CCG Supertagging with a Recurrent Neural Network

Wenduan Xu
University of Cambridge
Computer Laboratory
wx217@cam.ac.uk

Michael Auli*
Facebook AI Research
michaelauli@fb.com

Stephen Clark
University of Cambridge
Computer Laboratory
sc609@cam.ac.uk

Abstract

Recent work on supertagging using a feed-forward neural network achieved significant improvements for CCG supertagging and parsing (Lewis and Steedman, 2014). However, their architecture is limited to considering local contexts and does not naturally model sequences of arbitrary length. In this paper, we show how directly capturing sequence information using a recurrent neural network leads to further accuracy improvements for both supertagging (up to 1.9%) and parsing (up to 1% F1), on CCGBank, Wikipedia and biomedical text.

1 Introduction

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a highly lexicalized formalism; the standard parsing model of Clark and Curran (2007) uses over 400 lexical categories (or *supertags*), compared to about 50 POS tags for typical CFG parsers. This makes accurate disambiguation of lexical types much more challenging. However, the assignment of lexical categories can still be solved reasonably well by treating it as a sequence tagging problem, often referred to as supertagging (Bangalore and Joshi, 1999). Clark and Curran (2004) show that high tagging accuracy can be achieved by leaving some ambiguity to the parser to resolve, but with enough of a reduction in the number of tags assigned to each word so that parsing efficiency is greatly increased.

In addition to improving parsing efficiency, supertagging also has a large impact on parsing accuracy (Curran et al., 2006; Kummerfeld et al., 2010), since the derivation space of the parser is determined by the supertagger, at both training and test time. Clark and Curran (2007) enhanced supertagging using a so-called adaptive strategy, such that additional categories are supplied to the parser only if a spanning analysis cannot be found. This strategy is used in the de facto C&C parser (Curran et al., 2007), and the two-stage CCG parsing pipeline (supertagging and parsing) continues to be the choice for most recent CCG parsers (Zhang and Clark, 2011; Auli and Lopez, 2011; Xu et al., 2014).

Despite the effectiveness of supertagging, the most widely used model for this task (Clark and Curran, 2007) has a number of drawbacks. First, it relies too heavily on POS tags, which leads to lower accuracy on out-of-domain data (Rimell and Clark, 2008). Second, due to the sparse, indicator feature sets mainly based on raw words and POS tags, it shows pronounced performance degradation in the presence of rare and unseen words (Rimell and Clark, 2008; Lewis and Steedman, 2014). And third, in order to reduce computational requirements and feature sparsity, each tagging decision is made without considering any potentially useful contextual information beyond a local context window.

Lewis and Steedman (2014) introduced a feed-forward neural network to supertagging, and addressed the first two problems mentioned above. However, their attempt to tackle the third problem by pairing a conditional random field with their feed-forward tagger provided little accuracy improvement and vastly increased computational complexity, incurring a large efficiency penalty.

We introduce a recurrent neural network-based (RNN) supertagging model to tackle all the above problems, with an emphasis on the third one. RNNs are powerful models for sequential data, which can potentially capture long-term dependencies, based on an *unbounded* history of previous words (§2); similar to Lewis and Steedman (2014) we only use distributed word representa-

*All work was completed before the author joined Facebook.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers), pages 250–255,
Beijing, China, July 26-31, 2015.

tions (§2.2). Our model is highly accurate, and by integrating it with the C&C parser as its adaptive supertagger, we obtain substantial accuracy improvements, outperforming the feed-forward setup on both supertagging and parsing.

2 Supertagging with a RNN

2.1 Model

We use an Elman recurrent neural network (Elman, 1990) which consists of an input layer x_t, a hidden state (layer) h_t with a recurrent connection to the previous hidden state h_{t-1} and an output layer y_t. The input layer is a vector representing the surrounding context of the current word at position t, whose supertag is being predicted.[1] The hidden state h_{t-1} keeps a representation of all context history up to the current word. The current hidden state h_t is computed using the current input x_t and hidden state h_{t-1} from the previous position. The output layer represents probability scores of all possible supertags, with the size of the output layer being equal to the size of the lexical category set.

The parameterization of the network consists of three matrices which are learned during supervised training. Matrix $\mathbf{U}$ contains weights between the input and hidden layers, $\mathbf{V}$ contains weights between the hidden and output layers, and $\mathbf{W}$ contains weights between the previous hidden state and the current hidden state. The following recurrence[2] is used to compute the activations of the hidden state at word position t:

$$h_t = f(x_t\mathbf{U} + h_{t-1}\mathbf{W}), \tag{1}$$

where f is a non-linear activation function; here we use the sigmoid function $f(z) = \frac{1}{1+e^{-z}}$. The output activations are calculated as:

$$y_t = g(h_t\mathbf{V}), \tag{2}$$

where g is the softmax activation function $g(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ that squeezes raw output activations into a probability distribution.

2.2 Word Embeddings

Our RNN supertagger only uses continuous vector representations for features and each feature type has an associated look-up table, which maps a feature to its distributed representation. In total, three feature types are used. The first type is word embeddings: given a sentence of N words, $(w_1, w_2, \ldots, w_N)$, the embedding feature of w_t (for $1 \leq t \leq N$) is obtained by projecting it onto a n-dimensional vector space through the look-up table $L_w \in \mathbb{R}^{|w|\times n}$, where $|w|$ is the size of the vocabulary. Algebraically, the projection operation is a simple vector-matrix product where a one-hot vector $b_j \in \mathbb{R}^{1\times|w|}$ (with zeros everywhere except at the jth position) is multiplied with L_w:

$$e_{w_t} = b_j L_w \in \mathbb{R}^{1\times n}, \tag{3}$$

where j is the look-up index for w_t.

In addition, as in Lewis and Steedman (2014), for every word we also include its 2-character suffix and capitalization as features. Two more look-up tables are used for these features. $L_s \in \mathbb{R}^{|s|\times m}$ is the look-up table for suffix embeddings, where $|s|$ is the suffix vocabulary size. $L_c \in \mathbb{R}^{2\times m}$ is the look-up table for the capitalization embeddings. L_c contains only two embeddings, representing whether or not a given word is capitalized.

We extract features from a context window surrounding the current word to make a tagging decision. Concretely, with a context window of size k, $\lfloor k/2 \rfloor$ words either side of the target word are included. For a word w_t, its continuous feature representation is:

$$f_{w_t} = [e_{w_t}; s_{w_t}; c_{w_t}], \tag{4}$$

where $e_{w_t} \in \mathbb{R}^{1\times n}$, $s_{w_t} \in \mathbb{R}^{1\times m}$ and $c_{w_t} \in \mathbb{R}^{1\times m}$ are the output vectors from the three different look-up tables, and $[e_{w_t}; s_{w_t}; c_{w_t}]$ denotes the concatenation of three vectors and hence $f_{w_t} \in \mathbb{R}^{1\times(n+2m)}$. At word position t, the input layer of the network x_t is:

$$x_t = [f_{w_{t-\lfloor k/2 \rfloor}}; \ldots f_{w_t}; \ldots; f_{w_{t+\lfloor k/2 \rfloor}}], \tag{5}$$

where $x_t \in \mathbb{R}^{1\times k(n+2m)}$ and the right-hand side is the concatenation of all feature representations in a size k context window.

We use pre-trained word embeddings from Turian et al. (2010) to initialize look-up table L_w, and we apply a set of word pre-processing techniques at both training and test time to reduce sparsity. All words are first lower-cased, and all numbers are collapsed into a single digit '0'. If a lower-cased hyphenated

[1] This is different from some RNN models (e.g., Mikolov et al. (2010)) where the input is a one-hot vector.

[2] We assume the input to any layer is a row vector unless otherwise stated.

word does not have an entry in the pre-trained word embeddings, we attempt to back-off to the substring after the last hyphen. For compound words and numbers delimited by "\/", we attempt to back-off to the substring after the delimiter. After pre-processing, the Turian embeddings have a coverage of 94.25% on the training data; for out-of-vocabulary words, three separate randomly initialized embeddings are used for lower-case alphanumeric words, upper-case alphanumeric words, and non-alphanumeric symbols. For padding at the start and end of a sentence, the "unknown" entry from the pre-trained embeddings is used. Look-up tables L_s and L_c are also randomly initialized, and all look-up tables are modified during supervised training using backpropagation.

3 Experiments

Datasets and Baseline. We follow the standard splits of CCGBank (Hockenmaier and Steedman, 2007) for all experiments using sections 2-21 for training, section 00 for development and section 23 as in-domain test set. The Wikipedia corpus from Honnibal et al. (2009) and the Bioinfer corpus (Pyysalo et al., 2007) are used as two out-of-domain test sets. We compare supertagging accuracy with the MaxEnt C&C supertagger and the neural network tagger of Lewis and Steedman (2014) (henceforth NN), and we also evaluate parsing accuracy using these three supertaggers as a front-end to the C&C parser. We use the same 425 supertag set used in both C&C and NN.

Hyperparameters and Training. For L_w, we use the scaled 50-dimensional Turian embeddings ($n = 50$ for L_w) as initialization. We have experimented during development with using 100-dimensional embeddings and found no improvements in the resulting model. Out-of-vocabulary embedding values in L_w and all embedding values in L_s and L_c are initialized with a uniform distribution in the interval $[-2.0, 2.0]$. The embedding dimension size m of L_s and L_c is set to 5. Other parameters of the network $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ are initialized with values drawn uniformly from the interval $[-2.0, 2.0]$, and are then scaled by their corresponding input vector size. We experimented with context window sizes of 3, 5, 7, 9 and 11 during development and found a window size of 7 gives the best performing model on the dev set. We use a fixed learning rate of 0.0025 and a hidden state size of 200.

Model	Accuracy	Time
C&C (gold POS)	92.60	-
C&C (auto POS)	91.50	0.57
NN	91.10	21.00
RNN	92.63	-
RNN+dropout	93.07	2.02

Table 1: 1-best tagging accuracy and speed comparison on CCGBank Section 00 with a single CPU core (1,913 sentences), tagging time in secs.

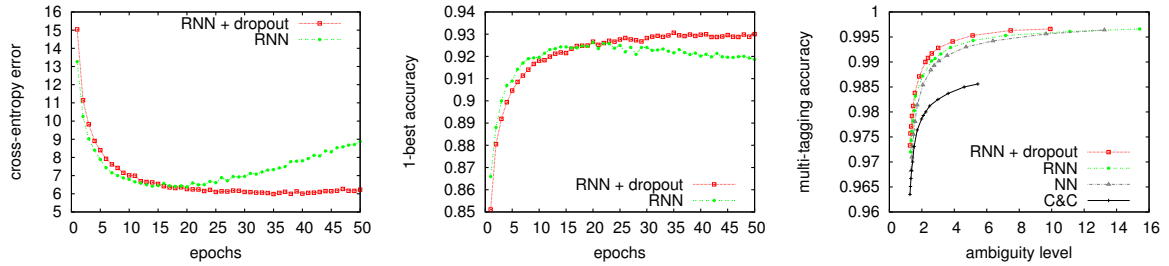
To train the model, we optimize cross-entropy loss with stochastic gradient descent using mini-batched backpropagation through time (BPTT; Rumelhart et al., 1988; Mikolov, 2012); the mini-batch size for BPTT, again tuned on the dev set, is set to 9.

Embedding Dropout Regularization. Without any regularization, we found cross-entropy error on the dev set started to increase while the error on the training set was continuously driven to a very small value (Fig. 1a). With the suspicion of overfitting, we experimented with l_1 and l_2 regularization and learning rate decay but none of these techniques gave any noticeable improvements for our model. Following Legrand and Collobert (2014), we instead implemented word embedding dropout as a regularization for all the look-up tables, since the capacity of our tagging model mainly comes from the look-up tables, as in their system. We observed more stable learning and better generalization of the trained model with dropout. Similar to other forms of droput (Srivastava et al., 2014), we randomly drop units and their connections to other units at training time. Concretely, we apply a binary dropout mask to x_t, with a dropout rate of 0.25, and at test time no mask is applied, but the input to the network, x_t, at each word position is scaled by 0.75. We experimented during development with different dropout rates, but found the above choice to be optimal in our setting.

3.1 Supertagging Results

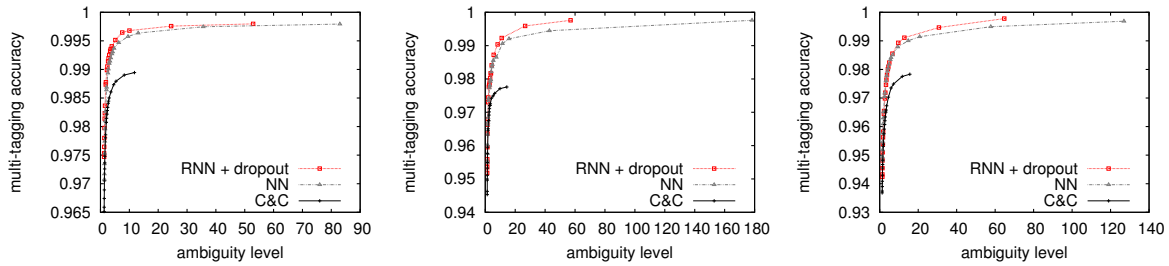
We use the RNN model which gives the highest 1-best supertagging accuracy on the dev set as the final model for all experiments. Without any form of regularization, the best model was obtained at the 20th epoch, and it took 35 epochs for the dropout model to peak (Fig. 1b). We use the dropout model for all experiments and, unlike the C&C supertagger, no tag dictionaries are used.

Table 1 shows 1-best supertagging accuracies on the dev set. The accuracy of the C&C supertag-



(a) learning curve on dev set (b) 1-best accuracy on dev set (c) multi-tagging accuracy on dev set

Figure 1: Learning curve and 1-best tagging accuracy of the RNN model on CCGBank Section 00. Plot (c) shows ambiguity vs. multi-tagging accuracy for all supertaggers (auto POS).



(a) CCBBank Section 23 (b) wikipedia (c) bio-GENIA

Figure 2: Multi-tagging accuracy for all supertagging models on CCGBank Section 23, Wikipedia and Bio-GENIA data (auto POS).

	RNN			NN			C&C (auto pos)			C&C (gold pos)		
β	WORD	SENT	amb.	WORD	SENT	amb.	WORD	SENT	amb.	WORD	SENT	amb.
0.075	97.33	66.07	1.27	96.83	61.27	1.34	96.34	60.27	1.27	**97.34**	**67.43**	1.27
0.030	**98.12**	**74.39**	1.46	97.81	70.83	1.58	97.05	65.50	1.43	97.92	72.87	1.43
0.010	**98.71**	**81.70**	1.84	98.54	79.25	2.06	97.63	70.52	1.72	98.37	77.73	1.72
0.005	**99.01**	**84.79**	2.22	98.84	83.38	2.55	97.86	72.24	1.98	98.52	79.25	1.98
0.001	**99.41**	**90.54**	3.90	99.29	89.07	4.72	98.25	80.24	3.57	99.17	87.19	3.00

Table 2: Multi-tagging accuracy and ambiguity comparison (supertags/word) at the default C&C β levels on CCGBank Section 00.

Model	Section 23	Wiki	Bio
C&C (gold POS)	93.32	88.80	91.85
C&C (auto POS)	92.02	88.80	89.08
NN	91.57	89.00	88.16
RNN	93.00	90.00	88.27

Table 3: 1-best tagging accuracy comparison on CCGBank Section 23 (2,407 sentences), Wikipedia (200 sentences) and Bio-GENIA (1,000 sentences).

ger drops about 1% with automatically assigned POS tags, while our RNN model gives higher accuracy (+0.47%) than the C&C supertagger with gold POS tags. All timing values are obtained on a single Intel i7-4790k core, and all implementations are in C++ except NN which is implemented using Torch and Java, and therefore we believe the efficiency of NN could be vastly improved with an implementation with a lower-level language.

Table 2 compares different supertagging models for multi-tagging accuracy at the default β levels used by the C&C parser on the dev set. The β parameter determines the average number of supertags assigned to each word (ambiguity) by a supertagger when integrated with the parser; categories whose probabilities are not within β times the probability of the 1-best category are pruned. At the first β level (0.075), the three supertagging models give very close ambiguity levels, but our RNN model clearly outperforms NN and C&C (auto POS) in both word (WORD) and sentence (SENT) level accuracies, giving similar word-level accuracy as C&C (gold POS). For other β levels (except $\beta = 0.001$), the RNN model gives comparable ambiguity levels to the C&C model which uses a tagdict, while being much more accurate than both the other two models.

	LP	LR	LF	SENT	CAT	cov.
C&C (normal)	85.18	82.53	83.83	31.42	92.39	100
C&C (hybrid)	86.07	82.77	84.39	32.62	92.57	100
C&C (normal + RNN)	86.74	84.58	85.65	34.13	93.60	100
C&C (hybrid + RNN)	**87.73**	**84.83**	**86.25**	**34.97**	**93.84**	100
C&C (normal)	85.18	84.32	84.75	31.73	92.83	99.01 (C&C cov)
C&C (hybrid)	86.07	84.49	85.28	32.93	93.02	99.06 (C&C cov)
C&C (normal + RNN)	86.81	86.01	86.41	34.37	93.80	99.01 (C&C cov)
C&C (hybrid + RNN)	**87.77**	**86.25**	**87.00**	**35.20**	**94.04**	99.06 (C&C cov)
C&C (normal + RNN)	86.74	86.15	86.45	34.33	93.81	**99.42**
C&C (hybrid + RNN)	87.73	86.41	87.06	35.17	94.05	**99.42**

Table 4: Parsing development results on CCGBank Section 00 (auto POS).

	CCGBank Section 23				Wikipedia				Bioinfer			
	LP	LR	LF	cov.	LP	LR	LF	cov.	LP	LR	LF	cov.
C&C	86.24	84.85	85.54	99.42	81.58	80.08	80.83	99.50	77.78	76.07	76.91	95.40
C&C (+ NN)	86.71	85.56	86.13	99.92	82.65	81.36	82.00	**100**	79.77	**78.62**	**79.19**	97.40
C&C (+ RNN)	**87.68**	**86.47**	**87.07**	**99.96**	**83.22**	**81.78**	**82.49**	**100**	**80.10**	78.21	79.14	**97.80**
C&C	86.24	84.17	85.19	100	81.58	79.48	80.52	100	77.78	71.44	74.47	100
C&C (+ NN)	86.71	85.40	86.05	100	-	-	-	-	79.77	75.35	77.50	100
C&C (+ RNN)	**87.68**	**86.41**	**87.04**	100	-	-	-	-	**80.10**	**75.52**	**77.74**	100

Table 5: Parsing test results on all three domains (auto POS). We evaluate on all sentences (100% coverage) as well as on only those sentences that returned spanning analyses (% cov.). RNN and NN both have 100% coverage on the Wikipedia data.

Fig. 1c compares multi-tagging accuracies of all the models on the dev set. For all models, the same β levels are used (ranging from 0.075 to 10^{-4}, and all C&C default values are included). The RNN model consistently outperforms other models across different ambiguity levels.

Table 3 shows 1-best accuracies of all models on the test data sets (Bio-GENIA gold-standard CCG lexical category data from Rimell and Clark (2008) are used, since no gold categories are available in the Bioinfer data). With gold-standard POS tags, the C&C model outperforms both the NN and RNN models on CCGBank and Bio-GENIA; with auto POS, the accuracy of the C&C model drops significantly, due to its high reliance on POS tags.

Fig. 2 shows multi-tagging accuracies on all test data (using β levels ranging from 0.075 to 10^{-6}, and all C&C default values are included). On CCGBank, the RNN model has a clear accuracy advantage, while on the other two data sets, the accuracies given by the NN model are closer to the RNN model at some ambiguity levels, representing these data sets are still more challenging than CCGBank. However, both the NN and RNN models are more robust than the C&C model on the two out-of-domain data sets.

3.2 Parsing Results

We integrate our supertagging model into the C&C parser, at both training and test time, using all default parser settings; C&C hybrid model is used for CCGBank and Wikipedia; the normal-form model is used for the Bioinfer data, in line with Lewis and Steedman (2014) and Rimell and Clark (2008). Parsing development results are shown in Table 4; for out-of-domain data sets, no separate development experiments were done. Final results are shown in Table 5, and we substantially improve parsing accuracies on CCGBank and Wikipedia. The accuracy of our model on CCGBank represents a F1 score improvement of $1.53\%/1.85\%$ over the C&C baseline, which is comparable to the best known accuracy reported in Auli and Lopez (2011). However, our RNN-supertagging-based model is conceptually much simpler, with no change to the parsing model required at all.

4 Conclusion

We presented a RNN-based model for CCG supertagging, which brings significant accuracy improvements for supertagging and parsing, on both in- and out-of-domain data sets. Our supertagger is fast and well-suited for large scale processing.

Acknowledgements

The first author acknowledges the Carnegie Trust for the Universities of Scotland and the Cambridge Trusts for providing funding. SC is supported by ERC Starting Grant DisCoTex (306920) and EPSRC grant EP/I037512/1. We also thank the anonymous reviewers for their helpful comments.

References

Michael Auli and Adam Lopez. 2011. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 333–343. Association for Computational Linguistics.

Srinivas Bangalore and Aravind K Joshi. 1999. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265.

Stephen Clark and James R Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th international conference on Computational Linguistics*, page 282. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

James R Curran, Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 697–704. Association for Computational Linguistics.

James R Curran, Stephen Clark, and Johan Bos. 2007. Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 33–36. Association for Computational Linguistics.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Matthew Honnibal, Joel Nothman, and James R Curran. 2009. Evaluating a statistical CCG parser on wikipedia. In *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, pages 38–41. Association for Computational Linguistics.

Jonathan K Kummerfeld, Jessika Roesner, Tim Dawborn, James Haggerty, James R Curran, and Stephen Clark. 2010. Faster parsing by supertagger adaptation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 345–355. Association for Computational Linguistics.

Joël Legrand and Ronan Collobert. 2014. Joint RNN-based greedy parsing and word composition. *arXiv preprint arXiv:1412.7028*.

Mike Lewis and Mark Steedman. 2014. Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

Tomáš Mikolov. 2012. *Statistical Language Models Based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

Sampo Pyysalo, Filip Ginter, Juho Heimonen, Jari Björne, Jorma Boberg, Jouni Järvinen, and Tapio Salakoski. 2007. Bioinfer: a corpus for information extraction in the biomedical domain. *BMC bioinformatics*, 8(1):50.

Laura Rimell and Stephen Clark. 2008. Adapting a lexicalized-grammar parser to contrasting domains. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 475–484. Association for Computational Linguistics.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling*, 5.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, Mass.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Wenduan Xu, Stephen Clark, and Yue Zhang. 2014. Shift-reduce CCG parsing with a dependency model. In *Proceedings of the 2014 ACL Conference*.

Yue Zhang and Stephen Clark. 2011. Shift-reduce CCG parsing. In *Proc. ACL 2011*, pages 683–692, Portland, OR.