Combining Grammars For Improved Learning

Glenn Carroll, and Eugene Charniak

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-94-08
Febuary 1994

Combining Grammars for Improved Learning

Glenn Carroll
Eugene Charniak *
Department of Computer Science
Brown University
Providence RI 02912

February 17, 1994

Abstract

We report experimental work on improving learning methods for probabilistic context-free grammars (PCFGs). From stacked regression we borrow the basic idea of combining grammars. Smoothing, a domain-independent method for combining grammars, does not offer noticeable performance gains. However, PCFGs allow much tighter, domain-dependent coupling, and we show that this may be exploited for significant performance gains. Finally, we compare two strategies for acquiring the varying grammars needed for any combining method. We suggest that an unorthodox strategy, "leave-one-in" learning, is more effective than the more familiar "leave-one-out".

1 Introduction

We are interested in ways of improving the performance of probabilistic grammars learned from tagged text. We intend to use these grammars for language modelling, which has applications in speech recognition, text retrieval, and the like. Language modelling makes two demands on a grammar, first that it parse all or nearly all sentences, and second that it assign each sentence a probability, preferably as high as possible. We describe these goals more formally below. In the last few years, large

*This research was supported in part by NSF contract IRI-8911122 and ONR contract N0014-91-J-1202.

corpora of text have become available online for the first time, and these, together with the abundance of relatively cheap computation make learning probabilistic grammars practical and even attractive.

This paper reports the results of experiments in adapting an idea from stacked regression and extending it in a domain-dependent fashion to improve the performance of an existing grammar learner. Wolpert [1] lays out a very general theoretical framework for stacked regression, based on supervised learning from noise-free data. The framework is sufficiently general that it can be reified in a number of quite different learning algorithms. Here we focus on the idea of combining information from more than one generalizer, which in our case are probabilistic context-free grammars (PCFGs). Wolpert points out that if one has gone to the trouble to build several generalizers from a single data set, then it is almost always better to combine the information from them, rather than, say, choosing one, as has been standard practice with cross-validation. In the stacked regression framework this combination process may take almost any form, some of which require an error signal (i.e., supervised learning), and are therefore infeasible in our domain. One of the more obvious and feasible ideas for combining generalizers is simply to average across them, and other researchers report good results from this [2]. We can do slightly better than that, by choosing optimal weights to smooth over our grammars.

To be more concrete, we are suggesting generating several (probabilistic) grammars G_i and smoothing them together. As we describe in more detail below, each grammar defines a probability distribution over sentences P_{G_i}, and so the smoothed combination will likewise define a distribution. To smooth across multiple grammars, we choose some positive weights λ_i, such that $\sum_i \lambda_i = 1.0$ and we define the smoothed probability of a sentence s to be

$$P(s) = \sum_i \lambda_i P_{G_i}(s) \tag{1}$$

We can choose the $lambda_i$ using the forward-backward algorithm [3] and some training data reserved prior to learning the grammars.

It will be noticed that this suggestion contains nothing that is domain-specific or, for that matter, original. Provided multiple generalizers (our grammars) are available, it could be applied anywhere. Note, however, that the mathematical arguments from stacked regression supporting this method do not apply to our domain. These arguments rest on assumptions concerning the construction of the generalizers, and those assumptions, in turn, are based on a supervised model of learning. (Supervised learning is impractical for PCFG learning, as an actual human would have to provide the supervision.) Perhaps because these assumptions do not hold, the method does not improve performance, as our first experiment shows.

Our real interest was not in the stock approach described above, but rather in a domain-dependent method of combining PCFGs, essentially by "adding" them to-

gether to produce a single grammar. The resulting tighter coupling, together with an appropriate training algorithm, results in a substantial performance increase, as we document in our second experiment.

Finally, in our last experiment we vary the way in which the grammars are generated, in order to find the best method to use with our "adding" procedure. Rather than varying an algorithmic parameter, we vary the data available to the learner. This bears some similarity to "leave-one-out" cross-validation, which builds multiple generalizers this way for the purpose of evaluating learner performance. Unlike cross-validation, in which the aim is to select the best generalizer, our aim is to *use* all of the acquired generalizers, so we find a rather different allocation of the data is more suitable.

The rest of this paper is laid out as follows. In the next section, we describe PCFGs, performance measures for PCFGs, and our basic PCFG induction scheme. Following that we give results for three experiments, evaluating the generic smoothing scheme, our domain dependent scheme, and "leave-one-in" versus "leave-one-out" learning. Finally, we summarize the results and implications of our experiments in the conclusion.

2 Probabilistic Context-Free Grammars

2.1 PCFG Definitions and Properties

A probabilistic context-free grammar is just an ordinary grammar with a weight, or probability, associated with each rule. For each non-terminal, NT, the sum of weights for all rules headed with NT is 1.0.[1] A probabilistic grammar implicitly defines a distribution over strings in its language; any string not in the language has probability zero. Formally we write this as

$$\forall s(s \notin L(G) \rightarrow \Pr_G(s) = 0).$$

A parse assigned by a PCFG is identical to the parse assigned by an ordinary, non-probabilistic, context-free grammar. It is a tree rooted at the grammar's start symbol, with each interior node representing the expansion of some non-terminal by a grammar rule, and with the parsed string distributed one symbol per leaf over the tree's leaves.

The probability of a parse p according to grammar G is

$$\Pr_G(p) = \prod_{r \in p} \Pr_G(r),$$

[1] N.B. that one does *not* sum over all rules in the grammar.

where $\Pr_G(r)$ is the probability (weight) G assigns to rule r.

The probability of a string according to grammar G is

$$\Pr_G(s) = \sum_{\forall p \in parses(G,s)} \Pr_G(p)$$

Since our grammars are for English text, our strings will always be sentences.

The probability of a corpus of sentences, C, according to grammar G is

$$\Pr_G(C) = \prod_{s \in C} \Pr_G(s) \tag{2}$$

This definition makes the standard assumption that sentences occur independently. For convenience, most calculations are actually made in terms of the *cross entropy* of a corpus, which is the negative log of its probability.

Our interest in PCFGs lies in their usefulness as language models. For language modeling we wish to maximize our predictive ability, which means maximizing the probability of a corpus, or, equivalently, minimizing its cross entropy. There exists a fairly well-known algorithm [3–5] for tuning a PCFG's probabilities to maximize the probability of a training corpus. The important properties of this algorithm, known as the Inside-Outside (I-O) algorithm, are that it allows unsupervised training, it can get stuck in local maxima, it is deterministic, and it approaches the maximum probability settings asymptotically with repeated iterations through the training data. The I-O algorithm does not induce new rules, although it can effectively delete rules by assigning them a probability of zero.

2.2 Performance

If our learned grammars had 100% coverage, i.e., they could be expected to parse all sentences in a test corpus, then cross entropy would be the right performance measure. Unfortunately, it is difficult to realize both perfect coverage and low cross entropy, and both are desirable properties for language models. Essentially, one must choose between inducing a very general grammar, which spreads probability thinly over more sentences, or a grammar that parses fewer sentences, but assigns a higher probability to the sentences that it does parse. Note that a single unparsable sentence makes the cross entropy of the corpus infinite (because the probability is zero).

One can imagine many tradeoff functions for balancing coverage versus cross entropy but it is hard to argue convincingly for any particular one. Rather than selecting one at random, we measure our own performance in comparison with the performance of the current industrial standard, a smoothed trigram language model[2]. Such mod-

[2] A trigram model predicts the next word from the previous two, and assigns a probability to a corpus as the product of the probability of all words. A smoothed trigram model combines a trigram model with a bigram model (next word predicted from only the previous word) and a unigram model (next word predicted by its frequency in text).

els are guaranteed to have 100% coverage and they have performed extremely well as language models for many years. They are serious competition, rather than a straw person.

We measure comparative performance by treating both models as components of an overall model, which is the smoothed combination of the the two. We assign both models an initial weight of 0.5, and tune these values using the forward-backward algorithm on held-out test data. We measure performance as the final weight assigned to the grammar model, and call it λ_G. The final weight assigned to the trigram model is $\lambda_T = 1.0 - \lambda_G$). A λ_G of 0.5 would indicate that the grammar and the trigram performed identically well, overall.

One way to think of this is that we are assigning each sentence an equal, fixed weight, and we are dividing that weight between the two models according to the probability each model assigns the sentence. (Actually we use the probability times λ_G and λ_T for the grammar and the trigram model, respectively.) Whichever model gets the most weight is the better model. The forward-backward algorithm successively re-estimates the weights using the lambdas, and vice-versa.

The key advantage of this measure is that it assigns a finite value to each sentence, whereas cross entropy may assign an infinite value, as it does when a sentence is unparsable. This makes cross entropy a "brittle" measure, as a single sentence of zero or even near-zero probability probability can determine a grammar's performance for an entire corpus. Our grammar's coverage of test data is generally around $99.2 \pm 0.5\%$, so some sentences inevitably fail.

Aside from this brittleness, we regard cross entropy as the right measure, and our λs track it faithfully. Other things being equal, any change in cross entropy will be mirrored by a change in λ. If either model has superior cross entropy on every sentence, its λ will reach 1.0, indicating complete dominance. In addition, our measure has the rather handy feature that, if the two models usually assign widely disparate probabilities, each λ may be read as the percentage of sentences for which the corresponding model is superior. Our two models meet this condition, most of the time, so it is a good approximate indicator for us.

2.3 PCFG induction

Stacked regression requires some existing means of induction to furnish the generalizers it needs. We discuss our induction scheme in detail elsewhere [6]. In skeleton form, there are two parts to PCFG induction: acquiring the rules, and setting their probabilities. In essence, we divide the process into corresponding phases, and handle them separately.

The rule acquisition phase is error-driven. It accepts an initial grammar, which may be empty, called the bias, and it extends the bias to cover sentences that it

cannot initially parse. As we report elsewhere, we obtain the best performance when we can extend the grammar incrementally, adding fewer and shorter rules first. We get this effect by sorting the input sentences, shortest first.

Our procedure does not guarantee that a sentence can be parsed even after rule acquisition. (Generally speaking, however, coverage of training data is $> 99\%$ after this learning, so there are few sentences for which it fails.) If the learner fails, that is, if the sentence remains unparsable, then no suggested rules are added to the grammar.

There are two ways in which learning can fail. First, it could fail because our constraints on new rules simply do not allow some necessary rule or rules to be suggested. This seldom happens, as our constraints have migrated away from being rigid, "yes-or-no" rules towards probabilistic preferences.

Another way in which learning can fail is that so many new rules are generated that the parser is effectively swamped. It is not hard for a single sentence to generate thousands of new rules, doubling the size of the grammar, although clearly only very limited information can be gleaned from any one sentence.) In this case, the learner aborts, discards the new rules, and indicates a failure. This bias the learner against learning from long sentences, which seems reasonable, as long sentences are harder to learn from.

The second phase, setting the probabilities, is a straightforward deployment of the Inside-Outside algorithm.

3 Results

Results reported in this section all use the basic induction algorithm described above, seeded with a hand-built bias grammar containing 600 rules. The training data used was a 300,000 word subset of the Brown Corpus of Tagged English [7], a collection of articles, excerpts from books, and the like. Each word is tagged with its part of speech, e.g., noun, verb, etc. Our grammars operate on the tags alone, ignoring the words entirely. Our subset of the corpus was created by filtering out all sentences that had more than 23 words or had undesirable words such as parentheses, headlines, or foreign words. A 10,000 word subset was reserved for testing.

For comparison purposes, we provide the average performance of the basic induction algorithm. We use the average because our grammar learning is dependent on sentence ordering, or, in other words, is not deterministic without a completely fixed sentence ordering. Consider two sentences of a given length, say A and B. Suppose that for A our learner will suggest rules that are also sufficient to parse B, but for B the learner would suggest a different set. What rules the grammar ends up with depends on which sentence occurs first. In figure 1 we give average performance (the λ) of the basic induction scheme, and the individual performance on 10 learning runs over randomized orderings of the corpus. (The shortest-first ordering was still, of

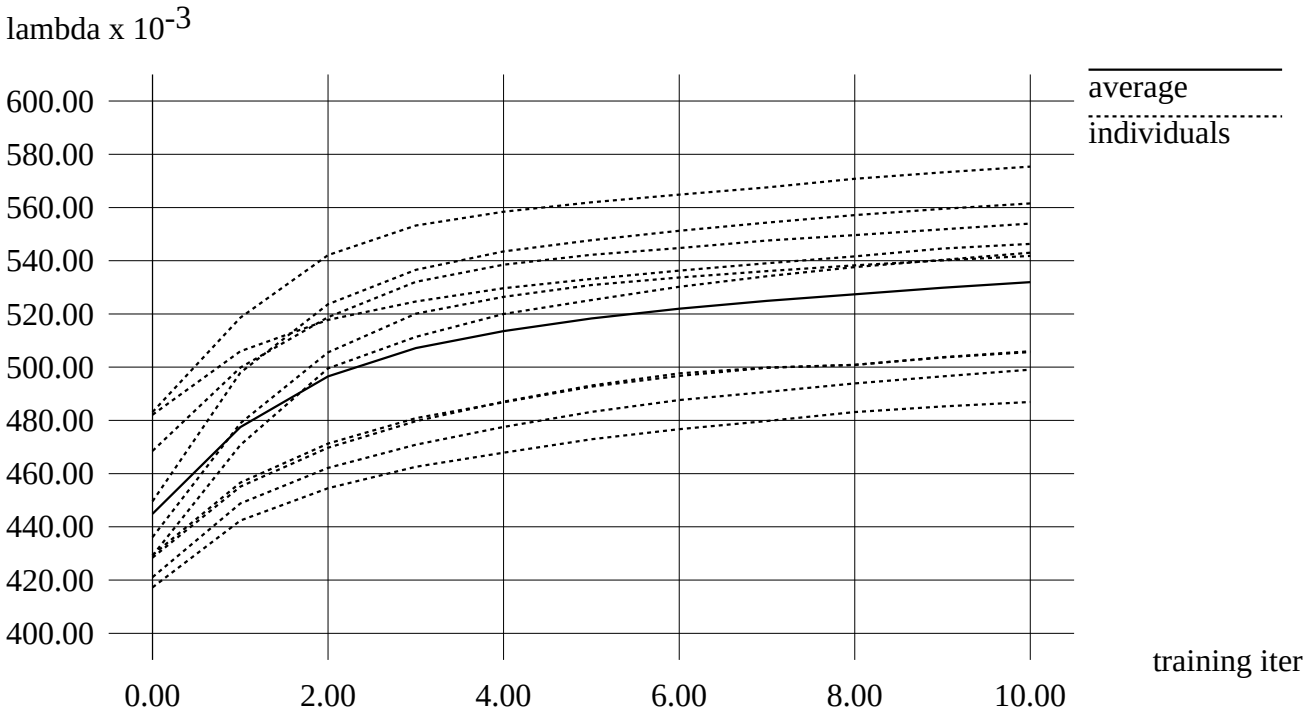

Figure 1: Basic Learning, showing individual runs from randomized orderings and overall average.

	lambda	lambda variance	cross entropy	coverage %
basic	0.53235	0.000869081	2.74561	98.96
smoothed	0.52857	0.000084159	2.7476	99.35
added, 10 segs, l-o-o	0.61753	0.000079539	2.72856	99.36
added, 10 segs, l-o-i	0.657562	0.000722075	2.7175	99.56
added, 100 segs, l-o-i	0.72071	0.00005106	2.7017	99.66
added, 25k segs, l-o-i	0.71114	0.0	2.70285	99.72

Figure 2: Comparison of smoothing versus unsmoothed grammars

course, preserved.) In all cases, we evaluate performance immediately after learning and after 10 iterations of training by the Inside-Outside algorithm, which in our experience is enough to reveal any systematic differences among grammars, although performance may continue to climb with further iterations.

Figure 1 should give one a feel for the behavior of our grammars through the training process. In general, performance immediately following learning is poor, and increases most rapidly for the first couple of training iterations. Performance curves occasionally cross, but there are no wild differences in the shape of the curves, nor are there dips indicating overtraining. Overall, there is more variation in performance than we would like, and it seems largely connected with the coverage of the learned grammar. Coverage ranges from 98.68 to 99.23% of total words parsed for these grammars, and is almost perfectly correlated to increased performance.

3.1 Experiment 1

Figure 2 summarizes the results for all our experiments. All results are averages from 10 runs over randomized orderings, except as otherwise noted below. The entry headed "smoothed" gives the graph of results for our "replication" of ordinary stacked regression, or at least our unsupervised learning approximation to it. For this experiment we divided the corpus into 10 equal sized pieces. Nine of these pieces were used to generate grammars. The different grammars were created by varying the data available to each learning run. In each case, one of the 9 remaining pieces of training data was left out, so the learner saw only 80% of the data. Each grammar was then trained once, using the same 90% as was used for learning. The grammars were trained only once in order to allocate training time comparable to 10 passes with a single grammar. Finally, smoothing parameters were chosen using the forward-backward algorithm on the reserved 10% of the data.

The first two lines of the table in figure 2 show that the smoothed grammars do not perform better than the "basic" scheme of learning over the whole corpus all at

once. Actually, they perform somewhat worse, but not significantly so.

3.2 Experiment 2

As we said earlier, our real hope for performance improvement lay in the idea of "adding" grammars together, so that rules distributed among the different grammars can cooperate in parsing a sentence. The technique relies on information gleaned by the Inside-Outside algorithm, which provides a count for each rule, indicating how many times it is used. I-O works by alternately using the rule probabilities and the training data to estimate the counts, and then re-estimating the probabilities from the counts. For our purposes, the key property of the counts is that they can act as a common currency among grammars. Rule probabilities, which are the most obvious alternative, lack this property of interchangeability, making them unsuitable for our purpose.

We add grammars together, then, by adding the rule counts.

Definition 1 *The sum of n grammars, designated SG, is the grammar created by summing the rule counts over the n grammars as follows.*

$$count(SG, r) = \sum_i count(G_i, r)$$

where count maps the rule r to its count in grammar G.

If a rule does not appear in some grammar, it has a count of zero. Probabilities for the new grammar are calculated from the counts, using the same equation that the Inside-Outside algorithm relies on. For a given non-terminal, NT, and some string of terminals and non-terminals, α^i, the probability of the rule $NT \rightarrow \alpha^i$ can be calculated from the counts as follows.

$$Pr_G(NT \rightarrow \alpha^i) = count(NT \rightarrow \alpha^i) / \sum_i count(NT \rightarrow \alpha^i) \quad (3)$$

In this experiment we added together the 9 untrained grammars generated in experiment 1, and then trained them 10 times. Results are shown in figure 2 under the entry "added, 10 segs, l-o-o", for adding the grammars, splitting the data into 10 segments, and "leave-one-out" rule induction. This penalizes the summing technique, as it has no chance to learn on 10% of the data, but as the table shows, performance is still superior.

3.3 Experiment 3

In this experiment we alter the way we allocate the training data to learn the varying grammars that we subsequently add together. While the "leave-one-out" strategy of

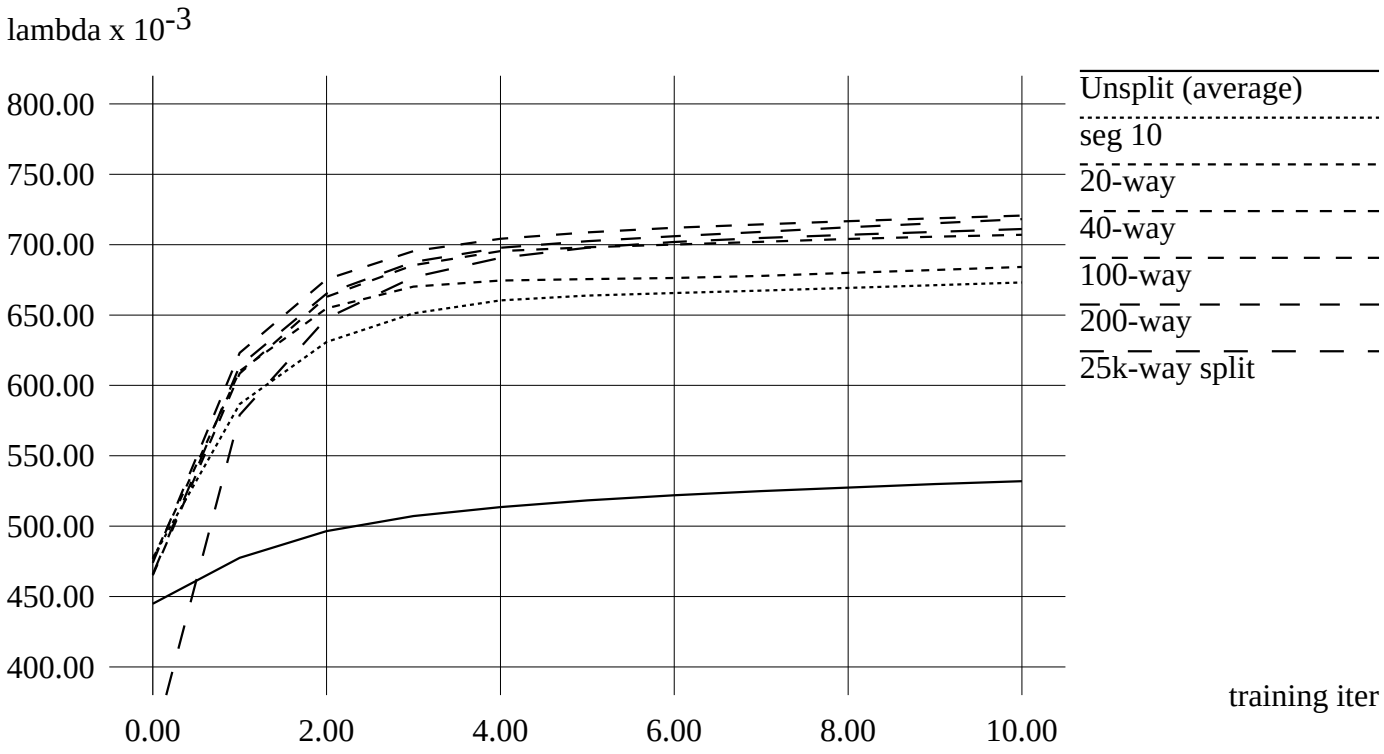

Figure 3: Performance of leave-one-in learning, for various segmentation values.

learning has a long and honorable history in machine learning, and it is a very natural starting point, it does not seem the most promising for our problem. It is computationally expensive and it does not scale well. By this we mean that increasing the segmentation, i.e., the number of ways one divides the data, leave-one-out produces grammars that look more and more alike, and, as variation appears to be essential for our needs, this is counter-productive. Instead we take the opposite approach to leave-one-out, and keep only a single segment for learning, rather than all but one. As before, we iteratively train the grammars only after they are added, as it is more efficient to do so after, when a single pass through the data trains all new rules. Under this scheme, which we call "leave-one-in", increasing the segmentation will tend to increase the variation among grammars. Moreover, the grammars are cheaper to learn, as fewer sentences are processed for each. Lastly, under leave-one-in, more sentences contribute rules at a given segmentation number; in the limit, when one increases the segmentation to the corpus size, all sentences get a chance to contribute rules. In contrast, under leave-one-out some sentences may never trigger learning, and so will never suggest rules, under *any* segmentation. Figure 3 shows results of leave-one-in

learning at various segmentation settings, up to the limit case of segmenting the corpus into one-sentence chunks. As the figure shows, performance does not increase up to the limit case, but appears to level off around a segmentation of 100. Our present belief, as yet untested, is that this results from the loss of ordering information at very high segmentations.

The loss of ordering information comes about because each segment has very few sentences. This, in turn, means that the grammar cannot develop incrementally. On the other hand, it also reduces the variance among grammars generated from different orderings, as the variance column in figure 2 shows. In the limit case, there is no variation at all, as there is zero ordering information. We intend to further explore both the levelling off in performance and the benefits from ordering in the future.

4 Conclusion

We have presented positive results from experiments in borrowing an idea from stacked regression and adapting it to our own domain. Our experiments do not show promising results from generic smoothing over multiple grammars, but merging them provides the synergy we were hoping for. Grammar merging relies on variety among the constituent grammars, which implies that "leave-one-out" learning will not provide much mileage. For our application, "leave-one-in" is both computationally cheaper and more thorough in extracting information, as the learning is error-driven.

One issue we have not discussed so far is the relative CPU cost of the various schemes. The current version of the basic learning scheme takes about 7 hours for rule acquisition and 1.5 hours for each training iteration. Since rule acquisition involves the same parsing as training, this means that about 5.5 hours are spent purely learning rules. The bad news is that for small segmentation levels, up to around 15, say, the increase in CPU cost is linear, i.e., it takes about 10 times longer to learn the rules when the segmentation is 10, whether the learning is leave-one-out or leave-one-in. (Training the rule probabilities remains about the same, as segmentation has no effect on training. The only increase in time comes from the increase in grammar size that comes with increased segmentation.) The reason is that rule learning is error-based, and for small segmentation levels, the error rate per segment does not significantly differ from the error rate for the basic, unsegmented learner. At high segmentation levels, say 100 and up, leave-one-out and leave-one-in behave differently. Leave-one-out continues to climb linearly in cost, as each learning pass comes closer to processing all of the data. For leave-one-in learning, the error rate per segment begins to drop, so performance cost likewise drops to less than linear. Note, also, that leave-one-out learning parses the entire corpus approximately $N-1$ times at a segmentation of N, whereas leave-one-in parses it only once.

In our own environment, we have all the data available in batch form, and enough computers to handle up to about 100 segments in parallel. This allows for very fast experimentation. In an online environment, one would not expect to have either these resources, or all the data available at once. Our results suggest a strategy such as the following would be effective even in this more constrained environment. Incoming data should be processed in chunks of a few hundred sentences. (For our data set, a segmentation of 100 means each learner sees about 250 sentences.) Sort the sentences, and use the basic learner and the bias grammar to acquire new rules. The learned grammar, which will use the new rules, is kept separate at all times. The new rules are added after learning, and then the I-O algorithm is run over the learned grammar and the new data. I-O will compute new rule counts, which periodically should be used to re-estimate the rule probabilities.

This strategy allows some incremental development from the chunking; it puts all rules in a single grammar to reap the benefits of cooperation; and it maintains a relatively high error-rate to drive continued learning. While overall the learning must, inevitably, be slower under this regime, only a few thousand sentences should suffice to reach reasonable performance, and the scheme has the advantage of accomodating arbitrarily large data sets.

5 Acknowledgements

We are grateful to Leslie Kaelbling for reading a draft of this paper and providing us with constructive comments.

References

[1] David H. Wolpert, "Stacked Generalization," 90.

[2] G.E. Schulz, "Comparison of predicted and experimentally determined structure of adenyl kinase," *Nature* 250 (1974), 140–142.

[3] L. E. Baum, "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes," *Inequalities* III (1972), 1–8.

[4] J. Baker, "Trainable Grammars for Speech Recognition," in *Speech Communication Papers for the 97th Meeting of the Acoustic Society of America*, D. Klatt and J. Wolf, ed., ASA, 1982, 547–550.

[5] Eugene Charniak, *Statistical Language Learning*, MIT, Cambridge, MA, 1993.

[6] Glenn Carroll & Eugene Charniak, "Learning Probabilistic Dependency Grammars from Labelled Text," *Proceedings of the AAAI Fall Symposium* (1992).

[7] W. Nelson Francis & Henry Kučera, *Frequency Analysis of English Usage: Lexicon and Grammar*, Houghton Mifflin, Boston, 1982.