of the end of the span.
[start-word]-scratched	Word at the start of the span.
[start-tag]-VB	POS tag at the start of the span.
[start-cap]-false	Whether the word at the start of the span is capitalized.
[start-word-tag]-scratched-VB	Word and POS tag of the start of the span.

Bigram Constituent Span Features	
Feature String	Comment
[bigram-w-w]-scratched-man	The words at the start and end of the span.
[bigram-w-p]-scratched-NN	The word at the start of the span and POS tag at the end of the span.
[bigram-p-w]-VB-man	The POS tag at the start of the span and word at the end of the span.
[bigram-p-p]-VB-NN	The POS tags at the start and end of the span.

Span-based and Miscellaneous Constituent Span Features	
Feature String	Comment
[bias]	For capturing the prior belief of a span being present
[tags-from-start-2]-DET-NN	POS tags from (and excluding) the start of the span.
[words-from-start-2]-the-man	Words from (and excluding) the start of the span.
[tags-from-end-2]-END-END	POS tags from (and excluding) the end of the span.
[words-from-end-2]-END-END	Words from (and excluding) the end of the span.
[outside-pos-2]-DET-NN-END-END	POS tags outside of the start and end of the span.
[start-footprint-1-2]-NN-VB-DET-NN	A number of preceding and following POS tags.
[contains-tag]-VB	Feature for if span contains the POS tag.
[contains-tag-no]-PRP	Feature for if span does not contain the POS tag.

TABLE 3.1: Features for span-factored constituent parsing. Tables list the features used in our parsing model. All example feature strings are computed for a span $(2, 5)$, on the sentence *The cat scratched the man*.

where one of the corresponding features for labeling is "$NP - width - 5$". Thus the factor potentials for SPAN and LABEL factors (for all possible labels) are each calculated from mutually exclusive sets of features, but are derived from the same information.

We divide the set of features into three conceptual categories: unigram, bigram, and span-based features. Unigram features (1) are derived from a single index i, and contain information related to the word, part-of-speech tag, capitalization, previous or following tags, etc., at a given index. For a span (i, j) we construct unigram features around indices $i, j, i - 1$, and $j + 1$. Bigram features (2) are constructed over pairs of indices. Here we use a subset of the cross-product of unigram features over each index. Many of these features are similar, if not identical to, the features used in edge-factored dependency parsing [102].

Finally, span-based features (3) capture useful information over the entire span, reducing it to events that are observed sufficiently frequently in the data to be useful. Span-based features include (a) POS tag concatenations over the entire span, (b) the presence of a particular POS tag anywhere in the span, (c) the absence of a POS tag from the span, and (d) the span width. Many of these features can be extremely informative for labeling. For instance, a span which does not

contain a verb is unlikely to be a verb phrase, but that classification might not be captured well by relying solely on information at the edges of the span. A more complete description of these features is provided in Table 3.1.

Features composed of words tend to be sparse, and part-of-speech tags provide a useful back-off for creating a more robust statistic. The use of part-of-speech tags in the construction of features is enabled by either relying on gold part-of-speech tags, or by first pre-processing the data with a statistical part-of-speech tagger [41]. While these annotations are quite common, it should be noted that grammar-based parsing systems can, and typically do, include grammar rules which generate part-of-speech tags and words. Thus these models do not require this information.

Features for Unary Rewrite Prediction

Features for unary predictions are much simpler, largely due to our decision to focus solely on leaf-level unary spans (i.e., spans of the form (NP (NN *cat*))). This not only reduces the number of edges for which it is necessary to predict unary spans from n^2 to n, but it eliminates the need for more sophisticated span-based features drawn from tree structure. Instead the unary prediction features are simply n-grams over a three-word window, including both words and POS tags. For the labels of unary chains, the labels are again concatenated onto the feature string prior to being converted into a parameter vector index.

Pruning

The labeled parsing model instantiates a single $Span$ variable and an L-sized set of $Label$ variables for each pair of indices (i, j). However, many of these spans are unlikely to be present in the predicted tree, and many can be pruned away using simple corpus statistics, removing them from the model's consideration. For instance, constituent label variables can be heuristically pruned from the graphical model based on span width, yielding significant improvements to training and decoding speed while having no discernible effect on performance. We observe that the number of distinct constituent labels found at each span width decreases quickly as the width of the span increases (Fig. 3.9). For instance, spans which contain more than 15 words are extremely unlikely to be noun phrases, and the size of the graph can be greatly reduced by choosing not to instantiate NP $Label$ variables for such spans.

Following this strategy we construct a pruned label set for each span width. Each set includes all constituent labels observed at that width in the training data, and all labels observed in spans with larger widths. We find this approximately triples the speed of graph construction and inference, while still guaranteeing that the correct label will not fall outside of the pruned space during training. The effect on performance is negligible, and often positive.

Another method of improving the speed of the parser is to reduce the set of span labels introduced in the conversion to binary-branching trees. This process creates a corresponding "binarized" label for most nonterminals (@NP, for NP), and therefore nearly doubles the size of the label set. Reducing this set to a single @ label can also significantly reduce the size of the graph and improve decoding speed. Both methods are simple to implement, and produce multiplicative increases in efficiency while having little effect on performance.

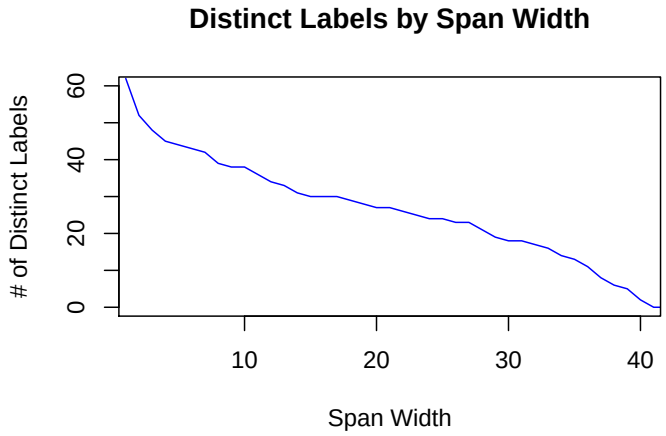


FIGURE 3.9: Number of distinct span labels at varying span widths. Statistics were collected from the OntoNotes version of the WallStreet Journal Corpus, converted to binary-branching trees, described in Sec. 3.1.4 (pg. 89).

3.1.3 Grammatical Rules as Factors

Almost without exception[8], research in constituent parsing has focused on grammar-based approaches. This is not surprising, as grammars have a long-established history in the linguistics literature, and probabilistic grammars have been used with great success as the basis of many probabilistic parsers. In contrast, the approach to parsing presented in this chapter is entirely grammarless: the model's variables are constrained such that any configuration of variables must represent a valid tree, but there are no dependencies between constituent labels (as there are in a grammar). This leads to very efficient parsing, as the size of the grammar can otherwise contribute significantly to the parsing algorithm's complexity. But this architecture is not without its disadvantages.

Constituents can be predicted with great accuracy when spans are small, and features are both informative and well-represented in the data. These features become sparser as span widths increase, making span-factored prediction more difficult and less accurate. In these circumstances, modeling dependencies between parent and child constituent spans may improve accuracy in ways that would be difficult to capture with word or tag-based features alone. These are the kind of dependencies captured by a grammar.

In this section we discuss how factors may be used to represent grammatical rules, and how the weights of these factors can be learned using the perceptron algorithm. These rule factors can then be added to the model *selectively* during inference. This approach offers a unique solution to the parsing problem: shorter spans can be predicted from local features alone, while wider spans benefit from a sparse set of rule factors.

[8]Shallow parsing [103] and NP-chunking [104] do not rely on grammars, but both rarely attempt to define a deep constituent tree from top to bottom, and therefore are not parsing in the same sense.

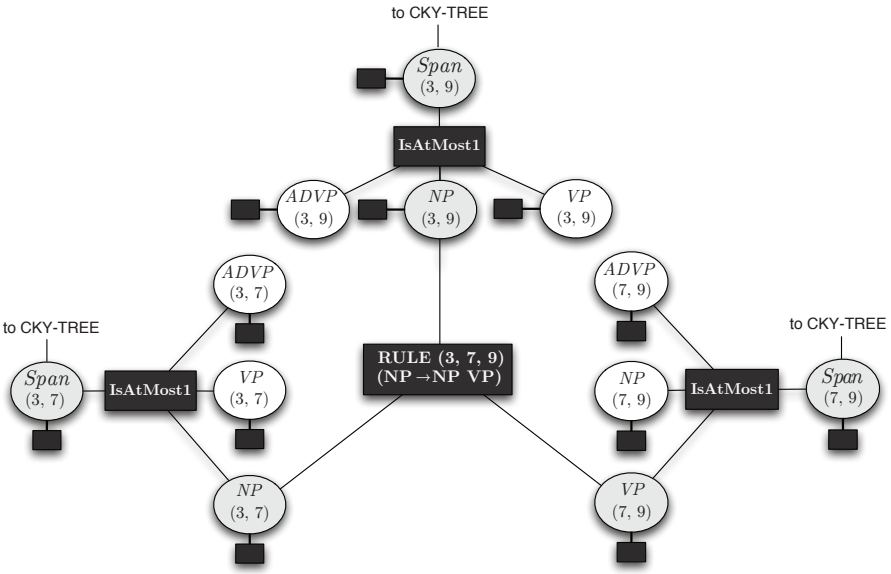


FIGURE 3.10: Grammatical rules as factors. A RULE factor connecting three $Label$ variables. Not every span or constituent label must participate in a rule and the number of rules added to the labeled bracket model may be quite small. Rules act only to fix up mistakes made by the labeled parsing model, where features derived from the input string lead it to incorrect solutions. Here shaded variables are Boolean variables with true values.

Rule Representation

In a binary-branching tree with no unary chains, a grammatical rule $r \in R$ has the form $X \rightarrow Y\,Z$, where $X, Y, Z \in V$, the nonterminal label set. Rule r represents a parent constituent (i, k) with label X, and its two child constituents, one of type Y over indices (i, j) and one of type Z over indices (j, k). In a weighted context-free grammar (WCFG) each rule r is paired with a real-valued non-negative weight $\phi(r)$. Representing a grammar rule as a factor provides a natural analog for this weight: the factor's potential. Formally we define this type of factor as follows:

- Let $\{\text{RULE}(i, j, k, X, Y, Z) : 0 \leq i < j < k \leq n; X, Y, Z \in L\}$, be a set of ternary factors which connect $Label(i, k, X)$, $Label(i, j, Y)$ and $Label(j, k, Z)$ Boolean variables. Each rule factor implements a soft NAND logic, contributing a potential of $\phi(r)$ when all three variables are true and mimicking a weighted CFG rule $X \rightarrow Y, Z$ with a weight of $\phi(r)$.

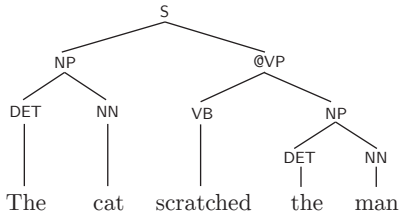
A rule factor connects a triple of $Label$ variables as described above (and as illustrated in Fig. 3.10), contributing its potential to the model's score. If any of the three variables is false, the factor contributes a potential of 1.0 and the score is unchanged. However, if all three variables are true the factor contributes a non-unit potential. The higher the potential, the higher the model's score for this variable configuration. Like other soft factors, this potential is the sum of exponentiated feature weights. Here we rely only on simple features: the string representation of the rule ("$X \rightarrow Y\,Z$"), and a pair of back-off features ("*X → Y LEFT*" and "*X → Z RIGHT*").

In theory rule factors could coordinate between every valid triplet of $Label$ variables (valid in the sense that the indices of the three spans are $(i, k), (i, j), (j, k)$, such that $i < j < k$). This would still differ from a grammar-based model, like a weighted CFG [105], as the score of the

tree is also calculated from unlabeled span variables, in addition to the rule factors over labeled span variables. However, what is truly unique about our approach to grammar-based parsing is that grammatical rules do not need to specify this full derivation, and can instead be selectively added to the graph. There are many ways in which this might be accomplished. The method presented here can be thought of as a coarse-to-fine strategy for grammar-based parsing with factor graphs: first, the factor graph for the labeled parsing model is constructed and inference is run on this graph. Second, subject to some criteria, rule factors are placed over selected triples of $Label$ variables. Third, inference is run on the full graph and the model is decoded identically to the labeled parsing model (Sec. 3.1.2, pg. 82).

To understand how such an architecture might improve parse accuracy, consider the example sentence "*The cat scratched the man*" and the beliefs of four label variables after inference:

$Label$(0,5,S)	1.0
$Label$(0,2,NP)	0.7
$Label$(2,5,VP)	0.2
$Label$(2,5,@VP)	0.3



A common mistake made by the labeled parser is the mis-labeling of constituents as their binarized counterpart. In this example the VP *scratched the man* is erroneously predicted as the binarized VP, @VP. This is a difficult problem to solve when relying solely on information in the input string to predict constituent spans. However, the rule S $\rightarrow$ NP VP is common in many English corpora. If a rule factor is added to the graph which connects $Label(0, 5, S)$, $Label(0, 2, NP)$, and $Label(2, 5, VP)$ variables, and the factor's non-unity potential is high enough, the model's beliefs regarding the label of span $(2, 5)$ can be significantly altered to predict the correct label.

What criteria should be used to determine where to add rule factors? There are many possible solutions, but the approach pursued in this dissertation is to threshold label variable beliefs. We inspect all variable beliefs, calculating the product of all true beliefs for variables in each triple whose indices form a valid parent/child relationship. For every product that is above some set threshold (which we set to 0.1), a rule factor is added to that location in the graph. Other sources of information that might serve as useful criteria are the span widths (to focus on more problematic sections of the tree), the types of labels being considered (whether any are introduced via binarization), or the presence of certain word or tag combinations, but exploring these criteria is beyond the scope of this dissertation. We now turn to a discussion of how to learn the feature weights associated with RULE factors in a coarse-to-fine setting.

Learning Rule Weights

We use the perceptron algorithm to learn a set of weights for grammar rules. This approach assumes a pre-trained labeled parsing model, and the parameters of this model are held fixed while rule weights are learned. For each sentence in the training data, we begin by constructing a labeled parsing model, performing belief propagation inference, and obtaining posteriors for $Label$ variables. Initially the set of rules is empty and does not contain any rules. Otherwise, a rule is placed over any triplet of $Label$ variables whose product of beliefs is above a threshold (0.1). We refer

```
1: function LEARN-RULES(x, w, iters, ϵ)          ▷ data, parameters, iterations and leaning rate
2:     for i ←iters do
3:         for x ← x⃗ do
4:             T_test ←LABEL-DECODE(RUN-BP(GRAMMAR-MODEL(x))          ▷ Tree predicted by model
5:             T_gold ← x                                              ▷ Tree from training data
6:             for r ← δ(T_test) ∪ δ(T_gold) do          ▷ All rules from both trees' derivations
7:                 for f ← φ(r) do
8:                     w(f) ⊕← (count(r, T_gold) − count(r, T_test)) ∗ ϵ
9:                 end for
10:            end for
11:        end for
12:    end for
13:    return w
14: end function
```

FIGURE 3.11: Pseudocode for rule factor learning with the Perceptron algorithm. The parameter vector w is initialized by training a labeled parsing model, and these parameters are held fixed during rule learning. Rule weights are updated proportional to the difference in the number of times the rule *should* occur in the tree, as specified by the gold standard, and the number of times it actually does occur in the predicted tree.

to the resulting model as GRAMMAR-MODEL. We then decode the model using the same strategy used for decoding the labeled parsing model.

The perceptron algorithm is error-driven. The parse tree decoded in the previous step is decomposed into a set of constituent spans, which are compared against the spans of the gold tree. Rule weights receive an update proportional to the amount of times they appear in the gold tree and not the test tree, multiplied by some learning rate ϵ. Rule factors can affect $Label$ variable beliefs in both ways: they can either increase the beliefs of the connected variables, or decrease them to prohibit unfavorable combinations. For instance, in the previous example the rule $S \rightarrow NP\ @VP$ is implied by the tree produced by the labeled parsing model. In comparison to the gold tree, it occurs once more than it should, and it is corrected with an adjustment of -ϵ, penalizing this configuration. In addition, the perceptron method has a advantageous base-case property: learning starts with an empty grammar, and rules are added to the grammar when their weights are first updated. We cap the size of the grammar at 1,000 rules. Further updates continue to update rule weight but do not introduce new grammatical rule factors. Pseudocode for this algorithm is listed in Fig. 3.11.

Why pursue an alternate form of optimization when this could be done using the same belief propagation inference and SGD optimization used in other models? This is due to the sheer number of potential variables that need to be scored by the model even before optimization. One strategy for circumventing the massive is to heuristically prune the graph *prior* to instantiating the full set of rule factors. This can be achieved using relaxed marginal inference [106], or the more recent delayed row and column generation strategies [107] which retain a guarantee of optimality. A second strategy, and the one we pursue, is to use an alternative optimization strategy where the set of rules can be explored piecewise, starting with an empty grammar and adding rules incrementally, without immediately including a large number of rule factors in the model. The error-driven nature of the perceptron algorithm offers this, and has the additional benefit of being very fast in practice.

	Train	Test
ABC	1195	199
CNN	5092	1521
MNB	509	245
NBC	552	245
PRI	1707	394
VOA	1512	383
WSJ	10793	1262

TABLE 3.2: OntoNotes data statistics. The train and test split is a recreation of previous work [6]. For the OntoNotes distribution of the Wall Street Journal corpus, which was not used for evaluation in previous work, we use the standard partition of sections 0-21 for training and section 23 for testing.

3.1.4 Experiments

In the previous sections we introduced three factor graph models of phrase structure syntax. The first model is an unlabeled parser presented in Section 3.1.2 (pg. 69), which consists only of the span variables necessary to represent the desired tree structure, the corresponding set of unary factors, and a globally-connected combinatorial CKY-TREE factor. The second model extends upon this to produce a labeled parser, and predicts unary chains using a separate classifier (Section 3.1.2, pg. 79). We refer to this model here as the labeled bracket model (LBM). Finally, we augment the label parser with a sparse set of grammatical rule factors with weights learned via the perceptron algorithm (which we refer to as +RULES). In this section we compare the performance of models, both in terms of parse accuracy and in decoding speed, to more established state-of-the-art parsers.

Unless otherwise noted all experiments are performed using 30 iterations of SGD, with a learning rate $\eta = 0.01$ and L2 regularization (Gaussian with a variance of 1.0). These parameters were chosen based on performance on the ABC corpus, which we use as a development set, but we find this training strategy to be very robust to changes in these parameters: $0.001 \leq \eta \leq 0.1$ all result in similar performance if given at least 30 iterations of training. Inference is performed via belief propagation, which is exact and converges in two iterations for the parsers which do not utilize grammar factors.

Data Sets

For evaluating parse accuracy we use the OntoNotes (v.2) data set [108], which has been used in previous work to evaluate discriminatively trained factor graph parsing models [6]. We reproduce the data preprocessing pursued in that work, partitioning the data to achieve an approximate 3:1 train to test set ratio, remove sentences longer than 40 words, remove empty leaf nodes, and coarsen all nonterminal labels (e.g., NP, not NP-PRD). Statistics for the resulting data set are shown in Table 3.2. In supplementary parsing experiments we make use of the OntoNotes distribution of the Wall Street Journal (WSJ) Corpus, filter out sentences with more than 40 words, and adopt the standard train/test split. Note that the annotations found in this corpus differ somewhat from the standard Penn Treebank distribution, and therefore the parse accuracy of the state-of-the-art models will differ from the well-established results published on the Penn data.

Parsers

We compare against three parsers representing the state-of-the-art in parsing in traditional grammar-based parsing, and a CRF parser which most closely resembles our own.

- **Stanford-PCFG**: A reference implementation of a PCFG parser [109], as found in the Stanford Parser distribution [9]. PCFGs are at the heart of many of the advances in parse accuracy made over the last two decades. They are simple and well-understood, and we therefore consider them a good candidate for a baseline parsing model.
- **Stanford-Lex**: An early improvement to standard PCFGs was the inclusion of lexicalization: additional markup which annotates constituents with their head word. We include results from the lexicalized parser included in the Stanford Parser [110].
- **Berkeley**: The Berkeley Parser iteratively refines grammar nonterminals, splitting coarse labels to more specific ones, and aiming to capture more subtle phenomena with a more sophisticated grammar [94].
- **F&M09**: [6] present a CRF-based parser which discriminatively learns rule weights, and like traditional PCFG-based approach relies on a dynamic programming chart for efficient decoding.

Evaluation

The standard metrics for evaluating phrase structure parsing are precision, recall, and F_1, as calculated over constituent spans. Precision describes what percentage of the predicted spans are correct:

$$precision = \frac{true\_spans}{predicted\_spans} \tag{3.16}$$

Recall describes what percentage of the gold standard spans are predicted by the model:

$$recall = \frac{true\_spans}{gold\_spans} \tag{3.17}$$

The F_1 score is the harmonic mean of precision and recall:

$$F_1 = \frac{2 \times precision \times recall}{2 + precision + recall} \tag{3.18}$$

For labeled evaluation, a span is considered correct if an identical span is found in the gold tree, matching in start index, end index, and label. In unlabeled evaluation a span is considered correct if it matches a span from the gold tree in just start and end index.

An additional metric is the crossing bracket score (CB) indicating the percentage of test spans (sometimes referred to as brackets) which cross gold spans:

[9] V. 1.68, `http://nlp.stanford.edu/software/lex-parser.shtml`

$$CB = \frac{1}{|T_{test}|} \sum_{Span(i,j) \in T_{test}} \begin{cases} 1, & \exists Span(k,l) \in T_{gold} : i < k < j < l \\ 0, & \text{otherwise} \end{cases} \tag{3.19}$$

where T_{test} is the parser-produced tree, T_{gold} is the gold standard tree, and $|T|$ denotes the number of spans in tree T. Crossing bracket errors are sometimes considered more serious than mislabeling a span, or predicting an extra span that nests properly within the surrounding structure. A final metric, no-crossing (NC), is the percentage of all sentences which contain no crossing brackets. We score parse trees using Evalb[10], a reference implementation of these metrics.

Unlabeled Parsing

In this section we present results for an unlabeled parsing task. This serves three goals: (1) it allows us to gauge approximately how accurate unlabeled tree structure will be in a joint model, (2) it demonstrates how sensitive the parser's performance is to different feature sets, and (3) we can determine whether information from $Label$ variables in the labeled parser improves the model's prediction of unlabeled structure.

Without constituent labels, evaluating parsed trees against gold trees puts the model at a severe disadvantage, as it is not possible to identify which spans are introduced through binarization. Therefore we present two sets of evaluations. One set of evaluations is performed against gold trees. Performance, measured in terms of F1, is expectedly lower in this scenario as all test trees are binarized, and all binarized trees have an identical number of constituent spans, n-1. The number of spans found in the gold trees will never exceed this number, and is in all likelihood much smaller than in their parser-produced counterparts. These results are presented in Table 3.3 (*right*).

A second evaluation uses binarized gold trees in place of the true gold trees. We also prune all unary chains from the tree, making these results more closely reflect the performance of the model on its training objective. These results are presented in Table 3.3 (*left*).

We evaluate three model configurations. To demonstrate the importance of rich feature sets we present one variant of the unlabeled model which uses very simple features, and one that uses the rich feature set presented in Section 3.1.2. In the simple feature model we use two features: the concatenation of all part-of-speech tags within the span, and the concatenation of the two part-of-speech tags immediately outside of it, making it a discriminatively-trained version of the constituent context model [68]. We refer to this model as CCM, and the unlabeled parser as BM (bracket model). Additionally we include results from the labeled parsing model (LBM), without unary chain prediction, but evaluate it solely on the accuracy of its (unlabeled) structure.

The results reveal the importance of a rich feature set. The rich features of the BM model result in an average F1 of 56.68% (gold evaluation), a 3.23% F1 improvement over the CCM model. This is not surprising: without a grammar, these features are the sole source of information assessing how likely a sequence of tokens corresponds to a constituent span. The global CKY--TREE constraint merely provides an optimal way of filtering out incongruent spans.

Perhaps the most interesting observation from these results is the significant improvement in unlabeled accuracy gained from incorporating a set of $Label$ variables into the model. The LBM

[10]Evalb, `http://nlp.cs.nyu.edu/evalb/`

		Unlabeled Evaluation									
		Binarized Trees					Gold Trees				
Data	Feats	CB	NC	P	R	F1	CB	NC	P	R	F1
ABC	CCM	4.4	26.1	73.1	62.4	67.3	3.4	33.3	53.0	54.7	53.8
	BM	3.5	32.2	78.5	67.1	72.3	3.0	40.6	56.1	57.8	57.0
	LBM	2.9	11.6	80.8	69.1	**74.5**	2.5	5.5	57.2	58.9	58.4
CNN	CCM	3.5	30.1	74.3	61.7	67.4	2.9	37.8	53.5	52.7	53.1
	BM	2.6	41.2	80.7	66.9	73.2	2.2	47.9	57.5	56.5	57.0
	LBM	2.6	40.8	82.2	68.4	**74.7**	2.1	51.2	59.1	60.7	**59.9**
MNB	CCM	2.6	24.1	70.0	58.2	63.6	3.6	34.4	50.5	49.7	50.1
	BM	3.9	28.7	73.4	60.8	66.5	3.3	37.4	51.9	51.0	51.4
	LBM	2.8	25.5	76.7	63.9	**69.7**	2.4	19.6	43.5	53.7	**54.1**
NBC	CCM	4.9	16.5	71.5	61.6	66.2	3.8	24.8	48.6	53.2	50.8
	BM	4.1	24.0	76.1	65.4	70.3	3.4	28.9	50.8	55.5	53.0
	LBM	3.7	27.7	78.2	67.3	**72.3**	3.2	32.7	52.9	57.5	**55.1**
PRI	CCM	4.4	21.5	72.1	61.2	66.2	3.7	27.0	53.4	53.8	53.6
	BM	3.0	39.7	80.8	68.6	74.2	2.7	45.7	58.4	58.7	58.5
	LBM	2.5	41.0	84.3	72.2	**77.8**	2.4	48.8	60.0	60.3	**60.1**
VOA	CCM	4.0	17.8	78.8	70.5	74.4	3.4	21.4	57.3	61.3	59.3
	BM	2.9	29.5	84.5	75.6	79.8	2.5	32.4	61.1	65.4	63.2
	LBM	2.5	5.7	87.2	78.1	**82.4**	2.1	18.2	62.6	67.4	**64.9**

TABLE 3.3: Performance of the parsers in an unlabeled evaluation. Not surprisingly, the feature rich unlabeled parser (BM) significantly outperforms the feature poor variant (CCM). More interestingly, the labeled parser (LBM) produces more accurate parse trees, demonstrating the influence of $Label$ variable beliefs on corresponding $Span$ variables. Naturally, all parsers perform better when compared against the binary-branching variants of the gold trees, with unary chains filtered out (*left*), than to the original gold trees.

performance is 2.0% F1 (gold) higher than the BM. Recall that the LBM contains a set of $Label$ variables, but in this evaluation these variable beliefs are not considered during decoding. This means that the beliefs of the $Label$ variables, determined in part from a distinct set of label features, influence the beliefs of $Span$ variables in a very significant, and positive way. As it pertains to joint inference, these results provide a reason to train a labeled parsing model even when $Label$ variables are not used in coordinating between the two models.

Label Parsing

Evaluating the labeled parsing model can be done in a more standard manner. Here we present the traditional labeled evaluation (precision, recall, and F1), where a constituent span is considered correct only if there is a corresponding span in the gold tree with a matching start index, end index, and label (Table 3.4, *right*). We also present a corresponding unlabeled parsing evaluation in order to isolate the parser's ability to predict constituent span boundaries from its ability to label these spans (Table 3.4, *left*).

Data	Model	Bracket Evaluation					Labeled Evaluation		
		Prec	Recall	F1	CB	NC	Prec	Recall	F1
ABC	LBM	82.80	79.65	81.20	1.72	50.78	79.75	76.72	78.20
	+Rules	–	–	–	–	–	80.66	77.92	**79.26**
	F&M '09	–	–	–	2.28	46.88	70.18	70.12	70.15
CNN	LBM	86.40	83.12	84.73	0.96	67.41	83.30	80.14	81.69
	+Rules	–	–	–	–	–	84.66	80.88	**82.72**
	F&M '09	–	–	–	1.11	70.06	76.92	77.14	77.03
MNB	LBM	80.77	76.18	78.41	1.40	59.26	76.98	72.61	74.73
	+Rules	–	–	–	–	–	79.09	73.05	**75.96**
	F&M '09	–	–	–	1.88	59.03	63.97	67.07	65.49
NBC	LBM	80.77	77.37	79.04	1.41	49.66	74.81	71.67	73.20
	+Rules	–	–	–	–	–	76.94	72.34	**74.57**
	F&M '09	–	–	–	2.67	48.92	59.72	63.67	61.63
PRI	LBM	85.01	82.15	83.56	1.44	57.40	82.70	79.92	**81.29**
	+Rules	–	–	–	–	–	81.90	80.44	81.17
	F&M '09	–	–	–	1.72	56.70	76.22	76.49	76.35
VOA	LBM	85.71	81.96	83.80	1.63	43.34	83.55	79.89	81.68
	+Rules	–	–	–	–	–	83.07	82.69	**82.88**
	F&M '09	–	–	–	2.44	38.89	76.56	75.74	76.15
Onto WSJ	LBM						84.88	80.33	82.54
	+Rules								84.32
	Stanford-PCFG						80.71	79.86	80.28
	Stanford-Lex						81.64	81.65	81.64
	Berkeley						86.61	85.81	**86.21**

TABLE 3.4: Labeled parsing performance. The feature-rich labeled parsing model (LBM) provides significant gains over previously published CRF parsing scores (F&M09), and the inclusion of rule factors provides a 0.96 average improvement in labeled F1. The state-of-the-art Berkeley parser still offers better performance than any of the factor graph-based parsing models, yielding an 1.89 labeled F1 improvement on the OntoNotes distribution of the Wall Street Journal corpus, but it is unclear how to perform joint training with the Berkeley parser as a component model. Factor graph-based approaches offer competitive performance, and are proven to work in a joint setting.

In comparison to the CRF-based approach of F&M09, the labeled parsing model (LBM) performs significantly better in terms of labeled F1, 78.47 vs. 71.13 average across the six smaller newswire corpora, an improvement of 7.34. Augmenting the model with rule factors increases performance by an average of 1.4 F1, to 79.43. These results are the highest reported for this data set. While we cannot explain why our model performs so much better than the F&M09 parser, there are two likely hypotheses. First, as illustrated in Section 3.1.4, different feature sets can lead to significantly different performance measures. A comparatively uninformative feature set could cause the F&M09 parser to perform poorly, but this is unlikely. Rich feature sets are one of the biggest attractions of discriminative models like the LBM and F&M09, we have done our best efforts to remain generally comparable to F&M09 in terms of features. Our feature set is far from exhaustive, and we have not deliberately incorporated any unorthodox or exceedingly sophisticated features.

A second hypothesis is that the model structure itself is responsible, and there are many aspects of the system which inherently differ from the F&M09 system (e.g., an alternate inference method, the span-factored vs. rule-factored nature of the model, the alternate method of labeling spans, or the separate classification of unary chains). However, it is beyond the scope of this work to perform piecewise analysis against the F&M09 system. We can only conclude that, in comparison to parsing models used previously for joint inference, our model produces more accurate parse trees when functioning as a standalone parser. Given that it is also part of a more flexible framework (and is capable of connecting to other models in ways that F&M09 cannot), it fulfills our needs as an attractive syntactic representation for joint inference.

In Comparison to State-of-the-art Parsers

The CRF-based parser of F&M09 produced the best previous results on the set of six smaller OntoNotes newswire corpora, prior to our work, but how do these CRF/factor graph-based approaches compare against more established parsers? Table 3.4 shows the performance of the Stanford-PCFG, Stanford-Lex (lexicalized), and Berkeley parsing models on the WSJ section of the OntoNotes. The LBM outperforms both Stanford parsers by 2.24 and 0.9 labeled F1. This is a surprising result: without relying on a probabilistic grammar, the LBM offers better performance than two reference implementations of previous grammar-based state-of-the-art models. Current state-of-the-art models, like the Berkeley parser, still offer better performance than the LBM or +Rules models. While this amounts to a very significant improvement of 3.67 over the LBM and 1.89 over the +Rules models in terms of labeled F1, it is still unclear how to perform a grammar-refining training method in the context of joint optimization.

But recall that our intent in pursuing the factor graph representation of phrase structure syntax is not to improve parse accuracy, but to improve performance on other NLP tasks which might benefit from joint training with this syntax. In that respect, there may be other aspects of the tree that are more important to our goals than labeled F1. If we restrict our evaluation to focus solely on noun phrase detection, we find the factor graph parsers, LBM and +Rules, outperform all other parsing models on the NP identification task (Figure 3.5). This result is of particular importance to joint inference, as tasks like named entity recognition hinge on the successful identification of noun phrases. For these tasks, a model of syntax which excels at NP identification may prove more valuable than one which sacrifices NP identification to improve general parse accuracy. This issue is discussed further in Section 4.1.1. However, one must concede that all margins, though statistically significant, are rather small. The ability to use the LBM representation as a component in a joint model is the principal advantage of these new models.

How does the LBM outperform the Berkeley parser on this task? The span-factored nature of the model, and the rich set of features used to make span predictions, leads to better parsing accuracy on smaller span widths where these features are extremely informative and less sparse. Noun phrases occur frequently at smaller span widths, and thus benefit more heavily from the LBM parser's strengths.

Grammar-based Parsing

In the previous section we showed the performance of the LBM parser exceeded that of traditional grammar-based approaches (Stanford-PCFG and Stanford-Lex), and other discriminatively trained

Model	NP-P	NP-R	NP-F1
LBM	90.07	90.02	90.42
+Rules	90.15	91.07	**90.60**
Stanford-PCFG	83.88	85.74	84.80
Stanford-Lex	85.07	87.83	86.43
Berkeley	89.17	90.96	90.06

TABLE 3.5: NP prediction results on WSJ. While some established parsers outperform LBM in general parsing F1 (82.54 LBM vs. 86.21 Berkeley), the LBM outperforms all evaluated parsers on a measure more reflective of its potential in NP-based joint modeling scenarios, while remaining asymptotically faster.

log-linear parsing models ([89]). If NLP tasks are to benefit from joint inference, as previously illustrated by [6], then grammarless parsing may provide a suitable syntactic representation. However, competing with modern parsers without the high-level dependencies, of the sort captured by grammatical rules, may not be a realistic goal.

In preliminary work, orthogonal to our goals in joint inference, we describe a method for incorporating a set of grammatical rule factors into the labeled parsing model (Section 3.1.3, pg. 85). We cap the maximum grammar size at 1000 rules, and train the model using 30 iterations of the perceptron algorithm with a learning rate of $\epsilon = 0.1$. For each training instance the LBM is used as a coarse parsing model, and grammatical rule factors are inserted into the model for all applicable 3-tuples of $Label$ variables where the product of variable beliefs exceeds 0.1.

On the six newswire corpora used in F&M09 we see a significant improvement in average labeled parse accuracy, from 78.5% to 79.4%, when incorporating rule factors into the model. This improvement is due to both better parse accuracy in the general sense, and also due to better identification of binarized nonterminals, a problem more endemic to our span-factored parsing models. On the largest corpus, the OntoNotes distribution of the Wall Street Journal, we find the largest increase in labeled parsing performance, from 82.54 to 84.32, a 1.78 improvement. While these results are still lower than the results of the Berkeley parser, there are many possible avenues for future work including different rule features, a more sophisticated criteria for determining where to add rule factors, and the prospect of integrating LBM and rule model training, without the need for a separate perceptron-based training procedure.

Decoding Speed

The grammarless factor graph parsers presented here differ drastically from previous grammar-based approaches, and the effects of these differences extend to parsing speed. In this section we analyze the decoding speed of these parsers in comparison to more established grammar-based parsing models. We begin by comparing the decoding speed of grammarless parsers to PCFG and lexicalized parsers and examining the effect of sentence length on decoding spend. In addition we also examine the decoding speed of the rule-based parsing model, and how it compares to the PCFG parser as the grammar size increases.

Grammarless Parsing vs. PCFG Parsing

The complexity of parsing using a PCFG model is $O(|G|n^3)$, where G is the grammar size (the number of rules in the grammar) and n is the length of the sentence. Like the PCFG model, the

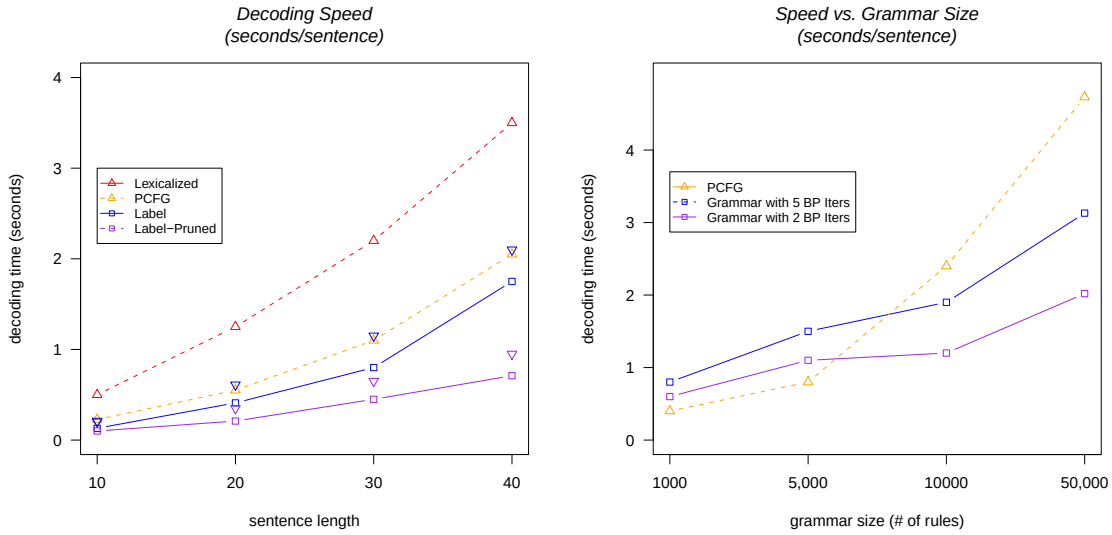


FIGURE 3.12: A comparison of parser decoding speeds. Comparing the decoding efficiency of the labeled parsing model, in both its standard and pruned configuration, with standard PCFG and lexicalized parsing baselines (*left*), shows that the factor graph models generally decode faster than their grammar-based counterparts. While feature extraction cost (illustrated by the inverted caret) hinders the speed of factor graph models, the pruned model remains the fastest across sentences of all lengths. Examining the effects of grammar size (*right*), the rule-based factor graph approach, decoded using five iterations of BP, offers better asymptotic performance than the PCFG model as the grammar size tends to infinity. The models are comparable when using grammars containing approximately 7,000 rules, a reasonable grammar size by modern standards.

grammarless approach uses the same chart-based decoding strategy (responsible for cubic complexity), but without the grammar its decoding speed is simply $O(n^3)$ in the unlabeled case. Adding the machinery for labeled parsing increases this to $O(|L|n^2+n^3)$, where L is the set of nonterminals used to label constituent spans. This is because when using belief propagation, adding additional factors to the model increases the complexity additively, reflecting the additional messages which need to be computed.

In practice this alternative complexity results in an improvement in decoding speed across all sentence lengths. The plot in Figure 3.12 (*left*) depicts the decoding times of four models: the Stanford lexicalized and PCFG models, the labeled parsing model, and the labeled parsing model using the pruning techniques described in Section 3.1.2 (pg. 84). With pruning the grammarless parsing model decodes four times faster than the lexicalized parser, while simultaneously producing more accurate parses. It is also evident from the plot that the grammarless parsing models scale to longer sentences better than their grammar-based counterparts, with the pruned model having a much lower delta between $n = 30$ and $n = 40$ (0.18) than the PCFG parser (0.98).

However, it must be noted that for each example feature extraction must be performed before the model can be constructed, and the sentence decoded. The position of the carets, shown above each data point for the labeled and pruned parsers, indicate the total decoding speed when including the feature extraction cost. In the case of the labeled model, the feature extraction cost eliminates any advantage over the PCFG in terms of decoding speed, but it still significantly outperforms this

model in parse accuracy.

The unlabeled parsing model (not shown in the plot) offers exceptional parsing speed, decoding length 40 sentences at more than 10 per second, inclusive of feature extraction cost. In joint models, where the cost of inference is inherently larger, the trade-off between speed and accuracy may make the unlabeled parsing model a more attractive option.

Decoding Speed vs. Grammar Size

Incorporating grammatical rules as ternary factors has an additive effect on the cost of runtime, resulting in a complexity of $O(|G| + |L|n^2 + n^3)$, in comparison to the $O(|G|n^3)$ complexity of PCFG parsing. In practice n is often quite small, and the size of the grammar, $|G|$, orders of magnitude larger. For parsers which use latent variable grammar refinement techniques [88], $|G|$ can be in excess of tens of thousands of rules. In this scenario, the additive vs. multiplicative relationships between the cubic n^3 factor and the grammar constant can have a pronounced effect on the decoding times of these two models. We now analyze the effect of varying grammar sizes on decoding speed, comparing the +RULES factor graph and PCFG models.

The plot in (Figure 3.12, *right*) shows the average decoding speed as the grammar is artificially inflated in size from a thousand to fifty thousand rules. We present results from three models: the same reference PCFG implementation used throughout this section (Stanford-PCFG), the rule-based factor graph model decoded after two iterations of belief propagation (as if it was performing exact inference in a non-cyclic graph), and the same model decoded after five iterations of belief propagation, the average number required for convergence. The factor graph-based parsers suffer from a higher overhead, and initially do not decode as efficiently as the PCFG model, when grammars are small. As the grammar size climbs above five thousand, both factor graph models quickly begin to outpace the PCFG model, and as the grammar reaches sizes of fifty thousand, the asymptotic performance for these models becomes much more compelling.

We conclude this section having shown that the grammarless factor graph models presented in this chapter are capable of producing more accurate parses than previous CRF-based parsing models, and both lexicalized and unlexicalized variants of PCFGs. In addition, grammarless models outperform all tested parsers on the task of NP-identification. We also demonstrate how rule factors can be incorporated into the model to provide a significant increase in parse accuracy, though we leave this area largely unexplored. Finally, we show that our factor graph parsing models decode faster than PCFG models (which they outperform in accuracy), and that representing rules as factors in a factor graph model offers better asymptotic behavior, in terms of decoding speed, than PCFGs as grammar size increases.

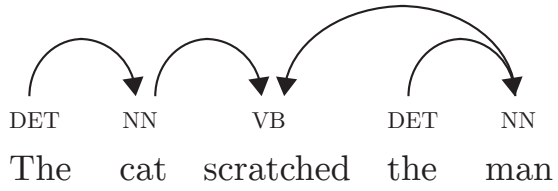
3.2 Dependency Parsing

Theories of phrase structure grammar, which posit that the constituent serves as the basic unit of syntactic structure, represent only a fraction of the many hypotheses regarding the nature of the human syntactic process. An alternate theory, dependency grammar [111], shifts the emphasis away from constituents, stressing instead the role of word-to-word relationships. In this section we formally introduce dependency grammar, and restate a method for efficiently representing dependency grammar within a factor graph model, originally presented by [11] . We also discuss

the advantages of this formalism and the corresponding model representation as it pertains to joint modeling.

3.2.1 Dependency Grammar

In dependency grammar, a syntactic analysis centers around the notion of a *dependency relation*, a syntactic relationship between a pair of words. Typically, and consistent with the presentation of dependency grammar here, a dependency relation expresses a syntactic head-modifier relationship. Recall from previous discussion (Section 3.1, pg. 62) that a syntactic head, also known as a *head-word*, is the word which defines the syntactic type of the encompassing phrase. Thus dependency grammar formulates a syntactic analysis not by the explicit grouping of words into phrases, but by the head words which implicitly define them. Such an analysis may be expressed solely in terms of these pairwise head-modifier relationships. The resulting structure is a directed acyclic graph (DAG) known as a dependency tree:



In a dependency tree all words are connected by a directed path to the finite verb, which serves as the root of the graph. Such an analysis does not represent phrases, only pairwise head-modifier relations, but it does implicitly acknowledge their presence. To illustrate this point, consider the phrase "*man*". As a noun phrase, we can expect the head of the phrase to have a noun part of speech. In this simple case, *the* is a determiner and *man* a noun, leaving only one reasonable choice, *man*, for the head of the phrase.[11] When represented as a pairwise relationship, *man* is considered the syntactic head of *the*, the syntactic modifier. Broadening our focus to the encompassing verb phrase, "*scratched the man*", the word *scratched* is the head of the phrase. The head of the phrase is the syntactic head for any word under the dependency which does not already specify one. Thus *scratched* becomes the syntactic head of *man*, demonstrating how a dependency analysis can be constructed from a corresponding phrase structure tree.

On a more theoretical note, not all dependency trees have a corresponding phrase structure tree. A **projective** tree, dependencies are not permitted to cross one another. For **non-projective** trees this constraint is relaxed, allowing the graph to capture a broader set of phenomena which, although rare in English, are common in languages with free word order (Fig. 3.13). As a consequence, non-projective trees do not have a corresponding phrase structure analysis.

For our purposes, in the context of a joint model for another NLP task, there are many instances where a dependency representation of syntax may be preferable to phrase structure grammar. For instance, it may be more convenient to model an NLP problem with variables whose semantics capture a head-to-modifier relationship. In this situation it is natural to use dependency grammar, as it allows for the two models to be connected using Boolean logic and a one-to-one mapping.

[11] The rules governing which word acts as the head of the phrase in more ambiguous circumstances are specific to each language.

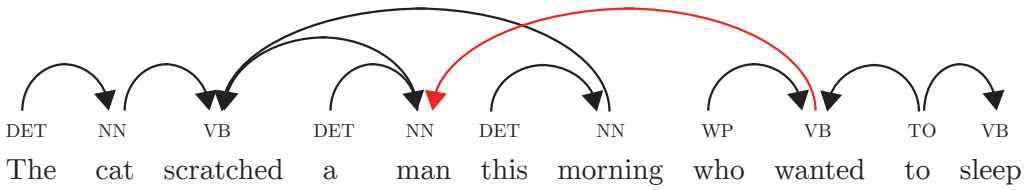


FIGURE 3.13: A non-projective dependency tree. Non-projective phenomena are rare in English, but prevalent in many of the world's languages. Here a dependency across a relative clause boundary makes this tree non-projective.

We find this to be the case for the task of semantic role labeling (Chapter 6), where the goal is to identify pairs of words which occur in a predicate-argument relationship. Predicates are often verbs and together with their arguments align closely with the head-modifier semantics of dependency structure. Capturing this in a span-factored model of phrase structure is difficult, as the verb is "hidden" within the constituent span and not immediately accessible by the model's SRL component.

3.2.2 Representing Dependency Syntax in Factor Graph Models

We propose representing dependency parsing in the edge-factored manner originally presented in [11]. This allows a one-to-one correspondence between syntactic variables and the variables of other edge-factored models (like semantic role labeling). As in phrase structure grammar, it will be convenient to represent the atomic units of syntactic analysis, in this case edges, with Boolean variables. This allows variables of each component model to be combined with using Boolean logic factors.

- Let $\{Link(i,j) : 0 \leq i < n, 1 \leq j \leq n, n \neq j\}$ be $O(n^2)$ Boolean variables corresponding to the possible arcs in a dependency parse. $Link(i,j)$ = true implies that there is a dependency from parent i to child j.
- Let $\{\text{LINK}(i,j) : 0 \leq i < n, 1 \leq j \leq n, n \neq j\}$ be $O(n^2)$ unary factors, each paired with a corresponding $Link(i,j)$ variable and expressing the independent belief that $Link(i,j)$ = true.

As was the case for phrase structure parsing Section 3.1.2 (pg. 69), these components provide the necessary structure for predicting, independently, the basic components of a syntactic analysis, but they also permit many invalid structures. A configuration of variables could represent a graph which contains cycles, multiple heads for a single word, or multiple roots. As before, these problems could be addressed using many pairwise factors, but would require an *exponential* number of such factors, and would hinder the effectiveness of inference. We instead rely upon combinatorial factors to provide an efficient solution to this problem. Depending on the type of syntactic analysis desired, [11] provide two constraints:

- **DEP-TREE**, a global combinatorial factor, as originally presented in [11]. A DEP-TREE factor attaches to all $Link(i,j)$ variables, contributing a potential of 1 iff the configuration

of variables results in a valid non-projective dependency graph. It otherwise contributes a protential of 0, prohibiting the configuration. The calculation of outgoing messages is based on the Matrix Tree Theorem [112].

- **PROJ-TREE**, a global combinatorial factor, also originally presented in [11]. It is similar to DEP-TREE, but further requires that the resulting structure form a valid *projective* dependency graph, prohibiting edges from crossing.

Both constraints calculate outgoing messages in $O(n^3)$ time. For implementation details of these algorithms, we defer to [11]. The features we utilize for this model are derived from the [69] feature set for graph-based dependency parsing, and only differ in that we do not construct features from coarse part-of-speech tags. We similarly refer the interested reader to the detailed description provided in that work.

This representation also allows for higher order models of dependency parsing through the use of additional factors. For instance, second order models, which coordinate between two $Link$ variables, can capture grandparent or sibling relationships. This has been shown to significantly improve parser performance [69]. However, when treated as latent structure in a joint model, a first order representation of syntax may learn a more direct form of syntax, collapsing longer directed paths into single dependencies, and approximating the benefits of a higher order model by capturing the types of structure most useful to the end task. In the context of semantic role labeling (Chapter 6), if a semantic relationship exists between word w_i and w_k, but the syntax has a directed path $w_i \rightarrow w_j \rightarrow w_k$, the desired latent syntactic representation might directly capture this relationship, assigning a strong belief that $Link(i, k) = \mathsf{true}$.

3.3 Conclusions

In this chapter we introduced factor graph formulations of two leading syntactic formalisms: phrase structure, in which a syntactic analysis is a hierarchy of nested constituent spans, and dependency structure, where a syntactic analysis is expressed via pairwise dependencies between a headword and modifier. Both are difficult to efficiently represent in a factor graph, with phrase structure requiring a quartic number of logical factors to prohibit constituents from crossing, and dependency grammar requiring an exponential number of logical factors to prohibit cycles in the graph. But complexity is only part of the problem. The multitude of logical constraint factors required to rule out invalid structures would also introduce many cycles into the graph, resulting in poor inference. To circumvent this we described three global, combinatorial factors which reduce the complexity of representing such structured models.

Our novel contributions are specific to phrase grammar. We begin by introducing the CKY--TREE factor and a factor graph representation of unlabeled phrase structure. This structure is perhaps of little use from a traditional parsing perspective, but may be a suitable representation for joint inference where models are connected across unlabeled components (e.g., a constituent span variable connected to a named entity span variable, both variables representing the boundaries of the span and not the type). We then introduce a set of factors and variables to provide labels to these constituent spans, and the necessary components to create a labeled phrase structure parser. On the OntoNotes data set we outperform a similar CRF-based parser, and against PCFGs and

their lexicalized variants we show that our parser provides a significant improvement in parse accuracy while remaining asymptotically faster. We also demonstrate that the labeled parsing model produces better unlabeled tree structure than the unlabeled variant, motivating the use of label variables even when they are not evaluated directly. This representation is used throughout the dissertation to construct joint models and to serve as latent structure.

Also included in this chapter is preliminary work toward incorporating grammar rule factors into the model. In this approach we utilize the perceptron algorithm to learn a set of rule factor weights in a coarse-to-fine manner on top of the labeled phrase structure parsing model. We show significant improvements in parse accuracy over the corresponding grammarless model, but the answers to many pertinent questions are beyond the scope of this work. These questions include what criteria should determine how rule factors are added to the model, what rule factor features are most useful, what is an optimal grammar size, and how might the learning of rule weights be incorporated into the same procedure which optimizes the other model parameters. In reference to the latter, recent work in delayed row and column generation solves a similar problem [107], where it is computationally difficult to score the number of structures *prior to optimization*, and may be a promising line of research for addressing this issue.

In the second section we introduced dependency structure, and positioned it as an alternative to phrase structure which may be more convenient when the end task relies on head-modifier type relationships. The combinatorial properties of dependency structure again poses a problem for representation in factor graph models, and we turn to the approach of [11] for a solution. We summarize the contributions of this work, introducing a projective and non-projective dependency tree constraint factor. We rely on these constraints to construct joint models for relation extraction and semantic role labeling.

We conclude with a final thought: *is it necessary to have factor graph representations of both phrase and dependency structure?* Both structures encode similar information, which may at first seem to lessen the need for more than a single representation. The underlying headwords of a phrase structure tree can be recovered using a set of language-specific heuristics, of the sort described in [84]. Similarly, a projective dependency tree implicitly defines constituent spans in the nesting of its dependency arcs, and an unlabeled phrase structure trees can be constructed from such a tree, making these representations largely isomorphic. Many NLP systems treat syntax trees as observed, skirting the issue by utilizing rich feature sets which can capture many of the strengths of each formalism, without the need to convert manually from one to another.

However, when pursuing joint models it is difficult to be ambivalent. While it is trivial to convert between dependency and constituent trees when trees are observed, it is far more difficult when trees are represented by hidden variables. It is then necessary to add additional model structure, including higher level dependencies or combinatorial factors. We believe the cost of attempting this kind of "on the fly" conversion to be prohibitive. It follows that if such conversion is ruled out, then some representations will naturally lend themselves better to particular NLP models.

This will become apparent throughout the remainder of the dissertation, where we leverage the syntactic representations introduced here in order to construct joint models of NLP tasks. Chapter 4 combines the phrase structure syntax with a span-factored model of named entity recognition, and we demonstrate state-of-the-art performance on the OntoNotes data set, outperforming both our own baseline and a previous joint model. In Chapter 5 we develop two joint models for relation extraction, one using phrase structure, the other using dependency structure. Our focus in this

chapter is on comparing the effectiveness of each representation on the task. We conclude in Chapter 6, where we develop the most sophisticated joint model presented in the dissertation, applying it to the task of semantic role labeling. Here we show competitive results against participating systems in a shared task competition, without observing any syntactic annotation.

For each of these tasks we provide more traditional results, where the joint model is trained in a supervised manner, either predicting syntax jointly or setting the syntactic variables to reflect gold syntactic analyses. However, we also provide results where syntactic structure is induced via marginalization-based training in a joint model, learning a syntax specific to each task and data set. Learning syntax without the need for syntactic annotations is a more scalable approach to joint modeling in NLP, and we show there is little loss in performance from doing so.

4

Jointly Modeling Syntax and Named Entity Recognition

In this chapter we apply our joint modeling framework to the task of named entity recognition (NER), coupling a phrase structure parser to a span-based NER model. The motivation for this coupling is clear: in theory, named entities should correspond to syntactic noun phrases. Thus knowledge of a sentence's syntactic structure constrains the possible locations of named entities, and knowledge regarding the location of a named entity span eliminates the possibility of any syntactic span that would cross it. We aim to leverage this interdependent relationship to improve performance on both tasks using joint inference.

This is a goal shared shared by previous work. [6] also present a joint model of phrase structure parsing and NER, coupling the two tasks by augmenting the parser's grammar with named entity labels. A grammar rule NP $\rightarrow$ DET NP might be replaced with an augmented rule NP-ORG $\rightarrow$ DET NP-ORG, which augments the purely syntactic NP grammar rule with the entity label for organizations. Here the use of joint inference resulted in a 3.92% average F1 improvement (8.95% max) in named entity recognition accuracy, and a modest 0.71% average F1 improvement (1.36% max) in parsing. However, in this approach augmented rules must exist for each entity label, and this yields a multiplicative increase in the grammar size (Section 4.2, pg. 107 details the adverse effect this has on overall complexity). Additionally, any mismatches between syntax and named entity annotations in the training data need to be filtered out during preprocessing as only entities which nest within the syntax tree can be represented. Our approach circumvents the need for such preprocessing, connecting the syntax and named entity models with soft Boolean logic constraints, and the model's label set grows only additively.

We present two models. One model is a span-based NER model with no syntactic information, to serve as a baseline. The second is the same model, coupled with a model of phrase structure syntax, trained and decoded using joint inference. We refer to these as NER-ONLY and NER-JOINT respectively. For each model we experiment with two feature sets: one intended to more

closely compare to the linear-chain CRF features used in previous work [6], and one meant to showcase the full capabilities of the model. Evaluating on the OntoNotes corpus [108] our joint model with linear-chain features significantly improves accuracy over the baseline with linear chain features, resulting in a 4.6% F1 increase in NER accuracy. When using the full feature set the gains provided by joint inference are less pronounced, but the performance of the feature rich model is comparable to all previous results, and is even able to outperform a system trained in a semi-supervised manner on additional labeled data [28] on some data sets.

We present two additional training scenarios for the joint model. One, which we refer to as the NER-ORACLE model, can observe the "true" values, given in the training data, of syntactic variables during both training and testing. Another, the NER-HIDDEN model, treats syntax as a latent variable during training and testing. We find that the NER-ORACLE model provides the highest overall accuracy, but inducing a latent syntactic structure using marginalization-based training is more accurate than the NER-JOINT model on most data sets.

We begin by presenting an overview of the named entity recognition task, and discuss the relationship between named entities and constituent syntax. We then present our span-based NER model (Section 4.3, pg. 109), which relies on a combinatorial factor for implementing a semi-CRF constraint. Extending this to a joint model with a phrase structure parser is straightforward, and relies only on the inclusion of the parsing architecture presented previously (in Section 3.1.2, pg. 69) and a set of logical factors to coordinate between variables of opposing models. We conclude with a set of experimental results on the OntoNotes data set (Section 4.4, pg. 116), in which we revisit respresent results from feature rich and feature poor variants of our baseline and joint models, and compare against previous work. The models presented in this Chapter were first described in [113].

4.1 Overview of Named Entity Recognition

Named entity recognition (NER) is a fundamental task in the field of information extraction. The goal of NER is to identify and categorize the important entities that participate in the sentence. Entities may be loosely defined as "things of interest", including people, places, organizations, events, dates, times, and so forth. Consider the sentence:

$[\text{The Darjeeling Limited}]_{\text{MISC}}$ reviews on $[\text{metacritic}]_{\text{ORG}}$ prove overwhelmingly positive

Here, "*The Darjeeling Limited*" is an entity of type MISC (a film), and "*metacritic*" an entity of type ORG (a movie review website).

The identification of such entities is a prerequisite step for other IE tasks. For instance, NER is considered an important prerequisite step to relation extraction (Chapter 5), which often begins with the entity boundaries defined, and seeks to identify the relationships between them. NER is also important to coreference, a task in which the underlying entities are identified and linked, even though the form of their mentions may differ (i.e., the president, President Obama, Barack Obama, Barack).

Named entities are typically small, often comprising only a couple of words, and models making use of local dependencies have performed very well on the task. HMMs were among the earliest statistical models applied to the task [114–116], but feature-rich discriminative methods

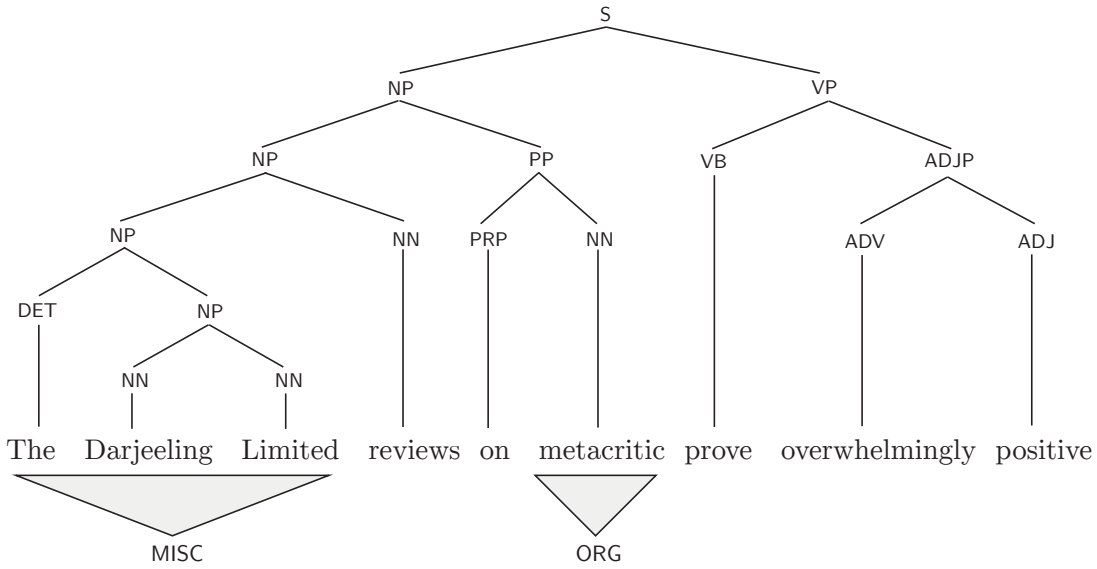


FIGURE 4.1: A sentence annotated with syntax and NER. A simple NER model which looks only at words and word contexts may be mislead by the lexical ambiguity of the phrase *limited reviews* and the rarity of words like *darjeeling*, which form the complex NP *The Darjeeling Limited*, a feature film. When viewed in light of the sentence's syntactic structure, these aspects are less confounding as the NP-structures provide strong cues as to where the named entities might lie.

represent the current state-of-the-art, with approaches based on SVMs [117], linear chain CRFs [39, 118], and extensions which capture some long-distance dependencies, like skip-chain CRFs [54, 119].

However, while sequence models can accurately predict named entities, these models do not reflect the hierarchical nature of syntactic structure. For small entities, this may not be problematic. A sequence model for NER may identify tag sequences like DET NN NN as likely named entity candidates. These patterns occur frequently enough that they can be successfully learned by the model, and the model may attain high NER accuracy over short entities. However, when named entities are longer, these types of sequential statistics naturally become much sparser. If a named entity is truly a type of noun phrase then knowing the underlying syntax could be beneficial: does the proposed entity span align with a syntactic span? Does it align with an NP? As motivation for a joint model of NER and phrase structure syntax, we now discuss the relationship between these two tasks.

4.1.1 The Relationship Between Syntax in NER

Joint inference may be warranted in situations where tasks are interdependent, and knowing the solution to one problem can assist in solving the other. In the case of named entity recognition, entities, being *things*, often closely correspond to the syntactic notion of a noun phrase (NP). In theory, whenever there is a named entity from i to j, there must be an NP from i to j. In practice this is not always the case. Table 4.1.1 shows the extent of this correlation in selections of the OntoNotes (v. 2) data set.

The statistics describe the percentage of named entity spans that align to particular constituent

	NP	NML	QP	ADJP	ADVP	PP	TOTAL	BIN
ABC	59.15	2.77	5.36	1.66	1.66	0.37	72.35	94.15
CNN	52.28	8.38	5.49	1.97	2.10	0.39	71.33	92.94
MNB	59.00	8.79	4.18	0.42	1.67	0.0	77.41	93.48
NBC	54.22	8.44	4.55	0.00	0.65	0.00	77.86	93.02
PRI	51.97	4.37	3.76	1.14	2.89	0.0	64.48	91.24
VOA	39.93	9.51	4.04	2.15	1.17	0.07	50.01	93.91

TABLE 4.1: NER and constituent syntax alignment statistics. The table entries contain the percentage of named entities corresponding to constituent spans of varying kinds in the newswire sections of the OntoNotes (v.2) corpus. Transforming trees using a right-branching binarization strategy (BIN) raises the total percentage of correlation significantly.

spans. The leading assumption, that named entities correspond to noun phrases, occurs approximately half the time. There is some degree of correlation to other constituents, notably NMLs, which mark NP-internal structure, and quantifier phrases (QPs), which describe a complex measure or amount. Both could be considered variants of NPs.

Between 30 to 50% of the named entities do not align to any constituent span in the gold data. Examining these mismatches reveals that this is more a product of the style of treebank annotation than it is a failure of our linguistic intuition that all named entities should correspond to NPs. Treebank-style parse trees occasionally lack structure that would produce a greater degree of correlation between the two tasks. For instance, many NPs fail to explicitly show right-branching internal structure (Fig. 4.2, *left*). Ideally a phrase like *an additional 265 million dollars* should not only correspond to an NP, but should also include an additional NP (*265 million dollars*) nested inside. Instead, many NPs found in the OntoNotes treebanks are much flatter structures, similar to the base-NPs of [120]. Here the nested structure is not explicitly marked, and the correlation between constituent and named entity span is lost.

However, this may not be as detrimental to joint inference as it may at first seem, because the trees used to train the model are *binarized*: each nonterminal constituent forced to have at most two children. This process introduces many more constituent spans, and consequently increases the chance that a syntactic span will align to a given entity. Trees are predominantly right-branching in English, and by using a right-branching binarization strategy many of the constituents (those which share a right edge with the entity) that are not explicitly annotated in the data are reintroduced. Table 4.1.1 shows that using binarized trees raises the degree of correlation to over 90%. The significance of a correlation is reduced, but the degree to which there is a correlation is strengthened.

We also note that the extent to which nested NP structure is made explicit is not consistent. For instance, the name *Al Gore* forms an NP when in isolation, and often forms an NP when nested within a larger structure. Yet there are still many exceptions in which they are not grouped into a single constituent (Fig. 4.2, *right*). While the joint model we present (Sec. 4.3.2, pg. 113) is robust to a lack of correlation in cases where we theorize one should exist (via *soft* Boolean logic factors handling the coordination between models), the inconsistent nature of the annotation scheme may have some effect on the performance of joint models in general.

A constraint that does hold true in nearly all cases is that named entity and constituent spans should not cross. This constraint is less susceptible to tree annotation style, though [6] note that

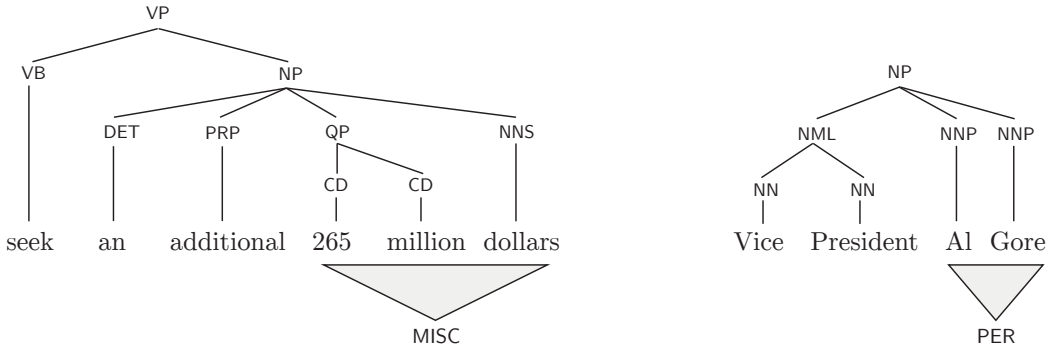


FIGURE 4.2: Unaligned entities. Flat NP structure prevents the entity (*265 million dollars*) from directly aligning to any syntactic constituent (*left*). Other annotation inconsistencies also contribute to the mismatch (*right*). Here the entity *Al Gore* does not correspond to a syntactic constituent, but is otherwise considered an NP in many other locations in the data, even when positioned similarly within a larger NP structure.

there are some inconsistencies based on whether trailing punctuation is treated as inside or outside the entity span. We now turn to a discussion of joint models, outlining the approach used by [6] and presenting our own joint model both of which aim to capture the relationships presented above in an effort to improve NER and parsing performance. [6] exploits dependencies between labeled syntactic constituents and NER labels. Our approach differs in that it models dependencies between *unlabeled* syntactic and NER spans, chosen based on the statistics above.

4.2 Joint Modeling via Grammar Augmentation

In this section we turn to discussing previous work which focuses on the joint modeling of NER and parsing. Despite previous work showing the usefulness of parse features in predicting NER [121], we are aware of only one other line of research which aims to perform both tasks jointly. This work, described primarily in [6] [1], and here on referred to as F&M09, serves as the primary comparison to our work.

The model presented in F&M09 couples NER and parsing using a method we refer to as *grammar augmentation*. A phrase structure parser groups words into constituent spans and associates each span with a constituent label. The intuition behind the F&M09 approach is that this label need not only indicate the span's syntactic category. A constituent label can also describe, in addition to syntactic category, the type of named entity of the span if applicable. This can be accomplished by augmenting the parser's grammar to include specialized nonterminal symbols (Fig. 4.3). The rule NP $\rightarrow$ NP PP might be augmented with entity information to produce the rule NP $\rightarrow$ NP-MISC PP to generate an entity of a miscellaneous type. Instead of PP $\rightarrow$ PRP NN the rule might be PP $\rightarrow$ PRP NN-ORG. This follows the pioneering work of [122], which makes use of a similar technique to phrase relation extraction as a parsing task.

However, this approach has unfortunate computational complexity. Starting with a grammar G of size $|G|$, expanding all grammar rules to accommodate the entity label set L produces a grammar

[1] [28] present a hierarchical joint model which incorporates the F&M09 work, but trains the model using additional data annotated solely for either task.

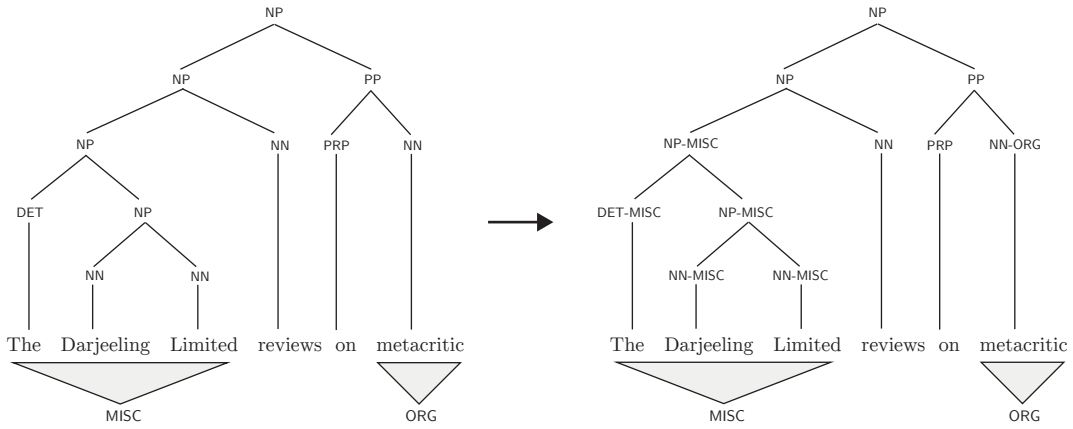


FIGURE 4.3: Grammar transformations for joint parsing and NER. Such transformations are used in [6].

of size $|G| * |L|$. This is a multiplicative growth in the model's label set size.[2] Having embedded NER labels within the grammar, both tasks can be solved simultaneously using standard statistical parsing techniques. Here the issue of label size comes to bear. The complexity of chart-based parser decoding is $O(|G|n^3)$. However, in the context of parsing, n, being the sentence length, is often quite small, whereas the grammar can often contain thousands of rules. In the case of grammars induced via nonterminal refinement techniques like split-merge [88], this can be closer to a hundred thousand. Thus the efficiency of decoding is particularly susceptible to the size of the grammar, and multiplicative increases can significantly slow the algorithm. Pursuing joint inference in this manner can be computationally burdensome, and F&M09 are forced to reduce the data's more refined label set of 18 entity types to a more smaller 4-label set.

The grammar augmentation approach is also limited in what types of problems it can represent. This approach is only a viable option for joint inference when one problem is capable of nesting within the phrase structure tree – the structures of the end task corresponding to constituent spans in the parse tree. For named entity recognition, this assumption is an appropriate one, but consider a problem that might also benefit from joint inference, but where the problem structure does not overlap. For instance, evidence suggests that the interaction between syntax and prosodic information (intonational phrases) is one such problem. In natural speech the pitch of one's voice rises and falls, providing many important intonational cues which have been shown to aid in disambiguating syntactic ambiguities [123, 124], and in parsing [125, 126]. Syntactic structure may also help identify prosodic boundaries [127].

The interdependent nature of syntax and prosody indicates that the two tasks may be a good candidate for joint inference, but the relationship between the two is fuzzy. [128] argues for the existence of linguistic constraints which favor left or right alignment of intonational phrases to constituent phrases, but importantly the boundary of one task *is* permitted to cross a boundary of the other. It is possible in a factor graph model to capture dependencies between an end task and phrase structure syntax that agree in just one edge, and whose other edge crosses syntactic boundaries. The grammar augmentation approach cannot represent the possibility of this type

[2] In practice and for the case of joint named entity recognition and phrase structure parsing, this need not be the case. Some nonterminals, like S, are unlikely to ever exactly dominate a named entity, making nonterminals like S-PER or S-ORG unnecessary.

of structure as the prediction of one task is rigidly constrained on both sides by the syntactic structure.[3]

We now present a joint factor graph model of NER and phrase structure parsing. Not only does this model rely on a combinatorial factor to prohibit invalid trees in the syntactic component, but it also features a new combinatorial factor to prohibit overlapping NER spans. The two models are connected via soft Boolean logic factors between corresponding NER and syntactic spans. In comparison to the grammar augmentation approach for joint inference, label set size suffers only additive growth when the two tasks are combined, and the soft nature of the constraints which connect the two models alleviates the need to filter out mismatched training data (as done in [6]).

4.3 Description of Joint Model

NER is well-suited to sequence models, and many state-of-the-art approaches combine sequential dependencies with the rich feature sets associated with discriminative training [39, 118]. In this section we present two NER models. The first is a standalone NER model, utilizing a combinatorial factor to implement a semi-CRF constraint which ensures that named entities do not cross one another. This is unlike many previous NER models in that it is *span-factored*: variables represent spans and not boundaries. This proves to be a convenient property when extending the standalone model to produce the second model: a joint model of NER and phrase structure syntax. Due to the span-based representation of NER, each NER span variable can be directly coupled to a corresponding syntactic span variable.

4.3.1 Modeling Named Entity Prediction

We present a simple span-factored model for handling NER span and label predictions. This model serves as a baseline and a basis of comparison for assessing the usefulness of joint inference. The basic model structure consists of three components: span variables, label variables, and factors which coordinate between them.

The presence of an NER span is represented by a Boolean variable, one for each $O(n^2)$ pairs of indices. This is almost identical to the manner in which we model constituent spans in the factor graph phrase structure parser (Sec. 3.1, pg. 62), differing only in that entities with more than ten words are exceedingly unlikely and so we impose a maximum value on the width of a span, $\mu = 10$. The method for labeling these spans also differs slightly, as constituent span labels were represented with Boolean variables (one per label) in order to make it more convenient to instantiate sparse grammatical rule factor connections. Here this is no longer necessary, and the set of NER labels for a particular span is represented as a single multinomial variable. The logic coordinating the multinomial label variable to the Boolean span variable is handled by a special-purpose "*e pluribus unum*" factor, introduced in [48], and described below.We formally describe these components:

- Let $\{NER\text{-}Span(i,j) : 0 \leq i < j \leq n, \text{ and } j - i \leq \mu\}$ be $O(n^2)$ Boolean variables such that $NER\text{-}Span(i,j) = \mathsf{true}$ if and only if there is a named entity in the sentence spanning

[3] One exception is combinatorial categorical grammar (CCG), where a joint model could potentially avoid this issue because CCG can explicitly represent structures that are not complete constituents [129].

from i to j (encompassing words i to $j-1$). The maximum span width, μ, is set to 10 in all experiments.

- Let $\{NER\text{-}Label(i,j) : 0 \leq i < j \leq n \text{ and } j - i \leq \mu\}$ be $O(n^2)$ multinomial variables ranging over values $\lambda \in L$, where L is the NER label set. The set L includes two additional values. One of these is a false value. $NER\text{-}Label(i,j) =$ false when $NER\text{-}Span(i,j)$ is false, and $j - i > 1$. The second of these is a null entity label O. $NER\text{-}Span(i,j) =$ O when $j - i = 1$. This ensures that all tokens which are not part of an entity are treated identically, i.e., there is presumably no regularities in non-entity tokens that would justify predicting non-entity spans of varying lengths. For all other labels $NER\text{-}Label(i,j) = \lambda$ if and only if the span from i to j corresponds to a named entity of type λ.

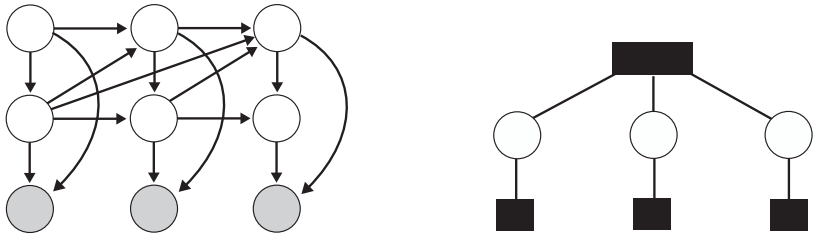
- Let EPU$(i,j) : 0 \leq i < j \leq n$ and $j - i \leq \mu\}$ be $O(n^2)$ "*e pluribus unum*" factors coordinating between $NER\text{-}Span(i,j)$ and $NER\text{-}Label(i,j)$ variables. This factor guarantees that $NER\text{-}Label(i,j) =$ false if and only if $NER\text{-}Span(i,j)$ is false.

Each $NER\text{-}Span$ and $NER\text{-}Label$ variable has a corresponding unary factor, associated with a set of features (described in Table 4.2, pg. 115), which locally provide each variable's local beliefs. Thus every variable and unary factor pair is equivalent to a log-linear (maximum entropy) classifier. However, unlike many previous approaches to NER that utilize similar classifiers [39], beliefs are shared between pairs of corresponding span and label variables during inference. One set of features is used to calculate a potential for an $NER\text{-}Span(i,j)$ variable, and a second set of features, disjoint from the first, is used to calculate a potential for each of the labels in the corresponding $NER\text{-}Label(i,j)$ variable. In our approach it is possible for the propagation of beliefs to sway the model's predictions in ways that wouldn't be possible in a pipelined approach to NER prediction.

For instance, a $NER\text{-}Span$ variable may have a strong belief that its value is false if the tokens in the span, and the features derived from them, are not generally indicative of a named entity. However, there may be features indicative of a *particular type of entity*. This would lead to a strong belief that a $NER\text{-}Label$ variable is true. In the pipelined approach this span would be classified as a non-entity span, and no further predictions would be necessary. In the factor graph approach, global inference makes it possible for the beliefs of $NER\text{-}Label$ variables to influence $NER\text{-}Span$ beliefs, allowing for correct prediction in such a scenario.

A Combinatorial Factor for Semi-CRFs

A recurring problem when representing structured outputs in factor graph models is the need to incorporate global information to prohibit invalid structures. In the case of NER the model needs to ensure that entity spans do not overlap. Hidden semi-Markov models [130, 131] and semi-Markov CRFs [57] offer this guarantee, but are not span-factored. Instead these models use a Boolean variable chain to represent segmentation points, implicitly defining spans, in conjunction with a second variable chain for labeling. A parameter γ specifies a maximum segment width. Variables from the second chain depend on the γ-1 previous segmentation variables, allowing these variables to predict labels in light of the model's predicted segmentation. Fig. 4.4 (*left*) illustrates a model with this type of structure.



a.) Hidden Semi-Markov Model b.) Global Semi-CRF Factor

FIGURE 4.4: Semi-Markov models. The Semi-Markov HMM (*left*) has three rows of variables: observation variables (*bottom row*), segmentation variables (*middle row*), and label variables (*top row*). Label variables depend on a number of previous segmentation variables (up to a max segmentation length, γ), allowing the model to jointly assess likely segmentations and labels, while inherently preventing spans from overlapping. In contrast, the combinatorial factor technique (*right*) implements similar logic within a globally connected factor. Variables in this model are span-factored, and the combinatorial factor prevents variable configurations in which spans overlap.

For our goals of coupling NER and syntactic span variables, we propose an alternate strategy for implementing this constraint which replaces structural dependencies with a globally-connected combinatorial factor. This allows us to efficiently prevent invalid structures in a span-factored NER model. NER span variables can then be connected to the span variables of a phrase structure grammar model in a convenient 1-to-1 correspondence (for all spans with widths $< \gamma$. We introduce this semi-CRF constraint factor formally as:

- Let SEMI-CRF be a combinatorial constraint connected to all $NER\text{-}Span(i,j)$ variables. The factor implements a weighted 0th order Semi-CRF with a maximum span width γ, as described in [57], over the log-odds of each $NER\text{-}Span$ variable's Boolean values. Unlike the valid-bracketing constraint imposed by the CKY-TREE factor, this effect can be achieved with a polynomial number of binary factors, but the $O(\gamma^2 n^2)$ such pairwise factors would result in an extremely loopy graph and inefficient inference.

$$\Psi_{\text{SEMI-CRF}}(NER - Span(i,j)) = \begin{cases} 0, & \text{if } \exists i,j,k,l : NER\text{-}Span(i,j) = \mathsf{true} \wedge NER\text{-} \\ & Span(k,l) = \mathsf{true} \wedge i < k < j < l \\ 1, & \text{otherwise} \end{cases}$$

Pseudocode for computing the out-going messages from this factor is provided in Fig. 4.5. The goal of preventing overlapping NER spans is similar to the goal of prohibiting invalid trees, and naturally the SEMI-CRF factor logic is similar to that of the CKY-TREE factor. In both algorithms the calculation of out-going messages is decomposed into two steps: in the first step the computation sweeps forward through the chain computing forward scores (analogous to the inside scores used in the CKY-TREE factor), and a second step that sweeps backward through the chain (analogous to the outside scores used in the CKY-TREE). The forward and backward scores can then be multiplied to produce the gradient, $\mu(i,j)$, which is used.

Algorithm 8 Propagator for Semi-CRF Factor

procedure PROPAGATE SEMI-CRF(n, γ) ▷ Sentence Length, Max Segment Size
  for $v \leftarrow \{v_1, ..., v_m\}$ **do**
    $\overline{q}_v \leftarrow v(\mathsf{true})/v(\mathsf{false})$
  end for
  $\alpha(0) \leftarrow 0$ ▷ Forward Computation
  for $k \leftarrow 1$ to n **do**
    for $w \leftarrow 1$ to $\min(\gamma, k)$ **do**
      $i = k - w$
      $\alpha(k) \overset{\oplus}{\leftarrow} \alpha(i) + score(i)(w)$
    end for
  end for
  $\beta(n) = -\alpha(n)$ ▷ Backward Computation
  for $i \leftarrow n - 1$ to 0 **do**
    for $w \leftarrow \min(n - i, \gamma)$ to 1 **do**
      $k \leftarrow i + w$
      $\beta(i) \overset{\oplus}{\leftarrow} \beta(k) + score(i)(w)$
      $\mu(i)(w) \overset{\oplus}{\leftarrow} \beta(k) + \alpha(i)$ ▷ Gradient Calculation
    end for
  end for
end procedure

FIGURE 4.5: Pseudocode for the SEMI-CRF factor propagator. Let $score(i)(w)$ be the odds ratio of incoming messages from variable $NER\text{-}Span(i, i + w)$. The $\overset{\oplus}{\leftarrow}$ indicates incremental assignment.

The forward computation answers the question, "*For any state k representing whether or not an entity span begins at position i, $1 \leq i \leq n$, what is the probability of beginning at the start of the sentence and generating up to state k?*" We define this quantity by the forward score, $\alpha(k)$. This can be computed efficiently using dynamic programming, beginning at the start of the chain ($k = i$), caching each $\alpha(k)$, and proceeding through to the end of the chain. As illustrated in the semi-Markov HMM in Fig. 4.4, the computation of each $\alpha(k)$ involves looking back at only the forward scores that fall within the maximum span width's distance away.

The goal is reversed for the backward computation. Each backward term, $\beta(k)$, is the score of all possible analyses beginning at state k and continuing through to the end of the chain. To compute this efficiently, a procedure analogous to the forward computation begins at the end of the chain, moving backward to the beginning of the sentence, incorporating the sum of all span analyses at each step. The gradient for a span, $\mu(i)(k)$ with width w, can be computed by combining the forward score for the start of the span, $\alpha(i)$, and the backward score for the end of the span, $\beta(k)$. The gradient can then be used to calculate out-going messages in a manner identical to their use in the CKY-TREE computation (pg. 72). Message computation for this factor has a time complexity of $O(\gamma^2)$.

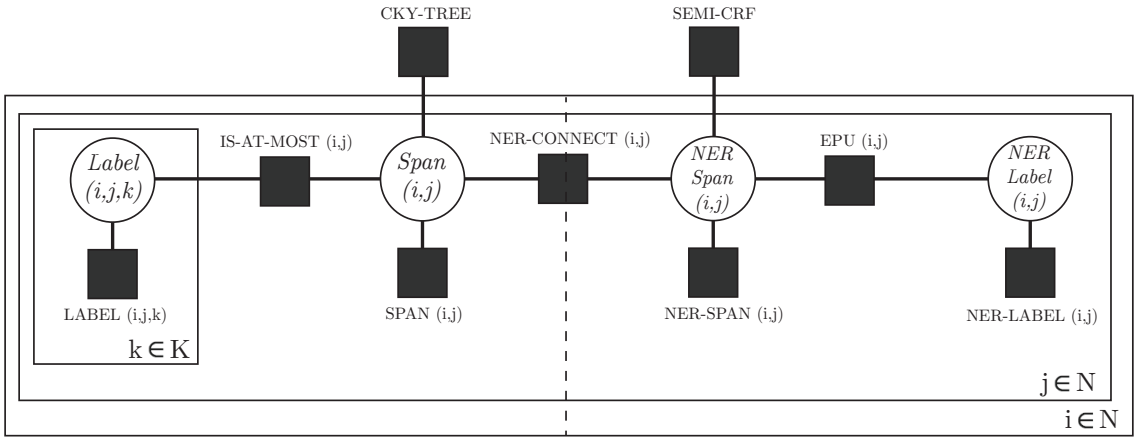


FIGURE 4.6: A joint model of NER and phrase structure syntax. Depicted in a plate diagram, the baseline NER model (*right*) is a span-factored model for predicting NER boundaries (using $NER\text{-}Span$ variables) and NER labels (using $NER\text{-}Label$ variables). The SEMI-CRF factor constrains $NER\text{-}Span$ variables to prohibit overlapping spans. The model of phrase structure syntax (*left*) is as described in Section 3.1.2 (pg. 69). The two models coordinate across pairs of unlabeled span variables, using NER-CONNECT factors to implement a soft Boolean NAND logic. The number in the bottom corner of each plate indicates the number of times the contained structure repeats.

Together with the modeling components presented in Section 4.3.1 (pg. 109), this constitutes the NER-ONLY model.

4.3.2 A Joint Model of NER and Constituent Syntax

The factor graph parsing model we present in Sec. 3.1 (pg. 62) is an improvement over previous CRF-based parsers, but it is not competitive with state-of-the-art re-ranking or latent variable grammar models. The advantage of parsing with this factor graph model is not absolute F1 performance, but rather the ease with which it can be combined with other models to create a joint model, with the guarantee that there is an approximate inference strategy (belief propagation) for whatever the resulting graph structure may be. Here we describe the construction of a joint model of phrase structure parsing and NER. This extension requires only an additional set of pairwise logical factors to coordinate the variables from the opposing component models.

First, in addition to the named entity model described above, we assume the presence of the phrase structure variables and factors described in Sec. 3.1.2 (pg. 69). This includes $Span(i, j)$ and $Label(i, j, k)$ variables to represent the space of all possible labeled parses, the corresponding unary factors to locally score these variables, and a CKY-TREE factor to globally constrain $Span(i, j)$ variables to form a valid tree.

Previous work has argued that every named entity should correspond to a noun phrase [6]. Following this logic we might implement a joint model which connects the $NER\text{-}Span(i, j)$ variables to $Label(i, j, k = \text{NP})$ variables. However, an analysis of the data shows that this correlation only holds for slightly more than half of all entities (Table 4.1.1), motivating us to choose a more general coupling between models in which solely the structural components of each model, $Span$ and $NER\text{-}Span$ variables, are coupled using soft Boolean logic factors.

Here the motivation for using a combinatorial factor for modeling the semi-CRF constraint becomes clear: although a traditional semi-Markov model can be expressed strictly through many pairwise factors and would also rule out over-lapping entity spans, spans are not explicitly represented in the model. By using a combinatorial factor to express the semi-Markov logic we adopt a span-based representation for the NER model, which allows a convenient mapping between the NER and syntactic models. Thus for every Boolean $NER\text{-}Span(i,j)$ variable there is a corresponding syntactic $Span(i,j)$ variable, and these variables can be coordinated using a Boolean logic factor.

The factors which coordinate between models impose a soft NAND logic. If either input is false, a logical NAND outputs true. Similarly, if either coordinated variable is false, the NAND factor contributes a potential of 1.0 and does not affect outgoing messages. When both variables are true the NAND factor applies in a potential derived from a set of associated features and feature weights. Optimization adjusts these feature weights, learning the degree of coordination between variables.

Formally we introduce this factor as follows:

- Let $\{\text{NER-CONNECT}(i,j) : 0 \leq i < j \leq n; 1 < j - i \leq \gamma\}$ be a set of at most $O(n^2)$ soft NAND factors coordinating syntactic $Span(i,j)$ and named entity $NER\text{-}Span(i,j)$ variables. A NAND factor (multiplicatively) contributes a potential of 1 to the model score unless both variables are true, in which case it multiplies a connective potential $\phi(i,j)$ derived from its features.

Intuitively the joint model might learn features weights such that $\phi(i,j) > 1$, i.e., constituents and NER spans are more likely to be coterminous. The number of these coordinating factors is constrained to the number of NER span variables, subject to the maximum span-width $\gamma = 10$.

The soft aspect of this factor is critical to the performance of the joint model. Only a small number of constituent spans will correspond to named entities, and even using binarized trees there will still be a significant number of named entities that, due to annotation style, do not correspond to a constituent span. To capture noisy correlations of this nature we introduce a set of features, described in Sec. 4.3.3, which are used to learn a context-specific weight for each NER-CONNECT factor's potential.

To illustrate how this might function, consider the feature comprising the concatenation of the sequence of part-of-speech tags for each word in the span. For shorter spans, many instantiations of this feature will occur many times in the data and this feature will often be a good indicator of 1.) what type of constituent is at that span, and 2.) whether or not the span is a named entity. In cases where there is a syntactic span and it is NP-like, this feature may strengthen the connection, assigning a high potential to the model score. In cases where there is a syntactic span and it is NP-like, this feature may elicit the opposite effect, assigning a low score to the configuration where both the syntactic span and named entity are present. Thus, despite having no connections between $Label(i,j,k)$ variables and $NER\text{-}Span(i,j)$ variables, features in the NER-CONNECT (i.e., NAND) factor can approximate the information that would be gained from connections with a constituent label predictions.

We now discuss the feature sets used in these soft constraints, as well as the rest of the features used in the NER model.

Span Features	
Feature String	Comment
BIAS	A bias factor to capture how likely any span is to be an entity.
The-Darjeeling-Limited	A concatenation of all words in the span.
DET-NN-NN	A concatenation of all tags in the span.
C-C-C	A concatenation of capitalization (C for yes, c for no).
The-DET-Darjeeling-NN-Limited-NN	A concatenation of words and tags.
w-3	Span width.
DET-NN	Part-of-speech tags of boundary words (inside the span).

Unigram Features		
Word	Feature String	Comment
Darjeeling	Darjeeling	The word.
Darjeeling	NN	The part-of-speech tag.
Darjeeling	is-capitalized	Capitalization feature.
Darjeeling	darjeeling	word to lowercase.
Darjeeling	Darjeeling-NN	Word and tag.
Darjeeling	s-ngram-1-3-Dar	Character grams. Substrings of 2-5 letters from start of word.
Darjeeling	e-ngram-1-3-ing	Character grams. Substrings of 2-5 letters from end of word.
Dec.	MONTH	Month-matching regex.
12:59PM	TIME	Time-matching regex.
third	ORDINAL	Ordinal-matching regex.
fourteenth	FRACTION	Fraction-matching regex.
Wed	DAY	Day-matching regex.
E.T.	INITIAL	Initial-matching regex.
ET	CAPS	All-capitalized-matching regex.
et	LOWER	All-lowercase-matching regex.
IV	ROMAN	Roman-numeral-matching regex.
!?	PUNC	Punctuation-matching regex.

TABLE 4.2: Features for NER factors. Span-based features (*top*) provide a barebones set of features for entity prediction. Unigram features (*bottom*) are included for each of the boundary words i and j for a span over i, j, as well as a previous token and following token when applicable. These features are concatenated with the word's location in the span.

4.3.3 Features

Five distinct sets of factors in the joint model rely on features: SPAN(i, j) and LABEL(i, j, k) parsing factors, NER-SPAN(i, j) and NER-LABEL(i, j) NER factors, and NER-CONNECT(i, j) coordinating factors.

Features for NER factors

We use an identical set of features for both identifying and labeling NER spans. Due to the need to identify and classify strings of names, the NER features are more lexically oriented than features used for parsing, and include word shape, character n-grams (windows of 2 to 5), capitalization, normalization, and regular expressions for initials, numerics, times, dates, and Roman numerals. Other features are included to capture more span-level information, including the combined part-of-speech of all words in the span, combined capitalization of all words in the span, span width, and the part-of-speech tags at the edge of each span. When used in NER-LABEL(i,j) factors, the associated label of the span, λ, is concatenated onto each feature string before being converted

Feature String	Comment
Agree-BIAS	A bias feature governing the general degree of agreement between variables.
Agree-3	Span width.
Agree-The-Darjeeling-Limited	A word concatenation feature.
Agree-DET-NN-NN	A part-of-speech concatenation feature.
Agree-DET	Initial part-of-speech tag.
Agree-NN	Final part-of-speech tag.
Agree-DET-NN	Start and end part-of-speech tag.

TABLE 4.3: Features for coordination factors.

into an integer representation to produce a distinct set of features for each label. NER features are summarized in Table 4.2.

Features for parse factors

The features utilized by the parsing factors are identical to those used in the standalone parsing model (See Section 3.1.2, pg. 82).

Features for coordinating factors

The data statistics (shown in Table 4.1.1) reveal only an average degree of correlation between named entity and syntactic spans, and the feature set for coordinating factors (described in Table 4.3.3) is necessary for providing a context-specific coordination potential. These features are essentially identical to the span-based features used for named entity identification. Tag concatenation features were good predictors of constituent span and constituent type for short spans. Since a majority of named entity spans contain only a few words, it is possible that similar features will be well-represented in the data and suitable for learning soft weights for the correlation between entity and syntactic spans.

4.4 Experiments

In this section we evaluate a number of model configurations, demonstrating the usefulness of joint inference, and the usefulness of inducing a latent syntactic representation in situations where gold syntax does not correlate highly with the end task.

4.4.1 Data

Data which is jointly annotated for NER and parsing is exceedingly rare. For evaluation purposes we rely on a selection of six corpora drawn from the English broadcast news section of the OntoNotes 2.0 dataset[108], the only data set we are aware of which is jointly annotated for both NER and phrase structure syntax. Each corpus corresponds to a different news broadcast organization (ABC, CNN, MNB, NBC, PRI, VOA). We partition each corpus to achieve an approximate 3:1 ratio between training and test sets, reproducing the partitioning found in F&M09. Corpus statistics are presented in Table 4.4. As in that work, we remove empty leaf nodes and coarsen nonterminal labels in the parse trees (NP, not NP-PRD), and filter out sentences longer than 40

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

TABLE 4.4: F&M09 OntoNotes data statistics. Ranges indicate the file IDs of files used to construct each data set.

words. We map the fine-grained entity label set found in OntoNotes into the four-label entity set found in F&M09. PER, GPE, and ORG are kept as is, while all other named entity labels are collapsed into MISC.

4.4.2 Experimental Design

Each set of experiments compares four different model configurations:

- **NER-Only**, a baseline model consisting only of the NER component as described in Section 4.3.1 (pg. 109). This includes the SEMI-CRF combinatorial factor.
- **NER-Joint**, the joint model described in Sec. 4.3.2 (pg. 113) using an unlabeled phrase structure parser and factors which coordinate between syntax and NER spans.
- **NER-Oracle**, the same joint model, but with constituent span variables set to their observed values as specified in the hand-annotated gold parse trees included in the OntoNotes corpus. These are observed and used both during training and testing.
- **NER-Hidden**, the same joint model, but syntactic annotations are ignored and the model learns a distribution over a latent syntactic representation using the marginalization-based training method described in Section 2.5.5 (pg. 50).

We train each model using 100 iterations of stochastic gradient descent, using a learning rate $\eta = 0.1$. We use L2 regularization, corresponding to a Gaussian prior with a variance of 1.0. A maximum of 40 iterations of belief propagation are used for each example, but convergence often occurs much earlier, and is exact (two iterations) for the NER-ONLY model. These parameters are set based on preliminary experiments using the ABC portion of the corpus as a development set. We did not perform an exhaustive sweep, noting only that η values of an order of magnitude higher or lower, and as few as 20 iterations of SGD, were generally sufficient for performance comparable with the results presented in Fig. 4.5. Decoding is performed by taking the entity set which maximizes the score of the SEMI-CRF factor. If the SEMI-CRF computation is thought of as sum-product inference with a structured constraint, decoding is done simply through max-product inference with the structure constraint, an application of minimum Bayes-risk decoding. All NER results were scored using macro-averaged F1, as used in conlleval, and all parse results scored using Evalb.

We perform two sets of experiments, each using a different feature set. First, the linear-chain feature set (Chain Feats) does not contain any features which require both a start and an end index of the span, and instead contain sliding windows of word/tag/capitalization n-grams along either the start or end index. This more closely approximates the features of F&M09. Formally the linear-chain feature set is the $Unigram$ set described in Table 4.2 (pg. 115), at both the start and end index of the span, and as bigrams between pairs of indices in a 3-word index around either of the span indices. The second feature set (Span Feats) is composed of span-based features, which includes span width, bigrams of information from the start and end of the span, and word/tag/capitalization concatenations from all of the tokens within the span, and both the tokens immediately outside the span. It is exactly the feature set described in Table 4.2.

The size of each model and its computational requirements for the smallest (MNB) and largest (CNN) data sets is given below. Training times listed are the amount of time necessary to perform one iteration of SGD.

	Chain Feats				Span Feats			
	MNB		CNN		MNB		CNN	
	#Feats	Time	#Feats	Time	#Feats	Time	#Feats	Time
NER-Only	30k	40s	150k	4m	35k	50s	180k	1m
NER-Joint†	115k	2m	770k	8m	123k	10m	800k	8m
NER-Oracle††	75k	1.5m	540k	6m	110k	2m	680k	7m
NER-Hidden	750k	7m	6.3M	19m	890k	7m	9.5M	23m

Models which require syntactic annotations during training are marked with a † symbol. Models which rely on syntactic annotations for both training and decoding are marked with †† symbol.

4.4.3 Results

The Case for Joint Inference

In this section we discuss the performance results of the baseline and joint models using the linear-chain feature set. We also revisit the results of previous work from [113], which are more directly comparable to F&M09.

The results of the NER accuracy experiments are given in Table 4.5. Included in these results is a set of experiments using the linear chain features (*left*), which were performed in an effort to more closely reproduce the baseline system of F&M09. In comparison to this system, our own baseline, the NER-ONLY system, performs much lower overall, with an averaged label F1 of 70.67% vs. F&M09's 73.17%. However, this lower performance baseline does help highlight the differences between the various chain feature models.

For all data sets, joint inference results in significant increases in labeled and unlabeled F1. The standard joint model, NER-JOINT, provides an average improvement of 2.71% labeled F1 (73.38% average F1). Perhaps unsurprisingly, the use of the observed trees leads to still higher performance in the ORACLE model, a gain of 3.33% F1 over NER-ONLY and 0.64% F1 over NER-JOINT. This is not always the case: predicting trees, rather than observing them, yields higher performance for the NER-JOINT model for the NBC and PRI data sets, but these margins are small.

		Chain Feats						Span Feats						
		Unlabeled			Labeled			Unlabeled			Labeled			F&M
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	'09
ABC	NER Only	85.7	78.4	81.9	71.8	65.6	68.6	89.3	85.9	87.6	78.9	75.9	77.4	74.5
	NER Joint†	87.2	79.7	83.3	76.7	70.1	73.2	89.3	85.9	87.6	79.6	76.6	77.4	78.7
	NER Oracle††	88.0	80.4	**84.0**	77.1	70.4	**73.6**	88.7	86.1	87.3	79.4	77.0	78.2	–
	NER Hidden	86.1	78.7	82.2	76.3	69.8	72.9	89.4	86.9	**88.2**	79.5	77.3	**78.4**	–
CNN	NER Only	83.6	80.9	82.2	72.6	70.2	71.4	87.9	87.6	87.8	78.4	77.7	78.0	75.8
	NER Joint†	86.7	84.7	85.7	74.9	73.1	74.0	89.1	88.6	88.9	78.4	77.9	78.1	78.7
	NER Oracle††	87.6	85.6	86.6	75.7	73.9	74.8	88.6	88.3	88.4	78.9	78.6	**78.7**	–
	NER Hidden	87.7	86.0	**86.9**	76.2	74.8	**75.5**	89.6	89.2	**89.4**	78.4	78.1	78.3	–
MNB	NER Only	79.7	64.6	71.4	70.9	57.4	63.5	82.5	77.4	79.9	69.4	65.1	67.2	62.2
	NER Joint†	80.6	66.2	72.7	71.2	58.5	64.2	83.5	77.9	80.6	69.8	65.1	67.4	66.5
	NER Oracle††	79.6	66.2	**72.3**	72.2	60.0	**65.5**	84.6	79.0	**81.7**	71.4	66.7	**69.0**	–
	NER Hidden	77.4	65.1	70.8	69.5	58.5	63.5	80.4	75.9	78.1	68.0	64.1	66.0	–
NBC	NER Only	76.2	70.6	73.3	64.9	60.1	62.8	79.2	75.2	77.2	70.0	66.4	68.2	63.9
	NER Joint†	77.2	72.9	75.0	67.0	63.3	65.1	79.4	76.2	77.8	70.8	67.9	69.3	68.0
	NER Oracle††	76.1	73.0	74.5	66.0	63.3	64.6	79.2	75.2	77.2	70.5	67.0	68.7	–
	NER Hidden	78.0	74.8	**76.3**	67.0	64.2	**65.6**	80.0	77.1	**78.5**	71.4	68.8	**70.1**	–
PRI	NER Only	84.5	80.6	82.5	78.1	74.5	76.2	91.7	89.7	90.7	86.4	84.6	**86.0**	83.4
	NER Joint†	87.2	83.9	85.5	81.7	78.6	80.1	91.2	89.1	90.1	85.7	83.7	84.7	86.3
	NER Oracle††	87.2	83.6	85.4	81.7	78.3	79.9	91.9	89.9	90.9	86.4	84.6	85.5	–
	NER Hidden	90.1	86.2	**88.1**	84.4	80.8	**82.5**	91.6	90.4	**91.0**	86.4	85.2	85.8	–
VOA	NER Only	89.6	85.1	87.3	83.7	79.5	81.5	92.5	90.1	91.3	87.5	85.2	86.3	79.2
	NER Joint†	91.7	87.9	89.8	85.5	82.0	83.7	92.7	90.8	91.7	87.7	85.9	86.8	88.2
	NER Oracle††	93.0	90.1	**91.5**	87.0	84.3	**85.6**	94.8	92.7	**93.4**	89.2	87.9	**88.5**	–
	NER Hidden	91.3	89.0	90.1	85.8	83.6	84.7	92.7	91.4	92.1	87.9	86.7	87.3	–

TABLE 4.5: NER baseline and joint model performance on the OntoNotes corpus. Joint inference consistently improves NER results across all corpora, providing the largest gains on corpora with the most data and best parser performance. When using the linear-chain feature set the overall performance is lower, but gains between the baseline and joint models is larger. When using span-based features most systems perform similarly.

While the average labeled F1 scores would rank the overall system performance as NER-HIDDEN, NER-ORACLE, NER-JOINT, NER-ONLY, best to worst, the relative performance of these systems varies largely from data set to data set. For the NBC data set, NER-JOINT outperforms NER-ORACLE. On the MNB data set, NER-ORACLE outperforms all other systems by at least a 1%F1.

What causes these discrepancies? It does not appear to be a factor of data size, as the MNB and NBC data sets are similarly small and produce two very different rankings. However, the ranking of the NER-ORACLE model appears to be correlated with the degree to which entity spans align to syntactic constituents, as presented in Table 4.1.1. The two datasets with the lowest correlations – NBC and PRI – are the same data sets where NER-HIDDEN has the highest relative performance over NER-ORACLE. If the gold standard syntax constituents are highly correlated with named entities, the NER-ORACLE model is able to take the best advantage of it. When the degree of correlation between NER and syntax is not as strong, the NER-HIDDEN model can induce a better syntactic representation. This also follows for the NER-JOINT model: when gold syntax is not strongly correlated with named entities, it can be better to predict trees which are similar, but not exactly the same as the gold trees.

Comparing Joint Inference Techniques

We more closely replicated the baseline results of F&M09 in previous work [113]. This was largely due to the use of dictionaries of proper names, cities, etc., when constructing features, as done in F&M09, which significantly improves the model's ability to correctly label entity spans. With these experiments we are able to better compare the two approaches to joint inference: soft Boolean logic factors in factor graphs, and grammar augmentation with CFGs. In this work our baseline model achieved an average labeled F1 of 73.67$\%$ across all corpora, while the F&M09 baseline has an average labeled F1 of 73.18$\%$. This is a difference of 0.49 F1$\%$, making the two baseline systems quite comparable.

While both systems benefit greatly from joint inference with a model of phrase structure syntax, the grammar augmentation approach of F&M09 results in an average F1 of 76.38% across all corpora, a 3.2 F1$\%$ increase over the F&M09 baseline. In comparison, our joint model not only provides a higher average absolute F1, at 78.27$\%$ F1, but also a higher margin, improving NER accuracy by 4.6% F1 over our own higher accuracy baseline model. In comparison, our joint model not only provides higher absolute performance, a 78.27% average labeled F1, but also a higher margin, improving NER accuracy by 4.6% average labeled F1 over our own higher accuracy baseline model.

These results alone cannot reveal the exact cause of the performance difference. One possible hypothesis is that there are inherent advantages to our approach to joint inference. Our soft Boolean logic constraints allow for a more flexible, context-specific coupling. However, it is difficult to assess how many F&M09 errors might be attributed to the limitations of the "hardness" of its coordinating constraints, or the extent to which our approach overcomes them. In an additional set of experiments we reduce the feature set for coordination factors to a single bias feature, thus forcing the model to learn a single weight to uniformly govern all coordination between syntax and NER models. This results in a 3.1 decrease in average F1 when compared to our full joint model, showing the importance of the more flexible coordinating factors.

A second, and perhaps more likely explanation of why our joint model outperforms F&M09 is

the higher parse accuracy of our parser. The factor graph parser (Sec. 3.1.2, pg. 69) has an average parse F1 of 78.47, significantly higher than F&M09's average F1 of 71.13. This constitutes a 7.34 difference in F1. Perhaps more pertinent to NER accuracy is the crossing bracket score, which measures the average number of constituents in the predicted tree which cross a constituent in the gold tree. An entity whose boundaries cross a predicted constituent span is unlikely to be found in the model solution, and is ruled out entirely in the case of the grammar augmentation approach to joint inference. Our parser has an average of 1.71 crossing brackets per sentence, whereas the baseline parser used in F&M09's has an average of 2.02, representing a15.3% reduction in crossing bracket errors over F&M09.

A motivating example for Joint Inference

How does coupling these tasks benefit NER prediction? Fig. 4.7 provides some insight into the types of errors which joint inference can avoid. In this example the baseline NER-Only system incorrectly predicts the entity *Social Party* of type ORG. This is not serious error: *Social Party* would likely be an entity span in another context, and the entity label is correct. The model only fails because it does not capture the encompassing entity, *Social Party of Serbia*. The tree produced by the standalone parser makes an similar mistake, correctly predicting *Social Party* as an NP but failing to predict the second encompassing NP.

By sharing beliefs between the syntax and NER models, the joint model finds a better global solution. The parse tree correctly groups *Social Party of Serbia* into its own NP, attaching the following VP to the root of the tree. The named entity model receives stronger evidence for the entity span *Social Party of Serbia*, and is able to correctly predict the full entity span. Both predictions are aided by a potential from the coordinating factor that is > 1, and thus increases the model's belief that they are both true. This example is also interesting because the two erroneous analyses of the standalone models are in agreement. This is the type of problem where a context-specific weight, provided by the features for the NER-CONNECT factor, may be a critical factor in reaching the correct solution.

The Effect of Joint Inference on Parsing Performance

The traditional motivation for joint inference argues that both tasks stand to benefit from being performed jointly. However, while the previous example illustrates an instance where joint inference being mutually beneficial to both tasks, we find these occurrences to be rare when using our joint model on this data set. While joint inference significantly improves NER prediction across all corpora, the net effect of joint inference on parse accuracy is negative: joint inference *decreases* the average parse accuracy across all corpora by 0.35 F1. Across the six corpora, joint inference is just as likely to hurt parser performance as it is to help. This effect is most pronounced in the ABC corpus, one of the smaller corpora, where it decreases performance by 0.98 on the 199 sentence test set. The maximum improvement is 0.28 F1 on the NBC data set, another of the smaller corpora. Table 4.6 lists these results.

Our findings differ from those of F&M09, where joint inference provides significant gains on five of the six corpora, with an average improvement of 0.73 F1. Both approaches to joint modeling improve parsing the most on the NBC corpus, where the approach of F&M09 increases parse accuracy by 1.36 F1 (in comparison to our 0.28 F1 increase). However, our baseline parser

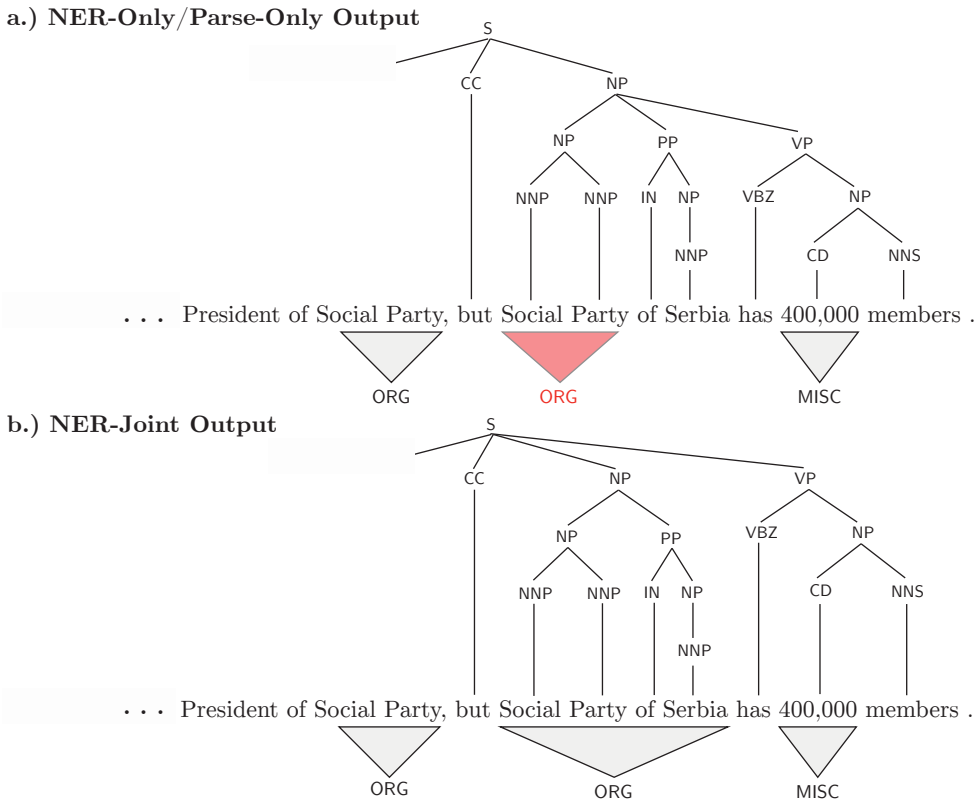


FIGURE 4.7: An example of joint inference improving both syntax and NER analyses. The standalone NER model (a) is attracted to the shorter NP, *Social Party*, predicting the correct entity label but failing to recognize that it is part of a larger entity. The parse-only model makes a similar mistake, correctly classifying *Social Party* as an NP, but failing to correctly recognize its role in an encompassing NP. The joint model (b) correctly predicts this larger NP, and the aligning named entity.

outperforms the F&M09 joint model on all parsing tasks, and it is possible that we are observing a ceiling effect, with decreases in performance caused by incorrect generalizations and what is essentially overfitting. In spite of efforts to regularize the model to prevent this, these data sets are quite small, and more complex joint models may be prone towards finding generalizations in the training data that are not as present in the test set.

Advancing the State-of-the-Art

In the previous section we showed that joint inference can produce significant improvements in performance, raising labeled F1 performance by 2.71% with linear-chain features, 4.6% with the feature set of [113], and 3.2% in F&M09. However, in practice it is hard to outperform a discriminatively-trained sequence model with good feature selection. In this section we examine the difference features can make on model performance, attempting to achieve the highest labeled F1 scores.

Purely sequential models, like the linear-chain CRF, make one prediction, an entity label (which could be null), for each token. The identities of entities can then be determined from these purely local labels, but without the notion of a span, the model's features are limited. Our

		Prec	Recall	F1	F&M '09
ABC	Parse Only	79.75	76.72	**78.20**	**70.15**
	Parse Joint	78.15	78.15	77.17	69.99
CNN	Parse Only	83.30	80.14	81.69	77.03
	Parse Joint	83.41	80.09	**81.72**	**77.71**
MNB	Parse Only	76.98	72.61	**74.73**	65.49
	Parse Joint	75.10	71.47	73.20	**65.59**
NBC	Parse Only	74.81	71.67	73.20	61.63
	Parse Joint	75.15	71.88	**73.48**	**62.93**
PRI	Parse Only	82.70	79.92	81.29	76.35
	Parse Joint	82.98	80.03	**81.48**	**77.41**
VOA	Parse Only	83.55	79.89	**81.68**	76.15
	Parse Joint	83.43	79.91	81.63	**77.51**

TABLE 4.6: Standalone and joint parsing performance on the OntoNotes corpus. Results are calculated in terms of labeled accuracy using Evalb. In comparison to F&M09 (*right*) our models offer better parsing performance, but joint inference provides less significant and less consistent improvements.

model is span-based, making a prediction for each possible span, and thus the features used in our model can be constructed around spans. The performance of these models is provided in Table 4.5 ($right$).

Here, by using span-based features, we are able to improve the performance of our baseline model to levels significantly beyond that of the joint F&M09 model, and mitigating most of the performance improvements of our own joint model. The average labeled F1 performance of the model is 85.7% for NER-ONLY, 86.1% for NER-JOINT, 86.2% for NER-HIDDEN, and 86.4% for NER-ORACLE. The margin of performance between the NER-ONLY and NER-JOINT models falls to just 0.7% average labeled F1.

There are several factors that contribute to this: the small data size, the predominantly short NER spans, and perhaps most importantly, the similarity between the information captured by the baseline and joint models. The small data size limits what can be learned, and pushes the accuracy of the span-featured models close to a performance ceiling: only a handful of mistakes are made by any model, and most occur when labeling the spans, not when identifying the spans, which is where joint inference may be most useful since we couple unlabeled span variables. The tendency for NER spans to be short, and their position at the leaves of the tree, do limit the extent to which a joint model can improve over a strong baseline sequence model. However, when using the span-based feature set, the information captured by the NER-ONLY and joint models is quite similar, the only significant difference being the structural constraints imposed on each component in the model. Even here, the constraint imposed on NER variables by the SEMI-CRF factor is very similar to the constraint imposed on syntactic variables by the CKY-TREE factor.

While this is largely a negative result, we still observe some interesting behavior. In terms of labeled accuracy, all joint models improve compared to the baseline, but the advantage of the NER-HIDDEN model is reduced. Inducing a syntax using the marginalization-based training method renders the optimization search space non-convex, introducing the possibility that optimization will not find the optimal set of model parameters. We find this to be a recurring problem: the

hidden syntax models perform better on the unlabeled tasks, which are most directly affected by the coupling of the models, but do not manage to label these structures quite as well as the model configurations with convex objectives. Here a hybrid approach which predicts structure using the NER-HIDDEN and labels it using the predictions of one of the remaining models. Another alternative would be tune the optimization parameters on development data, though we do not pursue that approach here.

4.5 Conclusions

In this chapter we discussed the task of named-entity recognition and showed how a model which performs this task jointly with phrase structure parsing can outperform the performance of the NER model alone. Our NER baseline model is essentially a common CRF-based model, though it utilizes a span-based representation and a novel semi-Markov combinatorial constraint to prohibit NER spans from crossing. This model is connected to a representation of phrase structure syntax using Boolean logic factors which learn a context-specific coordination potential: the strength of the coordination between NER and syntax differs based on the width of the span and the words and part-of-speech tags contained within it.

Our baseline model outperforms a comparable baseline from previous work, our joint model outperforms the joint model from previous work, and the margin of improvement associated with joint inference are greater in our framework [113]. Additionally, the joint models which induce a latent syntactic representation outperform the joint model trained on gold syntactic annotations on most corpora. However, these very positive findings are ultimately met with a negative result: a feature-rich baseline can all but eliminate the benefit of the joint inference on our test data set. We identify several causes contributing to this effect.

In the feature-rich setting, using the gold syntax provides the best performance, but in terms of unlabeled accuracy the joint model which induces a latent syntax is the next best configuration, implying that there is hope of outperforming the joint NER model without the need for data jointly annotated with syntactic trees. These are the best results ever reported for this data set, and it is possible that we are observing a ceiling effect, reducing the performance differences between models as we approach the maximum achievable performance with these kinds of methods.

The data set is also problematic. As noted in F&M09, there are many instances where a named entity should correspond to a syntactic constituent, but doesn't. This includes nested entities, which aren't explicitly annotated in the data. Given the right data, other joint models may yield better results. A simple and intuitive extension would be to connect NER labeling to syntactic structure or to syntactic labels. For instance, it is straightforward to add a dependency between the named entity span and the set of syntactic label variables which can correspond to entities. We leave this model as future work, as it would be better suited to a data set that more accurately reflected our intuitions about the relationship between named entities and constituents.

NER couples naturally to phrase structure syntax, as both tasks aim to identify contiguous spans of words, and NER spans frequently correspond to constituent spans. In the next chapter we explore the task of relation extraction, where it is less clear which syntactic representation best suits the problem. We present two joint models, one coupling to phrase structure, the other to dependency structure, and show that it is possible to induce a latent syntax which allows the model to outperform the use of gold trees, provided by pre-trained state-of-the-art parsers.

5

Joint Models for Relation Extraction

In the preceding chapter we demonstrated that our factor graph modeling framework can be used to produce a state-of-the-art baseline NER model, and accuracy can be improved by incorporating a model of phrase structure grammar and performing inference jointly. However, we also find that these improvements can be alternatively achieved through careful feature selection and rich feature sets. But NER is unique among the tasks we present in this dissertation, as many state-of-the-art approaches to NER do not utilize any syntactic information. Many traditional approaches to NER rely on purely sequential models, often relying on nothing more than lexical and part-of-speech information.

The tasks we turn to now – relation extraction and semantic role labeling – often require syntactic information in order to achieve state-of-the-art performance. Often this information will be incorporated into the system via features or kernels, and not explicitly modeled, as in the pipelined approach discussed in Chapter 1. We argue that, unlike NER, the task and syntactic structures are more inherently coupled, and thus both the potential benefits from joint modeling will be greater.

In this chapter we take advantage of the inherently coupled relationship that exists between syntax and relation extraction, modeling these tasks jointly, and using relation extraction annotations to guide the induction of a latent syntactic structure. We devise two methods for coupling syntax to relation extraction, one with phrase structure grammar and one with dependency grammar, and examine the effect of this choice on relation extraction performance. We contrast the effectiveness of these joint models with a model which makes use of syntactic predictions from a state-of-the-art parser, as would be common in the pipelined approach. Models which utilize syntax outperform the RE-only baselines, and, on the larger English data set, we find that inducing a latent syntax using the marginalization-based training method outperforms the use of parser-produced trees.

5.1 An Overview of Relation Extraction

Relation extraction (RE) is the task of identifying semantic relationships between sets of entities in text. Typically the entities in the text are assumed to be identified and labeled, making the goal of an RE system to identify which pairs of entities share in a relation, and to classify the type of relationship it is. Common relation types include[1]:

- PHYSICAL, where an entity is situated with respect to another.
  ex: *The Bridge Tavern is north of Riverrun.*
- PART-WHOLE, expressing a hierarchical relationship between entities.
  ex: *NBC is a subsidiary of Sheinhardt Wig Company.*
- PER, for personal or social relationships.
  ex: *She's my sister and my daughter!*
- ORG, for affiliation or ownership.
  ex: *Elon Musk is the Chief Product Architect of Tesla Motors.*
- AGENT-ARTIFACT, when an agent has possession of or creates an item.
  ex: *The Lady of the Lake, her arm clad in the purist shimmering samite, held aloft Excalibur from the bosom of the water.*

This kind of structured information can then be used to populate databases and answer queries, making it a useful task which may align closely with a user's goal.

Now consider the role that syntax might play in expressing these relationships. We identify two common relationships, depending on the nature of the relation:

- **Verbal Relations**

  In English, verbs play a central role in expressing many relations. A verb may carry a meaning which is highly informative of the relation type, and the arguments of the verb are commonly the entities which share in the relation. For instance, in the sentence "*Shigeru Miyamoto currently manages the Nintendo EAD branch*" the verb *manages* is very indicative of an EMPLOYS relation, as to manage an organization implies employment by that organization.

  But the correlation we focus on here is between syntactic *structure* and relations. Figure 5.1a illustrates the syntactic analysis of this sentence using dependency grammar. In relations of this type, it is common for the verb which specifies the relation to be the syntactic head, either directly or indirectly, of the relation's arguments. ACE English relation annotation guidelines categorizes this set of relations as *verbal* relations.

- **Nominal Relations**

  We classify another set of relations into the category of *nominal* relations. These are relations the ACE guidelines describe as being possessive, prepositional, or formulaic, among other

[1] These types are drawn from the ACE 2005 taxonomy [132]

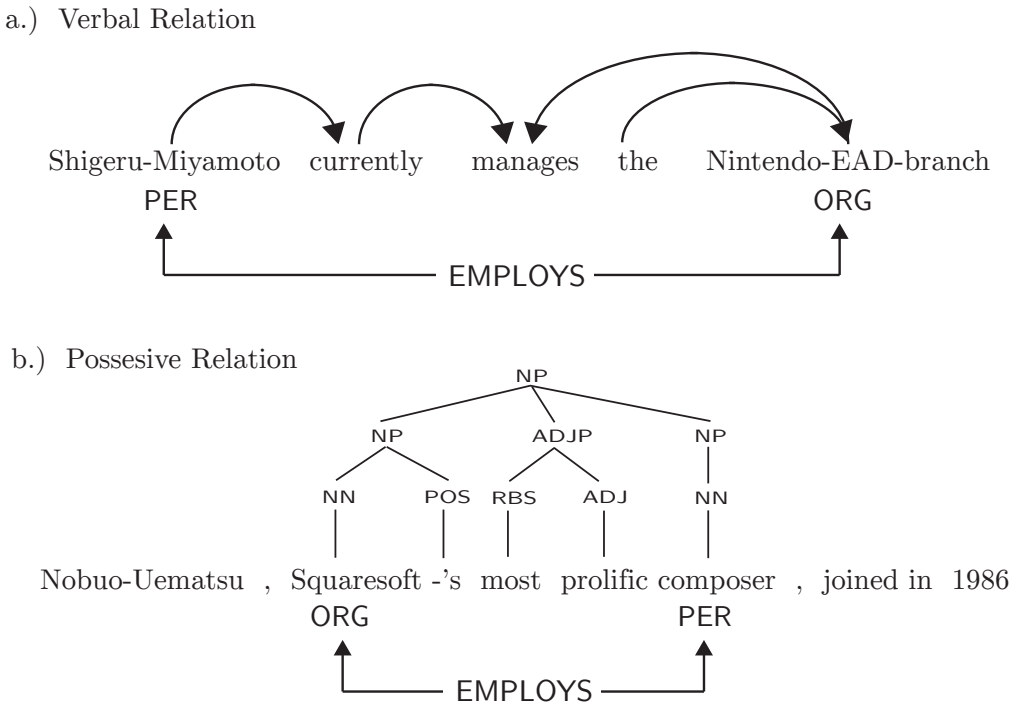


FIGURE 5.1: Relationships between syntax and relation extraction. *Verbal* relations (*top*) are a class of relations in which the relationship is generally captured by a verb (*manages*). This relation is expressible using a dependency graph, as the arguments are typically syntactic modifiers, either directly or indirectly, to the verb that defines the relation. *Nominal* relations (*bottom*) still have predictable underlying dependency structure, but also frequently occur at the edges of constituent spans. In this case it may be preferable to utilize phrase structure grammar when making RE predictions.

miscellaneous types. In the sentence "*Nobuo Uematsu, Squaresoft's most prolific composer, joined in 1986.*", the relation between the entities *Squaresoft* and *composer* does not involve any verb, and is instead expressed through the use of the possessive particle (*'s*). There are many relations of this sort, each type having its own method (or methods) of being expressed syntactically. We refer to relations of this type as *nominal* relations.

Some examples of nominal relations are the POSSESSIVE relation, as described above. Relations of this type can also be expressed through the use of a preposition (*the hounds of Baskerville*). Relations classified as FORMULAIC have the cookie-cutter constructions of newswire headlines (*April O'Neil, News Channel 6, Northampton MA*), and COORDINATION relations are expressed using conjunctions (*The Starks and the Lannisters*). Despite each type being expressed syntactically in slightly different ways, in many of these cases the two entities will each correspond to a syntactic noun phrase, and a noun phrase will begin at the first entity and end with the second entity. This correlation is illustrated in Figure 5.1b

In previous work the inclusion of syntax in RE systems has produced mixed results. The work of [133] utilizes multiclass SVMs to make RE predictions, and finds that syntactic information is only marginally useful. The authors reason that this is due to the large number of highly local relations in the ACE data set, citing more than 70% of relation extraction instances as being nested

entity mentions, or adjacent terms, separated by no more than one word. In contrast, syntactic information has shown to be very useful in graph-based approaches to relation extraction [134], and systems which rely on graphical models [135]. And in unsupervised and distantly-supervised work, dependency path features are one of the important kinds of information [136, 137]

In our approach we model syntax explicitly, and connect syntactic variables to relation extraction variables using soft Boolean logic constraints. We now propose two methods for integrating syntax and relation extraction, motivated by the two types of correlations discussed above.

5.1.1 Factor Graph Models for Relation Extraction

We represent relation extraction as a pairwise model: a set of Boolean variables is used to predict the presence or absence of a relation between any two entities, and a separate set of variables is used to classify the type of relation if one exists. This is implemented in a manner which is nearly identical in structure to way constituent spans are predicted and labeled in the phrase structure parser from Section 3.1.2 (pg. 79).

We observe a sequence of words $W = \{w\}_1^N$. Entities have been identified and collapsed into single word expressions.

- Let $\{Rel(i,j) : 0 \leq i < j \leq n\}$ be $O(n^2)$ Boolean variables such that $Rel(i,j) = \textsf{true}$ iff a relation is present between the entity at position i and the entity at position j. We treat relations as *undirected* pairwise relationships.

- Let $\{Rel\text{-}Label(i,j,\lambda) : \lambda \in L$, and $0 \leq i < j \leq n\}$ be $O(|L|n^2)$ Boolean variables, where L is the relation label set. $Rel\text{-}Label(i,j,\lambda) = \textsf{true}$ iff a relation of type λ is present between the entity at position i and the entity at position j.

- Let $\{\text{IsAtMost1}(i,j) : 0 \leq i < j \leq n\}$ be $O(n^2)$ factors, each coordinating between a relation prediction variable $Rel(i,j)$ and the set of corresponding label variables $Rel\text{-}Label(i,j)$. This factor ensures that a single label variable is true if the corresponding relation variable is true, otherwise all variables are false.

Here the $Rel(i,j)$ and $Rel\text{-}Label(i,j)$ variables simply express the representation of the problem, while the AtMost1 factors are logical constraints ensuring that only one label will apply to a particular relation. There is also a corresponding set of unary factors for each of the variables mentioned above, but their definitions are omitted for clarity.

In practice this space is heavily pruned: only a small subset of words in a sentence are likely to be entities, and because we observe the location of entities and their labels, only variables for each potential pairwise relation between variables will be instantiated. If there are m entities in a sentence, the factor graph for this sentence will contain $\frac{m(m+1)}{2} - 1$ sets of $Rel(i,j)$ and corresponding $Rel\text{-}Label(i,j,\lambda)$ variables.

5.1.2 Factors for Coordinating Relation Extraction and Syntax

The novelty of our approach hinges on our ability to easily and effectively couple end tasks, like relation extraction, to syntactic representations. In this section we detail two methods of coupling

syntax with relation extraction, one with dependency grammar, the other with phrase structure grammar, reflecting our prior beliefs of the nature of their relationships (as described in Section 5.1, pg. 126).

- **Coupling with Dependency Structure**

  We first assume a representation of dependency syntax, identical to the model presented in Section 3.2.2, including the DEP-TREE factor. We coordinate between this structure and the relation extraction model with the addition of the following factors:

  Let $\{\text{D-CONNECT}(i, j, k) : 0 \leq i < k \leq n; 0 \leq j \leq n\}$ be $O(n^3)$ *ternary* soft NAND factors, each coordinating between two syntactic dependencies, represented with $Link(j, i)$ and $Link(j, k)$ variables, and a single relation variable, $Rel(i, k)$. In this context word j functions as a latent syntactic head shared by both entity i and k, based on the assumption that two entities which appear in a relation will be the syntactic children of the same head word, as may be the case with many verbal relations.

  As in previous sections, the connections between components models are learned via the weights of the features associated with the coordinating factors. Extending the Boolean logic to the ternary case, when all three variables are true, the factor contributes a potential ϕ to the model score. It otherwise contributes a potential of 1.0. If ϕ is > 1.0, we might learn something akin to, "*entity i of type t_i is likely in a relation with entity k of types t_k when they share the syntactic head j, and j is the word 'employs'.*" Note that each $Link$ variable will be connected to many D-CONNECT factors.

- **Coupling with Phrase Structure**

  We first assume the presence of a labeled phrase structure representation, identical to the model presented in Section 3.1.2, including the CKY-TREE factor. We coordinate between this structure and the relation extraction model with the addition of the following factors:

  Let $\{\text{C-CONNECT}(i, j) : 0 \leq i < j \leq n\}$ be $O(n^2)$ soft NAND factors coordinating between syntactic $Span(i, j)$ and relation $Rel(i, j)$. This coupling is identical to the method used to coordinate between $Span$ and NER variables in Section 4.3.2 (pg. 113): one variable from the syntactic representation pairs directly with one variable from the end task.

The logic behind the D-CONNECT approach to dependency syntax coupling is that by decomposing a relation into a two dependency arcs, each specifying a syntactic head/modifier relationship, we make explicit the notion of a head word in the model. This head word may contain important cues for determining whether a relation is present, and what type of relation it is (though our model captures only the former).

For instance, in the example sentence (Fig. 5.1a) the model may already have a strong belief that *Shigeru-Miyamoto*, of type PER, and *Nintendo-EAD-branch*, of type ORG, may share a relation, because PER and ORG entity types are often related. The additional knowledge that these two entities share a syntactic head strengthens this hypothesis. Similarly, entities of type PER and ORG may frequently occur in relations, but knowing that they are both headed by the verb *manages* is indicative of its relation type label, EMPLOYS, though we leave this modeling extension as future work.

This information can be used in a constituent-based model, but it is not simple to obtain. Here we collect features based on the words and word types that fall between the entities that define the the span. This may work well for nominal relations, but may also capture local verbal relations via rich features. For instance, in many cases the span will contain just a single verb, and thus a feature can embody the ORG-EMPLOYS-PER information we aim to capture in the dependency variant for verbal relations. For larger spans this approximation is less useful, as spans will be more likely to contain multiple verbs. In sentences with multiple entities the relationship between entities and their heads will be lost as the notion of a syntactic head is not made explicit in the model.

5.2 Experiments

Our experiments contrast three modeling scenarios, based on how a real-world NLP system might be constructed to perform relation extraction. In the simplest case, a relation extraction system would have no syntactic information. Instead the model would rely only on information like word and part-of-speech information that can be reliably predicted using pre-trained models. In one scenario, a model might be trained solely from part-of-speech and named entity information. Few data sets which are annotated with both syntactic trees and relations, and thus the most realistic method for incorporating syntactic annotations into the model is to first parse the training data using a pre-trained parser, and then use these trees to construct features for training the relation extraction model (i.e., the pipeline approach).

Alternatively, in our approach the relation extraction annotations are used to guide the induction of a latent syntactic structure. This eliminates the need for syntactic annotations in the training data. Another advantage of this approach is that the syntactic representation learned by the model is guaranteed to be well-suited to the training data. Pre-trained parsers can suffer from domain drift, where the data used to train the parser is not representative of the test data causing the parser to generalize poorly to new data. A disadvantage of this approach is that the learning signal from the relation extraction annotations may not be strong enough to induce a useful latent structure (unsupervised approaches to syntax are significantly less accurate than their supervised counterparts).

We examine the performance of these three configurations on English and Chinese relation extraction data. This allows us to compare the effectiveness of our two proposed syntactic coupling methods.

5.2.1 Data

We evaluate these models using the 2005 Automatic Content Extraction (ACE) data set [132], using the English (dual-annotated) and Chinese (annotator #1 only) sections. Each corpus is annotated with entity mentions using a seven-type entity label set: PER, ORG, LOC, as introduced in Section 4.1, as well as GPE for geo-political entity, FAC for facility, VEH for vehicle, and WEA for weapon. In addition, many of these entities are annotated with entity subtypes, e.g., a government organization has type ORG and subtype GOV.

Formatting the data and generating the necessary annotations is a lengthy process, and we relied heavily on the Stanford CoreNLP toolkit. The data was first split from raw text paragraphs into sentences, and then tokenized. The English data was tokenized according to the Penn Treebank

standardization, and the Chinese according to Penn Chinese Treebank standards [138]. Sentences were then annotated with part-of-speech tags using a pre-trained, state-of-the-art CRF-based tagging model [41], and parsed using the Stanford factored parsing model [110]. The pre-trained grammar is trained from a variety of sources, including biomedical, translation, and question data in addition to the standard Wall Street Journal corpus. We annotate two versions of the corpus, one using the unaltered constituent trees produced by the Stanford parser, and a second corpus annotated with the dependency trees taken from the corresponding Stanford typed dependencies.

We then reintroduced the entity and relation annotations to the data, collapsing multi-word entities into a single term. Unfortunately in many instances the tokenization, tagger or parser would fail, or we could not guarantee that the alignment between the ACE annotations and the resulting text was correct. In these cases, the sentence was discarded from the data set. In addition, we filter out sentences with fewer than two entities (which are incapable of containing relations) and sentences with more than 40 words (because they may be unreliable). This yields 6966 sentences in the English data, and 747 sentences for the Chinese. Nine of every ten sentences constitute the training set, with every tenth sentence reserved for test.

5.2.2 Model Configurations

We evaluated six different systems. Two were variants of a baseline system, two systems use parses produced by the pre-trained Stanford parser (one for each formalism), and two systems infer syntax in a joint model:

- **Baseline**
  The baseline model is the edge-factored model described in Section 5.1.1, consisting of $Rel(i, j)$ variables, corresponding $Rel\text{-}Label(i, j, \lambda)$ variables, and the set of factors that attach to them and coordinate between them. Features for this system, which are common to all model configurations, are combinations of lexical information (i.e., the words that form the entity, the pos-tags of the entities, etc.) as well as the distance between the relation. This is a simple model and does not attempt to exhaustively leverage all proven sources of useful information [75].

- **Baseline-Ent**
  The features in the BASELINE model are very simple, but represent a reasonable choice when an NER system is not available, or not reliable. As noted previously, ACE data annotations specify an entity type and a fine-grained entity subtype. This information is very useful for determining the presence of relations and their relation types. BASELINE-ENT represents a variant of the BASELINE model in which additional features are added using entity type and sub-type information.

- **Oracle-DP**
  In this configuration we instantiate the joint dependency syntax model, as described in Section 5.1.2. We set the syntax variables to their observed values in the data, as determined by the Stanford parser-produced dependency trees, allowing the DEP-TREE factor to be omitted from the model. Training is fully supervised, and the values of syntactic variables are observed both during training and testing.

- **Oracle-CP**
  A constituent syntax analogue of ORACLE-DP, using the constituent parses provided by the Stanford factored parser.

- **Hidden-DP**
  A model in which dependency syntax is induced using marginalization-based training. It is structurally identical to ORACLE-DP, except that the DEP-TREE factor is required, as it is needed to constrain the set of latent syntactic variables. Because syntax is treated as a latent variable, no syntactic annotations are required during training or testing.

- **Hidden-CP**
  The constituent syntax analogue of HIDDEN-DP.

As in previous results, models which require syntactic annotations during training are marked with a † symbol and models which rely on syntactic annotations during both training and testing are marked with †† symbol.

5.2.3 Features

The features in the BASELINE system are simple, and are used only for predicting the presence of a relation and its type. These features include the words, part-of-speech tag, and capitalization information of each entity in the candidate relation, as well as the distance between them. For the relation from w_1 to w_5 in the sentence "*Shigeru-Miyamoto currently manages the Nintendo-EAD-Branch*", these features are:

Feature	Comment
[e1]-Shigeru-Miyamoto	String form of left entity
[e2]-Nintendo-EAD-Branch	String form of right entity
[e1-w]-Shigeru	Each word contained within left entity
[e2-w]-Nintendo	Each word contained within right entity
[e1-pos]-NN-NN	Concatenated POS-tags of left entity
[e2-pos]-NN-NN-NN	Concatenated POS-tags of right entity
[e1-e2-pos]-NN-NN-NN+NN-NN	Concatenated POS-tags of both entities
[dist]-4	Distance between entities
[e1-e2]-Shigeru-Miyamoto-Nintendo-EAD-Branch	Entity bigram
[e1-w-e2-w]-Miyamoto-Nintendo	Word-level Bigram

The BASELINE-ENT model extends upon the BASELINE model to incorporate entity label and sub-type label features.

Feature	Comment
[e1-t]-PER	Left entity type
[e2-t]-ORG	Right entity type
[e1-e2-t]-PER-ORG	Entity type bigram
[e1-st]-EMPLOYEE	Left entity sub-type
[e2-st]-COMPANY	Right entity sub-type
[e1-e2-st]-EMPLOYEE-COMPANY	Entity subtype bigram

In models with syntax, the features associated with the syntactic factors are those described previously in Chapter 3. For dependency parsing this is the [102] feature set, and for phrase structure parsing we use the features listed in Section 3.1.2 (pg. 3.1.2).

The remaining features are those which govern coordination between component models.

Feature	Comment
[co-dist]-4	Distance between entities
[co-e1-e2]-Shigeru-Miyamoto-Nintendo-EAD-Branch	Entity bigram
[co-e1-w-e2-w]-Miyamoto-Nintendo	Word-level Bigram
[co-e1-t]-PER	Left entity type
[co-e2-t]-ORG	Right entity type
[co-e1-e2-t]-PER-ORG	Entity type bigram
[co-e1-st]-EMPLOYEE	Left entity sub-type
[co-e2-st]-COMPANY	Right entity sub-type
[co-e1-e2-st]-EMPLOYEE-COMPANY	Entity subtype bigram

For the coordination factors in the dependency syntax models, we add the following features which include information about the latent head word:

Feature	Comment
[co-e1-e2-t-hw3]-PER-ORG-manages	Type bigram with headword
[co-e1-e2-st-hw3]-EMPLOYEE-COMPANY-manages	Subtype bigram with headword

In initial experiments we found the baseline models were able to outperform the syntactic models when coordination features consisted solely of distance, word, and part-of-speech information. The inclusion of entity label features appears to be crucial for learning a useful latent syntax.

5.2.4 Design

All models are trained using 20 iterations of stochastic gradient descent, with a learning rate $\eta = 0.01$, and using L2 regularization (a Gaussian prior, i.e., variance = 1.0). We use fewer iterations of SGD than in other experiments because the ACE data sets used here are comparatively much smaller. Models which contain a representation of dependency syntax are cyclic and loopy belief propagation is used, a max of 10 iterations per example, to find an approximate inference solution. For the remaining models the graph is acyclic and inference is exact, requiring two iterations of BP. Models are decoded by positing a relation for all i, j where the beliefs of $Rel(i, j) > 0.5$, and taking the label λ which maximizes $Rel\text{-}Label(i, j, \lambda)$ for each predicted relation span.

On a quad-core 2.8GHz intel processor, calculating parameter updates in parallel (mini-batches of size 4), training for the BASELINE model takes 45 seconds per iteration (21k features), 55 seconds per iteration for the ORACLE-CP model (26k features), 3.2 minutes for the ORACLE-DP model (44k features), 1.7 minutes for the HIDDEN-CP model (81k features), and 5.4 minutes HIDDEN-DP model (120k features). The BASELINE and BASELINE-ENT models have approximately identical training times despite the BASELINE-ENT model containing some additional features.

Models are evaluated in terms of precision, recall, and F-measure, for both labeled and unlabeled cases. An unlabeled prediction for (i, j) is considered correct if there is a relation between the entities at i and j in the gold data. A labeled prediction is considered correct if this is true, and the predicted label matches the label of the relation type in the gold data.

5.2.5 Results

The results of our experiments are presented in Table 5.1. In both languages we find that at least one joint model significantly outperforms the baseline, demonstrating the usefulness of the marginalization-based training method when no syntactic training data is available. In some circumstances these models even outperform those which rely on the parses produced by the pre-trained Stanford parsing model, indicating that this method's usefulness is not limited only to low-resource languages and domains.

Performance of Latent Syntactic Structure Models

On the English data we find that all models which utilize syntactic information outperform their respective syntactically-uninformed baselines. This results in an 1.9% average F1 improvement for the dependency-based models, and a 3.2% improvement for phrase structure models. Here we find that both HIDDEN models, which learn a latent syntax, outperform their ORACLE counterparts, which use trees produced by the Stanford parser.

There are many possible explanations for this. Because the parser is pre-trained, it is possible that the data used to train the Stanford parser may differ greatly from the ACE data sets we use in our evaluations, and the parse trees it produces are not very accurate. Performance drops in out of domain parsing is well-documented [13], and while both the ACE data set and the Stanford parser training data consist largely of newswire, it may still affect the parser's performance. This is a disadvantage of a traditional pipeline approach where models are trained independently from different data sources.

It is also possible that the assumptions made by the models – that a constituent span or a pair of dependency edges align directly to a relation – may be too constraining for the ORACLE systems. Note that in Fig. 5.1 the dependency from *Shigeru-Miyamoto* to *manages* is represented with two dependency arcs, and is therefore not captured by the D-CONNECT-style of model coordination. A model would need to utilize higher-order connections to capture this phenomena. In other words, D-CONNECT coordinates between $Rel(i, j)$ with latent head k, and all possible *pairs* of directed arcs between (i, k) and (j, k). A higher order version of this factor would need to connect *triples* of syntactic dependencies before it could correctly capture the relationship between the relation and the gold syntax in the example above. Prior work has shown how to model higher-order connections with additional factors [11], but we leave these extensions unexplored. Thus it is

ACE Results												
	English						Chinese					
	Unlabeled			Labeled			Unlabeled			Labeled		
Model	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Baseline	85.4	57.0	68.4	83.0	55.3	66.4	42.9	26.8	33.0	42.6	21.3	28.4
Baseline-Ent	87.2	65.4	74.8	85.8	64.4	73.6	55.2	31.1	39.8	51.2	29.4	37.4
Oracle D-P††	89.3	67.4	76.8	89.3	66.2	75.4	60.0	32.6	42.2	58.1	31.3	40.7
Hidden D-P	87.8	69.8	77.7	85.3	67.8	75.6	48.0	32.0	38.4	47.2	30.0	36.7
Oracle C-P††	89.1	68.7	77.6	87.5	67.5	76.2	66.8	37.8	**48.3**	63.8	37.0	**46.8**
Hidden C-P	90.5	69.9	**78.9**	88.8	68.6	**77.4**	56.3	32.3	41.0	53.4	31.6	39.7

TABLE 5.1: Relation extraction results on ACE. Models using hidden phrase structure syntax provide significant gains over the syntactically-uniformed baseline model in both languages, but the advantages of the latent syntax were mitigated on the smaller Chinese data set.

possible that the parser-produced trees are of high quality, but these modeling assumptions cannot take full advantage of them. A pipeline approach would avoid this shortcoming by allowing the construction of arbitrarily complex features from the parser-produced trees, which can then be used to aid relation extraction. In order to directly compare identical models with various training methods, we do not pursue this method here.

On the smaller Chinese data set the HIDDEN-CP model still provides significant improvements over the baseline, but is outperformed, by a large margin, by the ORACLE-CP model. The HIDDEN-DP model does not improve over the BASELINE-ENT. The lower absolute performance, relative to the English data, may be indicative of having too little training data to generalize to new data, and this effect may be more pronounced in the latent syntax models which have far more parameters to learn.

Dependency Structure vs. Phrase Structure

Models based on phrase structure consistently outperform their dependency structure counterparts. This may be due to the larger number of factors and parameters that constitute the interface between relation extraction and dependency grammar, an issue that may be exacerbated by the small size of the training data. Alternatively, [133] cite the large number of local relations in the ACE data as a contributing factor to the lack of significant performance increases by incorporating syntactic information into their model. It is possible that constituent spans are more useful in identifying and supporting these local nominal relation predictions than dependency structure is able at successfully identifying verbal predictions.

On the Chinese data we again find the phrase structure models produce the highest accuracy, but here the parser-produced trees significantly outperform both the induced syntax models. The Chinese data set is very small, and potentially too small to learn the large number of parameters associated with latent syntax and coordination between component models. The small data is also likely a cause of the low overall performance of all models. Even in this scenario, the HIDDEN C-PARSE model still represents a low-resource method for recovering some of the gains associated with syntactic information, in situations where pre-trained parsers and syntactic annotations are unavailable.

5.3 Conclusion

In this section we demonstrated the usefulness of the marginalization-based approach for training joint models in the presence of latent syntactic variables. We also compared the effectiveness of different syntactic representations in the context of a joint model. For the task of relation extraction, at least on these relatively small data sets, we find the simplest approach to latent syntactic structure to be the best: latent phrase structure outperforms latent dependency structure by between 1-3% labeled F1.

For the larger English data set we find that inducing a latent syntax is not merely an attractive approach for when syntactic training data is unavailable, but the highest performing approach in general. This is a promising result: in practice the use of pre-trained parsers is necessary for providing parse trees for downstream NLP, with the only alternative being the use of fully unsupervised models with significantly lower accuracy. Outperforming these models with marginalization-based training indicates that it may be an approach suitable for both low-resource domains, and for NLP in general. For languages or domains where syntactic training data is scarce or unavailable, the latent syntax approach may offer a level of performance that would otherwise be unattainable.

A limiting factor in these experiments is the size of the data. The need to heavily pre-process the raw text to produce high-quality tokenized, part-of-speech tagged data with parse, entity, and relation annotations meant that a disappointingly large amount of the original text was discarded. Producing the data set for these experiments has yet another disadvantage: because the resulting data set is unique to our work, we cannot compare to current state-of-the-art systems. In the next chapter we aim to address these concerns by evaluating our joint modeling framework and marginalization-based training method on a standardized task, with a standardized data set.

6

Semantic Role Labeling with Latent Syntax

In this dissertation we present several joint models, covering a number of common NLP tasks. We have shown previously how our belief propagation approach to joint inference in factor graphs can produce state-of-the-art results on a fully supervised named entity recognition task, outperforming other joint inference methods. We then demonstrated the effectiveness of marginalization-based training with hidden structure, using it to induce latent syntactic representations and improving a model of relation extraction. The models which induced a latent syntax were also shown to be preferable to models which utilized parse trees produced by a pre-trained state-of-the-art parser, which might be used in an NLP pipeline.

Our aim in this chapter is to demonstrate the effectiveness of marginalization-based training with latent variable syntax models, but in a more sophisticated model and on a task with standard data sets and evaluation. Here we turn to semantic role labeling, where we present a simple factor graph model for the task, and describe novel extensions for capturing unique aspects of the problem – sense prediction and valency – by using variable chains with specially-defined transition matrices. We extend this baseline to produce a joint model of SRL and dependency syntax.

We use a standard data set in this chapter which was constructed for a shared task competition. This allows us to directly compare our system to many other systems. As in the relation extraction experiments (Section 5.2.2, pg. 131), we compare the performance of our latent syntax model using marginalization-based training to an "oracle" system and a syntactically-uninformed baseline model. The latent syntax and oracle models are structurally identical, but differ in that the oracle is given the true parse trees during training and at testing. Unlike the relation extraction scenario, where these trees were produced from a pre-trained state-of-the-art parser, the shared task data contains very accurate dependency parses. For most languages these trees are hand-annotated, making them a potentially more reliable source of syntactic information than the trees used in the previous chapter.

Despite this, we obtain results similar to those for relation extraction presented in Chapter 6. For most languages the latent variable syntax models perform comparably to the observed syntax

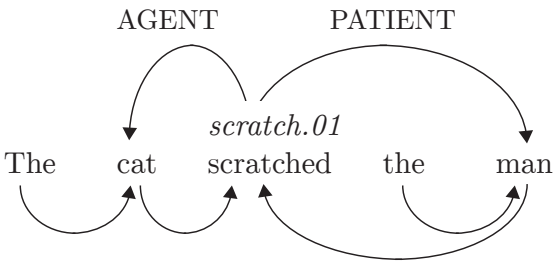


FIGURE 6.1: An example sentence with dependency tree and SRL annotations. Here the semantic dependencies of the SRL analysis correspond directly to syntactic dependencies over the same indices. The identity of a particular predicate sense is generally marked with a numeric suffix on the predicate's lemma. Here *scratch.01* indicates that this sense of *scratch* is the first out of a potentially larger set of senses, distinguishing the current sense "*to score the surface (of something) with a sharp or pointed object*" from alternatives like "*to cancel or strike out (writing) with a pen or pencil*."

models, achieving on average more than 90% of the performance difference between the baseline models and the oracles. In some languages, for reasons we explain below (pg. 157), the use of marginalization-based training with latent syntax provides an improvement over the models which use of gold parses.

We begin by reviewing the task, and discussing the relationship between dependency structure and the predicate-argument structure involved in semantic role labeling. In Section 6.2 (pg. 143) we introduce a series of factor graph models for semantic role labeling. A simple edge-factored model serves as the baseline, an extension improves upon this by modeling valency and sense prediction jointly. The full joint model couples this structure with a dependency parsing model. We conclude (Section 6.3.5, pg.153) by comparing the performance of our model to competing systems in the 2009 CoNLL shared task [139], and examining the syntactic structures induced by the latent syntax model.

6.1 Semantic Role Labeling

The application of statistical methods to syntactic parsing resulted in significant improvements to parse accuracy [84, 140], but aside from the construction of treebanks, these parse trees serve little practical purpose in and of themselves [1]. In the case of Penn Treebank style annotations, constituent trees are complicated structures with a large sets of constituent labels that precisely define the syntactic relationships present in the corpus, but ultimately this is still far removed from a theory of meaning. For practical NLP, where tasks like question answering and database construction require knowledge of the semantic relationships between entities, a syntactic tree only provides part of the answer.

In contrast, the aim of semantic role labeling (SRL) is to capture the core meaning of a sentence. SRL annotations (Fig: 6.1), inspired by the frame semantics of Charles Fillmore [144],

[1] Notable exceptions include much of the work pursued in corpus linguistics, from both historical [141] and psychological [142] perspectives. Treebanks can also be utilized to train models which do not aim to predict trees, as is the case in tree-based MT systems [143].

decompose the semantics of a sentence into a set of semantic frames. Each frame is built around a *predicate*, an important verb or verb-functioning word. Due to homonyms and subtle semantic variation even within the same word, one token can refer to a number of possible underlying senses. Verbs like *saw* are ambiguous, and can refer to the act of seeing or to the act of cutting – two different predicates with the same orthographic form. To identify the semantics of the sentence it is necessary to identify precisely which of these senses is intended. This is indicated using a numeric suffix attached to the predicate's lemma (i.e. *saw.01* vs. *saw.02*).

Each predicate takes zero or more words as *arguments*. Typically these are nouns or noun-functioning words which play a particular *semantic role* within the frame. Figure 6.1 describes a semantic frame where the word *scratched* is the predicate, and *cat* and *man* are arguments. In this context the cat is the one who has inflicted the scratches, and hence the role it plays is that of the AGENT. The man has been scratched *by the cat*, and as the victim of the scratching he plays the role of PATIENT. Note that the semantic roles are identical regardless of whether the sentence is phrased in active voice, as illustrated, or in the passive voice, as in"*The man was scratched by the cat*". That is, SRL annotations abstract over simple grammatical alternations like voice [2].

6.1.1 The Role of Syntax in SRL

In many ways SRL is a simpler task than parsing. Parse trees, especially constituent trees, are often more deeply structured than SRL analyses, which are multi-rooted trees of height one. Simple SRL models do not enforce the same structural constraints that necessitate the use of dynamic programming for efficient parsing, and many SRL systems predict and label predicate-argument pairs with maxent classifiers [146, 147] or SVMs [148]. However, the presence of semantic relationships is still heavily constrained by the underlying syntactic structure of the sentence, and thus the two tasks are inherently linked: to solve SRL at a state-of-the-art level, one must first obtain syntactic analyses.

Evidence of this comes from the CoNLL shared tasks, which helped define SRL as a standard task and where early competitions even included a set of parses, for use at both train and test time, to provide a level playing field in the competition [149]. The highest performing system in the competition made use of parse structure to prune away conflicting analyses from the space of solutions, making it feasible to use a global model for SRL predictions [19], while the use of additional parsers in the open challenge further improved performance in that track. In addition, many researchers have sought to improve SRL performance by better exploiting syntactic analyses, either by incorporating new syntactic information from other syntactic formalisms [150], or by training a joint model to rerank parses to best suit SRL [147].

What is the precise nature of the relationship between SRL and syntax? For the sake of continuity with the models that will follow, let us focus on the relationship between semantic role labeling and dependency parsing. Recall that verbs play an important role in dependency grammar – a verb serves as the root of the dependency graph, and verbs serve as the parents, either directly or indirectly, of the other word classes which modify them. The nature of its correlation with

[2] Though it also common for this distinction to be made in syntax, where constituent labels often discriminate between subject NPs and object NPs. For instance, the Penn Treebank II guidelines suggest including annotations for these grammatical roles [145]. However, semantic role labels exist for specifying many additional roles not addressed by syntax alone.

		Gold Trees				Parse Trees			
Language	# sents	1	≤ 2	≤ 3	% cross	1	≤ 2	≤ 3	% cross
Catalan	13200	100.00	100.00	100.00	0.0	91.12	93.82	94.80	1.05
Chinese	22277	78.09	78.58	78.68	4.97	64.98	68.83	70.13	7.08
Czech	38727	63.90	86.26	89.97	2.74	61.21	80.26	83.84	4.29
English	39279	63.62	64.56	64.70	2.42	59.08	61.29	62.09	3.52
German	36020	77.22	77.98	78.03	3.99	72.34	75.55	76.48	4.12
Japanese	4393	14.32	51.52	60.79	14.68	13.13	48.30	56.74	16.65
Spanish	14329	100.00	100.00	100.00	0.0	91.12	93.99	95.29	0.85

FIGURE 6.2: Correspondence between syntax and SRL predicate-argument pairs. Statistics calculated from the CoNLL 2009 shared task data set. This table indicates the % of SRL predicate-argument pairs that correspond to directed paths in the dependency tree. The statistics in the left partition reflect the correlation between SRL and the gold trees provided by the data set, while the trees used in the right partition are from parser-produced trees, also provided in the data set (HEAD vs. PHEAD fields).

semantic role labeling is therefore straightforward: predicates are often verbs, and thus function, directly or indirectly, as the syntactic heads of their arguments. If two words exist in a predicate-argument relationship, then we can expect that a directed path exists from argument to predicate in the dependency syntax.

We formally define the *order* of the relationship between syntax and SRL as follows:

*Let $A(i, j)$ represent an SRL predicate-argument pair from predicate i to argument j and $head(i)$ denote the syntactic head of word w_i. For a given pair of indices i and j, a **first order** correlation between SRL and syntax exists if $A(i, j) \implies head(j) = i$, where $A(i, j)$ specifies j is an argument of i, and $head(j)$ returns the syntactic head of the word at index j.*

A second order relationship can be defined similarly:

*For a given pair of indices i and j, a **second order** correlation between a dependency tree and an SRL argument pair exists if $A(i, j) \implies \exists k : head(j) = k \wedge head(k) = i$.*

The table in Figure 6.1.1 specifies the extent to which this correlation holds for dependency paths of various lengths (1 = first order, 2 = second order, etc.) across a collection of seven corpora. For a majority of the languages represented there is a strong implication between the two task structures: if word i is a predicate and word j its argument, there is likely a direct correspondence in the dependency syntax, with word i functioning as the syntactic head of word j. This is especially true of the Catalan and Spanish data sets, where it holds without exception. However, there are instances where the degree of first order correlation can be exceedingly low. This is the case for Japanese, where a mere 14.32% of SRL predicate-argument pairs have a corresponding dependency path of length 1.

It is worth noting that the extent of the correlation between syntax and SRL is as much a product of the annotation style as it is one of language characteristics. This is highlighted in the Japanese data, where the annotation style causes a low degree of first-order correlation, and a relatively high degree of second-order correlation. Japanese marks topic, subject, and object using suffix particles, and in Japanese language processing it is common to tokenize these separately. If a syntactic dependency exists between two words, these particles are included in the analysis and will

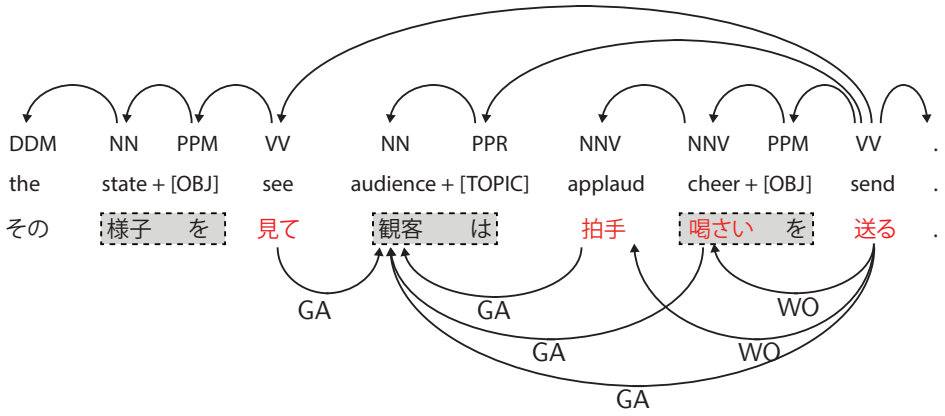


FIGURE 6.3: An example of crossing syntactic and SRL dependencies. Syntactic dependencies are shown above the syntax, semantic dependencies below. Predicates are shown in red, multi-morpheme expressions boxed in grey. In Japanese, where the correlation between syntax and SRL is the weakest in the CoNLL data, morphological segmentation is a common pre-processing step which precedes parsing and other NLP tasks. A syntactic analysis will often attach a word to its head-word's particle before the head-word's root, while roots can be linked directly in the SRL analysis. This leads to a high degree of mismatch between the two annotations, including a larger degree of crossing dependencies.

intervene in the dependency path. In contrast, SRL dependencies link directly between the roots of words, often omitting their particles. Thus few SRL dependencies correspond to a single syntactic dependency in Japanese, but many correspond to 2-arc dependency paths. This phenomenon also contributes to the higher degree of crossing brackets, where for an SRL dependency over (i, j), there exists a syntactic dependency (k, l) such that $i < k < j < l$. Both types of mismatch are illustrated in Fig. 6.3.

In a many situations it is difficult to obtain high-quality syntactic analyses, and so the possibility of SRL without parse trees is a promising endeavor. Marginalization-based training with latent syntax provides one method to accomplish this, and also circumvents problems associated with ill-suited syntactic representations. In the following sections we will show that in cases of extreme mismatch, as in the CoNLL 2009 Japanese data set, an induced syntax can align more directly with the end task of SRL and improve performance over similar models which use the parses specified in the data.

6.1.2 Related Work

SRL has benefitted greatly from being the focus of a series of shared task competitions. An initial surge of work occurred in response to the 2004 CoNLL shared task on SRL[151], and many systems used simple maximum entropy [146] or SVM classifiers [148]. These classifiers predict "*Is there a predicate-argument relationship between a given pair of words?*" and "*What is the most likely label for this relation?*" The shared task data set paired SRL annotations with hand-annotated constituent trees from the Wallstreet Journal Treebank, often reducing the SRL task to one of mapping trees to useful features for these pairwise classifications.

However, there are disadvantages to modeling these predictions independently. Naturally the sense of the predicate is inherently linked to the number and type of roles that predicate takes,

which in turn influence each other. A predicate *scratch* may tend to have one sense if there is an agent and patient who are likely to claw and be clawed, and a different sense if it has only an agent, and that agent is a billiards player. Modeling these dependencie jointly significantly improves accuracy compared to independent predictions. Popular approaches for performing or approximating global inference in SRL includes integer linear programming [16], n-best lists [152], and Markov Logic, which reduces such constraints to weighted first-order logic formulas [153, 154]. The Markov Logic approach has connections to our own, as we express weighted logic constraints in terms of factors whose potential tables encode such logic, and both approaches rely on graphical models (though it is common to perform inference via sampling in Markov Logic). We discuss this issue in further detail in Section 6.2.2 (pg. 144).

The goals of the final SRL shared task broadened to focus on the joint predicting both semantic roles and syntax [139]. Annotations for both SRL and dependency syntax are provided at training, but competitors were encouraged to predict both at test time (joint track). Optionally, competitors could also observe syntactic annotations at test time (SRL-only track). It might stand to reason that a competition of this nature would serve to demonstrate the usefulness of joint inference over pipelined system architecture, but, echoing the negative results of earlier joint SRL and parsing systems [147], was not the case. The top system in both tracks were based on pipelined systems.

In the joint track, [155] utilized a second order graph-based parser in the style of [140] to provide a syntactic analysis. Based on features derived from these parses, a maximum entropy classifier scores the potential of each word to serve as an argument, and to function as a particular semantic role. An ILP solver attempts to find the global optimum by coordinating these independent scores, subject to a series of high-level constraints. Sense labeling is done separately by an SVM. Many systems use similar approaches, but differ in the type of parser, the method of solving the global optimization, and their respective feature sets. The best system in the SRL-only track, [156] was largely an exploration of automatic feature discovery, again using maximum entropy models, but relying on a state-of-the-art transition-based dependency parser to serve as a source of features guiding the SRL predictions.

Thus the general approach common amongst many of the current state-of-the-art systems is to first obtain the most accurate syntactic analyses as possible, extract features from these parses, and utilize these features in a global solver. Few systems attempt to solve SRL without the use of syntactic training data. Two notable exceptions include the work of [157] and [158]. [157] present a system for SRL in Polish, where both syntactic and SRL annotations are scarce. They circumvent the lack of syntactic training data by hand-crafting a shallow ("chunk-based") parser. Argument identification improves significantly when using features derived from the shallow parser, but this still requires manual creation of a grammar for the parser. More similar to our approach, [158] propose utilizing SRL annotations to find useful parse trees without requiring a pre-trained parser. During training a baseline parser produces a forest (an efficient packed representation of trees), and the SRL annotations are used to heuristically identify and prune away the most incompatible entries. However, the syntactic formalism used in this work, combinatorial categorical grammar (CCG), places a great deal of syntactic information in the lexicon. While this approach does not require a pre-trained parser, it does require a dictionary of lexical terms (which also specifies part of its syntactic role), it is not clear to what degree this method can be considered unreliant on syntactic training data.

More recently, the work of [159] builds upon our own method to explore training SRL models in the absence of syntax, as we do, as well as other information like part-of-speech tags and morphological attributes, which we assume to be provided in the data.

6.2 Factor Graph Models of SRL

We now present a series of factor graph models for SRL. The first model, the baseline, incorporates no syntactic information, and treats argument/role prediction and sense prediction as entirely independent processes. To address some of the shortcomings of the baseline model, we describe extensions to capture valency (the number and type of arguments a predicate takes) and link argument prediction to sense prediction. We then present a joint model of dependency syntax and SRL, connecting these two models with logical factors.

6.2.1 Baseline Model: SRL without syntax

We present a simple edge-factored SRL model as a baseline. It is similar to our previous edge-factored models (5.2.2, pg.131) in that for each pair of indices $i, j \in n$ there is a Boolean variable representing the presence of a dependency, which is then labeled via a set of corresponding Boolean label variables (each representing one label from the set of L possible labels). In this context, the dependency is a predicate-argument pair from predicate i to argument j, and L is the set of possible semantic role labels. These are connected by a factor ensuring that at any predicted argument receives exactly one role label. Formally we define this model as follows:

- Let $\{Arg(i,j) : 1 \leq i \leq n; 1 \leq j \leq n\}$ be $O(n^2)$ Boolean variables such that $Arg(i,j) =$ true iff predicate i takes token j as an argument [3].
- Let $\{Role(i,j,\lambda) : \lambda \in L$, and $1 \leq i \leq n; 1 \leq j \leq n\}$ be $O(|L|n^2)$ Boolean variables such that $Role(i,j,\lambda)$ is true iff there is a semantic role of type λ between predicate i and argument j. This implies that $Arg(i,j)$ is also true, a property guaranteed by the following factor.
- Let IsAtMost1$(i,j) : 1 \leq i \leq n; 1 \leq j \leq n\}$ be $O(n^2)$ factors coordinating between $Arg(i,j)$ and all $Role(i,j,\lambda)$ label variables. This factor guarantees that iff $Arg(i,j)$ is true, exactly one $Role(i,j,\lambda)$ will be true. If $Arg(i,j)$ is false, all $Role(i,j,\lambda)$ are false.
- Let $\{Sense(i,\sigma) : 1 \leq i \leq n, \sigma \in S\}$, be $O(|S|n)$ Boolean variables, where S is the set of senses observed for the lemma of predicate i. $Sense(i,\sigma)$ is true iff predicate i has sense σ.

The Boolean $Arg(i,j)$ and $Role(i,j,\lambda)$ variables, and the multinomial $Sense(i,\sigma)$ variables can represent any SRL analysis. Each of these variables each have a corresponding unary factor, referred to as ARG(i,j), ROLE(i,j,λ), and SENSE(i,σ) respectively, which locally score the values of

[3] On a practical note, the SRL task is often defined as including the position of predicates in the data at both training and test time. This means that in practice there are a much smaller number of Arg variables than implied by the $O(n^2)$ maximum as described above.

the corresponding variable. When there are no other connections in the model, these are equivalent to a set of maximum entropy classifiers.

This baseline is similar to the system proposed by [147], which uses a maximum entropy model to classify whether or not there is a predicate-argument relationship for any pair of indices i, j. If an argument is present, it uses a separate maximum entropy model to predict its role label. Our baseline differs slightly, in that argument and role prediction are linked. If the features associated with a ROLE(i,j,λ) factor strongly indicate that variable $Role(i, j, \lambda)$ should be true, then these beliefs may influence the corresponding $Arg(i, j)$ during inference. This can have the effect of altering variable $Arg(i, j)$'s value to true, when its own local features would have otherwise classified it as false.

Few other systems rely solely on a similar maximum entropy or factor graph architecture, though maxent classifiers have been used in conjunction with other methods. [155] propose using maxent classifiers as a preliminary classification step to assist in pruning, filtering out unlikely analyses whose posterior probabilities are below a specified threshold. This reduces the search space for a more complex global model, which performs inference using an ILP solver. ILPs and other global inference techniques often outperform other approaches to SRL that make more independence assumptions, and we do not claim the baseline system is a state-of-the-art system. We now propose an extension to the baseline model which predicts the number of arguments and roles predicted for each predicate jointly, and and which couples role labeling to sense prediction.

6.2.2 Joint Approaches to Sense and Role Prediction

The baseline model proposed above has a number of limitations. There are no dependencies between pairs of $Arg(i, j)$ variables in the baseline model, nor are there dependencies between pairs of $Role(i, j, \lambda)$ variables. Each set of predicate-argument and role label predictions (variables which share the same j index) is independent from next. Similarly, the variables and factors for sense prediction are isolated from the rest of the graph. The lack of dependencies between these aspects of the model leads to many unfavorable outcomes:

1. The values of other argument variables (for the same predicate) cannot be considered by $Arg(i, j)$, and thus a predicate can be assigned many more or many less arguments than is likely. Many predicates take only a couple of arguments, and even analyses which feature dozens of arguments per predicate are not penalized by the baseline model.

2. The lack of dependencies between role variables gives rise to a similar problem, and a predicate may erroneously have multiple arguments with an idenitical role label. There are valid scenarios (for instance, coordination) in which a predicate might have two or more arguments assigned the same role label, but these are rare occurrences in our data. To illustrate where this would be the correct analysis, in the sentence "*Jack and Jill ran up the hill.*" both Jack and Jill perform the action of running and therefore have identical semantic role labels (AGENT).

3. The predicate sense determines the roles which are associated with that predicate. Naturally predicate sense and predicate roles should be considered jointly, and independent prediction of these variables can easily lead to mismatches. A sense which has only been observed

taking a single argument (for instance, an intransitive verb like *thrive* does not take a direct object), may erroneously take too many arguments, or in other cases, too few.

Both theoretical and empirical sources alike suggest that these decisions should be made jointly. From the theoretical side, Fillmore's theory of frame semantics strongly supports the need for joint inference by conceptualizing argument, role, and predicate sense as a single unit: a semantic frame[4]. According to frame semantics a predicate is associated with a particular set of roles "characterizing a small abstract 'scene' or 'situation', so that to understand the semantic structure of the verb it was necessary to understand the properties of such schematized scenes" [144, pg. 115].

From a more practical and empirical standpoint, the case for joint inference is illustrated by the performance of SRL systems in shared competitions. In the most recent shared task competitions, top performing systems have consistently relied on global inference strategies. A precursor to many current state-of-the-art SRL systems, [152] has shown that modeling dependencies between arguments can greatly improve performance over a comparable baseline consisting only of local predictions. This trend has continued with a variety of global inference strategies, including ILP [155], Markov Logic [153], and dual decomposition [161].

The intuition underlying why joint inference is useful for SRL is readily apparent, even in the simple example sentence from Fig. 6.1. In a joint model that couples sense and argument prediction, it is useful to know *how many* arguments the predicate sense will take. If there is a single argument, we may have strong evidence for the billiards sense of *scratched*, as an intransitive verb. If there is more than one argument, the predicate's sense is likely that of clawing an object, which is a transitive or ditransitive verb. In either case, the sense of the predicate determines how many arguments are likely, and vice versa. Similarly, the predicate sense determines which roles these arguments are likely to have.

A shortcoming of the baseline model is that it can easily overpredict a particular role, often creating predicates with multiple AGENT or PATIENT roles. This is only acceptable in limited circumstances, like coordination, and is not frequently represented in the CoNLL 2009 data. For instance, both men and cats are capable of scratching, with men tending to scratch objects like touchscreens, and cats having a natural inclination for scratching things in general. An SRL model trained on the right data (perhaps with a combination of gadget blogs and veterinarian logs) could reasonably predict that either is the AGENT of *scratched*. If two similar predicates appear in the same sentence, it is just as likely that *both* would be labeled as AGENT by the baseline model. In the following section we propose an extension to the baseline model which predicts argument, role, and sense variables jointly, and penalizes analyses where a particular role is uncharacteristically duplicated for the same predicate.

Modeling Valency with Accumulating Chains

Modeling predicate valency requires an additional set of factors beyond those previously discussed in this dissertation. The problem lies in how $Role(i, j, \lambda)$ variables have index-specific semantics, of the sort "*Does predicate i have an argument j with role label λ?*", when the question most relevant to sense prediction is "*Does predicate i have an argument (anywhere) with role λ?*",

[4]Frame semantics was preceded by Fillmore's case grammar [160], which perhaps even more directly addresses the concerns of valency. As it pertains to joint versus independent processes, both theories take similar positions.

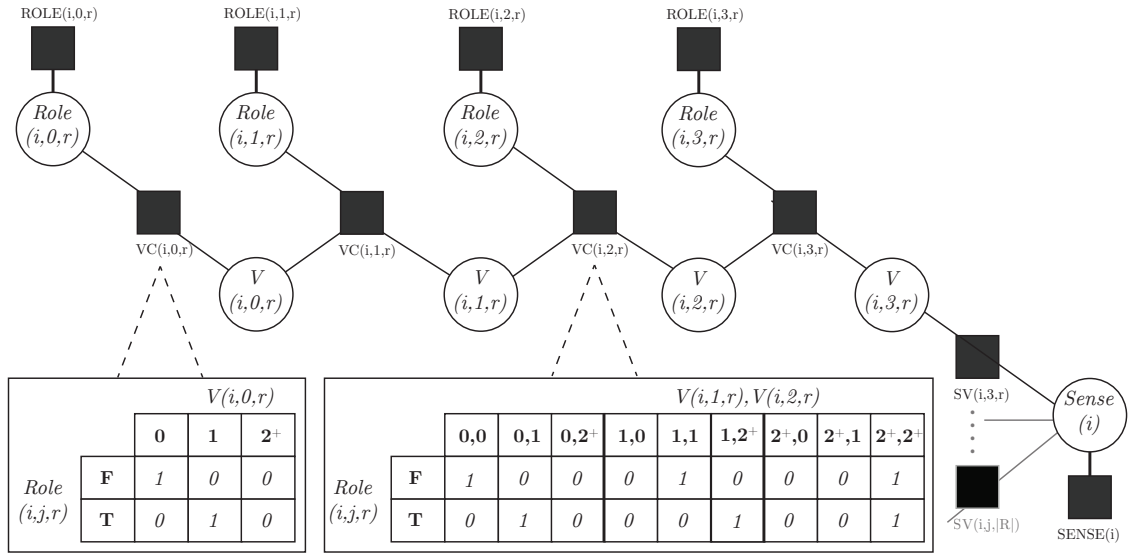


FIGURE 6.4: Modeling valency in SRL. Each V variable has three count states: no roles, 1 role, and more-than-1 role. Explicitly specified potential tables in R factors process the counting logic, rule out any invalid transitions between V variables. SV factors attach to the final V variable of each chain, allowing coordination between the set of observed roles and the different senses which might be in use.

and arguably, "*If so, how many?*". Our approach to defining variables with these semantics is to construct chains of $V(i, j, \lambda)$ variables, one chain for each predicate/role combination. Each $V(i, j, \lambda)$ variable acts as a counter over a limited number of bins (here, 0, 1, and > 1). At any point j, $V(i, j, \lambda)$ represents the number of times that $Role(i, k, \lambda)$ has been true, $0 < k \leq j$. The final variable in the chain, $V(i, j = n, \lambda)$, captures the intended semantics.

We implement this by deterministically defining the factors which connect chain variables, only allowing variable configurations which represent valid transitions from one chain variable to the next. It is straightforward to capture this logic by explicitly defining the potential table values for these factors, as illustrated in Fig. 6.2.2. To grasp the intuition, consider three variables, $V(i, j, r)$, $Role(i, j, r)$, and $V(i, j + 1, r)$, all connected by a factor $\mathbf{VC}(i, j, r)$. If $V(i, j, r)$ holds a count of 0, and $Role(i, j, r)$ is true, the count should be incremented to 1 in $V(i, j + 1, r)$. To accomplish this we set all potential table entries which correspond to invalid transitions to 0, and those which correspond to valid transitions to 1, enforcing the count logic as a hard constraint. This is equivalent to implementing a simple finite state machine, and the logic analogously extends to counting beyond 1. The final variable in the chain connects to the corresponding sense variable using the SENSEFRAME factors outlined below, allowing the count and type of each role label to influence sense prediction, and vice versa.

We argue for a 3-way binning for count variables, where the values of V variables correspond to a count state of 0, 1, or more-than-1 of a particular role at the current position in the chain. While few senses will have more than one of a particular role, having a bin for more than one occurrence creates competition between $Role$ variables. If it is highly likely that the predicate sense has just a single occurrence of role label r then this will be accounted for as beliefs propagate backwards through the chain, reducing the likelihood that more than one role variable will be true. We find

that a 3-way binning significantly outperforms a 2-bin model, which distinguishes only between seen and unseen (0 and > 0). These results are presented in Section 6.3.5 (pg. 153).

Formally we define the valency modeling structure as follows:

- Let $\{V(i,j,r) : 0 \leq i < j \leq n, r \in R\}$ be $O(|R|n^2)$ ternary variables, where $|R|$ is the number of role labels, forming $|R|$ chains. These variables capture the number of predicate-argument pairs of role r between pair $(i,0)$ and (i,j), using three bins: 0, 1, and more than 1. While there are few instances where a predicate will have multiple arguments with the same role label, using a bin of 3 or more helps create competition between role variables for the more typical event where a particular role occurs only once for each predicate.

- Let $\text{VC}(i,j,r) : 0 \leq i < j \leq n, r \in R\}$ be $O(|R|n^2)$ factors that deterministically implement the logic for calculating the values of V variables. These factors are binary at the start of the chain, and ternary elsewhere. The potential tables of these factors are described in Fig. 6.2.2.

- Let $\text{SENSE-VALENCY}(i,r) : 0 \leq i \leq n, r \in R\}$ be $O(|S||R|n)$ binary factors which connect the final V variable of each chain, $V(i,n,r)$, to the $Sense(i)$ variable for each predicate i. The features of these factors capture correlations between predicate senses and the number of particular roles associated with each.

Other models have also implemented forms of valency modeling, most notably the generative models of constituent parsing [84, 85, 162] and dependency parsing [140]. Valency information has also been utilized in discriminative models as part of the conditioning context in transition-based parsing [163], and in ILP models of the same task [164]. However we are unaware of previous work modeling valency in factor graph models.

6.2.3 A Joint Model of Dependency Parsing and SRL

Considering the limited attention that joint models have received in recent years, a comparably large number of approaches exist for joint parsing and semantic role labeling. There is no doubt that this is due to the CoNLL 2008/9 shared tasks, which focused on joint parsing and semantic role labeling and featured many participating systems. The best joint results were attained with graph-based approaches for the synchronous derivation of both structures [165], and later explored with cube-pruning [166]. ILP models and Markov Logic [153] also performed well in the competition. More recently, dual decomposition approaches have improved upon these results for English data [161] and Chinese data [167].

Our approach to model combination relies on logical factors in a joint factor graph model, and belief propagation for inference. Given the modeling structure we have outlined in previous chapters, it is a straightforward extension to construct a fully joint model of SRL and dependency parsing. We first assume the presence of a dependency syntax representation, identical to the first-order model presented in Section 3.2.2 (pg. 99, including a projective DEP-TREE factor). We coordinate this structure and the SRL model through the use of an additional set of factors:

- Let $\{\text{SRL-CONNECT}(i,j) : 1 \leq i \leq n; 1 \leq j \leq n\}$ be $O(n^2)$ soft NAND factors coordinating a single dependency parse edge variable, $Link(i,j)$, with a single predicate-argument

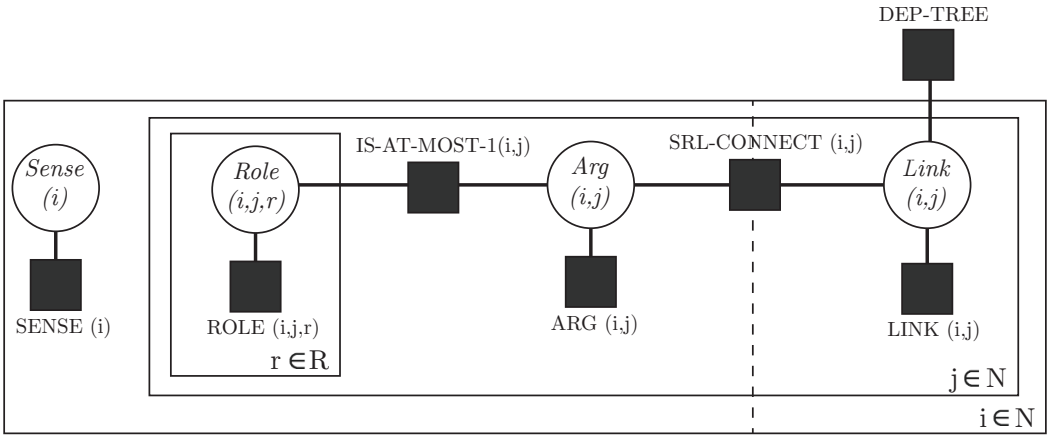


FIGURE 6.5: A joint model of SRL and dependency syntax. Depicted as a plate diagram, the baseline SRL model (*left*) connects to the combinatorially-constrained dependency parse model (*right*) via a series of SRL-CONNECT factors. Valency modeling is omitted here, and senses are predicted independently.

variable, $Arg(j, i)$. This factor is associated with a set of features, and learns a potential to determine the degree of coordination dependent on the context.

A graphical depiction of this model is shown in Figure 6.5.

This method for coordinating the models is essentially identical to the approach used in Chapter 4 to coordinate constituent spans and named entity spans. Unlike NER, SRL does not require a combinatorial constraint. Therefore, unlike the joint NER model, there are no cycles in the graph (i.e., the graph for SRL and joint SRL form a tree) and inference is exact. This is not the case for any model with the valency modeling extensions.

6.3 Experiments

6.3.1 Data

We evaluate our SRL models using a subset of the data developed for the CoNLL 2009 shared task competition for joint semantic and syntactic dependency prediction [139], which consists of data from seven languages: Catalan, Chinese, Czech, English, German, Japanese, and Spanish. This diversity in language provides a unique opportunity to examine the type and effect of the induced syntactic structure across a wide variety of different languages and annotation styles.

The data provides word, lemma, part-of-speech, and morphological feature information, along with both gold and parser-produced dependency trees. Predicates are identified, but the particular sense is intended to be predicted by the model. Sense labels are provided in the training data, along with their arguments and role labels.

6.3.2 Features

We now describe the feature set associated with ARG, ROLE, SENSE, LINK, SRL-CONNECT, and, in the case of valency models, SV factors.

- ARG and ROLE factors

  For the ARG and ROLE factors, which aim to capture arc-factored information, we use an identical feature set. These features are described below for the example sentence "*the cat scratched the man*", $w_i =$ "*scratched*", $w_j =$ "*cat*".

Feature String	Comment
arg-bias	A general bias for a pred-arg pair.
scratch-cat	Predicate word and argument word pair.
VB-NN	Predicate POS-tag and argument POS-tag.
scratch-NN	Predicate word and argument POS-tag.
VB-cat	Predicate POS-tag and argument word.
VB-scratch-NN-cat	Predicate and argument words and POS-tags.
VB-NN-1	Predicate POS-tag, argument POS-tag, and distance ($\|j - i\|$).
VB-NN-LEFT	Predicate POS-tag, argument POS-tag, and direction.
VB-LEFT-1	Predicate POS-tag, direction and distance to argument.
NN-LEFT-1	Argument POS-tag, direction and distance to argument.

  For labeling, the role label is concatenated onto each feature so that otherwise identical features correspond to different parameters in the model (i.e., *scratch-cat*, a feature used in the ARG(i, j) factor, is used as a basis for *PATIENT-scratch-cat*, a feature used in the ROLE$(i, j, r =$ *PATIENT*$)$ factor.) In other words, there is no feature sharing between any pair of the argument, role, sense, valency, syntax, or connection sets of factors.

- SENSE factors.

  Sense labeling is done with the following set of features:

Feature	Comment
scratched	Sense word.
scratch	Sense lemma.
VB	Sense POS-tag.
cat	Neighboring word. Each word within a 5-word window.
cat-1, the+1	Contextual word with offset. Each word within a 3-word window.
cat-scratched-1	Sense word and previous word. Chinese only.
the-scratched+1	Sense word and following word. Chinese only.

  Each feature string will also be concatenated with the sense label (i.e., "VB" becomes "VB-scratched.01", "VB-scratched.02", etc.) to produce a new feature for each sense.

- SENSE-VALENCY factors.

  The valency variables store the total count for each predicate and role (the final variables in each chain). They are connected with the sense prediction variables through SENSE-VALENCY factors, using the following features. Here we illustrate features for just one sense, one role, and one count value:

Feature	Comment
scratched.01-bin-ARG0-1	A predicate sense with a role and its count.
scratched.01-bin-ARG0-1-scratched	Same, but including word form.
scratched.01-bin-ARG0-1-scratch	Same, but including word lemma.
scratched.01-bin-ARG0-1-VB	Same, but including word POS tag.
bin-ARG0-1	The role and its count.
bin-ARG0-1-scratched	Role and count with word form.
bin-ARG0-1-scratch	Role and count with word lemma.
bin-ARG0-1-VB	Role and count with word POS tag.

The first four features are intended to help model the interactions between senses and role counts. The latter four features are backoffs, intended to help strictly with influencing the counts of each role and penalizing them when their values are higher or lower than expected.

- SRL-CONNECT factors

  Features for coordinating between SRL and dependency syntax are the following:

Feature	Comment
sli-bias	A bias for the coordination factor.
sli-scratched-cat	Predicate word, argument word.
sli-VB-NN	Predicate POS-tag, argument POS-tag.
sli-scratched-NN	Predicate word, argument POS-tag.
sli-VB-cat	Predicate POS-tag, argument word.
sli-VB-scratched-NN-cat	Predicate and argument words and POS-tags.
sli-LEFT	Attachment direction.
sli-1	Attachment distance.
sli-LEFT-1	Attachment distance and direction.
sli-scratched-cat-LEFT-1	Predicate and argument words, distance and direction.
sli-VB-NN-LEFT-1	Predicate and argument POS-tags, distance and direction.
sli-VB-NN	Predicate and argument POS-tags, distance.
sli-VB-LEFT-1	Predicate POS-tag, distance and direction.
sli-NN-LEFT-1	Argument POS-tag, distance and direction.
sli-VB-NN-LEFT	Predicate and argument POS-tags, direction.

  Because these features are orthographically similar to those used in other factors, the string representation of each connection feature includes an "sli" prefix to indicate it belongs to this feature set.

  The features for dependency syntax (LINK factors) are a subset of the McDonald feature set for supervised graph-based dependency parsing [69], omitting features that include coarse part-of-speech (as these are not provided in the data sets). Even though the SRL data contains a great deal of morphological information for some languages, we were not able to make use of these attributes in any manner that improved performance on development data. We tried incorporating morphological classes into sense, argument and role features, essentially creating new features where part-of-speech tags were replaced by morphological classes, but this often decreased SRL prediction accuracy.

6.3.3 Experimental Design

We perform a set of experiments to compare the latent syntax model to comparable models which instead rely on gold syntactic parses, and to models which do not observe parses. We refer to these four models as BASELINE, ORACLE, VALENCY, and HIDDEN. Due to the increased complexity of the model, valency modeling is used only as an extension to the baseline and not in conjunction with syntactic joint models.

- BASELINE: A model which contains no syntactic information, and is strictly argument, role, and sense prediction, as described in Section 6.2.1 (pg. 143). This model is intended to represent the best achievable performance from fully-supervised training on a resource-poor language where syntactic annotations are unavailable.

- ORACLE: A joint model (pg.147) with both SRL and syntax variables. Syntactic variables are set to their observed values based on the gold standard trees provided by the data, both during training and testing. Features are still learned for the SRL-CONNECT factors which sit at the intersection of the two models, giving the model the flexibility to utilize or ignore the syntactic information depending on the immediate context as captured by the coordinating feature set.

- HIDDEN: A joint model which treats all syntactic components as latent variables. Syntactic annotations are not used, and the true labels of SRL-CONNECT factors are not known. Instead this model is trained using the marginalization-based training in order to induce an latent syntactic representation specific to the task of SRL and the particular training data set.

- VALENCY: Valency modeling (pg.145), which connects role and sense prediction, and counts the number of times each role is taken by a given predicate. The 2-bin model count distinguishes only between 0 and greater than 0. The 3-bin model counts 0, 1, and greater than 1.

We include results from two additional systems from the CoNLL 2009 shared task:

- Che-09: The system described in [155] is the highest performing system in the joint track of the CoNLL 2009 shared task [5], . In the joint task no syntactic annotations are provided for the test data, making it comparable to the JOINT or HIDDEN models. [155] present a heavily pipelined system. Sentences are parsed using a 2nd-order graph based parser [69], with pseudo-projective extensions for the more non-projective languages: Czech, German, and English. An SVM is used to predict predicate senses. A log-linear classifier is used to predict a candidate set of semantic role dependencies, and ILP is used to for global inference over this candidate set. The feature set is carefully configured for each language, and different depending on whether the SRL predicate is a noun or verb.

- Zhao-09: The system described in [156] is the highest performing system in the SRL-only track of the CoNLL 2009 shared task. In the SRL-only task syntactic annotations are provided for the test data, making it comparable to the ORACLE model. [156] is self-described

[5]Technically [168] is the highest performing system in terms of SRL performance, but it is largely identical to ZHAO-09 (and was submitted by the same team), and produces results in line with [155].

as a "huge feature engineering method" for SRL, and the novelty of the approach lies in the automatic feature discovery method. An SRL analysis consists of many predictions: sense prediction, predicate-argument prediction, and role labeling, each requiring a set of features. The huge space of potential features motivates systems like [155] and [156] which focus more on feature engineering than on new modeling approaches. For predictions [156] utilizes a 2nd-order parser, and makes SRL predictions using log-linear classifiers.

All models are trained using 30 iterations of stochastic gradient descent (SGD) with a learning rate $\eta = 0.01$ and L2 regularization (a Gaussian prior with a variance of 1.0). We were limited by the available computation power in making this decision: 30 iterations of SGD is roughly the maximum number of training iterations computable in a week when training the most complex of these models on the largest data sets. It is conceivable that additional training iterations would lead to higher performance than is presented here. Inference is exact in most cases, but the valency models do contain cycles and in these instances we use a maximum of 40 iterations of belief propagation for each instance. The models are pruned, and only consider arguments that are within 20 words of each predicate. A dictionary is also constructed containing all sense labels seen in the training data, indexed by each predicate's lemma. For unknown predicates it is common to use a default sense label, which is typically the lemma with an ".01" suffix (the most frequent sense label in the data).

We implement feature pruning for the fully-supervised models, removing all features which are not associated with a single correct instance in the training data. For the smallest data set, German, there are 12M features for the BASELINE model, 14M for the VALENCY model, 15M for the ORACLE model, and 72M for the HIDDEN model. Using 4 cores on a 2.6 GHz Intel Xeon processor, the average amount of time required for performing one iteration of SGD on the German data is 2 minutes for the BASELINE model, 2 minutes for the ORACLE model, 12 minutes for the VALENCY model, and 31 minutes for the HIDDEN model. On the largest data set, Chinese, a single iteration of SGD requires on average 2 hours for the BASELINE model (120M features), 2.5 hours for the ORACLE model (145M features), 9 hours for the VALENCY model (185M features), and 14.5 hours for the HIDDEN model (885M features). This requires 6GB of RAM to train on the German data, and 24GB of RAM to train on the Chinese data.

As in previous chapters, models which require syntactic annotations during training are marked with a † symbol, and models which rely on syntactic annotations during both training and testing are marked with a †† symbol.

6.3.4 Evaluation

Models are evaluated in three different ways, as performed in the CoNLL 2009 shared task competition:

1. Sense prediction accuracy (SA). Each predicate has a predicate sense, and is evaluated using accuracy:
$$SA = \frac{true\_senses}{\#\_of\_predicates}$$
2. Unlabeled evaluation. We report precision, recall and F1 (harmonic mean of precision and recall) on classifying predictate-argument pairs. The unorthodox aspect of this evaluation is

that the number of predicates is included in the calculation. For instance, precision of the unlabeled case is calculated as follows:

$$Precision_{Unlabeled} = \frac{\#\_of\_predicates + true\_predicate\_argument\_pairs}{\#\_of\_predicates + test\_predicate\_argument\_pairs}$$

3. Labeled evaluation. Precision, recall, and F1 for predicting semantic roles (where a semantic role is a 3-tuple comprising the predicate, argument, and label). Sense accuracy is included in this metric. For instance, precision in the labeled case is calculated as follows:

$$Precision_{Labeled} = \frac{true\_senses + true\_roles}{\#\_of\_predicates + test\_roles}$$

Labeled F1 is the score used in the CoNLL 2009 shared task competition for the purpose of ranking competing systems. Table 6.1 provides a set of summary statistics from the competition, including the maximum, mean, and median results for each language.

6.3.5 Results

Here we present the results of our model configurations with respect to the systems entered in the CoNLL 2009 shared task [139]. Not surprisingly, the BASELINE model, with its independent predictions and lack of syntactic information, performs poorly. The inclusion of reliable syntactic information in the ORACLE model leads to performance that is generally comparable with the average CoNLL scores, and competitive with other joint models. Though the connections between SRL and syntax are based on first-order (parent-child) relationships, the average scores compare favorably with more powerful second-order approaches [169]. Despite their overall simplicity, these models still manage to perform at state-of-the-art levels in a few instances, most notably on Chinese. Here, even without observing any syntactic annotation, the HIDDEN model manages to outperform nearly all of the competing systems.

The most compelling results come from comparing ORACLE and HIDDEN models with the BASELINE. Here the unlabeled results are the most revealing evidence for assessing the effectiveness of hidden syntactic structure. Abstracting away from the performance comparisons against other systems, the average unlabeled HIDDEN model score (86.22% F1) is even higher than the ORACLE model (83.15% F1, and vs. BASELINE's 67.72% F1). This is mostly due to large margins over the ORACLE system on the unlabeled Japanese scores. The strong independence between sense prediction and argument prediction hinders performance on the labeled task, but on all languages we find an extremely significant improvement from the use of hidden syntactic structure — the HIDDEN model recovers 94.74% of the gap between the baseline and the observed syntax model.

There is one caveat to bear in mind when interpreting HIDDEN vs. ORACLE results, and that lies in how the SRL and dependency parsing models are connected. We have discussed the degree of agreement between SRL predicate-argument dependencies and corresponding dependency tree paths of varying lengths (Fig. 6.1.1), showing that for most languages the largest correlation exists when looking at paths of length 1. Our ORACLE model takes advantage of this correlation, as SRL-CONNECT factors link single predicate-argument to single dependency parent-child arcs. It

Data	Model	SA	Unlabeled P	Unlabeled R	Unlabeled F1	Labeled P	Labeled R	Labeled F1	CoNLL 2009 F1 MAX.	CoNLL 2009 F1 MEAN	CoNLL 2009 F1 MED.
Catalan	BASELINE	84.3	94.7	77.5	85.3	77.8	63.7	70.0			
	ORACLE††	84.3	98.5	96.2	**97.3**	75.7	73.4	74.5			
	ZHAO-09††	83.4	96.6	90.7	93.6	82.9	77.9	**80.3**	80.3	71.0	74.1
	HIDDEN	82.7	95.2	92.8	94.0	73.2	70.8	72.0			
	CHE-09†	80.1	94.6	91.1	92.8	78.6	75.7	77.1			
Chinese	BASELINE	94.8	72.5	64.8	68.4	66.0	59.0	62.3			
	ORACLE††	94.8	98.6	78.9	**87.7**	87.6	70.2	**78.0**			
	ZHAO-09††	94.7	86.8	81.2	83.9	80.4	75.2	77.7	78.6	72.2	70.4
	HIDDEN	94.8	90.8	79.1	84.5	81.97	71.4	76.3			
	CHE-09†	94.8	88.9	79.2	83.6	81.7	73.1	77.2			
English	BASELINE	90.1	92.5	71.6	80.7	84.6	65.5	73.8			
	ORACLE††	90.2	96.7	82.3	88.9	85.5	72.7	78.6			
	ZHAO-09††	91.3	94.3	91.3	88.5	88.2	82.8	85.4	85.6	75.6	72.1
	HIDDEN	89.9	95.1	79.1	86.3	83.8	69.7	76.1			
	CHE-09†	91.4	92.9	90.5	**91.7**	86.7	84.4	**85.5**			
German	BASELINE	82.6	93.3	62.9	75.1	79.1	53.3	63.7			
	ORACLE††	82.6	96.4	79.8	87.3	80.4	66.6	72.9			
	ZHAO-09††	78.4	92.3	89.4	90.8	77.2	74.8	76.0	79.7	68.1	67.8
	HIDDEN	82.1	93.2	88.1	90.5	77.0	72.8	74.8			
	CHE-09†	82.4	95.2	87.5	**91.2**	82.1	75.4	**78.6**			
Japanese	BASELINE	100	93.9	62.8	75.3	85.7	57.3	68.7			
	ORACLE††	100	94.4	62.4	75.1	86.1	57.0	68.6			
	ZHAO-09††	99.1	87.6	80.4	**83.8**	81.7	74.9	**79.2**	80.5	70.4	73.4
	HIDDEN	100	90.6	75.7	82.5	78.3	65.4	71.3			
	CHE-09†	99.1	93.2	75.8	83.6	87.3	71.0	78.3			

TABLE 6.1: SRL Results on CoNLL 2009 data sets. Without observing any syntactic annotation the HIDDEN model excels on the unlabeled prediction results, often improving on scores obtained with the ORACLE model. Both the ORACLE model and the systems represented in the CoNLL column could use gold standard parses.

is clear, from the same statistics, that this does not hold true for all languages, and Japanese in particular has a stronger correspondence between predicate-argument pairs and dependency tree paths of length 2. It could be argued that using method of model connection based on first-order dependencies on this data puts the ORACLE model at an inherent disadvantage.

However, determining which types of syntactic dependencies are most useful to SRL is a responsibility that should fall on the model, not the modeler. Higher order models are more robust to the syntactic variations we see here, but it is not always feasible to implement models of increasingly higher order simply to suit the idiosyncrasies of each data set. There are also clear benefits to learning which first-order syntactic representations are most useful. Compared to models with higher-order syntactic representations, the first-order latent syntax models are smaller in terms of the number of factors necessary to represent them. They do not contain cycles, which allows for more efficient exact inference. The disadvantage of this model is the greater cost of training, as the marginalization-based training requires two sets of marginals and thus two rounds of inference.

Data	Model	SA	Unlabeled			Labeled		
			P	R	F1	P	R	F1
Catalan	BASELINE	84.3	94.7	77.5	85.3	77.6	63.7	70.0
	VALENCY 2-BIN	82.3	92.3	80.0	85.7	77.4	67.6	72.7
	VALENCY 3-BIN	82.2	93.7	83.4	**88.3**	77.8	68.5	**72.9**
Chinese	BASELINE	94.8	72.5	64.8	68.4	66.0	59.0	62.3
	VALENCY 2-BIN	94.4	73.2	71.1	72.1	69.8	64.5	67.1
	VALENCY 3-BIN	94.5	74.8	75.2	**75.0**	71.9	66.3	**69.0**
English	BASELINE	90.1	92.5	71.6	80.7	84.6	65.5	73.8
	VALENCY 2-BIN	89.1	92.3	71.2	80.4	84.6	65.1	73.6
	VALENCY 3-BIN	88.6	92.5	73.4	**81.9**	84.8	67.6	**75.3**
German	BASELINE	82.6	93.3	62.9	75.1	79.1	53.3	63.7
	VALENCY 2-BIN	79.7	92.2	65.1	76.3	75.2	54.8	63.4
	VALENCY 3-BIN	79.5	91.4	70.4	**79.6**	74.9	57.7	**65.2**
Japanese	BASELINE	100	93.9	62.8	75.3	85.7	57.3	68.7
	VALENCY 2-BIN	100	92.1	61.7	73.9	85.5	58.7	69.6
	VALENCY 3-BIN	100	92.1	64.3	**75.7**	85.0	60.1	**70.4**

TABLE 6.2: SRL Valency Model Results. The 2-bin model (counting 0 or more-than-1) improves average labeled accuracy by 1.58 F1 over the baseline. The 3-bin model (counting 0, 1, or more-than-1) improves accuracy even further, a 2.86 F1 average improvement over the baseline.

6.3.6 Valency Results

Modeling valency structure makes all argument predictions for a given predicate dependent on one another, and allows argument and role prediction to affect sense prediction (and vice versa). Figure 6.2 shows the effect of including this structure in the baseline model. In the 2-bin model, the variables in the valency chains have two states, and expressing whether a particular role has or has not been observed in the chain. The 3-bin model extends this to an extra state, to represent having seen 0, 1, or more than 1 occurrence of a particular role. The intuition is that if a predicate has one sense which prefers one argument (i.e., an intransitive verb), and another prefers two (a transitive verb), this phenomenon would be better captured by the 3-bin valency model.

In contrast with the baseline, modeling valency nets significant improvements in both labeled and unlabeled F1 across all languages. The 2-bin model provides a 0.72% F1 gain in average unlabeled accuracy, and 1.58% F1 in labeled accuracy. The 3-bin model improves performance by 3.14% F1 in the unlabeled case and 2.86% F1 in the labeled case, in some cases closely approximating the gains we achieved previously through modeling syntactic structure.

Interestingly these additions, which were originally intended to improve sense accuracy, are unhelpful to this aspect of the model. The reason for this is uncertain but in preliminary experiments we find that the valency models are particularly sensitive to feature sets, especially those used for coordinating between the $Sense(i, r)$ and $V(i, j = n, r)$ variables. It is possible that different features or a different method of connecting these variables would remedy this issue.

Additionally, we do not know if the improvements from valency modeling are complimentary to the improvements from syntactic modeling, and whether combining these models would result in still greater accuracy. The reason we do not pursue this here is the greater complexity of these

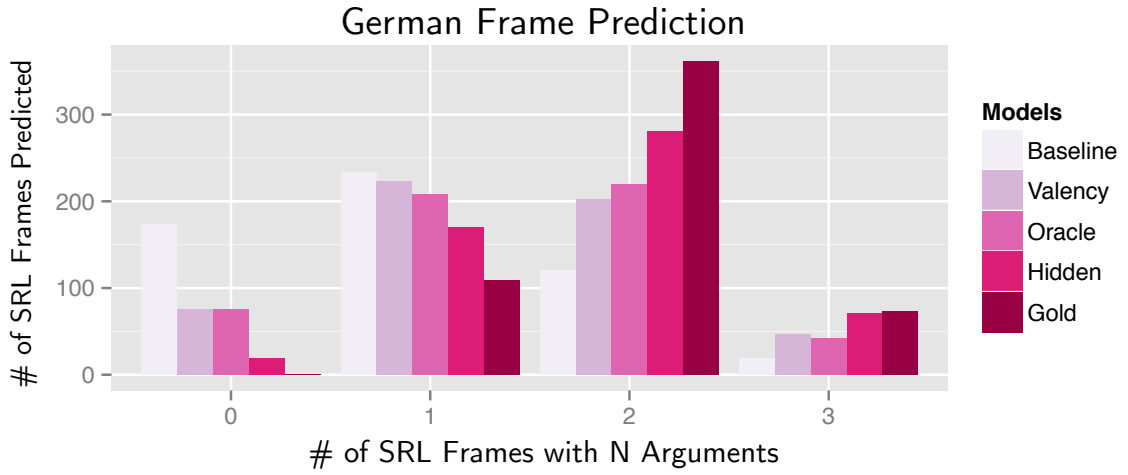


FIGURE 6.6: SRL frame arity across models. Statistics were collected from experiments on the German CoNLL 2009 data set. In terms of the number of arguments they contain, the SRL frames predicted by the HIDDEN more closely reflect the frames of the training data (*Gold*).

models, and its effect on inference and training. The model which contains both valency and syntactic variables contains many cycles, and often takes more than 20 iterations of inference to reach convergence. This results in very long training times, and we do not have the computationally resources to explore the effectiveness of these models on large CoNLL 2009 data sets. Future work could explore these issues, or pursue additional pruning techniques to limit the number of valency chains and the resulting cycles.

6.3.7 Performance on SRL Frames

We have discussed the performance of the various models in terms of their accuracy in making pairwise predictions, evaluating models in terms of their ability to correctly identify predicate-argument pairs and the corresponding role label, as is standard in the CoNLL shared tasks. We now examine the performance of these models from a different angle, assessing their ability to accurately identify the correct number of arguments for each predicate, sometimes referred to as an SRL frame.

The plot in Fig. 6.6 depicts the number of frames with n arguments predicted by each model, where n varies between 0 and 4, for experiments on the German data set. Very few predicates in the data lack arguments altogether ($n = 0$), but there is no explicit constraint, either in the model structure or in the decoding procedure, to prohibit our SRL system from predicting SRL frames without arguments. This is where the lack of syntax hinders the model: without any syntactic information to aid the BASELINE model, it must rely solely on lexical information between the predicate and candidate word. The results indicate that in many cases the model is not confident enough to predict any predicate-argument pair, severely reducing the model's performance.

When $n > 0$, the frame predictions of the HIDDEN model are most closely correlated to the GOLD model (170 vs. 109 for one argument, 281 vs. 362 for two arguments, 71 vs. 73 for three arguments, respectively). Interestingly, for $n > 0$, the ranking of how similar a model's frame

prediction matches the gold frame prediction is constant, from best to worst: HIDDEN, ORACLE, VALENCY (denoting the 3-bin system), and BASELINE. The inclusion of valency modeling offers a significant improvement over the BASELINE reducing the margin from the number of gold predictions by 99 ($n = 0$), 11 ($n = 1$), 82 ($n = 2$), but does not outperform the syntactic models. We conclude that syntactic information, even when induced, is a more useful source of information than what is offered by the valency chain modeling.

An Analysis of Induced Syntactic Structure

Exactly what kinds of structure does the HIDDEN model learn and why do the ORACLE models perform poorly on Japanese and German data? Before we dive into this question in detail, it is interesting to note that the unusually low performance of the ORACLE joint model on these data sets is not without precedent: in the CoNLL 2009 shared task competition, results on the German and Japanese data sets consistently had the highest gaps between parsing F1 and SRL labeled F1. Systems produced accurate parses, but were not able to leverage the strength of their parsing models to produce accurate SRL analyses. We conclude through an analysis of our own results that the style of syntactic annotation is a poor match for joint models with first-order connections, and that this may have contributed to the poor performance of SRL prediction relative to syntactic prediction.

Japanese

Despite having been trained without any access to gold syntactic annotation, the HIDDEN system outperforms both the BASELINE and ORACLE models by significant margins (2.6% F1 and 2.7% F1 respectively). In previous work, which differed only in feature sets and training method, the numbers reflected a greater divide between baselines and syntax, with the HIDDEN model providing a 13.99% and 4.57% improvement in labeled F1 [170]. What type of syntactic structure does this model induce to be preferable even to hand-annotated syntax? We examine this structure by constructing a heatmap (Fig. 6.7, a gloss for this example is provided in Fig. 6.8). A heatmap is capable of illustrating the correlation between the real-valued syntactic dependency variable beliefs and the discrete-valued (true or false) syntactic arcs found in the gold standard data. Shaded cells indicate the model's beliefs regarding each $Link(i, j)$ variable, i on the x-axis and j on the y-axis. A red square indicates a belief closer to 1.0 (true), and a solid grey square to a belief closer to 0.0. The correct SRL analyses are depicted by black box borders. Gold syntax is shown on the left, the latent syntax of the HIDDEN model on the right.

In addition, tree diagrams (*top*) provide an alternative visualization, showing the decoded syntactic dependencies above the sentence with predicted SRL dependencies below (*red*). The dependency tree for the HIDDEN model is produced by classifying all dependency variables whose beliefs are greater than 0.5 as true. This is purely for illustrative purposes. In comparison to the minimum Bayes risk decoding used in other chapters, this decoding strategy more accurately reflects the beliefs of the model but does not prevent cycles or non-projective dependencies.

In this example there are four SRL predicate-argument relations specified in the gold data. Our model uses first-order connections between SRL and syntax, and therefore the model coordinates between an SRL relation from predicate i to argument j and a syntactic dependency from headword

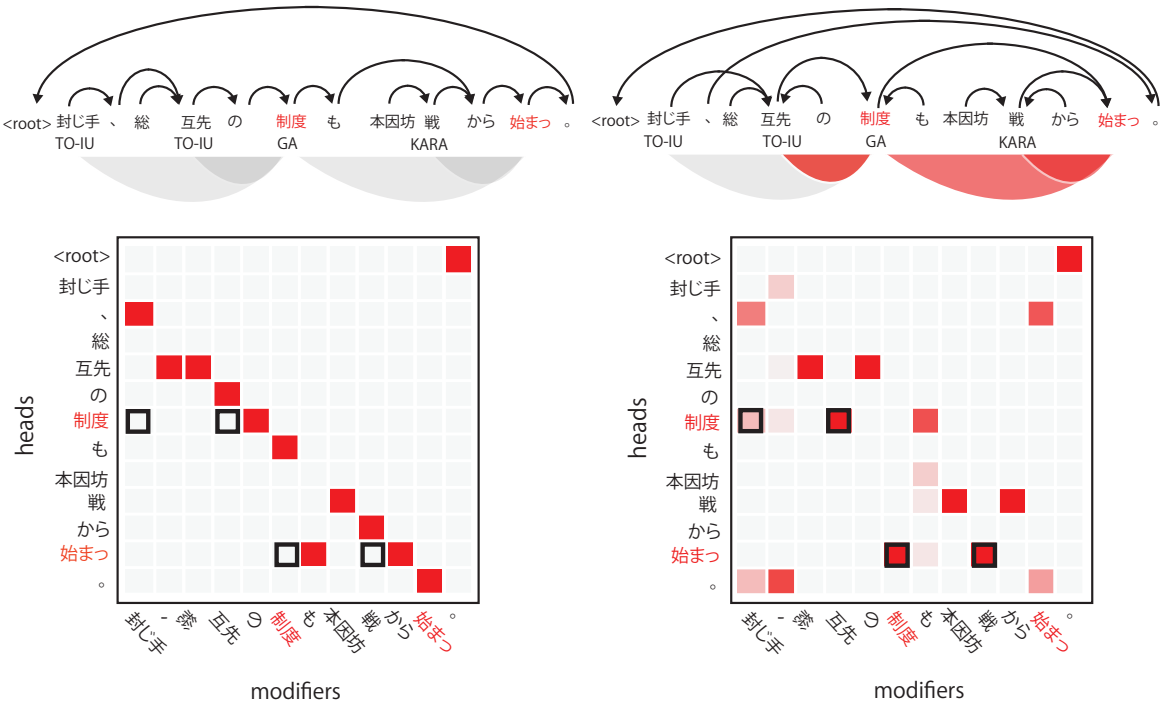


FIGURE 6.7: Examining the induced dependency syntax for SRL. In this Japanese example the syntactic dependency arcs derived from gold standard syntactic annotations are entirely disjoint from the correct predicate/arguments pairs (*left*, SRL relations depicted by black cell outlines, probability of syntactic relations depicted by strength of red cells). In contrast, the hidden model structure (*right*) learns a representation that closely parallels the desired end task predictions, helping it recover three of the four correct SRL predictions.

i to child word j. None of these SRL relations has a corresponding syntactic arc in the gold syntax, and, given this low degree of correlation corpus-wide, this provides an explanation for low performance of the ORACLE model on Japanese data. Due to this fact it is also unsurprising that none of the correct SRL dependencies are predicted by the ORACLE model in this example.

We now examine why the Japanese data lacks this correlation between syntax and SRL. While none of the SRL relations in this example have a directly corresponding syntactic arc, two have a corresponding syntactic *path* consisting of two dependency arcs, which together link the predicate word to the argument word. A third relation has a corresponding path of three arcs, and the last a path of four arcs. Why are words which are directly linked by a SRL relation separated by so many arcs in the syntactic structure? We touched on the primary reason for this when discussing the correlation statistics in the data (Sec. 6.1.1, pg. 139): in Japanese particles intercede in the dependency path, while SRL relations directly link root words.

For instance, the case-marking particle の functions here as a adjectival modifier for the preceding word. In this example it intercedes between 総互先 (*even match*) and 制度 (*system*), to describe *even match* as a kind of *system*. This is also the case for the binding particle, も . In short, particles in this data act as the syntactic heads of the words they modify. [6] This places

[6]This syntactic analysis may seem objectively incorrect, given the usual head-modifier semantics of dependency trees. However, by marking the gramamtical case of the headword, these particles provide important information about the headword's syntactic role in the sentence. It is therefore reasonable to annotate particles as syntactic heads,

封じ手	総 互先	の	制度	も	本因坊	戦	から	始まっ
sealed move	even match	(case-marking particle)	system	(binding particle)	Honinbou	tournament	(case-marking particle)	began

The sealed move, as well as the all-even-match system, started with the Honinbou Tournament.

FIGURE 6.8: English gloss for Japanese example in Fig. 6.7.

them between the predicate and argument words in the syntactic dependency path, increasing the dependency parse length. In contrast, in the English CoNLL data a similar modifier would attach to the word it modifies, allowing the predicate and argument words to link more directly. This is the syntactic analysis induced by the HIDDEN model for the Japanese data.

But particles are not the only tokens that hinder the correlation between syntax and SRL. Punctuation can also intervene in these dependency paths. For instance, a comma intervenes in the dependency path between 封じ手 (*sealed move*) and 制度 (*system*), extending the dependency path from three words to four. Again, punctuation could just as easily function as a modifier to the head of the clause, in which case it would not extend the dependency path. The HIDDEN model proposes an interesting solution to punctuation: it consistently anchors punctuation to the last token of the sentence, which, in this data set, is always itself a punctuation character. This prevents it from interfering with any syntactic path from predicate to argument.

It may be possible to address these issues by preprocessing the data and heuristically modifying the syntax to better suit the model. One could attempt to adjust the syntactic role of Japanese particles, setting the preceding root word as the syntactic head and therefore removing the particle from the predicate-argument dependency path. The preprocessing tactic may provide a simpler method for improving joint model performance than inducing a latent syntax, but this may be somewhat of a slippery slope. Performing such preprocessing would solve the correlation problem for two of the four spans, but even if we were to remove punctuation (reducing the dependency path of a third SRL relation) it would still be necessary to use second-order connection factors to capture the relationship between SRL and syntax in the data.

Ultimately the question of which approach is preferable may be an ideological one: is it the responsibility of the researcher to ensure that the data is tokenized or parsed in the manner which best suits the end task (and how does one determine this?), or is it the responsibility of the model and the optimization method to circumvent poor agreement between two sources of information. We argue for the latter: the marginalization-based may be slower than fully-supervised training, and inference in the joint model may be more costly than in the baseline, but utilizing this approach we are able to run our model across a wide variety of languages (and domains), with little to no language-specific treatment. This reduces the number of hours a human must spend preprocessing the data, feature engineering, and modifying the syntactic annotations to suit the model.

German

In the Japanese corpus the syntactic annotation style reduces the degree of first-order correlation between syntax and semantics, but in this case the syntactic trees are *intentionally* annotated, by

while the semantic dependencies of the SRL analysis generally ignore them.

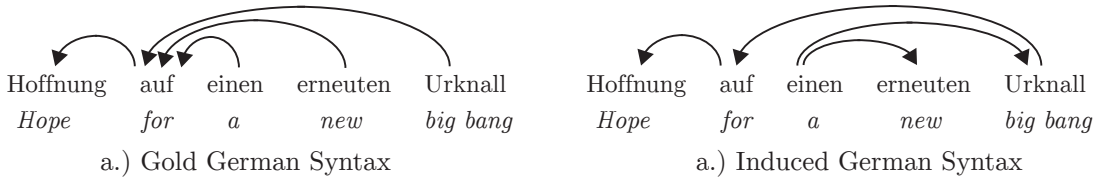


FIGURE 6.9: German NP structure. The gold syntax NP is completely flat, but the HIDDEN SRL model induces a richer NP structure.

hand, to conform to a set of annotation guidelines. But the correlation between syntax and semantics can also be skewed by automated pre-processing and conversion. We find that this is prevalent in the NP structures in the German data set.

The Tiger Corpus [171] is a German corpus comprising 50,000 sentences of German newswire from the Frankfurter Rundschau, a German daily newspaper. The corpus includes POS tags, phrase structure syntax, and a set of morphological information. These annotations were produced in a semi-automatic fashion, but corrected by human annotators. However, problems arise when adapting this data set to new tasks and formalisms. For the CoNLL 2007 and 2009 shared tasks, each data set contained dependency syntax annotations. To meet this requirement the Tiger Corpus's phrase structure syntax was converted to dependency structure using the Tiger2Dep conversion tool[7], resulting in a number of unusual annotations.

In Figure 6.9 we present two dependency trees, one taken from the CoNLL 2009 data set (*left*), the other induced by the HIDDEN model (*right*). The gold standard tree treats the prepositional phrase headed by *auf* as a flat structure: all words contained in the NP take *auf* as their heads. This removes the possibility of a directed path between any two of the words contained within the span, excluding the phrase's head. Consequently, if two of the non-head words within the span are in a predicate-argument relation, it is impossible to find a corresponding dependency in the syntax. This can hinder performance of the ORACLE model, as this is precisely the constraint that the model implements. [8]

In contrast, the trees recovered by the HIDDEN model capture a deeper NP structure in which *einen* and *erneuten* are children of the inner noun, *Urknall*. Not only does this structure aid the model in recovering more accurate SRL analyses, but in many cases it is more representative of the syntactic structure found in the Tiger Corpus prior to conversion. To examine how these trees matched native speaker intuitions, we presented pairs of trees, one from the CoNLL German data and the other produced by the HIDDEN model, to three native German speakers proficient in computational linguistics and tree annotation. HIDDEN model syntax was decoded by placing all arcs corresponding to $Link(i, j)$ variables with beliefs greater than 0.5 into the final tree. All speakers selected the HIDDEN model trees as being more representative of a correct linguistic analysis of the sentence.

[7] `http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/Tiger2Dep.en.html`

[8] [172] also observe that flat structure within NPs governed by PPs is a widespread artifact of the conversion process, and alter these trees prior to evaluating against them.

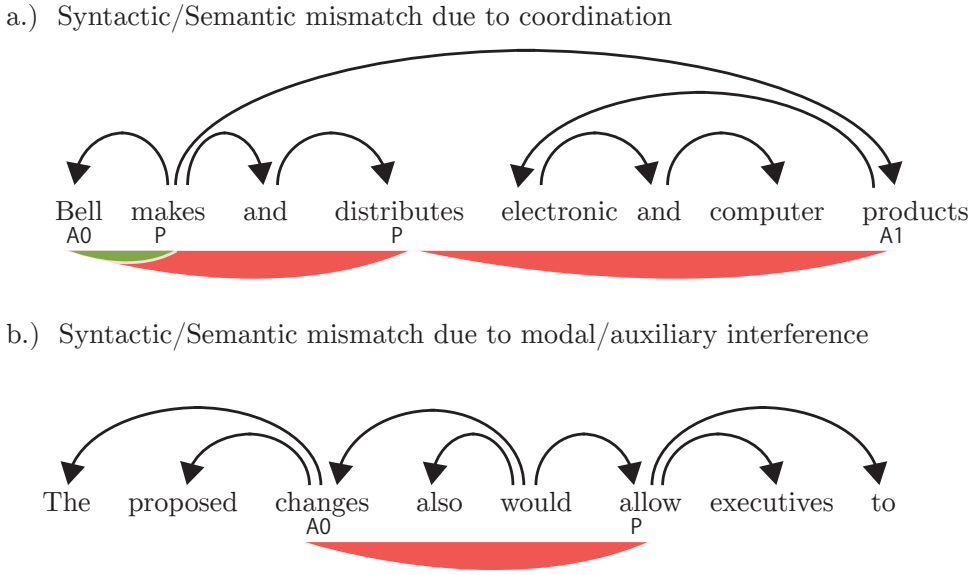


FIGURE 6.10: English ORACLE error analysis. Both coordination and the inclusion of modal verbs along the dependency path reduces ORACLE prediction accuracy (missed SRL predictions are indicated with red spans.)

English

Unlike the results on the two previously discussed languages, the HIDDEN system did not outperform the ORACLE system on the English data set. There are however several notable cases where the use of latent syntax consistently led the model to better solutions. One set of improvements stems from the HIDDEN model finding an alternate internal structure for noun phrases that would sometimes better align with semantic role dependencies. This is similar to the issue with the ORACLE model and gold syntax for the CoNLL German data discussed previously, although here it is not the result of a constituent-to-dependency tree conversion. We refrain from discussing this particular issue further and focus instead on more novel phenomena.

A second major cause of missed syntactic role predictions stems from coordination. Figure 6.10a shows a sentence from the English data set, the gold syntax provided with that data, and a subset of the semantic roles that should have been predicted by the ORACLE model. In this example *Bell* is described as a company which both makes and distributes electronic products. Therefore a semantic analysis should identify *makes* and *distributes* as predicates, with *Bell* and *products* as roles (subject (A0) and direct object (A1), respectively) in both cases. The gold syntactic tree directly supports this analysis for the *makes* predicate, but not for *distributes*. Instead, the head of *distributes* is the coordinate *and*, and is connected to *Bell* only through *makes*.

The HIDDEN model learns to correctly predict both sets of semantic roles. It accomplishes this by avoiding making any hard decision regarding which edges should be in the syntactic tree. Like the gold syntax, the most probable tree according to the HIDDEN model also lacks a dependency between *distributes* and *products*. However, there is a strong pairwise, the marginal belief corresponding to that edge is $\mu = 0.63$ (strong enough to be included in a tree using simple $\mu > 0.5$). This belief provides the necessary evidence to correctly predict *products* as an argument

of *distributes*, despite the edge not existing in the most probable tree according to this model, thus demonstrating the advantages of marginalization within a joint inference framework.

A third source of errors is the behavior of modal verbs. In Fig. 6.10b, the model verb *would* creates a conditional statement with its syntactic child, *allow*. Thus *would* intercedes in the syntactic path between actions like *allow* and participants like *changes*, preventing a direct dependency path between them. This hinders the ability of the ORACLE model to make the correct semantic role prediction. The marginal corresponding to that edge is low, $\mu = 0.39$, but this is still significantly more evidence of a syntactic dependency than offered by the gold tree, and through influencing the corresponding semantic argument variable allows the model to correctly capture this semantic frame. While the HIDDEN model performs better in this case, it's important to note that a pipelined model which uses richer path features ("changes $\leftarrow$ would $\rightarrow$ allow") when making SRL predictions may also be less prone to these errors.

6.4 Conclusion

In this chapter we presented several models of SRL, ranging from simple baselines to fully joint models of dependency parsing and SRL, and outlined how additional modeling structure could be added to capture valency. We find that the most sophisticated of these models perform comparably to the other joint systems from the 2009 CoNLL shared task competition, in some instances reaching state-of-the-art performance, and thus demonstrating the general usefulness of our approach to joint model construction and inference.

When combined with marginalization-based training, the latent syntactic joint model convincingly recovers a majority of the performance gains offered by observing gold syntactic annotations, a result that is consistent across all languages. We find that not only is the hidden syntax model competitive in its own right, outperforming nearly all previous systems on the Chinese data set, but it also outperforms the gold syntax joint models on German and Japanese. In these instances the syntactic annotation style of the corpora did not offer good support for the end task predictions.

This emphasizes an underlying assumption of this dissertation: a syntactic representation cannot be maximally informative for all end tasks, and thus any fully supervised joint model which aims to leverage its structure toward a single end task is susceptible to poorly suited supporting annotations. A well-designed model can avoid some of these problems, as might a second-order model in the case of the Japanese data, but a latent variable approach can guarantee that the induced syntactic representation is well-suited to the task.

In examining the labeled vs. unlabeled performance of our models, it is clear that weak role labeling and sense prediction is the primary cause of low overall performance. The unlabeled performance, which most directly reflects the effect of syntactic correspondence, often exceeds 90% F1, but the model is unable to maintain such performance gains through to the labeled evaluation. Future work could improve upon the labeling aspect of the model, either through additional model structure and features, or by inducing a labeled syntax so that labeling could leverage syntactic cluster information to help identify the types of predicted roles.

7 Conclusion

This thesis has considered how to construct joint models for NLP tasks. As syntactic information is crucial for many of these problems, and parsing a prerequisite step in many systems, we focus on models which include a syntactic component. This raises a number of questions:

1. How can we represent syntactic structure in graphical models in a manner that enables efficient and effective inference?

2. How do we formulate the connections between the component models in a joint model?

3. The data requirements for joint inference are high, requiring a single data set with annotations for each task. In most languages and for most combinations of tasks, this data is not available. This imposes a severe limitation on the potential widespread adoption of joint inference for practical applications, and begs the question: can we better utilize the annotated data we do have to help alleviate this requirement? More specifically, if we ultimately care only about the predictions for a single task, and we have annotated data for this task, can we utilize these annotations to aid in learning unobserved representations?

4. It is unlikely that a single type of linguistic annotation style can be ideal for all tasks. Statistical parsers are trained in a supervised manner with the aim of reproducing the syntactic annotation style of their training data. However, there is no guarantee that those trees will be be maximally beneficial for making end task predictions. Rather than attempting to train parsers to produce this kind of syntactic structure, can we instead induce a task-specific "syntactic" structure, subject to very general constraints, that is better suited to the desired task?

To this end we have presented a framework for constructing and reasoning with joint models using factor graphs. Our hypothesis is that factor graphs are especially well-suited to joint inference:

- Factor graphs are a well-established framework. From conditional random fields for sequence labeling, to semi-Markov CRFs for information extraction, to more arbitrary graphs for large-scale problems like coreference, factor graphs have led to state-of-the-art discriminatively-trained models in nearly all aspects of NLP.

- Factor graphs naturally provide a means for coordinating component problems, through the shared language of probability theory. If we have reason to believe that two variables from separate component models are dependent, they can be connected via a factor in the graph, with the factor specifying the nature of their relationship. Here soft Boolean logic constraints provide a particularly intuitive method of coupling models while remaining robust to exceptions that would likely hinder models with hard constraints.

- Factor graphs offer alternative ways of performing inference in otherwise densely connected structured models. A major shortcoming of factor graphs in NLP is that for modeling combinatorial structures, like those found in syntax, in order to impose the correct structural constraint it may be necessary to include a large, sometimes exponential, number of additional factors. By using globally-connected combinatorial factors, this approach leverages the results of decades of research, taking efficient dynamic programming inference algorithms for structured models, and inserting this logic into model where it can be used in conjunction with standard message passing algorithms.

- Factor graphs support well-established inference algorithms, sum-product and max-product, which are exact in chain or tree-structured models, and approximate in cyclic graphs. We show that while the loopy variant of such algorithms is only approximately correct, these approximations yield good results in practice. Most importantly, belief propagation inference provides a principled method for dealing with latent structure, and we marginalize over latent variables during training to induce syntactic distributions that suit the end task.

Among the contributions of this thesis is a novel combinatorial factor for phrase structure grammar, which we utilize to construct a factor graph parser. We show that a variant of the Inside-Outside algorithm can be used during inference, from within such a globally-connected combinatorial factor, to compute out-going messages for constituent span variables in $O(n^3)$ time. Extending this representation with label variables increases the complexity to $O(|L|n^3)$ where L is the set of constituent labels. Surprisingly the lack of a traditional grammar, one with dependencies between constituent labels, does not significantly hinder this model's performance on the task of parsing. Our parser outperforms comparable discriminative grammar-based parsers ([89]) by a large margin. While our intent is primarily to develop a representation to aid in joint inference, we also show how grammatical rules can be represented as factors and how they can be selectively added to the model to correct mistakes of the labeled parsing model alone.

Having provided a method for representing syntactic models in factor graphs, we construct several novel joint models for a diverse set of common NLP tasks. In named entity recognition, relation extraction, and semantic role labeling we demonstrate the effectiveness of joint inference by outperforming baselines trained without syntactic information. We outperform the best reported results for named entity recognition on the OntoNotes data set[1], though joint inference does not

[1] [28] present better results, but utilize additional singly-annotated training data.

provide a significant advantage for all OntoNotes newswire corpora when compared to a feature-rich span-based NER baseline. The usefulness of joint inference is again demonstrated on a relation extraction task (primarily on the English data set), where we present two joint models, one coupled to phrase structure and the other to dependency structure. In SRL our system is competitive with the top performing systems entered in the CoNLL 2009 shared task competition, many relying on heavy language-specific feature engineering and tuning.

Most importantly, factor graphs not only provide a method for joint inference, but also for joint training. For our most central contribution, we have shown how to train joint models with syntactic components in the absence of syntactic training data. Across a wide variety of tasks, joint models trained with latent syntactic structure provide performance comparable to or better than analogous models which utilize gold syntax trees (either from hand-annotated data or produced by pre-trained state-of-the-art parsers).

More staggering is the performance difference between models with induced syntactic structure and the baseline models, which do not utilize any syntactic information. Outside of the academic domain, where the syntactic training data necessary to train a parser is scarce or altogether absent, there are but two options: use an unsupervised grammar induction method as the parsing component in a pipeline, or forgo modeling syntax altogether. We have shown that our method offers substantial improvements over baselines which do not utilize any syntactic information and [159], in a continuation of our work (using identical models), show that our method outperforms pipelines with unsupervised parsers in an SRL task.

The degree of success associated with this method depends on a variety of factors. We find the largest gains in performance from the joint model occur when the end task relies on long-range syntactic dependencies. In the case of NER, entities are often simple noun phrases and occur toward the leaves of the tree, often spanning only a few tokens. A span-based model is able to predict these entities with such a high degree of accuracy that it leaves little room for improvement in our joint models. In this case, both the NER and syntax models capture similar information, and the gains are minimal. When the features of the two models are complementary (i.e., NER model is a linear-chain, with no span-based information), syntax can help a great deal in identifying entity spans.

Data can also be problematic. In the OntoNotes data the lack of explicit nested NP structure and inconsistent inclusion of sentence-final punctuation in entities greatly reduces the degree of correlation between entities and NPs, and likely hurts supervised joint model performance. For relation extraction on the ACE data set, where heavy preprocessing was necessary to tokenize and annotate the data with part-of-speech tags, syntax trees, entities, and relations, the accumulation of errors in the preprocessing pipeline left many sentences unusable. For the Chinese data in particular, the data amounted to only a few hundred sentences. This is likely too small to accurately estimate the parameters of a joint model, especially when treating syntax as a latent variable.

Ultimately we concede that a pipeline approach still offers some advantages over fully-supervised joint inference. It is easier to find many data sets, each annotated for a single task, than it is to find a single data set annotated for many. Furthermore, because the output of previous components is a complete decoded structure (i.e., all uncertainty is lost), it is easy to construct arbitrarily complex features from these structures with little additional computational cost. Consider a higher-order dependency parse feature, a feature indicating the presence of a "grandparent" relation. For a triple of indices, (i, j, k), k is the grandparent of i if there exists a j such that j is the parent of i and k is

the parent of j. To incorporate this feature into an SRL model in the pipeline approach, one merely needs to search the tree produced by the parser component for an appropriate structure. In the joint model, where the model maintains a belief over syntactic dependencies, modeling this dependency requires an additional set of factors. For every (i, j, k) triple, a ternary NAND (the strategy taken in [11]) factor can capture whether the grandparent relation exists, and this information could be propagated to the SRL model by connecting these factors to variables in the SRL model. Thus in cases where heavy feature engineering is beneficial, and many of the features are difficult to efficiently represent in a joint model, there is some performance benefit to pipelined components. This explains why pipelined models often outperform joint models in shared task competitions, even ones which are designed with joint models in mind.

However, our joint models offer competitive performance, and couple it with the unique ability to leverage end task annotations to induce latent structure for supporting tasks. This completely removes the need for syntactically annotated data and makes the system robust to annotation error and domain mismatch between the various training data sets often used in a pipeline. This is an advantage with potentially far-reaching implications, which may help bring state-of-the-art NLP to many languages and domains where syntactic annotations are unavailable.

7.1 Future Directions

In this dissertation we present a framework for joint inference with the intent of making it as general and widely-applicable to other NLP problems as possible. As such, there are many interesting research directions still to be investigated. We categorize these areas broadly as exploring upstream, into new latent structure below syntax, downstream, into applying these methods to new end tasks, and speed, utilizing pruning and coarse-to-fine strategies to allow for more sophisticated modeling structure.

7.1.1 Additional Latent Linguistic Structure

Of the many areas of formal study within the field of linguistics, not all have shared in receiving the same degree of attention as syntax. Syntactic information has become a critical component in many NLP systems, and for many NLP tasks. Its widespread use and combinatorial nature motivate our decision to focus on syntax as the latent component in joint models throughout this dissertation. However, there are many other aspects of linguistic theory that might be borrowed, and many other structures which might be learned in an unsupervised manner, in order to improve the performance of NLP systems.

Morphology

For morphology, the study of sub-word structure, this may be a very natural transition. Learning morphology segmentations has received considerable research attention, with a particular focus given to unsupervised methods, where models are developed from a cognitive slant. Many of these models are based around the minimum description length principle: assume a set of data, composed of words types or tokens, and a codebook which represents it. Representing each type or token as its own entry will have a set cost. The words "*throw*", "*throwing*", "*throws*" would

occupy three entries in the codebook. An alternative solution is to break up these words and store substrings. For instance, storing "*throw*" as its own entry in the codebook reduces this cost to one, assuming the codebook already contains the suffixes "+*s*", "+*ing*", and "+*s*". The model prefers smaller codebooks, thus favoring a sparse solution. This is the same intuition underlying Goldwater's morphology work, where a Bayesian interpretation of MDL is used [173].

This type of model is not particularly well-suited to a factor graph: if the data is represented as a consecutive string, ranging from $1 \leq i \leq n$, a segmentation model could place a boolean variable at each index i, representing the presence of a morphological (or word) boundary, but all of these variables must then connect to a global factor to impart the combinatorial MDL constraint. Such a combinatorial factor could also implement a similar semi-Markov constraint, limiting the size of segmentations, or of particular segment types (stems, prefixes, suffixes, etc.). However, there are no theoretic hurdles to overcome, and there are many benefits that could come from transitioning from unsupervised morphology learning to task-directed morphology learning.

This is especially true when thinking of NLP outside of English. Morphological representations have been shown to be important when parsing morphologically-rich languages. Similarly, parse information has been shown to guide morphological assignments [7]. Morphology has been shown to improve automatic speech recognition quality in Arabic [174]. It has also been a useful component in MT, both in preprocessing [175] and in jointly inferring an alignment model. Importantly, each of these tasks can be tackled with discriminative end-to-end models, and are all models where we observe some form of end task supervision, whether it be parse trees, properly transcribed speech, or parallel text. We can phrase them as a discriminative joint model where morphological segmentations, and perhaps other structures, are treated as unobserved variables, and the corresponding parameters learned as to optimize performance on the end task. This could also be true of morphological paradigms, which have already been learned in a fully unsupervised manner [176], and these models could be adapted to task-directed training.

Orthography and Phonology

Another source of linguistic information which might be accurately captured in a task-driven manner is phonology. Phonetics is the study of the physical properties of speech sounds the articulations that produce them, morphology studies the minimal meaningful linguistic units, and in between the two is phonology, which studies how speech sounds are categorized internally. For instance, the l sound (as at the start of *linguistics*) and the r sound (as in $rule$) are indistinguishable most Japanese native speakers, as the phonology of Japanese maps the physical sound associated with each to the same internal unit, knows as *phoneme*.

Just as humans internalize the speech stream as a sequence of phonemes, a computational model could learn a latent clustering of observed speech sounds. This latent structure could be constrained according to phonological theory. For instance, previous work has marginalized hidden metrical structures [177], and a rich latent phonological structure (including morae, syllable structure, and stress) could be modeled and marginalized over to aid in tasks like speech recognition or speech-to-speech machine translation.

But phonological theory is not entirely limited to speech processing, and even text bears artifacts of phonological processes. In English there are many predictable orthographic transformations, often mimicking phonological transformations. For instance, when the word *bet* is the basis of a gerund, formed by adding the +*ing* suffix, the '*t*' is duplicated. Systems that attempt to reduce

words to their stems or lemmas need to capture this kind of information to perform well. [178] show that this information can be induced in an unsupervised manner jointly with morphological segmentations, but it could also be induced in a task-directed manner guided by the annotations of a downstream task.

7.1.2 Exploring New End Tasks

In this thesis we have explored many potential end tasks, including part-of-speech tagging, named entity recognition, relation extraction, and semantic role labeling. Still, there are many other tasks which typically rely upon syntax and syntactic features to achieve state-of-the-art performance, and could potentially benefit from our approach. Here we will detail one such task, disfluency detection, but others may include machine translation or natural language generation.

In all tasks presented in this dissertation, the input is text, and is pre-processed to remove many of the noise, errors, and back-tracking found in natural human speech. Consider the sentence:

I want to uh book a flight to Boston , uh , I mean to New York

The speaker's intention is to inform the listener of his desire to fly to New York, but a performance error causes him to mistakenly replace "*New York*" with "*Boston*". What we observe is a common pattern in disfluent speech: an error is made (the *reparandum*), a noise or phrase indicates the desire to backtrack (the *interregnum*, "uh, I mean"), and the speaker ultimately substitutes the correct phrase (the *repair*). There is a long research history studying disfluent speech, and many characteristics which could serve as the basis of a modeling constraint. For our purposes, a disfluency has syntactic and semantic dependencies which can be modeled in our factor graph framework.

In terms of syntax, the connections between model variables might be similar to that of the joint named entity model of Chapter 4. However, the right edge of the reparandum may have uniquely important cues, as the abrupt speech error leaves a constituent fragment whose left side is typical and whose right side is unusual. Our feature-rich discriminative parser may be able to cope well with identifying these spans.

Often a subset of the words in the reparandum are semantically related to the a subset of the words in the repair, or serve similar syntactic functions. We can model this similarity through decomposition. Intuitively the degree to which a span is similar to another is the degree to which its subspans are similar to subspans in the opposing span. For span (7,9), *to Boston*, and span (14,17), *to New York*, the similarity measure could be taken as the sum or max of all possible alignments between subspans. Span (7,8) and (14,15) are identical, contributing a high similarity score. Spans (8,9) and (15,17) are both places, and this could either be learned in a supervised manner, or through clustering or distributional cues in training data or additional raw text. This is the intuition behind previous approaches [179], but instead of modeling the similarity with a tree-adjoining grammar, the alignment could be decomposed and expressed in terms of (hard) logic factors. In the complete joint model, syntax could be marginalized over to improve disfluency detection, or disfluencies could be marginalized over to improve parsing.

7.1.3 Incorporation of Approximate or Pruned Approaches to Inference

In this work we have focused primarily on demonstrating the *effectiveness* of joint inference and its use in inducing task-directed latent syntactic structure. However, it has become apparent throughout the course of this work that the *efficiency* of inference and training, though not our primary concern, may warrant additional attention. In general we believe our architecture has many advantageous properties in this area. Adding factors to the model increases the complexity of belief propagation inference additively (instead of multiplicatively), less tractable model structures can be replaced with combinatorial factors, and training can be performed in an online manner. Yet, the training time for the latent syntax models on larger data sets can be exorbitant. For the English and Chinese data sets for the SRL task, training can easily exceed a week's worth of time.

Thus an important area of future research would seek to perform inference and training more efficiently. Many of the procedures for this have already been outlined. In terms of training, there are limited ways in which to improve significantly upon stochastic gradient descent. Parallelism is difficult in online methods, and while it has been suggested that parameter updates can be done in an entirely non-locking manner (Hogwild!, [180]) we are limited in the number of cores readily available (16 cores) and we do find small performance drops in preliminary experimentation with this method. Parallelism during inference (parallelism in message computation) could also be a viable method for improving efficiency.

Another approach is to limit the number of model parameters by hashing features into a lower dimensional space. Here we find significant improvements in speed with little performance degradation in many of the fully supervised models, but more notable performance loss when hashing models with latent structure. Additionally, in the supervised case features which do not appear on correct structures can be removed, but this cannot be exploiting in the unsupervised case. There may be ways to similarly prune this space (one could use frequency as a simple criterion).

A promising approach to drastically speeding up inference is to construct the graph incrementally. This can be done in many ways, but most involve first constructing a coarse-grained or simplified version of the full graph. The usefulness of new factors and variables is then scored, and structure that meets a given criterion is added to the graph. [106] present one method, relaxed marginal inference, where KL divergence is used to score candidate factors. For dependency parsing (second order models) this approach resulted in a ten-fold increase in inference speed. Similarly, delayed row and column generation [107] provides another approach to selectively constructing the graph, with guarantees of optimality. A phrase structure parser could have as its simplest model an unlabeled parser, adding labeling components for a select set of brackets, and adding a sparse set of rules over select constituent labels. A tagger might begin by tagging with a coarse tagset, add variables representing a subset of fine tags (or variable values in the multinomial case), before adding connections to other morphological tag variables, or to syntactic dependencies. One might also leverage this technique to build models that go beyond the sentence level, performing parsing, relation extraction, or semantic role labeling over paragraphs or small documents.

The opposite may also be true: inference could start by operating on the full graph, and nodes may be selectively removed once their beliefs have converged to reduce the number of messages which must be calculated. Many joint models couple just two tasks, but [181] present a joint model of three – named entity labeling (where boundaries are already identified), relation extraction, and coreference. To reduce the speed of inference, variables whose beliefs are sufficiently close to 1 or

0 are pushed to the nearest value. Factors which connect only fixed variables can be removed from future computations.

There is a vast space of possible directions to take this in which might result in improved efficiency, but the importance of this improvement is to incorporate additional modeling structure. As we observe in the SRL models, both syntactic structure and valency modeling improve SRL performance. However, inference becomes too slow to train models with both for the same number of iterations on all but the smallest of CoNLL data sets. While we have not yet achieved state-of-the-art results in the SRL scores we present, it is possible that we have already presented state-of-the-art models. Further exploration into more efficient inference techniques may serve to verify this, and provide more compelling evidence for joint inference over pipelined approaches with heavy feature engineering.

7.1.4 Exploring the use of additional supervision

This thesis treats the nature of learning algorithms as a dichotomy: the joint models proposed here are trained in either a fully-supervised manner where syntactic annotations are observed for all instances, or in a semi-supervised manner in which all syntactic variables are unobserved and are marginalized out during training. This reflects a worldview in which either syntactic annotations are completely available, or completely unavailable. In practice this will often be inaccurate, as it is likely that either there is some syntactic annotation available, or that a small number of syntactic annotations could be produced for use in training.

Previous work has shown that a small number of labeled instances can significantly outperform fully unsupervised training [**?**], and whether or not such an approach would be useful in the proposed task-driven semi-supervised approach is an open question. However, the models and marginalization-based training scheme presented here can naturally accommodate training from whatever labels are available through the same two-step inference approach, clamping observed variables to their true values and marginalizing over unobserved variables during the calculation of the "gold" feature expectations. Therefore an obvious extension of this work could explore the use of partially-labeled syntactic data of varying sizes and sources in the marginalization-based training scheme. This could also help answer the question of how best to utilize human annotators, and how to maximize system performance vs. annotator time trade-off.

Additionally, an underlying hypothesis of this thesis is that as we move closer to the user's end task it generally becomes easier for non-experts to provide the annotations necessary for training them. For instance, it is likely easier to ask non-experts to create SRL annotations than treebank annotations, as humans are naturally more familiar with the concepts of agent and patient in SRL frames than they are with abstract syntactic concepts drawn from linguistic theory. While we do not pursue a large scale study to validate this hypothesis here, it is complementary to the aforementioned question of how useful a small number of syntactic annotations can be to semi-supervised training.

7.2 Final Thoughts

Language is a trait unique to our species, and we excel at the acquisition and understanding of language to a degree not yet matched by the most powerful computers or the most sophisticated

algorithms. Part of the explanation for our proficiency in all aspects of language processing is our ability to integrate linguistic information from varying sources. A misinterpreted word, an incorrect attachment, or a phrase out of context are potential sources of errors which consistently cause NLP systems to fail, but rarely pose any significant problem to the human listener. It stands to reason that computational models will also require this ability in order to succeed at mastering language processing.

What will these models of the future look like? How will they be trained? In such a fast-paced field it is too early to even begin to answer these questions, but in this thesis we have attempted to push the current state-of-the-art toward this vision of integrated language processing. It is exceedingly unlikely that we will ever have annotations to train intermediary models, like syntax, for all the domains and languages which we hope will benefit from state-of-the-art NLP systems. By utilizing the error signal of end task annotations, we aim to reduce the annotation burden associated with joint inference, and eliminate the possibility of training data that is poorly matched in domain or style. We believe this work serves as both a proof of concept of this approach and an outline of a flexible architecture for future exploration into ever more complex, ever more practical joint models.

References

[1] M. C. MacDonald. *The interaction of lexical and syntactic ambiguity*. Journal of Memory and Language **32**, 692 (1993).

[2] D. Plaut, J. Mcclelland, M. Seidenberg, and K. Patterson. *Understanding normal and impaired word reading: Computational principles in quasi-regular domains* (1996). URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.2570`.

[3] J. C. Trueswell, M. K. Tanenhaus, and S. M. Garnsey. *Semantic influences on parsing: Use of of thematic role information in syntactic ambiguity resolution*. Journal of Memory and Language **33**, 285 (1994).

[4] R. Treiman, C. Clifton, A. Meyer, and L. Wurm. *Language comprehension and production*, vol. 4, pp. 527–548 (John Wiley & Sons, 2003).

[5] H. Poon and P. Domingos. *Joint inference in information extraction*. In *In Proceedings of the 22nd National Conference on Artificial Intelligence (2007*, pp. 913–918 (2007).

[6] J. R. Finkel and C. D. Manning. *Joint parsing and named entity recognition*. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 326–334 (Association for Computational Linguistics, Boulder, Colorado, 2009). URL `http://www.aclweb.org/anthology/N/N09/N09-1037`.

[7] J. Lee, J. Naradowsky, and D. A. Smith. *A discriminative model for joint morphological disambiguation and dependency parsing*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 885–894 (Association for Computational Linguistics, Portland, Oregon, USA, 2011). URL `http://www.aclweb.org/anthology/P11-1089`.

[8] S. Singh, S. Riedel, B. Martin, J. Zheng, and A. McCallum. *Joint inference of entities, relations, and coreference*. In *CIKM Workshop on Automated Knowledge Base Construction (AKBC)* (2013).

[9] F. Kschischang, S. Member, B. J. Frey, and H. andrea Loeliger. *Factor graphs and the sum-product algorithm*. IEEE Transactions on Information Theory **47**, 498 (2001).

[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988).

[11] D. Smith and J. Eisner. *Dependency parsing by belief propagation*. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pp. 145–156 (Association for Computational Linguistics, Honolulu, Hawaii, 2008). URL http://www.aclweb.org/anthology/D08-1016.

[12] A. Viterbi. *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*. IEEE Trans. Inf. Theor. **13**(2), 260 (1967). URL http://dx.doi.org/10.1109/TIT.1967.1054010.

[13] D. Gildea. *Corpus variation and parser performance*. In L. Lee and D. Harman, eds., *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pp. 167–202 (2001).

[14] L. Huang and D. Chiang. *Better k-best parsing*. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pp. 53–64 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dl.acm.org/citation.cfm?id=1654494.1654500.

[15] S. Riedel and J. Clarke. *Incremental integer linear programming for non-projective dependency parsing*. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pp. 129–137 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2006). URL http://dl.acm.org/citation.cfm?id=1610075.1610095.

[16] V. Punyakanok, D. Roth, W. tau Yih, and D. Zimak. *Semantic role labeling via integer linear programming inference*. In *In Proceedings of COLING-04*, pp. 1346–1352 (2004).

[17] T. Marciniak and M. Strube. *Beyond the pipeline: discrete optimization in NLP*. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pp. 136–143 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dl.acm.org/citation.cfm?id=1706543.1706568.

[18] D. Roth and W. tau Yih. *A linear programming formulation for global inference in natural language tasks*. In *In Proceedings of CoNLL-2004*, pp. 1–8 (2004).

[19] V. Punyakanok, D. Roth, and W. tau Yih. *The necessity of syntactic parsing for semantic role labeling*. In *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI*, pp. 1117–1123 (2005).

[20] T. Tsai, C. wei Wu, Y. chun Lin, and W. lian Hsu. *Exploiting full parsing information to label semantic roles using an ensemble of me and svm via integer linear programming*. In *In Proceedings of CoNLL-2005* (2005).

[21] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola. *On dual decomposition and linear programming relaxations for natural language processing*. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1–11 (Association for Computational Linguistics, Cambridge, MA, 2010). URL http://www.aclweb.org/anthology/D10-1001.

[22] A. M. Rush and M. Collins. *Exact decoding of syntactic translation models through lagrangian relaxation*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 72–82 (Association for Computational Linguistics, Portland, Oregon, USA, 2011). URL http://www.aclweb.org/anthology/P11-1008.

[23] Y.-W. Chang and M. Collins. *Exact decoding of phrase-based translation models through lagrangian relaxation*. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 26–37 (Association for Computational Linguistics, Edinburgh, Scotland, UK., 2011). URL http://www.aclweb.org/anthology/D11-1003.

[24] M. Auli and A. Lopez. *A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 470–480 (Association for Computational Linguistics, Portland, Oregon, USA, 2011). URL http://www.aclweb.org/anthology/P11-1048.

[25] J. DeNero and K. Macherey. *Model-based aligner combination using dual decomposition*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 420–429 (Association for Computational Linguistics, Portland, Oregon, USA, 2011). URL http://www.aclweb.org/anthology/P11-1043.

[26] A. Martins, N. Smith, M. Figueiredo, and P. Aguiar. *Dual decomposition with many overlapping components*. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 238–249 (Association for Computational Linguistics, Edinburgh, Scotland, UK., 2011). URL http://www.aclweb.org/anthology/D11-1022.

[27] M. Ostendorf and N. Veilleux. *A hierarchical stochastic model for automatic prediction of prosodic boundary location*. Comput. Linguist. **20**(1), 27 (1994). URL http://dl.acm.org/citation.cfm?id=972509.972511.

[28] J. R. Finkel and C. D. Manning. *Hierarchical joint learning: improving joint parsing and named entity recognition with non-jointly labeled data*. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pp. 720–728 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2010). URL http://dl.acm.org/citation.cfm?id=1858681.1858755.

[29] M.-W. Chang, D. Goldwasser, D. Roth, and V. Srikumar. *Discriminative learning over constrained latent representations*. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pp. 429–437 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2010). URL http://dl.acm.org/citation.cfm?id=1857999.1858065.

[30] A. Clifton, M. Whitney, and A. Sarkar. *An online algorithm for learning over constrained latent representations using multiple views* (2013).

[31] V. I. Spitkovsky, D. Jurafsky, and H. Alshawi. *Profiting from mark-up: Hyper-text annotations for guided parsing*. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 1278–1287 (Association for Computational Linguistics, Uppsala, Sweden, 2010). URL http://www.aclweb.org/anthology/P10-1130.

[32] T. Naseem and R. Barzilay. *Using semantic cues to learn syntax.* In W. Burgard and D. Roth, eds., *AAAI* (AAAI Press, 2011).

[33] P. Blunsom and T. Cohn. *Unsupervised induction of tree substitution grammars for dependency parsing*. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pp. 1204–1213 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2010). URL http://dl.acm.org/citation.cfm?id=1870658.1870775.

[34] A. Gesmundo and K. Hall. *Projecting the knowledge graph to syntactic parsing*. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pp. 28–32 (Association for Computational Linguistics, Gothenburg, Sweden, 2014). URL http://www.aclweb.org/anthology/E14-4006.

[35] K. B. Hall, R. T. McDonald, J. Katz-Brown, and M. Ringgaard. *Training dependency parsers by jointly optimizing multiple objectives.* In *EMNLP*, pp. 1489–1499 (ACL, 2011).

[36] A. Culotta, D. Kulp, and A. McCallum. *Gene prediction with conditional random fields*. IR 419, University of Massachusetts, Amherst (2005).

[37] F. Jelinek. *Statistical methods for speech recognition* (MIT Press, Cambridge, MA, USA, 1997).

[38] A. Ratnaparkhi. *A maximum entropy model for part-of-speech tagging*. In *EMNLP 1997* (1997).

[39] A. McCallum and W. Li. *Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons*. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pp. 188–191 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2003). URL http://dx.doi.org/10.3115/1119176.1119206.

[40] E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. *Equations for part-of-speech tagging*. In *In Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 784–789 (1993).

[41] K. Toutanova and C. D. Manning. *Enriching the knowledge sources used in a maximum entropy part-of-speech tagger*. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, pp. 63–70 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2000). URL http://dx.doi.org/10.3115/1117794.1117802.

[42] C. D. Manning. *Part-of-speech tagging from 97% to 100%: Is it time for some linguistics?* In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, CICLing'11, pp. 171–189 (Springer-Verlag, Berlin, Heidelberg, 2011). URL http://dl.acm.org/citation.cfm?id=1964799.1964816.

[43] C. Sutton, A. McCallum, and K. Rohanimanesh. *Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data*. Journal of Machine Learning Research **8**, 693 (2007).

[44] J. M. Hammersley and P. E. Clifford. *Markov random fields on finite graphs and lattices*. *Unpublished manuscript* (1971).

[45] M. I. Jordan and C. Bishop. *Introduction to graphical models* (2004).

[46] L. R. Rabiner. *A tutorial on hidden Markov models and selected applications in speech recognition*. Proceedings of the IEEE **77**(2), 257 (1989).

[47] K. Lari and S. J. Young. *The estimation of stochastic context-free grammars using the inside-outside algorithm*. Computer Speech and Language **4**, 35 (1990).

[48] D. A. Smith. *Efficient Inference for Trees and Alignments: Modeling Monolingual and Bilingual Syntax with Hard and Soft Constraints and Latent Variables*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD (2010). URL http://cs.jhu.edu/~jason/papers/#smith-2010.

[49] Y. Weiss. *Correctness of local probability propagation in graphical models with loops*. Neural Comput. **12**(1), 1 (2000). URL http://dx.doi.org/10.1162/089976600300015880.

[50] J. Mooij and H. Kappen. *Sufficient conditions for convergence of loopy belief propagation*. In *Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pp. 396–403 (AUAI Press, Arlington, Virginia, 2005).

[51] S. C. Tatikonda and M. I. Jordan. *Loopy belief propagation and Gibbs measures*. In *In Uncertainty in Artificial Intelligence*, pp. 493–500 (Morgan Kaufmann, 2002).

[52] E. T. Ihler, J. W. F. Iii, and A. S. Willsky. *Message errors in belief propagation*. In *In Advances in Neural Information Processing Systems* (MIT Press, 2004).

[53] K. P. Murphy, Y. Weiss, and M. I. Jordan. *Loopy belief propagation for approximate inference: An empirical study*. In *In Proceedings of Uncertainty in AI*, pp. 467–475 (1999).

[54] C. Sutton and A. McCallum. *An introduction to conditional random fields*. Foundations and Trends in Machine Learning **4**(4), 267 (2012).

[55] S. Clark and J. R. Curran. *Parsing the WSJ using CCG and log-linear models*. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 103–110 (Barcelona, Spain, 2004). URL http://www.aclweb.org/anthology/P04-1014.

[56] J. Goodman. *Parsing algorithms and metrics*. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pp. 177–183 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1996). URL http://dx.doi.org/10.3115/981863.981887.

[57] S. Sarawagi and W. W. Cohen. *Semi-Markov conditional random fields for information extraction*. In *In Advances in Neural Information Processing Systems 17*, pp. 1185–1192 (2004).

[58] T. Koo and M. Collins. *Hidden-variable models for discriminative reranking*. In *In Proceedings of HLTEMNLP*, pp. 507–514 (2005).

[59] D. C. Liu and J. Nocedal. *On the limited memory method for large scale optimization*. Math. Prog. B **45**(3) (1989).

[60] F. Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review **65**(6), 386 (1958).

[61] M. Collins. *Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms*. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pp. 1–8 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2002). URL http://dx.doi.org/10.3115/1118693.1118694.

[62] L. Bottou. *Stochastic learning*. In O. Bousquet and U. von Luxburg, eds., *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pp. 146–168 (Springer Verlag, Berlin, 2004). URL http://leon.bottou.org/papers/bottou-mlss-2004.

[63] A. Quattoni, M. Collins, and T. Darrell. *Incorporating semantic constraints into a discriminative categorization and labelling model.* In *Proceedings of the Tenth IEEE International Conference on Computer Vision Workshops*, ICCVW '05, pp. 1877– (IEEE Computer Society, Washington, DC, USA, 2005). URL http://dx.doi.org/10.1109/ICCV.2005.256.

[64] G. Wahba. *Spline models for observational data*, vol. 59 (Society for industrial and applied mathematics, 1990).

[65] A. Y. Ng. *Feature selection, l1 vs. l2 regularization, and rotational invariance*. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pp. 78– (ACM, New York, NY, USA, 2004). URL http://doi.acm.org/10.1145/1015330.1015435.

[66] J. Bresnan. *Explaining morphosyntactic competition*. Handbook of contemporary syntactic theory pp. 11–44 (2001).

[67] A. Rush and S. Petrov. *Vine pruning for efficient multi-pass dependency parsing*. In *The 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL '12)*, p. Best Paper Award (2012). URL http://petrovi.de/data/naacl12.pdf.

[68] D. Klein and C. D. Manning. *A generative constituent-context model for improved grammar induction*. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pp. 128–135 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2002). URL http://dx.doi.org/10.3115/1073083.1073106.

[69] R. McDonald. *Discriminative learning and spanning tree algorithms for dependency parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA (2006). AAI3225503.

[70] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. *Building a large annotated corpus of English: the penn treebank*. Comput. Linguist. **19**(2), 313 (1993). URL http://dl.acm.org/citation.cfm?id=972470.972475.

[71] R. Johansson and P. Nugues. *Extended constituent-to-dependency conversion for English*. In *Proceedings of NODALIDA 2007* (Tartu, Estonia, 2007). URL http://dspace.utlib.ee/dspace/bitstream/10062/2560/1/reg-Johansson-10.pdf.

[72] S. Buchholz and E. Marsi. *Conll-x shared task on multilingual dependency parsing*. In *In Proc. of CoNLL*, pp. 149–164 (2006).

[73] D. Bamman and G. Crane. *The design and use of a latin dependency treebank*. In *Proceedings of the Fifth International Workshop on Treebanks and Linguistic Theories (TLT 2006)*, pp. 67–78 (Prague, 2006).

[74] R. Cann. *Formal Semantics: an introduction* (Cambridge University Press, 1993).

[75] Z. GuoDong, S. Jian, Z. Jie, and Z. Min. *Exploring various knowledge in relation extraction*. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 427–434 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dx.doi.org/10.3115/1219840.1219893.

[76] N. Chomsky. *Syntactic structures*. Ianua linguarum : Series minor ; 4 (Mouton, 1962).

[77] E. Charniak. *Statistical parsing with a context-free grammar and word statistics.* In B. Kuipers and B. L. Webber, eds., *AAAI/IAAI*, pp. 598–603 (AAAI Press / The MIT Press, 1997).

[78] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)* (Prentice Hall, 2008), 2 ed.

[79] E. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. *Decision tree parsing using a hidden derivation model*. In *Proc. Darpa Speech and Natural Language Workshop* (1994).

[80] D. M. Magerman. *Statistical decision-tree models for parsing*. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, pp. 276–283 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1995). URL http://dx.doi.org/10.3115/981658.981695.

[81] N. Chomsky. *Remarks on nominalization* (Waltham: Ginn, 1970).

[82] R. Jackendoff. *X-bar-Syntax: A Study of Phrase Structure*, vol. 2 (MIT Press, 1977).

[83] M. J. Collins. *A new statistical parser based on bigram lexical dependencies*. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pp. 184–191 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1996). URL http://dx.doi.org/10.3115/981863.981888.

[84] M. Collins. *Three generative, lexicalised models for statistical parsing*. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL '98, pp. 16–23 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1997). URL http://dx.doi.org/10.3115/976909.979620.

[85] E. Charniak and M. Johnson. *Coarse-to-fine n-best parsing and maxent discriminative reranking*. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pp. 173–180 (Association for Computational Linguistics, Ann Arbor, Michigan, 2005). URL http://www.aclweb.org/anthology/P05-1022.

[86] T. Matsuzaki, Y. Miyao, and J. Tsujii. *Probabilistic CFG with latent annotations*. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 75–82 (Stroudsburg, PA, USA, 2005). URL http://dx.doi.org/10.3115/1219840.1219850.

[87] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. *Learning accurate, compact, and interpretable tree annotation*. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pp. 433–440 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2006). URL http://dx.doi.org/10.3115/1220175.1220230.

[88] S. Petrov and D. Klein. *Discriminative log-linear grammars with latent variables*. In *In Proceedings of NIPS 20* (2008).

[89] J. Finkel, A. Kleeman, and C. Manning. *Efficient, feature-based, conditional random field parsing*. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 959–967 (2008).

[90] J. Bilmes. *A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden Markov models*. Tech. rep., University of California Berkeley (1998).

[91] J. K. Baker. *Trainable grammars for speech recognition*. The Journal of the Acoustical Society of America **65**(S1), S132 (1979).

[92] F. Pereira and Y. Schabes. *Inside-outside reestimation from partially bracketed corpora*. In *Proceedings of the 30th Annual Meeting on Association for Computational Linguistics*, ACL '92, pp. 128–135 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1992). URL http://dx.doi.org/10.3115/981967.981984.

[93] D. Chiang and D. M. Bikel. *Recovering latent information in treebanks*. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pp. 1–7 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2002). URL http://dx.doi.org/10.3115/1072228.1072354.

[94] S. Petrov and D. Klein. *Improved inference for unlexicalized parsing*. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pp. 404–411 (Association for Computational Linguistics, Rochester, New York, 2007). URL http://www.aclweb.org/anthology/N/N07/N07-1051.

[95] R. Backofen, J. Rogers, and K. Vijay-shanker. *A first-order axiomatization of the theory of finite trees*. Journal of Logic, Language and Information **4**, 5 (1995).

[96] D. A. McAllester, M. Collins, and F. Pereira. *Case-factor diagrams for structured probabilistic modeling*. CoRR **abs/1207.4135** (2012).

[97] N. A. Smith. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies (Morgan and Claypool, 2011).

[98] X. Song, S. Ding, and C.-Y. Lin. *Better binarization for the CKY parsing*. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pp. 167–176 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2008). URL http://dl.acm.org/citation.cfm?id=1613715.1613739.

[99] L. Huang, H. Zhang, D. Gildea, and K. Knight. *Binarization of synchronous context-free grammars*. Computational Linguistics **35**(4), 559 (2009). URL http://dx.doi.org/10.1162/coli.2009.35.4.35406.

[100] S. Petrov. *Coarse-to-Fine Natural Language Processing*. Ph.D. thesis, University of California at Berkeley (2009).

[101] N. Bodenstab, A. Dunlop, K. Hall, and B. Roark. *Beam-width prediction for efficient context-free parsing*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pp. 440–449 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2011). URL http://dl.acm.org/citation.cfm?id=2002472.2002529.

[102] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. *Non-projective dependency parsing using spanning tree algorithms*. In *Proceedings of the conference on Human Language*

Technology and Empirical Methods in Natural Language Processing, HLT '05, pp. 523–530 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dx.doi.org/10.3115/1220575.1220641.

[103] F. Sha and F. Pereira. *Shallow parsing with conditional random fields*. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pp. 134–141 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2003). URL http://dx.doi.org/10.3115/1073445.1073473.

[104] H. Shen and A. Sarkar. *Voting between multiple data representations for text chunking*. In *Proceedings of the 18th Canadian Society conference on Advances in Artificial Intelligence*, AI'05, pp. 389–400 (Springer-Verlag, Berlin, Heidelberg, 2005). URL http://dx.doi.org/10.1007/11424918_40.

[105] N. A. Smith and M. Johnson. *Weighted and probabilistic context-free grammars are equally expressive*. Computational Linguistics p. 491 (2007).

[106] S. Riedel and D. A. Smith. *Relaxed marginal inference and its application to dependency parsing*. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pp. 760–768 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2010). URL http://dl.acm.org/citation.cfm?id=1857999.1858116.

[107] S. Riedel, D. A. Smith, and A. McCallum. *Parse, price and cut - delayed column and row generation for graph based parsers*. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '12)* (2012).

[108] E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel. *Ontonotes: The 90% solution*. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pp. 57–60 (Association for Computational Linguistics, New York City, USA, 2006). URL http://www.aclweb.org/anthology/N/N06/N06-2015.

[109] D. Klein and C. D. Manning. *Accurate unlexicalized parsing*. In *In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pp. 423–430 (2003).

[110] D. Klein and C. Manning. *Fast exact inference with a factored model for natural language processing*. In *NIPS* (2002).

[111] L. Tesnière. *Elements de syntaxe structurale* (Editions Klincksieck, 1959).

[112] T. Koo, A. Globerson, X. Carreras, and M. Collins. *Structured prediction models via the matrix-tree theorem*. In *In EMNLP-CoNLL* (2007).

[113] J. Naradowsky, T. Vieira, and D. A. Smith. *Grammarless parsing for joint inference*. In *24th International Conference on Computational Linguistics (COLING)* (Mumbai, India, 2012).

[114] D. M. Bikel, R. Schwartz, and R. M. Weischedel. *An algorithm that learns what‘s in a name*. Machine Learning **34**(1-3), 211 (1999). URL http://dx.doi.org/10.1023/A:1007558221122.

[115] G. Zhou and J. Su. *Named entity recognition using an HMM-based chunk tagger*. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pp. 473–480 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2002). URL http://dx.doi.org/10.3115/1073083.1073163.

[116] S. Zhao. *Named entity recognition in biomedical texts using an HMM model*. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, JNLPBA '04, pp. 84–87 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2004). URL http://dl.acm.org/citation.cfm?id=1567594.1567613.

[117] H. Isozaki and H. Kazawa. *Efficient support vector classifiers for named entity recognition*. In *In Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002*, pp. 390–396 (2002).

[118] J. R. Finkel, T. Grenager, and C. Manning. *Incorporating non-local information into information extraction systems by Gibbs sampling*. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 363–370 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dx.doi.org/10.3115/1219840.1219885.

[119] J. Liu, M. Huang, and X. Zhu. *Recognizing biomedical named entities using skip-chain conditional random fields*. In *Proceedings of the 2010 Workshop on Biomedical Natural Language Processing*, BioNLP '10, pp. 10–18 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2010). URL http://dl.acm.org/citation.cfm?id=1869961.1869963.

[120] L. A. Ramshaw and M. P. Marcus. *Text chunking using transformation-based learning*. CoRR **cmp-lg/9505040** (1995).

[121] J. Finkel, S. Dingare, H. Nguyen, M. Nissim, C. Manning, and G. Sinclair. *Exploiting context for biomedical entity recognition: from syntax to the web*. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, JNLPBA '04, pp. 88–91 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2004). URL http://dl.acm.org/citation.cfm?id=1567594.1567614.

[122] S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. *A novel use of statistical parsing to extract information from text*. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, NAACL 2000, pp. 226–233 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2000). URL http://dl.acm.org/citation.cfm?id=974305.974335.

[123] I. Lehiste. *Phonetic Disambiguation of Syntactic Ambiguity*. The Journal of the Acoustical Society of America **53**(1), 380 (1973).

[124] M. Ostendorf, P. J. Price, J. Bear, and C. W. Wightman. *The use of relative duration in syntactic disambiguation*. In *Proceedings of the workshop on Speech and Natural Language*, HLT '90, pp. 26–31 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1990). URL http://dx.doi.org/10.3115/116580.116594.

[125] N. M. Veilleux, M. Ostendorf, and C. Wightman. *Parse scoring with prosodic information*. In *In Int. Conf. on Spoken Language Processing*, pp. 51–54 (1992).

[126] C. Clifton, K. Carlson, and L. Frazier. *Informative Prosodic Boundaries*. Language & Speech **45**(Peer Reviewed Journal), 87 (2002).

[127] K. Rayner, M. Carlson, and L. Frazier. *The interaction of syntax and semantics during sentence processing: eye movements in the analysis of semantically biased sentences*. Journal of Verbal Learning and Verbal Behavior **22**, 358 (1983).

[128] E. Selkirk. *Comments on intonational phrasing in English* (2005).

[129] M. Steedman. *The Syntactic Process* (MIT Press, Cambridge, MA, USA, 2000).

[130] L. E. Baum and T. Petrie. *Statistical inference for probabilistic functions of finite state Markov chains*. Annals of Mathematical Statistics **37**, 1554 (1966).

[131] J. Naradowsky and K. Toutanova. *Unsupervised bilingual morpheme segmentation and alignment with context-rich hidden semi-Markov models*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 895–904 (Association for Computational Linguistics, Portland, Oregon, USA, 2011). URL http://www.aclweb.org/anthology/P11-1090.

[132] C. Walker, S. Strassel, J. Medero, and K. Maeda. *Ace 2005 multilingual training corpus*. Linguistic Data Consortium, Philadelphia (2005). URL http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T06.

[133] Z. GuoDong, S. Jian, Z. Jie, and Z. Min. *Exploring various knowledge in relation extraction*. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 427–434 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dx.doi.org/10.3115/1219840.1219893.

[134] R. Mcdonald, F. Pereira, S. Kulick, S. Winters, Y. Jin, and P. White. *Simple algorithms for complex relation extraction with applications to biomedical ie*. In *In Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL-05*, pp. 491–498 (2005).

[135] S. Zhao and R. Grishman. *Extracting relations with integrated information using kernel methods*. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 419–426 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2005). URL http://dx.doi.org/10.3115/1219840.1219892.

[136] L. Yao, A. Haghighi, S. Riedel, and A. McCallum. *Structured relation discovery using generative models*. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pp. 1456–1466 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2011). URL http://dl.acm.org/citation.cfm?id=2145432.2145587.

[137] S. Riedel, L. Yao, B. M. Marlin, and A. McCallum. *Relation extraction with matrix factorization and universal schemas*. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '13)* (2013). URL http://www.riedelcastro.org/publications/papers/riedel13relation.pdf.

[138] N. Xue, F. Xia, F.-d. Chiou, and M. Palmer. *The penn chinese treebank: Phrase structure annotation of a large corpus*. Nat. Lang. Eng. **11**(2), 207 (2005). URL http://dx.doi.org/10.1017/S135132490400364X.

[139] J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, P. Straňák, M. Surdeanu, N. Xue, and Y. Zhang. *The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages*. In *CoNLL: Shared Task*, pp. 1–18 (2009).

[140] J. M. Eisner. *Three new probabilistic models for dependency parsing: an exploration*. In *Proceedings of the 16th conference on Computational linguistics - Volume 1*, COLING '96, pp. 340–345 (Association for Computational Linguistics, Stroudsburg, PA, USA, 1996). URL http://dx.doi.org/10.3115/992628.992688.

[141] D. Bamman and G. Crane. *Corpus linguistics, treebanks and the reinvention of philology.* In K.-P. Fähnrich and B. Franczyk, eds., *Informatik*, vol. 176 of *LNI*, pp. 542–551 (GI, 2010).

[142] D. Roland, F. D.ck, and J. L. Elman. *Frequency of basic English grammatical structures: A corpus analysis* (2007).

[143] J. Tinsley, M. Hearne, and A. Way. *Exploiting parallel treebanks to improve phrase-based statistical machine translation*. In *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '09, pp. 318–331 (Springer-Verlag, Berlin, Heidelberg, 2009). URL http://dx.doi.org/10.1007/978-3-642-00382-0_26.

[144] C. J. Fillmore. *Frame semantics*, pp. 111–137 (Hanshin Publishing Co., Seoul, South Korea, 1982).

[145] A. Bies, M. Ferguson, K. Katz, R. MacIntyre, V. Tredinnick, G. Kim, M. A. Marcinkiewicz, and B. Schasberger. *Bracketing guidelines for treebank ii style penn treebank project*. Tech. rep., University of Pennsylvania (1995).

[146] P. Blunsom. *Maximum entropy Markov models for semantic role labelling* (2004).

[147] C. Sutton and A. McCallum. *Joint parsing and semantic role labeling*. In *Proceedings of CoNLL-2005*, pp. 225–228 (2005).

[148] G. Ngai, D. Wu, M. Carpuat, C. shing Wang, and C. yung Wang. *Semantic role labeling with boosting, SVMs, maximum entropy, SNOW, and decision lists*. In *In Proceedings of SENSEVAL-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text* (2004).

[149] X. Carreras and L. Marquez. *Introduction to the conll-2005 shared task: Semantic role labeling* (2005).

[150] Y. Liu and A. Sarkar. *Using LTAG-based features for semantic role labeling*. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pp. 127–132 (Association for Computational Linguistics, Sydney, Australia, 2006). URL http://www.aclweb.org/anthology/W/W06/W06-1518.

[151] X. Carreras and L. Màrques. *Introduction to the conll-2004 shared task: Semantic role labeling*. In *Proceedings of CoNLL-2004*, pp. 89–97 (Boston, MA, USA, 2004).

[152] K. Toutanova, A. Haghighi, and C. D. Manning. *A global joint model for semantic role labeling*. Comput. Linguist. **34**(2), 161 (2008). URL http://dx.doi.org/10.1162/coli.2008.34.2.161.

[153] S. Riedel and I. Meza-Ruiz. *Collective semantic role labelling with Markov logic*. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pp. 193–197 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2008). URL http://dl.acm.org/citation.cfm?id=1596324.1596357.

[154] I. Meza-Ruiz and S. Riedel. *Jointly identifying predicates, arguments and senses using Markov logic*. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '09)* (2009).

[155] W. Che, Z. Li, Y. Li, Y. Guo, B. Qin, and T. Liu. *Multilingual dependency-based syntactic and semantic parsing*. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pp. 49–54 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2009). URL http://dl.acm.org/citation.cfm?id=1596409.1596417.

[156] H. Zhao, W. Chen, C. Kit, and G. Zhou. *Multilingual dependency learning: a huge feature engineering method to semantic dependency parsing*. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pp. 55–60 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2009). URL http://dl.acm.org/citation.cfm?id=1596409.1596418.

[157] K. Goluchowski and A. Przepiórkowski. *Semantic role labelling without deep syntactic parsing*. In *Advances in Natural Language Processing*, vol. 7614, pp. 192–197 (springer, 2012). URL http://nlp.ipipan.waw.pl/~adamp/Papers/2012-japtal.

[158] S. Boxwell, C. Brew, J. Baldridge, D. Mehay, and S. Ravi. *Semantic role labeling without treebanks?* In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pp. 192–200 (Asian Federation of Natural Language Processing, Chiang Mai, Thailand, 2011). URL http://www.aclweb.org/anthology/I11-1022.

[159] M. R. Gormley, M. Mitchell, B. Van Durme, and M. Dredze. *Low-resource semantic role labeling*. In *Proceedings of ACL* (2014).

[160] C. Fillmore. *The case for case*. In E. Bach and R. Harms, eds., *Universals in Linguistic Theory* (Holt, Rinehart, and Winston, New York, 1968).

[161] X. Lluís, X. Carreras, and L. Màrquez. *Joint arc-factored parsing of syntactic and semantic dependencies*. Transactions of the Association for Computational Linguistics (TACL) **1**(1), 219,230 (2013).

[162] D. Klein and C. D. Manning. *Corpus-based induction of syntactic structure: models of dependency and constituency*. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2004). URL http://dx.doi.org/10.3115/1218955.1219016.

[163] Y. Zhang and J. Nivre. *Transition-based dependency parsing with rich non-local features*. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pp. 188–193 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2011). URL http://dl.acm.org/citation.cfm?id=2002736.2002777.

[164] A. F. T. Martins, N. A. Smith, and E. P. Xing. *Concise integer linear programming formulations for dependency parsing*. 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: Volume 1 - Volume 1*, ACL '09, pp. 342–350 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2009). URL http://dl.acm.org/citation.cfm?id=1687878.1687928.

[165] I. Titov, J. Henderson, P. Merlo, and G. Musillo. *Online projectivisation for synchronous parsing of semantic and syntactic dependencies*. In *In Proceedings of the Internation Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1562–1567 (Pasadena, California, USA, 2009).

[166] R. Johansson. *Statistical bistratal dependency parsing.* In *EMNLP*, pp. 561–569 (ACL, 2009).

[167] M. Y. Luo Yanyan, Asahara Masayuki. *Robust integrated models for chinese predicate-argument structure analysis*. China Communications **9**(3), 10 (pages 8) (2012). URL http://www.chinacommunications.cn/EN/abstract/article_7863.shtml.

[168] H. Zhao, W. Chen, J. Kazama, K. Uchimoto, and K. Torisawa. *Multilingual dependency learning: Exploiting rich features for tagging syntactic and semantic dependencies*. In

Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task, CoNLL '09, pp. 61–66 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2009). URL http://dl.acm.org/citation.cfm?id=1596409.1596419.

[169] X. Lluís, S. Bott, and L. Màrquez. *A second-order joint eisner model for syntactic and semantic dependency parsing* (2012).

[170] J. Naradowsky, S. Riedel, and D. A. Smith. *Improving NLP through marginalization of hidden syntactic structure*. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing* (Association for Computational Linguistics, Jeju, Korea, 2012). URL http://www.cs.umass.edu/~narad/_papers/relmarg_emnlp2012.pdf.

[171] S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. *The TIGER treebank*. In *Proceedings of the workshop on treebanks and linguistic theories*, pp. 24–41 (2002).

[172] D. A. Smith and J. Eisner. *Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies*. In *Proceedings of the Workshop on Statistical Machine Translation*, StatMT '06, pp. 23–30 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2006). URL http://dl.acm.org/citation.cfm?id=1654650.1654655.

[173] S. Goldwater. *Nonparametric Bayesian Models of Lexical Acquisition*. Ph.D. thesis, Brown University (2006).

[174] D. Vergyri, K. Kirchhoff, K. Duh, and A. Stolcke. *Morphology-based language modeling for arabic speech recognition*. In *In Proc. of ICSLP*, pp. 2245–2248 (2004).

[175] S. Goldwater and D. McClosky. *Improving statistical mt through morphological analysis*. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (Vancouver, 2005).

[176] M. Dreyer and J. Eisner. *Discovering morphological paradigms from plain text using a Dirichlet process mixture model*. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pp. 616–627 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2011). URL http://dl.acm.org/citation.cfm?id=2145432.2145504.

[177] J. Naradowsky, J. Pater, D. Smith, and R. Staubs. *Learning hidden metrical structure with a log-linear model of grammar*. In *Workshop on Computational Modelling of Sound Pattern Acquisition* (2010).

[178] J. Naradowsky and S. Goldwater. *Improving morphology induction by learning spelling rules*. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pp. 1531–1536 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009). URL http://portal.acm.org/citation.cfm?id=1661445.1661691.

[179] M. Johnson and E. Charniak. *A tag-based noisy channel model of speech repairs*. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp. 33–39 (Association for Computational Linguistics, 2004).

[180] B. Recht, C. Re, S. Wright, and F. Niu. *Hogwild: A lock-free approach to parallelizing stochastic gradient descent*. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds., *Advances in Neural Information Processing Systems 24*, pp. 693–701 (2011).

[181] S. Singh, S. Riedel, B. Martin, J. Zheng, and A. McCallum. *Joint inference of entities, relations, and coreference*. In *CIKM Workshop on Automated Knowledge Base Construction (AKBC)* (2013).