Comparing Top-down and Bottom-up Neural Generative Dependency Models

Austin Matthews and **Graham Neubig**
Language Technologies Institute
Carnegie Mellon University
{austinma,gneubig}@cs.cmu.edu

Chris Dyer
DeepMind
cdyer@google.com

Abstract

Recurrent neural network grammars (RNNGs) generate sentences using phrase-structure syntax and perform very well in terms of both language modeling and parsing performance. However, since dependency annotations are much more readily available than phrase structure annotations, we propose two new generative models of projective dependency syntax, so as to explore whether generative dependency models are similarly effective. Both models use RNNs to represent the derivation history with making any explicit independence assumptions, but they differ in how they construct the trees: one builds the tree bottom up and the other top down, which profoundly changes the estimation problem faced by the learner. We evaluate the two models on three typologically different languages: English, Arabic, and Japanese. We find that both generative models improve parsing performance over a discriminative baseline, but, in contrast to RNNGs, they are significantly less effective than non-syntactic LSTM language models. Little difference between the tree construction orders is observed for either parsing or language modeling.

1 Introduction

Recurrent neural network grammars (Dyer et al., 2016, RNNGs) are syntactic language models that use predicted syntactic structures to determine the topology of the recurrent networks they use to predict subsequent words. Not only can they learn to model language better than non-syntactic language models, but the conditional distributions over parse trees given sentences produce excellent parsers (Fried et al., 2017).

In this paper, we introduce and evaluate two new dependency syntax language models which are based on a recurrent neural network (RNN) backbone (§2).[1] Dependency syntax is particularly appealing as many more languages have dependency treebanks (e.g. the Universal Dependencies Project (Nivre et al., 2017)) than have large numbers of phrase structure annotations.

Like RNNGs, our proposed models predict structure and words jointly, and the predicted syntactic structure is used to determine the structure of the neural network that is used to represent the history of actions taken by the model and to make a better estimate of the distribution over subsequent structure-building and word-generating actions. Because we use RNNs to encode the derivation history, our models do not make any explicit independence assumptions, but instead condition on the complete history of actions. The two proposed models do, however, differ in the order that they construct the trees. The first model operates top down (§2.1), starting at the root and recursively generating dependents until the last modifier has been generated. The second operates bottom up (§2.2), generating words from left to right and interleaving decisions about how they fit together to form tree fragments and finally a fully formed dependency tree.[2]

Because neither model makes explicit independence assumptions, given enough capacity, infinite data, and a perfect learner, both models would converge to the same estimate. However, in our limited, finite, and imperfect world, these two models will impose different biases on the learner: in one order, relevant conditioning information may be more local (which could mean the neural networks have an easier time learning to exploit the relevant information rather than becom-

[1] We release code for these two models, which can be found at https://github.com/armatthews/dependency-lm.

[2] In this work, we limit ourselves to models that are capable only of generating projective dependency trees.

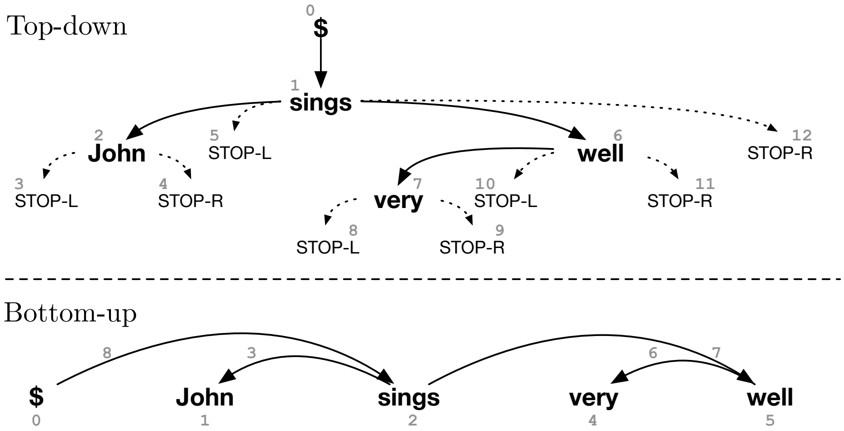


Figure 1: Generation process for the same dependency tree under the top-down and bottom-up models. As the indices of the generation events show, the top-down model generates recursively from the root, whereas the bottom-up model generates from left to right.

ing distracted by accidental correlations), while in the other it may be more distant. These differences thus imply that the two models will have different structural biases, but it is not at all clear whether one should out perform the other. We therefore explore to what extent this choice of construction order affects performance, and we evaluate the proposed models on language modeling and parsing tasks across three typologically different languages (§3).

Our findings (§4) show that, like RNNGs, generative dependency models make good parsers. Given the small scale of the Universal Dependency corpora, this result is also in line with previous work which shows that joint generative models offer very sample-efficient estimates of conditional distributions (Yogatama et al., 2017). Second, we find that both dependency models are *less effective* as language models than phrase structure RNNGs or than standard LSTM language models. This negative result is not entirely surprising. Although information about syntactic dependencies seems intuitively that it would be helpful for defining good conditioning contexts for language models, since its earliest days (Tesnière, 1959), work on dependency syntax has largely focused on discriminative models of *existing* sentences. In contrast, the phrase structure annotations found in, e.g., the Penn Treebank that were used to demonstrate improved language modeling performance with RNNGs are indebted to linguistic theories (e.g., government and binding theory, X-bar theory) which are broadly concerned with determining which sentences are grammatical and which are not—a crucial aspect of language modeling (Marcus et al., 1993). Finally, we observe only minimal differences in language modeling performance for top-down and bottom-up models. This result is surprising in light of how different the estimation problems are, but it is a clear demonstration of the ability of RNNs to learn to extract relevant features from data presented in any different but consistent orders.

2 Models

We present two models for jointly generating projective dependency trees and sentences. The processes are illustrated in Fig. 1. The first is a top-down model (§2.1), which starts by generating the root of the sentence, and then recursively generating its left and right modifiers. The second is a bottom-up model (§2.2), which generates terminals in a left to right order. In both cases, there is a deterministic mapping from well-formed sequences of generation actions into dependency trees. Following convention in parsing literature, we refer to such action sequences as oracles.

Both models both are parameterized with recursively structured neural networks that have access to the complete history of generation events. Thus, the factorization of the tree probability is justified by the chain rule. However, because of the difference in build orders, the conditional probabilities being estimated are quite different, and we thus expect these models might be more or less effective at either language modeling or (when used in conjunction with Bayes' rule) parsing.

To illustrate the different estimation problems posed by the two models, consider the first generation event in both cases. In the top-down model, the root word (usually the main verb of the sentence) is generated first; whereas in the bottom-up model, the first probability modeled is the probability of the first word in the sentence. Also, in the top-down model a verb always generates its dependentents (which has implications for how agreement is modeled), whereas in the bottom-up model, it the left dependents (whatever their function) will be generated first, and then the verb generation will be conditional on them. Again, we emphasize that these differences potentially result in differences in the difficulty of the estimation problem (or how much capacity the model needs to represent accurate conditionals), but do not impact the expressivity or correctness of the models.

2.1 Top-Down

Our first model is a top-down model. The model begins with an empty root node.[3] Starting from the root the model recursively generates child nodes using its GEN action. When the model chooses the GEN action it then selects a new head word from its vocabulary and creates a new node descended from the most recent open constituent. Each node created in this way is pushed onto a stack of open constituents, and begins creating its left children.

To indicate that the current node (i.e. the top node in the stack) is done generating left children the model takes its STOP-L ("stop left") action, after which the current node begins generating its right children. Analogously, to indicate that the current node is done generating right children the model selects its STOP-R ("stop right") action. With this the current constituent is complete and thus popped off the stack and attached as a child of the new top-most constituent. See Figure 2 for examples of the effects of each of these three actions on a partially built tree structure and Algorithm 1 for a sketch of their implementation.

At each decision point the model conditions on the output of an LSTM over the partially completed constituents on the stack, beginning with the root and ending with the top-most constituent. The result is passed through an MLP and then a softmax that decides which action to take next (Figure 3). If the model chooses the GEN action, the hidden vector from the MLP is used to separately choose a terminal.

Algorithm 1 Top-Down Tree Generation

```
procedure EMBEDTREE(node)
    $\mathbf{state} = lstm\_initial\_state$
    for child in node do
        if child is terminal then
            $\mathbf{state}$.add($\mathbf{WordEmbs}[child]$)
        else
            $\mathbf{state}$.add(EMBEDTREE($child$))
    return state
procedure PICKNEXTACTION(stack)
    $\mathbf{h} = \text{MLP}_{\text{action}}(\text{EmbedTree}(stack))$
    $action \sim \text{softmax}(\mathbf{h})$
    return action
procedure PICKWORD(stack)
    $\mathbf{h} = \text{MLP}_{\text{word}}(\text{EmbedTree}(stack))$
    $word \sim \text{softmax}(\mathbf{h})$
    return word
procedure GENERATENODE(stack)
    $action =$ PICKNEXTACTION($stack$)
    if action == GEN then
        word = PICKWORD(STACK)
        stack.push(new Node(word))
    else if action == STOP-L then
        stack.back().add_child(STOP-L)
    else if action == STOP-R then
        stack.back().add_child(STOP-R)
        $\mathbf{child\_emb} = stack.pop()$
        stack.back().add_child($\mathbf{child\_emb}$)
```

To embed each subtree on the stack we use another LSTM. First we feed in the head word of the constituent, followed by the embeddings of each of the constituent's children, including the special STOP-L and STOP-R symbols. We then additionally add a gated residual connection from the head word to final subtree representation to allow salient information of the head word to be captured without needing to pass through an arbitrary number of LSTM steps (Figure 4).

2.2 Bottom-Up

Our second model generates sentences bottom-up, in the same manner as a shift-reduce parser. A sentence is modeled as a series of actions (related to the arc-standard transitions used in parsing (Nivre, 2013)) that manipulate a stack of embedded tree fragments. There are three types of actions: SHIFT(x), which pushes a new terminal x onto the stack, REDUCE-L, which combines the two top elements on the stack into one single sub-

[3] The root node may never have left children. In this way it is though the root node has already generated its STOP-L, though this step is not explicitly modelled

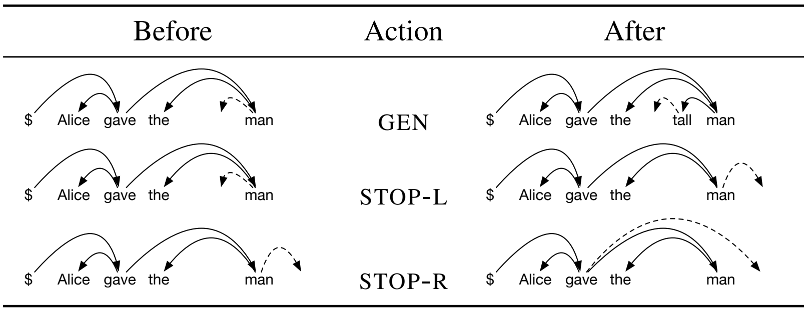


Figure 2: Examples of the three actions of our top-down model. The dotted arrow indicates where the new word will go if the GEN action is chosen next. GEN creates a new terminal node and moves the dotted arrow to point to the left of the new token. STOP-L moves the dotted arrow from the left of the current token to the right thereof. STOP-R moves the dotted arrow from the right of the current token back up to its parent node.

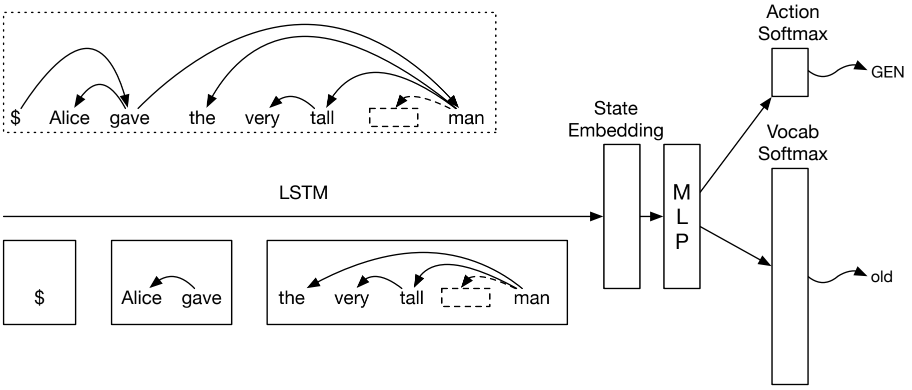


Figure 3: To encode the history of generation events in the top-down process, we use an LSTM over *subtree embeddings* (See Figure 4). The LSTM proceeds from the root of the tree down to the most recent open node. Each item in the LSTM is an embedding of a word and its already generated descendants. STOP symbols have been suppressed for clarity.

tree with the left of the two as the head (i.e. with a leftward arrow), and REDUCE-R which again combines the top two elements of the stack, this time making the right one the head. See Figure 5 for examples of how these three actions affect the stack of partially built tree structures during the parsing of an example sentence.

At each time step the model conditions on the state of the stack using an LSTM running over entries from oldest to newest. The resulting vector $\boldsymbol{h}$ is then passed through an MLP, and then a softmax over the three possible action types. If the SHIFT action is taken, the vector $\boldsymbol{h}$ is re-used and passed through a separate MLP and softmax over the vocabulary to choose an individual word to generate. If one of the two REDUCE actions is chosen, the top two elements from the stack are popped, concatenated (with the head-to-be first, followed by the child), and passed through an MLP. The result is a vector representing a new subtree that is then pushed onto the stack. Kuncoro et al. (2017) showed that this type of stack-based representation alone is sufficient for language modeling and parsing, and indeed that more involved models actually damage model performance. See Figure 6 for an example of how this bottom-up model chooses an action.

2.3 Marginalization

Traditionally a language model takes a sentence $\boldsymbol{x}$ and assigns it a probability $p(\boldsymbol{x})$. Since our syntax-based language models jointly predicts the probability $p(\boldsymbol{x}, \boldsymbol{y})$ of a sequence of terminals $\boldsymbol{x}$ and a tree $\boldsymbol{y}$, we must marginalize over trees to get the total probability assigned to a sentence $\boldsymbol{x}$, $p(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{T}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y})$, where $\mathcal{T}(\boldsymbol{x})$ represents the set of all possible dependency trees over a sentence $\boldsymbol{x}$. Unfortunately the size of $\mathcal{T}(\boldsymbol{x})$ grows exponentially in the length of $\boldsymbol{x}$, making explicit marginalization infeasible.

Instead we use importance sampling to approximate the marginal (Dyer et al., 2016). We use the

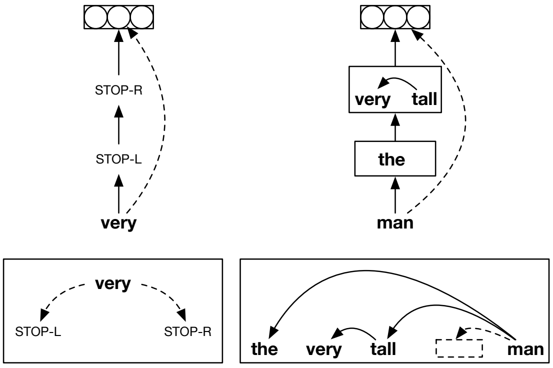


Figure 4: Examples of embedding two subtrees in the top-down model. A subtree is embedded using an LSTM over its child subtrees (solid lines) with a gated residual connection from the root word to the final embedding (dotted lines).

parser of Dyer et al. (2015), a discriminative neural stack-LSTM-based bottom-up parser, as our proposal distribution $q(\boldsymbol{x}, \boldsymbol{y})$ and compute the approximate marginal using $N = 1000$ samples per sentence: $p(\boldsymbol{x}) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(\boldsymbol{x},\boldsymbol{y})}{q(\boldsymbol{x},\boldsymbol{y})}$.

2.4 Parsing Evaluation through Reranking

In order to evaluate our model as a parser we would ideally like to efficiently find the MAP parse tree given an input sentence. Unfortunately, due to the unbounded dependencies across the sequences of actions used by our models this inference is infeasible. As such, we instead rerank a list of 1000 samples produced by the baseline discriminative parser, a combination process that has been shown to improve performance by combining the different knowledge learned by the discriminative and generative models (Fried et al., 2017).

For each hypothesis parse in the sample list we query the discriminative parser, our top-down model, and our bottom-up model to obtain a score for the parse from each. We combine these scores using weights learned to optimize performance on the development set (Och, 2003).

3 Experimental Setup

Our primary goal is to discover whether dependency-based generative neural models are able to improve the performance of their discriminative brethren, as measured on parsing and language modeling tasks. We also seek to determine the effect construction order and the biases implicit therein has on performance on these two tasks. To this end, we test a baseline discriminative parser, our two models, and all combinations of these three models on a parsing task in several languages, and we test a baseline and our two models' performance on a language modeling task on the same set of languages.

3.1 Data Sets

We use the Universal Dependency corpora (Nivre et al., 2017) for three languages with very different structures: English, Japanese, and Arabic, as provided for the 2017 CoNLL shared task on universal dependency parsing. In all languages we convert all singleton terminal symbols to a special UNK token. See Table 1 for details regarding the size of these data sets.

For language modeling we evaluate using the gold sentence segmentations, word tokenizations, and part of speech tags given in the data. For parsing, we evaluate in two scenarios. In the first, we train and test on the same gold-standard data using in our language modeling experiments. In the second, we again train on gold data, but use UDPipe (Straka and Straková, 2017) to segment, tokenize, and POS tag the dev and test sets starting from raw text, following the default scenario and most participants in the CoNLL 2017 shared task.

3.2 Baseline Models

On the language modeling task we compare against a standard LSTM-based language model baseline (Mikolov et al., 2010), using 1024-dimensional 2-layer LSTM cells, and optimized using Adam (Kingma and Ba, 2014).

For the parsing task we compare against the discriminative parser of Dyer et al. (2015), a bottom-up transition-based parser that uses stack-LSTMs, as well as the overall top system (Dozat et al., 2017) from the 2017 CoNLL shared task on multilingual dependency parsing (Zeman et al., 2017). That work uses a discriminative graph-based parser that uses a biaffine scoring function to score each potential arc. Moreover, it uses character-level representations to deal with morphology and a PoS tagger more sophisticated than UDPipe – two major changes from the shared task's default pipeline. These two differences afford them a substantial advantage over our approach which only modifies the parsing step of the pipeline.

Finally, we show the results of an oracle system looking at the 1000-best lists used for our reranking experiments. Note that since this oracle system

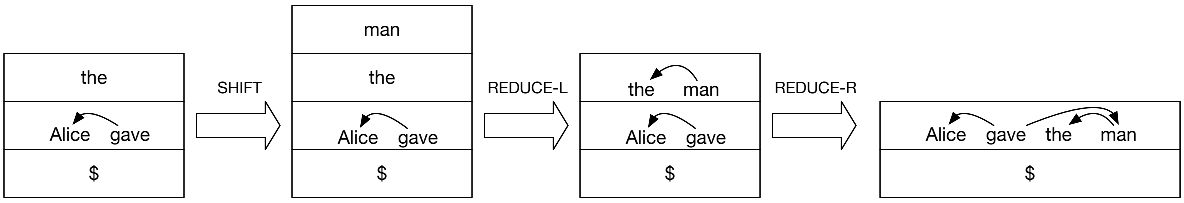


Figure 5: Examples of the three actions of our bottom-up model and their effects on the internal stack. SHIFT adds a new terminal to the top of the stack. REDUCE-L combines the top two elements of the stack with a left arc from the head of the top-most element to the head of the second element. REDUCE-R combines the top two elements with a right arc from the head of the second element to the head of the top-most element.

		Train		Dev		Test		Vocab
Language	Words	Sents	Words	Sents	Words	Sents	Singletons	Non-S'tons
English	204585	12543	25148	2002	25096	2077	9799	9873
Japanese	161900	7164	11556	511	12615	557	13091	9222
Arabic	223881	6075	30239	909	28264	680	9907	13242

Table 1: Statistics of the universal dependency data sets used in this paper. Size of the train, dev, and test sets are given in tokens. Vocabulary information is number of types.

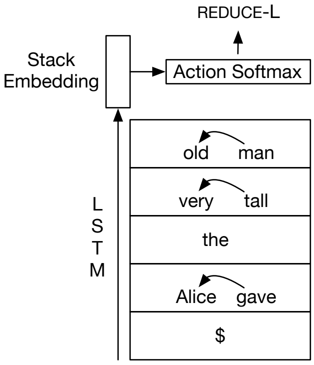


Figure 6: Our bottom-up model emulates a shift-reduce parser and maintains an explicit stack. At each timestep, we use the output of an LSTM over the stack to choose the next action, which is then executed to produce a new stack state.

is constrained to using only this list of samples it is not able to achieve 100% parsing accuracy.

3.3 Hyperparameters

All models use two-layer 1024-unit LSTMs and 1024-dimensional word/action embeddings. All other MLPs have a single hidden layer, again with 1024 hidden units. We implement all models using DyNet (Neubig et al., 2017), and train using Adam (Kingma and Ba, 2014) with a learning rate of 0.001, dropout with $p = 0.5$, and minibatches of 32 sentences. We evaluate the model on a held out dev set after 150 updates, and save the model to disk whenever the score is a new best. All other settings use DyNet defaults.

4 Results

Parsing results Results on the parsing task can be found in Table 2. We observe that in English with the gold-standard preprocessing our models perform particularly well, showing an improvement of 1.16% UAS F1 for the top-down and 0.82% UAS F1 for the bottom-up model when individually combined with our discriminative parser. Combining all three models together gives a total of 1.46% absolute improvement over the baseline, indicating that the models capture knowledge lacking in the baseline model, and knowledge that is complementary to each other.

The story is similar in Japanese and Arabic, though the gains are smaller in Japanese. We hypothesize that this is due to the fact that parsing Japanese is relatively easy because of its strict head-final and left-branching nature, and thus our baseline is already a remarkably strong parser. This hypothesis is backed up by the fact that the baseline parser alone is only 3-4% UAS away from the oracle by itself, compared to about 10% away on English and Arabic. Thus our relative improvement, measured in terms of the percentage of possible improvement achieved, is quite consistent across the three languages, at roughly 13%.

Results on the test set using UDPipe's noisy preprocessing also saw encouraging results from

		English				Japanese				Arabic			
		Gold → Gold		Gold → UDPipe		Gold → Gold		Gold → UDPipe		Gold → Gold		Gold → UDPipe	
Model	Reranked?	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
CoNLL Baseline	✗	-	-	-	79.24	-	-	-	74.40	-	-	-	70.14
Dozat et al. (2017)	✗	-	-	-	84.74	-	-	-	75.42	-	-	-	76.59
Disc (Greedy)	✗	87.00	85.92	78.85	78.12	96.16	95.20	76.67	75.70	82.14	82.39	69.74	70.34
Disc (Reranked)	✓	87.48	86.30	78.99	78.23	96.04	95.17	76.66	75.58	81.70	81.34	69.20	68.79
Top-Down	✓	82.94	82.48	76.73	76.88	92.99	92.51	74.84	74.18	80.99	80.85	69.78	69.64
Bottom-Up	✓	83.11	82.70	76.79	77.13	94.56	93.25	75.85	74.18	80.61	80.70	69.30	69.24
Disc + TD	✓	88.47	87.46	80.46	79.56	96.07	95.43	76.59	74.62	82.87	82.37	70.35	70.25
Disc + BU	✓	88.29	87.12	80.09	79.33	96.17	95.54	76.82	75.92	82.48	82.18	70.18	69.99
TD + BU	✓	84.93	84.56	78.71	78.72	94.87	94.03	76.15	75.30	81.84	81.56	70.52	70.03
Disc + TD + BU	✓	**88.74**	**87.76**	**80.49**	**80.22**	**96.18**	**95.58**	**76.86**	**75.98**	**83.06**	**82.58**	**70.85**	**70.40**
Oracle	✓	97.68	97.27	91.07	90.24	99.39	99.25	79.67	80.34	91.20	89.06	77.75	76.17

Table 2: Results of parsing using our baseline discriminative parser, our two generative models, combinations thereof, and two contrastive systems from the CoNLL 2017 shared task. Scores in bold are the highest of our models. Note that Dozat et al. (2017) use substantially different preprocessing. See §3.2 for details.

		$p(\boldsymbol{x}, \boldsymbol{y})$		$p(\boldsymbol{x})$	
Lang.	Model	Dev	Test	Dev	Test
EN	RNNLM	-	-	**5.24**	**5.18**
	Top-Down	5.80	5.72	5.73	5.66
	Bottom-Up	5.63	5.56	5.53	5.47
JA	RNNLM	-	-	**4.41**	**4.58**
	Top-Down	4.82	5.00	4.73	4.93
	Bottom-Up	4.83	5.03	4.75	4.95
AR	RNNLM	-	-	**5.42**	**4.34**
	Top-Down	6.08	6.23	5.98	4.79
	Bottom-Up	6.11	6.21	5.94	4.75

Table 3: Language modeling cross entropy of our model and an RNNLM baseline. Lower is better. All scores are expressed in nats.

the three-model ensemble gaining 1.99%, 0.40%, and 1.61% on English, Japanese, and Arabic respectively, solidly outperforming the 2017 CoNLL shared task baselines across the board, and beating Dozat et al. (2017), the overall shared task winner's, submission on Japanese.

Of particular note is that on both the gold and non-gold data, and across all three languages, the performance of the top-down and bottom-up models is quite similar; neither model consistently outperforms the other. In Japanese we do find the bottom-up parser beats the top-down one when used alone, but when combined with the discriminative model the lead evaporates, and in both of the other languages there is no clear trend.

These results are consistent with Fried et al. (2017) that has shown that generative models are particularly good at improving discriminative models through reranking, as they have an effect similar to ensembling dissimilar models.

Language modeling results We find that despite successes on parsing, our dependency models are not empirically suitable for language modeling. Table 3 shows the performance of our models on the language modeling task. Across all three languages, both of our models underperform a baseline RNNLM by a consistent margin of about 0.5 nats per word.

Again we note that the two models perform remarkably similarly, despite their completely different construction orders, and thus the completely different sets of information they condition on at each time step. Again neither model is a clear overall victor, and in each individual language the models are extremely close in performance.

5 Analysis

One of our most intriguing findings is that our two proposed models perform remarkably similarly in spite of their differing construction orders. One would naturally assume that the differing orders, as well as the wildly different history information available at each decision point, would lead to performance differences. We seek to hone in on *why* the two models' performances are so similar.

Unfortunately the fact that the models use different conditioning contexts makes direct comparison of sub-sentential scores impossible. The top-down model, which generates the verb before its subject noun, may have large entropy when choosing the verb, but an easier time choosing the subject since it can condition on the verb limiting its choices to appropriate semantic classes, person,

	Structure	Terminals
Top-Down	4.97	67.9
Bottom-Up	7.42	63.3

Table 4: Our models' average negative log likelihoods on the English dev set broken down into structure and terminal components

number, et cetera. The bottom-up model, on the other hand, will generate the subject noun from the entire list of possible nouns first, and then will focus its probability on relevant and agreeing verb forms when generating the verb.

To this end we plot the scores (i.e. the negative log probabilities) the models assign to each gold tree in the English dev set. The raw scores between the top-down and bottom-up models are highly correlated (Pearson's $r = 0.995$), largely due to the fact that longer sentences naturally have lower probabilities than shorter sentences. As such, we examine length-normalized scores, dividing each sentence's score by its length. The results are still largely correlated ($r = 0.88$), with a few outliers, all of which are very short (<3 tokens) sentences.

We hypothesize that much of this correlation stems from the fact that for a given sentence both models must generate the same sequence of terminal symbols. Some sentences will have rare sequences of terminals while others have more common words, leading to an obvious, but perhaps uninformative, correlation. To examine this possibility we factor our models' scores into a *terminal* component and a *structure* component so that the overall negative log likelihood of a sentence is decomposed as $\text{NLL} = \text{NLL}_{\text{terminals}} + \text{NLL}_{\text{structure}}$. We then examine the correlation between the two models' scores' terminal components and their structure components separately. We find that the terminal components are still strongly correlated ($r = 0.91$), while the structure components are largely uncorrelated ($r = 0.09$), hinting that information the two models learned about the correct structure of English sentences differs. See Appendix Figure 7 for a visual representation of these data. Overall the top-down model also assigns much higher probabilities to correct structures, but lower probabilities to the correct terminal sequences (Table 4).

6 Related Work

Most work on discriminative dependency parsing follows the bottom-up paradigm (Nivre, 2003; Nivre et al., 2007; Dyer et al., 2015; Kiperwasser and Goldberg, 2016), but top-down models have also shown some promise (Zhang et al., 2015).

Generative dependency models go back to Hays (1964), but most existing such models (Buys and Blunsom, 2015; Jiang et al., 2016) have relied on independence assumptions whether used for parsing, unsupervised dependency induction, or language modeling. Buys and Blunsom (2015) also describe a generative bottom-up neural parser, but use hand-crafted input features and limit the model to third-order features. Titov and Henderson (2010) explore a generative parsing model with no independence assumptions based on sigmoid belief networks (Neal, 1992) instead of RNNs.

The CoNLL 2017 shared task saw many different models succeed at parsing Universal Dependencies. Most of the top contenders, including the best scoring systems on the languages discussed in this work, use discriminative models.

Kanayama et al. (2017) had tremendous success on Japanese using a wildly different approach. They train a model to identify likely syntactic heads, then assume that all other words simply attach in a left-branching structure, which works due to the strictly head-final nature of Japanese.

Dozat et al. (2017) train a discriminative neural parser which uses a BiLSTM to generate hidden representations of each word (Kiperwasser and Goldberg, 2016). These representations are used to score arcs, which are greedily added to the tree.

Björkelund et al. (2017) perform best on Arabic, using an ensemble of many different types of bottom-up discriminative parsers. They have each of twelve parsers score potential arcs, learn a weighting function to combine them, and use the Chu-Liu-Edmonds algorithm (Chu, 1965; Edmonds, 1967) to output final parses.

All three of these discriminative models are very effective for analysis of a sentence, none of them are able to be converted into a similar generative model. At best, the biaffine model of Dozat et al. (2017) could generate a bag of dependencies without order information, which makes it impractical as the basis for a generative model.

There has been past work on building recurrent neural models that condition on the buffer to make parsing decisions in a shift-reduce parser. Henderson (2004) was among the first to introduce such a model. They introduce both a generative and discriminative model based on Simple Synchrony Networks (Lane and Henderson, 1998), and use a pre-cursor to attention mechanisms to choose which previous states are most relevant at the current timestep. More recently Dyer et al. (2015) created a similar model based on stack LSTMs.

There has also been past work on language modelling with generation orders other than the typical left-to-right. Ford et al. (2018) examine a variety of possibilities, but stop short of syntax-aware oderings. Buys and Blunsom (2018) investigate neural language models with latent dependency structure, also concluding that while dependencies perform well on parsing they underperform for language modelling.

7 Conclusion

In this paper we test our hypothesis that dependency structures can improve performance on language modeling and machine translation tasks, in the same way that constituency parsers have been shown to help. We conclude that generative dependency models do indeed make very good parsing models, and, as has been observed in phrase structure parsing, combining a generative dependency parser with a traditional discriminative one does indeed improve parsing performance. We however also find using dependency models information to structure the intermediate representations in language modeling does not easily lead to better outcomes.

This pattern of results suggests that while dependencies may be a useful tool for text analysis, but are less suited to characterizing a generation process of sentences than phrase structure grammars area. Finally, we find that the choice of top-down or bottom-up construction order affects performance minimally on both the parsing and language modeling tasks despite the large differences in the local conditioning contexts of each action choice.

Acknolwedgements

This work is sponsored in part by Defense Advanced Research Projects Agency Information Innovation Office (I2O). Program: Low Resource Languages for Emergent Incidents (LORELEI). Issued by DARPA/I2O under Contract No. HR0011-15-C0114. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

Anders Björkelund, Agnieszka Falenska, Xiang Yu, and Jonas Kuhn. 2017. Ims at the conll 2017 ud shared task: Crfs and perceptrons meet neural networks. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51.

Jan Buys and Phil Blunsom. 2015. Generative incremental dependency parsing with neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 863–869.

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. Association for Computational Linguistics.

Yoeng-Jin Chu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30.

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*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of NAACL-HLT*, pages 199–209.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71:233–240.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George Dahl. 2018. The importance of generation order in language modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2942–2946.

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.

David G Hays. 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 95. Association for Computational Linguistics.

Yong Jiang, Wenjuan Han, and Kewei Tu. 2016. Unsupervised neural dependency parsing. In *Proc. EMNLP*.

Hiroshi Kanayama, Masayasu Muraoka, and Katsumasa Yoshikawa. 2017. A semi-universal pipelined approach to the conll 2017 ud shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 265–273.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

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.

Peter CR Lane and James B Henderson. 1998. Simple synchrony networks: Learning to parse natural language with temporal synchrony variable binding. In *ICANN 98*, pages 615–620. Springer.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Radford M Neal. 1992. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer.

Joakim Nivre. 2013. Transition-based parsing.

Joakim Nivre, Lars Ahrenberg Željko Agic, et al. 2017. Universal dependencies 2.0. lindat/clarin digital library at the institute of formal and applied linguistics, charles university, prague.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Lucien Tesnière. 1959. Eléments de linguistique structurale. *Paris, Klincksieck*, 2.

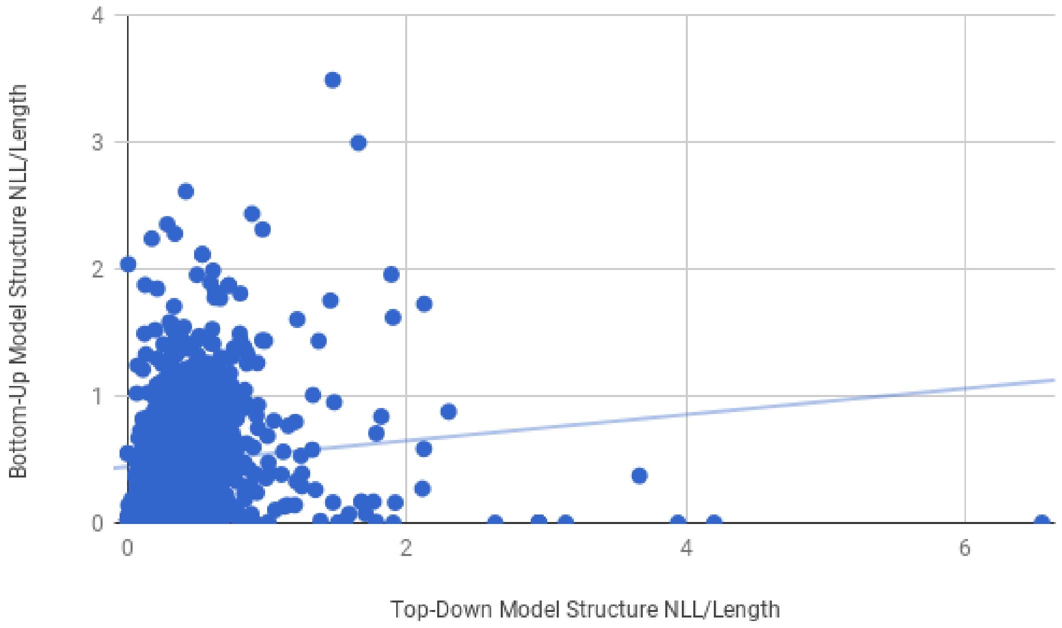
Ivan Titov and James Henderson. 2010. A latent variable model for generative dependency parsing. In *Trends in Parsing Technology*, pages 35–55. Springer.

Dani Yogatama, Chris Dyer, Wang Ling, and Phil Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *arXiv preprint arXiv:1703.01898*.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, et al. 2017. Conll 2017 shared task: multilingual parsing from raw text to universal dependencies. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015. Top-down tree long short-term memory networks. In *Proc. NAACL*.

Graphs



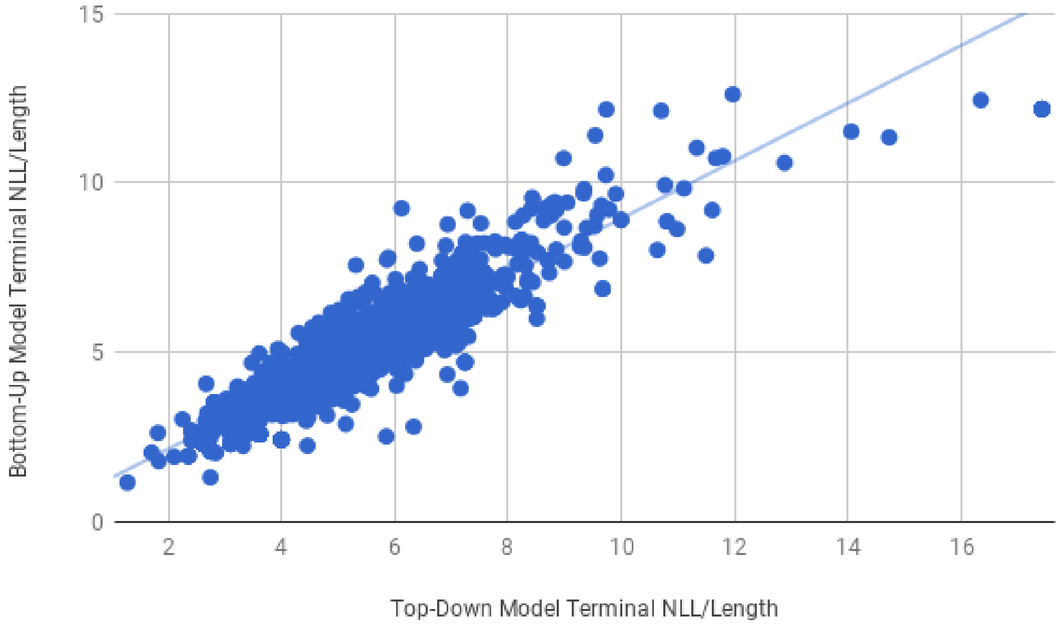


Figure 7: Analysis of the structure (top) and terminal (bottom) scores of our two models' performance on the English development set. We find that the structure scores are not correlated, while the terminal scores of the two models are highly correlated.