

Transfer of Grammatical Structure

Slav Petrov, Leon Barrett, Romain Thibaux & Dan Klein
Computer Science
University of California
Berkeley, CA 94720

December 16, 2005

Abstract

Recent research has demonstrated that PCFGs with latent annotations are an effective way to provide automated increases in parsing accuracy. We feel that they have more potential than the literature has so far demonstrated, and we further speculate that they could also provide transfer between different corpora. In this paper, we describe our efforts and show that they do indeed provide a significant level of transfer.

1 Introduction

The probabilistic context-free grammar (PCFG) formalism is the basis of most modern statistical parsers. The symbols in a PCFG encode context-freedom assumptions about statistical dependencies in the derivations of sentences, and the relative conditional probabilities of the grammar rules induce scores on trees. Compared to a basic treebank grammar ([1], F1=79.4), the grammars of high-accuracy parsers weaken independence assumptions by splitting grammar symbols and rules with either lexical ([2], F1=90.1; [3], F1=88.6) or non-lexical ([4], F1=86.3; [5], F1=86.6) conditioning information. While such splitting, or conditioning, can cause problems for statistical estimation, it can dramatically improve the accuracy of a parser. We believe that it is possible to close the performance gap between unlexicalized and lexicalized parsers. Unlexicalized parsers can be learned automatically and are very efficient, but their accuracy is currently lower than that of the manually tuned and relatively slow lexicalized parsers.

This conditioning comes in two flavors. The first is non-lexical, and works by annotating symbols with information about their context. This includes things such as parent symbols, grandparent symbols, siblings, and so on. The second is lexical and works by adding information about the words below the symbol. For good results with lexical annotation, humans are traditionally used to determine what information is critical; for example, head word rules have been devised to determine the most important word (“head word”) below a symbol. The

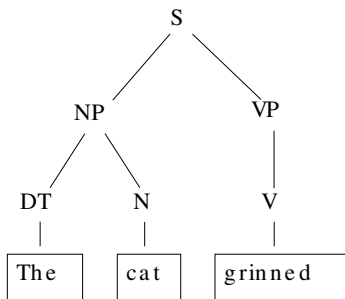


Figure 1: A CFG parse tree

non-lexical method is easy but less accurate. The lexical annotation provides better accuracy, but is more labor intensive, and it seems likely that humans are not exploiting the annotations as much as they could be. Therefore, it is advantageous to automatically find annotations.

Recent research [5] has shown that it is possible to automatically find annotations and get very good parsing results. This is done by assuming that there is a correct set of annotations and using an inference algorithm to approximate them. We reproduce some of these results and continue on to show that PCFGs with latent annotations give good transfer between different corpora. This transfer comes from not transferring all statistics between the corpora, but by fitting each corpus while finding and transferring the symbols and productions that have correspondences.

2 Latent Annotation

When using a PCFG, one often finds that there are correlations not captured by whatever context-free formulation is being used. For example, a noun phrase (NP) inside a prepositional phrase (PP) is likely to give a different subtree than a NP under a verb phrase (VP). This is because an NP under an VP is likely to be a (complicated) direct or indirect object, while NPs inside PPs tend to be small and simple. However, one can imagine a better PCFG that has more states, thus allowing it to describe a language with more correlations. Splitting states of a PCFG into substates carrying more information is known as *annotation*, because each state is annotated with extra information. When parsing using this new PCFG, first the sentence is parsed with the split states, and then the states are mapped back to the original states. Annotation is one of the most important advances making modern high-accuracy PCFG parsers possible.

In traditional PCFG annotation strategies, each nonterminal symbol in the training corpus is annotated in a deterministic way, depending on its context. For example, in *level n vertical markovization*, a non-lexical annotation technique, each nonterminal symbol is extended by its n parents, as in Fig. 2. Thus, more context is captured, dividing each state into new states that have different

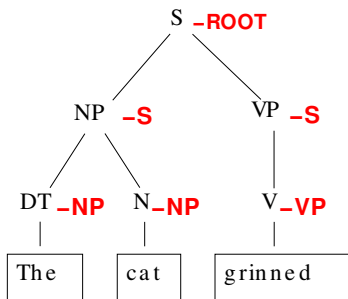


Figure 2: Non-lexical annotation in a CFG parse tree

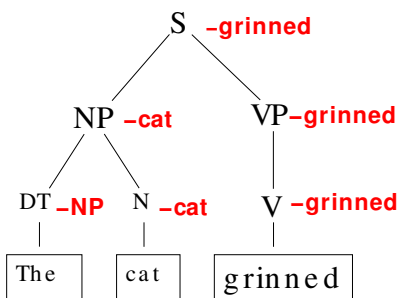


Figure 3: Lexical annotation in a CFG parse tree

distributions over productions. In lexical strategies, some of the added information is derived from the words of the sentence (leaves of the CFG parse tree), as in Fig. 3. Unfortunately, non-lexical techniques give lower scores, while lexical strategies require more human intervention.

The latent strategy, first described by [5], assumes that there is a correct annotation that we simply do not know. In particular, each nonterminal state is actually a grouping of n different symbols (see Fig. 4). When a nonterminal state is observed in a training parse tree, it is actually one of these n symbols, but we do not know which one.

3 Learning

Without annotations, a PCFG can be learned very easily from a “treebank,” a set of training sentences with their correct parse. One simply counts the number of occurrences of each rule under each symbol and renormalizes, obtaining a distribution over productions given the parent symbol. Since our PCFG now contains unobserved variables, we have no closed form for its maximum likelihood parameters. We therefore must use EM, which requires that we compute the expected sufficient statistics. In other words, for each training sentence we must compute the probability that a given rule is used over a given span. These

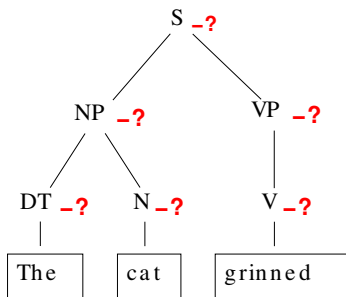


Figure 4: Latent annotation in a CFG parse tree

probabilities can be computed by the Inside-Outside algorithm (which generalizes to PCFGs the Forward-Backward algorithm for HMMs). This algorithm is $O(Rm^3n^3)$ in general, where R is the size of original the grammar, m the number of substates for each state, and n the length of the sentence. Worse, it must be run as the inner loop of EM. Fortunately in our case we can reduce this cost substantially. In particular, the span of each unannotated state is known and we only need to consider substates compatible with this. Using this fact leads to a simplified, $O(Rm^3n)$, Inside-Outside algorithm. This calculation and its aggregation over the training set is then repeated for each iteration.

Besides time, memory complexity quickly becomes an issue. By augmenting our grammar with substates we increase the size of our grammar (which grows as $O(n^3)$), as well as our training set trees. To address this we must optimize the grammar with respect to memory. To this end, we efficiently encode the grammar by grouping rules over substates of the same states, and we only allocate data arrays our training trees one at a time while running the Inside-Outside algorithm.

4 Parsing

The “right way” to parse with an annotated grammar is to find the most likely tree of *original*, un-annotated symbols. The probability of such a tree under the extended grammar is the sum of the probability of each compatible annotated tree. Let $P(T_X|w)$ be the probability of a tree rooted in X ; we want to find the tree with maximum probability.

$$\begin{aligned} \max_{T_X} P(T_X|w) &= \sum_i P(T_{X_i}|w) \\ &= \sum_{i,j,k} \max_{Y_j, Z_k} P(X_i \rightarrow Y_j Z_k) P(T_{Y_j}|w) P(T_{Z_k}|w) \end{aligned}$$

One can compute this probability efficiently for any single tree using dynamic programming. Unfortunately finding the most likely such tree among all trees is NP-Hard ([5]) because of the sum of the max.

Viterbi parsing. As an alternative, we turn to a Viterbi parse, in which we find the most likely tree of annotated symbols.

$$\max_{T_{X_i}} P(T_{X_i}|w) = \max_{Y_j, Z_k} P(X_i \rightarrow Y_j Z_k) P(T_{Y_j}|w) P(T_{Z_k}|w)$$

By eliminating the sum, this result becomes efficiently computable. Then, we remove the annotations to recover a parse tree with our original symbols. Although the resulting distribution over trees is not the same, we hope it is a good approximation. In particular, we hope that the maximum-likelihood annotated tree is similar to the highest-likelihood original symbol tree. The Viterbi parse is found by performing a pass of the Inside algorithm and recursively taking the annotated symbols with the highest inside probabilities. We based our parser on a form of the Stanford parser [6]. The parser uses an array-based chart representation for high parsing efficiency, and has a good unknown word model for high parsing accuracy.

Constrained parsing. Contrary to training, we have no observed variable to limit our search and must therefore pay the full $O(Rm^3n^3)$. To alleviate this computational cost we resort to an approximation. We first parse with the original (unsplit) grammar, and store for each span a list of the most likely symbols, throwing away the least likely ones using a relatively conservative threshold. We then extend the Viterbi algorithm for the full (split) grammar to make use of these lists by considering only these symbols (and their subsymbols) for these spans. (Note that the lists need be computed only once for the test set since they do not depend on the split grammar, only on the original grammar.) Though approximate, this method leads to virtually no reduction of accuracy yet reduces the cost of parsing the test set with four substates from approximately four days to only one hour.

Max-bracket parsing. If we get a score of 1 for finding exactly the right tree, and a score of 0 for any other answer, then returning the most likely parse tree makes sense since it maximizes our score on average. One can argue whether this is a sensible objective. From a purely practical point of view, state of the art parsers are compared in terms of the number of correct constituents (a state and a span). The tree that maximizes this score on average and the algorithm to find it are very different from the Viterbi parse tree and algorithm. Parsing to maximize constituent recall is described in [7]. Importantly for us this algorithm is based on the symbol marginal probabilities only. The marginal probabilities of the original states are simply the sum of that of their substates, so the algorithm goes through almost unchanged. We are still currently implementing this approach to compare it to the Viterbi method.

	multiplicative	$k = 0.04$	$k = 0.2$	$k = 1$	$k = 10$
F1	84.79	85.26	84.66	84.54	84.68

Table 1: Accuracy of random initialization schemes with four latent annotations, averaged over four trials. A small, additive randomization is best.

5 Initialization

The EM algorithm repeatedly modifies the PCFG’s parameters to always increase the model’s likelihood on the training data. In this, it is a gradient ascent algorithm, and it must be initialized with a guess as to a model. However, the obvious starting point (with all latent annotations given identical probabilities) is at a point of symmetry in likelihood space, and so the gradient is zero. Therefore, we must first break the symmetry of the model before we can learn a latent annotation.

We break the symmetry by adding a certain amount of randomness to the model. Unfortunately, there are many ways to add randomness, and they are not all equally useful. If too much randomness is added, then we may be likely to start near a local maximum, which would mean that we would find a suboptimal language model. Therefore, our intuition was to add a small amount of randomness only. However, [5] adds a surprisingly large amount of randomness and still gets good results, so we studied how the amount of randomness affects parser accuracy.

We examined two different randomization schemes. The first, belonging to [5], multiplies rule probabilities by a factor, mapping them like

$$p' = cpe^{6r-3}$$

where r is uniformly distributed in $[0, 1]$, p is the unannotated probability, and c is a normalization factor. The second, which we devised, transforms rule probabilities to

$$p' = c(p + kr)$$

where p is the symmetric probability, r is as given above, k is a tunable parameter, and c is a normalization factor. In Table 1, we show the F1 score, averaged over four runs, of various initialization schemes when used on a model with four latent annotations per symbol.

6 Number of Annotations

We improve the grammar by finding an optimal set of latent annotations. However, there is an inherent tradeoff as the number of annotations increases. On the one hand, adding more annotations gives the grammar more freedom to capture correlations within the corpus. On the other hand, our training set is limited, and with many annotations we must estimate each split symbol with

	1	2	4	8
mean F1	79.72	84.66	85.72	82.85

Table 2: Accuracy versus number of latent annotations, averaged over four trials. Note the drop in accuracy with eight latent annotations.

	1	2	4	8
mean F1	79.72	84.78	85.75	85.58

Table 3: Accuracy versus number of latent annotations, averaged over four trials, with an iterative initialization scheme. Grammars with eight latent annotations do not decline so much.

less and less data. Therefore, there should be an optimal number of annotations. However, because the previous work [5] shows improvement even up to 16 latent annotations, we expect that we will run out of computing resources before data sparsity becomes an issue.

When we trained the grammar with different numbers of latent annotations, we found a surprising result. The accuracy actually dropped when we went up to eight latent annotations (See Table 2.) We hypothesized that this drop is the result of poor initialization leading to EM being trapped in a local maximum. To avoid this, we devised a scheme in which we use a grammar with n latent annotations to initialize a grammar with $2n$ latent annotations. When we do this, we find that the accuracy does not drop so dramatically when states are split into eight (Table 3). Unfortunately, the accuracy still declines. However, this test was run with a random initialization (§ 5) with $k = 0.2$. It may be that with less randomness, we would find that the accuracy does not decline at all.

We also experimented with variable numbers of latent categories for the non-terminals. The motivation being that there can be many different “types” of noun phrases (NP), while determiners (DT) will typically serve the same function. In order to decide how often to split each symbol, we employed two different heuristics:

- Context count: Non-terminals which appear in a high number of different contexts are split more often. The nodes immediately above and below a particular node are regarded as the context.
- Symbol count: More frequent non-terminals are split more often.

While a variable number of categories leads to some moderate improvement, it became clear that a more sophisticated technique for determining the number of latent categories is necessary.

7 Cross-Corpus Transfer

The goal of our Transfer Learning project was to get a language model to transfer information between corpora. To test this, we used both the Brown and the

	Brown	Brown + WSJ	WSJ
1	80.46	80.20	77.18
2	82.89	83.61	79.74
4	82.24	84.46	79.64

Table 4: Accuracy (F1) on the **Brown** corpus depending on training corpora and number of substates, averaged over 4 random initializations.

	Brown	Brown + WSJ	WSJ
1	68.89	79.49	79.72
2	71.47	84.87	84.68
4	70.2	86.17	85.58

Table 5: Accuracy (F1) on the **WSJ** corpus depending on training corpora and number of substates, averaged over 4 random initializations.

WSJ parts of the Penn Treebank. We trained on one or both of the corpora, and then we tested on one or both of the corpora.

For transfer to be useful, we should observe that adding a second training corpus increases performance on the first corpus. However, without latent annotations, we find that this is not the case. In fact, we found that adding a second corpus hurt accuracy on both the Brown and WSJ corpora (see row 1 of Tables 4 and 5). However, if we find latent annotations, then the accuracy improves on both corpora when we add a second corpus. So, clearly, running EM helps to find a latent annotation that generalizes between the two corpora.

In fact, by training on both the WSJ and Brown corpora, we achieved an F1 score of 87.08 on the WSJ corpus. This is better than the 86.6 score from [5], and it was using 1/4 as many substates as they used, so we hope to obtain even better results in the coming months.

Note that the accuracy drops when corpora are mixed using a grammar without latent annotation. This is a surprise because there is plenty of mixing going on in these cases; the estimates of rule probabilities are based on pooled data. The question then is, what information does latent annotation prevent from being transferred? We studied the data and came to the conclusion that some annotated symbols are *not* being shared with the Brown corpus. We compared how often annotated symbols appeared in the Brown parses versus the WSJ parses, when the grammar was trained either with the Brown corpus or the mixture. Approximately 5% of annotated symbols appear much less frequently in the Brown corpus when trained with the mixture (see Fig. 5). That means that the latent annotation learns a grammar in which some symbols do not appear often in the Brown corpus – essentially, these symbols represent structure which may be found in the WSJ corpus but not in the Brown corpus.

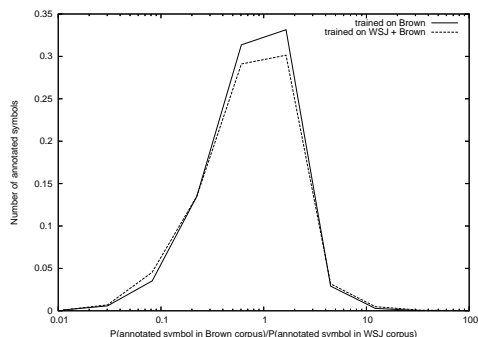


Figure 5: This graph shows what percentage of annotated symbols had a given likelihood ratio between the Brown and WSJ parses. Because we see that the likelihood ratio varies more when we train on both corpora, we conclude that many annotated symbols appear less in the Brown corpus when we train with the WSJ corpus also.

8 Conclusion

We have reproduced the results of [5] and shown that PCFGs with latent annotations find annotations giving high parsing accuracy. Also, they allow multiple corpora to share information in such a way that performance is improved on all of the corpora, whereas ordinary PCFG models fail at this transfer. We have already seen impressive performance, and we expect to see continuing gains in the future. However, our work is not yet complete; while we have laboriously optimized the parser to allow training and testing with high numbers of latent annotations, we have not yet had time to run a full battery of tests. This work is an ongoing project with Dan Klein.

References

- [1] E. Charniak. Tree-bank grammars. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 1031–1036, 1996.
- [2] E. Charniak, S. Goldwater, and M. Johnson. Edge-based best-first chart parsing. *Sixth Workshop on Very Large Corpora*, pages 127–133, 1998.
- [3] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, Univ. of Pennsylvania, 1999.
- [4] D. Klein and C. Manning. Accurate unlexicalized parsing. *Proc. of the 41st Meeting of the Association for Computational Linguistics (ACL)*, pages 423–430, 2003.
- [5] T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic CFG with latent annotations. In *Proc. of the 43rd Ann. Mtg. of the Assoc. for Comp. Ling. (ACL’05)*, pages 75–82. Assoc. for Comp. Ling.
- [6] The Stanford Natural Language Processing Group. <http://www-nlp.stanford.edu/software/lex-parser.shtml>.

- [7] J. Goodman. Parsing algorithms and metrics. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 177–183, San Francisco, 1996. Morgan Kaufmann Publishers.