Training Factored PCFGs with Expectation Propagation

David Hall and **Dan Klein**
Computer Science Division
University of California, Berkeley
{dlwh,klein}@cs.berkeley.edu

Abstract

PCFGs can grow exponentially as additional annotations are added to an initially simple base grammar. We present an approach where multiple annotations coexist, but in a factored manner that avoids this combinatorial explosion. Our method works with linguistically-motivated annotations, induced latent structure, lexicalization, or any mix of the three. We use a structured expectation propagation algorithm that makes use of the factored structure in two ways. First, by partitioning the factors, it speeds up parsing exponentially over the unfactored approach. Second, it minimizes the redundancy of the factors during training, improving accuracy over an independent approach. Using purely latent variable annotations, we can efficiently train and parse with up to 8 latent bits per symbol, achieving F1 scores up to 88.4 on the Penn Treebank while using two orders of magnitudes fewer parameters compared to the naïve approach. Combining latent, lexicalized, and unlexicalized annotations, our best parser gets 89.4 F1 on all sentences from section 23 of the Penn Treebank.

1 Introduction

Many high-performance PCFG parsers take an initially simple base grammar over treebank labels like NP and enrich it with deeper syntactic features to improve accuracy. This broad characterization includes lexicalized parsers (Collins, 1997), unlexicalized parsers (Klein and Manning, 2003), and latent variable parsers (Matsuzaki et al., 2005). Figures 1(a), 1(b), and 1(c) show small examples of context-free trees that have been annotated in these ways.

When multi-part annotations are used in the same grammar, systems have generally multiplied these annotations together, in the sense that an NP that was definite, possessive, and VP-dominated would have a single unstructured PCFG symbol that encoded all three facts. In addition, modulo backoff or smoothing, that unstructured symbol would often have rewrite parameters entirely distinct from, say, the indefinite but otherwise similar variant of the symbol (Klein and Manning, 2003). Therefore, when designing a grammar, one would have to carefully weigh new contextual annotations. Should a definiteness annotation be included, doubling the number of NPs in the grammar and perhaps overly fragmenting statistics? Or should it be excluded, thereby losing important distinctions? Klein and Manning (2003) discuss exactly such trade-offs and omit annotations that were helpful on their own because they were not worth the combinatorial or statistical cost when combined with other annotations.

In this paper, we argue for grammars with *factored annotations*, that is, grammars with annotations that have structured component parts that are partially decoupled. Our annotated grammars can include both latent and explicit annotations, as illustrated in Figure 1(d), and we demonstrate that these factored grammars outperform parsers with unstructured annotations.

After discussing the factored representation, we describe a method for parsing with factored annotations, using an approximate inference technique called expectation propagation (Minka, 2001). Our algorithm has runtime linear in the number of annotation factors in the grammar, improving on the naïve algorithm, which has runtime exponential in the number of annotations. Our method, the Expectation Propagation for Inferring Constituency (*EPIC*) parser, jointly trains a model over factored annotations, where each factor naturally leverages information from other annotation factors and improves on their mistakes.

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1146–1156, Jeju Island, Korea, 12–14 July 2012.

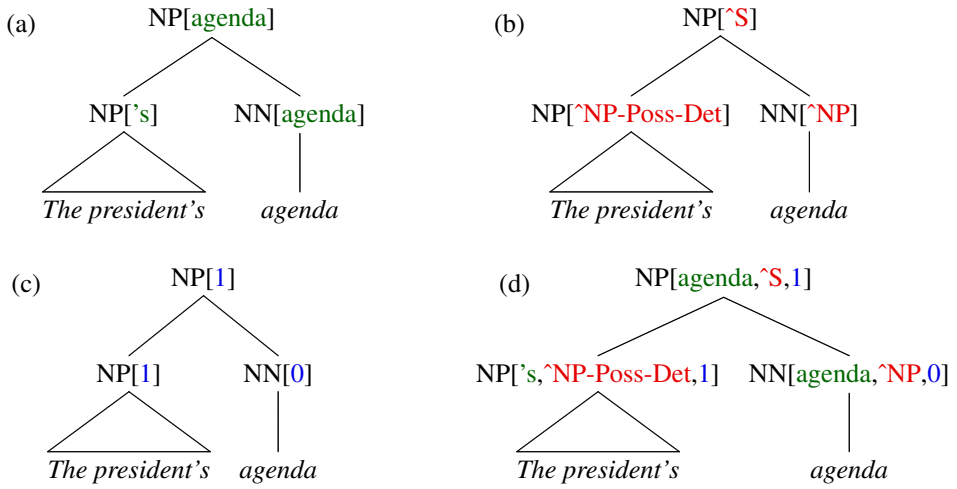


Figure 1: Parse trees using four different annotation schemes: (a) Lexicalized annotation like that in Collins (1997); (b) Unlexicalized annotation like that in Klein and Manning (2003); (c) Latent annotation like that in Matsuzaki et al. (2005); and (d) the factored, mixed annotations we argue for in our paper.

We demonstrate the empirical effectiveness of our approach in two ways. First, we efficiently train a latent-variable grammar with 8 disjoint one-bit latent annotation factors, with scores as high as 89.7 F1 on length $\leq$40 sentences from the Penn Treebank (Marcus et al., 1993). This latent variable parser outscores the best of Petrov and Klein (2008a)'s comparable parsers while using two orders of magnitude fewer parameters. Second, we combine our latent variable factors with lexicalized and unlexicalized annotations, resulting in our best F1 score of 89.4 on all sentences.

2 Intuitions

Modern theories of grammar such as HPSG (Pollard and Sag, 1994) and Minimalism (Chomsky, 1992) do not ascribe unstructured conjunctions of annotations to phrasal categories. Rather, phrasal categories are associated with sequences of metadata that control their function. For instance, an NP might have annotations to the effect that it is singular, masculine, and nominative, with perhaps further information about its animacy or other aspects of the head noun. Thus, it is appealing for a grammar to be able to model these (somewhat) orthogonal notions, but most models have no mechanism to encourage this. As a notable exception, Dreyer and Eisner (2006) tried to capture this kind of insight by allowing factored annotations to pass unchanged from parent label to child label, though they were not able to demonstrate substantial gains in accuracy.

Moreover, there has been to our knowledge no attempt to employ both latent and non-latent annotations at the same time. There is good reason for this: lexicalized or highly annotated grammars like those of Collins (1997) or Klein and Manning (2003) have a very large number of states and an even larger number of rules. Further annotating these rules with latent annotations would produce an infeasibly large grammar. Nevertheless, it is a shame to sacrifice expert annotation just to get latent annotations. Thus, it makes sense to combine these annotation methods in a way that does not lead to an explosion of the state space or a fragmentation of statistics.

3 Parsing with Annotations

Suppose we have a raw (binarized) treebank grammar, with productions of the form $A \rightarrow B\ C$. The typical process is to then annotate these rules with additional information, giving rules of the form $A[x] \rightarrow B[y]\ C[z]$. In the case of explicit annotations, an x might include information about the parent category, or a head word, or a combination of things. In the case of latent annotations, x will be an integer that may or may not correspond to some linguistic notion. We are interested in the specific case where each x is actually factored into M disjoint parts: $A[x_1, x_2, \ldots, x_M]$. (See Figure 1(d).) We call each component of x an *annotation factor* or an *annotation component*.

3.1 Annotation Classes

In this paper, we consider three kinds of annotation models, representing three of the major traditions in constituency parsing. Individually, none of our models are state-of-the-art, instead achieving F1 scores in the mid-80's on the Penn Treebank.

The first model is a relatively simple lexicalized parser. We are not aware of a prior discriminative lexicalized constituency parser, and it is quite different from the generative models of Collins (1997). Broadly, it considers features over a binary rule annotated with head words: $A[h] \rightarrow B[h]\ C[d]$ and $A[h] \rightarrow B[d]\ C[h]$, focusing on monolexical rule features and bilexical dependency features. It is our best individual model, scoring 87.3 F1 on the development set.

The second is similar to the unlexicalized model of Klein and Manning (2003). This parser starts from a grammar with labels annotated with sibling and parent information, and then adds specific annotations, such as whether an NP is possessive or whether a symbol rewrites as a unary. This parser gets 86.3, tying the original generative version of Klein and Manning (2003).

Finally, we use a straightforward *discriminative* latent variable model much like that of Petrov and Klein (2008a). Here, each symbol is given a latent annotation, referred to as a substate. Typically, these substates correlate at least loosely with linguistic phenomena. For instance, NP-1 might be associated with possessive NPs, while NP-3 might be for adjuncts. Often, these latent integers are considered as bit strings, with each bit indicating one latent annotation. Prior work in this area has considered the effect of splitting and merging these states (Petrov et al., 2006; Petrov and Klein, 2007), as well as "multiscale" grammars (Petrov and Klein, 2008b). With two states (or one bit of annotation), our version of this parser gets 81.7 F1, edging out the comparable parser of Petrov and Klein (2008a). On the other hand, our parser gets 83.2 with four states (two bits), short of the performance of prior work.[1]

[1]Much of the difference stems from the different binarization scheme we employ. We use head-outward binarization, rather than the left-branching binarization they employed. This change was to enable integrating lexicalization with our other models.

3.2 Model Representation

We employ a general exponential family representation of our grammar. This representation is fairly general, and—in its generic form—by no means new, save for the focus on annotation components.

Formally, we begin with a parse tree T over base symbols for some sentence $\mathbf{w}$, and we decorate the tree with annotations X, giving a parse tree $T[X]$. We focus on the case when X partitions into disjoint components $X = [X_1, X_2, \ldots, X_M]$. These components are decoupled in the sense that, conditioned on the coarse tree T, each column of the annotation is independent of every other column. However, they are crucially not independent conditioned only on the sentence $\mathbf{w}$. This model is represented schematically in Figure 2(a).

The conditional probability $\mathrm{P}(T[X]|\mathbf{w}, \theta)$ of an annotated tree given words is:

$$\begin{aligned}
&\mathrm{P}(T[X]|\mathbf{w}, \theta) \\
&\quad = \frac{\prod_m f_m(T[X_m]; \mathbf{w}, \theta_m)}{\sum_{T', X'} \prod_m f_m(T'[X'_m]; \mathbf{w}, \theta_m)} \\
&\quad = \frac{1}{Z(\mathbf{w}, \theta)} \prod_m f_m(T[X_m]; \mathbf{w}, \theta_m)
\end{aligned} \tag{1}$$

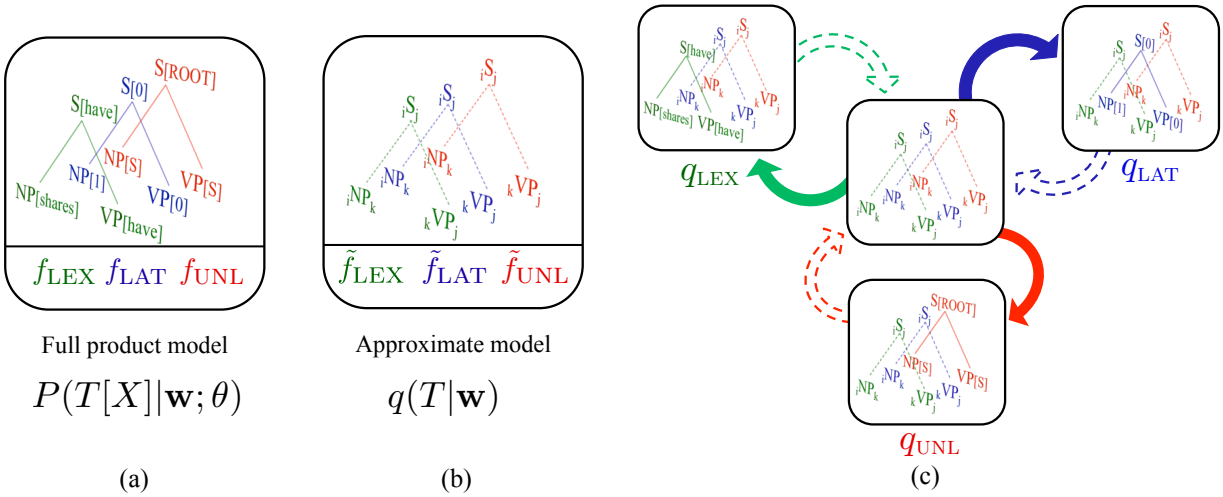
where the factors f_m for each model take the form:

$$f_m(T[X_m]; \mathbf{w}, \theta_m) = \exp\left(\theta_m^T \varphi_m(T, X_m, \mathbf{w})\right)$$

Here, X_m is the annotation associated with a particular model m. φ is a feature function that projects the raw tree, annotations, and words into a feature vector. The features φ need to decompose into features for each factor f_m; we do not allow features that take into account the annotation from two different components.

We further add a pruning filter that assigns zero weight to any tree with a constituent that a baseline unannotated grammar finds sufficiently unlikely, and a weight of one to any other tree. This filter is similar to that used in Petrov and Klein (2008a) and allows for much more efficient training and inference.

Because our model is discriminative, training takes the form of maximizing the probability of the training trees given the words. This objective is convex for deterministic annotations, but non-convex for latent annotations. We (locally) optimize the



Full product model

$P(T[X]|\mathbf{w};\theta)$

(a)

Approximate model

$q(T|\mathbf{w})$

(b)

(c)

Figure 2: Schematic representation of our model, its approximation, and expectation propagation. (a) The full joint distribution consists of a product of three grammars with different annotations, here lexicalized, latent, and unlexicalized. This model is described in Section 3.2. (b) The core approximation is an anchored PCFG with one factor corresponding to each annotation component, described in Section 5.1. (c) Fitting the approximation with expectation propagation, as described in Section 5.3. At the center is the core approximation. During each step, an "augmented" distribution q_m is created by taking one annotation factor from the full grammar and the rest from the approximate grammar. For instance, in upper left hand corner the full f_{LEX} is substituted for $\tilde{f}_{\text{LEX}}$. This new augmented distribution is projected back to the core approximation. This process is repeated for each factor until convergence.

(non-convex) log conditional likelihood of the observed training data $(T^{(d)}, \mathbf{w}^{(d)})$:

$$\begin{aligned}\ell(\theta) &= \sum_d \log \mathrm{P}(T^{(d)}|\mathbf{w}^{(d)},\theta) \\ &= \sum_d \log \sum_X \mathrm{P}(T^{(d)}[X]|\mathbf{w}^{(d)},\theta)\end{aligned} \tag{2}$$

Using standard results, the derivative takes the form:

$$\begin{aligned}\nabla\ell(\theta) = &\sum_d E[\varphi(T,X,\mathbf{w})|T^{(d)},\mathbf{w}^{(d)}] \\ &- \sum_d E[\varphi(T,X,\mathbf{w})|\mathbf{w}^{(d)}]\end{aligned} \tag{3}$$

The first half of this derivative can be obtained by the forward/backward-like computation defined by Matsuzaki et al. (2005), while the second half requires an inside/outside computation (Petrov and Klein, 2008a). The partition function $Z(\mathbf{w},\theta)$ is computed as a byproduct of the latter computation. Finally, this objective is regularized, using the L2 norm of θ as a penalty.

We note that we omit from our parser one major feature class found in other discriminative parsers, namely those that use features over the words in the span (Finkel et al., 2008; Petrov and Klein, 2008b). These features might condition on words on either side of the split point of a binary rule or take into account the length of the span. While such features have proven useful in previous work, they are not the focus of our current work and so we omit them.

4 The Complexity of Annotated Grammars

Note that the first term of Equation 3—which is conditioned on the coarse tree T—factors into M pieces, one for each of the annotation components. However, the second term does not factor because it is conditioned on just the words $\mathbf{w}$. Indeed, naïvely computing this term requires parsing with the fully articulated grammar, meaning that inference would be no more efficient than parsing with non-factored annotations.

Standard algorithms for parsing run in time $O(G|\mathbf{w}|^3)$, where $|\mathbf{w}|$ is the length of the sentence, and G is the size of the grammar, measured in the number of (binary) rules. Let G_0 be the number of binary rules in the unannotated "base" grammar.

Suppose that we have M annotation components. Each annotation component can have up to A primitive annotations per rule. For instance, a latent variable grammar will have $A = 8^b$ where b is the number of bits of annotation. If we compile all annotation components into unstructured annotations, we can end up with a total grammar size of $O(A^M G_0)$, and so in general parsing time scales exponentially with the number of annotation components. Thus, if we use latent annotations and the hierarchical splitting approach of Petrov et al. (2006), then the grammar has size $O(8^S G_0)$, where S is the number of times the grammar was split in two. Therefore, the size of annotated grammars can reach intractable levels very quickly, particularly in the case of latent annotations, where all combinations of annotations are possible.

Petrov (2010) considered an approach to slowing this growth down by using a set of M independently trained parsers P_m, and parsed using the product of the scores from each parser as the score for the tree. This approach worked largely because training was intractable: if the training algorithm could reach the global optimum, then this approach might have yielded no gain. However, because the optimization technique is local, the same algorithm produced multiple grammars.

In what follows, we propose another solution that exploits the factored structure of our grammar with expectation propagation. Crucially, we are able to jointly train and parse with all annotation factors, minimizing redundancy across the models. While not exact, we will see that expectation propagation is indeed effective.

5 Factored Inference

The key insight behind the approximate inference methods we consider here is that the full model is a product of complex factors that interact in complicated ways, and we will approximate it with a product of corresponding simple factors that interact in simple ways. Since each annotation factor is a reasonable model in both power and complexity on its own, we can consider them one at a time, replacing all others with their approximations, as shown in Figure 2(c).

The way we will build these approximations is with *expectation propagation* (Minka, 2001). Expectation propagation (EP) is a general method for approximate inference that generalizes belief propagation. We describe it here, but we first try to provide an intuition for how it functions in our system. We also describe a simplified version of EP, called *assumed density filtering* (Boyen and Koller, 1998), which is somewhat easier to understand and rhetorically convenient. For a more detailed introduction to EP in general, we direct the reader to either Minka (2001) or Wainwright and Jordan (2008). Our treatment most resembles the former.

5.1 Factored Approximations

Our goal is to build an approximation that takes information from all components into account. To begin, we note that each of these components captures different phenomena: an unlexicalized grammar is good at capturing structural relationships in a parse tree (e.g. subject noun phrases have different distributions than object noun phrases), while a lexicalized grammar captures preferred attachments for different verbs. At the same time, each of these component grammars can be thought of as a refinement of the raw unannotated treebank grammar. By itself, each of these grammars induces a different posterior distribution over *unannotated* trees for each sentence. If we can approximate each model's contribution by using only unannotated symbols, we can define an algorithm that avoids the exponential overhead of parsing with the full grammar, and instead works with each factor in turn.

To do so, we define a sentence specific *core approximation* over unannotated trees $q(T|\mathbf{w}) = \prod_m \tilde{f}_m(T, \mathbf{w})$. Figure 2(b) illustrates this approximation. Here, $q(T)$ is a product of M structurally identical factors, one for each of the annotated components. We will approximate each model f_m by its corresponding $\tilde{f}_m$. Thus, there is one color-coordinated approximate factor for each component of the model in Figure 2(a).

There are multiple choices for the structure of these factors, but we focus on *anchored PCFGs*. Anchored PCFGs have productions of the form ${}_iA_j \rightarrow {}_iB_k\ {}_kC_j$, where i, k, and j are indexes into the sentence. Here, ${}_iA_j$ is a symbol representing building the base symbol A over the span $[i, j]$.

Billott and Lang (1989) introduced anchored

CFGs as "shared forests," and Matsuzaki et al. (2005) have previously used these grammars for finding an approximate one-best tree in a latent variable parser. Note that, even though an anchored grammar is unannotated, because it is sentence specific it can represent many complex properties of the full grammar's posterior distribution for a given sentence. For example, it might express a preference for whether a PP token attaches to a particular verb or to that verb's object noun phrase in a particular sentence.

Before continuing, note that a pointwise product of anchored grammars is still an anchored grammar. The complexity of parsing with a product of these grammars is therefore no more expensive than parsing with just one. Indeed, anchoring adds no inferential cost at all over parsing with an unannotated grammar: the anchored indices i, j, k have to be computed just to parse the sentence at all. This property is crucial to EP's efficiency in our setting.

5.2 Assumed Density Filtering

We now describe a simplified version of EP: parsing with assumed density filtering (Boyen and Koller, 1998). We would like to train a sequence of M models, where each model is trained with knowledge of the posterior distribution induced by the previous models. Much as boosting algorithms (Freund and Schapire, 1995) work by focusing learning on as-yet-unexplained data points, this approach will encourage each model to improve on earlier models, albeit in a different formal way.

At a high level, assumed density filtering (ADF) proceeds as follows. First, we have an initially uninformative q: it assigns the same probability to all unpruned trees for a given sentence. Then, we factor in one of the annotated grammars and parse with this new augmented grammar. This gives us a new posterior distribution for this sentence over trees annotated with just that annotation component. Then, we can marginalize out the annotations, giving us a new q that approximates the annotated grammar as closely as possible without using any annotations. Once we have incorporated the current model's component, we move on to the next annotated grammar, augmenting it with the new q, and repeating. In this way, information from all grammars is incorporated into a final posterior distribution over trees using only unannotated symbols. The algorithm is then as follows:

- Initialize $q(T)$ uniformly.
- For each m in sequence:
  1. Create the augmented distribution $q_m(\mathbf{T}[X_m]) \propto q(\mathbf{T}) \cdot f_m(\mathbf{T}[X_m])$ and compute inside and outside scores.
  2. Minimize $\mathrm{D}_{\mathrm{KL}}\left(q_m(T)||\tilde{f}_m(T)q(T)\right)$ by fitting an anchored grammar $\tilde{f}_m$.
  3. Set $q(T) = \prod_{m'=1}^{m} \tilde{f}_{m'}(T)$.

Step 1 of the inner loop forms an approximate posterior distribution using f_m, which is the parsing model associated with component m, and q, which is the anchored core approximation to the posterior induced by the first $m-1$ models. Then, the marginals are computed, and the new posterior distribution is projected to an anchored grammar, creating $\tilde{f}_m$. More intuitively, we create an anchored PCFG that makes the approximation "as close as possible" to the augmented grammar. (We describe this procedure more precisely in Section 5.4.) Thus, each term f_m is approximated in the context of the terms that come before it. This contextual approximation is essential: without it, ADF would approximate the terms independently, meaning that no information would be shared between the models. This method would be, in effect, a simple method for parser combination, not all that dissimilar to the method proposed by Petrov (2010). Finally, note that the same inside and outside scores computed in the loop can be used to compute the expected counts needed in Equation 3.

Now we consider the runtime complexity of this algorithm. If the maximum number of annotations per rule for any factor is A, ADF has complexity $O\left(MAG_0|\mathbf{w}|^3\right)$ when using M factors. In contrast, parsing with the fully annotated grammar would have complexity $O\left(A^M G_0|\mathbf{w}|^3\right)$. Critically, for a latent variable parser with M annotation bits, the exact algorithm takes time exponential in M, while this approximate algorithm takes time linear in M.

It is worth pausing to consider what this algorithm does during training. At each step, we have

in q an approximation to what the posterior distribution looks like with the first $m-1$ models. In some places, q will assign high probabilities to spans in the gold tree, and in some places it will not be so accurate. θ_m will be particularly motivated to correct the latter, because they are less like the gold tree. On the other hand, θ_m will ignore the other "correct" segments, because q has already sufficiently captured them.

5.3 Expectation Propagation

While this sequential algorithm gives us a way to efficiently combine many kinds of annotations, it is not a fully joint algorithm: there is no backward propagation of information from later models to earlier models. Ideally, no model should be privileged over any other. To correct that, we use EP, which is essentially the iterative generalization of ADF.

Intuitively, EP cycles among the models, updating the approximation for that model in turn so that it closely resembles the predictions made by f_m in the context of all other approximations, as in Figure 2(c). Thus, each approximate term $\tilde{f}_m$ is created using information from all other $\tilde{f}_{m'}$, meaning that the different annotation factors can still "talk" to each other. The product of these approximations q will therefore come to act as an approximation to the true posterior: it takes into account joint information about all annotation components, all within one tractable anchored grammar.

With that intuition in mind, EP is defined as follows:

- Initialize contributions $\tilde{f}_m$ to the approximate posterior q.
- At each step, choose m.
  1. Include approximations to all factors other than m: $q^{\backslash m}(T) = \prod_{\mathrm{m}' \neq \mathrm{m}} \tilde{f}_{m'}(T)$.
  2. Create the augmented distribution by including the actual factor for component m
     $q_m(T[X_m]) \propto f_m(T[X_m]) q^{\backslash m}(T)$
     and compute inside and outside scores.
  3. Create a new $\tilde{f}_m(T)$ that minimizes $\mathrm{D}_{\mathrm{KL}}\left(q_m(T) || \tilde{f}_m(T) q^{\backslash m}(T)\right)$.
- Finally, set $q(T) \propto \prod_m \tilde{f}_m(T)$.

Step 2 creates the augmented distribution q_m, which includes f_m along with the approximate factors for all models except the current model. Step 3 creates a new anchored $\tilde{f}_m$ that has the same marginal distribution as the true model f_m in the context of the other approximations, just as we did in ADF.

In practice, it is usually better to not recompute the product of all $\tilde{f}_m$ each time, but instead to maintain the full product $q(T) \propto \prod_m \tilde{f}_m$ and to remove the appropriate $\tilde{f}_m$ by division. This optimization is analogous to belief propagation, where messages are removed from beliefs by division, instead of recomputing beliefs on the fly by multiplying all messages.

Schematically, the whole process is illustrated in Figure 2(c). At each step, one piece of the core approximation is replaced with the corresponding component from the full model. This augmented model is then reapproximated by a new core approximation q after updating the corresponding $\tilde{f}_m$. This process repeats until convergence.

5.4 EPIC Parsing

In our parser, EP is implemented as follows. q and each of the $\tilde{f}_m$ are anchored grammars that assign weights to unannotated rules. The product of anchored grammars with the annotated factor f_m need not be carried out explicitly. Instead, note that an anchored grammar is just a function $q(A \to B\ C, i, k, j) \in \mathbb{R}^+$ that returns a score for every anchored binary rule. This function can be easily integrated into the CKY algorithm for a single annotated grammar by simply multiplying in the value of q whenever computing the score of the respective production over some span. The modified inside recurrence takes the form:

$$\begin{aligned}
&\textsc{inside}(A[x], i, j) \\
&\quad = \sum_{B,y,C,z} \theta^T \varphi(A[x] \to B[y]\ C[z], \mathbf{w}) \\
&\qquad \cdot \sum_{i<k<j} \textsc{inside}(B[y], i, k) \cdot \textsc{inside}(C[z], k, j) \\
&\qquad\quad \cdot q(A \to B\ C, i, k, j)
\end{aligned} \tag{4}$$

Thus, parsing with a pointwise product of an anchored grammar and an annotated grammar has no increased combinatorial cost over parsing with just the annotated grammar.

To actually perform the projection in step 3 of EP, we create an anchored grammar from inside and outside probabilities. First, we compute the expected number of times the rule ${}_iA_j \rightarrow {}_iB_k\ {}_kC_j$ occurs, and then then we locally normalize for each symbol ${}_iA_j$. This actually creates the new q distribution, and so we have to divide out $q^{\backslash m}$ This process minimizes KL divergence subject to the local normalization constraints.

All in all, this gives an algorithm that takes time $O\left(IMAG_0|\mathbf{w}|^3\right)$, where I is the maximum number of iterations, M is the number of models, and A is the maximum number of annotations for any given rule.

5.5 Other Inference Algorithms

To our knowledge, expectation propagation has been used only once in the NLP community; Daumé III and Marcu (2006) employed an unstructured version in a Bayesian model of extractive summarization. Therefore, it is worth describing how EP differs from more familiar techniques.

EP can be thought of as a more flexible generalization of belief propagation, which has been used several times in NLP (Smith and Eisner, 2008; Niehues and Vogel, 2008; Cromières and Kurohashi, 2009; Burkett and Klein, 2012). In particular, EP allows for the arbitrary choice of messages (the $\tilde{f}_m$), meaning that we can use structured messages like anchored PCFGs.

Mean field (Saul and Jordan, 1996) is another approximate inference technique that allows for structured approximations (Xing et al., 2003; Burkett et al., 2010), but here the natural version of mean field for our model would still be intractable. However, it is possible to adapt mean field into allowing for tractable updates that are similar to the ones we proposed. We do not pursue that approach here.

Dual decomposition (Dantzig and Wolfe, 1960; Komodakis et al., 2007) has recently become popular in the community (Rush et al., 2010; Koo et al., 2010). In fact, EP can be seen as a particular kind of dual decomposition of the log normalization constant $\log Z(\mathbf{w}, \theta)$ that is optimized with message passing rather than (sub-)gradient descent or LP relaxations. Indeed, Minka (2001) argues that the EP objective is more efficiently optimized with message passing than with gradient updates. This assertion should be examined for the structured models common in NLP, but that is beyond the scope of this paper.

Finally, note that EP, like belief propagation but unlike mean field, is not guaranteed to converge, though in practice it usually seems to. In our experiments, typically three or four iterations are enough for almost all sentences to reach convergence, and we found no loss in cutting off the number of iterations to four.

6 Experiments

In what follows, we describe three experiments. First, in a small experiment, we examine how effective the different inference algorithms are for both training and testing. Second, we scale up our latent variable model into successively larger products. Finally, we present a selection of the many possible model combinations, showing that combining latent and expert annotation can be quite effective.

6.1 Experimental Setup

For our experiments, we trained and tested on the Penn Treebank using the standard splits: sections 2-21 were training, 22 development, and 23 testing. In preliminary experiments, we report development set F1 on sentences up to length 40. For our final test set experiment, we report F1 on sentences from section 23 up to length 40, as well as all sentences from that section. Scores reported are computed using EVALB (Sekine and Collins, 1997). We binarize trees using Collins' head rules (Collins, 1997).

Each discriminative parser was trained using the Adaptive Gradient variant of Stochastic Gradient Descent (Duchi et al., 2010). Smaller models were seeded from larger models. That is, before training a grammar of 5 models with 1 latent bit each, we started with weights from a parser with 4 factored bits. Initial experiments suggested this step did not affect final performance, but greatly decreased total training time, especially for the latent variable parsers. For extracting a one-best tree, we use a version of the Max-Recall algorithm of Goodman (1996). When using EP or ADF, we initialized the core approximation q to the uniform distribution over unpruned trees.

	Parsing			
Training	ADF	EP	Exact	Petrov
ADF	84.3	84.5	84.5	82.5
EP	84.1	84.6	84.5	78.7
Exact	83.8	84.5	**84.9**	81.5
Indep.	82.3	82.1	82.2	82.6

Table 1: The effect of algorithm choice for training and parsing on a product of two 2-state parsers on F1. Petrov is the product parser of Petrov (2010), and Indep. refers to independently trained models. For comparison, a four-state parser achieves a score of 83.2.

When counting parameters, we consider the number of parameters per binary rule. Hence, a single four-state latent model would have 64 (= 4^3) parameters per rule, while a product of 5 two-state models would have just 40 (= $5 \cdot 2^3$).

6.2 Comparison of Inference Algorithms

In our first experiment, we test the relative performance of the various approximate inference methods at both train and test time. In order to include exact inference, we necessarily need to look at a smaller scale example for which exact inference is still feasible. We examined development performance for training and inference on a small product of two parsers, each with two latent states per symbol.

During training, we have several options. We can use exact training by parsing with the fully articulated product of both grammars, or, we can instead use EP, ADF, or independent training. At test time, we can parse using the full product of both grammars, or, we can instead use EP, ADF, or we can use the method of Petrov (2010) wherein we multiply the parsers together in an *ad hoc* fashion.

The results are in Table 1. The best reported score, unsurprisingly, is for using exact training and parsing, but using EP for training and parsing results in a relatively small loss of 0.3 F1. ADF, however, suffers a loss of 0.6 F1 over Exact when used for training and parsing. Otherwise, Exact and EP seem to perform fairly similarly at parse time for all training conditions.

In general, there seems to be a gain for using the same method for training and testing. Each testing method performs at its best when using models trained with the same method. Moreover, except for ADF, the converse holds true: the grammars trained with a given parsing method are best decoded using the same method.

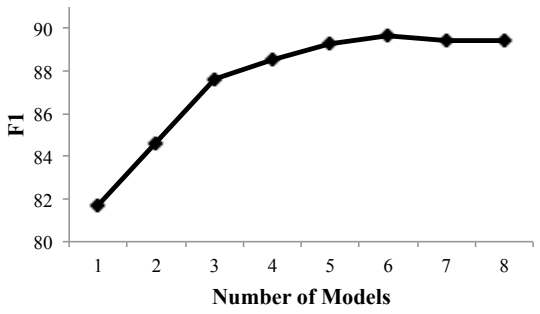


Figure 3: Development F1 plotted against the number M of one-bit latent annotation components. The best grammar has 6 one-bit annotations, with 89.7 F1.

Oddly, using Petrov (2010)'s method does not seem to work well at all for jointly trained models, except for ADF. Similarly, joint parsing underperforms Petrov (2010)'s method when using independently trained models. Likely, the joint parsing algorithms are miscalibrating the redundant information present in the two independently-trained models, while the two jointly-trained components come to depend on each other. In fact, the F1 scores for the two separate models of the EP parser are in the 60's.

As expected, ADF does not perform as well as EP. Therefore, we exclude it from our subsequent experiments, focusing exclusively on EP.

6.3 Latent Variable Experiments

Most of the previous work in latent variable parsing has focused on splitting smaller unstructured annotations into larger unstructured annotations. Here, we consider training a joint model consisting of a large number of disjoint one-bit (i.e. two-state) latent variable annotations. Specifically, we consider the performance of products of up to 8 one-bit annotations.

In Figure 3, we show development F1 as a function of the number of latent bits. Improvement is roughly linear up to 3 components. Performance levels off afterwards, with the top performing system scoring 89.7 F1. Nevertheless, these parsers outperform the comparable parsers of Petrov and Klein (2008a) (89.3), even though our six-bit parser has many fewer effective parameters per binary rule:

Models	F1, ≤ 40	F1, All
Lexicalized	87.3	86.5
Unlexicalized	86.3	85.4
3xLatent	88.6	87.6
Lex+Unlex	90.2	89.5
Lex+Lat	90.0	89.4
Unlex+Lat	90.0	89.4
Lex+Unlex+Lat	90.2	89.7

Table 2: Development F1 score for various model combinations for sentences less than length 40 and all sentences. 3xLatent refers to a latent annotation model with 3 factored latent bits.

48 instead of the 4096 in their best parser. We also ran our best system on Section 23, where it gets 89.1 and 88.4 on sentences less than length 40 and on all sentences, respectively. This result compares favorably to the 88.8/88.3 of Petrov and Klein (2008a).

6.4 Heterogeneous Models

We now consider factored models with different kinds of annotations. Specifically, we tested grammars comprising all subsets of {Lexicalized, Unlexicalized, Latent}. We used a model with 3 factored bits as our representative of the latent variable class, because it was closest in performance to the other models. Of course, other smaller and larger combinations are possible, but we found this selection to be representative.

The development results are in Table 2. Unsurprisingly, adding more kinds of annotations helps for the most part, though the combination of all three components is not much better than a combination of just the lexicalized and unlexicalized models. Indeed, our best systems involved combining the lexicalized model with some other model. This is probably because the lexicalized model can represent very different syntactic relationships than the latent and unlexicalized models, meaning there is more diversity in the joint model's capacity when using combinations involving the lexicalized annotations.

Finally, we ran our best system (the fully combined one) on Section 23 of the Penn Treebank. It scored 90.1/89.4 F1 on length 40 and all sentences respectively, slightly edging out the 90.0/89.3 F1 of Petrov and Klein (2008a). However, it is not quite as good at exact match: 37.7/35.3 vs 40.1/37.7. Note, though, that their parser makes use of span features, which deliver a gain of +0.3/0.2F1 respectively, while ours does not. We suspect that similar gains could be had by incorporating these features, but we leave that for future work.

7 Conclusion

Factored representations capture a fundamental linguistic insight: grammatical categories are not monolithic, unanalyzable entities. Instead, they are composed of numerous facets that together govern how categories combine into parse trees.

We have developed a new model for grammars with factored annotations and presented two methods for parsing with these grammars. Our experiments have demonstrated that our approach produces higher performance parsers with many fewer parameters. Moreover, our model works with both latent and explicit annotations, allowing us to combine linguistic knowledge with machine learning. Finally, our source code is available at `http://nlp.cs.berkeley.edu/Software.shtml`.

Acknowledgments

We would like to thank Slav Petrov, David Burkett, Adam Pauls, Greg Durrett and the anonymous reviewers for helpful comments. We would also like to thank Daphne Koller for originally suggesting the assumed density filtering approach. This work was partially supported by BBN under DARPA contract HR0011-12-C-0014, and by an NSF fellowship to the first author.

References

Sylvie Billott and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June.

Xavier Boyen and Daphne Koller. 1998. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence—UAI 1998*, pages 33–42. San Francisco: Morgan Kaufmann.

David Burkett and Dan Klein. 2012. Fast inference in phrase extraction models with belief propagation. In *NAACL*.

David Burkett, John Blitzer, and Dan Klein. 2010. Joint parsing and alignment with weakly synchronized grammars. In *NAACL*.

Noam Chomsky. 1992. *A minimalist program for linguistic theory*, volume 1. MIT Working Papers in Linguistics, MIT, Cambridge Massachusetts.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL*, pages 16–23.

Fabien Cromières and Sadao Kurohashi. 2009. An alignment algorithm using belief propagation and a structure-based distortion model. In *EACL*.

G. B. Dantzig and P. Wolfe. 1960. Decomposition principle for linear programs. *Operations Research*, 8:101–111.

Hal Daumé III and Daniel Marcu. 2006. Bayesian query-focused summarization. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, Sydney, Australia.

Markus Dreyer and Jason Eisner. 2006. Better informed training of latent syntactic features. In *EMNLP*, pages 317–326, July.

John Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *COLT*.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *ACL 2008*, pages 959–967.

Yoav Freund and Robert E. Schapire. 1995. A decision-theoretic generalization of on-line learning and an application to boosting.

Joshua Goodman. 1996. Parsing algorithms and metrics. In *ACL*, pages 177–183.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL*, pages 423–430.

Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. 2007. Mrf optimization via dual decomposition: Message-passing revisited. In *ICCV*, pages 1–8.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *ACL*, pages 75–82, Morristown, NJ, USA.

Thomas P. Minka. 2001. Expectation propagation for approximate Bayesian inference. In *UAI*, pages 362–369.

Jan Niehues and Stephan Vogel. 2008. Discriminative word alignment via alignment matrix modeling. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 18–25, June.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *NAACL-HLT*, April.

Slav Petrov and Dan Klein. 2008a. Discriminative log-linear grammars with latent variables. In *NIPS*, pages 1153–1160.

Slav Petrov and Dan Klein. 2008b. Sparse multi-scale grammars for discriminative latent variable parsing. In *EMNLP*, pages 867–876, Honolulu, Hawaii, October.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July.

Slav Petrov. 2010. Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Los Angeles, California, June.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *EMNLP*, pages 1–11, Cambridge, MA, October.

Lawrence Saul and Michael Jordan. 1996. Exploiting tractable substructures in intractable networks. In *NIPS 1995*.

Satoshi Sekine and Michael J. Collins. 1997. Evalb – bracket scoring program.

David A. Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *EMNLP*, pages 145–156, Honolulu, October.

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

Eric P. Xing, Michael I. Jordan, and Stuart J. Russell. 2003. A generalized mean field algorithm for variational inference in exponential families. In *UAI*, pages 583–591.