# LSTM Hypertagging

**Reid Fu**
Department of Computer Science
The Ohio State University
fu.349@buckeyemail.osu.edu

**Michael White**
Department of Linguistics
The Ohio State University
mwhite@ling.osu.edu

## Abstract

Hypertagging, or supertagging for surface realization, is the process of assigning lexical categories to nodes in an input semantic graph. Previous work has shown that hypertagging significantly increases realization speed and quality by reducing the search space of the realizer. Building on recent work using LSTMs to improve accuracy on supertagging for parsing, we develop an LSTM hypertagging method for OpenCCG, an open source NLP toolkit for CCG. Our results show significant improvements in both hypertagging accuracy and downstream realization performance.

## 1 Introduction

Hypertagging, or supertagging for surface realization, is the process of assigning lexical categories to nodes in a semantic graph as a step in grammar-based surface realization. It significantly increases realization speed and quality by reducing the search space of the realizer. Espinosa et al. (2008) built the original single-stage maximum entropy hypertagger for CCG in OpenCCG,[1] which obtained a single-best hypertagging accuracy of 93.6%. This hypertagger was later expanded into a two-stage model, which obtained a hypertagging accuracy of 95.1%.

Recent work has used neural networks to make significant improvements in supertagging for parsing (i.e., predicting lexical categories for a sequence of words), improving upon earlier work with maximum entropy supertagging (Curran et al., 2006). Lewis and Steedman (2014) used a feedforward neural network to obtain 91.3% category accuracy in the CCGBank (Hockenmaier and Steedman, 2007) development section (Section 00). Lewis et al. (2016) improved on this result using an LSTM, obtaining 94.9% category accuracy on the development section and 94.7% on the test section (Section 23) of the CCGBank.

Our work uses techniques from Lewis et al. (2016) to implement an improved hypertagger. To do so, we first linearize the input graph using a method adapted from Konstas et al.'s (2017) approach to generating from Abstract Meaning Representations (AMRs).[2] Unlike in their work though, we found that the input ordering method substantially impacted hypertagging accuracy, with an English-like ordering yielding substantial improvements over random ordering while substantially trailing oracle ordering.

We evaluated the LSTM hypertagger on both tagging accuracy and its downstream effect on realization performance. Our results show significant improvement over the original hypertagger on both. We obtained 96.47% on tagging accuracy (up from 95.1%) and an increase in realization BLEU scores from 0.8429 with the original hypertagger to 0.8683 with the neural hypertagger. As expected, the LSTM hypertagger yielded large gains in accuracy on the difficult cases of unseen predicates and predicates not seen with the gold tag in training, helping to achieve a 7.8% increase in sentences with grammatically complete derivations. A human evaluation confirmed that using the LSTM hypertagger yielded significant improvements in adequacy and fluency, especially in cases where the LSTM hypertagger was essential for obtaining a complete derivation.

This paper is structured as follows. Section 2 provides background on surface realization with CCG, the maximum entropy hypertagger and LSTM supertagging. Section 3 describes our fea-

[1] http://openccg.sf.net

[2] https://amr.isi.edu/



*Proceedings of The 11th International Natural Language Generation Conference*, pages 210–220,
Tilburg, The Netherlands, November 5-8, 2018. 

tures and model along with our approach to input linearization. The results and analysis appear in Section 4. Related work is discussed in Section 5, including where grammar-based realization stands in the current research landscape. Section 6 concludes.

## 2 Background

### 2.1 Surface Realization with OpenCCG

The OpenCCG realizer generates surface strings for input semantic dependency graphs (or logical forms) using a chart-based algorithm (White, 2006) for Combinatory Categorial Grammar (Steedman, 2000) together with a hypertagger for probabilistically assigning lexical categories to lexical predicates in the input, as noted above. An example input appears in Figure 1. In the figure, nodes correspond to discourse referents labeled with lexical predicates, and dependency relations between nodes encode argument structure; gold standard CCG lexical categories (i.e, what the hypertagger learns to predict) are also shown. Note that semantically empty function words such as infinitival-*to* are missing. Generally speaking, the semantic dependency graphs are more abstract than unordered dependency trees, but more detailed than AMRs. The grammar is extracted from a version of the CCGbank (Hockenmaier and Steedman, 2007) enhanced for realization, where the enhancements include: better analyses of punctuation (White and Rajkumar, 2008); less error prone handling of named entities (Rajkumar et al., 2009); re-inserting quotes into the CCGbank; and assignment of consistent semantic roles across diathesis alternations (Boxwell and White, 2008), using PropBank (Palmer et al., 2005).

As in other work with OpenCCG (e.g., Duan and White, 2014), we use OpenCCG's realization ranking model off the shelf in order to select preferred outputs from the chart; in particular, we use White & Rajkumar's (2009; 2012) averaged perceptron realization ranking model augmented with a large-scale 5-gram model based on the Gigaword corpus. The ranking model makes choices addressing all three interrelated sub-tasks traditionally considered part of the surface realization task in natural language generation research (Reiter and Dale, 2000): inflecting lemmas with grammatical word forms, inserting function words and linearizing the words in a grammatical and natural order.

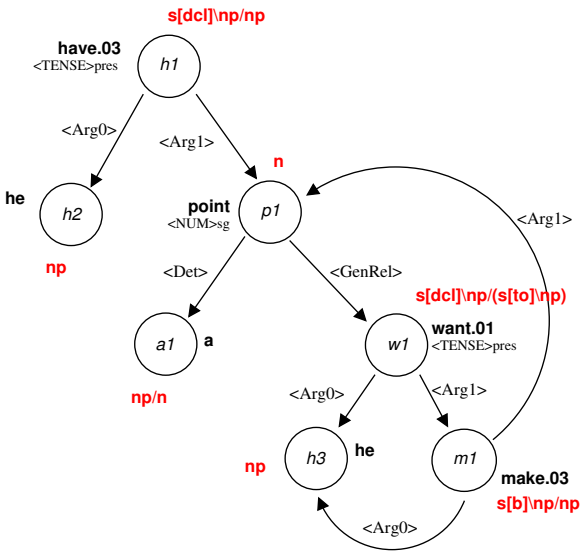


Figure 1: Example OpenCCG semantic dependency input for *he has a point he wants to make*, with gold standard lexical categories for each node

Notably, to improve word ordering decisions, White & Rajkumar (2012) demonstrated that incorporating a feature into the ranker inspired by Gibson's (2000) **dependency locality theory** can deliver statistically significant improvements in automatic evaluation scores, better match the distributional characteristics of sentence orderings, and significantly reduce the number of serious ordering errors. With function words, Rajkumar and White (2011) showed that they could improve upon the earlier model's predictions for when to employ *that*-complementizers using features inspired by Jaeger's (2010) work on using the principle of **uniform information density**, which holds that human language use tends to keep information density relatively constant in order to optimize communicative efficiency.

Finally, to reduce the number of subject-verb agreement errors, Rajkumar and White (2010) extended the earlier model with features enabling it to make correct verb form choices in sentences involving complex coordinate constructions and with expressions such as *a lot of* where the correct choice is not determined solely by the head noun. They also improved animacy agreement with relativizers, reducing the number of errors where *that* or *which* was chosen to modify an animate noun rather than *who* or *whom* (and vice-versa), while also allowing both choices where corpus evidence was mixed.

### 2.2 Original Hypertagger

The original MaxEnt hypertagger uses three general types of features from logical forms: lexical features, graph structural features, and node attribute features. Lexical features are the words associated with the logical form nodes. Graph structural features are those pertaining to word dependency relations, and include the number of children (and argument children) of each node as well as the names of dependency relations. Node attribute features are those pertaining to semantic or syntactic features of words, and include tense and number.

The published single-stage MaxEnt hypertagger has an intermediate stage in which it predicts POS tags. These POS tags are then included in the feature set used to predict supertags. The two-stage hypertagger stacks on an additional stage in which predicted supertags in the local graph context are used as features for making final predictions.

The realizer uses the hypertagger in an iterative $\beta$-best algorithm in which the realizer repeatedly queries the hypertagger for $\beta$-best tags. The hypertagger has a list of $\beta$-values sorted from most restrictive to least. It first returns a $\beta$-best list of supertags for the most restrictive beta. If the realizer fails to find a complete realization with the returned supertags, it asks for the supertags associated with the next most restrictive beta, and so on until either a complete realization is found, time runs out, or there are no more betas in the list.

### 2.3 LSTM Supertagger

The model from Lewis et al. (2016) is summarized in Figure 2. Start and end tokens are added to each sentence. Each word in each sentence (including each start and end token) is mapped to a 50-element word embedding. Word embeddings are initialized using pre-trained word embeddings from Turian et al. (2010). Embeddings for features of the word are concatenated to the word's 50-element embedding. The concatenated embeddings are used as input to a stacked, bi-directional LSTM with depth 2. Lewis et al. used 1-4 character prefixes and suffixes as their features.

The LSTM cell used is a variant that has coupled input and forget gates. If a cell is at position t, we refer to the cell at t-1 as the *previous* cell and the cell at t+1 as the *next* cell. Each cell takes cell state $c_{t-1}$ and hidden state $h_{t-1}$ from the previous cell, and $x_t$ from the previous layer, passing $c_t$ and $h_t$ to the next cell, and $h_t$ to the next layer. $c_t$ and $h_t$ are calculated as follows, with * indicating component-wise product:

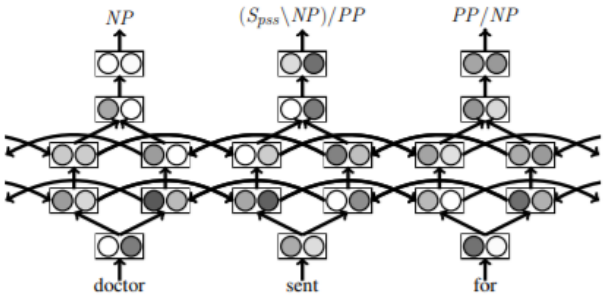


Figure 2: LSTM model used in Lewis et al. (2016) and our hypertagger (image from Lewis et al.). Concatenated embedding representations of each word are passed to stacked, bi-directional LSTM that reads sentence in both directions. Outputs of the directional LSTM's are combined. Applying a softmax over the combined outputs yields the probability distributions over supertags.

$$i_t = \sigma(W_i[c_{t-1}, h_{t-1}, x_t] + b_i) \quad (1)$$
$$\tilde{c}_t = tanh(W_c[h_{t-1}, x_t] + b_{\tilde{c}}) \quad (2)$$
$$o_t = \sigma(W_o[\tilde{c}_t, h_{t-1}, x_t] + b_o) \quad (3)$$
$$c_t = i_t * \tilde{c}_t + (1 - i_t)c_{t-1} \quad (4)$$
$$h_t = o_t * tanh(c_t) \quad (5)$$

The model is trained using stochastic gradient descent with minibatch size of 1, learning rate of 0.01, and momentum of 0.7. The input layer has a dropout probability of 0.5. Order of sentences is shuffled after every epoch. We retained these settings for our hypertagger experiments.[3]

The outputs of the LSTM are passed to another hidden layer, a bias is added, and a rectified linear function is applied. The result is a set of logits that give the probability distributions over supertags for each word when passed to a softmax.

## 3 Approach

### 3.1 Features

We use a subset of the features from the original MaxEnt hypertagger (shown below). If a node has fewer than five parent or child relations, the val-

[3] As an anonymous reviewer points out, the training would be faster with a larger minibatch size, and inference would be faster using highly optimized implementations of standard LSTM cells.

ues of the missing relations were set to the empty string. This kept the size of the input constant.

- **Lexical Features**: Lemmatized words associated with elementary predication nodes
- **Node Attribute Features**: Named entity category, determiner, mood, number, particle, tense
- **Graph Structural Features**: Number of children, number of argument children, names of 5 parent relations, names of 5 child relations

Our hypertagger does not include a stage for predicting POS tags or for using the predicted supertags as features in a stacked model. The recurrent nature of the bi-LSTMs means that the node-level representations informing supertag prediction can be propagated to nearby nodes in the model.

Features were extracted from logical forms corresponding to Section 00 and Sections 02 to 21. Section 00 was used as the development set, and has 1883 sentences and 36,247 nodes. Sections 02 to 21 were used as the training set, and have a total of 35,765 sentences and 680,705 nodes.

### 3.2 Input Linearization

Each logical form was converted to a sequence of nodes, with each node containing all features discussed in the last sub-section. We refer to this conversion process as input linearization. Early experiments showed that the order of nodes in the linearized sequence made a significant difference in accuracy. Oracle linearization, in which nodes are ordered according to the order of words in the original sentence, outperformed random linearization by about 5%. Depth-first search linearization, in which the parent node came before all the child nodes and the child nodes were in random order relative to each other, improved over random linearization by about 2%. These experiments showed that more English-like input linearizations led to higher performance.

To approximate English ordering, we developed a heuristic of how nodes and their children should be ordered. The heuristic was applied recursively from the root node(s) of the logical form down to the leaves. The child ordering is shown below. The heuristic performed around 3% higher than random linearization, and around 2% lower than oracle linearization.

1. Determiner (Det) child
2. Possessor (GenOwn) child
3. First argument (Arg0)
4. One or two word modifiers
5. Parent node
6. Remaining arguments in order of argument number (Arg1, ..., Arg5)
7. Remaining children sorted ascending by subtree size. If two child subtrees have the same size, the one with a lesser sum of predicate name lengths comes before the other.

Konstas et al. (2017) were able to improve their model by adding parentheses and relation names into their AMR linearization sequence. We experimented with both. Adding parentheses around subsequences corresponding to subtrees of size five or more consistently resulted in an improvement of 0.2%. Adding relation names to the linearization sequence resulted in either no improvement or a slight decrease in accuracy.

### 3.3 Model

We adapted the bi-LSTM model from Lewis et al. (2016) to use our features and list of possible supertags. Each feature value was mapped to a feature embedding. Embeddings for lexical features were initialized using pre-trained word embeddings. Embeddings for other features were randomly initialized 8-element vectors. The embeddings for other features were concatenated onto the embeddings for lexical features, and the concatenated embeddings were used as input to the bi-LSTM.

We experimented with two sets of pre-trained word embeddings: those that were used in Lewis et al., which were of size 50, and a custom set, which were of size 100. Early experiments showed that performance was similar for both sets. Later experiments exclusively used the set used in Lewis et al.

Sentences with linearized sequences longer than 180 tokens (nodes and parentheses) were filtered out of the training and development sets. This filter removed one sentence from the development set, leaving it with 1882 sentences and 36,103 nodes.

Our list of possible supertags was originally of size 1210. We experimented with filtering the tags: tags that occurred fewer than 5 times in the training set were not listed as possible supertags. Instead, when a rare tag is encountered, it was replaced with the unknown tag and the associated word was counted as wrong in the accuracy. We found that filtering tags boosted performance. Our filtered list of possible supertags is of size 528. There were 1543 words in the training set and 88 words in the development set with gold tags filtered out.

The model was trained on the training set, and evaluated on the development set every 2 minutes during training. Training was run until either 7 hours had elapsed, or accuracy had not improved for 30 evaluations. Typically, training ran between 30 and 36 epochs.

### 3.4 Realization

The realizer is implemented in Java, while the LSTM model is implemented in Python. To establish communication between the realizer and the hypertagger, we used a server thread to run the LSTM model. The realizer got supertags from the hypertagger by running a client thread, then parsing the results returned by it. The LSTM hypertagger was otherwise used in the same way as the original MaxEnt hypertagger, with the realizer calling the hypertagger in an iterative $\beta$-best algorithm.

## 4 Results and Discussion

The LSTM hypertagger was tested against the original, baseline hypertagger on both tagging accuracy and downstream effect on realizer performance. Tagging accuracy was evaluated on Section 00, and realizer performance was evaluated on Sections 00 and 23. The lists of $\beta$-values for the LSTM hypertagger and baseline two-stage MaxEnt hypertagger were adjusted so that both would on average return about the same number of tags per word as the published single-stage MaxEnt hypertagger at each corresponding $\beta$-level. Section 00 was used to tune the list of $\beta$-values for the LSTM hypertagger.

### 4.1 Hypertagging Accuracy

The comparison in hypertagging accuracy is summarized in Table 1. In comparison to previously published results, the LSTM hypertagger achieves a nearly 3% absolute increase in single-best tagging accuracy, and achieves 99% accuracy at a multitagging level of only 1.2 tags per predicate, in comparison to 3.2 tags per predicate previously. Single-best tagging accuracy for the LSTM hypertagger had a mean of 96.384% and a variance of 0.031% (statistics taken over the six most recent runs). Figures in Table 1 are for one of the higher-performing runs.

We were interested in how the hypertaggers would perform on hard cases such as predicates that appeared in the development set but not in training, as well as predicates that appeared in training, but not with the correct tag in the development set. On these hard cases, our model obtained accuracies of 98.69% and 80.13%, respectively, while the two-stage MaxEnt hypertagger obtained accuracies of 94.58% and 69.96%, respectively. These improvements represent very large respective reductions in error of 76% and 34%. These results are summarized in Table 2.

The difference in accuracy between the LSTM hypertagger with English-like input linearization and the LSTM hypertagger with oracle input linearization is largely attributed to imperfections in the English-like linearization. There were many cases in which the English-like input linearization reversed the order between two phrases (e.g. *take these events place ago years, you but have to recognize*) or between two words in a phrase (e.g. *three times than more*, *among of us those*). In many of these cases, the hypertagger still predicted the correct tags despite the word order switches. For the words that the hypertagger did not tag correctly, when the word order switch was still grammatical, the assigned supertag was often similar enough to the gold tag to not significantly impact realization. By contrast, when the word order switch was ungrammatical, the assigned supertag was sometimes very different from the gold tag, which impacted realization negatively.

### 4.2 Realization Performance

The comparison in realization performance is summarized in Table 3. Using the LSTM hypertagger, the realizer obtained complete derivations for more than 6% more realizations (more than 100 more logical forms) in both the development and test sections. The increase in the number of complete derivations helped achieve a more than 2.5% absolute increase in BLEU scores for both

| $\frac{Tags}{Pred}$ | LSTM | | MaxEnt1 | |
|---|---|---|---|---|
| | $\beta$ | Accuracy | $\beta$ | Accuracy |
| 1 | 1.0 | 96.5 | 1.0 | 93.6 |
| 1.1 | 0.13 | 98.5 | 0.16 | 95.8 |
| 1.2 | 0.04 | 99.0 | 0.05 | 96.6 |
| 1.5 | 7e-3 | 99.4 | 5.8e-3 | 97.9 |
| 1.8 | 2.8e-3 | 99.5 | 1.75e-3 | 98.4 |
| 2.2 | 1.27e-3 | 99.6 | 6.25e-4 | 98.7 |
| 3.2 | 3.65e-4 | 99.7 | 1.25e-4 | 99.0 |
| 3.9 | 2e-4 | 99.7 | 5.8e-5 | 99.1 |

Table 1: Comparison of tagging accuracies between LSTM hypertagger and published single-stage MaxEnt hypertagger on the development section (Section 00) of CCGBank. Results (in percentages) are for per-predicate tagging accuracies.

| | LSTM | MaxEnt2 |
|---|---|---|
| Unseen Predicates | 98.69 | 94.58 |
| Unseen Predicate-Tag Pairs | 80.13 | 69.96 |

Table 2: Comparison of tagging accuracies between LSTM hypertagger and unpublished two-stage MaxEnt hypertagger on hard cases in the development set, namely predicates that are not seen in training and predicates that are seen in training but not with correct supertag.

sections.

Most of the incomplete or suboptimal realizations made with the LSTM hypertagger occurred due to one or more of the following reasons. Some sentences had several words in which the correct tag was in the list of $\beta$-best tags for the least restrictive beta, but had a low probability. This caused the realizer to time out due to the large size of the search space. Others had a single word in which the correct tag was not in the $\beta$-best tag list for the least restrictive beta. Instead there were many tags in the list that were similar to the gold tag, but that did not allow a complete realization. For example, the word *recognize* in the sentence *but you have to recognize that these events took place 35 years ago* had a gold tag of s[b]\np/s[em] (a bare verb subcategorizing for an embedded clause), but had a single-best assigned tag of s[b]\np (a bare intransitive verb), making it impossible to derive a constituent containing the embedded clause. These errors were likely caused by imperfections in the English-like input linearization. For the example above, the input linearization placed *recognize* at the end of the sequence, with no subsequent predicates to help predict the gold tag. Another reason for incomplete or suboptimal realizations is that the hypertagger can't predict certain low-frequency tags, so some words will not have the correct tag in their $\beta$-best tag list regardless of the $\beta$ value.

### 4.3 Human Evaluation

We also did a targeted human evaluation to determine whether the improvements in hypertagging accuracy generally led to noticeable improvements in realization quality, especially where it enabled a complete realization to be found.[4] Our procedure was as follows. We randomly chose 100 sentences from the devset where the realizations differed between using the LSTM hypertagger and the baseline two-stage MaxEnt hypertagger. 50 of the sentences were ones for which one system got a complete realization but the other did not—where we expected to find substantial differences—while the other 50 were ones for which either both systems got a complete realization or both did not get a complete realization. We generated a spreadsheet with the following columns: Reference sentence, Realization A, Realization B, Adequacy, and Fluency. Which system was A for each sentence was randomized,

[4]The complete set of examples and the evaluation script are available in the following supplement to the paper: https://osf.io/a4czs/?view_only=b7a0a49046a0408bb844ff7ea63c4e08

| Section | LSTM | | | MaxEnt2 | | |
|---|---|---|---|---|---|---|
| | BLEU | Complete | Exact | BLEU | Complete | Exact |
| 00 | 0.8783 | 90.38 | 49.36 | 0.8458 | 83.86 | 45.62 |
| 23 | 0.8683 | 87.75 | 48.35 | 0.8429 | 81.69 | 44.16 |

Table 3: Comparison in realization performance between systems using the LSTM and two-stage MaxEnt hypertaggers. The 'Complete' column indicates the percentage of logical forms realized with a complete (non-fragmentary) derivation, while the 'Exact' column indicates the percentage of realizations that exactly matched the reference sentence.

as was the order of sentences. A separate key file kept track of which system was A for each sentence, and which sentence belonged to which of the above sets of 50. Two linguists who had no familiarity with the research evaluated the realizations, marking in the Adequacy and Fluency columns whether they believed A or B was better in adequacy and fluency, respectively, or whether they were equally good for one or both measures. Raw agreement was relatively high, with the two judges agreeing on adequacy 70% of the time and fluency 73% of the time. However, nearly all the disagreements involved cases where only one judge found the pair of realizations to be the same on adequacy or fluency; on the subset of items where neither judge found the pair to be equal, agreement was 96% for adequacy and 95% for fluency.

The results of the human evaluations are summarized in Table 4. For the sentences where only one system produced a complete realization (Set 1), the LSTM hypertagger system outperformed the baseline one most of the time on both adequacy and fluency. For the other sentences (Set 2), the two systems were mostly tied on adequacy and fluency, but when the realizations were of distinct quality, the LSTM hypertagger system usually outperformed the original one. All differences in the counts of Better/Worse judgments were highly significant ($p < 0.001$, sign test).

Examples of the changes yielded by the LSTM hypertagger appear in Table 5, where the first two examples improve both adequacy and fluency, the next example makes adequacy and fluency worse, and the final one leaves adequacy and fluency the same. With wsj_0080.21, it seems that the two-stage MaxEnt system failed to match the subject with the verb, yielding a realization where *respond* doesn't have a subject and *them* is not clearly linked with its antecedent. With wsj_0004.8, the LSTM system switched *yields* and *nevertheless*, making a realization that's different from the original sentence, but still grammatical. The two-stage MaxEnt system seemed to have trouble combining the words *yields*, *nevertheless*, and *may*, making a realization that gives an awkward order for these words and splits the sentence with the phrase *said Brenda Malizia Negus* at an awkward place; moreover, with *may* appearing initially, the sentence can be read as a wish rather than a declarative statement. With wsj_0097.19, the LSTM system incorrectly inverts the main subject and verb, and makes several other word order mistakes. Finally, wsj_0037.9 is an example where leaving out the complementizer *that* or the contraction does not substantially affect adequacy or fluency (though the reference sentence arguably makes the best choices here). More generally, while the choice whether to include a *that*-complementizer occasionally made a crucial difference, they were a frequent source of insubstantial differences, along with contractions and adverbial placement. There were also cases where both realizations made distinct but important mistakes that yielded equally bad realizations.

## 5 Related Work

Hypertagging can potentially benefit other grammar-based methods using lexicalized grammars, e.g. using HPSG (Velldal and Oepen, 2005; Carroll and Oepen, 2005; Nakanishi et al., 2005) or TAG (Gardent and Perez-Beltrachini, 2017). Much recent work in NLG (Wen et al., 2015; Dušek and Jurcicek, 2016; Mei et al., 2016; Kiddon et al., 2016; Konstas et al., 2017; Wiseman et al., 2017) has made use of neural sequence-to-sequence methods for generation rather than grammar-based methods. The learning flexibility of neural methods make it possible to develop very knowledge lean systems, but they continue to suffer from a tendency to hallucinate content and have not been used with texts exhibiting

| Set | Adequacy | | | Fluency | | |
|---|---|---|---|---|---|---|
| | Better | Same | Worse | Better | Same | Worse |
| 1 (±complete) | 84 | 12 | 4 | 88 | 7 | 5 |
| 2 (=complete) | 31 | 62 | 7 | 37 | 52 | 11 |

Table 4: Results (counts of judgments) of human evaluations of realizations, which indicate how often the new system produced better, same, and worse realizations for the given aspect. Set 1 is the set of sentences in which one system had a complete realization while the other did not. Set 2 is the set of sentences in which either both systems had complete realizations or both did not.

| | |
|---|---|
| wsj_0080.21 | it was n't clear **how NL and Mr. Simmons would respond** if Georgia Gulf spurns them again . |
| LSTM | [same] |
| MAXENT2 | it was n't clear **how to would respond** if Georgia Gulf spurns them again **NL and Mr. Simmons** . |
| wsj_0004.8 | nevertheless , said Brenda Malizia Negus , editor of Money Fund Report , yields may blip up again before they blip down because of recent rises in short-term interest rates . |
| LSTM | **yields nevertheless may** blip up again before they blip down because of recent rises in short-term interest rates , said Brenda Malizia Negus , editor of Money Fund Report . |
| MAXENT2 | **may nevertheless yields** , said Brenda Malizia Negus , editor of Money Fund Report , again blip up before they blip down because of recent rises in short-term interest rates . |
| wsj_0097.19 | -lrb- Morgan Stanley last week joined a growing list of U.S. securities firms that have stopped doing index arbitrage for their own accounts . -rrb- |
| LSTM | **last week joined Morgan Stanley** . U.S. securities firms that growing a list of has stopped doing index arbitrage for their own accounts |
| MAXENT2 | -lrb- **Morgan Stanley last week joined** a growing list of U.S. securities firms that have stopped doing index arbitrage for their own accounts . |
| wsj_0037.9 | if " a Wild Sheep Chase " carries an implicit message for international relations , it 's that the Japanese are more like us than most of us think . |
| LSTM | if " a Wild Sheep Chase " carries an implicit message for international relations , **it 's the Japanese** are more like us than most of us think . |
| MAXENT2 | if " a Wild Sheep Chase " carries an implicit message for international relations , **it is that the Japanese** are more like us than most of us think . |

Table 5: Examples of devset sentences where the LSTM hypertagger improved adequacy/fluency (top), made it worse (middle) or left it the same (bottom).

the full complexity of genres such as news text. Approaches based on dependency grammar (Guo et al., 2008; Bohnet et al., 2010, 2011; Zhang and Clark, 2015; Liu et al., 2015; Puduppully et al., 2016, 2017; King and White, 2018) are also simpler than constraint-based grammar approaches, making them more robust to unexpected inputs and easier to deploy across languages, but it is difficult to determine whether they can fully substitute for precise grammars because these approaches have not used compatible inputs.

Although approaches using constraint-based grammars are clearly more difficult to implement and deploy, there is some evidence that they are beneficial for parsing, while for realization the question remains largely open. For parsing, Buys and Blunsom (2017) have recently shown that even though their incremental neural semantic graph parser substantially outperforms standard attentional sequence-to-sequence models, it still lags 4-6% behind an HPSG parser using a simple log-linear model (Toutanova et al., 2005) on a variety of parsing accuracy measures on DeepBank (Flickinger et al., 2012), a conversion of the Penn Treebank to Minimal Recursion Semantics (Copestake et al., 2005, MRS). The MRS representations in DeepBank are qualitatively similar to the OpenCCG semantic graphs used in this work, which are again qualitatively similar to the deep representations used in the First Surface Realization Shared Task (Belz et al., 2010, 2011). On the deep shared task representations, Bohnet et al. (2011) achieved a BLEU score of 0.7943, which Puduppully et al. (2017) later improved upon with a score of 0.8077. These scores are substantially lower than our BLEU score of 0.8683 reported here, though since the inputs are not exactly the same, the BLEU scores are of course not directly

comparable.

Given the flexibility of neural methods, it would be interesting in future work to examine how well neural sequence-to-sequence generation methods would fare in a direct, head-to-head comparison using the kinds of detailed, deep inputs used with HPSG and CCG. To the extent that neural approaches continue to hallucinate content and fail to observe constraints and preferences implemented by grammar-based approaches in such a comparison, it would also be worthwhile to investigate additional ways of combining neural and grammar-based methods.

## 6 Conclusion

We have implemented a new LSTM hypertagger that significantly outperforms the existing OpenCCG hypertagger on both tagging accuracy and its downstream effect on realization performance. Since we have observed that the order in which input nodes are linearized substantially affects tagging accuracy, in future work we would like to explore whether graph-based neural tagging methods could yield further improvements in performance. Another direction of interest is exploring ways of incorporating hypertagging into architectures that synergistically combine grammar-based and neural generation methods.

## Acknowledgments

We thank the OSU Clippers Group, Alan Ritter and the anonymous reviewers for helpful comments and discussion, and Sarah Ewing and Amad Hussain for their assistance with the evaluation. This work was supported in part by NSF grant IIS-1319318.

## References

Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 217–226. Association for Computational Linguistics.

Anja Belz, Mike White, Josef van Genabith, Deirdre Hogan, and Amanda Stent. 2010. Finding common ground: Towards a surface realisation shared task. In *Proceedings of INLG-10, Generation Challenges*, pages 267–272.

Bernd Bohnet, Simon Mille, Benoît Favre, and Leo Wanner. 2011. Stumaba : From deep representation to surface. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 232–235. Association for Computational Linguistics.

Bernd Bohnet, Leo Wanner, Simon Mill, and Alicia Burga. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 98–106. Coling 2010 Organizing Committee.

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

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226. Association for Computational Linguistics.

John Carroll and Stefan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proc. IJCNLP-05*.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A Sag. 2005. Minimal recursion semantics: An introduction. *Research on language and computation*, 3(2-3):281–332.

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

Manjuan Duan and Michael White. 2014. That's Not What I Meant! Using Parsers to Avoid Structural Ambiguities in Generated Text. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 413–423, Baltimore, Maryland. Association for Computational Linguistics.

Ondřej Dušek and Filip Jurcicek. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 45–51. Association for Computational Linguistics.

Dominic Espinosa, Michael White, and Dennis Mehay. 2008. Hypertagging: Supertagging for surface realization with CCG. In *Proceedings of ACL-08: HLT*, pages 183–191, Columbus, Ohio. Association for Computational Linguistics.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Claire Gardent and Laura Perez-Beltrachini. 2017. A statistical, grammar-based approach to microplanning. *Computational Linguistics*, 43(1).

Edward Gibson. 2000. Dependency locality theory: A distance-based theory of linguistic complexity. In Alec Marantz, Yasushi Miyashita, and Wayne O'Neil, editors, *Image, Language, brain: Papers from the First Mind Articulation Project Symposium*. MIT Press, Cambridge, MA.

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. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

T. Florian Jaeger. 2010. Redundancy and reduction: Speakers manage information density. *Cognitive Psychology*, 61(1):23–62.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339. Association for Computational Linguistics.

David King and Michael White. 2018. The osu realizer for srst '18: Neural sequence-to-sequence inflection and incremental locality-based linearization. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 39–48. Association for Computational Linguistics.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 146–157. Association for Computational Linguistics.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. Lstm ccg parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014. Improved CCG parsing with semi-supervised supertagging. *TACL*.

Yijia Liu, Yue Zhang, Wanxiang Che, and Bing Qin. 2015. Transition-based syntactic linearization. In *Proceedings of NAACL*, Denver, Colorado, USA.

Hongyuan Mei, Mohit Bansal, and R. Matthew Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 720–730. Association for Computational Linguistics.

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

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

Ratish Puduppully, Yue Zhang, and Manish Shrivastava. 2016. Transition-based syntactic linearization with lookahead features. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Li nguistics: Human Language Technologies*, pages 488–493, San Diego, California. Association for Computational Linguistics.

Ratish Puduppully, Yue Zhang, and Manish Shrivastava. 2017. Transition-based deep input linearization. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 643–654. Association for Computational Linguistics.

Rajakrishnan Rajkumar and Michael White. 2010. Designing agreement features for realization ranking. In *Proc. Coling 2010: Posters*, pages 1032–1040, Beijing, China.

Rajakrishnan Rajkumar and Michael White. 2011. Linguistically motivated complementizer choice in surface realization. In *Proceedings of the UCNLG+Eval: Language Generation and Evaluation Workshop*, pages 39–44, Edinburgh, Scotland. Association for Computational Linguistics.

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

Ehud Reiter and Robert Dale. 2000. *Building Natural-Language Generation Systems*. Cambridge University Press.

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

Kristina Toutanova, Christopher D Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic hpsg parse disambiguation using the redwoods corpus. *Research on Language and Computation*, 3(1):83–105.

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

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721. Association for Computational Linguistics.

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

Michael White and Rajakrishnan Rajkumar. 2008. A more precise analysis of punctuation for broad-coverage surface realization with CCG. In *Coling 2008: Proceedings of the workshop on Grammar Engineering Across Frameworks*, pages 17–24.

Michael White and Rajakrishnan Rajkumar. 2009. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 410–419, Singapore. Association for Computational Linguistics.

Michael White and Rajakrishnan Rajkumar. 2012. Minimal dependency length in realization ranking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 244–255, Jeju Island, Korea. Association for Computational Linguistics.

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2015. Syntax-based word ordering using learning-guided search. *Computational Linguistics*, 41(3).