Recurrent Neural Network Grammars

Chris Dyer[♠] **Adhiguna Kuncoro**[♠] **Miguel Ballesteros**[♢♠] **Noah A. Smith**[♡]
[♠]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[♢]NLP Group, Pompeu Fabra University, Barcelona, Spain
[♡]Computer Science & Engineering, University of Washington, Seattle, WA, USA
{cdyer,akuncoro}@cs.cmu.edu, miguel.ballesteros@upf.edu, nasmith@cs.washington.edu

Abstract

We introduce recurrent neural network grammars, probabilistic models of sentences with explicit phrase structure. We explain efficient inference procedures that allow application to both parsing and language modeling. Experiments show that they provide better parsing in English than any single previously published supervised generative model and better language modeling than state-of-the-art sequential RNNs in English and Chinese.

1 Introduction

Sequential recurrent neural networks (RNNs) are remarkably effective models of natural language. In the last few years, language model results that substantially improve over long-established state-of-the-art baselines have been obtained using RNNs (Zaremba et al., 2015; Mikolov et al., 2010) as well as in various conditional language modeling tasks such as machine translation (Bahdanau et al., 2015), image caption generation (Xu et al., 2015), and dialogue generation (Wen et al., 2015). Despite these impressive results, sequential models are *a priori* inappropriate models of natural language, since relationships among words are largely organized in terms of latent nested structures rather than sequential surface order (Chomsky, 1957).

In this paper, we introduce **recurrent neural network grammars** (RNNGs; §2), a new generative probabilistic model of sentences that explicitly models nested, hierarchical relationships among words and phrases. RNNGs operate via a recursive syntactic process reminiscent of probabilistic context-free grammar generation, but decisions are parameterized using RNNs that condition on the entire syntactic derivation history, greatly relaxing context-free independence assumptions.

The foundation of this work is a top-down variant of transition-based parsing (§3). We give two variants of the algorithm, one for parsing (given an observed sentence, transform it into a tree), and one for generation. While several transition-based neural models of syntactic generation exist (Henderson, 2003, 2004; Emami and Jelinek, 2005; Titov and Henderson, 2007; Buys and Blunsom, 2015b), these have relied on structure building operations based on parsing actions in shift-reduce and left-corner parsers which operate in a largely bottom-up fashion. While this construction is appealing because inference is relatively straightforward, it limits the use of top-down grammar information, which is helpful for generation (Roark, 2001).[1] RNNGs maintain the algorithmic convenience of transition-based parsing but incorporate top-down (i.e., root-to-terminal) syntactic information (§4).

The top-down transition set that RNNGs are based on lends itself to discriminative modeling as well, where sequences of transitions are modeled conditional on the full input sentence along with the incrementally constructed syntactic structures. Similar to previously published discriminative bottom-up transition-based parsers (Henderson, 2004; Sagae and Lavie, 2005; Zhang and Clark, 2011, *inter alia*), greedy prediction with our model yields a linear-time deterministic parser (provided an upper bound on the number of actions taken between processing subsequent terminal symbols is imposed); however, our algorithm generates arbitrary tree structures directly, without the binarization required by shift-reduce parsers. The discriminative model also lets us use ancestor sampling to obtain samples of parse trees for sentences, and this is used to solve

[1]The left-corner parsers used by Henderson (2003, 2004) incorporate limited top-down information, but a complete path from the root of the tree to a terminal is not generally present when a terminal is generated. Refer to Henderson (2003, Fig. 1) for an example.

Proceedings of NAACL-HLT 2016, pages 199–209,
San Diego, California, June 12-17, 2016.

a second practical challenge with RNNGs: approximating the marginal likelihood and MAP tree of a sentence under the generative model. We present a simple importance sampling algorithm which uses samples from the discriminative parser to solve inference problems in the generative model (§5).

Experiments show that RNNGs are effective for both language modeling and parsing (§6). Our generative model obtains (i) the best-known parsing results using a single supervised generative model and (ii) better perplexities in language modeling than state-of-the-art sequential LSTM language models. Surprisingly—although in line with previous parsing results showing the effectiveness of generative models (Henderson, 2004; Johnson, 2001)—parsing with the generative model obtains significantly better results than parsing with the discriminative model.

2 RNN Grammars

Formally, an RNNG is a triple (N, Σ, Θ) consisting of a finite set of nonterminal symbols (N), a finite set of terminal symbols (Σ) such that $N \cap \Sigma = \emptyset$, and a collection of neural network parameters Θ. It does not explicitly define rules since these are implicitly characterized by Θ. The algorithm that the grammar uses to generate trees and strings in the language is characterized in terms of a transition-based algorithm, which is outlined in the next section. In the section after that, the semantics of the parameters that are used to turn this into a stochastic algorithm that generates pairs of trees and strings are discussed.

3 Top-down Parsing and Generation

RNNGs are based on a top-down generation algorithm that relies on a stack data structure of partially completed syntactic constituents. To emphasize the similarity of our algorithm to more familiar bottom-up shift-reduce recognition algorithms, we first present the parsing (rather than generation) version of our algorithm (§3.1) and then present modifications to turn it into a generator (§3.2).

3.1 Parser Transitions

The parsing algorithm transforms a sequence of words $\boldsymbol{x}$ into a parse tree $\boldsymbol{y}$ using two data structures (a stack and an input buffer). As with the bottom-up algorithm of Sagae and Lavie (2005), our algorithm begins with the stack (S) empty and the complete sequence of words in the input buffer (B). The buffer contains unprocessed terminal symbols, and the stack contains terminal symbols, "open" nonterminal symbols, and completed constituents. At each timestep, one of the following three classes of operations (Fig. 1) is selected by a classifier, based on the current contents on the stack and buffer:

- NT(X) introduces an "open nonterminal" X onto the top of the stack. Open nonterminals are written as a nonterminal symbol preceded by an open parenthesis, e.g., "(VP", and they represent a nonterminal whose child nodes have not yet been fully constructed. Open nonterminals are "closed" to form complete constituents by subsequent REDUCE operations.
- SHIFT removes the terminal symbol x from the front of the input buffer, and pushes it onto the top of the stack.
- REDUCE repeatedly pops completed subtrees or terminal symbols from the stack until an open nonterminal is encountered, and then this open NT is popped and used as the label of a new constituent that has the popped subtrees as its children. This new completed constituent is pushed onto the stack as a single composite item. A single REDUCE operation can thus create constituents with an unbounded number of children.

The parsing algorithm terminates when there is a single completed constituent on the stack and the buffer is empty. Fig. 2 shows an example parse using our transition set. Note that in this paper we do not model preterminal symbols (i.e., part-of-speech tags) and our examples therefore do not include them.[2]

Our transition set is closely related to the operations used in Earley's algorithm which likewise introduces nonterminals symbols with its PREDICT

[2]Preterminal symbols are, from the parsing algorithm's point of view, just another kind of nonterminal symbol that requires no special handling. However, leaving them out reduces the number of transitions by $O(n)$ and also reduces the number of action types, both of which reduce the runtime. Furthermore, standard parsing evaluation scores do not depend on preterminal prediction accuracy.

operation and later COMPLETEs them after consuming terminal symbols one at a time using SCAN (Earley, 1970). It is likewise closely related to the "linearized" parse trees proposed by Vinyals et al. (2015) and to the top-down, left-to-right decompositions of trees used in previous generative parsing and language modeling work (Roark, 2001, 2004; Charniak, 2010).

A further connection is to $LL(^*)$ parsing which uses an unbounded lookahead (compactly represented by a DFA) to distinguish between parse alternatives in a top-down parser (Parr and Fisher, 2011); however, our parser uses an RNN encoding of the lookahead rather than a DFA.

Constraints on parser transitions. To guarantee that only well-formed phrase-structure trees are produced by the parser, we impose the following constraints on the transitions that can be applied at each step which are a function of the parser state (B, S, n) where n is the number of open nonterminals on the stack:

- The NT(X) operation can only be applied if B is not empty and $n < 100$.[3]
- The SHIFT operation can only be applied if B is not empty and $n \geq 1$.
- The REDUCE operation can only be applied if the top of the stack is not an open nonterminal symbol.
- The REDUCE operation can only be applied if $n \geq 2$ or if the buffer is empty.

To designate the set of valid parser transitions, we write $\mathcal{A}_D(B, S, n)$.

3.2 Generator Transitions

The parsing algorithm that maps from sequences of words to parse trees can be adapted with minor changes to produce an algorithm that stochastically generates trees and terminal symbols. Two changes are required: (i) there is no input buffer of unprocessed words, rather there is an output buffer (T), and (ii) instead of a SHIFT operation there are GEN(x) operations which generate terminal symbol $x \in \Sigma$ and add it to the top of the stack and the output buffer. At each timestep an action is stochastically selected according to a conditional distribution that depends on the current contents of B and T. The algorithm terminates when a single completed constituent remains on the stack. Fig. 4 shows an example generation sequence.

Constraints on generator transitions. The generation algorithm also requires slightly modified constraints. These are:

- The GEN(x) operation can only be applied if $n \geq 1$.
- The REDUCE operation can only be applied if the top of the stack is not an open nonterminal symbol and $n \geq 1$.

To designate the set of valid generator transitions, we write $\mathcal{A}_G(T, S, n)$.

This transition set generates trees using nearly the same structure building actions and stack configurations as the "top-down PDA" construction proposed by Abney et al. (1999), albeit without the restriction that the trees be in Chomsky normal form.

3.3 Transition Sequences from Trees

Any parse tree can be converted to a sequence of transitions via a depth-first, left-to-right traversal of a parse tree. Since there is a unique depth-first, left-ro-right traversal of a tree, there is exactly one transition sequence of each tree. For a tree $\boldsymbol{y}$ and a sequence of symbols $\boldsymbol{x}$, we write $\boldsymbol{a}(\boldsymbol{x}, \boldsymbol{y})$ to indicate the corresponding sequence of generation transitions, and $\boldsymbol{b}(\boldsymbol{x}, \boldsymbol{y})$ to indicate the parser transitions.

3.4 Runtime Analysis

A detailed analysis of the algorithmic properties of our top-down parser is beyond the scope of this paper; however, we briefly state several facts. Assuming the availability of constant time push and pop operations, the runtime is linear in the number of the nodes in the parse tree that is generated by the parser/generator (intuitively, this is true since although an individual REDUCE operation may require

[3] Since our parser allows unary nonterminal productions, there are an infinite number of valid trees for finite-length sentences. The $n < 100$ constraint prevents the classifier from misbehaving and generating excessively large numbers of nonterminals. Similar constraints have been proposed to deal with the analogous problem in bottom-up shift-reduce parsers (Sagae and Lavie, 2005).

Stack$_t$	**Buffer**$_t$	**Open NTs**$_t$	**Action**	**Stack**$_{t+1}$	**Buffer**$_{t+1}$	**Open NTs**$_{t+1}$
S	B	n	NT(X)	$S \mid (\mathrm{X}$	B	$n+1$
S	$x \mid B$	n	SHIFT	$S \mid x$	B	n
$S \mid (\mathrm{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	B	n	REDUCE	$S \mid (\mathrm{X}\ \tau_1\ \ldots\ \tau_\ell)$	B	$n-1$

Figure 1: Parser transitions showing the stack, buffer, and open nonterminal count before and after each action type. S represents the stack, which contains open nonterminals and completed subtrees; B represents the buffer of unprocessed terminal symbols; x is a terminal symbol, X is a nonterminal symbol, and each τ is a completed subtree. The top of the stack is to the right, and the buffer is consumed from left to right. Elements on the stack and buffer are delimited by a vertical bar (|).

Input: *The hungry cat meows .*

	Stack	**Buffer**	**Action**
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

Figure 2: Top-down parsing example.

Stack$_t$	**Terms**$_t$	**Open NTs**$_t$	**Action**	**Stack**$_{t+1}$	**Terms**$_{t+1}$	**Open NTs**$_{t+1}$
S	T	n	NT(X)	$S \mid (\mathrm{X}$	T	$n+1$
S	T	n	GEN(x)	$S \mid x$	$T \mid x$	n
$S \mid (\mathrm{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	T	n	REDUCE	$S \mid (\mathrm{X}\ \tau_1\ \ldots\ \tau_\ell)$	T	$n-1$

Figure 3: Generator transitions. Symbols defined as in Fig. 1 with the addition of T representing the history of generated terminals.

	Stack	**Terminals**	**Action**
0			NT(S)
1	(S		NT(NP)
2	(S | (NP		GEN(*The*)
3	(S | (NP | *The*	*The*	GEN(*hungry*)
4	(S | (NP | *The* | *hungry*	*The* | *hungry*	GEN(*cat*)
5	(S | (NP | *The* | *hungry* | *cat*	*The* | *hungry* | *cat*	REDUCE
6	(S | (NP *The hungry cat*)	*The* | *hungry* | *cat*	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*The* | *hungry* | *cat*	GEN(*meows*)
8	(S | (NP *The hungry cat*) | (VP *meows*	*The* | *hungry* | *cat* | *meows*	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	*The* | *hungry* | *cat* | *meows*	GEN(.)
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	*The* | *hungry* | *cat* | *meows* | .	REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)	*The* | *hungry* | *cat* | *meows* | .	

Figure 4: Joint generation of a parse tree and sentence.

applying a number of pops that is linear in the number of input symbols, the total number of pop operations across an entire parse/generation run will also be linear). Since there is no way to bound the number of output nodes in a parse tree as a function of the number of input words, stating the runtime complexity of the parsing algorithm as a function of the input size requires further assumptions. Assuming our fixed constraint on maximum depth, it is linear.

3.5 Comparison to Other Models

Our generation algorithm algorithm differs from previous stack-based parsing/generation algorithms in two ways. First, it constructs rooted tree structures top down (rather than bottom up), and second, the transition operators are capable of directly generating arbitrary tree structures rather than, e.g., assuming binarized trees, as is the case in much prior work that has used transition-based algorithms to produce phrase-structure trees (Sagae and Lavie, 2005; Zhang and Clark, 2011; Zhu et al., 2013).

4 Generative Model

RNNGs use the generator transition set just presented to define a joint distribution on syntax trees ($\boldsymbol{y}$) and words ($\boldsymbol{x}$). This distribution is defined as a sequence model over generator transitions that is parameterized using a continuous space embedding of the algorithm state at each time step ($\mathbf{u}_t$); i.e.,

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} p(a_t \mid \boldsymbol{a}_{<t}) = \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} \frac{\exp \mathbf{r}_{a_t}^\top \mathbf{u}_t + b_{a_t}}{\sum_{a' \in \mathcal{A}_G(T_t,S_t,n_t)} \exp \mathbf{r}_{a'}^\top \mathbf{u}_t + b_{a'}},$$

and where action-specific embeddings $\mathbf{r}_a$ and bias vector $\mathbf{b}$ are parameters in Θ.

The representation of the algorithm state at time t, $\mathbf{u}_t$, is computed by combining the representation of the generator's three data structures: the output buffer (T_t), represented by an embedding $\mathbf{o}_t$, the stack (S_t), represented by an embedding $\mathbf{s}_t$, and the history of actions ($\boldsymbol{a}_{<t}$) taken by the generator, represented by an embedding $\mathbf{h}_t$,

$$\mathbf{u}_t = \tanh\left(\mathbf{W}[\mathbf{o}_t; \mathbf{s}_t; \mathbf{h}_t] + \mathbf{c}\right),$$

where $\mathbf{W}$ and $\mathbf{c}$ are parameters. Refer to Figure 5 for an illustration of the architecture.

The output buffer, stack, and history are sequences that grow unboundedly, and to obtain representations of them we use recurrent neural networks to "encode" their contents (Cho et al., 2014). Since the output buffer and history of actions are only appended to and only contain symbols from a finite alphabet, it is straightforward to apply a standard RNN encoding architecture. The stack (S) is more complicated for two reasons. First, the elements of the stack are more complicated objects than symbols from a discrete alphabet: open nonterminals, terminals, and full trees, are all present on the stack. Second, it is manipulated using both push and pop operations. To efficiently obtain representations of S under push and pop operations, we use stack LSTMs (Dyer et al., 2015).

4.1 Syntactic Composition Function

When a REDUCE operation is executed, the parser pops a sequence of completed subtrees and/or tokens (together with their vector embeddings) from the stack and makes them children of the most recent open nonterminal on the stack, "completing" the constituent. To compute an embedding of this new subtree, we use a composition function based on bidirectional LSTMs, which is illustrated in Fig. 6.

The first vector read by the LSTM in both the forward and reverse directions is an embedding of the label on the constituent being constructed (in the figure, NP). This is followed by the embeddings of the child subtrees (or tokens) in forward or reverse order. Intuitively, this order serves to "notify" each LSTM what sort of head it should be looking for as it processes the child node embeddings. The final state of the forward and reverse LSTMs are concatenated, passed through an affine transformation and a $\tanh$ nonlinearity to become the subtree embedding.[4] Because each of the child node embeddings ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ in Fig. 6) is computed similarly (if it corresponds to an

[4] We found the many previously proposed syntactic composition functions inadequate for our purposes. First, we must contend with an unbounded number of children, and many previously proposed functions are limited to binary branching nodes (Socher et al., 2013b; Dyer et al., 2015). Second, those that could deal with n-ary nodes made poor use of nonterminal information (Tai et al., 2015), which is crucial for our task.

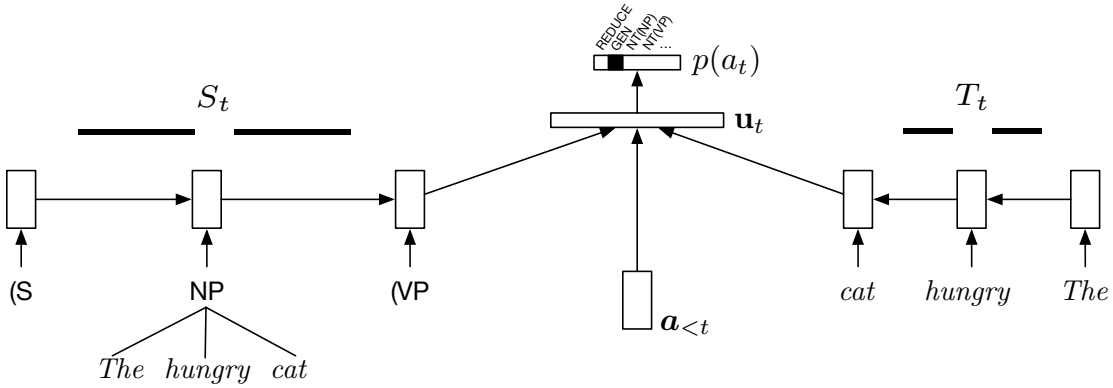

Figure 5: Neural architecture for defining a distribution over a_t given representations of the stack (S_t), output buffer (T_t) and history of actions ($\boldsymbol{a}_{<t}$). Details of the composition architecture of the NP, the action history LSTM, and the other elements of the stack are not shown. This architecture corresponds to the generator state at line 7 of Figure 4.

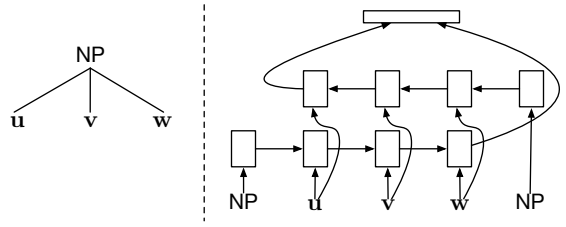

Figure 6: Syntactic composition function based on bidirectional LSTMs that is executed during a REDUCE operation; the network on the right models the structure on the left.

internal node), this composition function is a kind of recursive neural network.

4.2 Word Generation

To reduce the size of $\mathcal{A}_G(S, T, n)$, word generation is broken into two parts. First, the decision to generate is made (by predicting GEN as an action), and then choosing the word, conditional on the current parser state. To further reduce the computational complexity of modeling the generation of a word, we use a class-factored softmax (Baltescu and Blunsom, 2015; Goodman, 2001). By using $\sqrt{|\Sigma|}$ classes for a vocabulary of size $|\Sigma|$, this prediction step runs in time $O(\sqrt{|\Sigma|})$ rather than the $O(|\Sigma|)$ of the full-vocabulary softmax. To obtain clusters, we use the greedy agglomerative clustering algorithm of Brown et al. (1992).

4.3 Training

The parameters in the model are learned to maximize the likelihood of a corpus of trees.

4.4 Discriminative Parsing Model

A discriminative parsing model can be obtained by replacing the embedding of T_t at each time step with an embedding of the input buffer B_t. To train this model, the conditional likelihood of each sequence of actions given the input string is maximized.[5]

5 Inference via Importance Sampling

Our generative model $p(\boldsymbol{x}, \boldsymbol{y})$ defines a joint distribution on trees ($\boldsymbol{y}$) and sequences of words ($\boldsymbol{x}$). To evaluate this as a language model, it is necessary to compute the marginal probability $p(\boldsymbol{x}) = \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y}')$. And, to evaluate the model as a parser, we need to be able to find the MAP parse tree, i.e., the tree $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$ that maximizes $p(\boldsymbol{x}, \boldsymbol{y})$. However, because of the unbounded dependencies across the sequence of parsing actions in our model, exactly solving either of these inference problems is intractable. To obtain estimates of these, we use a variant of importance sampling (Doucet and Johansen, 2011).

Our importance sampling algorithm uses a conditional proposal distribution $q(\boldsymbol{y} \mid \boldsymbol{x})$ with the following properties: (i) $p(\boldsymbol{x}, \boldsymbol{y}) > 0 \implies q(\boldsymbol{y} \mid \boldsymbol{x}) > 0$; (ii) samples $\boldsymbol{y} \sim q(\boldsymbol{y} \mid \boldsymbol{x})$ can be obtained efficiently; and (iii) the conditional probabilities $q(\boldsymbol{y} \mid \boldsymbol{x})$ of these samples are known. While many such distributions are available, the discrim-

[5] For the discriminative parser, the POS tags are processed similarly as in (Dyer et al., 2015); they are predicted for English with the Stanford Tagger (Toutanova et al., 2003) and Chinese with Marmot (Mueller et al., 2013).

inatively trained variant of our parser (§4.4) fulfills these requirements: sequences of actions can be sampled using a simple ancestral sampling approach, and, since parse trees and action sequences exist in a one-to-one relationship, the product of the action probabilities is the conditional probability of the parse tree under q. We therefore use our discriminative parser as our proposal distribution.

Importance sampling uses **importance weights**, which we define as $w(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x}, \boldsymbol{y})/q(\boldsymbol{y} \mid \boldsymbol{x})$, to compute this estimate. Under this definition, we can derive the estimator as follows:

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} q(\boldsymbol{y} \mid \boldsymbol{x}) w(\boldsymbol{x}, \boldsymbol{y}) \\ &= \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})} w(\boldsymbol{x}, \boldsymbol{y}). \end{aligned}$$

We now replace this expectation with its Monte Carlo estimate as follows, using N samples from q:

$$\boldsymbol{y}^{(i)} \sim q(\boldsymbol{y} \mid \boldsymbol{x}) \quad \text{for } i \in \{1, 2, \ldots, N\}$$

$$\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})} w(\boldsymbol{x}, \boldsymbol{y}) \overset{\text{MC}}{\approx} \frac{1}{N} \sum_{i=1}^{N} w(\boldsymbol{x}, \boldsymbol{y}^{(i)})$$

To obtain an estimate of the MAP tree $\hat{\boldsymbol{y}}$, we choose the sampled tree with the highest probability under the joint model $p(\boldsymbol{x}, \boldsymbol{y})$.

6 Experiments

We present results of our two models both on parsing (discriminative and generative) and as a language model (generative only) in English and Chinese.

Data. For English, §2–21 of the Penn Treebank are used as training corpus for both, with §24 held out as validation, and §23 used for evaluation. Singleton words in the training corpus with unknown word classes using the the Berkeley parser's mapping rules.[6] Orthographic case distinctions are preserved, and numbers (beyond singletons) are not normalized. For Chinese, we use the Penn Chinese Treebank Version 5.1 (CTB) (Xue et al., 2005).[7] For the Chinese experiments, we use a single unknown word class. Corpus statistics are given in Table 1.[8]

Table 1: Corpus statistics.

	PTB-train	PTB-test	CTB-train	CTB-test
Sequences	39,831	2,416	50,734	348
Tokens	950,012	56,684	1,184,532	8,008
Types	23,815	6,823	31,358	1,637
UNK-Types	49	42	1	1

Model and training parameters. For the discriminative model, we used hidden dimensions of 128 and 2-layer LSTMs (larger numbers of dimensions reduced validation set performance). For the generative model, we used 256 dimensions and 2-layer LSTMs. For both models, we tuned the dropout rate to maximize validation set likelihood, obtaining optimal rates of 0.2 (discriminative) and 0.3 (generative). For the sequential LSTM baseline for the language model, we also found an optimal dropout rate of 0.3. For training we used stochastic gradient descent with a learning rate of 0.1. All parameters were initialized according to recommendations given by Glorot and Bengio (2010).

English parsing results. Table 2 (last two rows) gives the performance of our parser on Section 23, as well as the performance of several representative models. For the discriminative model, we used a greedy decoding rule as opposed to beam search in some shift-reduce baselines. For the generative model, we obtained 100 independent samples from a flattened distribution of the discriminative parser (by exponentiating each probability by $\alpha = 0.8$ and renormalizing) and reranked them according to the generative model.[9]

Chinese parsing results. Chinese parsing results were obtained with the same methodology as in English and show the same pattern (Table 6).

Language model results. We report held-out per-word perplexities of three language models, both sequential and syntactic. Log probabilities are normalized by the number of words (excluding the stop

[6] `http://github.com/slavpetrov/berkeleyparser`

[7] §001–270 and 440–1151 for training, §301–325 development data, and §271–300 for evaluation.

[8] This preprocessing scheme is more similar to what is standard in parsing than what is standard in language modeling. However, since our model is both a parser and a language model, we opted for the parser normalization.

[9] The value $\alpha = 0.8$ was chosen based on the diversity of the samples generated on the development set.

Table 2: Parsing results on PTB §23 (D=discriminative, G=generative, S=semisupervised).

Model	type	$\mathbf{F_1}$
Henderson (2004)	D	89.4
Socher et al. (2013a)	D	90.4
Zhu et al. (2013)	D	90.4
Vinyals et al. (2015) – WSJ only	D	90.5
Petrov and Klein (2007)	G	90.1
Bod (2003)	G	90.7
Shindo et al. (2012) – single	G	91.1
Shindo et al. (2012) – ensemble	G	92.4
Zhu et al. (2013)	S	91.3
McClosky et al. (2006)	S	92.1
Vinyals et al. (2015) – single	S	92.5
Vinyals et al. (2015) – ensemble	S	92.8
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$	D	89.8
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$	G	92.4

Table 3: Parsing results on CTB 5.1.

Model	type	$\mathbf{F_1}$
Zhu et al. (2013)	D	82.6
Wang et al. (2015)	D	83.2
Huang and Harper (2009)	D	84.2
Charniak (2000)	G	80.8
Bikel (2004)	G	80.6
Petrov and Klein (2007)	G	83.3
Zhu et al. (2013)	S	85.6
Wang and Xue (2014)	S	86.3
Wang et al. (2015)	S	86.6
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$	D	80.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$	G	82.7

symbol), inverted, and exponentiated to yield the perplexity. Results are summarized in Table 4.

7 Discussion

It is clear from our experiments that the proposed generative model is quite effective both as a parser and as a language model. This is the result of (i) relaxing conventional independence assumptions (e.g., context-freeness) and (ii) inferring continuous representations of symbols alongside non-linear models of their syntactic relationships. The most significant question that remains is why the discriminative model—which has more information available to it than the generative model—performs

Table 4: Language model perplexity results.

Model	**test ppl** (PTB)	**test ppl** (CTB)
IKN 5-gram	169.3	255.2
LSTM LM	113.4	207.3
RNNG	102.4	171.9

worse than the generative model. This pattern has been observed before in neural parsing by Henderson (2004), who hypothesized that larger, unstructured conditioning contexts are harder to learn from, and provide opportunities to overfit. Our discriminative model conditions on the entire history, stack, and buffer, while our generative model only accesses the history and stack. The fully discriminative model of Vinyals et al. (2015) was able to obtain results similar to those of our generative model (albeit using much larger training sets obtained through semisupervision) but similar results to those of our discriminative parser using the same data. In light of their results, we believe Henderson's hypothesis is correct, and that generative models should be considered as a more statistically efficient method for learning neural networks from small data.

8 Related Work

Our language model combines work from two modeling traditions: (i) recurrent neural network language models and (ii) syntactic language modeling. Recurrent neural network language models use RNNs to compute representations of an unbounded history of words in a left-to-right language model (Zaremba et al., 2015; Mikolov et al., 2010; Elman, 1990). Syntactic language models jointly generate a syntactic structure and a sequence of words (Baker, 1979; Jelinek and Lafferty, 1991). There is an extensive literature here, but one strand of work has emphasized a bottom-up generation of the tree, using variants of shift-reduce parser actions to define the probability space (Chelba and Jelinek, 2000; Emami and Jelinek, 2005). The neural-network–based model of Henderson (2004) is particularly similar to ours in using an unbounded history in a neural network architecture to parameterize generative parsing based on a left-corner model. Dependency-only language models have also been explored (Titov and Henderson, 2007;

Buys and Blunsom, 2015a,b). Modeling generation top-down as a rooted branching process that recursively rewrites nonterminals has been explored by Charniak (2000) and Roark (2001). Of particular note is the work of Charniak (2010), which uses random forests and hand-engineered features over the entire syntactic derivation history to make decisions over the next action to take.

The neural networks we use to model sentences are structured according to the syntax of the sentence being generated. Syntactically structured neural architectures have been explored in a number of applications, including discriminative parsing (Socher et al., 2013a; Kiperwasser and Goldberg, 2016), sentiment analysis (Tai et al., 2015; Socher et al., 2013b), and sentence representation (Socher et al., 2011; Bowman et al., 2006). However, these models have been, without exception, discriminative; this is the first work to use syntactically structured neural models to generate language. Earlier work has demonstrated that sequential RNNs have the capacity to recognize context-free (and beyond) languages (Sun et al., 1998; Siegelmann and Sontag, 1995). In contrast, our work may be understood as a way of incorporating a context-free inductive bias into the model structure.

9 Outlook

RNNGs can be combined with a particle filter inference scheme (rather than the importance sampling method based on a discriminative parser, §5) to produce a left-to-right marginalization algorithm that runs in expected linear time. Thus, they could be used in applications that require language models.

A second possibility is to replace the sequential generation architectures found in many neural network transduction problems that produce sentences conditioned on some input. Previous work in machine translation has showed that conditional syntactic models can function quite well without the computationally expensive marginalization process at decoding time (Galley et al., 2006; Gimpel and Smith, 2014).

A third consideration regarding how RNNGs, human sentence processing takes place in a left-to-right, incremental order. While an RNNG is not a processing model (it is a grammar), the fact that it is left-to-right opens up several possibilities for developing new sentence processing models based on an explicit grammars, similar to the processing model of Charniak (2010).

Finally, although we considered only the supervised learning scenario, RNNGs are joint models that could be trained without trees, for example, using expectation maximization.

10 Conclusion

We introduced recurrent neural network grammars, a probabilistic model of phrase-structure trees that can be trained generatively and used as a language model or a parser, and a corresponding discriminative model that can be used as a parser. Apart from out-of-vocabulary preprocessing, the approach requires no feature design or transformations to treebank data. The generative model outperforms every previously published parser built on a single supervised generative model in English, and a bit behind the best-reported generative model in Chinese. As language models, RNNGs outperform the best single-sentence language models.

Acknowledgments

We thank Brendan O'Connor, Swabha Swayamdipta, and Brian Roark for feedback on drafts of this paper, and Jan Buys, Phil Blunsom, and Yue Zhang for help with data preparation. This work was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) Information Innovation Office (I2O) under the Low Resource Languages for Emergent Incidents (LORELEI) program issued by DARPA/I2O under Contract No. HR0011-15-C-0114; it was also supported in part by Contract No. W911NF-15-1-0543 with the DARPA and the Army Research Office (ARO). Approved for public release, distribution unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Miguel Ballesteros was supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA).

References

Steven Abney, David McAllester, and Fernando Pereira. 1999. Relating probabilistic grammars and automata. In *Proc. ACL*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine transation by jointly learning to align and translate. In *Proc. ICLR*.

James K. Baker. 1979. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132.

Paul Baltescu and Phil Blunsom. 2015. Pragmatic neural modelling in machine translation. In *Proc. NAACL*.

Dan Bikel. 2004. *On the parameter space of generative lexicalized statistical parsing models*. Ph.D. thesis, University of Pennsylvania.

Rens Bod. 2003. An efficient implementation of a new DOP model. In *Proc. EACL*.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2006. A fast unified model for parsing and sentence understanding. *CoRR*, abs/1603.06021.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Jan Buys and Phil Blunsom. 2015a. A Bayesian model for generative transition-based dependency parsing. *CoRR*, abs/1506.04334.

Jan Buys and Phil Blunsom. 2015b. Generative incremental dependency parsing with neural networks. In *Proc. ACL*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. NAACL*.

Eugene Charniak. 2010. Top-down nearly-context-sensitive parsing. In *Proc. EMNLP*.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14(4).

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proc. EMNLP*.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton, The Hague/Paris.

Arnaud Doucet and Adam M. Johansen. 2011. A tutorial on particle filtering and smoothing: Fifteen years later. In *Handbook of Nonlinear Filtering*. Oxford.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14:179–211.

Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine Learning*, 60:195–227.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proc. ACL*.

Kevin Gimpel and Noah A. Smith. 2014. Phrase dependency machine translation with quasi-synchronous tree-to-tree features. *Computational Linguistics*, 40(2).

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. ICML*.

Joshua Goodman. 2001. Classes for fast maximum entropy training. *CoRR*, cs.CL/0108006.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proc. NAACL*.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proc. ACL*.

Zhongqiang Huang and Mary Harper. 2009. Self-training PCFG grammars with latent annotations across languages. In *Proc. EMNLP*.

Frederick Jelinek and John D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

Mark Johnson. 2001. Joint and conditional estimation of tagging and parsing models. In *Proc. ACL*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Easy-first dependency parsing with hierarchical tree LSTMs. ArXiv:1603.00375.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proc. NAACL*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. Interspeech*.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Process-*

ing, pages 322–332. Association for Computational Linguistics, Seattle, Washington, USA. URL http://www.aclweb.org/anthology/D13-1032.

Terence Parr and Kathleen Fisher. 2011. LL(*): The foundation of the ANTLR parser generator. In *Proc. PLDI*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. NAACL*.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2).

Brian Roark. 2004. Robust garden path parsing. *JNLE*, 10(1):1–24.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proc. IWPT*.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proc. ACL*.

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013a. Parsing with compositional vectors. In *Proc. ACL*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proc. NIPS*.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. EMNLP*.

Guo-Zheng Sun, C. Lee Giles, and Hsing-Hen Chen. 1998. The neural network pushdown automaton: Architecture, dynamics and training. In *Adaptive Processing of Sequences and Data Structures*, volume 1387 of *Lecture Notes in Computer Science*, pages 296–345.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. ACL*.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proc. IWPT*.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. NAACL*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proc. ICLR*.

Zhiguo Wang, Haitao Mi, and Nianwen Xue. 2015. Feature optimization for constituent parsing via neural networks. In *Proc. ACL-IJCNLP*.

Zhiguo Wang and Nianwen Xue. 2014. Joint POS tagging and transition-based constituent parsing in Chinese with non-local features. In *Proc. ACL*.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proc. EMNLP*.

Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. ICML*.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.*, 11(2).

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2015. Recurrent neural network regularization. In *Proc. ICLR*.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1).

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. ACL*.