Products of Random Latent Variable Grammars

Slav Petrov
Google Research
New York, NY, 10011
slav@google.com

Abstract

We show that the automatically induced latent variable grammars of Petrov et al. (2006) vary widely in their underlying representations, depending on their EM initialization point. We use this to our advantage, combining multiple automatically learned grammars into an unweighted product model, which gives significantly improved performance over state-of-the-art individual grammars. In our model, the probability of a constituent is estimated as a product of posteriors obtained from multiple grammars that differ only in the random seed used for initialization, without any learning or tuning of combination weights. Despite its simplicity, a product of eight automatically learned grammars improves parsing accuracy from 90.2% to 91.8% on English, and from 80.3% to 84.5% on German.

1 Introduction

Learning a context-free grammar for parsing requires the estimation of a more highly articulated model than the one embodied by the observed treebank. This is because the naive treebank grammar (Charniak, 1996) is too permissive, making unrealistic context-freedom assumptions. For example, it postulates that there is only one type of noun phrase (NP), which can appear in all positions (subject, object, etc.), regardless of case, number or gender. As a result, the grammar can generate millions of (incorrect) parse trees for a given sentence, and has a flat posterior distribution. High accuracy grammars therefore add soft constraints on the way categories can be combined, and enrich the label set with additional information. These constraints can be lexicalized (Collins, 1999; Charniak, 2000), unlexicalized (Johnson, 1998; Klein and Manning, 2003b) or automatically learned (Matsuzaki et al., 2005; Petrov et al., 2006). The constraints serve the purpose of weakening the independence assumptions, and reduce the number of possible (but incorrect) parses.

Here, we focus on the latent variable approach of Petrov et al. (2006), where an Expectation Maximization (EM) algorithm is used to induce a hierarchy of increasingly more refined grammars. Each round of refinement introduces new constraints on how constituents can be combined, which in turn leads to a higher parsing accuracy. However, EM is a local method, and there are no guarantees that it will find the same grammars when initialized from different starting points. In fact, it turns out that even though the final performance of these grammars is consistently high, there are significant variations in the learned refinements.

We use these variations to our advantage, and treat grammars learned from different random seeds as independent and equipotent experts. We use a product distribution for joint prediction, which gives more peaked posteriors than a sum, and enforces all constraints of the individual grammars, without the need to tune mixing weights. It should be noted here that our focus is on improving parsing performance using a single underlying grammar class, which is somewhat orthogonal to the issue of parser combination, that has been studied elsewhere in the literature (Sagae and Lavie, 2006; Fossum and Knight, 2009; Zhang et al., 2009). In contrast to that line of work, we also do not restrict ourselves to working with k-best output, but work directly with a packed forest representation of the posteriors, much in the spirit of Huang (2008), except that we work with several forests rather than rescoring a single one.

In our experimental section we give empirical answers to some of the remaining theoretical questions. We address the question of averaging versus multiplying classifier predictions, we investigate different ways of introducing more diversity into the underlying grammars, and also compare combining partial (constituent-level) and complete (tree-level) predictions. Quite serendipitously, the simplest approaches work best in our experiments. A product of eight latent variable grammars, learned on the same data, and only differing in the seed used in the random number generator that initialized EM, improves parsing accuracy from 90.2% to 91.8% on English, and from 80.3% to 84.5% on German. These parsing results are even better than those obtained by discriminative systems which have access to additional non-local features (Charniak and Johnson, 2005; Huang, 2008).

2 Latent Variable Grammars

Before giving the details of our model, we briefly review the basic properties of latent variable grammars. Learning latent variable grammars consists of two tasks: (1) determining the data representation (the set of context-free productions to be used in the grammar), and (2) estimating the parameters of the model (the production probabilities). We focus on the randomness introduced by the EM algorithm and refer the reader to Matsuzaki et al. (2005) and Petrov et al. (2006) for a more general introduction.

2.1 Split & Merge Learning

Latent variable grammars split the coarse (but observed) grammar categories of a treebank into more fine-grained (but hidden) subcategories, which are better suited for modeling the syntax of natural languages (e.g. NP becomes NP_1 through NP_k). Accordingly, each grammar production A→BC over observed categories A,B,C is split into a set of productions $\mathrm{A}_x \rightarrow \mathrm{B}_y\mathrm{C}_z$ over hidden categories A_x,B_y,C_z. Computing the joint likelihood of the observed parse trees T and sentences w requires summing over all derivations t over split subcategories:

$$\prod_i \mathrm{P}(w_i, T_i) = \prod_i \sum_{t:T_i} \mathrm{P}(w_i, t) \qquad (1)$$

Matsuzaki et al. (2005) derive an EM algorithm for maximizing the joint likelihood, and Petrov et al. (2006) extend this algorithm to use a split&merge procedure to adaptively determine the optimal number of subcategories for each observed category. Starting from a completely markovized X-Bar grammar, each category is split in two, generating eight new productions for each original binary production. To break symmetries, the production probabilities are perturbed by 1% of random noise. EM is then initialized with this starting point and used to climb the highly non-convex objective function given in Eq. 1. Each splitting step is followed by a merging step, which uses a likelihood ratio test to reverse the least useful half of the splits. Learning proceeds by iterating between those two steps for six rounds. To prevent overfitting, the production probabilities are linearly smoothed by shrinking them towards their common base category.

2.2 EM induced Randomness

While the split&merge procedure described above is shown in Petrov et al. (2006) to reduce the variance in final performance, we found after closer examination that there are substantial differences in the patterns learned by the grammars. Since the initialization is not systematically biased in any way, one can obtain different grammars by simply changing the seed of the random number generator. We trained 16 different grammars by initializing the random number generator with seed values 1 through 16, but without biasing the initialization in any other way. Figure 1 shows that the number of subcategories allocated to each observed category varies significantly between the different initialization points, especially for the phrasal categories. Figure 2 shows posteriors over the most frequent subcategories given their base category for the first four grammars. Clearly, EM is allocating the latent variables in very different ways in each case.

As a more quantitative measure of difference,[1] we evaluated all 16 grammars on sections 22 and 24 of the Penn Treebank. Figure 3 shows the performance on those two sets, and reveals that there is no single grammar that achieves the best score on both. While the parsing accuracies are consistently high,[2] there

[1]While cherry-picking similarities is fairly straight-forward, it is less obvious how to quantify differences.

[2]Note that despite their variance, the performance is always higher than the one of the lexicalized parser of Charniak (2000).

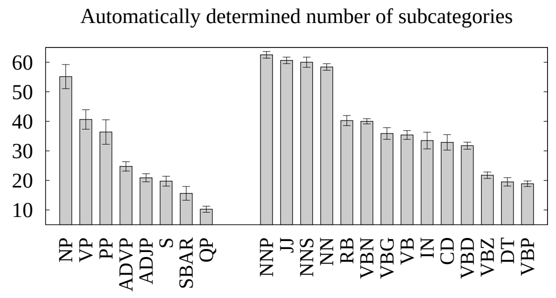

Figure 1: There is large variance in the number of subcategories (error bars correspond to one standard deviation).

is only a weak correlation between the accuracies on the two evaluation sets (Pearson coefficient 0.34). This suggests that no single grammar should be preferred over the others. In previous work (Petrov et al., 2006; Petrov and Klein, 2007) the final grammar was chosen based on its performance on a held-out set (section 22), and corresponds to the second best grammar in Figure 3 (because only 8 different grammars were trained).

A more detailed error analysis is given in Figure 4, where we show a breakdown of F_1 scores for selected phrasal categories in addition to the overall F_1 score and exact match (on the WSJ development set). While grammar G_2 has the highest overall F_1 score, its exact match is not particularly high, and it turns out to be the weakest at predicting quantifier phrases (QP). Similarly, the performance of the other grammars varies between the different error measures, indicating again that no single grammar dominates the others.

3 A Simple Product Model

It should be clear by now that simply varying the random seed used for initialization causes EM to discover very different latent variable grammars. While this behavior is worrisome in general, it turns out that we can use it to our advantage in this particular case. Recall that we are using EM to learn both, the data representation, as well as the parameters of the model. Our analysis showed that changing the initialization point results in learning grammars that vary quite significantly in the errors they make, but have comparable overall accuracies. This suggests that the different local maxima found by EM correspond to different data representations rather than to

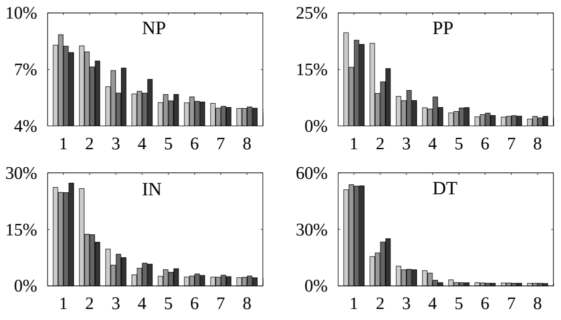

Figure 2: Posterior probabilities of the eight most frequent hidden subcategories given their observed base categories. The four grammars (indicated by shading) are populating the subcategories in very different ways.

suboptimal parameter estimates.

To leverage the strengths of the individual grammars, we combine them in a product model. Product models have the nice property that their Kullback-Liebler divergence from the true distribution will always be smaller than the average of the KL divergences of the individual distributions (Hinton, 2001). Therefore, as long as no individual grammar G_i is significantly worse than the others, we can only benefit from combining multiple latent variable grammars and searching for the tree that maximizes

$$\mathrm{P}(T|w) \propto \prod_i \mathrm{P}(T|w, \mathrm{G}_i) \tag{2}$$

Here, we are making the assumption that the individual grammars are conditionally independent, which is of course not true in theory, but holds surprisingly well in practice. To avoid this assumption, we could use a sum model, but we will show in Section 4.1 that the product formulation performs significantly better. Intuitively speaking, products have the advantage that the final prediction has a high posterior under *all* models, giving each model veto power. This is exactly the behavior that we need in the case of parsing, where each grammar has learned different constraints for ruling out improbable parses.

3.1 Learning

Joint training of our product model would couple the parameters of the individual grammars, necessitating the computation of an intractable global partition function (Brown and Hinton, 2001). Instead, we use EM to train each grammar independently,

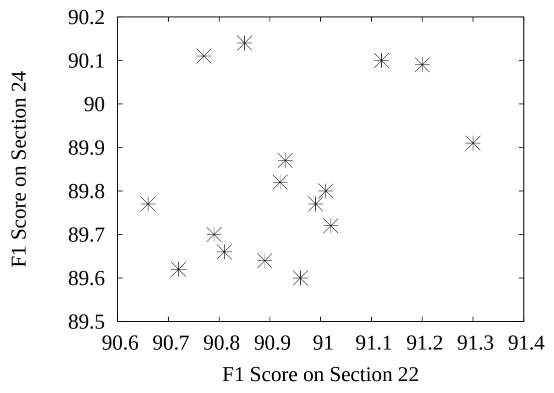

Figure 3: Parsing accuracies for grammars learned from different random seeds. The large variance and weak correlation suggest that no single grammar is to be preferred.

but from a different, randomly chosen starting point. To emphasize, we do not introduce any systematic bias (but see Section 4.3 for some experiments), or attempt to train the models to be maximally different (Hinton, 2002) – we simply train a random collection of grammars by varying the random seed used for initialization. We found in our experiments that the randomness provided by EM is sufficient to achieve diversity among the individual grammars, and gives results that are as good as more involved training procedures. Xu and Jelinek (2004) made a similar observation when learning random forests for language modeling.

Our model is reminiscent of Logarithmic Opinion Pools (Bordley, 1982) and Products of Experts (Hinton, 2001).[3] However, because we believe that none of the underlying grammars should be favored, we deliberately do not use any combination weights.

3.2 Inference

Computing the most likely parse tree is intractable for latent variable grammars (Sima'an, 2002), and therefore also for our product model. This is because there are exponentially many derivations over split subcategories that correspond to a single parse tree over unsplit categories, and there is no dynamic program to efficiently marginalize out the latent variables. Previous work on parse risk minimization has addressed this problem in two different ways: by changing the objective function, or by constraining

[3]As a matter of fact, Hinton (2001) mentions syntactic parsing as one of the motivating examples for Products of Experts.

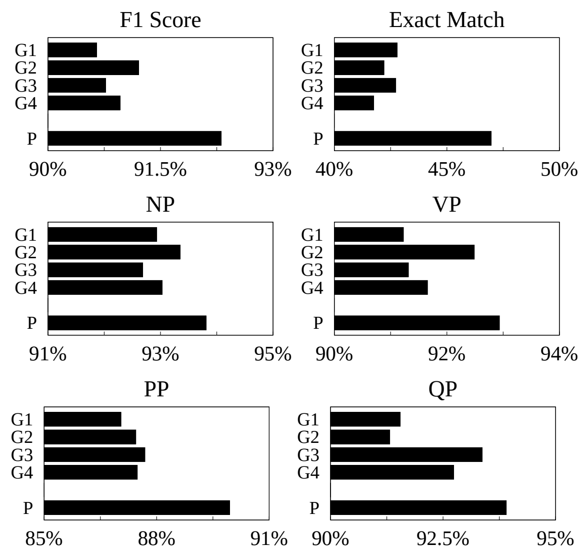

Figure 4: Breakdown of different accuracy measures for four randomly selected grammars (G_1-G_4), as well as a product model (P) that uses those four grammars. Note that no single grammar does well on all measures, while the product model does significantly better on all.

the search space (Goodman, 1996; Titov and Henderson, 2006; Petrov and Klein, 2007).

The simplest approach is to stick to likelihood as the objective function, but to limit the search space to a set of high quality candidates $\mathcal{T}$:

$$T^* = \underset{T \in \mathcal{T}}{\operatorname{argmax}} \mathrm{P}(T|w) \qquad (3)$$

Because the likelihood of a given parse tree can be computed exactly for our product model (Eq. 2), the quality of this approximation is only limited by the quality of the candidate list. To generate the candidate list, we produce k-best lists of Viterbi derivations with the efficient algorithm of Huang and Chiang (2005), and erase the subcategory information to obtain parse trees over unsplit categories. We refer to this approximation as TREE-LEVEL inference, because it considers a list of complete trees from the underlying grammars, and selects the tree that has the highest likelihood under the product model. While the k-best lists are of very high quality, this is a fairly crude and unsatisfactory way of approximating the posterior distribution of the product model, as it does not allow the synthesis of new trees based on tree fragments from different grammars.

An alternative is to use a tractable objective function that allows the efficient exploration of the entire

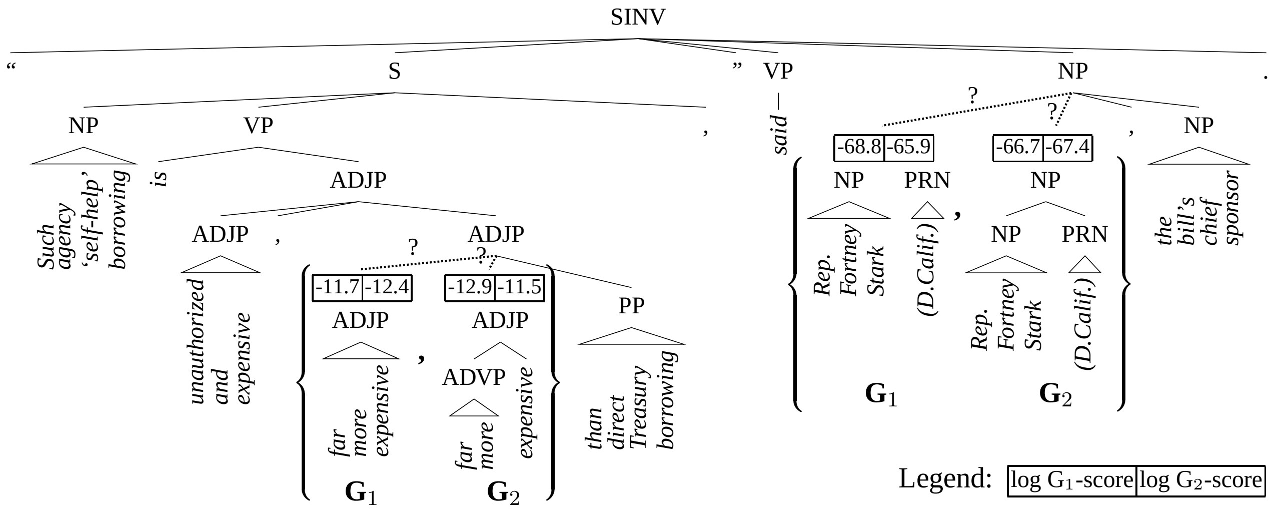

Figure 5: Grammar G_1 has a preference for flat structures, while grammar G_2 prefers deeper hierarchical structures. Both grammars therefore make one mistake each on their own. However, the correct parse tree (which uses a flat ADJP in the first slot and a hierarchical NP in the second) scores highest under the product model.

search space. Petrov and Klein (2007) present such an objective function, which maximizes the product of expected correct productions r:

$$T^* = \underset{T}{\operatorname{argmax}} \prod_{r \in T} \mathbb{E}(r|w) \tag{4}$$

These expectations can be easily computed from the inside/outside scores, similarly as in the maximum bracket recall algorithm of Goodman (1996), or in the variational approximation of Matsuzaki et al. (2005). We extend the algorithm to work over posterior distributions from multiple grammars, by aggregating their expectations into a product. In practice, we use a packed forest representation to approximate the posterior distribution, as in Huang (2008). We refer to this approximation as CONSTITUENT-LEVEL, because it allows us to form new parse trees from individual constituents.

Figure 5 illustrates a real case where the product model was able to construct a completely correct parse tree from two partially correct ones. In the example, one of the underlying grammars (G_1) had an imperfect recall score, because of its preference for flat structures (it missed an NP node in the second part of the sentence). In contrast, the other grammar (G_2) favors deeper structures, and therefore introduced a superfluous ADVP node. The product model gives each underlying grammar veto power, and picks the least controversial tree (which is the correct one in this case). Note that a sum model allows the most confident model to dominate the decision, and would chose the incorrect hierarchical ADJP construction here (as one can verify using the provided model scores).

To make inference efficient, we can use the same coarse-to-fine pruning techniques as Petrov and Klein (2007). We generate a hierarchy of projected grammars for each individual grammar and parse with each one in sequence. Because only the very last pass requires scores from the different underlying grammars, this computation can be trivially parallelized across multiple CPUs. Additionally, the first (X-Bar) pruning pass needs to be computed only once because it is shared among all grammars. Since the X-Bar pass is the bottleneck of the multipass scheme (using nearly 50% of the total processing time), the overhead of using a product model is quite manageable. It would have also been possible to use A*-search for factored models (Klein and Manning, 2003a; Sun and Tsujii, 2009), but we did not attempt this in the present work.

4 Experiments

In our experiments, we follow the standard setups described in Table 1, and use the EVALB tool for computing parsing figures. Unless noted otherwise, we use CONSTITUENT-LEVEL inference. All our experiments are based on the publicly available BerkeleyParser.[4]

[4]http://code.google.com/p/berkeleyparser

	Training Set	Dev. Set	Test Set
ENGLISH-WSJ (Marcus et al., 1993)	Sections 2-21	Section 22	Section 23
ENGLISH-BROWN (Francis et al. 1979)	see ENGLISH-WSJ	10% of the data[5]	10% of the the data[5]
GERMAN (Skut et al., 1997)	Sentences 1-18,602	Sentences 18,603-19,602	Sentences 19,603-20,602

Table 1: Corpora and standard experimental setups.

4.1 (Weighted) Product vs. (Weighted) Sum

A great deal has been written on the topic of products versus sums of probability distributions for joint prediction (Genest and Zidek, 1986; Tax et al., 2000). However, those theoretical results do not apply directly here, because we are using multiple randomly permuted models from the same class, rather models from different classes. To shed some light on this issue, we addressed the question empirically, and combined two grammars into an unweighted product model, and also an unweighted sum model. The individual grammars had parsing accuracies (F_1) of 91.2 and 90.7 respectively, and their product (91.7) clearly outperformed their sum (91.3). When more grammars are added, the gap widens even further, and the trends persist independently of whether the models use TREE-LEVEL or CONSTITUENT-LEVEL inference. At least for the case of unweighted combinations, the product distribution seems to be superior.

In related work, Zhang et al. (2009) achieve excellent results with a weighted sum model. Using weights learned on a held-out set and rescoring 50-best lists from Charniak (2000) and Petrov et al. (2006), they obtain an F_1 score of 91.0 (which they further improve to 91.4 using a voting scheme). We replicated their experiment, but used an unweighted product of the two model scores. Using TREE-LEVEL inference, we obtained an F_1 score of 91.6, suggesting that weighting is not so important in the product case, as long as the classifiers are of comparable quality.[6] This is in line with previous work on product models, where weighting has been important when combining heterogenous classifiers (Heskes, 1998), and less important when the classifiers are of similar accuracy (Smith et al., 2005).

[5]See Gildea (2001) for the exact setup.

[6]The unweighted sum model, however, underperforms the individual models with an F_1 score of only 90.3.

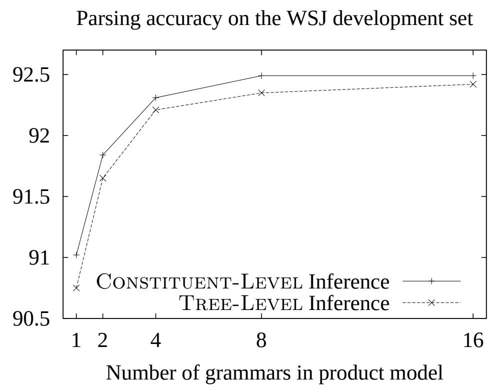

Figure 6: Adding more grammars to the product model improves parsing accuracy, while CONSTITUENT-LEVEL inference gives consistently better results.

4.2 Tree-Level vs. Constituent-Level Inference

Figure 6 shows that accuracy increases when more grammars are added to the product model, but levels off after eight grammars. The plot also compares our two inference approximations, and shows that CONSTITUENT-LEVEL inference results in a small (0.2), but consistent improvement in F_1 score.

A first thought might be that the improvement is due to the limited scope of the k-best lists. However, this is not the case, as the results hold even when the candidate set for CONSTITUENT-LEVEL inference is constrained to trees from the k-best lists. While the packed forrest representation can very efficiently encode an exponential set of parse trees, in our case the k-best lists appear to be already very diverse because they are generated by multiple grammars. Starting at 96.1 for a single latent variable grammar, merging two 50-best lists from different grammars gives an oracle score of 97.4, and adding more k-best lists further improves the oracle score to 98.6 for 16 grammars. This compares favorably to the results of Huang (2008), where the oracle score over a pruned forest is shown to be 97.8 (compared to 96.7 for a 50-best list).

The accuracy improvement can instead be explained by the change in the objective function. Recall from section Section 3.2, that CONSTITUENT-LEVEL inference maximizes the expected number of correct productions, while TREE-LEVEL inference maximizes tree-likelihood. It is therefore not too surprising that the two objective functions select the same tree only 41% of the time, even when limited to the same candidate set. Maximizing the expected number of correct productions is superior for F_1 score (see the one grammar case in Figure 6). However, as to be expected, likelihood is better for exact match, giving a score of 47.6% vs. 46.8%.

4.3 Systematic Bias

Diversity among the underlying models is what gives combined models their strength. One way of increasing diversity is by modifying the feature sets of the individual models (Baldridge and Osborne, 2008; Smith and Osborne, 2007). This approach has the disadvantage that it reduces the performance of the individual models, and is not directly applicable for latent variable grammars because the features are automatically learned. Alternatively, one can introduce diversity by changing the training distribution. Bagging (Breiman, 1996) and Boosting (Freund and Shapire, 1996) fall into this category, but have had limited success for parsing (Henderson and Brill, 2000). Furthermore boosting is impractical here, because it requires training dozens of grammars in sequence.

Since training a single grammar takes roughly one day, we opted for a different, parallelizable way of changing the training distribution. In a first experiment, we divided the training set into two disjoint sets, and trained separate grammars on each half. These truly disjoint grammars had low F_1 scores of 89.4 and 89.6 respectively (because they were trained on less data). Their combination unfortunately also achieves only an accuracy of 90.9, which is lower than what we get when training a single grammar on the entire training set. In another experiment, we used a cross-validation setup where individual sections of the treebank were held out. The resulting grammars had parsing accuracies of about 90.5, and the product model was again not able to overcome the lower starting point, despite the potentially larger diversity among the underlying grammars. It appears that any systematic bias that lowers the accuracy of the individual grammars also hurts the final performance of the product model.

4.4 Product Distribution as Smoothing

Smith et al. (2005) interpret Logarithmic Opinion Pools (LOPs) as a smoothing technique. They compare regularizing Conditional Random Fields (CRFs) with Gaussian priors (Lafferty et al., 2001), to training a set of unregularized CRFs over different feature sets and combining them in an LOP. In their experiments, both approaches work comparably well, but their combination, an LOP of regularized CRFs works best.

Not too surprisingly, we find this to be the case here as well. The parameters of each latent variable grammar are typically smoothed in a linear fashion to prevent excessive overfitting (Petrov et al., 2006). While all the experiments so far used smoothed grammars, we reran the experiments also with a set of unsmoothed grammars. The individual unsmoothed grammars have on average an 1.2% lower accuracy. Even though our product model is able to increase accuracy by combining multiple grammars, the gap to the smoothed models remains consistent. This suggests that the product model is doing more than just smoothing. In fact, because the product distribution is more peaked, it seems to be doing the opposite of smoothing.

4.5 Final Results

Our final model uses an unweighted product of eight grammars trained by initializing the random number generator with seeds 1 through 8. Table 2 shows our test set results (obtained with CONSTITUENT-LEVEL inference), and compares them to related work. There is a large body of work that has reported parsing accuracies for English, and we have grouped the different methods into categories for better overview.

Our results on the English in-domain test set are higher than those obtained by any single component parser (SINGLE). The other methods quoted in Table 2 operate over the output of one or more single component parsers and are therefore largely orthogonal to our line of work. It is nonetheless exciting to see that our product model is competitive with the discriminative rescoring methods (RE) of Charniak and Johnson (2005) and Huang (2008), achieving higher F_1 scores but lower exact match. These two methods work on top of the Charniak (2000) parser, and it would be possible to exchange that parser with our product model. We did not attempt this experiment, but we expect that those methods would stack well with our model, because they use primarily non-local features that are not available in a context-free grammar.

Techniques like self-training (SELF) and system combinations (COMBO) can further improve parsing accuracies, but are also orthogonal to our work. In particular the COMBO methods seem related to our work, but are very different in their nature. While we use multiple grammars in our work, all grammars are from the same model class for us. In contrast, those methods rely on a diverse set of individual parsers, each of which requires a significant effort to build. Furthermore, those techniques have largely relied on different voting schemes in the past (Henderson and Brill, 1999; Sagae and Lavie, 2006), and only more recently have started using actual posteriors from the underlying models (Fossum and Knight, 2009; Zhang et al., 2009). Even then, those methods operate only over k-best lists, and we are the first to work directly with parse forests from multiple grammars.

It is also interesting to note that the best results in Zhang et al. (2009) are achieved by combining k-best lists from a latent variable grammar of Petrov et al. (2006) with the self-trained reranking parser of McClosky et al. (2006). Clearly, replacing the single latent variable grammar with a product of latent variable grammars ought to improve performance.

The results on the other two corpora are similar. A product of latent variable grammars very significantly outperforms a single latent variable grammar and sets new standards for the state-of-the-art.

We also analyzed the errors of the product models. In addition to the illustrative example in Figure 5, we computed detailed error metrics for different phrasal categories. Figure 4 shows that a product of four random grammars is always better than even the best underlying grammar. The individual grammars seem to learn different sets of constraints, and the product model is able to model them all at once, giving consistent accuracy improvements across all metrics.

Type	Parser	all sentences		
		LP	LR	EX
ENGLISH-WSJ				
	This Paper	**92.0**	**91.7**	**41.9**
SINGLE	Charniak (2000)	89.9	89.5	37.2
SINGLE	Petrov and Klein (2007)	90.2	90.1	36.7
SINGLE	Carreras et al. (2008)	91.4	90.7	-
RE	Charniak et al. (2005)	91.8	91.2	44.8
RE	Huang (2008)	92.2	91.2	43.5
SELF	Huang and Harper (2009)	91.3[7]	91.5[7]	39.3[7]
SELF	McClosky et al. (2006)	92.5	92.1	45.3
COMBO	Sagae and Lavie (2006)	93.2	91.0	-
COMBO	Fossum and Knight (2009)	93.2	91.7	-
COMBO	Zhang et al. (2009)	93.3	92.0	-
ENGLISH-BROWN				
	This Paper	**86.5**	**86.3**	**35.8**
SING	Charniak (2000)	82.9	82.9	31.7
SING	Petrov and Klein (2007)	83.9	83.8	29.6
RE	Charniak et al. (2005)	86.1	85.2	36.8
GERMAN				
	This Paper	**84.5**	**84.0**	**51.2**
SING	Petrov and Klein (2007)	80.0	80.2	42.4
SING	Petrov and Klein (2008)	80.6	80.8	43.9

Table 2: Final test set accuracies for English and German.

5 Conclusions

We presented a simple product model that significantly improves parsing accuracies on different domains and languages. Our model leverages multiple automatically learned latent variable grammars, which differ only in the seed of the random number generator used to initialize the EM learning algorithm. As our analysis showed, the grammars vary widely, making very different errors. This is in part due to the fact that EM is used not only for estimating the parameters of the grammar, but also to determine the set of context-free productions that underlie it. Because the resulting data representations are largely independent, they can be easily combined in an unweighted product model. The product model does not require any additional training and is capable of significantly improving the state-of-the-art in parsing accuracy. It remains to be seen if a similar approach can be used in other cases where EM converges to widely varying local maxima.

Acknowledgements

I would like to thank Ryan McDonald for numerous discussions on this topic and his feedback on earlier versions of this paper. This work also benefited from conversations with Gideon Mann, Fernando Pereira, Dan Klein and Mehryar Mohri.

[7]Note that these results are on a modified version of the treebank where unary productions are removed.

References

J. Baldridge and M. Osborne. 2008. Active learning and logarithmic opinion pools for HPSG parse selection. *Natural Language Engineering*.

R. F. Bordley. 1982. A multiplicative formula for aggregating probability assessments. *Management Science*.

L. Breiman. 1996. Bagging predictors. *Machine Learning*.

A. Brown and G. Hinton. 2001. Products of hidden Markov models. In *AISTATS '01*.

X. Carreras, M. Collins, and T. Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *CoNLL '08*.

E. Charniak and M. Johnson. 2005. Coarse-to-Fine N-Best Parsing and MaxEnt Discriminative Reranking. In *ACL'05*.

E. Charniak. 1996. Tree-bank grammars. In *AAAI '96*.

E. Charniak. 2000. A maximum–entropy–inspired parser. In *NAACL '00*.

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, UPenn.

V. Fossum and K. Knight. 2009. Combining constituent parsers. In *NAACL '09*.

W. N. Francis and H. Kucera. 1979. Manual of information to accompany a standard corpus of present-day edited American English. Technical report, Brown University.

Y. Freund and R. E. Shapire. 1996. Experiments with a new boosting algorithm. In *ICML '96*.

C. Genest and J. V. Zidek. 1986. Combining probability distributions: A critique and an annotated bibliography. *Statistical Science*.

D. Gildea. 2001. Corpus variation and parser performance. *EMNLP '01*.

J. Goodman. 1996. Parsing algorithms and metrics. *ACL '96*.

J. Henderson and E. Brill. 1999. Exploiting diversity in natural language processing: combining parsers. In *EMNLP '99*.

J. Henderson and E. Brill. 2000. Bagging and boosting a treebank parser. In *NAACL '00*.

T. Heskes. 1998. Selecting weighting factors in logarithmic opinion pools. In *NIPS '98*.

G. Hinton. 2001. Products of experts. In *ICANN '01*.

G. Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation*.

L. Huang and D. Chiang. 2005. Better k-best parsing. In *IWPT '05*.

Z. Huang and M. Harper. 2009. Self-training PCFG grammars with latent annotations across languages. In *EMNLP '09*.

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL '08*.

M. Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24.

D. Klein and C. Manning. 2003a. A* parsing: fast exact viterbi parse selection. In *NAACL '03*.

D. Klein and C. Manning. 2003b. Accurate unlexicalized parsing. In *ACL '03*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. In *Computational Linguistics*.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic CFG with latent annotations. In *ACL '05*.

D. McClosky, E. Charniak, and M. Johnson. 2006. Effective self-training for parsing. In *NAACL '06*.

S. Petrov and D. Klein. 2007. Improved inference for unlexicalized parsing. In *NAACL '07*.

S. Petrov and D. Klein. 2008. Sparse multi-scale grammars for discriminative latent variable parsing. In *EMNLP '08*.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *ACL '06*.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *NAACL '06*.

K. Sima'an. 2002. Computatoinal complexity of probabilistic disambiguation. *Grammars*.

W. Skut, B. Krenn, T. Brants, and H. Uszkoreit. 1997. An annotation scheme for free word order languages. In *ANLP '97*.

A. Smith and M. Osborne. 2007. Diversity in logarithmic opinion pools. *Lingvisticae Investigationes*.

A. Smith, T. Cohn, and M. Osborne. 2005. Logarithmic opinion pools for conditional random fields. In *ACL '05*.

X. Sun and J. Tsujii. 2009. Sequential labeling with latent variables: An exact inference algorithm and its efficient approximation. In *EACL '09*.

D. Tax, M. Van Breukelen, R. Duin, and J. Kittler. 2000. Combining multiple classifiers by averaging or by multiplying? *Pattern Recognition*.

I. Titov and J. Henderson. 2006. Loss minimization in parse reranking. In *EMNLP '06*.

P. Xu and F. Jelinek. 2004. Random forests in language modeling. In *EMNLP '04*.

H. Zhang, M. Zhang, C. L. Tan, and H. Li. 2009. K-best combination of syntactic parsers. In *EMNLP '09*.