

# Global Neural CCG Parsing with Optimality Guarantees

Kenton Lee    Mike Lewis    Luke Zettlemoyer

Computer Science & Engineering

University of Washington

Seattle, WA 98195

{kentonl,mlewis,lsz}@cs.washington.edu

## Abstract

We introduce the first global recursive neural parsing model with optimality guarantees during decoding. To support global features, we give up dynamic programs and instead search directly in the space of all possible subtrees. Although this space is exponentially large in the sentence length, we show it is possible to learn an efficient A\* parser. We augment existing parsing models, which have informative bounds on the outside score, with a global model that has loose bounds but only needs to model non-local phenomena. The global model is trained with a novel objective that encourages the parser to search both efficiently and accurately. The approach is applied to CCG parsing, improving state-of-the-art accuracy by 0.4 F1. The parser finds the optimal parse for 99.9% of held-out sentences, exploring on average only 190 subtrees.

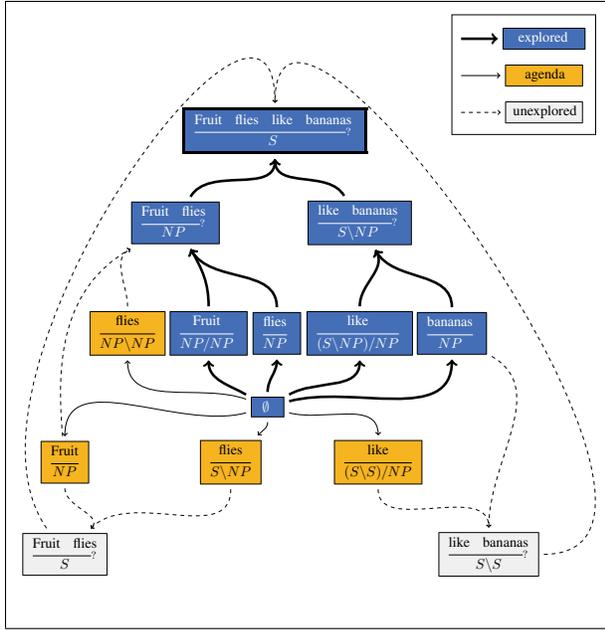
## 1 Introduction

Recursive neural models perform well for many structured prediction problems, in part due to their ability to learn representations that depend globally on all parts of the output structures. However, global models of this sort are incompatible with existing exact inference algorithms, since they do not decompose over substructures in a way that allows effective dynamic programming. Existing work has therefore used greedy inference techniques such as beam search (Vinyals et al., 2015; Dyer et al., 2015) or reranking (Socher et al., 2013). We introduce the first global recursive neural parsing approach

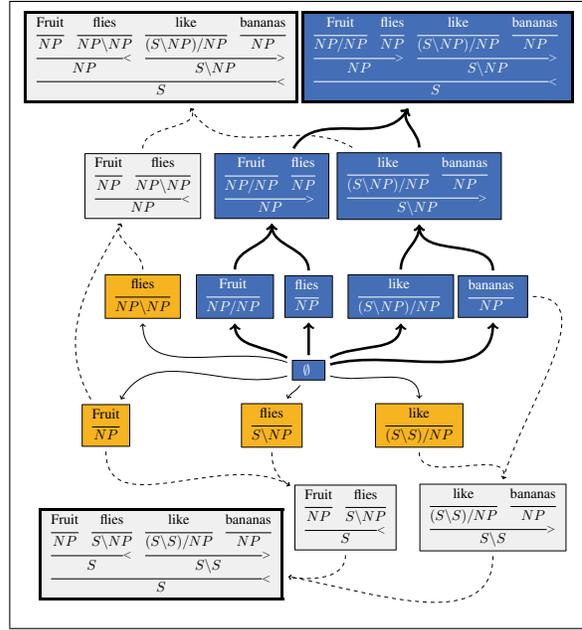
with optimality guarantees for decoding and use it to build a state-of-the-art CCG parser.

To enable learning of global representations, we modify the parser to search directly in the space of all possible parse trees with no dynamic programming. Optimality guarantees come from A\* search, which provides a certificate of optimality if run to completion with a heuristic that is a bound on the future cost. Generalizing A\* to global models is challenging; these models also break the locality assumptions used to efficiently compute existing A\* heuristics (Klein and Manning, 2003; Lewis and Steedman, 2014). Rather than directly replacing local models, we show that they can simply be augmented by adding a score from a global model that is constrained to be non-positive and has a trivial upper bound of zero. The global model, in effect, only needs to model the remaining non-local phenomena. In our experiments, we use a recent factored A\* CCG parser (Lewis et al., 2016) for the local model, and we train a Tree-LSTM (Tai et al., 2015) to model global structure.

Finding a model that achieves these A\* guarantees in practice is a challenging learning problem. Traditional structured prediction objectives focus on ensuring that the gold parse has the highest score (Collins, 2002; Huang et al., 2012). This condition is insufficient in our case, since it does not guarantee that the search will terminate in sub-exponential time. We instead introduce a new objective that optimizes efficiency as well as accuracy. Our loss function is defined over states of the A\* search agenda, and it penalizes the model whenever the top agenda item is not a part of the gold parse.



(a) The search space in chart parsing, with one node for each labeling of a span.



(b) The search space in this work, with one node for each partial parse.

**Figure 1:** Illustrations of CCG parsing as hypergraph search, showing partial views of the search space. Weighted hyperedges from child nodes to a parent node represent rule productions scored by a parsing model. A path starting at  $\emptyset$ , for example the set of bolded hyperedges, represents the derivation of a parse. During decoding, we find the highest scoring path to a complete parse. Both figures show an ideal exploration that efficiently finds the optimal path. Figure 1a depicts the traditional search space, and Figure 1b depicts the search space in this work. Hyperedge scores can only depend on neighboring nodes, so our model can condition on the entire parse structure, at the price of an exponentially larger search space.

Minimizing this loss encourages the model to return the correct parse as quickly as possible.

The combination of global representations and optimal decoding enables our parser to achieve state-of-the-art accuracy for Combinatory Categorical Grammar (CCG) parsing. Despite being intractable in the worst case, the parser in practice is highly efficient. It finds optimal parses for 99.9% of held out sentences while exploring just 190 subtrees on average—allowing it to outperform beam search in both speed and accuracy.

## 2 Overview

**Parsing as hypergraph search** Many parsing algorithms can be viewed as a search problem, where parses are specified by paths through a hypergraph.

A node  $y$  in this hypergraph is a labeled span, representing structures within a parse tree, as shown in Figure 1. Each hyperedge  $e$  in the hypergraph represents a rule production in a parse. The head node

of the hyperedge  $\text{HEAD}(e)$  is the parent of the rule production, and the tail nodes of the hyperedge are the children of the rule production. For example, consider the hyperedge in Figure 1b whose head is *like bananas*. This hyperedge represents a forward application rule applied to its tails, *like* and *bananas*.

To define a path in the hypergraph, we first include a special start node  $\emptyset$  that represents an empty parse.  $\emptyset$  has outgoing hyperedges that reach every leaf node, representing assignments of labels to words (supertag assignments in Figure 1). We then define a path to be a set of hyperedges  $E$  starting at  $\emptyset$  and ending at a single destination node. A path therefore specifies the derivation of the parse constructed from the labeled spans at each node. For example, in Figure 1, the set of bolded hyperedges form a path deriving a complete parse.

Each hyperedge  $e$  is weighted by a score  $s(e)$  from a parsing model. The score of a path  $E$  is the

sum of its hyperedge scores:

$$g(E) = \sum_{e \in E} s(e)$$

Viterbi decoding is equivalent to finding the highest scoring path that forms a complete parse.

**Search on parse forests** Traditionally, the hypergraph represents a packed parse chart. In this work, our hypergraph instead represents a *forest* of parses. Figure 1 contrasts the two representations.

In the parse chart, labels on the nodes represent local properties of a parse, such as the category of a span in Figure 1a. As a result, multiple parses that contain the same property include the same node in their path, (e.g. the node spanning the phrase *Fruit flies* with category NP). The number of nodes in this hypergraph is polynomial in the sentence length, permitting exhaustive exploration (e.g. CKY parsing). However, the model scores can only depend on local properties of a parse. We refer to these models as *locally factored* models.

In contrast, nodes in the parse forest are labeled with entire subtrees, as shown in Figure 1b. For example, there are two nodes spanning the phrase *Fruit flies* with the same category NP but different internal substructures. While the parse forest requires an exponential number of nodes in the hypergraph, the model scores can depend on entire subtrees.

**A\* parsing** A\* parsing has been successfully applied in locally factored models (Klein and Manning, 2003; Lewis and Steedman, 2014; Lewis et al., 2015; Lewis et al., 2016). We present a special case of A\* parsing that is conceptually simpler, since the parse forest constrains each node to be reachable via a unique path. During exploration, we maintain the unique (and therefore highest scoring) path to a hyperedge  $e$ , which we define as  $\text{PATH}(e)$ .

Similar to the standard A\* search algorithm, we maintain an agenda  $\mathcal{A}$  of hyperedges to explore and a forest  $\mathcal{F}$  of explored nodes that initially contains only the start node  $\emptyset$ .

Each hyperedge  $e$  in the agenda is sorted by the sum of its inside score  $g(\text{PATH}(e))$  and an admissible heuristic  $h(e)$ . A heuristic  $h(e)$  is admissible if it is an upper bound of the sum of hyperedge scores leading to any complete parse reachable from  $e$  (the Viterbi outside score). The efficiency of the search

improves when this bound is tighter.

At every step, the parser removes the top of the agenda,  $e_{max} = \text{argmax}_{e \in \mathcal{A}} (g(\text{PATH}(e)) + h(e))$ .  $e_{max}$  is expanded by combining  $\text{HEAD}(e_{max})$  with previously explored parses from  $\mathcal{F}$  to form new hyperedges. These new hyperedges are inserted into  $\mathcal{A}$ , and  $\text{HEAD}(e_{max})$  is added to  $\mathcal{F}$ . We repeat these steps until the first complete parse  $y^*$  is explored. The bounds provided by  $h(e)$  guarantee that the path to  $y^*$  has the highest possible score. Figure 1b shows an example of the agenda and the explored forest at the end of perfectly efficient search, where only the optimal path is explored.

**Approach** The enormous search space described above presents a challenge for an A\* parser, since computing a tight and admissible heuristic is difficult when the model does not decompose locally.

Our key insight in addressing this challenge is that existing locally factored models with an informative A\* heuristic can be augmented with a global score (Section 3). By constraining the global score to be non-positive, the A\* heuristic from the locally factored model is still admissible.

While the heuristic from the local model offers some estimate of the future cost, the efficiency of the parser requires learning a well-calibrated global score, since the heuristic becomes looser as the global score provides stronger penalties (Section 5).

As we explore the search graph, we incrementally construct a neural network, which computes representations of the parses and allows backpropagation of errors from bad search steps (Section 4).

In the following sections, we present our approach in detail, assuming an existing locally factored model  $s_{local}(e)$  for which we can efficiently compute an admissible A\* heuristic  $h(e)$ .

In the experiments, we apply our model to CCG parsing, using the locally factored model and A\* heuristic from Lewis et al. (2016).

### 3 Model

Our model scores a hyperedge  $e$  by combining the score from the local model with a global score that conditions on the entire parse at the head node:

$$s(e) = s_{local}(e) + s_{global}(e)$$

In  $s_{global}(e)$ , we first compute a hidden representation encoding the parse structure of  $y = \text{HEAD}(e)$ . We use a variant of the Tree-LSTM (Tai et al., 2015) connected to a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) at the leaves. The combination of linear and tree LSTMs allows the hidden representation of partial parses to condition on both the partial structure and the full sentence. Figure 2 depicts the neural network that computes the hidden representation for a parse.

Formally, given a sentence  $\langle w_1, w_2, \dots, w_n \rangle$ , we compute hidden states  $h_t$  and cell states  $c_t$  in the forward LSTM for  $1 < t \leq n$ :

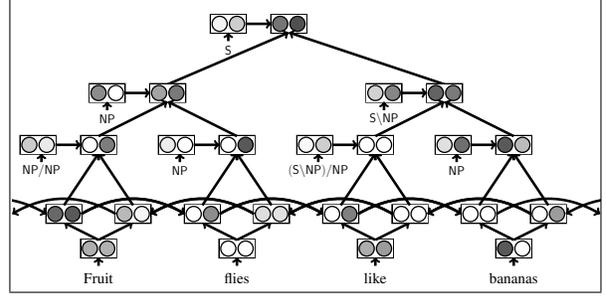
$$\begin{aligned} i_t &= \sigma(W_i[c_{t-1}, h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o[\tilde{c}_t, h_{t-1}, x_t] + b_o) \\ \tilde{c}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\ c_t &= i_t \circ \tilde{c}_t + (\mathbf{1} - i_t) \circ c_{t-1} \\ h_t &= o_t \circ \tanh(c_t) \end{aligned}$$

where  $\sigma$  is the logistic sigmoid,  $\circ$  is the component-wise product, and  $x_t$  denotes a learned word embedding for  $w_t$ . We also construct a backward LSTM, which produces the analogous hidden and cell states starting at the end of the sentence, which we denote as  $c'_t$  and  $h'_t$  respectively. The start and end latent states,  $c_{-1}, h_{-1}, c'_{n+1}$ , and  $h'_{n+1}$ , are learned embeddings. This variant of the LSTM includes peephole connections and couples the input and forget gates.

The bidirectional LSTM over the words serves as a base case when we recursively compute a hidden representation for the parse  $y$  using the tree-structured generalization of the LSTM:

$$\begin{aligned} i_y &= \sigma(W_i^R[c_l, h_l, c_r, h_r, x_y] + b_i^R) \\ f_y &= \sigma(W_f^R[c_l, h_l, c_r, h_r, x_y] + b_f^R) \\ o_y &= \sigma(W_o^R[\tilde{c}_y, h_l, h_r, x_y] + b_o^R) \\ c_{lr} &= f_y \circ c_l + (\mathbf{1} - f_y) \circ c_r \\ \tilde{c}_y &= \tanh(W_c^R[h_l, h_r, x_y] + b_c^R) \\ c_y &= i_y \circ \tilde{c}_y + (\mathbf{1} - i_y) \circ c_{lr} \\ h_y &= o_y \circ \tanh(c_y) \end{aligned}$$

where the weights and biases are parametrized by the rule  $R$  that produces  $y$  from its children, and  $x_y$  denotes a learned embedding for the category at the root of  $y$ . For example, in CCG, the rule would correspond to the CCG combinator, and the label would



**Figure 2:** Visualization of the Tree-LSTM which computes vector embeddings for each parse node. The leaves of the Tree-LSTM are connected to a bidirectional LSTM over words, encoding lexical information within and outside of the parse.

correspond to the CCG category.

We assume that nodes are binary, unary, or leaves. Their left and right latent states,  $c_l, h_l, c_r$ , and  $h_r$  are defined as follows:

- In a binary node,  $c_l$  and  $h_l$  are the cell and hidden states of the left child, and  $c_r$  and  $h_r$  are the cell and hidden states of the right child.
- In a unary node,  $c_l$  and  $h_l$  are learned embeddings, and  $c_r$  and  $h_r$  are the cell and hidden states of the singleton child.
- In a leaf node, let  $w$  denote the index of the corresponding word. Then  $c_l$  and  $h_l$  are  $c_w$  and  $h_w$  from the forward LSTM, and  $c_r$  and  $h_r$  are  $c'_w$  and  $h'_w$  from the backward LSTM.

The cell state of the recursive unit is a linear combination of the intermediate cell state  $\tilde{c}_y$ , the left cell state  $c_l$ , and the right cell state  $c_r$ . To preserve the normalizing property of coupled gates, we perform coupling in a hierarchical manner: the input gate  $i_y$  decides the weights for  $\tilde{c}_y$ , and the forget gate  $f_y$  shares the remaining weights between  $c_l$  and  $c_r$ .

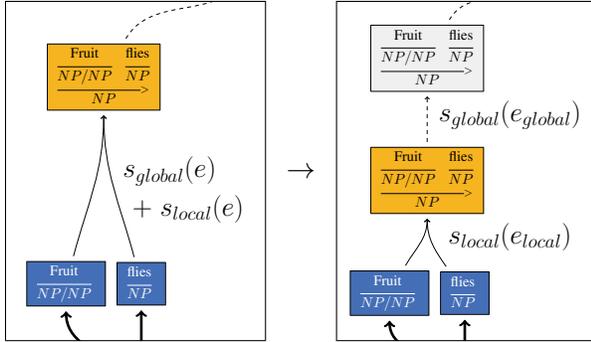
Given the hidden representation  $h_y$  at the root, we score the global component as follows:

$$s_{global}(e) = \log(\sigma(W \cdot h_y))$$

This definition of the global score ensures that it is non-positive—an important property for inference.

## 4 Inference

Using the hyperedge scoring model  $s(e)$  described in Section 3, we can find the highest scoring path that derives a complete parse tree by using the  $A^*$  parsing algorithm described in Section 2.



**Figure 3:** The hyperedge on the left requires computing both the local and global score when placed on the agenda. Splitting the hyperedge, as shown on the right, saves expensive computation of the global score if the local score alone indicates that the parse is not worth exploring.

**Admissible  $A^*$  heuristic** Since our full model adds non-positive global scores to the existing local scores, path scores under the full model cannot be greater than path scores under the local model. Upper bounds for path scores under the local model also hold for path scores under the full model, and we simply reuse the  $A^*$  heuristic from the local model to guide the full model during parsing without sacrificing optimality guarantees.

**Incremental neural network construction** The recursive hidden representations used in  $s_{global}(e)$  can be computed in constant time during parsing. When scoring a new hyperedge, its children must have been previously scored. Instead of computing the full recursion, we reuse the existing latent states of the children and compute  $s_{global}(e)$  with an incremental forward pass over a single recursive unit in the neural network. By maintain the latent states of each parse, we incrementally build a single DAG-structured LSTM mirroring the explored subset of the hypergraph. This not only enables quick forward passes during decoding, but also allows back-propagation through the search space after decoding, which is crucial for efficient learning (see Section 5).

**Lazy global scoring** The global score is expensive to compute. We introduce an optimization to avoid computing it when provably unnecessary. We split each hyperedge  $e$  into two successive hyperedges,  $e_{local}$  and  $e_{global}$ , as shown in Figure 3. The score for  $e$ , previously  $s(e) = s_{local}(e) + s_{global}(e)$ , is

also split between the two new hyperedges:

$$s(e_{local}) = s_{local}(e_{local})$$

$$s(e_{global}) = s_{global}(e_{global})$$

Intuitively, this transformation requires  $A^*$  to verify that the local score is good enough before computing the global score, which requires an incremental forward pass over a recursive unit in the neural network. In the example, this involves first summing the supertag scores of *Fruit* and *flies* and inserting the result back into the agenda. The score for applying the forward application rule to the recursive representations is only computed if that item appears again at the head of the agenda. In practice, the lazy global scoring reduces the number of recursive units by over 91%, providing a 2.4X speed up.

## 5 Learning

During training (Algorithm 1), we assume access to sentences labeled with gold parse trees  $\hat{y}$  and gold derivations  $\hat{E}$ . The gold derivation  $\hat{E}$  is a path from  $\emptyset$  to  $\hat{y}$  in the parse forest.

$A^*$  search with our global model is not guaranteed to terminate in sub-exponential time. This creates challenges for learning—for example, it is not possible in practice to use the standard structured perceptron update (Collins, 2002), because the search procedure rarely terminates early in training. Other common loss functions assume inexact search (Huang et al., 2012), and do not optimize efficiency.

Instead, we optimize a new objective that is tightly coupled with the search procedure. During parsing, we would like hyperedges from the gold derivation to appear at the top of the agenda  $\mathcal{A}$ . When this condition does not hold,  $A^*$  is searching inefficiently, and we refer to this as a *violation* of the agenda, which we formally define as:

$$v(\hat{E}, \mathcal{A}) = \max_{e \in \mathcal{A}} (g(\text{PATH}(e)) + h(e))$$

$$- \max_{e \in \mathcal{A} \cap \hat{E}} (g(\text{PATH}(e)) + h(e))$$

where  $g(\text{PATH}(e))$  is the score of the unique path to  $e$ , and  $h(e)$  is the  $A^*$  heuristic. If all violations are zero, we find the gold parse without exploring any incorrect partial parses—maximizing both accuracy and efficiency. Figure 1b shows such a case—if any other nodes were explored, they would be violations.

Update	Loss( $\mathcal{V}$ )
Greedy	$\mathcal{V}_1$
Max violation	$\max_{t=1}^T \mathcal{V}_t$
All violations	$\sum_{t=1}^T \mathcal{V}_t$

**Table 1:** Loss functions optimized by the different update methods. The updates depend on the list of  $T$  non-zero violations,  $\mathcal{V} = \langle \mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_T \rangle$ , as defined in Section 5.

In existing work on violation-based updates, comparisons are only made between derivations with the same number of steps (Huang et al., 2012; Clark et al., 2015)—whereas our definition allows subtrees of arbitrary spans to compete with each other, because hyperedges are not explored in a fixed order. Our violations also differ from Huang et al.’s in that we optimize efficiency as well as accuracy.

We define loss functions over these violations, which are minimized to encourage correct and efficient search. During training, we parse each sentence until either the gold parse is found or we reach computation limits. We record  $\mathcal{V}$ , the list of non-zero violations of the agenda  $\mathcal{A}$  observed:

$$\mathcal{V} = \langle v(\hat{E}, \mathcal{A}) \mid v(\hat{E}, \mathcal{A}) > 0 \rangle$$

We can optimize several loss functions over  $\mathcal{V}$ , as defined in Table 1. The greedy and max-violation updates are roughly analogous to the violation-fixing updates proposed by Huang et al. (2012), but adapted to exact agenda-based parsing. We also introduce a new *all-violations* update, which minimizes the sum of all observed violations. The all-violations update encourages correct parses to be explored early (similar to the greedy update) while being robust to parses with multiple deviations from the gold parse (similar to the max-violation update).

The violation losses are optimized with subgradient descent and backpropagation. For our experiments,  $s_{local}(e)$  and  $h(e)$  are kept constant. Only the parameters  $\theta$  of  $s_{global}(e)$  are updated. Therefore, a subgradient of a violation  $v(\hat{E}, \mathcal{A})$  can be computed by summing subgradients of the global scores.

$$\frac{\partial v(\hat{E}, \mathcal{A})}{\partial \theta} = \sum_{e \in \text{PATH}(e_{max})} \frac{\partial s_{global}(e)}{\partial \theta} - \sum_{e \in \text{PATH}(\hat{e}_{max})} \frac{\partial s_{global}(e)}{\partial \theta}$$

where  $e_{max}$  denotes the hyperedge at the top of the agenda  $\mathcal{A}$  and  $\hat{e}_{max}$  denotes the hyperedge in the gold derivation  $\hat{E}$  that is closest to the top of  $\mathcal{A}$ .

---

## Algorithm 1 Violation-based learning algorithm

---

**Definitions**  $D$  is the training data containing input sentences  $x$  and gold derivations  $\hat{E}$ .  $e$  variables denote scored hyperedges.  $\text{TAG}(x)$  returns a set of scored pre-terminals for every word.  $\text{ADD}(\mathcal{F}, y)$  adds partial parse  $y$  to forest  $\mathcal{F}$ .  $\text{RULES}(\mathcal{F}, y)$  returns the set of scored hyperedges that can be created by combining  $y$  with entries in  $\mathcal{F}$ .  $\text{SIZE\_OK}(\mathcal{F}, \mathcal{A})$  returns whether the sizes of the forest and agenda are within predefined limits.

```

1: function VIOLATIONS( $\hat{E}, x, \theta$ )
2:    $\mathcal{V} \leftarrow \emptyset$  ▷ Initialize list of violations  $\mathcal{V}$ 
3:    $\mathcal{F} \leftarrow \emptyset$  ▷ Initialize forest  $\mathcal{F}$ 
4:    $\mathcal{A} \leftarrow \emptyset$  ▷ Initialize agenda  $\mathcal{A}$ 
5:   for  $e \in \text{TAG}(x)$  do
6:     PUSH( $\mathcal{A}, e$ )
7:   while  $|\mathcal{A} \cap \hat{E}| > 0$  and SIZE_OK( $\mathcal{F}, \mathcal{A}$ ) do
8:     if  $v(\hat{E}, \mathcal{A}) > 0$  then
9:       APPEND( $\mathcal{V}, v(\hat{E}, \mathcal{A})$ ) ▷ Record violation
10:       $e_{max} \leftarrow \text{EXTRACT\_MAX}(\mathcal{A})$  ▷ Pop agenda
11:      ADD( $\mathcal{F}, \text{HEAD}(e_{max})$ ) ▷ Explore hyperedge
12:      for  $e \in \text{RULES}(\mathcal{F}, \text{HEAD}(e_{max}), \theta)$  do
13:        PUSH( $\mathcal{A}, e$ ) ▷ Expand hyperedge
14:      return  $\mathcal{V}$ 
15:
16: function LEARN( $D$ )
17:   for  $i = 1$  to  $T$  do
18:     for  $x, \hat{E} \in D$  do
19:        $\mathcal{V} \leftarrow \text{VIOLATIONS}(\hat{E}, x, \theta)$ 
20:        $L \leftarrow \text{LOSS}(\mathcal{V})$ 
21:        $\theta \leftarrow \text{OPTIMIZE}(L, \theta)$ 
22:   return  $\theta$ 

```

---

## 6 Experiments

### 6.1 Data

We trained our parser on Sections 02-21 of CCG-bank (Hockenmaier and Steedman, 2007), using Section 00 for development and Section 23 for test. To recover a single gold derivation for each sentence to use during training, we find the right-most branching parse that satisfies the gold dependencies.

### 6.2 Experimental Setup

For the local model, we use the *supertag-factored* model of Lewis et al. (2016). Here,  $s_{local}(e)$  corresponds to a supertag score if a  $\text{HEAD}(e)$  is a leaf and zero otherwise. The outside score heuristic is computed by summing the maximum supertag score for every word outside of each span. In the reported results, we back off to the supertag-factored model after the forest size exceeds 500,000, the agenda size exceeds 2 million, or we build more than 200,000 recursive units in the neural network.

Model	Dev F1	Test F1
C & C	83.8	85.2
C & C + RNN	86.3	87.0
Xu (2016)	87.5	87.8
Vaswani et al. (2016)	87.8	88.3
Supertag-factored	87.5	88.1
<b>Global A*</b>	<b>88.4</b>	<b>88.7</b>

**Table 2:** Labeled F1 for CCGbank dependencies on the CCGbank development and test set for our system **Global A\*** and the baselines.

Our full system is trained with all-violations updates. During training, we lower the forest size limit to 2000 to reduce training times. The model is trained for 30 epochs using ADAM (Kingma and Ba, 2014), and we use early stopping based on development F1. The LSTM cells and hidden states have 64 dimensions. We initialize word representations with pre-trained 50-dimensional embeddings from Turian et al. (2010). Embeddings for categories have 16 dimensions and are randomly initialized. We also apply dropout with a probability of 0.4 at the word embedding layer during training. Since the structure of the neural network is dynamically determined, we do not use mini-batches. The neural networks are implemented using the CNN library,<sup>1</sup> and the CCG parser is implemented using the EasySRL library.<sup>2</sup> The code is available online.<sup>3</sup>

### 6.3 Baselines

We compare our parser to several baseline CCG parsers: the C&C parser (Clark and Curran, 2007); C&C + RNN (Xu et al., 2015), which is the C&C parser with an RNN supertagger; Xu (2016), a LSTM shift-reduce parser; Vaswani et al. (2016) who combine a bidirectional LSTM supertagger with a beam search parser using global features (Clark et al., 2015); and *supertag-factored* (Lewis et al., 2016), which uses deterministic A\* decoding and an LSTM supertagging model.

### 6.4 Parsing Results

Table 2 shows parsing results on the test set. Our global features let us improve over the supertag-factored model by 0.6 F1. Vaswani et al. (2016) also

<sup>1</sup><https://github.com/clab/cnn>

<sup>2</sup><https://github.com/mikelewis0/EasySRL>

<sup>3</sup><https://github.com/kentonl/neuralccg>

Model	Dev F1	Optimal	Explored
Supertag-factored	87.5	100.0%	402.5
– dynamic program	87.5	97.1%	17119.6
Span-factored	87.9	99.9%	176.5
– dynamic program	87.8	99.5%	578.5
<b>Global A*</b>	<b>88.4</b>	<b>99.8%</b>	<b>309.6</b>
– lexical inputs	87.8	99.6%	538.5
– lexical context	88.1	99.4%	610.5

**Table 3:** Ablations of our full model (**Global A\***) on the development set. *Explored* refers to the size of the parse forest. Results show the importance of global features and lexical information in context.

use global features, but our optimal decoding leads to an improvement of 0.4 F1.

Although we observed an overall improvement in parsing performance, the supertag accuracy was not significantly different after applying the parser.

On the test data, the parser finds the optimal parse for 99.9% sentences before reaching our computational limits. On average, we parse 27.1 sentences per second,<sup>4</sup> while exploring only 190.2 subtrees.

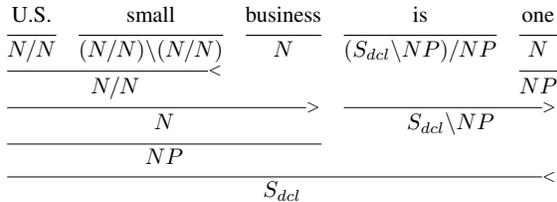
### 6.5 Model Ablations

We ablate various parts of the model to determine how they contribute to the accuracy and efficiency of the parser, as shown in Table 3. For each model, the comparisons include the average number of parses explored and the percentage of sentences for which an optimal parse can be found without backing off.

**Structure ablation** We first ablate the global score,  $s_{global}(y)$ , from our model, thus relying entirely on the local supertag-factors that do not explicitly model the parse structure. This ablation allows dynamic programming and is equivalent to the back-off model (*supertag-factored* in Table 3). Surprisingly, even in the exponentially larger search space, the global model explores *fewer* nodes than the supertag-factored model—showing that the global model efficiently prune large parts of the search space. This effect is even larger when not using dynamic programming in the supertag-factored model.

**Global structure ablation** To examine the importance of global features, we ablate the recursive hidden representation (*span-factored* in Table 3). The model in this ablation decomposes over labels for

<sup>4</sup>We use a single 3.5GHz CPU core.



**Figure 4:** Example of an incorrect partial parse that appears syntactically plausible in isolation. The full sentence is ‘Indeed, for many Japanese trading companies, the favorite **U.S. small business is one** whose research and development can be milked for future Japanese use.’ The global model heavily penalizes this garden path, thereby avoiding regions that lead to dead ends and allowing the global model to explore fewer nodes.

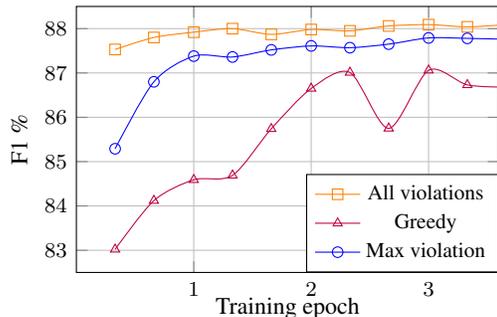
spans, as in Durrett and Klein (2015). In this model, the recursive unit uses, instead of latent states from its children, the latent states of the backward LSTM at the start of the span and the latent states of the forward LSTM at the end of the span. Therefore, this model encodes the lexical information available in the full model but does not encode the parse structure beyond the local rule production. While the dynamic program allows this model to find the optimal parse with fewer explorations, the lack of global features significantly hurts its parsing accuracy.

**Lexical ablation** We also show lexical ablations instead of structural ablations. We remove the bidirectional LSTM at the leaves, thus delexicalizing the global model. This ablation degrades both accuracy and efficiency, showing that the model uses lexical information to discriminate between parses.

To understand the importance of contextual information, we also perform a partial lexical ablation by using word embeddings at the leaves instead of the bidirectional LSTM, thus propagating only lexical information from within the span of each parse. The degradation in F1 is about half of the degradation from the full lexical ablation, suggesting that a significant portion of the lexical cues comes from the context of a parse. Figure 4 illustrates the importance of context with an incorrect partial parse that appears syntactically plausible in isolation. These bottom-up garden paths are typically problematic for parsers, since their incompatibility with the remaining sentence is difficult to recognize until later stages of decoding. However, our global model learns to heavily penalize these garden paths by using the context provided by the bidirectional LSTM

Update	Dev F1	Optimal	Explored
Greedy	87.9	99.2%	2313.8
Max-violation	88.1	99.9%	217.3
<b>All-violations</b>	<b>88.4</b>	99.8%	309.6

**Table 4:** Parsing results trained with different update methods. Our system uses **all-violations** updates and is the most accurate.



**Figure 5:** Learning curves for the first 3 training epochs on the development set when training with different updates strategies. The all-violations update shows the fastest convergence.

and avoid paths that lead to dead ends or bad regions of the search space.

## 6.6 Update Comparisons

Table 4 compares the different violation-based learning objectives, as discussed in Section 5. Our novel *all-violation* updates outperform the alternatives. We attribute this improvement to the robustness over poor search spaces, which the greedy update lacks, and the incentive to explore good parses early, which the max-violation update lacks. Learning curves in Figure 5 show that the all-violations update also converges more quickly.

## 6.7 Decoder Comparisons

Lastly, to show that our parser is both more accurate and efficient than other decoding methods, we decode our full model using best-first search, reranking, and beam search. Table 5 shows the F1 scores with and without the backoff model, the portion of the sentences that each decoder is able to parse, and the time spent decoding relative to the A\* parser.

In the best-first search comparison, we do not include the informative A\* heuristic, and the parser completes very few parses before reaching computational limits—showing the importance of heuristics in large search spaces. In the reranking comparison,

Decoder	Dev F1	Dev F1 – backoff	Relative Time
<b>Global A*</b>	<b>88.4</b>	88.4 (99.8%)	1X
Best-first	87.5	2.8 (6.7%)	293.4X
10-best reranking	87.9	87.9 (99.7%)	8.5X
100-best reranking	88.2	88.0 (99.4%)	72.3X
2-best beam search	88.2	85.7 (94.0%)	2.0X
4-best beam search	88.3	88.1 (99.2%)	6.7X
8-best beam search	88.2	86.8 (98.1%)	26.3X

**Table 5:** Comparison of various decoders using the same model from our full system (**Global A\***). We report F1 with and without the backoff model, the percentage of sentences that the decoder can parse, and the time spent decoding relative to A\*.

we obtain  $n$ -best lists from the backoff model and rerank each result with the full model. In the beam search comparison, we use the approach from Clark et al. (2015) which greedily finds the top- $n$  parses for each span in a bottom-up manner. Results indicate that both approximate methods are less accurate and slower than A\*.

## 7 Related Work

Many structured prediction problems are based around dynamic programs, which are incompatible with recursive neural networks because of their real-valued latent variables. Some recent models have neural factors (Durrett and Klein, 2015), but these cannot condition on global parse structure, making them less expressive. Our search explores fewer nodes than dynamic programs, despite an exponentially larger search space, by allowing the recursive neural network to guide the search.

Previous work on structured prediction with recursive or recurrent neural models has used beam search—e.g. in shift reduce parsing (Dyer et al., 2015), string-to-tree transduction (Vinyals et al., 2015), or reranking (Socher et al., 2013)—at the cost of potentially recovering suboptimal solutions. For our model, beam search is both less efficient and less accurate than optimal A\* decoding. In the non-neural setting, Zhang et al. (2014) showed that global features with greedy inference can improve dependency parsing. The CCG beam search parser of Clark et al. (2015), most related to this work, also uses global features. By using neural representations and exact search, we improve over their results.

A\* parsing has been previously proposed for lo-

cally factored models (Klein and Manning, 2003; Pauls and Klein, 2009; Auli and Lopez, 2011; Lewis and Steedman, 2014). We generalize these methods to enable global features. Vaswani and Sagae (2016) apply best-first search to an unlabeled shift-reduce parser. Their use of error states is related to our global model that penalizes local scores. We demonstrated that best-first search is infeasible in our setting, due to the larger search space.

A close integration of learning and decoding has been shown to be beneficial for structured prediction. SEARN (Daumé III et al., 2009) and DAGGER (Ross et al., 2011) learn greedy policies to predict structure by sampling classification examples over actions from single states. We similarly generate classification examples over hyperedges in the agenda, but actions from multiple states compete against each other. Other learning objectives that update parameters based on a beam or agenda of partial structures have also been proposed (Collins and Roark, 2004; Daumé III and Marcu, 2005; Huang et al., 2012; Andor et al., 2016; Wiseman and Rush, 2016), but the impact of search errors is unclear.

## 8 Conclusion

We have shown for the first time that a parsing model with global features can be decoded with optimality guarantees. This enables the use of powerful recursive neural networks for parsing without resorting to approximate decoding methods. Experiments show that this approach is effective for CCG parsing, resulting in a new state-of-the-art parser. In future work, we will apply our approach to other structured prediction tasks, where neural networks—and greedy beam search—have become ubiquitous.

## Acknowledgements

We thank Luheng He, Julian Michael, and Mark Yatskar for valuable discussion, and the anonymous reviewers for feedback and comments.

This work was supported by the NSF (IIS-1252835, IIS-1562364), DARPA under the DEFT program through the AFRL (FA8750-13-2-0019), an Allen Distinguished Investigator Award, and a gift from Google.

## References

- Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2442–2452.
- Michael Auli and Adam Lopez. 2011. Efficient CCG parsing: A\* versus Adaptive Supertagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*.
- Stephen Clark and James R Curran. 2007. Wide-coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4).
- Stephen Clark, Darren Foong, Luana Bulat, and Wenduan Xu. 2015. The Java Version of the C&C Parser: Version 0.95. Technical report, University of Cambridge Computer Laboratory, August.
- Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics.
- Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics.
- Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate Large Margin Methods for Structured Prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 169–176. ACM.
- Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.
- 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*.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation*, 9(8):1735–1780.
- 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).
- Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured Perceptron with Inexact Search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics.
- Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.
- Dan Klein and Christopher D Manning. 2003. A\* Parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*.
- Mike Lewis and Mark Steedman. 2014. A\* CCG Parsing with a Supertag-factored Model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.
- Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint A\* CCG Parsing and Semantic Role Labelling. In *Empirical Methods in Natural Language Processing*.
- Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG Parsing. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics*.
- Adam Pauls and Dan Klein. 2009. K-best A\* Parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 958–966. Association for Computational Linguistics.
- Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 627–635.
- Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with Compositional Vector Grammars. In *Proceedings of the ACL conference*.
- Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved Semantic Representations from Tree-structured Long Short-term Memory Networks. *arXiv preprint arXiv:1503.00075*.
- 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 and Kenji Sagae. 2016. Efficient Structured Inference for Transition-Based Parsing with

- Neural Networks and Error States. *Transactions of the Association for Computational Linguistics*.
- Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging With LSTMs. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics*.
- 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*.
- Sam Wiseman and Alexander M Rush. 2016. Sequence-to-Sequence Learning as Beam-Search Optimization. In *Proceedings of EMNLP*.
- Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG Supertagging with a Recurrent Neural Network. *Volume 2: Short Papers*, page 250.
- Wenduan Xu. 2016. LSTM Shift-Reduce CCG Parsing . In *Empirical Methods in Natural Language Processing*.
- Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2014. Greed is Good if Randomized: New Inference for Dependency Parsing. In *Proceedings of EMNLP*.