Neural Generative Rhetorical Structure Parsing

Amandla Mabona
Dept. of Computer Science and Technology
University of Cambridge
amandla.mabona@cl.cam.ac.uk

Laura Rimell
DeepMind
London, UK
laurarimell@google.com

Stephen Clark
DeepMind
London, UK
clarkstephen@google.com

Andreas Vlachos
Dept. of Computer Science and Technology
University of Cambridge
andreas.vlachos@cst.cam.ac.uk

Abstract

Rhetorical structure trees have been shown to be useful for several document-level tasks including summarization and document classification. Previous approaches to RST parsing have used discriminative models; however, these are less sample efficient than generative models, and RST parsing datasets are typically small. In this paper, we present the first generative model for RST parsing. Our model is a document-level RNN grammar (RNNG) with a bottom-up traversal order. We show that, for our parser's traversal order, previous beam search algorithms for RNNGs have a left-branching bias which is ill-suited for RST parsing. We develop a novel beam search algorithm that keeps track of both structure- and word-generating actions without exhibiting this branching bias and results in absolute improvements of 6.8 and 2.9 on unlabelled and labelled F1 over previous algorithms. Overall, our generative model outperforms a discriminative model with the same features by 2.6 F1 points and achieves performance comparable to the state-of-the-art, outperforming all published parsers from a recent replication study that do not use additional training data.

1 Introduction

Understanding a document's discourse-level organization is important for correctly interpreting it, and discourse analyses have been shown to be helpful for several NLP tasks (Bhatia et al., 2015; Ji and Smith, 2017; Feng and Hirst, 2014b; Ferracane et al., 2017). A popular formalism for discourse analysis is Rhetorical Structure Theory (RST) (Mann and Thompson, 1988) (Fig. 1) which represents a document as a tree of discourse units recursively built by connecting smaller units through rhetorical relations. Learning to predict RST trees is difficult because it depends on pragmatics as well as literal meaning, and the English RST Discourse Treebank (RST-DT) (Carlson et al., 2003) is small by the standards of modern parsing datasets, with 347 training documents.

Previous approaches to RST parsing (Ji and Eisenstein, 2014; Feng and Hirst, 2014a; Joty et al., 2015; Braud et al., 2017) have used locally normalized discriminative models. However, these are known to have worse performance than generative models when there is little training data (Ng and Jordan, 2002; Yogatama et al., 2017).

Unlike locally normalised discriminative models, generative models are not susceptible to label bias (Lafferty et al., 2001). The success of generative (Dyer et al., 2016; Charniak et al., 2016) and globally normalised (Andor et al., 2016) syntactic parsers suggests that reducing label bias leads to better performance. We hypothesize that using a generative parser would also lead to improved performance on RST parsing. However, while they are free from label bias, generative parsers require more sophisticated search algorithms for decoding. Fried et al. (2017) presented a word-level beam search algorithm that made it possible to decode directly from neural generative parsers rather than using them as rerankers.

In this paper, we present the first generative RST parser[1]. Our model is a document-level version of an RNN Grammar (RNNG, Dyer et al. (2016)) defined through a transition system with both word- and structure-generating actions. It uses distributed representations of discourse units and transition probabilities parametrized by RNNs to model unbounded dependencies in a document.

For our discourse parser, we find that Fried et al. (2017)'s word-level beam search algorithm is bi-

[1] Ji et al. (2016) introduced a neural generative discourse parser, but they used the annotation scheme of the Penn Discourse Treebank (Prasad et al., 2008) and Switchboard Dialog Act (Godfrey et al., 1992) corpora, predicting flat discourse representations between adjacent sentences, rather than hierarchical relations among clauses.

*Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing
and the 9th International Joint Conference on Natural Language Processing*, pages 2284–2295,
Hong Kong, China, November 3–7, 2019. ©2019 Association for Computational Linguistics

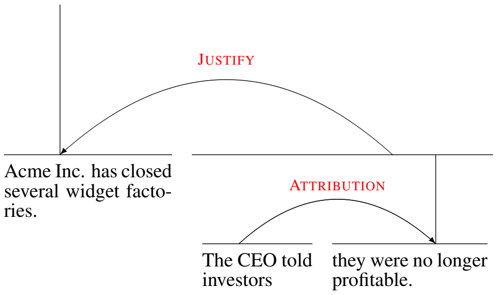
JUSTIFY
Acme Inc. has closed several widget factories.
ATTRIBUTION
The CEO told investors
they were no longer profitable.

Figure 1: An example of an RST tree.

ased towards producing left-branching trees. We analyse the source of this bias and develop a novel beam search algorithm that removes it by tracking both word- and structure-generating actions. On the RST-DT development set, our algorithm leads to improvements of 6.8% and 2.9% on unlabelled and labelled attachment accuracies when decoding from the same parser, compared to word-level beam search. On the RST-DT test set, our generative parser outperforms a discriminative version with the same features by 2.6% on labelled attachment accuracy. Overall, our parser obtains a labelled attachment score of 45.0%, outperforming all published parsers in a recent replication study that do not use additional training data.

2 Rhetorical Structure Theory

Rhetorical Structure Theory describes the structure of a document in terms of text spans that form discourse units and the relations between them. The basic unit of analysis is an *elementary discourse unit* (EDU) which can be assumed to be a syntactic clause. A *unit* is made up of two or more adjacent discourse units (which can be EDUs or other units) that are in some *rhetorical relation*.

Most rhetorical relations are binary and asymmetric with one argument, the *nucleus*, being more important than the other, the *satellite*. *Importance* is defined through a deletion test: a text becomes incoherent if a nucleus is deleted, but not if a satellite is. These binary asymmetric relations are called *mononuclear* relations. The remaining relations are symmetric, having two or more arguments of equal importance, and are called *multinuclear* relations.

An *RST tree* or *analysis* is a nested collection of discourse units that are either EDUs or units, where the top unit spans the whole text (Mann and Thompson, 1988). RST parsing is the task of automatically predicting RST trees for documents.

3 Rhetorical Structure RNNGs

In this section, we present a generative model for predicting RST trees given a document segmented into a sequence of EDUs $e_{1:m}$[2]. The model is a document-level RNNG in bottom-up traversal order (Kuncoro et al., 2018). We first describe syntactic RNNGs in section 3.1. We then describe our parser's transition system in section 3.2, and its transition model in section 3.3.

3.1 Recurrent Neural Network Grammars

Recurrent neural network grammars are a class of syntactic language models that define a joint probability distribution $p(\boldsymbol{x}, \boldsymbol{y})$ over sentences and their phrase structure trees. An RNNG is defined by a triple (N, Σ, Θ) with N a finite set of nonterminal symbols, Σ a finite set of terminal symbols and Θ neural network parameters.

RNNGs generate sentences and their parse trees through actions[3] in an abstract state machine. A machine state is a tuple $\langle S, B \rangle$ where S is a stack which holds partial phrase structure trees and B is a buffer which holds sentence prefixes. The transitions push new subtrees onto the stack, combine subtrees already there, and append terminals to the buffer until the stack contains a single phrase structure tree and the buffer contains a complete sentence. The original presentation in Dyer et al. (2016) used the following transition system:

NT(X) Push the nonterminal node $(X$ onto the top of the stack, where $X \in N$.

GEN(w) Push the terminal symbol $w \in \Sigma$ onto the top of the stack and the end of the buffer.

REDUCE Pop subtrees $\tau_1, \cdots, \tau_l$ from the top of the stack until the first nonterminal node $(X$ is reached and push the subtree $(X \tau_1 \cdots \tau_l)$ onto the top of the stack.

A sentence $\boldsymbol{x}$ and phrase structure tree $\boldsymbol{y}$ are generated by a unique sequence of actions $a_{1:k}$. The joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$ is defined as the probability of the action sequence $a_{1:k}$:

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{j=1}^{k} p(a_j | a_{<j}) = \prod_{j=1}^{k} p(a_j | S_j, B_j) \quad (1)$$

[2] As in most previous work on RST parsing, we use gold EDU segmentations in our experiments, but our parser would use the output of an EDU segmenter in practice.

[3] We use "action" and "transition" interchangeably.

Action	Before	After	Probability	Condition
$\mathsf{GEN}(e)$	$\langle S, B\rangle$	$\langle S \mid \mathsf{EDU}(e), B \mid e\rangle$	$p_{trans}(\mathsf{GEN} \mid S) \cdot p_{gen}(e \mid S)$	$\lvert B\rvert < m$
$\mathsf{RE}(r, n)$	$\langle S \mid U_L \mid U_R, B\rangle$	$\langle S \mid (\mathsf{Unit}(r, n)\ U_L\ U_R), B\rangle$	$p_{trans}(\mathsf{RE}(r, n) \mid S)$	$\lvert S\rvert \geq 2$

Table 1: Our transition system. $|S|$ is the number of discourse units on the stack, $|B|$ is the number of EDUs in the buffer and m is the number of EDUs in the whole document, r is a relation label and n is a nuclearity label.

Stack	Buffer	Prediction
ϵ	ϵ	$\mathsf{GEN}(e_1)$
$\mathsf{EDU}(e_1)$	e_1	$\mathsf{GEN}(e_2)$
$\mathsf{EDU}(e_1) \mid \mathsf{EDU}(e_2)$	$e_1 \mid e_2$	$\mathsf{GEN}(e_3)$
$\mathsf{EDU}(e_1) \mid \mathsf{EDU}(e_2) \mid \mathsf{EDU}(e_3)$	$e_1 \mid e_2 \mid e_3$	$\mathsf{RE}(\mathit{ATTR}, \mathit{SN})$
$\mathsf{EDU}(e_1) \mid (\mathsf{Unit}(\mathit{ATTR}, \mathit{SN})\ \mathsf{EDU}(e_2)\ \mathsf{EDU}(e_3))$	$e_1 \mid e_2 \mid e_3$	$\mathsf{RE}(\mathit{JUST}, \mathit{NS})$
$\big(\mathsf{Unit}(\mathit{JUST}, \mathit{NS})\ \mathsf{EDU}(e_1)\ \big(\mathsf{Unit}(\mathit{ATTR}, \mathit{SN})\ \mathsf{EDU}(e_2)\ \mathsf{EDU}(e_3)\big)\big)$	$e_1 \mid e_2 \mid e_3$	

[e_1 Acme Inc. has closed several widget factories.] [e_2 The CEO told investors] [e_3 they were no longer profitable.]

Table 2: An example of a completed computation in our transition system.

The next action distribution $p(a_j | S_j, B_j)$ is parametrized using neural embeddings of the stack and buffer. Briefly, the next action distribution is computed using a softmax on the output of a linear transformation on the state embedding, which is the concatenation of a buffer embedding and a stack embedding. The buffer embedding is the final hidden state of an LSTM that reads the word embeddings of the words in the buffer. The stack embedding is the hidden state of a stack LSTM that reads the embeddings of the subtrees on the stack. The embeddings of the subtrees on the stack are computed recursively using a bidirectional LSTM that reads the embeddings of the nonterminal symbol and its children.

3.2 Transition System

We modify the RNNG generative process so that it generates an EDU-segmented document and its RST tree. In our model, the stack S holds partial RST trees and the buffer B holds a sequence of EDUs (a prefix of a document's EDU segmentation). The transitions generate EDUs and push them onto the stack and buffer, and combine RST subtrees on the stack into new subtrees. The process terminates when the buffer contains a complete document and the stack a single RST tree.

Kuncoro et al. (2018) presented an RNNG variant with a bottom-up transition system that replaces the $\mathsf{NT}(X)$ and REDUCE transitions with a single $\mathsf{REDUCE}(X, n)$ transition, as in traditional shift-reduce parsers. In initial experiments, we found this variant outperformed a model using the original top-down transition system. We hypothesize this is because an RST non-terminal's label is more difficult to predict from its parent's label than is the case in phrase structure trees, while a parent's label can be predicted once its children have been seen.

Finally, RST trees are traditionally binarized so we modify the REDUCE transitions accordingly, resulting in the following transition system (see also Table 1):

$\mathsf{GEN}(e)$ Generate the EDU e and push it onto the top of the stack and the end of the buffer.

$\mathsf{RE}(r, n)$ Pop the top two discourse units (U_L and U_R) from the stack and push the unit $\big(\mathsf{Unit}(r, n)\ U_L\ U_R\big)$ onto the top of the stack, with r and n relation and nuclearity labels.

In our experiments, the relation labels r are the 18 coarse-grained relations of Carlson and Marcu (2001), while the nuclearity labels n are in $\{\mathit{SN}, \mathit{NS}, \mathit{NN}\}$ corresponding respectively to a mononuclear relation with the nucleus on the right or the left and a binarized multinuclear relation.

Both transitions have conditions on when they can be performed (Table 1). A *computation* is a sequence of transitions where the condition for each transition is satisfied in its preceding state. A *completed computation* for an input sequence is a computation where the final state buffer contains the input sequence and the final state stack contains a single tree. Table 2 shows an example of a completed computation for our transition system.

3.3 Transition Model

In initial experiments we found, as did Kuncoro et al. (2017) for syntactic parsing, that conditioning only on the stack led to better parsing accuracy, so we specify the next action distribution as $p(a_j|S_j)$. To handle the unbounded number of possible EDUs, we parametrize the probabilities of $\mathsf{GEN}(e)$ actions using a neural language model. The next action distribution is factorised into a structural action distribution p_{trans} and a generation distribution p_{gen} as in Buys and Blunsom (2018), so that $p(\mathsf{RE}(r,n)|S) = p_{trans}(\mathsf{RE}(r,n)|S)$ and $p(\mathsf{GEN}(e)|S) = p_{trans}(\mathsf{GEN}|S) \cdot p_{gen}(e|S)$ where p_{gen} is the neural language model.

We parametrize p_{trans} as a feedforward neural network on an embedding of the stack $\mathbf{h}_S(S)$. In initial experiments we found, consistent with Morey et al. (2017), that a model with neural embeddings as its only features performed poorly. We therefore compute the representation using both neural embeddings of the discourse units on the stack (Section 3.3.1) and a set of structural features extracted from the stack (Section 3.3.2).

3.3.1 Neural Embeddings

To produce the stack embedding, we first require embeddings for both EDUs and units. We embed EDUs with bidirectional LSTMs[4]. If e is an EDU consisting of the word sequence $w_{1:k}$, then

$$\begin{aligned} \mathbf{h}_k^{\rightarrow} &= \mathsf{LSTM}^{(\rightarrow)}(\mathbf{w}_{1:k}, \mathbf{h}_0^{\rightarrow}) \\ \mathbf{h}_k^{\leftarrow} &= \mathsf{LSTM}^{(\leftarrow)}(\mathbf{w}_{k:1}, \mathbf{h}_0^{\leftarrow}) \end{aligned} \tag{2}$$

where $\mathbf{w}_t$ is the word embedding of w_t. The embedding for e, $\mathbf{h}_{EDU}(e)$, is the concatenation of the final forward and backward hidden states:

$$\mathbf{h}_{EDU}(e) = [\mathbf{h}_k^{\rightarrow}; \mathbf{h}_k^{\leftarrow}] \tag{3}$$

We embed units by composing their arguments with a Tree LSTM[5] (Teng and Zhang, 2017). A Tree LSTM recursively composes vectors while using memory cells to track long-term dependencies. We produce a new representation for each EDU e by applying a linear transformation to the EDU embeddings (omitting bias terms for brevity):

$$\mathbf{h}_U(e) = \mathbf{W}_{U,h} \cdot \mathbf{h}_{EDU}(e) \tag{4}$$

For a unit, we define the "nuclear" EDU of a unit recursively as the nucleus if the nucleus is an EDU, or the nuclear EDU of the nucleus if the nucleus is itself a unit. For multinuclear relations, we take the left-most nucleus. Then, if $(\mathsf{Unit}(r,n)\ U_L\ U_R)$ is a unit and e_N is its nuclear EDU, $\mathbf{h}_{EDU}(e_N)$ is the embedding of the nuclear EDU, and $\mathbf{h}_R(r,n)$ is an embedding of the nuclearity-relation pair (r,n) in a lookup table:

$$\begin{aligned} \mathbf{h}_U(U) = \mathsf{TREELSTM}([&\mathbf{h}_{EDU}(e_N); \mathbf{h}_R(r,n)], \\ &\mathbf{h}_U(U_L), \mathbf{h}_U(U_R)) \end{aligned} \tag{5}$$

where $\mathbf{h}_U(U_L)$ and $\mathbf{h}_U(U_R)$ are the hidden state and memory cell of the left and right argument of the unit respectively.

We embed the stack with a stack LSTM (Dyer et al., 2015). If the stack contents are $D_1|\cdots|D_m$ with each D_i being a discourse unit, then

$$\mathbf{h}_S^N(S) = \mathsf{LSTM}^S(\mathbf{h}_U(D_{1:m}), \mathbf{h}_0^S) \tag{6}$$

3.3.2 Structural Features

We extract additional features from the stack that have been found to be useful in prior work. As in Braud et al. (2017), for each discourse unit, we extract the word embeddings of up to three words whose syntactic head is not in the unit, adding padding if there are fewer than three. We concatenate these features for the top two discourse units on the stack, using a dummy embedding if the stack only contains one discourse unit. We write $\mathbf{h}_S^{head}(S)$ for these features.

We use a categorical feature for whether the top two discourse units are: in the same sentence; in different sentences; or incomparable since one of them spans multiple sentences. We also use an equivalent feature for paragraphs. Feature values are represented by embeddings in a lookup table. We write $\mathbf{h}_S^{comp}(S)$ for these features.

Finally, we extract features describing the dominance relation (Soricut and Marcu, 2003) between the top two discourse units on the stack. If there is a word in one discourse unit whose syntactic head is in the other, we extract the word embeddings of these two words as well as an embedding of the dependency relation between them, otherwise we use a single dummy embedding. We write $\mathbf{h}_S^{dom}(S)$ for these features.

[4] We track memory cells and use them when updating the hidden state in LSTMs and Tree LSTMs, but use only the hidden states for stack embeddings. Initial hidden states and memory cells are learned parameters.

[5] Since constituency trees are n-ary branching, RNNGs for constituency parsing have used a bidirectional LSTM composition function (Dyer et al., 2016; Kuncoro et al., 2017, 2018) to compose the variable number of children. RST trees are binarized so we do not need this feature.

The structural feature representation is then the concatenation of these three features:

$$\mathbf{h}_S^F(S) = [\mathbf{h}_S^{head}(S); \mathbf{h}_S^{comp}(S); \mathbf{h}_S^{dom}(S)] \quad (7)$$

and the full stack representation is the concatenation of the neural embedding and the feature representation:

$$\mathbf{h}_S(S) = [\mathbf{h}_S^N(S); \mathbf{h}_S^F(S)] \quad (8)$$

3.3.3 Probability Distributions

The action distribution is parametrized using the stack representation and an MLP:

$$\begin{aligned} p_{trans}(a|S) &= p_{trans}(a|\mathbf{h}_S(S)) \\ &= \text{softmax}(\mathbf{W}_{trans} \cdot \mathbf{h}_S(S)) \end{aligned} \quad (9)$$

We parametrize the EDU generation distribution $p_{gen}(e|S)$ with an LSTM decoder:

$$\mathbf{h}_t^{DEC} = \mathsf{LSTM}^{DEC}(\mathbf{w}_t, \mathbf{h}_{t-1}^{DEC}) \quad (10)$$

If $e = w_{1:k}$ then

$$\begin{aligned} p_{gen}(e|S) &= p_{gen}(w_{1:k}|S) && (11) \\ &= \prod_{t=1}^{k} p_{gen}(w_t|w_{<t}, S) && (12) \\ &= \prod_{t=1}^{k} p_{gen}(w_t|\mathbf{h}_{t-1}^{DEC}, \mathbf{h}_S(S)) && (13) \end{aligned}$$

where

$$\begin{aligned} & p_{gen}(w_t|\mathbf{h}_{t-1}^{DEC}, \mathbf{h}_S(S)) && (14) \\ & = \text{softmax}(\mathbf{W}_{gen} \cdot [\mathbf{h}_S(S); \mathbf{h}_{t-1}^{DEC}]) && (15) \end{aligned}$$

4 Inference

Our generative model specifies a joint probability $p(\boldsymbol{x}, \boldsymbol{y})$. We parse a document $\boldsymbol{x}$ by finding the MAP tree $\boldsymbol{y}^*$:

$$\begin{aligned} \boldsymbol{y}^* &= \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})}{\text{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x}) && (16) \\ &= \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})}{\text{argmax}}\, p(\boldsymbol{x}, \boldsymbol{y}) && (17) \end{aligned}$$

The search space grows exponentially with the input length, so we must perform inexact search as our model conditions on the entire relation structure of every subtree on the stack.

Search is generally more difficult for generative models than for discriminative ones, requiring more complex search algorithms. For this reason, Dyer et al. (2016) used RNNGs only to rerank the output of a discriminative parser. Fried et al. (2017) presented the first algorithm for decoding directly from RNNGs to give competitive performance. They found that action-level beam search (Zhang and Clark, 2008) gave poor performance for constituency parsing with RNNGs. The problem was that GEN actions almost always have lower probabilities than structure-generating actions, causing computations where GEN actions come earlier to "fall off the beam" even if the completed computation would have a higher probability than other completed computations.

To address this problem, Fried et al. (2017) proposed *word-level beam search* (Algorithm 1). Briefly, the algorithm keeps an array of beams indexed by the current position in the sequence and the number of structure-generating actions taken since this position was reached. The first beam for the current position $\mathcal{B}(i, 0)$ is filled from the successors of beams for the previous position $\mathcal{B}(i-1, j)$ (lines 4 to 17) starting with $\mathcal{B}(i-1, 0)$ (line 4) and incrementing j (line 17) until there are at least k items in $\mathcal{B}(i, 0)$ (line 5). The intuition is that analyses with the smallest number of structural actions since the previous beam was pruned have priority on the current beam.

We applied Fried et al. (2017)'s algorithm[6] to our model, but found it was biased towards producing left-branching trees. This led to poor performance as the *right-frontier constraint* (Polanyi, 1988; Webber, 1988; Asher, 2012; Asher et al., 2003) suggests discourse trees should be generally right-branching. In the next section, we present an analysis of the source of this bias and a novel beam search algorithm that corrects it.

4.1 Diagnosing Branching Bias

The trees returned by a trained parser depend on both the (learned) scoring model and the search algorithm. We can isolate bias in search algorithms by studying the trees they return when the scoring model contains no information. Intuitively, if the scoring model has no preference over trees, then any preference shown by the parser is the result of biases in the search algorithm.

We tested whether the left-branching bias came from the word-level beam search (the search algorithm of Fried et al. (2017)) by using it to parse sequences of various lengths using a bottom-up RNNG with a uniform scoring model. We broke

[6] We used *candidate fast-tracking* as described in Stern et al. (2017)'s extension to Fried et al. (2017)'s algorithm.

Algorithm 1 Word-level Beam Search

```
1: function SEARCH($x_{1:m}, k$)
2:     $\mathcal{B}[0,0], \leftarrow \{(1, (\epsilon, \epsilon))\}$
3:     for $i \leftarrow$ RANGE(0, $m$)
4:         $j \leftarrow 0$
5:         while $|\mathcal{B}[i,j]| \geq 0$ and $|\mathcal{B}[i+1,0]| < k$
6:             for $(v, s) \leftarrow$ TOP($\mathcal{B}[i,j], b$)
7:                 for $(a, s') \leftarrow$ SUCC($s$)
8:                     $v' \leftarrow v \cdot p(a|s)$
9:                     if COMPLETE($s'$) then
10:                        PUSH($\mathcal{B}[m+1,0]$, $(v', s')$)
11:                    else
12:                        switch $a$
13:                            case GEN($e_{i+1}$)
14:                                PUSH($\mathcal{B}[i+1,0]$, $(v', s')$)
15:                            case RE($r, n$)
16:                                PUSH($\mathcal{B}[i,j+1]$, $(v', s')$)
17:            $j \leftarrow j + 1$
18:    return TOP($\mathcal{B}[m+1,0], 1$)
```

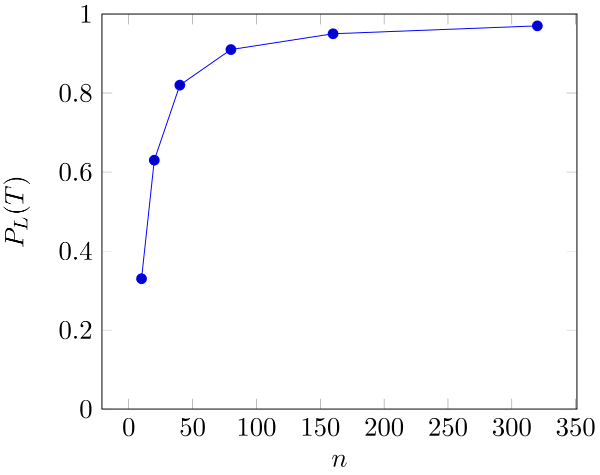

Figure 2: Median degree of left branching for trees obtained from a bottom-up RNNG with a uniform scoring model using word-level beam search for sequences of various lengths (n).

ties at beam cut-offs by uniform sampling without replacement. We measured branching bias using Sampson (1997)'s production-based measure of left-branching for parse trees which we write as $P_L(T)$ for a tree T. The measure is the fraction of non-terminals whose left child is also a non-terminal, and varies from 0 for a fully right-branching tree to $\frac{n-2}{n-1} \to 1$ for a fully left-branching tree, where n is the number of leaves. Figure 2 shows the median value of this measure for 100 trees each for sequences of various lengths from our uniform scoring model. It shows substantial left-branching bias which increases with sequence length.

Word-level beam search has two sources of bias: first, computations with fewer RE actions since the last GEN action are added to the next beam first (lines 4 and 17 in Alg. 1). Computations with more actions are only considered if the next beam is not already full by the time they are reached (line 5 in Alg. 1). This means the next beam may fill up before these computations are even considered and they will "fall off the beam". A right-branching subtree over k leaves has k consecutive GEN actions followed by $k-1$ consecutive RE actions, meaning it results in a computation in $\mathcal{B}(i_k, k-1)$ where i_k is the position of the k-th leaf. Thus right-branching subtrees are later in line to be considered and are increasingly likely to fall off the beam as they span more leaves.

Second, the beams $\mathcal{B}(i,j)$ contain computations with unequal numbers of actions. For a binary tree with m leaves, all completed computations have m GEN and $m-1$ RE actions. The total number of actions up to the k-th GEN action varies, though, from k to $2k-2$. Since the probability of a computation $a_{1:l}$ is $\prod_{j=1}^{l} p(a_j|a_{<j})$, this means word-level beam search compares computations with different numbers of factors contributing to their probabilities. This bias does not necessarily favour left-branching trees, but it does introduce a potential problem when comparing computations.

4.2 Bag-Level Beam Search

We now present a beam search algorithm without these sources of bias (Algorithm 2). Our algorithm is based on a simple dynamic program that keeps track of the number of GEN and RE actions separately. This (i) allows us to consider computations from all source beams simultaneously and (ii) ensures all computations in a beam have the same number of actions. Since this is equivalent to keeping separate beams for different bags of unlabelled actions, we call the algorithm bag-level beam search.

We write $\mathcal{C}(i,j)$ for the set of computations with i GEN actions and j RE actions. Then all completed computations are in $\mathcal{C}(m, m-1)$ for an input sequence of length m.

For each computation $c \in \mathcal{C}(i,j)$, the last action was either a GEN or an RE action, so c is either of the form $c = \text{GEN}|c^{'}$ where $c' \in \mathcal{C}(i-1,j)$ or it is of the form $c = \text{RE}|c^{''}$ where $c^{''} \in \mathcal{C}(i,j-1)$.

The highest scoring computation in $\mathcal{C}(i,j)$,

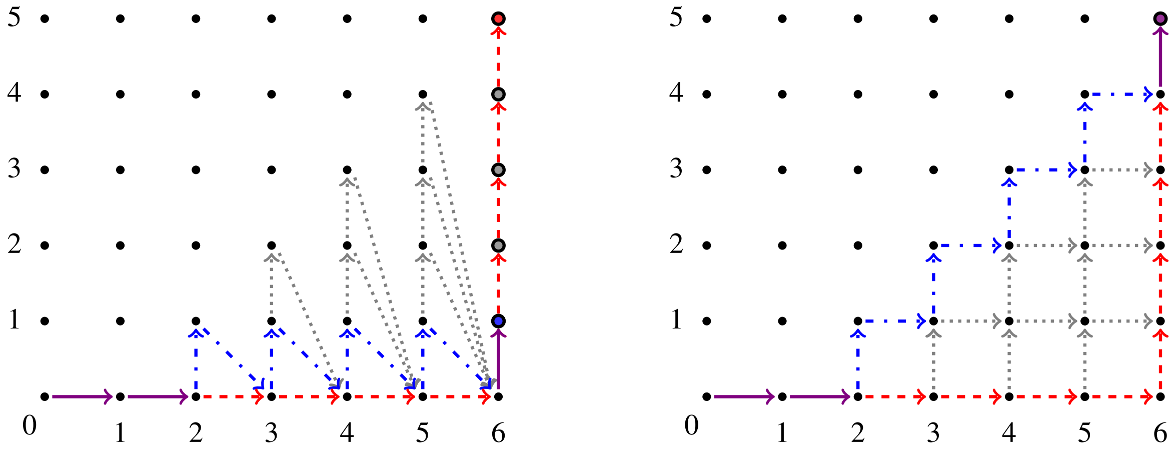

Figure 3: Word-level and bag-level beam search (left and right respectively) for an input sequence with 6 tokens. Nodes represent beams and paths represent computations. The horizontal axis is the number of GEN actions and the vertical axis is the number of RE actions for bag-level search and the number of RE actions since the last GEN for word-level search. We show the path of a left-branching tree in blue with dashed and dotted lines and a right-branching tree in red with dashed lines. We show possible transitions between beams that do not belong to either of these paths in gray with dotted lines. Red, blue and purple dots respectively show the beam where the computation of a right-branching tree, left-branching tree or both are completed.

Algorithm 2 Bag-level Beam Search

```
function SEARCH($x_{1:m}, k$)
    $B[0,0] \leftarrow \{(1, (\epsilon, \epsilon))\}$
    for $i \leftarrow$ RANGE(0, $m$)                         ▷ GEN($e_i$)
        for $j \leftarrow$ RANGE(0, $i-1$)                   ▷ RE($r, n$)
            for $(v, s) \leftarrow$ TOP($B[i,j], k$)
                for $(a, s') \leftarrow$ SUCC($s$)
                    $v' \leftarrow v \cdot p(a|s)$
                    switch $a$
                        case GEN($e_{i+1}$)
                            PUSH($B[i+1, j], (v', s')$)
                        case RE($r, n$)
                            PUSH($B[i, j+1], (v', s')$)

    return TOP($B[m, m-1], 1$)
```

$c^*(i,j) = \operatorname{argmax}_{c \in \mathcal{C}(i,j)} p(c)$, is then the highest scoring computation ending on a GEN or an RE[7]:

$$c^*(i,j) = \operatorname{argmax} \left\{ p(c) \middle| \begin{array}{l} c = \mathsf{GEN}|c', c' \in \mathcal{C}(i-1,j); \\ c = \mathsf{RE}|c'', c'' \in \mathcal{C}(i,j-1) \end{array} \right\} \tag{18}$$

There are exponentially many computations in $\mathcal{C}(i,j)$ so taking exact maxima is intractable. Therefore we only take maxima over beams $\mathcal{B}(i,j)$ which we update according to

$$\mathcal{B}(i,j) = \operatorname{argmax}_k \left\{ p(c) \middle| \begin{array}{l} c = \mathsf{GEN}|c', c' \in \mathcal{B}(i-1,j); \\ c = \mathsf{RE}|c'', c'' \in \mathcal{B}(i,j-1) \end{array} \right\} \tag{19}$$

where, in the set notation, ";" means "or".

We perform this recursive calculation for all i and j using the dynamic program in Algorithm 2.

Figure 3 shows the differences between word-level and bag-level beam search with example trajectories through the array of beams for computations corresponding to a left-branching (blue, dashed and dotted) and a right-branching (red, dashed) tree. Each path through the lattice from $(0,0)$ to (i,j) defines a computation and shows the beams it must pass through to end up in $\mathcal{B}(i,j)$.

The path length is equal to the number of actions taken to reach (i,j). In word-level beam search, paths through beams with more consecutive RE actions (higher values on the vertical axis) are only explored if the next word beam is not already full (lines 4, 5 and 17 in Algorithm 1). This means the final beam may be full before the red path is considered, causing it to "fall off the beam". In bag-level beam search, paths into a beam from both source beams are considered and pruned simultaneously. This addresses the first source of bias, namely that sequences with fewer consecutive RE actions are given priority. All paths to (i,j) also have the same length; in other words, all computations in $\mathcal{B}(i,j)$ have the same number of actions $(i+j)$, addressing the second source of bias.

[7] We omit actions' parameters for conciseness.

Metric	Algorithm	Beam Size			
		10	20	40	80
S	Word-level Search	58.3	58.4	58.4	60.8
	Bag-level Search	66.4	67.3	67.0	67.6
N	Word-level Search	50.3	50.5	50.5	51.8
	Bag-level Search	56.1	56.4	56.2	56.9
R	Word-level Search	43.1	43.2	43.2	43.7
	Bag-level Search	45.4	46.8	46.5	46.9
F	Word-level Search	42.0	42.3	42.3	42.9
	Bag-level Search	44.7	45.5	45.3	45.8

Table 3: Dev. set micro-averaged F_1 scores on labelled attachment for word-level and bag-level beam search.

5 Experiments

5.1 Dataset

We train and evaluate our models on the RST Discourse Treebank (RST-DT) (Carlson and Marcu, 2001). We evaluate on the standard test set, but we use 25 documents from the training set as a development set. To reduce the rare token count, we use the spaCy (Honnibal and Montani, 2017) named entity recognition model to replace named entities with their named entity tags.

5.2 Evaluation

We follow the evaluation setup used by Morey et al. (2017). They performed a replication study of several competitive RST parsers and implemented a consistent evaluation procedure. They found that micro- and macro-averaged F_1 had been used inconsistently in the RST parsing literature, and that the standard evaluation metrics (RST Parseval) gave inflated results. Following this study we evaluate using micro-averaged F_1 scores on labelled attachment decisions as calculated by the EDUCE python package[8]. We report F_1 for predicting span attachments (S), span attachments with nuclearity (N), span attachments with relation labels (R) and span attachments with nuclearity and relation labels (F).

We compare our results against the numbers from Morey et al. (2017), since they include several competitive parers under a consistent evaluation scheme.[9]

As a baseline, we use a discriminative version of our model. This is a shift-reduce parser with the same EDU, unit and stack representations as our model, but with a lookahead buffer representation as well. For the buffer representation, we run a backward LSTM over the representations of the remaining EDUs in the buffer.

5.3 Training and Hyperparameters

We use 300-dimensional word embeddings initialized to word2vec vectors (Mikolov et al., 2013). We tie the embeddings in the EDU LSTM and the decoder LSTM input and output embeddings. We use a 2-layer bidirectional LSTM with 512-dimensional hidden state for the EDU LSTM. The TreeLSTM composition function also has a 512-dimensional (in total) hidden state with 100-dimensional relation embeddings. The stack LSTM and decoder LSTM also have 512-dimensional hidden states. For the structural features, we use 10-dimensional sentence and paragraph boundary feature embeddings and 50-dimensional dependency relation embeddings.

We train the models with Adam (Kingma and Ba, 2014) using an initial learning rate of 10^{-3} and default values for the other hyperparameters. We apply blank noise variational smoothing (Kong et al., 2019) with a dropout rate of 0.25 to the tied embeddings to regularize the model. In particular, for each document we sample a set of word types to drop and replace their word embeddings with the $<$ UNK $>$ token's word embedding.

We extract structural features using the sentence and paragraph boundary annotation in the RST-DT, and dependency trees obtained from the spaCy parser. Our models were implemented in PyTorch (Paszke et al., 2017).

5.4 Results

5.4.1 Search Comparison

Table 3 shows RST-DT development set labelled attachment metrics for our parser using word-level and bag-level beam search. Our search algorithm outperforms word-level beam search on all of the metrics across beam sizes.[10] On spans with nuclearity (N), bag-level beam search outperforms word-level beam search by 5.9% to 8.1%. This

[8] https://github.com/irit-melodi/educe

[9] We do not compare against Yu et al. (2018), Zhang et al. (2018) and Lin et al. (2019)'s recent neural RST parsers since they do not evaluate labelled attachment decisions so their results are not comparable to ours.

[10] Word-level beam search has three beam size parameters: the structural action beam size k, the next-word beam size k_w and the fast-track beam size k_s. For top-down parsers, Stern et al. (2017) set $k_w = k/10$, $k_s = k/100$, but in tuning these parameters on the development set we found the best performance with $k_w = k$, $k_s = k/10$ for our parser.

Model	S	N	R	F
Feature-based parsers				
Hayashi et al. (2016)	65.1	54.6	44.7	44.1
Surdeanu et al. (2015)	65.3	54.2	45.1	44.2
Joty et al. (2015)	65.1	55.5	45.1	44.3
Feng and Hirst (2014a)	***68.6***	*55.9*	***45.8***	*44.6*
Neural parsers				
Braud et al. (2016)	59.5	47.2	34.7	34.3
Li et al. (2016)	64.5	54.0	38.1	36.6
Braud et al. (2017) (mono)	61.9	53.4	44.5	44.0
Our work				
Discriminative Baseline	65.2	54.9	42.8	42.4
Generative Model	*67.1*	***57.4***	*45.5*	***45.0***
Unpublished				
Ji and Eisenstein (2014) (updated)	64.1	54.2	46.8	46.3
Additional data				
Braud et al. (2017) (cross + dev)	62.7	54.5	45.5	45.1

Table 4: Test set micro-averaged F_1 scores on labelled attachment decisions. We report numbers for other parsers from Morey et al. (2017)'s replication study. For each metric, the highest score for all the parsers in the comparison is shown in bold, while the highest score among parsers of that type (neural or feature-based) is in italics.

is consistent with the branching bias in word-level search leading it to return trees whose structure differs from the trees in the RST-DT. The poor performance on structure prediction also seems to have a knock-on effect on the relation and full tree prediction accuracy.

5.4.2 Parsing Performance

Table 4 shows RST-DT test set labelled attachment metrics for various parsers. Our model outperforms all of the published[11] neural models that do not use additional training data[12] in Morey et al. (2017)'s replication study on all of the metrics. On span accuracy (S), we outperform all of the other parsers except for Feng and Hirst (2014a)'s graph CRF model. On spans with nuclearity (N), the equivalent of the unlabelled attachment score for discourse dependencies, we outperform all of the parsers in the study. We perform competitively on spans with relations (R), and we outperform all of the published parsers that do not use additional data on spans with nuclearity and relations (F).

Our model also outperforms the discriminative baseline using the same features and implementation on all metrics by between 1.9% and 2.7%.

6 Conclusion

We introduced the first generative model for RST parsing. We showed that word-level beam search has a branching bias for bottom-up RNNGs which hurt performance on our task. We proposed a novel beam search algorithm that does not have this branching bias and that outperformed word-level beam search across beam sizes and with different evaluation metrics. With our search algorithm, our generative model achieved state-of-the art-level RST parsing performance, outperforming all of the published RST parsers from a recent study that do not use additional training data on labelled attachment F_1. Our results show that generative modelling is an effective approach to RST parsing, with superior structure prediction and competitive relation prediction performance.

Acknowledgments

Andreas Vlachos is supported by the EPSRC grant eNeMILP (EP/R021643/1). Amandla Mabona is supported by Commonwealth and Sansom Scholarships. We thank the anonymous reviewers for their helpful comments.

[11] Ji and Eisenstein (2014) presented a transition-based parser that used continuous bag-of-words representations for EDUs and an SVM as the next action classifier. For Morey et al. (2017)'s study, they submitted predicted discourse trees from an updated, unpublished version of their parser.

[12] In the cross+dev setting, Braud et al. (2017) train their parser on RST discourse treebanks for several languages.

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 (Volume 1: Long Papers)*, pages 2442–2452.

Nicholas Asher. 2012. *Reference to abstract objects in discourse*, volume 50. Springer Science & Business Media.

Nicholas Asher, Nicholas Michael Asher, and Alex Lascarides. 2003. *Logics of conversation*. Cambridge University Press.

Parminder Bhatia, Yangfeng Ji, and Jacob Eisenstein. 2015. Better document-level sentiment analysis from rst discourse parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2212–2218.

Chlo Braud, Maximin Coavoux, and Anders Sgaard. 2017. Cross-lingual RST Discourse Parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 292–304, Valencia, Spain. Association for Computational Linguistics.

Chlo Braud, Barbara Plank, and Anders Sgaard. 2016. Multi-view and multi-task training of RST discourse parsers. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1903–1913, Osaka, Japan. The COLING 2016 Organizing Committee.

Jan Buys and Phil Blunsom. 2018. Neural Syntactic Generative Models with Exact Marginalization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 942–952, New Orleans, Louisiana. Association for Computational Linguistics.

Lynn Carlson and Daniel Marcu. 2001. Discourse tagging reference manual. *ISI Technical Report ISI-TR-545*, 54:56.

Lynn Carlson, Daniel Marcu, and Mary Ellen Okurowski. 2003. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Current and new directions in discourse and dialogue*, pages 85–112. Springer.

Eugene Charniak et al. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. 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 334–343.

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.

Vanessa Wei Feng and Graeme Hirst. 2014a. A Linear-Time Bottom-Up Discourse Parser with Constraints and Post-Editing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 511–521, Baltimore, Maryland. Association for Computational Linguistics.

Vanessa Wei Feng and Graeme Hirst. 2014b. Patterns of local discourse coherence as a feature for authorship attribution. *Literary and Linguistic Computing*, 29(2):191–198.

Elisa Ferracane, Su Wang, and Raymond Mooney. 2017. Leveraging Discourse Information Effectively for Authorship Attribution. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 584–593, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Daniel Fried, Mitchell Stern, and Dan Klein. 2017. Improving neural parsing by disentangling model combination and reranking effects. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 161–166.

John J Godfrey, Edward C Holliman, and Jane McDaniel. 1992. Switchboard: Telephone speech corpus for research and development. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 517–520. IEEE.

Katsuhiko Hayashi, Tsutomu Hirao, and Masaaki Nagata. 2016. Empirical comparison of dependency conversions for RST discourse trees. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 128–136, Los Angeles. Association for Computational Linguistics.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Yangfeng Ji and Jacob Eisenstein. 2014. Representation Learning for Text-level Discourse Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24, Baltimore, Maryland. Association for Computational Linguistics.

Yangfeng Ji, Gholamreza Haffari, and Jacob Eisenstein. 2016. A latent variable recurrent neural network for discourse relation language models. In *Proceedings of NAACL-HLT*, pages 332–342.

Yangfeng Ji and Noah A Smith. 2017. Neural discourse structure for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 996–1005.

Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. 2015. CODRA: A Novel Discriminative Framework for Rhetorical Analysis. *Computational Linguistics*, 41(3):385–435.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. ArXiv: 1412.6980.

Lingpeng Kong, Gabor Melis, Wang Ling, Lei Yu, and Dani Yogatama. 2019. Variational Smoothing in Recurrent Neural Network Language Models. *arXiv:1901.09296 [cs]*. ArXiv: 1901.09296.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What Do Recurrent Neural Network Grammars Learn About Syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain. Association for Computational Linguistics.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436.

John D Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann Publishers Inc.

Qi Li, Tianshi Li, and Baobao Chang. 2016. Discourse Parsing with Attention-based Hierarchical Neural Networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 362–371, Austin, Texas. Association for Computational Linguistics.

Xiang Lin, Shafiq Joty, Prathyusha Jwalapuram, and Saiful Bari. 2019. A unified linear-time framework for sentence-level discourse parsing. *arXiv preprint arXiv:1905.05682*.

William C. Mann and Sandra A. Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mathieu Morey, Philippe Muller, and Nicholas Asher. 2017. How much progress have we made on RST discourse parsing? A replication study of recent results on the RST-DT. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1319–1324, Copenhagen, Denmark. Association for Computational Linguistics.

Andrew Y. Ng and Michael I. Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Livia Polanyi. 1988. A formal model of the structure of discourse. *Journal of pragmatics*, 12(5-6):601–638.

Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind K Joshi, and Bonnie L Webber. 2008. The penn discourse treebank 2.0. In *LREC*. Citeseer.

Geoffrey Sampson. 1997. Depth in English grammar. *Journal of Linguistics*, 33(1):131–151.

Radu Soricut and Daniel Marcu. 2003. Sentence Level Discourse Parsing using Syntactic and Lexical Information. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 228–235.

Mitchell Stern, Daniel Fried, and Dan Klein. 2017. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1695–1700.

Mihai Surdeanu, Tom Hicks, and Marco Antonio Valenzuela-Escarcega. 2015. Two Practical Rhetorical Structure Theory Parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 1–5, Denver, Colorado. Association for Computational Linguistics.

Zhiyang Teng and Yue Zhang. 2017. Head-Lexicalized Bidirectional Tree LSTMs. *Transactions of the Association for Computational Linguistics*, 5:163–177.

Bonnie L Webber. 1988. Discourse deixis and discourse processing. *Discourse*, 9.

D Yogatama, C Dyer, W Ling, and P Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. In *Thirty-fourth International Conference on Machine Learning (ICML 2017)*. International Machine Learning Society.

Nan Yu, Meishan Zhang, and Guohong Fu. 2018. Transition-based neural rst parsing with implicit syntax features. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 559–570.

Longyin Zhang, Cheng Sun, Xin Tan, and Fang Kong. 2018. Rst discourse parsing with tree-structured neural networks. In *China Workshop on Machine Translation*, pages 15–26. Springer.

Yue Zhang and Stephen Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing Using Beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 562–571, Stroudsburg, PA, USA. Association for Computational Linguistics. Event-place: Honolulu, Hawaii.