EXPLOITING LONG-DISTANCE CONTEXT IN TRANSITION-BASED DEPENDENCY PARSING WITH RECURRENT NEURAL NETWORKS

Jon Gauthier, Danqi Chen & Christopher D. Manning
Department of Computer Science
Stanford University
Stanford, CA 94305, USA
{jgauthie,danqi,manning}@cs.stanford.edu

ABSTRACT

We augment a state-of-the-art dependency parser with sentence-level knowledge using a recurrent neural network language model. These encodings of sentence-level knowledge produced by the RNNLM help to correct a key shortcoming of the transition-based parser, which otherwise cannot make use of long-distance information in making parsing decisions. We demonstrate quantitative performance increases over a competitive baseline model on the Penn Treebank, and give a qualitative analysis of the effect of the fixed-length encodings on parser output.

1 INTRODUCTION

Traditionally, natural language parsing was dominated by dynamic programming methods using a parse chart, so that various hypotheses could be evaluated and combine with some amount of global information about the whole sentences being used to aid the search for a good parse. However, a combination of the desire for high-speed parsing for web-scale application and the development of machine learning approaches which are very good at predicting parse decisions has led to the decline of these methods, and a growing dominance of greedy transition-based parsers, mainly dependency parsers, which decide parser moves using a classifier which examines a local context.

These parsers (also called shift-reduce parsers; MaltParser, (Nivre et al., 2006) is a canonical example) are fast, have a high accuracy, and are hence very widely used. Nevertheless, they have two key defects. One is that they were traditionally based on the use of classifiers which used millions of sparse categorical features (both elementary features, and many interaction term features which are crucial to good parsing performance). While this approach has basically been workable, it suffers from the usual problems of sparseness, statistical inefficiency, and slow feature computation. These problems were considerably addressed in Chen & Manning (2014), who show how a neural network dependency parser over distributed representations can give higher quality parses more quickly than standard feature-based greedy dependency parsers.

However, this work did not address a second important problem, which is that the classifier-based parsers are greedily making decisions based only on a very local context. While such an approach is very effective in terms of speed, it is well-known that it leads to these methods being less accurate in incorporating long-distance context and in making more global decisions about long distance dependencies (McDonald & Nivre, 2007). In this paper, we address this problem by using a recurrent neural network (RNN) to provide a global view of upcoming sentence context to help guide the decisions of the dependency parser. At any point in the parse, the remainder of the sentence can be summarized by the state of a long short-term memory (LSTM) neural unit, where the recurrent model is run backwards from the end of the sentence, so that each state summarizes the remainder of the sentence at that point. This hidden state is then incorporated into a feed-forward neural network that determines a sequence of parse decisions inside a shift-reduce parsing architecture.

In this paper, we introduce transition-based dependency parsing and previous work on using neural network approaches (Section 2), analyze performance problems in such parsers (Section 3), develop a novel parsing model that incorporates an RNN to model future context (Section 3) and then provide

experiments showing the effectiveness of this model (Section 5), and finally do some analysis of the results (Section 5). Our major contributions are emphasizing the need to model global context in making local decisions, introducing a new parsing model exploiting RNNs that does this, and showing modestly improved parsing numbers as a result.

2 PREVIOUS WORK: TRANSITION-BASED DEPENDENCY PARSING

A *transition-based* dependency parser (Yamada & Matsumoto, 2003; Zhang & Clark, 2009) predicts a full dependency parse tree by proceeding left to right over the words of an input sentence, making greedy parsing decisions at each word. Such parsers are very efficient, but often sacrifice some amount of accuracy in making greedy local decisions.

Our parser is based on the transition-based *arc-standard* system (Nivre & Scholz, 2004). The parser is formally defined as a three-tuple (S, B, A), where S is a *stack* of partial parses, B is a *buffer* of words yet to be parsed, and A is a set of arc labels produced so far by the parser. The parser's initial configuration for a sequence of words $w_1, \ldots, w_n$ is $S = [], B = [w_1, \ldots, w_n], A = \emptyset$. For any configuration three transition types are available:[1]

1. SHIFT: Pop the first element from the buffer and push it onto the stack. (Buffer must be nonempty.)
2. LEFTARC(l): Create an arc with label l from the first item on the stack to the second. Removes the second item from the stack. (Stack must have at least two items.)
3. RIGHTARC(l): Create an arc with label l from the second item on the stack to the first. Removes the first item from the stack. (Stack must have at least two items.)

The LEFTARC and RIGHTARC transitions are both parameterized by a dependency label l. Our system will thus have $2L + 1$ possible transitions, where L is the number of different relations in the dependency grammar being learned. For more details on the arc-standard system, see Nivre & Scholz (2004).

2.1 PREVIOUS WORK: NEURAL NETWORK DEPENDENCY PARSING

In this paper, we extend the neural-network model of Chen & Manning (2014), which achieved state-of-the-art accuracy among greedy dependency parsers. The key change in this parser compared to prior work was the use of *dense* feature representations rather than categorical inputs or conjunctions of such inputs. This change in feature representation helped to relieve issues in data sparsity and runtime.

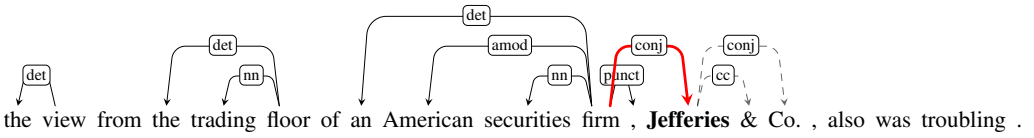
This transition-based parser makes local parsing decisions based on dense feature representations of the parser configuration. For a given parser configuration, the Chen & Manning (2014) model extracts dense feature representations for 18 elements in total: (1) The top 3 words on the stack and buffer, (2) the first and second leftmost / rightmost children of the top two words on the stack, and (3) the leftmost of leftmost / rightmost of rightmost children of the top two words on the stack.

The model of Chen & Manning (2014) extracts separate representations for the words, parts of speech, and corresponding arc labels of these elements. It then feeds forward these embedding features through a fully connected neural network with a single hidden layer, using the network output to predict parser transitions.

A key remaining issue not addressed by Chen & Manning (2014) is that the parser can only make use of knowledge from its local parsing context — namely, three words to the left on the stack (including dependents) and three words to the right on the buffer. This leaves the parser unable to reason about long-distance grammatical constructions when such constructions do not fall within the narrow range established by the given local features. We expand this claim in the following section with examples and preliminary linguistic analysis.

[1]Note that each transition has preconditions on the parser state, and so not all three are always legal decisions.

Figure 1: Erroneous parse produced by the Chen & Manning (2014) parser. The parser incorrectly detects a list phrase (this is an appositive) and makes an invalid `conj` arc, making a correct parse of the appositive's structure impossible. The word at the head of the stack at the time of the error is bolded, the erroneous arc is drawn in bold red, and later arcs which the erroneous arc prevents from being drawn are shown in dashed gray.



2.2 Analysis

We analyze the first errors committed by the Chen & Manning (2014) parser for each sentence in the development corpus. A common trend we notice in these parser errors is that the parser **fails to notice information later in the sentence that informs the grammatical role of the word currently being parsed**. Figure 1 gives an example error, where the parser decides to draw an arc (and an incorrect one at that) too early. Without any view of sentence-level information, it fails to notice that the word being parsed (*Jefferies*) is part of an appositive phrase, not a list.

This analysis suggests that the parser might perform better with a representation of long-distance features of the sentence.

Confusion over long-distance modifier attachments, especially in the cases of clausal complements and prepositional phrases, may also be resolved by such an extension. It is unclear from a first-error analysis, however, when these mistakes are made due to lack of sentence-level knowledge and when they are made due to a genuine confusion over the attachment of modifiers already visible to the parser. We will return to these questionable error cases in Section 5 in analyzing parser results.

3 Model: Buffer-aware parsing

Motivated by our analysis in Section 2.2, we propose an extension to the model of Chen & Manning (2014) which allows the classifier built into the parser to incorporate sentence-level knowledge.

Recent work in neural machine translation shows that sentence meaning can be effectively encoded in a fixed-length vector in order to produce translations with near state-of-the-art accuracy (Cho et al., 2014; Sutskever et al., 2014). Inspired by this work, we attempt to incorporate long-distance buffer features through a similar encoding process. We encode the buffer of a given parser configuration using a recurrent neural network language model (RNNLM) trained on backwards English text, and append the fixed-length vector encoding to the classifier's hidden layer. These buffer encodings are then integrated into an existing neural-network dependency parser. The next sections explain this additional language-modeling feature and its parser integration in more detail.

3.1 Recurrent neural network language model

We use a recurrent neural network language model (Mikolov et al., 2010) with an implementation of Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997; Graves, 2013). With the LSTM unit in place, a recurrent neural network language model can learn to adaptively remember and forget words it has observed. It can, for example, favor remembering content words over content-less particles. More relevantly for our purposes, it can learn to retain information about grammatical structure in a parser's buffer that may be of use for current parsing decisions.

To predict the next word in a sentence given a current word embedding w_i and previous LSTM value cell $\ell_{i-1} = \langle h_{i-1}, c_{i-1} \rangle$, we compute

$$\ell_i = \langle h_i, c_i \rangle = \mathcal{H}(\ell_{i-1}, w_i) \tag{1}$$

$$p_i = \text{softmax}(W_{soft} h_i) \tag{2}$$

where $\mathcal{H}$ is an LSTM activation as described in Graves (2013), W_{soft} is a learned weight matrix. p_i is an estimate of a probability distribution describing the next possible word in the sentence: $P(w_{i+1} \mid w_i, w_{i-1}, \ldots, w_1) \approx p_i(w_{i+1})$ (where $p_i(\cdot)$ represents vector indexing).

3.2 INCORPORATING RNNLM FEATURES

We use a recurrent neural network language model (Mikolov et al., 2010) described in Section 3.1 to encode the buffer of the transition-based parser, which will be some subsequence beginning at the start or in the middle of the sentence being parsed and ending after the final word in the sentence. We perform the actual language modeling in *reverse order*, from right to left across the buffer. In this way we can encourage the resulting RNN hidden layer value to be more informative for this purpose by completing (rather than starting) the language-modeling process at the word in question. This motivation is similar to that of Sutskever et al. (2014), who also reverse input data in order to promote better memory of short-distance relationships.

This feedforward process is as defined in the language modeling LSTM in Equation (1), except that our time-steps proceed right-to-left through a sentence rather than in the more intuitive direction. Mathematically, for a given parser buffer with words $w_i, w_{i+1}, \ldots, w_N$, where $i \geq 1$ and N is the number of words in the sentence, we define the buffer encoding B_i by the recurrence relation

$$\begin{aligned} \ell_{N+1} &= \langle \vec{0}, \vec{0} \rangle \\ \ell_i &= \langle B_i, c_i \rangle = \mathcal{H}(\ell_{i+1}, w_i). \end{aligned} \tag{3}$$

This buffer encoding B_i will be useful in the next section, where it is used to augment the existing representation of parser state.

3.3 AN EXTENDED CLASSIFIER FOR DEPENDENCY PARSING

We insert this RNNLM feature representation into the framework of Chen & Manning (2014). We augment the classifier central to making parsing decisions, which accepts a current transition-based parser state and suggests the next transition which the parser should take. Suppose for purposes of explanation that we wish to use the classifier to determine a transition while word w_{i-1} is the top word on the stack (i.e., the buffer contains words $w_i, w_{i+1}, \ldots, w_N$), and all words before w_{i-1} have either been completely parsed or still remain on the stack below w_{i-1}.

We represent each word in the sentence as a d-dimensional dense vector $e^w_j \in \mathbb{R}^d$.[2] Part-of-speech tags and labels (used to build state representations from the feature templates described in Section 2) are also represented with dense vectors $e^t_j, e^l_k \in \mathbb{R}^d$, where e^t_j is a representation of the part of speech of the jth word and e^l_k is a representation of the kth label in the grammar being constructed. These embeddings are retrieved from matrices E^w, E^t, E^l, respectively.

We first construct the input layer. All word embedding features are concatenated into a vector $x^w \in \mathbb{R}^{d\,n_w}$, where n_w is the number of word-based features used. We similarly construct part-of-speech and label concatenations $x^t \in \mathbb{R}^{d\,n_t}, x^l \in \mathbb{R}^{d\,n_l}$.

The *parser-focused* hidden layer (of dimension d_h) is a nonlinear transform of these inputs:

$$P_i = \left(W^w_1 x^w + W^t_1 x^t + W^l_1 x^l + b_1\right)^3 \tag{4}$$

where $W^w_1 \in \mathbb{R}^{d_h \times (dn_w)}, W^t_1 \in \mathbb{R}^{d_h \times (dn_t)}, W^l_1 \in \mathbb{R}^{d_h \times (dn_l)}$ are learned weight matrices, and $b_1 \in \mathbb{R}^{d_h}$ is a learned bias term. We use the same cube activation function as in Chen & Manning (2014).

We use a learned RNNLM to construct a fixed-length representation B_i of the buffer from the end of the sentence back to and including w_i, given in the buffer-modeling LSTM in Equation (3). This hidden layer component, the *buffer-focused* hidden layer, is concatenated to the parser-focused hidden layer P_i in a final softmax activation to produce the output layer p_{trans}:

$$p_{trans} = \text{softmax}\left(W_2 \begin{bmatrix} P_i \\ B_i \end{bmatrix}\right). \tag{5}$$

[2] In our present code, we train these embeddings separately from the embeddings used within the RNNLM.

The given weight matrix W_2 can be thought of as the block matrix $[W_{2P} \quad W_{2B}]$, where W_{2P} is an $l \times d_h$ matrix mapping the parser-focused hidden layer to the output layer, and W_{2B} is a $l \times d_L$ matrix mapping the buffer-focused hidden layer to the output layer.

As in Chen & Manning (2014), p_{trans} is a probability distribution over parser transitions. The highest-probability legal transition will be selected by the parser and used to update the parser state.

3.4 TRAINING

For each sentence we augment the training examples of the Chen & Manning (2014) parser with RNNLM buffer encodings. We generate training examples $\{(c_i, T_i, B(c_i))\}_{i=1}^{m}$, where c_i represents a parser configuration, T_i represents the correct transition for this configuration to reach the correct parse for the corresponding sentence, and $B(c_i)$ represents an RNNLM encoding of the buffer within c_i. In training we aim to minimize cross-entropy loss, with an l_2 regularization term:

$$L(\theta) = -\sum_i \log p_{T_i} + \frac{\lambda}{2}||\theta||^2 \tag{6}$$

where θ is the set of all parameters $\{W_1^w, W_1^t, W_1^l, b_1, W_{2P}, W_{2B}, E^w, E^t, E^l\}$. Softmax probabilities and backpropagation are performed only for legal transitions on a given configuration during training.

We backpropagate errors in parser transitions to W_{2P} and W_{2B}, the weight matrices mapping from the parser- and buffer-focused hidden layers to the output layer. The errors in W_{2P} are further backpropagated to the bias b_1, weight matrices W_1^w, W_1^t, W_1^l, and embedding matrices E^w, E^t, E^l. Errors are **not backpropagated through the RNNLM**. We our training focus to outside the RNNLM for the experiments in this paper; future work will explore the potential benefit of jointly training the language model along with the parser.

We train with mini-batched adaptive gradient descent (Duchi et al., 2011). We also perform hidden layer dropout (Hinton et al., 2012).

4 EXPERIMENTS

We train this parser model on the WSJ section of the Penn Treebank, using the standard splits: sections 2–21 are used for training, section 22 for development, and section 23 for testing. Using the Stanford Parser we convert the PTB to Stanford Dependencies representations, v3.3.0 (de Marneffe et al., 2006). Gold part-of-speech tags in the corpus are replaced with tags predicted by the Stanford Tagger, using the `bidirectional5words` model (Toutanova et al., 2003).

The hyperparameters of the Chen & Manning (2014) parser are retained in these experiments. We train the neural network with word, part-of-speech, and label embeddings of size $d = 50$ and regularization parameter $\lambda = 10^{-8}$. We use an initial AdaGrad learning rate of $\alpha = 0.01$, and dropout probability of 0.5.

We incorporate an RNNLM model as described in Sections 3.2 and 3.3.[3] We train with a hidden layer size $d_L = 200$, and an output vocabulary of 20,000 words.[4] All LSTM weights (along with word embeddings and softmax weights for the output layer) are randomly sampled from a uniform distribution over $[-0.1, 0.1]$. The RNN trains on a batch of 128 sentences at a time.

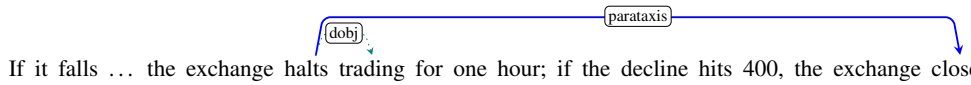
We use the weights of a competitive transition-based model (labeled as "Baseline" in Table 1) trained with the Chen & Manning (2014) parser as initializations for the extended parser weights. The portion of the second weight matrix W_2 which maps from the buffer-focused hidden layer section to the output layer (named W_{2B} in our earlier discussion) is randomly initialized in the range $[-0.1, 0.1]$. We train with mini-batched AdaGrad (Duchi et al., 2011) for 20,000 iterations (with a batch size of 10^4) and retain the model which yields the highest unlabeled attachment score (UAS; percent of tokens with correct head token) on a held-out development set.

[3] Thanks to Thang Luong (Stanford CS) for providing us with MATLAB code to train an RNNLM.

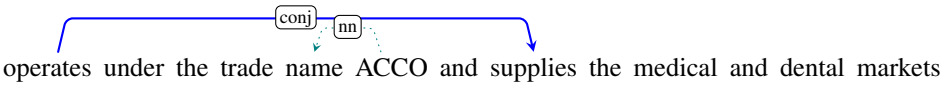
[4] We focus on predicting the 20,000 most common words in the corpus; all other words are mapped to an `<unk>` token.

Model	Dev		Test	
	UAS	LAS	UAS	LAS
Baseline (Chen & Manning, 2014)	91.66	89.27	91.21	89.06
Control	91.70	89.21	91.29	89.13
With RNNLM	**91.80**	**89.36**	**91.39**	**89.23**

Table 1: Model performance on PTB development and test sets (Stanford Dependencies labeling scheme)



(a) Parataxis arc (bold blue) which the RNNLM-augmented parser gets correct. To draw this arc correctly, it is necessary to arc onto the prepositional phrase *for one hour* (dotted green arc) before the parataxis is actually observed.



(b) `conj` arc (bold blue) which the RNNLM-augmented parser gets correct. In order to draw this arc correctly, it is necessary to recognize that *ACCO* is part of a noun compound (and not within the scope of a new conjunct phrase).

Figure 2: Parses drawn from our development split of the Penn Treebank WSJ data which are corrected by the RNNLM-augmented parser, while incorrectly parsed by the control parser

4.1 Results

Unlabeled and labeled attachment scores calculated on section 23 of the WSJ portion of the Penn Treebank are shown in Table 1. As is standard, punctuation arcs are excluded from evaluation. The model labeled "Baseline" is the best model trained using the released Chen & Manning (2014) parser (no extensions or retraining).[5]

Since we used the weights of this model to initialize our own RNNLM-augmented parser, it would be fair to train a separate "Control" model as well with the same weight initializations for the same number of iterations as we train our own parser. The line in Table 1 labeled "Control" is the model resulting from retraining using the baseline model's weights for 20,000 iterations (the same number of iterations for which the RNNLM-augmented parser is trained) on the same training data used by the RNNLM parser.

The RNNLM-augmented parser shows an improvement of about 0.1% in UAS and LAS over the control model ($\sim$ 0.2% UAS and LAS over the baseline). While this is only a modest quantitative improvement, we discuss in the next section qualitative performance improvements that indicate that the RNNLM is doing something positive for the parser.

5 Discussion

We find that the RNNLM-augmented parser learns to correct the error discussed earlier in Figure 1. The next paragraphs discuss a different type of error resolved by the parser: cases where arcs were not being drawn *soon enough* (rather than being drawn too early).

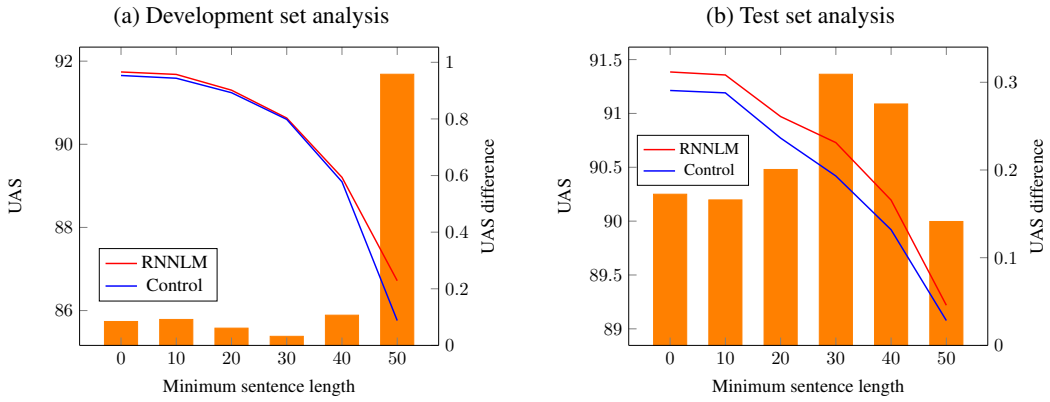
Figure 2a shows an example sentence where the RNNLM-augmented parser is able to successfully recognize a long-term parataxis relation. It is actually necessary to recognize that this parataxis relation exists far earlier during parsing: we must draw the green `dobj` arc sooner rather than later

[5]The released Java parser code has been acknowledged to produce inferior numbers, below those attained in the paper by an original MATLAB implementation. The baseline numbers shown here are the best of some 20 runs of the training code.

Effect on modifier attachment	Count
Fixed attachment	17
Broke attachment	21
Rearranged attachment	11

Table 2: Survey of modifier attachment changes with the RNNLM augmented parser (versus the output of the control parser)

Figure 3: Parser performance on development and test sets when restricting evaluation to sentences of a particular minimum length (horizontal axes). The orange bars show the difference between the RNNLM-augmented parser UAS (red line) and control parser UAS (blue line).



in order to be able to eventually draw the right arc from *halt* onto *close*. Using the added RNNLM features, the parser was able to recognize this and make the proper arc. Though we did not expect it from our initial first-error analysis, we find that **parataxis** (either as a quasi-coordination relation as in Figure 2a or as a relation indicating reported speech) is **one of the key parse issues resolved** by the RNNLM-augmented parser.

Figure 2b shows a sentence with scope ambiguity. Our control parser misunderstands the scope to read *operates under the trade name [[ACCO] and [supplies the medical and dental markets]]*. The RNNLM-augmented parser correctly recognizes at the point of parsing the word *ACCO* that the following phrase coordinates not with a noun phrase but with an entire verb phrase: that is, the proper bracketing is *[operates under the trade name ACCO] and [supplies the medical and dental markets]*. It makes the necessary left `nn` arc (shown in green in the figure) in order to be able to form the correct coordination construction later on.

In Section 2.2 we posed the question of whether an RNNLM-augmented parser might be able to resolve more general modifier attachment problems. To try to answer this question, we took a high-level survey of 100 parses from the development set for which the RNNLM-augmented parser output and control parser output differed. Table 2 shows the number of modifier attachment ambiguities which were resolved in the RNNLM-augmented parse. We also count the number of modifier attachment issues introduced by the RNNLM parser, and attachment problems which were modified but not fixed by the RNNLM parser. There is no clear evidence from this survey that the RNNLM can help or hurt resolve modifier attachment ambiguity in general.

We do see a clearer trend, however, in RNNLM parser performance on different length sentences. During evaluation we hypothesized that the effect of the RNNLM augmentation would be more evident on longer sentences, where sentence-level knowledge is of increasing importance (where the chances of missing modifiers later in the sentence increase with sentence size). Figure 3 shows an analysis of the performance of the control and RNNLM-augmented parsers on subsets of the evaluation data (filtering by minimum sentence length). The effect is extremely clear on the development set: RNNLM-augmented parser performance (relative to the control performance) spikes at greater sentence lengths. Results are less clear on the test set, but we still see a trend of greater relative performance at greater lengths.

6 CONCLUSION

In this paper we evaluated the effects of introducing sentence-level knowledge in the form of RNNLM-constructed parser buffer encodings into parsing decisions. We demonstrated both (minor) quantitative gains and qualitative improvements in parser output.

There are two obvious avenues of future work for this project:

1. Develop the ability to **jointly train the parser and the language model**, backpropagating errors in parser predictions through time to the RNNLM weights. In this way the RNNLM can be trained to adaptively remember only the information in a parser buffer which is most useful for the purposes of parsing (rather than for the purposes of predicting the following words in the buffer sequence).
2. Investigate **alternatives to the complex RNNLM solution** (e.g., sparse features) which might yield the same qualitative effects as described in the analysis of RNNLM parser output.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Chen, Danqi and Manning, Christopher D. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Cho, Kyunghyun, Merrienboer, Bart van, Gulcehre, Caglar, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

de Marneffe, Marie-Catherine, MacCartney, Bill, Manning, Christopher D., and others. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pp. 449–454, 2006. URL http://t3-1.yum2.net/index/nlp.stanford.edu/manning/papers/LREC_2.pdf.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011. URL http://dl.acm.org/citation.cfm?id=2021068.

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv:1308.0850 [cs]*, August 2013. URL http://arxiv.org/abs/1308.0850. arXiv: 1308.0850.

Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. URL http://arxiv.org/abs/1207.0580.

Hochreiter, Sepp and Schmidhuber, Jrgen. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

McDonald, Ryan and Nivre, Joakim. Characterizing the errors of data-driven dependency parsing models. In *Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL)*, 2007.

Mikolov, Tomas, Karafit, Martin, Burget, Lukas, Cernockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, pp. 1045–1048, 2010. URL http://www.fit.vutbr.cz/research/groups/speech/servite/2010/rnnlm_mikolov.pdf.

Nivre, Joakim and Scholz, Mario. Deterministic dependency parsing of english text. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. doi: 10.3115/1220355.1220365. URL http://dx.doi.org/10.3115/1220355.1220365.

Nivre, Joakim, Hall, Johan, and Nilsson, Jens. Maltparser: A data-driven parser-generator for dependency parsing. In *LREC*, 2006.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. *arXiv:1409.3215 [cs]*, September 2014. URL http://arxiv.org/abs/1409.3215. arXiv: 1409.3215.

Toutanova, Kristina, Klein, Dan, Manning, Christopher D., and Singer, Yoram. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 173–180. Association for Computational Linguistics, 2003. URL http://dl.acm.org/citation.cfm?id=1073478.

Yamada, Hiroyasu and Matsumoto, Yuji. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, 2003. URL http://www.jaist.jp/~h-yamada/pdf/iwpt2003.pdf.

Zhang, Yue and Clark, Stephen. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pp. 162–171, Paris, France, October 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W09-3825.