# LSTM CCG Parsing

**Mike Lewis** **Kenton Lee** **Luke Zettlemoyer**
Computer Science & Engineering
University of Washington
Seattle, WA 98195
{mlewis,kentonl,lsz}@cs.washington.edu

## Abstract

We demonstrate that a state-of-the-art parser can be built using only a lexical tagging model and a deterministic grammar, with no explicit model of bi-lexical dependencies. Instead, all dependencies are implicitly encoded in an LSTM supertagger that assigns CCG lexical categories. The parser significantly outperforms all previously published CCG results, supports efficient and optimal A* decoding, and benefits substantially from semi-supervised tri-training. We give a detailed analysis, demonstrating that the parser can recover long-range dependencies with high accuracy and that the semi-supervised learning enables significant accuracy gains. By running the LSTM on a GPU, we are able to parse over 2600 sentences per second while improving state-of-the-art accuracy by 1.1 F1 in domain and up to 4.5 F1 out of domain.

## 1 Introduction

Combinatory Categorial Grammar (CCG) is a strongly lexicalized formalism—the vast majority of attachment decisions during parsing are specified by the selection of lexical entries for words (see Figure 1 for examples). State-of-the-art parsers typically include a supertagging model, to select possible lexical categories, and a bi-lexical dependency model, to resolve the remaining parse attachment ambiguities. In this paper, we introduce a long short-term memory (LSTM) CCG parsing model that has no explicit model of bi-lexical dependencies, but instead relies on a bi-directional recurrent neural network (RNN) supertagger to capture all long distance dependencies. This approach has a number of advantages: it is conceptually simple, allows for the reuse of existing optimal and efficient parsing algorithms, benefits significantly from semi-supervised learning, and is highly accurate both in and out of domain. The parser is publicly released.[1]

Neural networks have shown strong performance in a range of NLP tasks; however they can break the dynamic programs for structured prediction problems, such as parsing, when vector embeddings are recursively computed for subparts of the output. Existing neural net parsers either (1) use greedy inference techniques including shift-reduce parsing (Henderson et al., 2013; Chen and Manning, 2014; Weiss et al., 2015; Dyer et al., 2015), constituency parse re-ranking (Socher et al., 2013), and string-to-string transduction (Vinyals et al., 2015), or (2) avoid recursive computations entirely (Durrett and Klein, 2015). Our approach gives a simple alternative: we only train a model for tagging decisions, where we can easily use recurrent architectures such as LSTMs (Hochreiter and Schmidhuber, 1997), and rely on the highly lexicalized nature of the CCG grammar to allow this tagger to specify nearly every aspect of the complete parse.

Our LSTM supertagger is bi-directional and includes a softmax potential over tags for each word in the sentence. During training, we jointly optimize all LSTM parameters, including the word embeddings, to maximize the conditional likelihood of supertag sequences. For inference, we use a recently introduced A* CCG parsing algorithm (Lewis and Steedman, 2014a), which efficiently searches for the

[1] http://github.com/mikelewis0/EasySRL



*Proceedings of NAACL-HLT 2016*, pages 221–231,
San Diego, California, June 12-17, 2016. 

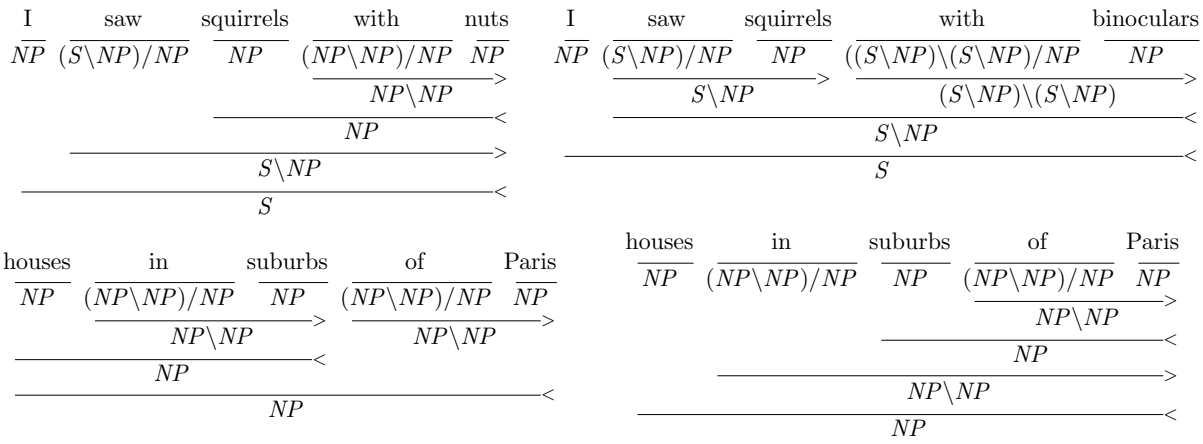


Figure 1: Four examples of prepositional phrase attachment in CCG. In the upper two parses, the attachment decision is determined by the choice of supertags. In the lower parses, the attachment is ambiguous given the supertags. In such cases, our parser deterministically attaches low (i.e. preferring the lower-right parse).

highest probability sequence of tags that combine to produce a complete parse tree. Whenever there is parsing ambiguity not specified by the supertags, the model attaches low (see Figure 1).

This approach is not only conceptually simple but also highly effective, as we demonstrate with extensive experiments. Because the A* algorithm is extremely efficient and the LSTMs can be run in parallel on GPUs, the end-to-end parser can process over 2600 sentences per second. This is more than three times the speed of any publicly available parser for any formalism. Apart from Hall et al. (2014), we are not aware of efficient algorithms for running other state-of-art-parsers on GPUs. The LSTM parameters also benefit from semi-supervised training, which we demonstrate by employing a recently introduced tri-training scheme (Weiss et al., 2015). Finally, the recurrent nature of the LSTM allows for effective modelling of long distance dependencies, as we show empirically. Our approach significantly advances the state-of-the-art on benchmark datasets—improving accuracy by 1.1 F1 in domain and up to 4.5 F1 out of domain.

## 2 Background

**Combinatory Categorial Grammar (CCG)** Compared to a phrase-structure grammar, CCG contains a much smaller set of binary rules (we use 11), but a much larger set of lexical tags (we use 425). The binary rules are conjectured to be language-universal, and most language-specific information is lexicalized (Steedman, 2000). The large tag set means that most (but not all) attachment decisions are determined by tagging decisions. Figure 1 shows how a prepositional phrase attachment decision can be encoded in the choice of tags.

The process of assigning CCG categories to words is called *supertagging*. All supertaggers used in practice are probabilistic, providing a distribution over possible tags for each word. Parsing models either use these scores directly (Auli and Lopez, 2011b), or as a form of beam search (Clark and Curran, 2007), typically in conjunction with models of the dependencies or derivation.

**Supertag-Factored A$^*$ CCG Parsing** Lewis and Steedman (2014a) introduced supertag-factored CCG parsers, in which the score for a parse is simply the sum of the scores of its supertags. The parser takes in a distribution over supertags for each word, and outputs the highest scoring parse—subject to the hard constraint that the parse only uses standard CCG combinators (resolving any remaining ambiguity by attaching low). One advantage of the supertag-factored model is that it allows a simple A$^*$ parsing algorithm, which provably finds the highest scoring supertag sequence that can be combined to construct a complete parse.

In A$^*$ parsing, partial parses $y_{i,j}$ of span $i \dots j$ are maintained in a sorted agenda and added to the chart in order of their cost, which is the sum of their Viterbi inside score $g(y_{i,j})$ and an upper bound on their Viterbi outside score $h(y_{i,j})$. When $y_{i,j}$ is

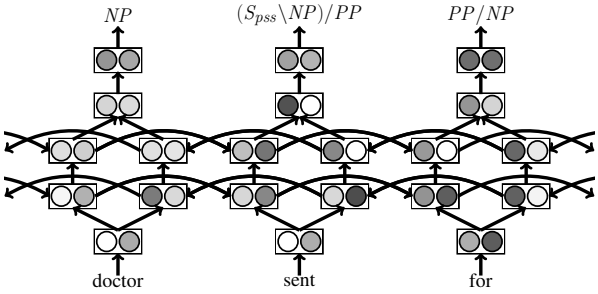


Figure 2: Visualization of our supertagging model, based on stacked bi-directional LSTMs. Each word is fed into stacked LSTMs reading the sentence in each direction, the outputs of the LSTMs are combined, and there is a final softmax over categories.

added to the chart, the agenda is updated with any new partial parses that can be created by combining $y_{i,j}$ with existing chart items (Algorithm 1). If $h$ is a monotonic upper bound on the outside score, the first chart entry for a span with a given category is guaranteed to be optimal—all other possible completions of the competing partial parses provably have lower scores, due to the outside score bounds. There is no guarantee this certificate of optimality is achieved efficiently for parses of the whole sentence, and in the worst case the algorithm could fill the entire parse chart. However, as we will see later, A* parsing is very efficient in practice for the models we present in this paper.

In the supertag-factored model, $g$ and $h$ are computed as follows, where $g(y_k)$ is the score for word $k$ having tag $y_k$.

$$g(y_{i,j}) = \sum_{k=i}^{j} g(y_k) \quad (1)$$

$$h(y_{i,j}) = \sum_{k=1}^{i-1} \max_{y_k} g(y_k) + \sum_{k=j+1}^{N} \max_{y_k} g(y_k) \quad (2)$$

where Eq. 1 follows from the definition of the supertag factored model and Eq. 2 combines this definition with the fact that the max score over all supertags for a word is an upperbound on the score for the actual supertag used in the best parse.

## 3 LSTM CCG Supertagging Model

Supertagging is *almost parsing* (Bangalore and Joshi, 1999)—consequently the task is very challenging, with hundreds of tags, and the correct assignment often depending on long-range dependencies. For example, in *The doctor sent for the patient arrived*, the category for *sent* depends on the final word. Recent work has made dramatic progress, using feed-forward neural networks (Lewis and Steedman, 2014b) and RNNs (Xu et al., 2015).

We make several extensions to previous work on supertagging. Firstly, we use bi-directional models, to capture both previous and subsequent sentence context into supertagging decisions. Secondly, we use LSTMs, rather than RNNs. Many tagging decisions rely on long-range context, and RNNs typically struggle to account for sequences of longer than a few words (Hochreiter and Schmidhuber, 1997). Finally, we use a deep architecture, to allow the modelling of complex interactions in the context.

Our supertagging model is summarized in Figure 2. Each word is mapped to an embedding vector. This vector is a concatenation of an embedding for the word (lower-cased), and embeddings for features of the word (we use 1 to 4 character prefixes and suffixes). The embedding vector is used as input to two stacked LSTMs (with depth 2), one processing the sentence left-to-right, and the other right-to-left.

**Algorithm 1** Agenda-based parsing algorithm

**Definitions** $x_{1...N}$ is the input words, and $y$ variables denote scored partial parses. TAG($x_{1...N}$) returns a set of scored pre-terminals for every word. ADD(C, y) adds partial parse $y$ to chart $C$. RULES(C, y) returns the set of scored partial parses that can be created by combining $y$ with existing entries in $C$. The agenda $A$ is ordered as described in Section 2.

**function** PARSE($x_{1...N}$)
  $A \leftarrow \emptyset$ ▷ Empty agenda $A$
  **for** $y \in$ TAG($x_{1...N}$) **do**
    PUSH($A, y$)
  $C \leftarrow \emptyset$ ▷ Empty chart $C$
  **while** $C_{1,N} = \emptyset \wedge A \neq \emptyset$ **do**
    y $\leftarrow$ EXTRACT_MAX($A$)
    **if** $y \notin C$ **then**
      ADD($C, y$)
      **for** $y' \in$ RULES($C, y$) **do**
        INSERT($A, y'$)
  **return** $C_{1,N}$

The outputs from the LSTMs are projected into a further hidden layer, a bias is added, and a RELU non-linearity is applied. This layer gives a context-dependent representation of the word that is fed into a softmax over supertags.

We use a variant on the standard LSTM with coupled 'input' and 'forget' gates, and peephole connections. Each LSTM cell at position $t$ takes three inputs: a cell state vector $c_{t-1}$ and hidden state vector $h_{t-1}$ from the cell at position $t-1$, and $x_t$ from the layer below. It outputs $h_t$ to the layer above, and $c_t$ and $h_t$ to the cell at $t+1$. $c_t$ and $h_t$ are computed as follows, where $\sigma$ is the component-wise logistic sigmoid, and $\circ$ is the component-wise product:

$$i_t = \sigma(W_i[c_{t-1}, h_{t-1}, x_t] + b_i) \quad (3)$$

$$\tilde{c_t} = \tanh(W_c[h_{t-1}, x_t] + b_{\tilde{c}}) \quad (4)$$

$$o_t = \sigma(W_o[\tilde{c_t}, h_{t-1}, x_t] + b_o) \quad (5)$$

$$c_t = i_t \circ \tilde{c_t} + (\mathbf{1} - i_t)c_{t-1} \quad (6)$$

$$h_t = o_t \circ \tanh(c_t) \quad (7)$$

We train the model using stochastic gradient descent, with a minibatch size of 1, a learning rate of 0.01, and using momentum with $\mu = 0.7$. We then fine-tune models using a larger minibatch size of 32. Gradients whose $L_2$ norm exceeds 5 are clipped. Training was run for 30 epochs, shuffling the order of sentences after each epoch, and we used the model parameters with the highest development supertagging accuracy. The input layer uses dropout with a rate of 0.5. All trainable parameters have $L_2$ regularization of $\Lambda = 10^{-6}$. Word embedding are initialized using 50-dimensional pre-trained values from Turian et al. (2010). For prefix and suffix embeddings, we use randomly initialized 32-dimensional vectors—features occurring less than 3 times are replaced with an 'unknown' embedding. We add special start and end tokens to each sentence, with trainable parameters. The LSTM state size is 128 and the RELU layer has a size of 64.

## 4 Parsing Models

Our experiments focus on two parsing models:

**Supertag-Factored** We use the supertagging model described in Section 3 to build a supertag-factored parser, closely following the approach described in Section 2. We also add a penalty of 0.1 (tuned on development data) for every time a unary rule is applied in a parse. The attach-low heuristic is implemented by adding a small penalty of $-\epsilon d$ at every binary rule instantiation, where $d$ is the absolute distance between the heads of the left and right children, and $\epsilon$ is a small constant. We increase the penalty to $10\epsilon$ for clitics, to encourage these to attach locally. Because these penalties are $\leq 0$, they do not affect the A* upper bound calculations.

**Dependencies** We also train a model with dependency features, to investigate how much they improve accuracy beyond the supertag-factored model. We adapt a joint CCG and SRL model (Lewis et al., 2015) to CCGbank parsing, by assigning every CCGbank dependency a role based on its argument number (i.e., the first argument of every category has role *ARG0*). A global log-linear model is trained to maximize the marginal likelihood of the gold dependencies. We use the same features and hyperparameters as Lewis et al. (2015), except that we do not use the supertagger score feature (to separate the effect of the dependencies features from the supertagger). We choose this model because it has an A$^*$ parsing algorithm, meaning that we do not need to use aggressive beam search.

## 5 Semi-supervised Learning

A number of papers have shown that strong parsers can be improved by exploiting text without gold-standard annotations. Recent work suggests *tri-training*, in which the output of two parsers is intersected to create training data for a third parser, is highly effective (Weiss et al., 2015).

We perform the first application of tri-training to a lexicalized formalism. Following Weiss et al., we parse the corpus of Chelba et al. (2013) with a shift-reduce parser and a chart-based model. We use the shift-reduce parser from Ambati et al. (2016) and our dependency model (without using a supertagger feature, to limit the correlation with our tagging model). On development sentences where the parsers produce the same supertags (40%), supertagging accuracy is 98.0%. This subset is considerably easier than general text—our CCGbank-trained supertagger is 97.4% accurate on this data—but tri-training still provides useful additional training data.

In total, we include 43 million words of text that

the parsers annotate with the same supertags and 15 copies of the gold CCGbank training data. Our experiments show that tri-training improves both supertagging and parsing accuracy.

## 6 GPU Parsing

Our parser makes an unusual trade-off, by combining a complex tagging model with a deterministic parsing model. The A* parsing algorithm is extremely efficient, and the overall time required to process a sentence is dominated by the supertagger.

GPUs can improve performance over CPUs by computing many vector operations in parallel. There are two major obstacles to using GPUs for parsing. First, most models use sparse rather than dense features, which are difficult to compute efficiently on GPUs. The most successful implementation we are aware of exploits the fact that the Berkeley parser is unlexicalized to run parsing operations in parallel (Hall et al., 2014). Second, most neural models have features that depend on the current parse or stack state (e.g. Chen and Manning (2014)). This makes it difficult to exploit the parallelism of GPUs, because these data structures are typically built incrementally on CPU. It may be possible to write GPU-specific code that maintains the entire parse state on GPU, but we are not aware of any such implementations.

In contrast, our supertagger only uses matrix operations, and does not take any parse state as input—meaning it is straightforward to run on a GPU. To exploit the parallelism of GPUs, we process thousands of sentences simultaneously—improving parsing efficiency by an order-of-magnitude over CPU. A major advantage of our model is that it allows all of the computationally intensive decisions to occur on GPUs. Unlike existing GPU parsers, the LSTM can be run with generic library code.[2]

## 7 Experiments

### 7.1 Experimental setup

We trained our parser on Sections 02-21 of CCGbank (Hockenmaier and Steedman, 2007), using Section 00 for development, and Section 23 for test. Our experiments use a supertagger beam of $10^{-4}$—which does not affect the final scores, but reduces overheads such as building the initial agenda.

[2]We use TensorFlow (Abadi et al., 2015).

| Model | Dev | Test |
|---|---|---|
| C&C tagger | 91.5 | 92.0 |
| NN | 91.3 | 91.6 |
| RNN | 93.1 | 93.0 |
| LSTM | 94.1 | 94.3 |
| LSTM + Tri-training | **94.9** | **94.7** |

Table 1: Supertagging accuracy on CCGbank.

| Model | P | R | F1 |
|---|---|---|---|
| C&C | 86.2 | 84.2 | 85.2 |
| C&C + RNN | 87.7 | 86.4 | 87.0 |
| EASYCCG | 83.7 | 83.0 | 83.3 |
| Dependencies | 86.5 | 85.8 | 86.1 |
| LSTM | 87.7 | 86.7 | 87.2 |
| LSTM + Dependencies | 88.2 | 87.3 | 87.8 |
| LSTM + Tri-training | **88.6** | **87.5** | **88.1** |
| LSTM + Tri-training + Dependencies | 88.2 | 87.3 | 87.8 |

Table 2: Labelled F1 for CCGbank dependencies on the CCGbank test set (Section 23).

Where results are available, we compare our work with the following models: EASYCCG, which has the same parsing model as our parser, but uses a feed-forward neural-network supertagger (NN); the C&C parser (Clark and Curran, 2007), and C&C+RNN (Xu et al., 2015), which is the C&C parser with an RNN supertagger. All results are for 100% coverage of the test data.

We refer to the models described in Section 4 as LSTM and DEPENDENCIES respectively. We also report the performance of LSTM+DEPENDENCIES, which combines the model scores (weighting the LSTM score by 1.8, tuned on development data).

### 7.2 Supertagging Results

The most direct measure of the effectiveness of our LSTM and tri-training is on the supertagging task. Results are shown in Table 1. The improvement of our deep LSTM over the RNN model is greater than the improvement of the RNN over C&C model. Further gains follow from tri-training, improving the state-of-the-art by 1.7%.

### 7.3 English Parsing Results

Parsing results are shown in Figure 2. Surprisingly, our CCGBank-trained LSTM outperforms any previous approach.[3] The ensemble of the LSTM

[3]We cannot compare directly with Fowler and Penn (2010)'s adaptation of the Berkeley parser to CCG, or Auli and Lopez

| Model | QUESTIONS | | | BIOINFER | | |
|---|---|---|---|---|---|---|
| | P | R | F1 | P | R | F1 |
| C&C | - | - | 86.6 | 77.8 | 71.4 | 74.5 |
| EASYCCG | 78.1 | 78.2 | 78.1 | 76.8 | 77.6 | 77.2 |
| C&C + RNN | - | - | - | 80.1 | 75.5 | 77.7 |
| LSTM | 87.6 | 87.4 | 87.5 | 80.1 | 80.9 | 80.5 |
| LSTM + Dependencies | **88.2** | **87.9** | **88.0** | 77.8 | 80.1 | 79.4 |
| LSTM + Tri-training | - | - | - | **81.8** | **82.6** | **82.2** |

Table 3: Out-of-domain experiments.

| **Parser** | **Sentences per second** |
|---|---|
| SpaCy*[4] | 778 |
| Berkeley GPU* (Hall et al., 2014) | 687 |
| Chen and Manning (2014)* | 391 |
| C&C | 66 |
| EASYCCG | 606 |
| LSTM | 214 |
| LSTM + Dependencies | 58 |
| LSTM GPU | 2670 |

Table 4: Sentences parsed per second on our hardware. Parsers marked * use non-CCG formalisms but are the fastest available CPU and GPU parsers.

and the Dependency model outperforms the LSTM alone, showing that dependency features are capturing some generalizations that the LSTM does not. However, semi-supervised learning substantially improves the LSTM, matching the accuracy of the ensemble—showing that the LSTM is expressive enough to compensate given sufficient data.

### 7.4 Out-of-domain Experiments

We also evaluate on two out-of-domain datasets used by Rimell and Clark (2008), but did no development on this data. In both cases, we use Rimell and Clark's scripts for converting CCG parses to the target dependency representations. The datasets are:

**QUESTIONS** 500 questions from TREC (Rimell and Clark, 2008). Questions frequently contain very long range dependencies, providing an interesting test of the LSTM supertagger's ability to capture unbounded dependencies. We follow Rimell and Clark by re-training the supertagger on the concatenation of the CCGbank training data and 10 copies of the QUESTIONS training data.

**BIOINFER** 500 sentences from biomedical abstracts. This dataset tests the parser's robustness to a large amount of unseen vocabulary.

Results are shown in Table 3. Our LSTM parser outperforms existing work on question parsing, showing that it can successfully model the long-range dependencies found in questions. Adding dependency features yields only a small improvement.

On the BIOINFER corpus, our tri-trained LSTM parser is 4.5 F1 better than the previous state-of-the-art. Dependency features appear to be much less robust to unseen words than the LSTM tagging model, and are unhelpful. Because the parser was not trained or developed on this domain, it is likely to perform similarly well on other domains.

### 7.5 Efficiency Experiments

In contrast to standard parsing algorithms, the efficiency of our model depends directly on the accuracy of the supertagger in guiding the search. We therefore measure the efficiency empirically.

Results are shown in Table 4.[5] Our parser runs more slowly than EASYCCG on CPU, due to the more complex tagging model (but is 4.8 F1 more accurate). Adding dependencies substantially reduces efficiency, due to calculating sparse features. Without dependencies, the run time is dominated by the LSTM supertagger. Running the supertagger on a GPU reduces parsing times dramatically—outperforming SpaCy, the fastest publicly available parser (Choi et al., 2015). Roughly half the parsing time is spent on GPU supertagging, and half on CPU parsing. To better exploit batching in the GPU, our implementation dynamically buckets sentences by length (bins of width 10), and tags batches when the bucket size reaches 3072 (the number of threads on our GPU). We are not aware of any GPU implementations of shift-reduce parsers or lexicalized chart parsers, so it is unclear if most other state-of-the-art parsers can be adapted to exploit GPUs.

(2011b)'s joint parsing and supertagging model, due to differences in the experimental setup. These models are 0.3 and 1.5 F1 more accurate than the C&C baseline respectively, which is well within the margin of improvement obtained by our model.

[4] honnibal.github.io/spaCy

[5] All timing experiments use a single 3.5GHz core and (where applicable) a single NVIDIA TITAN X GPU.

| Supertagger | Accuracy |
|---|---|
| Bidirectional RNNs | 93.4 |
| Forward LSTM only | 83.5 |
| Backward LSTM only | 89.5 |
| Bidirectional LSTMs | **94.1** |

Table 5: Development supertagging accuracy.

| Word Class | NN | LSTM | LSTM+ Tri-training |
|---|---|---|---|
| All | 91.32 | 94.14 | **94.90** |
| Unseen Words | 90.39 | 94.21 | **95.26** |
| Unseen Usages | 45.80 | 59.37 | **62.46** |
| Prepositions | 78.11 | 84.40 | **85.98** |
| Verbs | 82.55 | 87.85 | **89.24** |
| Wh-words | 90.47 | 92.09 | **94.16** |
| Long range | 74.80 | 83.99 | **86.31** |

Table 6: Development supertagging accuracy on several classes of words. *Long range* refers to words taking an argument at least 5 words away.

## 8 Ablations

We also measure performance while removing different aspects of the full parsing model.

### 8.1 Supertagger Model Architecture

Numerous variations are possible on our supertagging architecture. Apart from tri-training, the major differences from the previous state-of-the-art (Xu et al., 2015) are that we use LSTMs rather than RNNs, and that we use bidirectional networks rather than only a forward-directional RNN. These modifications lead to a 1.3% improvement in accuracy. Table 5 shows performance while ablating these changes; they all contribute substantially to tagging accuracy.

Table 6 shows several classes of words where the LSTM model outperforms the baseline neural network that uses only local context (NN). The performance increase on unseen words is likely due to the fact that the LSTM can model more context to determine the category for a word. Unsurprisingly, this leads to a large improvement in accuracy for words taking non-local arguments. Finally, we see a large improvement in prepositional phrase attachment. This improvement is likely to be due to the deep architecture, which can better take into account the interaction between the preposition, its argument noun phrase, and its nominal or verbal attachment.

| Relaxation | LSTM+ Tri-train F1 | LSTM+ Dependencies F1 |
|---|---|---|
| - | 87.9 | 87.9 |
| $\{NP, N\}$ | 87.8 | 87.7 |
| $\{NP, PP\}$ | 87.7 | 87.6 |
| $\{NP, PP, N, N_{num}\}$ | 87.4 | 87.2 |
| * | 78.3 | 79.3 |

Table 7: Effect of simulating weaker grammars, by allowing the specified atomic categories to unify. * allows all atomic categories to unify, except conjunctions and punctuation. Results are on development sentences of length $\leq$40.

### 8.2 Semi-supervised learning

Table 6 also shows cases where the semi-supervised models perform better. Accuracy improves on unseen words—showing that tri-training can be a more effective way of generalizing to unseen words than pre-trained word embeddings alone. We also see improvement in accuracy on wh-words, which we attribute to the training data containing more examples of rare categories used for wh-words in pied-piping and similar constructions. One case where performance remains weak for all models is on unseen usages—where words occur in the CCGbank training data, but not with the category required in the test data. The improvement from tri-training is limited, likely due to the weakness of the baseline parses, and new techniques will be required to correct such errors.

### 8.3 Effect of Grammar

A subtle but crucial point is that our method depends on the strictness of the CCGbank grammar to exclude ungrammatical derivations. Because there is no dependency model, we rely on the deterministic CCG grammar as a hard constraint. There is a trade-off between restrictive grammars which may be brittle on noisy text, and weaker grammars that may overgenerate ungrammatical sentences.

We measure this trade-off by testing weaker grammars, which merge categories that are normally distinct. For example, if we merge $PP$ and $NP$, then an $S\backslash NP$ can take either a $PP$ or $NP$ argument.

Table 7 shows that relaxing the grammar significantly hurts performance; the deterministic constraints are crucial to training a high quality LSTM

CCG parser. With a very relaxed grammar in which all atoms can unify, dependencies features help compensate for the weakened grammar. Future work should explore further strengthening the grammar—-e.g. marking plurality on $NP$s to enforce plural agreement, or using slash-modalities to prevent over-generation arising from composition (Baldridge and Kruijff, 2003).

### 8.4 Effect of Dependency Features

Perhaps our most surprising result is that high accuracy can be achieved with a rule-based grammar and no dependency features. We performed several experiments to verify whether the model can capture long-range dependencies, and the extent to which dependency features are required to further improve parsing performance.

**Supertagging accuracy is still the bottleneck** A natural question is whether further improvements to our model will require a more powerful parsing model (such as adding dependency or derivation features), or if future work should focus on the supertagger. We found that on sentences where all the supertags are correct in the final parse (51%), the F1 is very high: 97.7. On parses containing supertag errors, the F1 drops to just 80.3. This result suggests that parsing accuracy can be significantly increased by improving the supertagger, and that very high performance could be attained only using a supertagging model.

**'Attach low' heuristic is surprisingly effective** Given a sequence of supertags, our grammar is still ambiguous. As explained in Section 2, we resolve these ambiguities by attaching low. To investigate the accuracy of this heuristic, we performed oracle decoding given the highest scoring supertags—and found that F1 improved by 1.3, showing that there are limits to what can be achieved with a rule-based grammar. In contrast, an 'attach high' heuristic scores 5.2 F1 less than attaching low, suggesting that these decisions are reasonably frequent, but that attaching low is much more common.

Would adding a dependency model help here? We consider several dependencies whose attachment is often ambiguous given the supertags. Results are shown in Table 8. Any improvements from the dependency model are small—it is difficult to improve on the 'attach low' heuristic with current models.

| Dependency | Attach Low Heuristic | Dependencies Model |
|---|---|---|
| Relative clause | 84.66 | **85.44** |
| Adnominal PP | 91.67 | **93.67** |
| Adverbial PP | **97.78** | 95.86 |
| Adverb | **99.09** | 98.09 |

Table 8: Per-relation accuracy for several dependencies whose attachments are often ambiguous given the supertags. Results are only on sentences where the parsers assign the correct supertags.

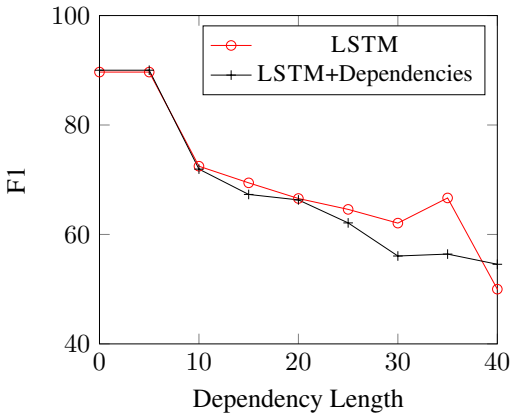


Figure 3: F1 on dependencies of various lengths.

**Supertag-factored model is accurate on long-range dependencies** One motivation for CCG parsing is to recover long-range dependencies. While we do not explicitly model these dependencies, they can still be extracted from the parse. Instead, we rely on the LSTM supertagger to implicitly model the dependencies—a task that becomes more challenging with longer dependencies. We investigate the accuracy of our parser for dependencies of different lengths. Figure 3 shows that adding dependencies features does not improve the recovery of long-range dependencies over the LSTM alone; the LSTM accurately models long-range dependencies.

## 9 Related Work

Recent work has applied neural networks to parsing, mostly using neural classifiers in shift-reduce parsers (Henderson et al., 2013; Chen and Manning, 2014; Dyer et al., 2015; Weiss et al., 2015). Unlike our approach, none of these report both state-of-the-art speed and accuracy. Vinyals et al. (2015) in-

stead propose embedding entire sentences in a vector space, and then generating parse trees as strings. Our model achieves state-of-the-art accuracy with a non-ensemble model trained on the standard training data, whereas their model requires ensembles or extra supervision to match the state of the art.

Most work on CCG parsing has either used CKY chart parsing (Hockenmaier, 2003; Clark and Curran, 2007; Fowler and Penn, 2010; Auli and Lopez, 2011a) or shift-reduce algorithms (Zhang and Clark, 2011; Xu et al., 2014; Ambati et al., 2015). These methods rely on beam-search to cope with the huge space of possible CCG parses. Instead, we use Lewis and Steedman (2014a)'s A* algorithm. By using a semi-supervised LSTM supertagger, we improved over Lewis and Steedman's parser by 4.8 F1.

CCG supertagging was first attempted with maximum-entropy Markov models (Clark, 2002)—in practice, the combination of sparse features and a large tag set makes such models brittle. Lewis and Steedman (2014b) applied feed-forward neural networks to supertagging, motivated by using pre-trained work embeddings to reduce sparsity. Xu et al. (2015) showed further improvements by using RNNs to condition on non-local context. Concurrently with this work, Xu et al. (2016) explored bidirectional RNN models, and Vaswani et al. (2016) use bidirectional LSTMs with a different training procedure.

Our tagging model is closely related to the bidirectional LSTM POS tagging model of Ling et al. (2015). We see larger gains over the state-of-the-art—likely because supertagging involves more long-range dependencies than POS tagging.

Other work has successfully applied GPUs to parsing, but has required GPU-specific code and algorithms (Yi et al., 2011; Johnson, 2011; Canny et al., 2013; Hall et al., 2014). GPUs have also been used for machine translation (He et al., 2015).

## 10 Conclusions and Future Work

We have shown that a combination of deep learning, linguistics and classic AI search can be used to build a parser with both state-of-the-art speed and accuracy. Future work will explore using our parser to recover other representations from CCG, such as Universal Dependencies (McDonald et al., 2013) or semantic roles. The major obstacle is the mismatch between these representations and CCGbank—we will therefore investigate new techniques for obtaining other representations from CCG parses. We will also explore new A* parsing algorithms that explicitly model the global parse structure using neural networks, while maintaining optimality guarantees.

## Acknowledgements

We thank Bharat Ram Ambati, Greg Coppola, Chloé Kiddon, Luheng He, Yannis Konstas and the anonymous reviewers for comments on an earlier version, and Mark Yatskar for helpful discussions.

This research was supported in part by the NSF (IIS-1252835), DARPA under the DEFT program through the AFRL (FA8750-13-2-0019), an Allen Distinguished Investigator Award, and a gift from Google.

## References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. TensorFlow: Large-scale Machine Learning on Heterogeneous Systems. *Software available from tensorflow.org*.

Bharat Ram Ambati, Tejaswini Deoskar, Mark Johnson, and Mark Steedman. 2015. An Incremental Algorithm for Transition-based CCG Parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2016. Shift-Reduce CCG Parsing using Neural Network Models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*.

Michael Auli and Adam Lopez. 2011a. A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*.

Michael Auli and Adam Lopez. 2011b. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Jason Baldridge and Geert-Jan M Kruijff. 2003. Multi-modal Combinatory Categorial Grammar. In *Proceed-*

*ings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2).

John Canny, David Hall, and Dan Klein. 2013. A multi-teraflop constituency parser using gpus. *Architecture*, 3:3–5.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. Technical report, Google.

Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750.

Jinho D. Choi, Joel Tetreault, and Amanda Stent. 2015. It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 387–396, Beijing, China, July. Association for Computational Linguistics.

Stephen Clark and James R Curran. 2007. Wide-coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4).

Stephen Clark. 2002. Supertagging for Combinatory Categorial Grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+ 6)*, pages 19–24.

Greg Durrett and Dan Klein. 2015. Neural CRF Parsing. In *Proceedings of the Association for Computational Linguistics*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based Dependency Parsing with Stack Long Short-Term Memory. In *Proc. ACL*.

Timothy AD Fowler and Gerald Penn. 2010. Accurate Context-free Parsing with Combinatory Categorial Grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

David Hall, Taylor Berg-Kirkpatrick, and Dan Klein. 2014. Sparser, better, faster gpu parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 208–217, Baltimore, Maryland, June. Association for Computational Linguistics.

Hua He, Jimmy Lin, and Adam Lopez. 2015. Gappy Pattern Matching on GPUs for On-Demand Extraction of Hierarchical Translation Grammars. *TACL*, 3:87–100.

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multi-lingual Joint Parsing of Syntactic and Semantic Dependencies with a Latent Variable Model. *Computational Linguistics*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation*, 9(8):1735–1780.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: a Corpus of CCG derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3).

Julia Hockenmaier. 2003. *Data and models for statistical parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh. College of Science and Engineering. School of Informatics.

Mark Johnson. 2011. Parsing in Parallel on Multiple Cores and GPUs. In *Proceedings of the Australasian Language Technology Association Workshop 2011*, pages 29–37, Canberra, Australia, December.

Mike Lewis and Mark Steedman. 2014a. A* CCG Parsing with a Supertag-factored Model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Mike Lewis and Mark Steedman. 2014b. Improved CCG parsing with Semi-supervised Supertagging. *Transactions of the Association for Computational Linguistics*.

Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint A* CCG Parsing and Semantic Role Labelling. In *Empirical Methods in Natural Language Processing*.

Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *EMNLP*, pages 1520–1530. The Association for Computational Linguistics.

Ryan T McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith B Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal Dependency Annotation for Multilingual Parsing. In *ACL (2)*, pages 92–97.

Laura Rimell and Stephen Clark. 2008. Adapting a Lexicalized-grammar Parser to Contrasting Domains. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with Compositional Vector Grammars. In *Proceedings of the ACL conference*.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: A Simple and General Method for Semi-supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging With LSTMs . In *Proceedings of the Human Language Technology Conference of the NAACL*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured Training for Neural Network Transition-Based Parsing. In *Proceedings of ACL 2015*, pages 323–333.

Wenduan Xu, Stephen Clark, and Yue Zhang. 2014. Shift-Reduce CCG Parsing with a Dependency Model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG Supertagging with a Recurrent Neural Network. *Volume 2: Short Papers*, page 250.

Wenduan Xu, Michael Auli, and Stephen Clark. 2016. Shift-Reduce CCG Parsing with Recurrent Neural Networks and Expected F-Measure Training. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Youngmin Yi, Chao-Yue Lai, Slav Petrov, and Kurt Keutzer. 2011. Efficient parallel cky parsing on gpus. In *Proceedings of the 12th International Conference on Parsing Technologies*, IWPT '11, pages 175–185, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2011. Shift-reduce CCG Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*.