Unsupervised Parsing via Constituency Tests

Steven Cao   **Nikita Kitaev**   **Dan Klein**
Computer Science Division
University of California, Berkeley
{stevencao,kitaev,klein}@berkeley.edu

Abstract

We propose a method for unsupervised parsing based on the linguistic notion of a constituency test. One type of constituency test involves modifying the sentence via some transformation (e.g. replacing the span with a pronoun) and then judging the result (e.g. checking if it is grammatical). Motivated by this idea, we design an unsupervised parser by specifying a set of transformations and using an unsupervised neural acceptability model to make grammaticality decisions. To produce a tree given a sentence, we score each span by aggregating its constituency test judgments, and we choose the binary tree with the highest total score. While this approach already achieves performance in the range of current methods, we further improve accuracy by fine-tuning the grammaticality model through a refinement procedure, where we alternate between improving the estimated trees and improving the grammaticality model. The refined model achieves 62.8 F1 on the Penn Treebank test set, an absolute improvement of 7.6 points over the previous best published result.

1 Introduction

When developing a phrase structure grammar for a language, one powerful tool that linguists use is constituency tests. Given a sentence and some span within it, one type of constituency test involves modifying the sentence via some transformation (e.g. replacing the span with a pronoun) and then judging the result (e.g. checking if it is grammatical). If a span passes constituency tests, then linguists have evidence that it is a constituent. Motivated by this idea, as well as recent advancements in neural acceptability (grammaticality) models via pre-training (Warstadt et al., 2018; Devlin et al., 2019; Liu et al., 2019), in this paper we propose a method for unsupervised parsing that operationalizes the way linguists use constituency tests.

Focusing on constituency tests that are judged via grammaticality, we begin by specifying a set of transformations that take as input a span within a sentence and output a new sentence (Section 3). Given these transformations, we then describe how to use a (potentially noisy) grammaticality model for parsing (Section 4). Specifically, we score the likelihood that a span is a constituent by applying the constituency tests and averaging their grammaticality judgments, i.e. the probability that the transformed sentence is grammatical under the model. We then parse via minimum risk decoding, where we score each binary tree by summing the scores of its contained spans, with the interpretation of maximizing the expected number of constituents. Importantly, this scoring system accounts for false positives and negatives because it allows some spans in the tree to have low probability if the model is confident about the rest of the tree.

To learn the grammaticality model, we note that given gold trees, we can train the model to accept constituency test transformations of gold constituents and reject those of gold distituents. On the other hand, given the model parameters, we can estimate trees via the parsing algorithm in Section 4. Therefore, we learn the model via alternating optimization. First, we learn an initial model by fine-tuning BERT on unlabeled data to distinguish between real sentences and distractors produced by random corruptions like shuffling (Section 5). Then, we refine the model by alternating between (1) producing trees, and (2) maximizing/minimizing the scores of predicted constituents/distituents in those trees (Section 6).

To evaluate the effectiveness of our approach, we compare to existing methods for unsupervised parsing (Section 7). Our refined model achieves 62.8 F1 averaged over four random restarts on the Penn Treebank (PTB) test set, an absolute improvement of 7.6 points over the previous best published result,

showing that constituency tests provide powerful inductive bias. Analyzing our parser (Section 8), we find that despite its strong numbers, it makes some mistakes that we might expect from the parser's reliance on this class of constituency tests, like attaching modifying phrases incorrectly. As one possible solution to these shortcomings, we use our method to induce the unsupervised recurrent neural network grammar (URNNG) (Kim et al., 2019b) following the approach in Kim et al. (2019a), where we use our induced trees as supervision to initialize the RNNG model and then perform unsupervised fine tuning via language modeling. The resulting model achieves 67.9 F1 averaged over four random restarts, approaching the supervised binary tree RNNG with a gap of 4.9 points.

2 Related Work

Grammar induction. There has been a long history of research on grammar induction. Here, we touch on just a couple threads of work most related to our method. Early works focused on building probabilistic context-free grammars (PCFGs) but found that inducing them with expectation-maximization (EM) did not produce meaningful trees (Carroll and Charniak, 1992). We highlight some themes since then that have produced successful unsupervised parsers.

Directly modeling spans rather than mediating structure through a grammar: In contrast with previous work based on probabilistic grammars, the constituent-context model of Klein and Manning (2002) proposed a different probabilistic formulation that modeled the constituency of each span directly, where each span yielded words conditioned on whether or not it was a constituent. Parsing then proceeded via minimum risk decoding (Smith and Eisner, 2006), where they chose the tree containing the maximum expected number of constituents.

Explicitly defining criteria for what it means to be a constituent: Rather than designing a generative model over sentences and trees, Clark (2001) proposed that constituents could be identified based on their span statistics, e.g. the mutual information between the left and right contexts of the span.

Finding external signals of constituency: To perform noun compound bracketings ("[*liver cell*] *line*" vs "*liver* [*cell line*]"), Nakov and Hearst (2005) extracted a series of features from Web text, like the frequency of "*liver-cell line*" vs "*liver cell-line*." With a similar idea of extracting signal from

Web text, Spitkovsky et al. (2010) found evidence for constituency from HTML markup, e.g. hyperlinks and italicized phrases.

Designing neural latent variable models: Many works have taken the approach of designing a neural language model with tree-valued latent variables and optimizing it via EM, some of which can also be seen as probabilistic grammars parameterized by neural networks. For example, the compound PCFG (Kim et al., 2019a), found that the original PCFG is sufficient to induce trees if it uses a neural parameterization, and they further enhanced the model via latent sentence vectors to reduce the independence assumptions. Another model, the unsupervised recurrent neural network grammar (URNNG) (Kim et al., 2019b), uses variational inference over latent trees to perform unsupervised optimization of the RNNG (Dyer et al., 2016), an RNN model that defines a joint distribution over sentences and trees via shift and reduce operations. Unlike the PCFG, the URNNG makes no independence assumptions, making it more expressive but also harder to induce from scratch. Shen et al. (2018) proposed the Parsing-Reading-Predict Network (PRPN), where the latent tree structure determines the flow of information in a neural language model, and they found that optimizing for language modeling produced meaningful latent trees. On the other hand, the Deep Inside-Outside Recursive Autoencoder (DIORA) (Drozdov et al., 2019) computes a representation for each node in a tree by recursively combining child representations following the structure of the inside-outside algorithm, and it optimizes an autoencoder objective such that the representation for each leaf in the tree remains unchanged after an inside and outside pass.

Extracting trees from neural language models: The Ordered Neuron (ON) model (Shen et al., 2019) modifies the LSTM to enforce a hierarchy of long- to short-term neurons, with the idea that the forget operation should naturally occur at phrase boundaries. After training on language modeling, they parse by recursively finding splitpoints based on each neuron's decision of where to forget. More recently, Kim et al. (2020) extract trees from pretrained transformers. Using the model's representations for each word in the sentence, they score fenceposts (positions between words) by computing distance between the two adjacent words, and they parse by recursively splitting the tree at the fencepost with the largest distance.

Name	Applied to "A [B] C"	Example
Clefting	it {is, was} B that A C	*it {is, was} the london market that by midday , was in full retreat*
Coordination	A B and B C	*by midday , the london market and the london market was in full retreat*
Substitution	A {it, ones, did so} C	*by midday , {it, ones, did so} was in full retreat*
Front Movement	B , A C	*the london market , by midday , was in full retreat*
End Movement	A C B	*by midday , was in full retreat the london market*

Table 1: The constituency tests we use in this paper, using the span *"by midday , [the london market] was in full retreat"* as an example.

Name	Description
Shuffle	Choose a random subset of words in the sentence and randomly permute them.
Swap	Choose two words and swap them.
Drop	Choose a random subset of words in the sentence and drop them.
Span Drop	Choose a random contiguous span of words and drop it.
Span Movement	Choose a random contiguous span of words and move it to the front or back.
Bigram	Generate a sentence of the same length using a bigram language model trained on the source corpus.

Table 2: The corruptions we use to train the initial grammaticality model using unlabeled data, where the model must determine whether a given sentence is real or corrupted.

Neural grammaticality models. Pre-training has recently produced large gains on a wide range of tasks, including the task of judging whether a sentence is grammatical (Devlin et al., 2019; Liu et al., 2019). Most works evaluate on the Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2018), which compiles acceptable and unacceptable sentences from linguistics publications. The paper also investigates the question of whether grammaticality can be learned from unlabeled data, where fake sentences are generated via either random shuffling or an LSTM language model, and the model must determine whether a given sentence is real or fake. They find that real/fake models perform comparably to supervised models trained on the CoLA training set. Lau et al. (2017) also investigate unsupervised acceptability models, where they instead augment language models with a variety of acceptability measures, e.g. perplexity renormalized to remove the influence of unigram frequency. They find that such models achieve an encouraging level of agreement with crowd-sourced human judgments.

3 Constituency Tests

We begin by specifying a set of constituency tests. The constituency tests we focus on involve transformation functions $c : (\texttt{sent}, i, j) \mapsto \texttt{sent}'$ that take in a span and output a new sentence, and a judgment function $g : \texttt{sent} \mapsto \{0, 1\}$ that judges

the resulting transformed sentence. A span $(\texttt{sent}, i, j)$ passes a constituency test if the judgment function approves of the transformed sentence, or $g(c(\texttt{sent}, i, j)) = 1$. Then, parsing via constituency tests involves specifying a set of transformation functions (this section), learning the judgment function (Sections 5 and 6), and aggregating these test results to produce a tree (Section 4).

We will focus on constituency tests that are judged via grammaticality because it is feasible to learn a grammaticality model using unlabeled data. We describe the set of transformations in Table 1. As future work, modeling semantic preservation could also prove fruitful as a way to correct some false positives, e.g. *"stock [prices rose after the announcement]"* → *"stock it."*

Because we specify constituency tests, while the parser is unsupervised in that it doesn't use labeled data, it is not tabula rasa in that we provide it with linguistically-inspired inductive bias, in contrast with past methods that may have less inductive bias or encode it more implicitly. To induce more and specify less, another interesting line of future work would involve inducing the tests as well.

4 Parsing Algorithm

With this set of transformations, in this section we describe how to parse sentences using a (potentially noisy) grammaticality model. In the supervised setting, Stern et al. (2017) and Kitaev and Klein (2018)

showed that independently scoring each span and then choosing the tree with the best total score produced a very accurate and simple parser, while Klein and Manning (2002) showed a similar result in the unsupervised setting. Therefore, we also use a span-based approach.

We will use $g_\theta : \texttt{sent} \mapsto [0, 1]$ to denote the grammaticality model with parameters θ, which outputs the probability that a given sentence is grammatical. First, we score each span by averaging the grammaticality judgments of its constituency tests, or

$$s_\theta(\texttt{sent}, i, j) = \frac{1}{|C|} \sum_{c \in C} g_\theta(c(\texttt{sent}, i, j)),$$

where C denotes the set of constituency tests. Then, we score each tree by summing the scores of its spans and choose the highest scoring binary tree via CKY, or

$$t^*(\texttt{sent}) = \underset{t \in T(\texttt{len(sent)})}{\arg\max} \sum_{(i,j) \in t} s_\theta(\texttt{sent}, i, j),$$

where $T(\texttt{len(sent)})$ denotes the set of binary trees with $\texttt{len(sent)}$ leaves. If we interpret the score $s_\theta(\texttt{sent}, i, j)$ as estimating the probability that the span $(\texttt{sent}, i, j)$ is a constituent, then this formulation corresponds to choosing the tree with the highest expected number of constituents, i.e. minimum risk decoding (Smith and Eisner, 2006). This scoring system accounts for noisy judgments, which lead to false positives and negatives, by allowing some spans to have low probability if the model is confident about the rest of the tree.

If we want $s_\theta(\texttt{sent}, i, j)$ to estimate the posterior probability that the span is a constituent given the judgments of its constituency tests, or

$$\mathbf{P}((\texttt{sent}, i, j) \text{ is a constituent } | \\ \{g_\theta(c(\texttt{sent}, i, j)) : c \in C\}),$$

then we might want to do something more sophisticated than taking the average. However, we find that the average performs well while being both parameter-less and simple to interpret, so we leave this avenue of exploration to future work.

5 Initializing the Grammaticality Model

In this section and the next, we describe how we learn the grammaticality model. Given gold trees, we can train the model to accept constituency test transformations of gold constituents and reject those of gold distituents. On the other hand, given model parameters, we can estimate trees using the parsing algorithm in Section 4. Therefore, we first initialize the model (this section), and we then refine it via alternating optimization (Section 6).

Previously, Warstadt et al. (2018) found that LSTM grammaticality models trained with supervision versus those trained on a real/fake task achieved similar correlation with human judgments when evaluating on the Corpus of Linguistic Acceptability (CoLA), a dataset with examples of acceptable and unacceptable taken from linguistic publications. Given an unlabeled corpus of sentences and a set of corruptions, the real/fake task involves predicting whether a given sentence is real or corrupted. Motivated by their result, we train our model via a real/fake task but a wider range of corruptions, as described in Table 2.

Rather than training from scratch, we fine-tune the RoBERTa model (Liu et al., 2019), a BERT variant pre-trained on masked word prediction and next sentence prediction. As our unlabeled sentences, we use 5 million sentences from English Gigaword (Graff and Cieri, 2003), and we do not perform any early stopping. We report optimization hyperparameters in the appendix.

Comparing the real/fake RoBERTa model to a supervised version, we find that the former achieves 0.21 MCC (Matthews Correlation Coefficient) on the CoLA development set, while the latter achieves 0.73 MCC, in contrast with the finding in Warstadt et al. (2018) that real/fake and supervised LSTMs achieved similar accuracy (both around 0.2 to 0.3 MCC).[1] This gap is not totally surprising given how high the supervised RoBERTa numbers are. However, when used for parsing via constituency tests, the real/fake RoBERTa model outperforms the supervised model by about 6 F1 (before refinement), likely because invalid constituency tests look more like random corruptions than examples from the CoLA training set, which are taken from linguistics publications.

6 Refining the Grammaticality Model

While the unrefined grammaticality model achieves 48.2 F1, which is in the range of current methods (Table 3), we further improve accuracy via alternating optimization, which proceeds as follows:

1. Using the span-based algorithm in Section 4, parse a batch B of sentences to produce trees.

---

[1]We did not optimize the corruption set for CoLA MCC.

2. Use these trees as pseudo-gold labels to update the span judgments. Specifically, for each sentence, minimize the loss function

$$\sum_{(i,j)\in t^*(\texttt{sent})} \log(s_\theta(\texttt{sent},i,j))$$
$$+ \sum_{(i,j)\notin t^*(\texttt{sent})} \log(1-s_\theta(\texttt{sent},i,j)),$$

i.e. binary cross-entropy on each span with inclusion into the predicted tree as the label, summed over the sentences in the batch.

Note that the span scores $s_\theta(\texttt{sent},i,j)$ are derived from grammaticality judgments of constituency tests, so the only parameters are those in the grammaticality model. Therefore, this step can be thought of as increasing the grammaticality judgment of every constituency test applied to every predicted constituent, while decreasing the judgments for predicted distituents.

3. Repeat for the next batch of sentences.

This step can be thought of as encouraging self-consistency between the model's grammaticality judgments and the trees that result from them. For example, CKY might choose a tree where a few of the spans are considered invalid if the model is confident about the other spans in the tree. The refinement procedure would then increase the probability of these initially invalid spans, which might help the model catch spans that it initially missed. We see evidence of this effect in Section 8. In addition, there is an inherent mismatch between the real/fake task that the model was trained on and the constituency test judgment task it is being used for. For example, many of the sentences resulting from constituency tests are far out of distribution from sentences seen during training. Therefore, this step can also be thought of as helping the grammar model adapt to its new setting.

One problem, however, is that the loss function takes a gradient through the grammaticality judgments of all of the constituency tests for every span in the sentence. This computation takes up too much memory, given that a length-30 sentence has about 400 spans and thus about 3000 constituency tests. Therefore, to reduce memory usage, for every sentence we only take the gradient through 16 of the constituency tests, chosen randomly.

While early stopping would likely improve performance, we instead perform refinement for a

Model	PTB F1	
	Mean	Max
PRPN[†] (Shen et al., 2018)	37.4	38.1
URNNG (Kim et al., 2019b)	–	45.4
ON[†] (Shen et al., 2019)	47.7	49.4
Neural PCFG[†] (Kim et al., 2019a)	50.8	52.6
DIORA (Drozdov et al., 2019)	–	58.9
Compound PCFG[†] (Kim et al., 2019a)	55.2	60.1
Left Branching	8.7	
Balanced	18.5	
Right Branching	39.5	
Ours (before refinement)	48.2	
Ours (after refinement)	**62.8**	**65.9**
Oracle Binary Trees	84.3	

Table 3: Unlabeled sentence-level F1 on the PTB test set without punctuation or unary chains. "Before refinement" denotes the parser using the acceptability model after real/fake training, which we only run once. Starting from this initial model, we report the mean and maximum score out of 4 random restarts of refinement. Baseline numbers are taken from Kim et al. (2019a). After refinement, the parser outperforms the previous best method by 7.6 points.
† denotes models trained without punctuation.

fixed number of iterations because we don't have access to labeled data. Specifically, we perform refinement for one epoch on 5000 sentences from the PTB training set (sections 2 to 21), combined with the 2416 sentences in the PTB test set (section 23). We find that the training curve is relatively consistent across runs. We use the same optimization parameters as the ones for the real/fake task, as described in the Appendix.

7 Results

7.1 F1 on the Penn Treebank

For evaluation, we report the F1 score with respect to gold trees in the Penn Treebank test set (section 23). Following prior work (Kim et al., 2019a; Shen et al., 2018, 2019), we strip punctuation and collapse unary chains before evaluation, and we calculate F1 ignoring trivial spans. The averaging is sentence-level rather than span-level, meaning that we compute F1 for each sentence and then average over all sentences. Because most unsupervised parsing methods only consider fully binary trees, we include the oracle binary tree ceiling, produced by taking the (often flat) gold trees and binarizing them arbitrarily.

Model	PTB F1	
	Initial (Max)	+URNNG
PRPN	47.9	51.6
ON	50.0	55.1
Neural PCFG	52.6	58.7
Compound PCFG	60.1	66.9
Ours (after refinement)	65.9	**71.3**
Supervised Binary RNNG	71.9	72.8

Table 4: Unlabeled sentence-level F1 on the PTB test set without punctuation or unary chains. Following the experimental setup in Kim et al. (2019a), "Initial (Max)" denotes the induced trees resulting from running the method four times and selecting the best result. Next, we use the induced trees as supervision for RNNG and then run unsupervised RNNG fine-tuning, denoted by the "+URNNG" column. "Supervised Binary RNNG" denotes training the RNNG on binarized gold trees. Baseline numbers are taken from Kim et al. (2019a). When selecting the best parser out of four runs, our method combined with URNNG approaches the supervised binary RNNG, with a gap of 1.5 points. Departing from the setup of Kim et al. (2019a), we also induced URNNG three more times using the other three runs, which resulted in a mean score of 67.9 across the four runs and a minimum of 61.1.

Table 3 displays the resulting F1 numbers for our method compared to existing unsupervised parsers, where we report mean, maximum, and minimum out of four random restarts. Before refinement, at 48.2 F1, the parser is already in the range of existing methods. After refinement, the parser achieves 62.8 F1 averaged over four runs, outperforming the previous best result by 7.6 points.[2]

7.2 Inducing URNNG

Kim et al. (2019a) found that while URNNG (described in Section 2) fails to outperform right-branching trees on average when trained from scratch, it achieves very good performance when initialized using another method's induced trees. Specifically, they first train RNNG using the induced trees from another method as supervision. Then, they perform unsupervised fine-tuning with a language modeling objective. They find that this procedure produces substantial gains when combined with existing unsupervised parsers.

Following their experimental setup, we use our

---

[2]While other methods do not report the minimum, our minimum score was 60.4 F1. We also evaluate in the setting where the test set sentences are not available during refinement, and we find similar results (mean: 62.8, max: 64.6, min: 61.5).

	Before refinement	After (best parser)	Best parser + URNNG
SBAR	0.229	0.661	**0.853**
NP	0.604	0.794	**0.843**
VP	0.325	0.682	**0.808**
PP	0.571	**0.862**	0.844
ADJP	**0.664**	0.626	0.556
ADVP	0.620	**0.639**	0.546
F1	48.2	65.9	**71.3**

Table 5: Recall by label, or the fraction of gold constituents predicted to be constituents by each model, along with F1 (calculated over all spans). We report numbers for the parser before refinement, the best parser out of four runs of refinement, and URNNG induced from the best parser. Refinement and URNNG both produce large improvements for all categories except ADJPs and ADVPs.

best parser out of four runs to parse both the PTB training set and test set, and we induce URNNG using these predicted trees. We use the default parameters in the Kim et al. (2019b) github, which we report in the Appendix. Table 4 shows the resulting F1 on the PTB test set. After URNNG, we achieve 71.3 F1, approaching the performance of the supervised binary RNNG + URNNG with a gap of 1.5 points. However, selecting the best parser out of four requires labeled data, so we also induce URNNG from each of the three other parsers. We find that the mean score across the four runs is 67.9. To close the gap between the max and mean across the four runs, ensembling might be an effective approach; we leave this direction to future work.

One possible reason for why URNNG helps is that the URNNG model makes no independence assumptions, making it very expressive but also also difficult to induce from scratch. Therefore, we can think of this method as removing some of the independence assumptions and other biases of the original model once they have sufficiently guided the unsupervised training.

8 Analysis

8.1 Recall by Label

First, we compute recall by label for the parser before refinement, after refinement, and after refinement + URNNG, displayed in Table 5. Before refinement, the parser is strongest in ADJPs and ADVPs and weakest for VPs and SBARs. Refine-
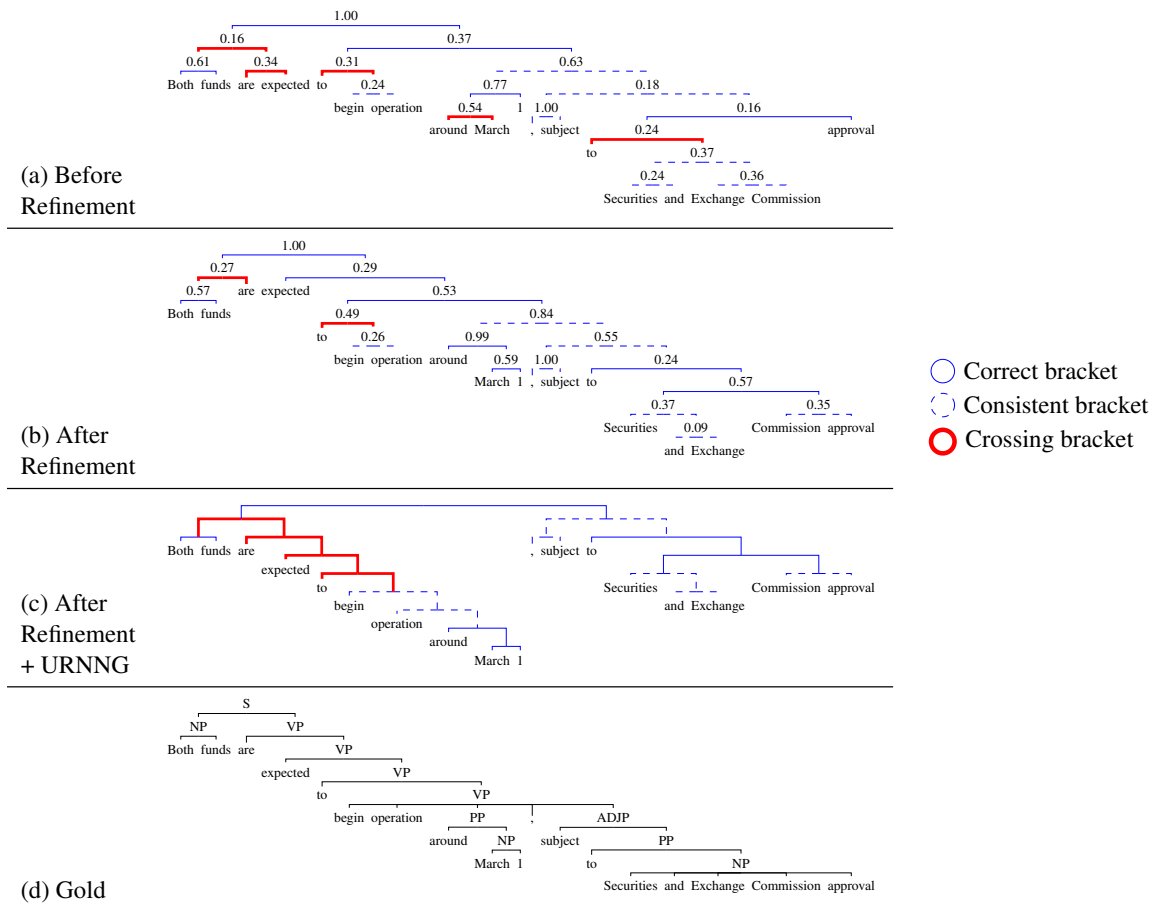
Figure 1: Example trees (a) before refinement, (b) after refinement, (c) after refinement + URNNG, and (d) gold, where we use the first PTB train sentence whose F1 was within 1 of the average. Each non-trivial span is labeled with its score under the model, i.e. the average grammaticality of its constituency tests. Each span is labeled blue if it is present in the gold, dashed blue if it is consistent (ignoring punctuation), and thick red if it is crossing. After refinement (tree b), the parser makes two mistakes: attaching "are" to the subject, and attaching the phrase "around March ... Commission approval" one level too high. After refinement + URNNG (tree c), the only mistake is attaching the phrase "subject to ... Commission approval" at the top level, which produces four crossing brackets.

ment causes all categories except ADJP and ADVP to receive a boost of about 0.3 in recall. Afterward, URNNG produces a boost for SBAR and VP, resulting in the four categories being above 0.8, except with ADJP and ADVP still both around 0.55. In Section 8.3, we analyze the sources of these mistakes in more detail and find that the model is less effective in identifying ADJPs that serve as NP adjuncts (e.g. "[*most recent*] *news*").

8.2 Analyzing the Constituency Tests

To better understand how well each category is covered by constituency tests, in Table 6 we display the recall per phrase type for each test, along with F1 computed over all spans. Using each test, we judge each span in the PTB development set individually by thresholding the grammaticality judgment at 0.5, and for each phrase type we report the frac-

tion that pass the test. Before refinement, the tests behave roughly as expected. Coordination fires consistently for all phrase types but also half the distituents, while the NP and VP proforms fire for NPs and VPs respectively. Clefting and movement are more mixed, with clefting sometimes firing for all phrase types except VP, and movement sometimes firing for SBARs, PPs, and ADVPs. Interestingly, the individual F1 numbers are all quite low at around 10-20 F1, even though the parser achieves 48.1 F1, suggesting that the constraint of outputting a well-formed tree provides substantial information. After refinement, all of the tests have better F1, potentially because refinement allows the grammar model to use the well-formedness constraint to improve its span judgments (see Section 6). In particular, we find that coordination no longer has false positives, and clefting exhibits

	Clefting		Proform Substitution			Movement		Coord-
	is	was	ones	did so	it	front	end	ination
Before Refinement								
SBAR	0.294	0.260	0.113	0.130	0.146	0.279	0.319	0.942
NP	0.353	0.347	0.458	0.178	0.555	0.055	0.048	0.934
VP	0.091	0.089	0.067	0.479	0.127	0.060	0.144	0.944
PP	0.427	0.412	0.238	0.165	0.154	0.308	0.606	0.906
ADJP	0.383	0.361	0.286	0.241	0.346	0.127	0.172	0.911
ADVP	0.396	0.395	0.143	0.185	0.198	0.290	0.307	0.893
Distituent	0.066	0.063	0.184	0.098	0.123	0.033	0.052	0.456
F1	20.8	20.9	10.9	13.2	17.8	12.3	13.6	16.1
After Refinement								
SBAR	0.237	0.223	0.237	0.770	0.374	0.250	0.225	0.539
NP	0.718	0.712	0.571	0.428	0.539	0.063	0.035	0.792
VP	0.105	0.118	0.171	0.707	0.359	0.108	0.083	0.601
PP	0.744	0.741	0.202	0.730	0.332	0.354	0.531	0.707
ADJP	0.543	0.556	0.219	0.324	0.263	0.217	0.108	0.686
ADVP	0.565	0.582	0.187	0.627	0.338	0.353	0.292	0.655
Distituent	0.031	0.032	0.052	0.060	0.045	0.012	0.026	0.086
F1	51.1	50.9	29.5	38.6	37.4	18.6	15.0	43.1

Table 6: For each constituency test and each phrase type XP, we report the fraction of XPs in the PTB development set that pass the constituency test, where we judge each span individually and threshold the grammaticality judgment at 0.5. We also report F1 (calculated over all spans). Before refinement, coordination consistently fires for all categories but also for almost half of the distituents. The other tests behave roughly as expected; for example, the NP proforms ("ones" and "it") fire for NPs, while the VP proform ("did so") fires for VPs. After refinement, coordination no longer fires for distituents, and all of the tests have higher F1. In addition, the proforms now fire for a much wider range of phrase types. See the appendix for a grayscale version.

greatly improved recall. We also see that the proform substitution tests now fire for a wider range of phrase types; for example, "did so" now fires for 70% of SBARs, VPs, PPs, and ADVPs, even though it was originally a VP substitution.

8.3 Common Mistakes

In Table 7, we show the most common crossing brackets predicted by the parser, where for analysis we categorize the brackets by part-of-speech. We find that the model after refinement commonly makes the following mistakes, and we suggest possible explanations for each:

1. Bracketing the verb with the subject:

   [*they 're*] *squaring off*

   As shown in Table 6, there is less support for VPs via consituency tests. This observation is also reflected in the example trees in Figure 1, where the VPs have consistently lower scores. Therefore, while the parser usually chooses to bracket VPs (achieving 0.682 recall, as shown

Common mistakes after refinement			
Parts of speech	Example	Percentage of mistakes	Δ in # mistakes after URNNG
PRP VBD/P/Z	[*they 're*] *squaring off*	1.72%	-81.0%
IN NN(S)	[*in letters*] *to the agency*	1.07%	-57.6%
CD NN(S)	*about* [*1,200 cars*]	1.06%	+4.4%
IN DT NN(S)	[*in an effort*] *to streamline*	0.99%	-74.7%
TO VB	[*to work*] *a lot*	0.93%	-95.0%

Common mistakes after refinement + URNNG			
Parts of speech	Example	Percentage of mistakes	Δ in # mistakes after URNNG
CD NN(S)	*about* [*1,200 cars*]	1.51%	+4.4%
JJ NN(S)	*socially* [*responsible companies*]	0.69%	+47.0%
IN NN(S)	[*in letters*] *to the agency*	0.61%	-57.6%
NN(S) IN NN(S)	[*plenty of reasons*] *to stay*	0.57%	-27.3%
NNP VBD/P/Z	*Mr.* [*Lane said*]	0.47%	-21.2%

Table 7: The five most common crossing brackets categorized by part-of-speech, computed on the first 5,000 sentences in the PTB training set. We also report percentage of crossing predicted brackets (i.e. mistakes) that fall under that category, as well as the change in the number of mistakes after adding URNNG. We group (VBD, VBP, VBZ) (past, present, present 3rd-person) and (NN, NNS) (noun, noun plural). We find that the model commonly makes the following mistakes: (1) bracketing the verb with the subject, (2) in a nested PP, attaching the inner PP outside, (3) grouping the cardinal or adjective with the noun instead of with its adverb, and (4) bracketing "to + infinitive." After URNNG, each of the mistakes are corrected except (3).

in Table 5), there seem to be cases in which it prefers the [subject verb] bracketing.

2. In a nested PP, attaching the inner PP outside the outer PP:

   [*in letters*] *to the agency*

   The spans resulting from incorrect attachments still tend to produce grammatical constituency tests (e.g. *"they argue* [*in letters*] *to the agency that ..."* → *"in letters , they argue to the agency that ..."*).

3. Grouping cardinals and adjectives with the noun, instead of with the adverb:

   about [*1,200 cars*]

   This span passes some constituency tests, like *"about {it, ones},"* while none of the tests except coordination accept *"about 1,200."*

4. Bracketing "to + infinitive":

   they want [*to work*] *a lot*

   Infinitive VPs (e.g. *"work a lot"*) typically don't pass any of our tests except coordination, while "to + infinitive" is often replaceable by a noun proform, like *"they want it a lot."*

After URNNG, the VP errors (1 and 4) are corrected almost completely, while the PP attachment error also decreases in frequency by about half. In contrast, the ADJP error (3) is exacerbated, with [CD NN] and [JJ NN] incorrect bracketings increasing by 4.4% and 47.0% (Table 7). Therefore, URNNG is effective in correcting many but not all of the parser's systematic errors, suggesting paths for future improvement, e.g. by adding tests that fire for currently missing brackets.

8.4 Example Trees

Finally, to qualitatively understand the parser's performance, in Figure 1 we display the trees before refinement, after refinement, and after refinement + URNNG for the sentence *"Both funds are expected to begin operation around March 1 , subject to Securities and Exchange Commission approval."* To produce a representative example, we selected this sentence by choosing the first sentence in PTB train whose F1 was within 1 of the average. Comparing the trees before and after refinement, the parser corrects two mistakes, *"[around March] 1"* and *"[to Securities and Exchange Commission] approval,"* which both involve bracketing the preposition with part of its NP complement. As a result, ignoring punctuation and binarization, the parser after refinement makes only two mistakes: attaching *"are"* to the subject, and attaching the phrases *"around March"* and *"subject to ... Commission approval"* one level too high. After URNNG, the first mistake is corrected, such that the only mistake is in the attachment of *"subject to ... Commission approval"* (but because it attaches this phrase very high, this mistake produces four crossing brackets). This example provides some characterization of each step's improvement to the predicted trees.

9 Conclusion

In this paper, we showed that using constituency tests to parse sentences is an effective approach, achieving strong performance for unsupervised parsing. Furthermore, we used the interpretability of constituency tests to highlight and explain the parser's strengths and shortcomings, like the "[subject verb]" and "adverb [adjective noun]" misbracketings, revealing potential next steps for improvement. Therefore, we see parsing via constituency tests as a promising new approach with both strong results and many open questions.

References

Glenn Carroll and Eugene Charniak. 1992. Two experiments on learning probabilistic dependency grammars from corpora. In *AAAI Workshop on Statistically-Based NLP Techniques*.

Alexander Clark. 2001. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the ACL 2001 Workshop on Computational Natural Language Learning (ConLL)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141, Minneapolis, Minnesota. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

David Graff and Christopher Cieri. 2003. English gigaword LDC2003T05. *Linguistic Data Consortium*.

Taeuk Kim, Jihun Choi, Daniel Edmiston, and Sanggoo Lee. 2020. Are pre-trained language models aware of phrases? simple but strong baselines for grammar induction. In *Proceedings of the International Conference for Learning Representations*.

Yoon Kim, Chris Dyer, and Alexander Rush. 2019a. Compound probabilistic context-free grammars for

grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy. Association for Computational Linguistics.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019b. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1105–1117, Minneapolis, Minnesota. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations*.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Jey Han Lau, Alexander Clark, and Shalom Lappin. 2017. Grammaticality, acceptability, and probability: A probabilistic view of linguistic knowledge. *Cognitive Science*, 41(5):1202–1241.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

Preslav Nakov and Marti Hearst. 2005. Search engine statistics beyond the n-gram: Application to noun compound bracketing. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 17–24, Ann Arbor, Michigan. Association for Computational Linguistics.

Yikang Shen, Zhuohan Lin, Chin-Wei Huang, and Aaron Courville. 2018. Neural language modeling by jointly learning syntax and lexicon. In *Proceedings of the International Conference for Learning Representations*.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2019. Ordered neurons: Integrating tree structures into recurrent neural networks. In *Proceedings of the International Conference for Learning Representations*.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794, Sydney, Australia. Association for Computational Linguistics.

Valentin I. Spitkovsky, Daniel Jurafsky, and Hiyan Alshawi. 2010. Profiting from mark-up: Hyper-text annotations for guided parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1278–1287, Uppsala, Sweden. Association for Computational Linguistics.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Zhiyang Teng and Yue Zhang. 2018. Two local models for neural constituent parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 119–132, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

A Appendix

A.1 Optimization Hyperparameters and Other Training Details

For both real/fake training and refinement, we use a learning rate of 3×10^{-5} with Adam (Kingma and Ba, 2015) hyperparameters $\beta = (0.9, 0.999)$, $\epsilon = 10^{-6}$ and linear learning rate warmup for the first 10% of the training data. For real/fake training, each batch contains 32 real and 32 fake sentences, while for refinement we parse a batch of 32 sentences for each gradient step. We did not perform any hyperparameter search.

We fine-tuned the RoBERTa base model, which has 125M parameters, and we performed classification for sentences by applying a linear layer and softmax to the `[CLS]` embedding.

For real/fake training, we used a single Nvidia K80 with 12GB RAM, which took about 3 days to run for 5 million sentences. For refinement, we either used a single Quadro 8000 with 48GB RAM, which took about 1 day to run, or a single Nvidia K80, which took about 6 days to run.

For URNNG, we used the default hyperparameters in the Kim et al. (2019b) github. Specifically, we used a batch size of 16, and we performed 18 epochs of supervised RNNG training with a learning rate of 0.0001, and 10 epochs of unsupervised fine-tuning with a learning rate of 0.1. Other optimization details can be found in the original paper (Kim et al., 2019b). We used a single Quadro 6000 with 24GB RAM, which took about 3 days.

As our data, we used the first 5M sentences from the English Gigaword corpus (Graff and Cieri, 2003) for real/fake training, and we used the standard train/development/test splits (sections 02-21, 22, 23) of the Penn Treebank for parsing (Marcus et al., 1993), which have 39832, 1700, and 2416 examples, respectively. Both datasets are already tokenized. For preprocessing, we converted all letters to lowercase and removed quotation marks and any ending punctuation.

A.2 Some Ablations of the Refinement Procedure

Having analyzed the output of our parser, next we describe some ablations to determine how much of the performance is due to constituency tests versus the refinement procedure.

First, if we ablate the refinement procedure (Table 3), the initial parser still performs quite well – it is much better than right-branching and relatively

	Clefting		Proform Substitution			Movement		Coord-
	is	was	ones	did so	it	front	end	ination
Before Refinement								
SBAR	0.294	0.260	0.113	0.130	0.146	0.279	0.319	0.942
NP	0.353	0.347	0.458	0.178	0.555	0.055	0.048	0.934
VP	0.091	0.089	0.067	0.479	0.127	0.060	0.144	0.944
PP	0.427	0.412	0.238	0.165	0.154	0.308	0.606	0.906
ADJP	0.383	0.361	0.286	0.241	0.346	0.127	0.172	0.911
ADVP	0.396	0.395	0.143	0.185	0.198	0.290	0.307	0.893
Distituent	0.066	0.063	0.184	0.098	0.123	0.033	0.052	0.456
F1	20.8	20.9	10.9	13.2	17.8	12.3	13.6	16.1
After Refinement								
SBAR	0.237	0.223	0.237	0.770	0.374	0.250	0.225	0.539
NP	0.718	0.712	0.571	0.428	0.539	0.063	0.035	0.792
VP	0.105	0.118	0.171	0.707	0.359	0.108	0.083	0.601
PP	0.744	0.741	0.202	0.730	0.332	0.354	0.531	0.707
ADJP	0.543	0.556	0.219	0.324	0.263	0.217	0.108	0.686
ADVP	0.565	0.582	0.187	0.627	0.338	0.353	0.292	0.655
Distituent	0.031	0.032	0.052	0.060	0.045	0.012	0.026	0.086
F1	51.1	50.9	29.5	38.6	37.4	18.6	15.0	43.1

Table 8: A grayscale version of Table 6, where higher numbers are shaded with darker shades of gray.

close in performance to current methods. We can also try ablating the constituency tests. Specifically, following the suggestion of an anonymous reviewer, we randomly initialized a Roberta-based span classification parser and performed refinement of the span scores (Section 6). The resulting parser did not achieve very high accuracy (initial F1: 11.95, final F1: 12.33; F1 is computed including punctuation). These ablations suggest that constituency tests are the main driving force behind our method. We discuss a few possible reasons below.

First, because the refinement method has the effect of enforcing self-consistency, the initialization is important, and constituency tests are important for the initialization.

Next, the refinement procedure itself also relies heavily on constituency tests because the gradient step involves maximizing the grammaticality of constituency tests for spans within the imputed trees. In particular, all span judgments originate from grammaticality judgments, and the only parameters are those in the grammaticality model. Therefore, the procedure exploits the fact that grammaticality and constituency are linked.