Recurrent Neural Network Grammars

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.

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 BIBREF0 , BIBREF1 as well as in various conditional language modeling tasks such as machine translation BIBREF2 , image caption generation BIBREF3 , and dialogue generation BIBREF4 . 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 BIBREF5 .

In this paper, we introduce recurrent neural network grammars (RNNGs; § SECREF2), 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 (§ SECREF3). 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 BIBREF6 , BIBREF7 , BIBREF8 , BIBREF9 , BIBREF10 , 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 BIBREF11 . RNNGs maintain the algorithmic convenience of transition-based parsing but incorporate top-down (i.e., root-to-terminal) syntactic information (§ SECREF4).

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 BIBREF7 , BIBREF12 , BIBREF13 , 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 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 (§ SECREF5).

Experiments show that RNNGs are effective for both language modeling and parsing (§ SECREF6). 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 BIBREF7 , BIBREF14 —parsing with the generative model obtains significantly better results than parsing with the discriminative model.

RNN Grammars

Formally, an RNNG is a triple INLINEFORM0 consisting of a finite set of nonterminal symbols (INLINEFORM1), a finite set of terminal symbols (INLINEFORM2) such that INLINEFORM3 , and a collection of neural network parameters INLINEFORM4 . It does not explicitly define rules since these are implicitly characterized by INLINEFORM5 . 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.

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 (§ SECREF4) and then present modifications to turn it into a generator (§ SECREF19).

Parser Transitions

The parsing algorithm transforms a sequence of words INLINEFORM0 into a parse tree INLINEFORM1 using two data structures (a stack and an input buffer). As with the bottom-up algorithm of BIBREF12 , our algorithm begins with the stack (INLINEFORM2) empty and the complete sequence of words in the

input buffer (INLINEFORM3). 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. FIGREF9) is selected by a classifier, based on the current contents on the stack and buffer:

INLINEFORM0 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 INLINEFORM0 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. FIGREF10 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.

Our transition set is closely related to the operations used in Earley's algorithm which likewise introduces nonterminals symbols with its predict operation and later completes them after consuming terminal

symbols one at a time using scan BIBREF15 . It is likewise closely related to the “linearized” parse trees proposed by BIBREF16 and to the top-down, left-to-right decompositions of trees used in previous generative parsing and language modeling work BIBREF11 , BIBREF17 , BIBREF18 .

A further connection is to INLINEFORM0 parsing which uses an unbounded lookahead (compactly represented by a DFA) to distinguish between parse alternatives in a top-down parser BIBREF19 ; however, our parser uses an RNN encoding of the lookahead rather than a DFA.

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 INLINEFORM0 where INLINEFORM1 is the number of open nonterminals on the stack:

The INLINEFORM0 operation can only be applied if INLINEFORM1 is not empty and INLINEFORM2 .

The shift operation can only be applied if INLINEFORM0 is not empty and INLINEFORM1 .

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 INLINEFORM0 or if the buffer is empty.

To designate the set of valid parser transitions, we write INLINEFORM0 .

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 (INLINEFORM0), and (ii) instead of a shift operation there are INLINEFORM1 operations which generate terminal symbol INLINEFORM2 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 INLINEFORM3 and INLINEFORM4 . The algorithm terminates when a single completed constituent remains on the stack. Fig. FIGREF12 shows an example generation sequence.

The generation algorithm also requires slightly modified constraints. These are:

The INLINEFORM0 operation can only be applied if INLINEFORM1 .

The reduce operation can only be applied if the top of the stack is not an open nonterminal symbol and INLINEFORM0 .

To designate the set of valid generator transitions, we write INLINEFORM0 .

This transition set generates trees using nearly the same structure building actions and stack configurations as the “top-down PDA” construction proposed by BIBREF20 , albeit without the restriction that the trees be in Chomsky normal form.

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 INLINEFORM0 and a sequence of symbols INLINEFORM1 , we write INLINEFORM2 to indicate the corresponding sequence of generation transitions, and

INLINEFORM3 to indicate the parser transitions.

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 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.

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 BIBREF12 , BIBREF13 , BIBREF21 .

Generative Model

RNNGs use the generator transition set just presented to define a joint distribution on syntax trees (INLINEFORM0) and words (INLINEFORM1). 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 (INLINEFORM2); i.e., INLINEFORM3

and where action-specific embeddings INLINEFORM0 and bias vector INLINEFORM1 are parameters in INLINEFORM2 .

The representation of the algorithm state at time INLINEFORM0 , INLINEFORM1 , is computed by combining the representation of the generator's three data structures: the output buffer (INLINEFORM2), represented by an embedding INLINEFORM3 , the stack (INLINEFORM4), represented by an embedding INLINEFORM5 , and the history of actions (INLINEFORM6) taken by the generator, represented by an embedding INLINEFORM7 , INLINEFORM8

where INLINEFORM0 and INLINEFORM1 are parameters. Refer to Figure FIGREF26 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 BIBREF22 . 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 (INLINEFORM0) 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 INLINEFORM1 under push and pop operations, we use stack LSTMs BIBREF23 . To represent complex parse trees, we define a new syntactic composition function that recursively defines representations of trees.

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. FIGREF28 .

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 INLINEFORM0 nonlinearity to become the subtree embedding. Because each of the child node embeddings (INLINEFORM2 , INLINEFORM3 , INLINEFORM4 in Fig. FIGREF28) is computed similarly (if it corresponds to an internal node), this composition function is a kind of recursive neural network.

Word Generation

To reduce the size of INLINEFORM0 , word generation is broken into two parts. First, the decision to generate is made (by predicting INLINEFORM1 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 BIBREF26 , BIBREF27 . By using INLINEFORM2 classes for a vocabulary of size INLINEFORM3 , this prediction step runs in time INLINEFORM4 rather than the INLINEFORM5 of the full-vocabulary softmax. To obtain clusters, we use the greedy agglomerative clustering algorithm of BIBREF28 .

Training

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

Discriminative Parsing Model

A discriminative parsing model can be obtained by replacing the embedding of INLINEFORM0 at each time step with an embedding of the input buffer INLINEFORM1 . To train this model, the conditional likelihood of each sequence of actions given the input string is maximized.

Inference via Importance Sampling

Our generative model INLINEFORM0 defines a joint distribution on trees (INLINEFORM1) and sequences of words (INLINEFORM2). To evaluate this as a language model, it is necessary to compute the marginal probability INLINEFORM3 . And, to evaluate the model as a parser, we need to be able to find the MAP parse tree, i.e., the tree INLINEFORM4 that maximizes INLINEFORM5 . 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 BIBREF31 .

Our importance sampling algorithm uses a conditional proposal distribution INLINEFORM0 with the following properties: (i) INLINEFORM1 ; (ii) samples INLINEFORM2 can be obtained efficiently; and (iii) the conditional probabilities INLINEFORM3 of these samples are known. While many such distributions are available, the discriminatively trained variant of our parser (§ SECREF32) 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 INLINEFORM4 . We therefore use our discriminative parser as our proposal distribution.

Importance sampling uses importance weights, which we define as INLINEFORM0 , to compute this estimate. Under this definition, we can derive the estimator as follows: INLINEFORM1

We now replace this expectation with its Monte Carlo estimate as follows, using INLINEFORM0 samples from INLINEFORM1 : INLINEFORM2

To obtain an estimate of the MAP tree INLINEFORM0 , we choose the sampled tree with the highest probability under the joint model INLINEFORM1 .

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.

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 worse than the generative model. This pattern has been observed before in neural parsing by BIBREF7 , 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 BIBREF16 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.

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 BIBREF0 , BIBREF1 , BIBREF44 . Syntactic language models jointly generate a syntactic structure and a sequence of words BIBREF45 , BIBREF46 . 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 BIBREF47 , BIBREF8 . The neural-network–based model of BIBREF7 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 BIBREF9 , BIBREF48 , BIBREF10 . Modeling generation top-down as a rooted branching process that recursively rewrites nonterminals has been explored by BIBREF41 and BIBREF11 . Of particular note is the work of BIBREF18 , 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 BIBREF34 , BIBREF49 , sentiment analysis BIBREF25 ,

BIBREF24 , and sentence representation BIBREF50 , BIBREF51 . 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 BIBREF52 , BIBREF53 . In contrast, our work may be understood as a way of incorporating a context-free inductive bias into the model structure.

Outlook

RNNGs can be combined with a particle filter inference scheme (rather than the importance sampling method based on a discriminative parser, § SECREF5) 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 BIBREF54 , BIBREF55 .

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 BIBREF18 .

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

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).

2pt Chris Dyer INLINEFORM0 Adhiguna Kuncoro INLINEFORM1 Miguel Ballesteros INLINEFORM2 Noah A. Smith INLINEFORM3 INLINEFORM4 School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA INLINEFORM5 NLP Group, Pompeu Fabra University, Barcelona, Spain

INLINEFORM6 Google DeepMind, London, UK INLINEFORM7 Computer Science & Engineering, University of Washington, Seattle, WA, USA {cdyer,akuncoro}@cs.cmu.edu miguel.ballesteros@upf.edu, nasmith@cs.washington.edu [Corrigendum to Recurrent Neural Network Grammars] Due to an implentation bug in the RNNG's recursive composition function, the results reported in Dyer et al. (2016) did not correspond to the model as it was presented. This corrigendum describes the buggy implementation and reports results with a corrected implementation. After correction, on the PTB §23 and CTB 5.1 test sets, respectively, the generative model achieves language modeling perplexities of 105.2 and 148.5, and phrase-structure parsing F1 of 93.3 and 86.9, a new state of the art in phrase-structure parsing for both languages. RNNG Composition Function and Implementation Error The composition function reduces a completed constituent into a single vector representation using a bidirectional LSTM (Figure FIGREF47) over embeddings of the constituent's children as well as an embedding of the resulting nonterminal symbol type. The implementation error (Figure FIGREF47) composed the constituent (NP the hungry cat) by reading the sequence “NP the hungry NP”, that is, it discarded the rightmost child of every constituent and replaced it with a second copy of the constituent's nonterminal symbol. This error occurs for every constituent and means crucial information is not properly propagated upwards in the tree. Results after Correction The implementation error affected both the generative and discriminative RNNGs. We summarize corrected English phrase-structure PTB §23 parsing result in Table TABREF49 , Chinese (CTB 5.1 §271–300) in Table TABREF50 (achieving the the best reported result on both datasets), and English and Chinese language modeling perplexities in Table TABREF51 . The considerable improvement in parsing accuracy indicates that properly composing the constituent and propagating information upwards is crucial. Despite slightly higher language modeling perplexity on PTB §23, the fixed RNNG still outperforms a highly optimized sequential LSTM baseline.