Perceptron Reranking for CCG Realization

Michael White and **Rajakrishnan Rajkumar**
Department of Linguistics
The Ohio State University
Columbus, OH, USA
{mwhite,raja}@ling.osu.edu

Abstract

This paper shows that discriminative reranking with an averaged perceptron model yields substantial improvements in realization quality with CCG. The paper confirms the utility of including language model log probabilities as features in the model, which prior work on discriminative training with log linear models for HPSG realization had called into question. The perceptron model allows the combination of multiple n-gram models to be optimized and then augmented with both syntactic features and discriminative n-gram features. The full model yields a state-of-the-art BLEU score of 0.8506 on Section 23 of the CCGbank, to our knowledge the best score reported to date using a reversible, corpus-engineered grammar.

1 Introduction

In this paper, we show how discriminative training with averaged perceptron models (Collins, 2002) can be used to substantially improve surface realization with Combinatory Categorial Grammar (Steedman, 2000, CCG). Velldal and Oepen (2005) and Nakanishi et al. (2005) have shown that discriminative training with log-linear (maximum entropy) models is effective in realization ranking with Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994, HPSG). Here we show that averaged perceptron models also perform well for realization ranking with CCG. Averaged perceptron models are very simple, just requiring a decoder and a simple update function, yet despite their simplicity they have been shown to achieve state-of-the-art results in Treebank and CCG parsing (Huang, 2008; Clark and Curran, 2007a) as well as on other NLP tasks.

Along the way, we address the question of whether it is beneficial to incorporate n-gram log probabilities as baseline features in a discriminatively trained realization ranking model. On a limited domain corpus, Velldal & Oepen found that including the n-gram log probability of each candidate realization as a feature in their log-linear model yielded a substantial boost in ranking performance; on the Penn Treebank (PTB), however, Nakanishi et al. found that including an n-gram log prob feature in their model was of no benefit (with the use of bigrams instead of 4-grams suggested as a possible explanation). With these mixed results, the utility of n-gram baseline features for PTB-scale discriminative realization ranking has been unclear. In our particular setting, the question is: Do n-gram log prob features improve performance in broad coverage realization ranking with CCG, where factored language models over words, part-of-speech tags and supertags have previously been employed (White et al., 2007; Espinosa et al., 2008)?

We answer this question in the affirmative, confirming the results of Velldal & Oepen, despite the differences in corpus size and kind of language model. We show that including n-gram log prob features in the perceptron model is highly beneficial, as the discriminative models we tested without these features performed worse than the generative baseline. These findings are in line with Collins & Roark's (2004) results with incremental parsing with perceptrons, where it is suggested that a generative baseline feature provides the perceptron algorithm with a much better starting point for learning. We also show that discriminative training allows the combination of multiple n-gram models to be optimized, and that the best model augments the n-gram log prob features with both syntactic features and discriminative n-gram features. The full model yields a state-of-the-art BLEU (Papineni et al., 2002) score of 0.8506 on Section 23 of the CCGbank, which is to our knowledge the best score reported to date using a reversible, corpus-engineered grammar.

The paper is organized as follows. Section 2 reviews previous work on broad coverage realization with OpenCCG. Section 3 describes our approach to realization reranking with averaged perceptron models. Section 4 presents our evaluation of the perceptron models, comparing the results of different feature sets. Section 5 compares our results to those obtained by related systems and discusses the difficulties of cross-system comparisons. Finally, Section 6 concludes with a summary and discussion of future directions for research.

2 Background

2.1 Surface Realization with CCG

CCG (Steedman, 2000) is a unification-based categorial grammar formalism which is defined almost entirely in terms of lexical entries that encode sub-categorization information as well as syntactic feature information (e.g. number and agreement). Complementing function application as the standard means of combining a head with its argument, type-raising and composition support transparent analyses for a wide range of phenomena, including right-node raising and long distance dependencies. An example syntactic derivation appears in Figure 1, with a long-distance dependency between *point* and *make*. Semantic composition happens in parallel with syntactic composition, which makes it attractive for generation.

OpenCCG is a parsing/generation library which works by combining lexical categories for words using CCG rules and multi-modal extensions on rules (Baldridge, 2002) to produce derivations. Surface realization is the process by which logical forms are transduced to strings. OpenCCG uses a hybrid symbolic-statistical chart realizer (White, 2006) which takes logical forms as input and produces sentences by using CCG combinators to combine signs. Edges are grouped into equivalence classes when they have the same syntactic category and cover the same parts of the input logical form. Alternative realizations are ranked using integrated n-gram or perceptron scoring, and pruning takes place within equivalence classes of edges. To more robustly support broad coverage surface realization, OpenCCG greedily assembles fragments in the event that the realizer fails to find a complete realization.

To illustrate the input to OpenCCG, consider the semantic dependency graph in Figure 2. In

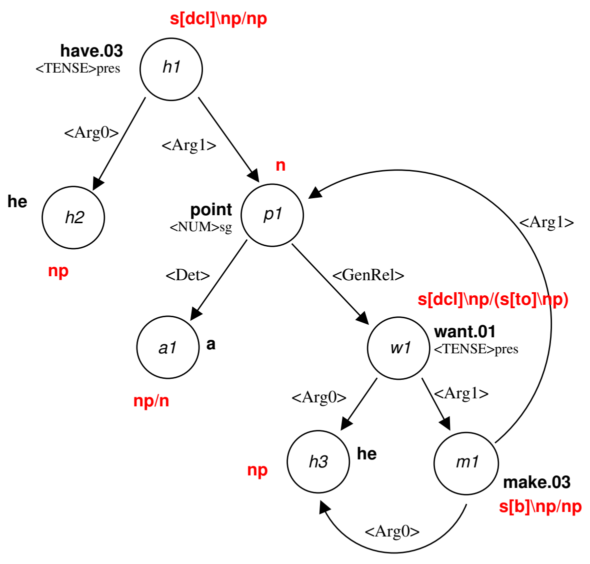

Figure 2: Semantic dependency graph from the CCGbank for *He has a point he wants to make [. . .]*, along with gold-standard supertags (category labels)

the graph, each node has a lexical predication (e.g. **make.03**) and a set of semantic features (e.g. ⟨NUM⟩sg); nodes are connected via dependency relations (e.g. ⟨ARG0⟩). (Gold-standard supertags, or category labels, are also shown; see Section 2.4 for their role in hypertagging.) Internally, such graphs are represented using Hybrid Logic Dependency Semantics (HLDS), a dependency-based approach to representing linguistic meaning (Baldridge and Kruijff, 2002). In HLDS, each semantic head (corresponding to a node in the graph) is associated with a nominal that identifies its discourse referent, and relations between heads and their dependents are modeled as modal relations.

2.2 Realization from an Enhanced CCGbank

Our starting point is an enhanced version of the CCGbank (Hockenmaier and Steedman, 2007)—a corpus of CCG derivations derived from the Penn Treebank—with Propbank (Palmer et al., 2005) roles projected onto it (Boxwell and White, 2008). To engineer a grammar from this corpus suitable for realization with OpenCCG, the derivations are first revised to reflect the lexicalized treatment of coordination and punctuation assumed by the multi-modal version of CCG that is implemented in OpenCCG (White and Rajkumar, 2008). Further changes are necessary to support semantic dependencies rather than surface syntactic ones; in

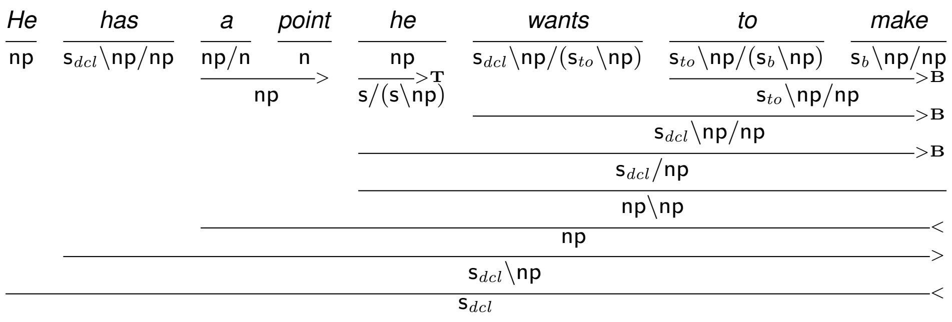

Figure 1: Syntactic derivation from the CCGbank for *He has a point he wants to make [. . .]*

particular, the features and unification constraints in the categories related to semantically empty function words such complementizers, infinitival-*to*, expletive subjects, and case-marking prepositions are adjusted to reflect their purely syntactic status.

In the second step, a grammar is extracted from the converted CCGbank and augmented with logical forms. Categories and unary type changing rules (corresponding to zero morphemes) are sorted by frequency and extracted if they meet the specified frequency thresholds. A separate transformation then uses a few dozen generalized templates to add logical forms to the categories, in a fashion reminiscent of (Bos, 2005). As shown in Figure 2, numbered semantic roles are taken from PropBank when available, and more specific relations are introduced in the categories for closed-class items such as determiners.

After logical form insertion, the extracted and augmented grammar is loaded and used to parse the sentences in the CCGbank according to the gold-standard derivation. If the derivation can be successfully followed, the parse yields a logical form which is saved along with the corpus sentence in order to later test the realizer. Currently, the algorithm succeeds in creating logical forms for 98.85% of the sentences in the development section (Sect. 00) of the converted CCGbank, and 97.06% of the sentences in the test section (Sect. 23). Of these, 95.99% of the development LFs are semantic dependency graphs with a single root, while 95.81% of the test LFs have a single root. The remaining cases, with multiple roots, are missing one or more dependencies required to form a fully connected graph. Such missing dependencies usually reflect remaining inadequacies in the logical form templates.

An error analysis of OpenCCG output by Rajkumar et al. (2009) recently revealed that out of 2331 named entities (NEs) annotated by the BBN corpus (Weischedel and Brunstein, 2005), 238 were not realized correctly. For example, multi-word NPs like *Texas Instruments Japan Ltd.* were realized as *Japan Texas Instruments Ltd.* Accordingly, inspired by Hogan et al.'s (2007)'s Experiment 1, Rajkumar et al. used the BBN corpus NE annotation to collapse certain classes of NEs. But unlike Hogan et al.'s experiment where all the NEs annotated by the BBN corpus were collapsed, Rajkumar et al. chose to collapse into single tokens only NEs whose exact form can be reasonably expected to be specified in the input to the realizer. For example, while some quantificational or comparatives phrases like *more than $ 10,000* are annotated as MONEY in the BBN corpus, Rajkumar et al. only collapse *$_10,000* into an atomic unit, with *more than* handled compositionally according to the semantics assigned to it by the grammar. Thus, after transferring the BBN annotations to the CCGbank corpus, Rajkumar et al. (partially) collapsed NEs which are CCGbank constituents according to the following rules: (1) completely collapse the PERSON, ORGANIZATION, GPE, WORK OF ART major class type entitites; (2) ignore phrases like *three decades later*, which are annotated as DATE entities; and (3) collapse all phrases with POS tags CD or NNP(S) or lexical items *%* or *$*, ensuring that all prototypical named entities are collapsed.

It is worth noting that improvements in our corpus-based grammar engineering process—including a more precise treatment of punctuation, better named entity handling and the addition of catch-all logical form templates—have resulted in a 13.5 BLEU point improvement in our baseline realization scores on Section 00 of the CCGbank, from a score of 0.6567 in (Espinosa et al., 2008) to 0.7917 in (Rajkumar et al., 2009), contributing greatly to the state-of-the-art results reported in Section 4. A further 4.5 point improvement is obtained from the use of named entity classes in language modeling and hypertagging (Rajkumar et al., 2009), as described next, and from our perceptron reranking model, described in Section 3.

2.3 Factored Language Models

As in (White et al., 2007; Rajkumar et al., 2009), we use factored language models (Bilmes and Kirchhoff, 2003) over words, part-of-speech tags and supertags[1] to score partial and complete realizations. The trigram models were created using the SRILM toolkit (Stolcke, 2002) on the standard training sections (02–21) of the CCGbank, with sentence-initial words (other than proper names) uncapitalized. While these models are considerably smaller than the ones used in (Langkilde-Geary, 2002; Velldal and Oepen, 2005), the training data does have the advantage of being in the same domain and genre. The models employ interpolated Kneser-Ney smoothing with the default frequency cutoffs. The best performing model interpolates three component models using rank-order centroid weights: (1) a word trigram model; (2) a word model with semantic classes replacing named entities; and (3) a trigram model that chains a POS model with a supertag model, where the POS model (P) conditions on the previous two POS tags, and the supertag model (S) conditions on the previous two POS tags as well as the current one, as shown below:

$$p^{PS}(\vec{F}_i \mid \vec{F}_{i-2}^{i-1}) = p(P_i \mid P_{i-2}^{i-1})p(S_i \mid P_{i-2}^{i}) \quad (1)$$

Training data for the semantic class–replaced model was created by replacing (collapsed) words with their NE classes, in order to address data sparsity issues caused by rare words in the same semantic class. For example, the Section 00 sentence *Pierre_Vinken , 61 years old , will join the board as a nonexecutive director Nov._29 .* becomes *PERSON , DATE:AGE DATE:AGE old , will join the ORG_DESC:OTHER as a nonexecutive PER_DESC DATE:DATE DATE:DATE .* During realization, word forms are generated, but are then replaced by their semantic classes for scoring using the semantic class–replaced model, similar to Oh and Rudnicky (2002).

Note that the use of supertags in the factored language model to score possible realizations is distinct from the prediction of supertags for lexical category assignment: the former takes the words in the local context into account (as in supertagging for parsing), while the latter takes features of the logical form into account. This latter process we call hypertagging, to which we now turn.

2.4 Hypertagging

A crucial component of the OpenCCG realizer is the *hypertagger* (Espinosa et al., 2008), or supertagger for surface realization, which uses a maximum entropy model to assign the most likely lexical categories to the predicates in the input logical form, thereby greatly constraining the realizer's search space.[2] Figure 2 shows gold-standard supertags for the lexical predicates in the graph; such category labels are predicted by the hypertagger at run-time. As in recent work on using supertagging in parsing, the hypertagger operates in a multitagging paradigm (Curran et al., 2006), where a variable number of predictions are made per input predicate. Instead of basing category assignment on linear word and POS context, however, the hypertagger predicts lexical categories based on contexts within a directed graph structure representing the logical form (LF) of the sentence to be realized. The hypertagger generalizes Bangalore and Rambow's (2000) method of using supertags in generation by using maximum entropy models with a larger local context.

During realization, the hypertagger returns a β-best list of supertags in order of decreasing probability. Increasing the number of categories returned clearly increases the likelihood that the most-correct supertag is among them, but at a corresponding cost in chart size. Accordingly, the hypertagger begins with a highly restrictive value for β, and backs off to progressively less-restrictive values if no complete realization can be found using the set of supertags returned. Clark and Curran (2007b) have shown this iterative relaxation strategy to be highly effective in CCG parsing.

3 Perceptron Reranking

As Collins (2002) observes, perceptron training involves a simple, on-line algorithm, with few iterations typically required to achieve good performance. Moreover, *averaged* perceptrons—which

[1]With CCG, supertags (Bangalore and Joshi, 1999) are lexical categories considered as fine-grained syntactic labels.

[2]The approach has been dubbed *hypertagging* since it operates at a level "above" the syntax, moving from semantic representations to syntactic categories.

```
**Input:** training examples $(x_i, y_i)$
**Initialization:** set $\overline{\alpha} = 0$, or use optional input model
**Algorithm:**
    for $t = 1 \ldots T, i = 1 \ldots N$
        $z_i = \text{argmax}_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \overline{\alpha}$
        if $z_i \neq y_i$
            $\overline{\alpha} = \overline{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
**Output:** $\overline{\alpha} = \sum_{t=1}^{T} \sum_{i=1}^{N} \overline{\alpha}_{ti} / TN$
```

Figure 3: Averaged perceptron training algorithm

approximate voted perceptrons, a maximum-margin method with attractive theoretical properties—seem to work remarkably well in practice, while adding little further complexity. Additionally, since features only take on non-zero values when they appear in training items requiring updates, perceptrons integrate feature selection with, and often produce quite small models, especially when starting with a good baseline.

The generic averaged perceptron training algorithm appears in Figure 3. In our case, the algorithm trains a model for reranking the n-best realizations generated using our existing factored language model for scoring, with the oracle-best realization considered the correct answer. Accordingly, the input to the algorithm is a list of pairs (x_i, y_i), where x_i is a logical form, GEN(x_i) are the n-best realizations for x_i, and y_i is the oracle-best member of GEN(x_i). The oracle-best realization is determined using a 4-gram precision metric (approximating BLEU) against the reference sentence.

We have followed Huang (2008) in using oracle-best targets for training, rather than gold standard ones, in order to better approximate test conditions during training. However, following Clark & Curran (2007a), during training we seed the realizer with the gold-standard supertags, augmenting the hypertagger's β-best list, in order to ensure that the n-best realizations are generally of high quality; consequently, the gold standard realization (i.e., the corpus sentence) usually appears in the n-best list.[3] In addition, we use a hypertagger trained on all the training data, to improve hypertagger performance, while excluding the current training section (in jack-knifed fashion) from the word-based parts of the language model, in order to make the language model scores more realistic. It remains for future work to determine whether using a different compromise between ensuring high-quality training data and remaining faithful to the test conditions would yield better results.

Since realization of the n-best lists for training is the most time-consuming part of the process, in our current implementation we perform this step once, generating event files along the way containing feature vectors for each candidate realization. The event files are used to calculate the frequency distribution for the features, and minimum cutoffs are chosen to trim the feature alphabet to a reasonable size. Training then takes place by iterating over the event files, ignoring features that do not appear in the alphabet. As Figure 3 indicates, training consists of calculating the top-ranked realization according to the current model $\overline{\alpha}$, and performing an update when the top-ranked realization does not match the oracle-best realization. Updates to the model add the feature vector $\Phi(x_i, y_i)$ for the missed oracle-best realization, and subtract the feature vector $\Phi(x_i, z_i)$ for the mistakenly top-ranked realization. The final model averages the models across the T iterations over the training data, and N test cases within each iteration.

Note that while training the perceptron model involves n-best reranking, realization with the resulting model can be viewed as forest rescoring, since scoring of all partial realizations is integrated into the realizer's beam search. In future work, we intend to investigate saving the realizer's packed charts, rather than event files, and integrating the unpacking of the charts with the perceptron training algorithm.

The features we employ in our perceptron models are of three kinds. First, as in the log-linear models of Velldal & Oepen and Nakanishi et al., we incorporate the log probability of the candidate realization's word sequence according to our factored language model as a single feature in the perceptron model. Since our language model linearly interpolates three component models, we also include the log prob from each component language model as a feature, so that the combination of these components can be optimized.

Second, we include syntactic features in our

[3]As in Clark & Curran's approach, we use a single β value during training, rather than iteratively loosening the β value; the chosen β value determines the size of the discrimation space.

Feature Type	Example
LexCat + Word	$\mathsf{s/s/np}$ + before
LexCat + POS	$\mathsf{s/s/np}$ + IN
Rule	$\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}$
Rule + Word	$\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}$ + bought
Rule + POS	$\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}$ + VBD
Word-Word	$\langle$company, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}$, bought$\rangle$
Word-POS	$\langle$company, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}$, VBD$\rangle$
POS-Word	$\langle$NN, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}$, bought$\rangle$
Word + Δ_w	$\langle$bought, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}\rangle$ + d_w
POS + Δ_w	$\langle$VBD, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}\rangle$ + d_w
Word + Δ_p	$\langle$bought, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}\rangle$ + d_p
POS + Δ_p	$\langle$VBD, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}\rangle$ + d_p
Word + Δ_v	$\langle$bought, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}\rangle$ + d_v
POS + Δ_v	$\langle$VBD, $\mathsf{s}_{dcl} \rightarrow \mathsf{np}\ \mathsf{s}_{dcl}\backslash\mathsf{np}\rangle$ + d_v

Table 1: Basic and dependency features from Clark & Curran's (2007b) normal form model; distances are in intervening words, punctuation marks and verbs, and are capped at 3, 3 and 2, respectively

model by implementing Clark & Curran's (2007b) normal form model in OpenCCG.[4] The features of this model are listed in Table 1; they are integer-valued, representing counts of occurrences in a derivation. These syntactic features are quite comparable to the dominance-oriented features in the union of the Velldal & Oepen and Nakanishi et al. models, except that our feature set does not include grandparenting, which has been found to have limited utility in CCG parsing. Our syntactic features also include ones that measure the distance between headwords in terms of intervening words, punctuation marks or verbs; these features generalize the ones in Nakanishi et al.'s model. Note that in contrast to parsing, in realization distance features are non-local, since different partial realizations in the same equivalence class typically differ in word order; as we are working in a reranking paradigm though, the non-local nature of these features is unproblematic.

Third, we include discriminative n-gram features in our model, following Roark et al.'s (2004) approach to discriminative n-gram modeling for speech recognition. By discriminative n-gram features, we mean features counting the occurrences of each n-gram that is scored by our factored language model, rather than a feature whose value is the log prob determined by the language model. As Roark et al. note, discriminative training with n-gram features has the potential to learn to negatively weight n-grams that appear in some of the GEN(x_i) candidates, but which never appear in the naturally occurring corpus used to train a standard, generative language model. Since our factored language model considers words, semantic classes, part-of-speech tags and supertags, our n-gram features represent a considerable generalization of the sequence-oriented features in Velldal & Oepen's model, which never contain more than one word and do not include semantic classes.

[4]We have omitted Clark & Curran's root features, since the category we use for the full stop ensures that it must appear at the root of any complete derivation.

Model	#Alph-feats	#Feats	Acc	Time
full-model	2402173	576176	96.40%	08:53
lp-ngram	1127437	342025	94.52%	05:19
lp-syn	1274740	291728	85.03%	05:57

Table 2: Perceptron Training Details—number of features in the alphabet, number of features in the model, training accuracy and training time (hours) for 10 iterations on a single commodity server

4 Evaluation

4.1 Experimental Conditions

For the experiments reported below, we used a lexico-grammar extracted from Sections 02–21 of our enhanced CCGbank, a hypertagging model incorporating named entity class features, and a trigram factored language model over words, named entity classes, part-of-speech tags and supertags, as described in the preceding section. BLEU scores were calculated after removing the underscores between collapsed NEs.

Events were generated for each training section separately. As already noted, the hypertagger and POS/supertag language model was trained on all the training sections, while separate word-based models were trained excluding each of the training sections in turn. Event files for 26530 training sentences with complete realizations were generated in 7 hours and 16 minutes on a cluster using one commodity server per section, with an average n-best list size of 18.2. Perceptron models were trained on single machines; details for three of the models appear in Table 2. The complete set of models is listed in Table 3.

4.2 Results

Realization results on the development section are given in Table 4. As the first block of rows after the baseline shows, of the models incorporating a single kind of feature, only the one with the n-gram log prob features beats the baseline BLEU

Model	Description
baseline-w3	No perceptron (3g wd only)
baseline	No perceptron
syn-only-nodist	All syntactic features except distance
ngram-only	Just ngram features
syn-only	Just syntactic features
lp-only	Just log prob features
lp-ngram	Log prob + Ngram features
lp-syn	Log prob + Syntactic features
full-model	Log prob + Ngram +Syntactic features

Table 3: Legend for Experimental Conditions

score, with the other models falling well below the baseline (though faring better than the trigram-word LM baseline). This result confirms the importance of including n-gram log prob features in discriminative realization ranking models, in line with Velldal & Oepen's findings, and contra those of Nakanishi et al., even though it was Nakanishi et al. who experimented with the Penn Treebank corpus, while Velldal & Oepen's experiments were on a much smaller, limited domain corpus. The second block of rows shows that both the discriminative n-gram features and the syntactic features provide a substantial boost when used with the n-gram log prob features, with the syntactic features yielding a more than 3 BLEU point gain. The final row shows that the full model works best, though the boosts provided by the syntactic and discriminative n-gram features are clearly not independent. The BLEU point trends are mirrored in the percentage of exact match realizations, which goes up by more than 10% from the baseline. The percentage of complete (i.e., non-fragmentary) realizations, however, goes down; we expect that this is due to the time taken up by our current naive method of feature extraction, which does not cache the features calculated for partial realizations. Realization results on the standard test section appear in Table 5, confirming the gains made by the full model over the baseline.[5]

We calculated statistical significance for the main results on the development section using bootstrap random sampling.[6] After re-sampling 1000 times, significance was calculated using a paired t-test (999 d.f.). The results indicated that lp-only exceeded the baseline, lp-ngram and lp-syn exceeded lp-only, and the full model exceeded lp-syn, with $p < 0.0001$ in each case.

[5]Note that the baseline for Section 23 uses 4-grams and a filter for balanced punctuation (White and Rajkumar, 2008), unlike the other reported configurations, which would explain the somewhat smaller increase seen with this section.

[6]Scripts for running these tests are available at `http://projectile.sv.cmu.edu/research/public/tools/bootStrap/tutorial.htm`

Model	%Exact	%Compl.	BLEU	Time
baseline-w3	26.00	83.15	0.7646	1.8
baseline	29.00	83.28	0.7963	2.0
syn-only-nodist	26.02	82.69	0.7754	3.2
ngram-only	27.67	82.95	0.7777	3.0
syn-only	28.34	82.74	0.7838	3.4
lp-only	32.01	83.02	0.8009	2.1
lp-ngram	36.31	80.47	0.8183	3.1
lp-syn	39.47	79.74	0.8323	3.5
full-model	**40.11**	79.63	**0.8373**	3.6

Table 4: Section 00 Results (98.9% coverage)—percentage of exact match and grammatically complete realizations, BLEU scores and average times, in seconds

Model	%Exact	%Complete	BLEU
baseline	33.74	85.04	0.8173
full-model	**40.45**	83.88	**0.8506**

Table 5: Section 23 Results (97.06% coverage)

4.3 Examples

Table 6 presents four examples where the full model improves upon the baseline. Example sentence wsj_0020.10 in Table 6 is a case where the perceptron successfully weights the component ngram models, as the lp-ngram model and those that build on it get it right. Note that here, the modifier ordering in *small video-viewing* is not specified in the logical form and either ordering is possible syntactically. In wsj_0024.2, number agreement between the conjoined subject noun phrase and verb is obtained only with the full model. This suggests that the full model is more robust to cases where the grammar is insufficiently precise (number agreement is enforced by the grammar in only the simplest cases). Example wsj_0034.9 corrects a VP ordering mismatch, where the corpus sentence is clearly preferred to the one where *into oblivion* is shifted to the end. Finally, wsj_0047.13 corrects an animacy mismatch on the *wh*-pronoun, in large part due to the high negative weight assigned to the discriminative n-gram feature PERSON_,_which. Note that the full model still differs from the original sentence in its placement of the adverb *reportedly*, choosing the arguably more natural position following the auxiliary.

4.4 Comparison to Other Systems

Table 7 lists our results in the context of those reported for other systems on PTB Section 23. The

Ref-wsj_0020.10	that measure could compel Taipei 's growing number of small video-viewing parlors to pay ...
baseline,syn-only,ngram-only	that measure could compel Taipei 's growing number of *video-viewing small* parlors to ...
lp-only, lp-ngram, full-model	that measure could compel Taipei 's growing number of **small video-viewing** parlors to ...
Ref-wsj_0024.2	Esso Australia Ltd. , a unit of new york-based Exxon Corp. , and Broken Hill Pty. operate the fields ...
all except full-model	Esso Australia Ltd. , a unit of new york-based Exxon Corp. , and Broken Hill Pty. *operates* the fields ...
full-model	Esso Australia Ltd. , a unit of new york-based Exxon Corp. , and Broken Hill Pty. **operate** the fields ...
Ref-wsj_0034.9	they fell into oblivion after the 1929 crash .
baseline, lp-ngram	they fell *after the 1929 crash into oblivion* .
lp-only, ngram-only, syn-only, full-model	they fell **into oblivion after the 1929 crash** .
Ref-wsj_0047.13	Antonio Novello , whom Mr. Bush nominated to serve as surgeon general , reportedly has assured . . .
baseline,baseline-w3, lp-syn, lp-only	Antonio Novello , *which* Mr. Bush nominated to serve as surgeon general , has *reportedly* assured . . .
full-model, lp-ngram, syn-only, ngram-syn	Antonio Novello , **whom** Mr. Bush nominated to serve as surgeon general , has *reportedly* assured . . .

Table 6: Examples of realized output

System	Coverage	BLEU	%Exact
Callaway (05)	98.5%	0.9321	57.5
OpenCCG (09)	97.1%	0.8506	40.5
Ringger et al. (04)	100.0%	0.836	35.7
Langkilde-Geary (02)	83%	0.757	28.2
Guo et al. (08)	100.0%	0.7440	19.8
Hogan et al. (07)	≈100.0%	0.6882	–
OpenCCG (08)	96.0%	0.6701	16.0
Nakanishi et al. (05)	90.8%	0.7733	–

Table 7: PTB Section 23 BLEU scores and exact match percentages in the NLG literature (Nakanishi et al.'s results are for sentences of length 20 or less)

most similar systems to ours are those of Nakanishi et al. (2005) and Hogan et al. (2007), as they both involve chart realizers for reversible grammars engineered from the Penn Treebank. While direct comparisons across systems cannot really be made when inputs vary in their semantic depth and specificity, we observe that our all-sentences BLEU score of 0.8506 exceeds that of Hogan et al., who report a top score of 0.6882 (though with coverage near 100%), and also surpasses Nakanishi et al.'s score of 0.7733, despite their results being limited to sentences of length 20 or less (with 91% coverage). Velldal & Oepen's (2005) system is also closely related, as noted in the introduction, but as their experiments are on a limited domain corpus, their results cannot be compared at all meaningfully.

5 Related Work and Discussion

As alluded to above, realization systems cannot be easily compared, even on the same corpus, when their inputs are not the same. This point is dramatically illustrated in Langkilde-Geary's (2002) system, where a BLEU score of 0.514 is reported for minimally specified inputs on PTB Section 23, while a score of 0.757 is reported for the 'Permute, no dir' case (which perhaps most closely resembles our inputs), and a score of 0.924 is reported for the most fully specified inputs; note, however, that in the latter case word order is determined by sibling order in the inputs, an assumption not commonly made. As another example, Guo et al. (2008) report a competitive result of 0.7440 (with 100% coverage) using a dependency-based approach; however, their inputs, like those of Hogan et al., include more surface syntactic information than ours, as they specify case-marking prepositions, *wh*-pronouns and complementizers. In a recent experiment to assess the impact of input specificity, we found that including predicates for all prepositions in our logical forms boosted our baseline results by more than 3 BLEU points, with complete realizations found in more than 90% of the test cases, indicating that generating from a more surfacy input is indeed an easier task than generating from a deeper representation. Given the current lack of consensus on realizer input specificity, we believe it is important to keep in mind that within-system comparisons (such as those in the preceding section) are the ones that should be given the most credence.

Returning to our cross-system comparison, it is perhaps surprising that Callaway (2005) reports the best PTB BLEU score to date, 0.9321, with 98.5% coverage, using a purely symbolic, hand-crafted grammar augmented to handle the most frequent coverage issues for the PTB. While Callaway's inputs are unordered, word order is often determined by positional features (e.g. `front`) or by the type of modification (e.g. `describer` vs. `qualifier`), and parts-of-speech are included for lexical items. Additionally, in contrast to our approach, Callaway makes use of a generation-only grammar, rather than a reversible one, and his approach is less well-suited to producing n-best outputs. Nevertheless, his high scores do suggest the potential for precise grammar engineering to improve realization quality.

While we have yet to perform a thorough error analysis, our impression is that although the current set of syntactic features substantially improves clausal constituent ordering, a variety of disfluent cases remain. More thorough investigations of features for constituent ordering in English have been performed by Ringger et al. (2004), Filippova and Strube (2009) and Zhong and Stent (2009), all of whom develop classifiers for determining linear order. In future work, we plan to investigate whether features inspired by these approaches can be usefully integrated into our perceptron reranker.

Also related to the present work is discriminative training in syntax-based MT (Turian et al., 2007; Watanabe et al., 2007; Blunsom et al., 2008; Chiang et al., 2009). Not surprisingly, since MT is a harder problem than surface realization, syntax-based MT systems have made use of less precise grammars and more impoverished (target-side) feature sets than those tackling realization ranking. With progress on discriminative training with large numbers of features in syntax-based MT, the features found to be useful for high-quality surface realization may become increasingly relevant for MT as well.

6 Conclusions

In this paper, we have shown how discriminative reranking with an averaged perceptron model can be used to achieve substantial improvements in realization quality with CCG. Using a comprehensive feature set, we have also confirmed the utility of including language model log probabilities as features in the model, which prior work on discriminative training with log linear models for HPSG realization had called into question. The perceptron model allows the combination of multiple n-gram models to be optimized and then augmented with both syntactic features and discriminative n-gram features, inspired by related work in discriminative parsing and language modeling for speech recognition. The full model yields a state-of-the-art BLEU score of 0.8506 on Section 23 of the CCGbank, to our knowledge the best score reported to date using a reversible, corpus-engineered grammar, despite our use of deeper, less specific inputs. Finally, the perceptron model paves the way for exploring the utility of richer feature spaces in statistical realization, including the use of linguistically-motivated and non-local features, a topic which we plan to investigate in future work.

Acknowledgements

This work was supported in part by NSF grant IIS-0812297 and by an allocation of computing time from the Ohio Supercomputer Center. Our thanks also to the OSU Clippers group and the anonymous reviewers for helpful comments and discussion.

References

Jason Baldridge and Geert-Jan Kruijff. 2002. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proc. ACL-02*.

Jason Baldridge. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proc. COLING-00*.

Jeff Bilmes and Katrin Kirchhoff. 2003. Factored language models and general parallelized backoff. In *Proc. HLT-03*.

Phil Blunsom, Trevor Cohn, and Miles Osborne. 2008. A discriminative latent variable model for statistical machine translation. In *Proc. ACL-08: HLT*.

Johan Bos. 2005. Towards wide-coverage semantic interpretation. In *Proc. IWCS-6*.

Stephen Boxwell and Michael White. 2008. Projecting Propbank roles onto the CCGbank. In *Proc. LREC-08*.

Charles Callaway. 2005. The types and distributions of errors in a wide coverage surface realizer evaluation. In *Proceedings of the 10th European Workshop on Natural Language Generation*.

David Chiang, Kevin Knight, and Wei Wang. 2009. 11,001 new features for statistical machine translation. In *Proc. NAACL HLT 2009*.

Stephen Clark and James Curran. 2007a. Perceptron training for a wide-coverage lexicalized-grammar parser. In *ACL 2007 Workshop on Deep Linguistic Processing*.

Stephen Clark and James R. Curran. 2007b. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. ACL-04*.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. EMNLP-02*.

James R. Curran, Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proc. COLING/ACL-06*.

Dominic Espinosa, Michael White, and Dennis Mehay. 2008. Hypertagging: Supertagging for surface realization with CCG. In *Proc. ACL-08: HLT*.

Katja Filippova and Michael Strube. 2009. Tree linearization in English: Improving language model based approaches. In *Proc. NAACL HLT 2009 Short Papers*.

Yuqing Guo, Josef van Genabith, and Haifeng Wang. 2008. Dependency-based n-gram models for general purpose sentence realisation. In *Proc. COLING-08*.

Julia Hockenmaier and Mark Steedman. 2007. CCG-bank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Deirdre Hogan, Conor Cafferkey, Aoife Cahill, and Josef van Genabith. 2007. Exploiting multi-word units in history-based probabilistic generation. In *Proc. EMNLP-CoNLL*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. ACL-08: HLT*.

Irene Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proc. INLG-02*.

Hiroko Nakanishi, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic methods for disambiguation of an HPSG-based chart generator. In *Proc. IWPT-05*.

Alice H. Oh and Alexander I. Rudnicky. 2002. Stochastic natural language generation for spoken dialog systems. *Computer, Speech & Language*, 16(3/4):387–407.

Martha Palmer, Dan Gildea, and Paul Kingsbury. 2005. The proposition bank: A corpus annotated with semantic roles. *Computational Linguistics*, 31(1).

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA.

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

Rajakrishnan Rajkumar, Michael White, and Dominic Espinosa. 2009. Exploiting named entity classes in CCG surface realization. In *Proc. NAACL HLT 2009 Short Papers*.

Eric Ringger, Michael Gamon, Robert C. Moore, David Rojas, Martine Smets, and Simon Corston-Oliver. 2004. Linguistically informed statistical models of constituent structure for ordering in sentence realization. In *Proc. COLING-04*.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. 2004. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proc. ACL-04*.

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

Andreas Stolcke. 2002. SRILM — An extensible language modeling toolkit. In *Proc. ICSLP-02*.

Joseph Turian, Benjamin Wellington, and I. Dan Melamed. 2007. Scalable discriminative learning for natural language parsing and translation. In *Proc. NIPS 19*.

Erik Velldal and Stephan Oepen. 2005. Maximum entropy models for realization ranking. In *Proc. MT Summit X*.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proc. EMNLP-CoNLL-07*.

Ralph Weischedel and Ada Brunstein. 2005. BBN pronoun coreference and entity type corpus. Technical report, BBN.

Michael White and Rajakrishnan Rajkumar. 2008. A more precise analysis of punctuation for broad-coverage surface realization with CCG. In *Proc. of the Workshop on Grammar Engineering Across Frameworks (GEAF08)*.

Michael White, Rajakrishnan Rajkumar, and Scott Martin. 2007. Towards broad coverage surface realization with CCG. In *Proc. of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*.

Michael White. 2006. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language and Computation*, 4(1):39–75.

Huayan Zhong and Amanda Stent. 2009. Determining the position of adverbial phrases in English. In *Proc. NAACL HLT 2009 Short Papers*.