# Improving Neural Parsing by Disentangling Model Combination and Reranking Effects

**Daniel Fried*** **Mitchell Stern*** **Dan Klein**
Computer Science Division
University of California, Berkeley
{dfried,mitchell,klein}@cs.berkeley.edu

## Abstract

Recent work has proposed several generative neural models for constituency parsing that achieve state-of-the-art results. Since direct search in these generative models is difficult, they have primarily been used to rescore candidate outputs from base parsers in which decoding is more straightforward. We first present an algorithm for direct search in these generative models. We then demonstrate that the rescoring results are at least partly due to implicit model combination rather than reranking effects. Finally, we show that explicit model combination can improve performance even further, resulting in new state-of-the-art numbers on the PTB of 94.25 F1 when training only on gold data and 94.66 F1 when using external data.

## 1 Introduction

Recent work on neural constituency parsing (Dyer et al., 2016; Choe and Charniak, 2016) has found multiple cases where generative *scoring models* for which inference is complex outperform *base models* for which inference is simpler. Let A be a parser that we want to parse with (here one of the generative models), and let B be a base parser that we use to propose candidate parses which are then scored by the less-tractable parser A. We denote this cross-scoring setup by B $\rightarrow$ A. The papers above repeatedly saw that the cross-scoring setup B $\rightarrow$ A under which their generative models were applied outperformed the standard single-parser setup B $\rightarrow$ B. We term this a *cross-scoring gain*.

This paper asks two questions. First, *why* do recent discriminative-to-generative cross-scoring setups B $\rightarrow$ A outperform their base parsers B? Perhaps generative models A are simply superior to the base models B and direct generative parsing (A $\rightarrow$ A) would be better still if it were feasible. If so, we would characterize the cross-scoring gain from B $\rightarrow$ B to B $\rightarrow$ A as a *reranking gain*. However, it's also possible that the hybrid system B $\rightarrow$ A shows gains merely from subtle model combination effects. If so, scoring candidates using some combined score A + B would be even better, which we would characterize as a *model combination gain*. It might even be the case that B is a better parser overall (i.e. B $\rightarrow$ B outperforms A $\rightarrow$ A).

Of course, many real hybrids will exhibit both reranking and model combination gains. In this paper, we present experiments to isolate the degree to which each gain occurs for each of two state-of-the-art generative neural parsing models: the Recurrent Neural Network Grammar generative parser (RG) of Dyer et al. (2016), and the LSTM language modeling generative parser (LM) of Choe and Charniak (2016).

In particular, we present and use a beam-based search procedure with an augmented state space that can search directly in the generative models, allowing us to explore A $\rightarrow$ A for these generative parsers A independent of any base parsers. Our findings suggest the presence of model combination effects in both generative parsers: when parses found by searching directly in the generative parser are added to a list of candidates from a strong base parser (the RNNG discriminative parser, RD (Dyer et al., 2016)), performance decreases when compared to using just candidates from the base parser, i.e., B $\cup$ A $\rightarrow$ A has lower evaluation performance than B $\rightarrow$ A (Section 3.1).

This result suggests that both generative models benefit from fortuitous search errors in the rescoring setting – there are trees with higher probability

*Equal contribution.



*Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers)*, pages 161–166
Vancouver, Canada, July 30 - August 4, 2017. 
https://doi.org/10.18653/v1/P17-2025

under the generative model than any tree proposed by the base parser, but which would decrease evaluation performance if selected. Because of this, we hypothesize that model combination effects between the base and generative models are partially responsible for the high performance of the generative reranking systems, rather than the generative model being generally superior.

Here we consider our second question: if cross-scoring gains are at least partly due to implicit model combination, can we gain even more by combining the models explicitly? We find that this is indeed the case: simply taking a weighted average of the scores of both models when selecting a parse from the base parser's candidate list improves over using only the score of the generative model, in many cases substantially (Section 3.2). Using this technique, in combination with ensembling, we obtain new state-of-the-art results on the Penn Treebank: 94.25 F1 when training only on gold parse trees and 94.66 F1 when using external silver data.

## 2 Decoding in generative neural models

All of the parsers we investigate in this work (the discriminative parser RD, and the two generative parsers RG and LM, see Section 1) produce parse trees in a depth-first, left-to-right traversal, using the same basic *actions*: NT($X$), which opens a new constituent with the non-terminal symbol $X$; SHIFT / GEN($w$), which adds a word; and REDUCE, which closes the current constituent. We refer to Dyer et al. (2016) for a complete description of these actions, and the constraints on them necessary to ensure valid parse trees.[1]

The primary difference between the actions in the discriminative and generative models is that, whereas the discriminative model uses a SHIFT action which is fixed to produce the next word in the sentence, the generative models use GEN($w$) to define a distribution over all possible words $w$ in the lexicon. This stems from the generative model's definition of a joint probability $p(x, y)$ over all possible sentences $x$ and parses $y$. To use a generative model as a parser, we are interested in finding the maximum probability parse for a given sentence. This is made more complicated by not having an explicit representation for $p(y|x)$, as we do in the discriminative setting. However, we can start by applying similar approximate search procedures as are used for the discriminative parser, constraining the set of actions such that it is only possible to produce the observed sentence: i.e. only allow a GEN($w$) action when $w$ is the next terminal in the sentence, and prohibit GEN actions if all terminals have been produced.

### 2.1 Action-synchronous beam search

Past work on discriminative neural constituency parsers has shown the effectiveness of beam search with a small beam (Vinyals et al., 2015) or even greedy search, as in the case of RD (Dyer et al., 2016). The standard beam search procedure, which we refer to as *action-synchronous*, maintains a beam of $K$ partially-completed parses that all have the same number of actions taken. At each stage, a pool of successors is constructed by extending each candidate in the beam with each of its possible next actions. The $K$ highest-probability successors are chosen as the next beam.

Unfortunately, we find that action-synchronous beam search breaks down for both generative models we explore in this work, failing to find parses that are high scoring under the model. This stems from the probabilities of the actions NT($X$) for all labels $X$ almost always being greater than the probability of GEN($w$) for the particular word $w$ which must be produced next in a given sentence. Qualitatively, the search procedure prefers to open constituents repeatedly up until the maximum number allowed by the model. While these long chains of non-terminals will usually have lower probability than the correct sequence at the point where they finally generate the next word, they often have higher probability up until the word is generated, and so they tend to push the correct sequence off the beam before this point is reached. This search failure produces very low evaluation performance: with a beam of size $K = 100$, action-synchronous beam search achieves 29.1 F1 for RG and 27.4 F1 for LM on the development set.

### 2.2 Word-synchronous beam search

To deal with this issue, we force partial parse candidates to compete with each other on a word-by-word level, rather than solely on the level of individual actions. The *word-synchronous* beam search we apply is very similar to approximate

[1] The action space for LM differs from RG in two ways: 1) LM has separate reduce actions REDUCE($X$) for each non-terminal $X$, and 2) LM allows any action to have non-zero probability at all times, even those that may be structurally invalid.

| | Word-synchronous beam size, $K_w$ | | | | | |
|---|---|---|---|---|---|---|
| model | 10 | 20 | 40 | 60 | 80 | 100 |
| RG | 74.1 | 80.1 | 85.3 | 87.5 | 88.7 | 89.6 |
| LM | 83.7 | 88.6 | 90.9 | 91.6 | 92.0 | 92.2 |

Table 1: F1 on the development set for word-synchronous beam search when searching in the RNNG generative (RG) and LSTM generative (LM) models. $K_a$ is set to $10 \times K_w$.

decoding procedures developed for other generative models (Henderson, 2003; Titov and Henderson, 2010; Buys and Blunsom, 2015) and can be viewed as a simplified version of the procedure used in the generative top-down parsers of Roark (2001) and Charniak (2010).

In word-synchronous search, we augment the beam state space, identifying beams by tuples $(|W|, |A_w|)$, where $|W|$ is the number of words that have been produced so far in the sentence, and $|A_w|$ is the number of structural actions that have been taken since the last word was produced. Intuitively, we want candidates with the same $|W| = w$ to compete against each other. For a beam of partial parses in the state $(|W| = w, |A_w| = a)$, we generate a beam of successors by taking all of the next possible actions for each partial parse in the beam. If the action is NT($X$) or REDUCE, we place the resulting partial parse in the beam for state $(|W| = w, |A_w| = a + 1)$; otherwise, if the action is GEN, we place it in a list for $(|W| = w + 1, |A_w| = 0)$. After all partial parses in the beam have been processed, we check to see if there are a sufficient number of partial parses that have produced the next word: if the beam $(|W| = w + 1, |A_w| = 0)$ contains at least $K_w$ partial parses (the *word beam size*), we prune it to this size and continue search using this beam. Otherwise, we continue building candidates for this word by pruning the beam $(|W| = w, |A_w| = a + 1)$ to size $K_a$ (the *action beam size*), and continuing search from there.

In practice, we found it to be most effective to use a value for $K_w$ that is a fraction of the value for $K_a$. In all the experiments we present here, we fix $K_a = 10 \times K_w$, with $K_w$ ranging from 10 to 100. Table 1 shows F1 for decoding in both generative models on the development set, using the top-scoring parse found for a sentence when searching with the given beam size. RG has comparatively larger gains in performance between the larger beam sizes, while still underperforming LM, suggesting that more search is necessary in this model.

## 3 Experiments

Using the above decoding procedures, we attempt to separate reranking effects from model combination effects through a set of reranking experiments. Our base experiments are performed on the Penn Treebank (Marcus et al., 1993), using sections 2-21 for training, section 22 for development, and section 23 for testing. For the LSTM generative model (LM), we use the pre-trained model released by Choe and Charniak (2016). We train RNNG discriminative (RD) and generative (RG) models, following Dyer et al. (2016) by using the same hyperparameter settings, and using pre-trained word embeddings from Ling et al. (2015) for the discriminative model. The automatically-predicted part-of-speech tags we use as input for RD are the same as those used by Cross and Huang (2016).

In each experiment, we obtain a set of candidate parses for each sentence by performing beam search in one or more parsers. We use action-synchronous beam search (Section 2.1) with beam size $K = 100$ for RD and word-synchronous beam (Section 2.2) with $K_w = 100$ and $K_a = 1000$ for the generative models RG and LM.

In the case that we are using only the scores from a single generative model to rescore candidates taken from the discriminative parser, this setup is close to the reranking procedures originally proposed for these generative models. For RG, the original work also used RD to produce candidates, but drew samples from it, whereas we use a beam search to approximate its $k$-best list. The LM generative model was originally used to rerank a 50-best list taken from the Charniak parser (Charniak, 2000). In comparison, we found higher performance for the LM model when using a candidate list from the RD parser: 93.66 F1 versus 92.79 F1 on the development data. This may be attributable to having a stronger set of candidates: with beam size 100, RD has an oracle F1 of 98.2, compared to 95.9 for the 50-best list from the Charniak parser.

### 3.1 Augmenting the candidate set

We first experiment with combining the candidate lists from multiple models, which allows us to look for potential model errors and model combination effects. Consider the standard reranking setup B $\rightarrow$ A, where we search in B to get a set of candidate parses for each sentence, and

| Candidates | Scoring models RD | RG | RD + RG |
|---|---|---|---|
| RD | 92.22 | 93.45 | 93.87 |
| RG | 90.24 | 89.55 | 90.53 |
| RD ∪ RG | 92.22 | 92.78 | 93.92 |

| Candidates | Scoring models RD | LM | RD + LM |
|---|---|---|---|
| RD | 92.22 | 93.66 | 93.99 |
| LM | 92.57 | 92.20 | 93.07 |
| RD ∪ LM | 92.24 | 93.47 | 94.15 |

Table 2: Development F1 scores on section 22 of the PTB when using various models to produce candidates and to score them. ∪ denotes taking the union of candidates from each of two models; + denotes using a weighted average of the models' log-probabilities.

choose the top scoring candidate from these under A. We extend this by also searching directly in A to find high-scoring candidates for each sentence, and combining them with the candidate list proposed by B by taking the union, A ∪ B. We then choose the highest scoring candidate from this list under A. If A generally prefers parses outside of the candidate list from B, but these decrease evaluation performance (i.e., if B ∪ A → A is worse than B → A), this suggests a model combination effect is occurring: A makes errors which are hidden by having a limited candidate list from B.

This does seem to be the case for both generative models, as shown in Table 2, which presents F1 scores on the development set when varying the models used to produce the candidates and to score them. Each row is a different candidate set, where the third row in each table presents results for the augmented candidate sets; each column is a different scoring model, where the third column is the *score combination* setting described below. Going from RD → RG to the augmented candidate setting RD ∪ RG → RG decreases performance from 93.45 F1 to 92.78 F1 on the development set. This difference is statistically significant at the $p < 0.05$ level under a paired bootstrap test. We see a smaller, but still significant, effect in the case of LM: RD → LM achieves 93.66, compared to 93.47 for RD ∪ LM → LM.

We can also consider the performance of RG → RG and LM → LM (where we do not use candidates from RD at all, but return the highest-scoring parse from searching directly in one of the generative models) as an indicator of reranking effects: absolute performance is higher for LM (92.20 F1) than for RG (89.55). Taken together, these results suggest that model combination contributes to the success of both models, but to a larger extent for RG. A reranking effect may be a larger contributor to the success of LM, as this model achieves stronger performance on its own for the described search setting.

### 3.2 Score combination

If the cross-scoring setup exhibits an implicit model combination effect, where strong performance results from searching in one model and scoring with the other, we might expect substantial further improvements in performance by explicitly combining the scores of both models. To do so, we score each parse by taking a weighted sum of the log-probabilities assigned by both models (Hayashi et al., 2013), using an interpolation parameter which we tune to maximize F1 on the development set.

These results are given in columns RD + RG and RD + LM in Table 2. We find that combining the scores of both models improves on using the score of either model alone, regardless of the source of candidates. These improvements are statistically significant in all cases. Score combination also more than compensates for the decrease in performance we saw previously when adding in candidates from the generative model: RD ∪ RG → RD + RG improves upon both RD → RG and RD ∪ RG → RG, and the same effect holds for LM.

### 3.3 Strengthening model combination

Given the success of model combination between the base model and a single generative model, we also investigate the hypothesis that the generative models are complementary. The Model Combination block of Table 3 shows full results on the test set for these experiments, in the PTB column. The same trends we observed on the development data, on which the interpolation parameters were tuned, hold here: score combination improves results for all models (row 3 vs. row 2; row 6 vs. row 5), with candidate augmentation from the generative models giving a further increase (rows 4 and 7).[2] Combining candidates and scores from all three models (row 9), we obtain 93.94 F1.

[2]These increases, from adding score combination and candidate augmentation, are all significant with $p < 0.05$ in the PTB setting. In the +S data setting, all are significant except for the difference between row 5 and row 6.

| Model | PTB | +S |
|---|---|---|
| Liu and Zhang (2017) | 91.7 | – |
| Dyer et al. (2016)-discriminative | 91.7 | – |
| Dyer et al. (2016)-generative | 93.3 | – |
| Choe and Charniak (2016) | 92.6 | 93.8 |
| Model Combination | | |
| 1) RD → RD | 91.51 | 91.73 |
| 2) RD → RG | 92.73 | 93.29 |
| 3) RD → RD + RG | 93.27 | 93.64 |
| 4) RD ∪ RG → RD + RG | 93.45 | 93.75 |
| 5) RD → LM | 93.31 | 94.18 |
| 6) RD → RD + LM | 93.71 | 94.27 |
| 7) RD ∪ LM → RD + LM | 93.89 | 94.63 |
| 8) RD → RD + RG + LM | 93.63 | 94.33 |
| 9) RD ∪ RG ∪ LM → RD + RG + LM | **93.94** | **94.66** |
| Ensembling | | |
| 10) RD (8) → RD (8) | 92.72 | 92.53 |
| 11) RD (8) → RD (8) + RG (8) | 94.09 | 94.22 |
| 12) RD (8) → RD (8) + LM | 93.97 | 94.56 |
| 13) RD (8) → RD (8) + RG (8) + LM | **94.25** | **94.62** |

Table 3: Test F1 scores on section 23 of the PTB, by treebank training data conditions: either using only the training sections of the PTB, or using additional silver data (+S).

**Semi-supervised silver data** Choe and Charniak (2016) found a substantial increase in performance by training on external data in addition to trees from the Penn Treebank. This *silver dataset* was obtained by parsing the entire New York Times section of the fifth Gigaword corpus using a product of eight Berkeley parsers (Petrov, 2010) and ZPar (Zhu et al., 2013), then retaining 24 million sentences on which both parsers agreed. For our experiments we train RD and RG using the same silver dataset.[3] The +S column in Table 3 shows these results, where we observe gains over the PTB models in nearly every case. As in the PTB training data setting, using all models for candidates and score combinations is best, achieving 94.66 F1 (row 9).

**Ensembling** Finally, we compare to another commonly used model combination method: ensembling multiple instances of the same model type trained from different random initializations. We train ensembles of 8 copies each of RD and RG in both the PTB and silver data settings, combining scores from models within an ensemble by averaging the models' distributions for each action (in beam search as well as rescoring). These results are shown in the bottom section, Ensembling, of Table 3.

Performance when using only the ensembled RD models (row 10) is lower than rescoring a single RD model with score combinations of single models, either RD + RG (row 3) or RD + LM (row 6). In the PTB setting, ensembling with score combination achieves the best overall result of 94.25 (row 13). In the silver training data setting, while this does improve on the analogous unensembled result (row 8), it is not better than the combination of single models when candidates from the generative models are also included (row 9).

## 4 Discussion

Searching directly in the generative models yields results that are partly surprising, as it reveals the presence of parses which the generative models prefer, but which lead to lower performance than the candidates proposed by the base model. However, the results are also unsurprising in the sense that explicitly combining scores allows the reranking setup to achieve better performance than implicit combination, which uses only the scores of a single model. Additionally, we see support for the hypothesis that the generative models can achieve good results on their own, with the LSTM generative model showing particularly strong and self-contained performance.

While this search procedure allows us to explore these generative models, disentangling reranking and model combination effects, the increase in performance from augmenting the candidate lists with the results of the search may not be worth the required computational cost in a practical parser. However, we do obtain a gain over state-of-the-art results using simple model score combination on only the base candidates, which can be implemented with minimal cost over the basic reranking setup. This provides a concrete improvement for these particular generative reranking procedures for parsing. More generally, it supports the idea that hybrid systems, which rely on one model to produce a set of candidates and another to determine which candidates are good, should explore combining their scores and candidates when possible.

[3] When training with silver data, we use a 1-to-1 ratio of silver data updates per gold data updates, which we found to give significantly faster convergence times on development set perplexity for RD and RG compared to the 10-to-1 ratio used by Choe and Charniak (2016) for LM.

## Acknowledgments

We would like to thank Adhiguna Kuncoro and Do Kook Choe for their help providing data and answering questions about their work, as well as Jacob Andreas, John DeNero, and the anonymous reviewers for their suggestions. DF is supported by an NDSEG fellowship. MS is supported by an NSF Graduate Research Fellowship.

## References

Jan Buys and Phil Blunsom. 2015. Generative incremental dependency parsing with neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*.

Eugene Charniak. 2010. Top-down nearly-context-sensitive parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics* 1:139–150.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Jiangming Liu and Yue Zhang. 2017. Shift-reduce constituent parsing with neural lookahead features. *Transactions of the Association for Computational Linguistics* 5:45–58.

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

Slav Petrov. 2010. Products of random latent variable grammars. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics* 27(2):249–276.

Ivan Titov and James Henderson. 2010. A latent variable model for generative dependency parsing. In *Trends in Parsing Technology*, Springer, pages 35–55.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*. pages 2773–2781.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.