Learning from errors: Using vector-based compositional semantics for parse reranking

Phong Le, Willem Zuidema, Remko Scha
Institute for Logic, Language, and Computation
University of Amsterdam, the Netherlands
{p.le,zuidema,scha}@uva.nl

Abstract

In this paper, we address the problem of how to use semantics to improve syntactic parsing, by using a hybrid reranking method: a k-best list generated by a symbolic parser is reranked based on parse-correctness scores given by a compositional, connectionist classifier. This classifier uses a recursive neural network to construct vector representations for phrases in a candidate parse tree in order to classify it as syntactically correct or not. Tested on the WSJ23, our method achieved a statistically significant improvement of 0.20% on F-score (2% error reduction) and 0.95% on exact match, compared with the state-of-the-art Berkeley parser. This result shows that vector-based compositional semantics can be usefully applied in syntactic parsing, and demonstrates the benefits of combining the symbolic and connectionist approaches.

1 Introduction

Following the idea of compositionality in formal semantics, compositionality in vector-based semantics is also based on the *principle of compositionality*, which says that "The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined" (Partee, 1995). According to this principle, composing the meaning of a phrase or sentence requires a syntactic parse tree, which is, in most current systems, given by a statistical parser. This parser, in turn, is trained on syntactically annotated corpora.

However, there are good reasons to also consider information flowing in the opposite direction: from semantics to syntactic parsing. Performance of parsers trained and evaluated on the Penn WSJ treebank has reached a plateau, as many ambiguities cannot be resolved by syntactic information alone. Further improvements in parsing may depend on the use of additional sources of information, including semantics. In this paper, we study the use of semantics for syntactic parsing.

The currently dominant approach to syntactic parsing is based on extracting symbolic grammars from a treebank and defining appropriate probability distributions over the parse trees that they license (Charniak, 2000; Collins, 2003; Klein and Manning, 2003; Petrov et al., 2006; Bod et al., 2003; Sangati and Zuidema, 2011; van Cranenburgh et al., 2011). An alternative approach, with promising recent developments (Socher et al., 2010; Collobert, 2011), is based on using neural networks. In the present paper, we combine the 'symbolic' and 'connectionist' approaches through reranking: a symbolic parser is used to generate a k-best list which is then reranked based on parse-correctness scores given by a connectionist compositional-semantics-based classifier.

The idea of reranking is motivated by analyses of the results of state-of-the-art symbolic parsers such as the Brown and Berkeley parsers, which have shown that there is still considerable room for improvement: oracle results on 50-best lists display a dramatic improvement in accuracy (96.08% vs. 90.12% on F-score and 65.56% vs. 37.22% on exact match with the Berkeley parser). This suggests that parsers that rely on syntactic corpus-statistics, though not sufficient by themselves, may very well serve as a basis for systems that integrate other sources of information by means of reranking.

One important complementary source of information is the semantic plausibility of the constituents of the syntactically viable parses. The exploitation of that kind of information is the topic of the research we report here. In this work, we follow up on a proposal by Mark Steedman

Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, pages 11–19,
Sofia, Bulgaria, August 9 2013.

(1999), who suggested that the realm of semantics lacks the clearcut hierarchical structures that characterise syntax, and that semantic information may therefore be profitably treated by the classificatory mechanisms of neural nets—while the treatment of syntactic structures is best left to symbolic parsers. We thus developed a hybrid system, which parses its input sentences on the basis of a symbolic probabilistic grammar, and reranks the candidate parses based on scores given by a neural network.

Our work is inspired by the work of Socher and colleagues (2010; 2011). They proposed a parser using a recursive neural network (RNN) for encoding parse trees, representing phrases in a vector space, and scoring them. Their experimental result (only 1.92% lower than the Stanford parser on unlabelled bracket F-score for sentences up to a length of 15 words) shows that an RNN is expressive enough for syntactic parsing. Additionally, their qualitative analysis indicates that the learnt phrase features capture some aspects of phrasal semantics, which could be useful to resolve semantic ambiguity that syntactical information alone can not. Our work in this paper differs from their work in that we replace the parsing task by a reranking task, and thus reduce the object space significantly to a set of parses generated by a symbolic parser rather than the space of all parse trees. As a result, we can apply our method to sentences which are much longer than 15 words.

Reranking a k-best list is not a new idea. Collins (2000), Charniak and Johnson (2005), and Johnson and Ural (2010) have built reranking systems with performances that are state-of-the-art. In order to achieve such high F-scores, those rerankers rely on a very large number of features selected on the basis of expert knowledge. Unlike them, our feature set is selected automatically, yet the reranker achieved a statistically significant improvement on both F-score and exact match.

Closest to our work is Menchetti et al. (2005) and Socher et al. (2013): both also rely on symbolic parsers to reduce the search space and use RNNs to score candidate parses. However, our work differs in the way the feature set for reranking is selected. In their methods, only the score at the tree root is considered whereas in our method the scores at all internal nodes are taken into account. Selecting the feature set like that gives us a flexible way to deal with errors accumulated from the leaves to the root.

Figure 1 shows a diagram of our method. First, a parser (in this paper: the Berkeley parser) is used to generate k-best lists of the Wall Street Journal (WSJ) sections 02-21. Then, all parse trees in these lists and the WSJ02-21 are preprocessed by marking head words, binarising, and performing error-annotation (Section 2). After that, we use the annotated trees to train our parse-correctness classifier (Section 3). Finally, those trees and the classifier are used to train the reranker (Section 4).

2 Experimental Setup

The experiments presented in this paper have the following setting. We use the WSJ corpus with the standard splits: sections 2-21 for training, section 22 for development, and section 23 for testing. The latest implementation (version 1.7) of the Berkeley parser[1] (Petrov et al., 2006) is used for generating 50-best lists. We mark head words and binarise all trees in the WSJ and the 50-best lists as in Subsection 2.1, and annotate them as in Subsection 2.2 (see Figure 2).

2.1 Preprocessing Trees

We preprocess trees by marking head words and binarising the trees. For head word marking, we used the head finding rules of Collins (1999) which are implemented in the Stanford parser. To binarise a k-ary branching, e.g. $P \rightarrow C_1 \ ... \ H \ ... \ C_k$ where H is the top label of the head constituent, we use the following method. If H is not the left-most child, then

$$P \rightarrow C_1 \ @P \ ; \ @P \rightarrow C_2 \ ... \ H \ ... \ C_k$$

otherwise,

$$P \rightarrow @P \ C_k \ ; \ @P \rightarrow H \ ... \ C_{k-1}$$

where $@P$, which is called extra-P, now is the head of P. We then apply this transformation again on the children until we reach terminal nodes. In this way, we ensure that every internal node has one head word.

2.2 Error Annotation

We annotate nodes (as *correct* or *incorrect*) as follows. Given a parse tree T in a 50-best list and a corresponding gold-standard tree G in the WSJ,

[1] https://code.google.com/p/berkeleyparser

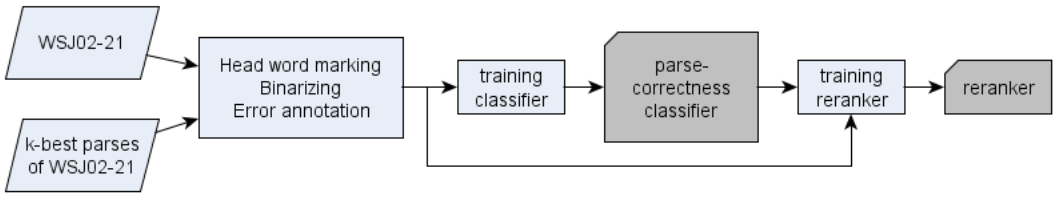

Figure 1: An overview of our method.

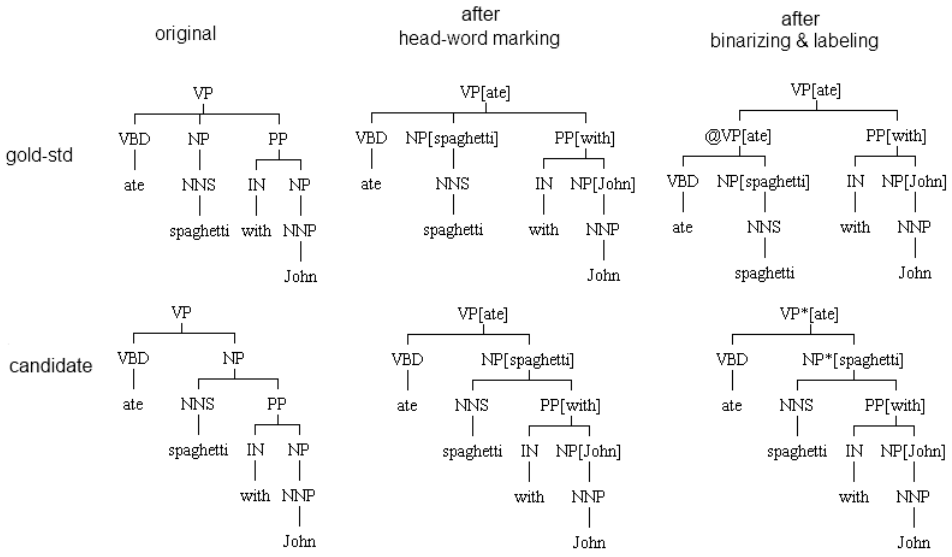

Figure 2: Example for preprocessing trees. Nodes marked with (*) are labelled *incorrect* whereas the other nodes are labelled *correct*.

we first attempt to align their terminal nodes according to the following criterion: a terminal node t is aligned to a terminal node g if they are at the same position counting from-left-to-right and they have the same label. Then, a non-terminal node $P[w_h]$ with children $C_1, ..., C_k$ is aligned to a gold-standard non-terminal node $P^*[w_h^*]$ with children $C_1^*, ..., C_l^*$ ($1 \leq k, l \leq 2$ in our case) if they have the same word head, the same syntactical category, and their children are all aligned in the right order. In other words, the following conditions have to be satisfied

$$P = P^* \; ; \; w_h = w_h^* \; ; \; k = l$$
$$C_i \textit{ is aligned to } C_i^*, \text{ for all } i = 1..k$$

Aligned nodes are annotated as *correct* whereas the other nodes are annotated as *incorrect*.

3 Parse-Correctness Classification

This section describes how a neural network is used to construct vector representations for phrases given parse trees and to identify if those trees are syntactically correct or not. In order to encode tree structures, we use an RNN[2] (see Figure 3 and Figure 4) which is similar to the one proposed by Socher and colleagues (2010). However, unlike their RNN, our RNN can handle unary branchings, and also takes head words and syntactic tags as input. It is worth noting that, although we can use some transformation to remove unary branchings, handling them is helpful in our case because the system avoids dealing with so many syntactic tags that would result from the transfor-

[2]The first neural-network approach attempting to operate and represent compositional, recursive structure is the Recursive Auto-Associative Memory network (RAAM), which was proposed by Pollack (1988). In order to encode a binary tree, the RAAM network contains three layers: an input layer for two daughter nodes, a hidden layer for their parent node, and an output layer for their reconstruction. Training the network is to minimise the reconstruction error such that we can decode the information captured in the hidden layer to the original tree form. Our RNN differs from the RAAM network in that its output layer is not for reconstruction but for classification.

mation. In addition, using a new set of weight matrices for unary branchings makes our RNN more expressive without facing the problem of sparsity thanks to a large number of unary branchings in the treebank.

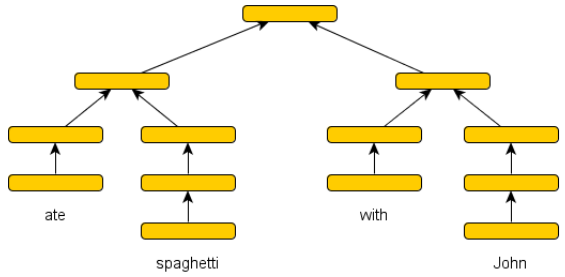

Figure 3: An RNN attached to the parse tree shown in the top-right of Figure 2. All unary branchings share a set of weight matrices, and all binary branchings share another set of weight matrices (see Figure 4).

An RNN processes a tree structure by repeatedly applying itself at each internal node. Thus, walking bottom-up from the leaves of the tree to the root, we compute for every node a vector based on the vectors of its children. Because of this process, those vectors have to have the same dimension. It is worth noting that, because information at leaves, i.e. lexical semantics, is composed according to a given syntactic parse, what a vector at each internal node captures is some aspects of compositional semantics of the corresponding phrase. In the remainder of this subsection, we describe in more detail how to construct compositional vector-based semantics geared towards the parse-correctness classification task.

Similar to Socher et al. (2010), and Collobert (2011), given a string of words $(w_1, ..., w_l)$, we first compute a string of vectors $(x_1, ..., x_l)$ representing those words by using a look-up table (i.e., word embeddings) $L \in \mathbb{R}^{n \times |V|}$, where $|V|$ is the size of the vocabulary and n is the dimensionality of the vectors. This look-up table L could be seen as a storage of lexical semantics where each column is a vector representation of a word. Hence, let b_i be the binary representation of word w_i (i.e., all of the entries of b_i are zero except the one corresponding to the index of the word in the dictionary), then

$$x_i = Lb_i \in \mathbb{R}^n \quad (1)$$

We also encode syntactic tags by binary vectors but put an extra bit at the end of each vector to mark if the corresponding tag is extra or not (i.e., $@P$ or P).

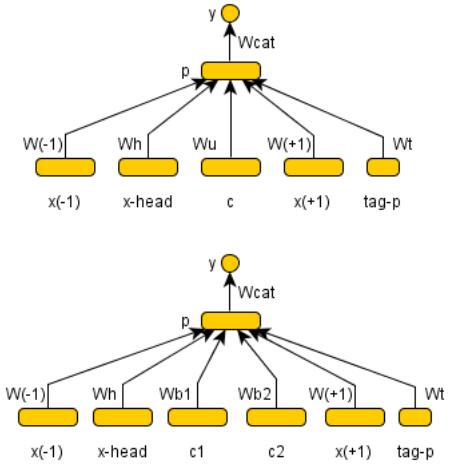

Figure 4: Details about our RNN for a unary branching (top) and a binary branching (bottom). The bias is not shown for the simplicity.

Then, given a unary branching $P[w_h] \rightarrow C$, we can compute the vector at the node P by (see Figure 4-top)

$$p = f\big(W_u c + W_h x_h + W_{-1} x_{-1} + W_{+1} x_{+1} + W_t t_p + b_u\big)$$

where c, x_h are vectors representing the child C and the head word, x_{-1}, x_{+1} are the left and right neighbouring words of P, t_p encodes the syntactic tag of P, $W_u, W_h, W_{-1}, W_{+1} \in \mathbb{R}^{n \times n}$, $W_t \in \mathbb{R}^{n \times (|T|+1)}$, $|T|$ is the size of the set of syntactic tags, $b_u \in \mathbb{R}^n$, and f can be any activation function ($tanh$ is used in this case). With a binary branching $P[w_h] \rightarrow C_1\ C_2$, we simply change the way the children's vectors added (see Figure 4-bottom)

$$p = f\big(W_{b1} c_1 + W_{b2} c_2 + W_h x_h + W_{-1} x_{-1} + W_{+1} x_{+1} + W_t t_p + b_b\big)$$

Finally, we put a sigmoid neural unit on the top of each internal node (except pre-terminal nodes because we are not concerned with POS-tagging) to detect the correctness of the subparse tree rooted at that node

$$y = sigmoid(W_{cat} p + b_{cat}) \quad (2)$$

where $W_{cat} \in \mathbb{R}^{1 \times n}, b_{cat} \in \mathbb{R}$.

3.1 Learning

The error on a parse tree is computed as the sum of classification errors of all subparses. Hence, the learning is to minimise the objective

$$J(\theta) = \frac{1}{N} \sum_{T} \sum_{(y(\theta),t) \in T} \frac{1}{2}(t - y(\theta))^2 + \lambda \|\theta\|^2 \tag{3}$$

where θ are the model parameters, N is the number of trees, λ is a regularisation hyperparameter, T is a parse tree, $y(\theta)$ is given by Equation 2, and t is the class of the corresponding subparse ($t = 1$ means *correct*). The gradient $\frac{\partial J}{\partial \theta}$ is computed efficiently thanks to backpropagation through the structure (Goller and Kuchler, 1996). L-BFGS (Liu and Nocedal, 1989) is used to minimise the objective function.

3.2 Experiments

We implemented our classifier in Torch7[3] (Collobert et al., 2011a), which is a powerful Matlab-like environment for machine learning. In order to save time, we only trained the classifier on 10-best parses of WSJ02-21. The training phase took six days on a computer with 16 800MHz CPU-cores and 256GB RAM. The word embeddings given by Collobert et al. (2011b)[4] were used as L in Equation 1. Note that these embeddings, which are the result of training a language model neural network on the English Wikipedia and Reuters, have been shown to capture many interesting semantic similarities between words.

We tested the classifier on the development set WSJ22, which contains $1,700$ sentences, and measured the performance in positive rate and negative rate

$$\text{pos-rate} = \frac{\#\text{true\_pos}}{\#\text{true\_pos} + \#\text{false\_neg}}$$

$$\text{neg-rate} = \frac{\#\text{true\_neg}}{\#\text{true\_neg} + \#\text{false\_pos}}$$

The positive/negative rate tells us the rate at which positive/negative examples are correctly labelled *positive/negative*. In order to achieve high performance in the reranking task, the classifier must have a high positive rate as well as a high negative rate. In addition, percentage of positive examples is also interesting because it shows the unbalancedness of the data. Because the accuracy is not a reliable measurement when the dataset is highly unbalanced, we do not show it here. Table 1, Figure 5, and Figure 6 show the classification results.

	pos-rate (%)	neg-rate (%)	%-Pos
gold-std	75.31	-	1
1-best	90.58	64.05	71.61
10-best	93.68	71.24	61.32
50-best	95.00	73.76	56.43

Table 1: Classification results on the WSJ22 and the k-best lists.

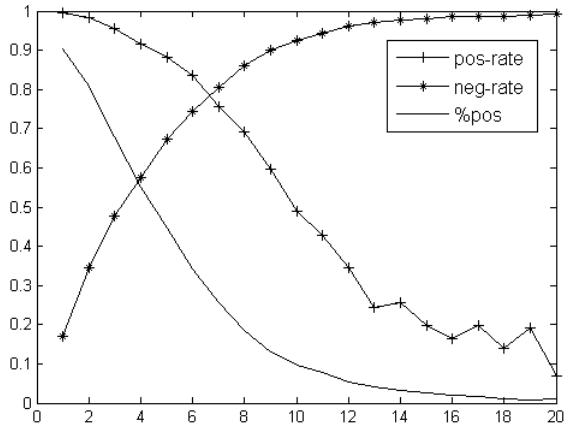

Figure 5: Positive rate, negative rate, and percentage of positive examples w.r.t. subtree depth.

3.3 Discussion

Table 1 shows the classification results on the gold-standard, 1-best, 10-best, and 50-best lists. The positive rate on the gold-standard parses, 75.31%, gives us the upper bound of %-pos when this classifier is used to yield 1-best lists. On the 1-best data, the classifier missed less than one tenth positive subtrees and correctly found nearly two third of the negative ones. That is, our classifier might be useful for avoiding many of the mistakes made by the Berkeley parser, whilst not introducing too many new mistakes of its own. This fact gave us hope to improve parsing performance when using this classifier for reranking.

Figure 5 shows positive rate, negative rate, and percentage of positive examples w.r.t. subtree depth on the 50-best data. We can see that the positive rate is inversely proportional to the subtree depth, unlike the negative rate. That is because the

[3] http://www.torch.ch/

[4] http://ronan.collobert.com/senna/

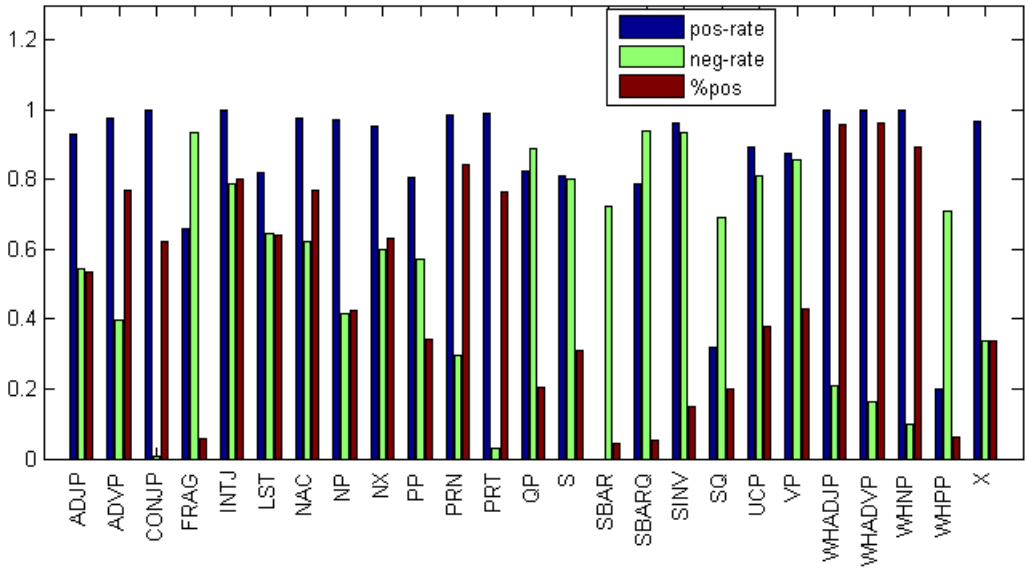

Figure 6: Positive rate, negative rate, and percentage of positive samples w.r.t. syntactic categories (excluding POS tags).

deeper a subtree is, the lower the a priori likelihood that the subtree is positive (we can see this in the percentage-of-positive-example curve). In addition, deep subtrees are difficult to classify because uncertainty is accumulated when propagating from bottom to top.

4 Reranking

In this section, we describe how we use the above classifier for the reranking task. First, we need to represent trees in one vector space, i.e., $\mu(T) = \big(\mu_1(T), ..., \mu_v(T)\big)$ for an arbitrary parse tree T. Collins (2000), Charniak and Johnson (2005), and Johnson and Ural (2010) set the first entry to the model score and the other entries to the number of occurrences of specific discrete hand-chosen properties (e.g., how many times the word *pizza* comes after the word *eat*) of trees. We here do the same with a trick to discretize results from the classifier: we use a 2D histogram to store predicted scores w.r.t. subtree depth. This gives us a flexible way to penalise low score subtrees and reward high score subtrees w.r.t. the performance of the classifier at different depths (see Subsection 3.3). However, unlike the approaches just mentioned, we do not use any expert knowledge for feature selection; instead, this process is fully automatic.

Formally speaking, a vector feature $\mu(T)$ is computed as following. $\mu_1(T)$ is the model score (i.e., max-rule-sum score) given by the parser, $\big(\mu_2(T), ..., \mu_v(T)\big)$ is the histogram of a set of (y, h) where y is given by Equation 2 and h is the depth of the corresponding subtree. The domain of y (i.e., $[0, 1]$) is split into γ_y equal bins whereas the domain of h (i.e., $\{1, 2, 3, ...\}$) is split into γ_h bins such that the i-th ($i < \gamma_h$) bin corresponds to subtrees of depth i and the γ_h-th bin corresponds to subtrees of depth equal or greater than γ_h. The parameters γ_y and γ_h are then estimated on the development set.

After extracting feature vectors for parse trees, we then find a linear ranking function

$$f(T) = w^\top \mu(T)$$

such that

$$f(T_1) > f(T_2) \text{ iff } \textit{fscore}(T_1) > \textit{fscore}(T_2)$$

where *fscore*(.) is the function giving F-score, and $w \in \mathbb{R}^v$ is a weight vector, which is efficiently estimated by SVM ranking (Yu and Kim, 2012). SVM was initially used for binary classification. Its goal is to find the hyperplane which has the largest margin to best separate two example sets. It was then proved to be efficient in solving the ranking task in information retrieval, and in syntactic parsing (Shen and Joshi, 2003; Titov and Henderson, 2006). In our experiments, we used SVM-

Rank[5] (Joachims, 2006), which runs extremely fast (less than two minutes with about $38,000$ 10-best lists).

4.1 Experiments

Using the classifier in Section 3, we implemented the reranker in Torch7, trained it on WSJ02-21. We used WSJ22 to estimate the parameters γ_y and γ_h by the grid search and found that $\gamma_y = 9$ and $\gamma_h = 4$ yielded the best F-score.

Table 2 shows the results of our reranker on 50-best WSJ23 given by the Berkeley parser, using the standard evalb. Our method improves 0.20% on F-score for sentences with all length, and 0.22% for sentences with ≤ 40 words. These differences are statistically significant[6] with *p-value* < 0.003. Our method also improves exact match (0.95% for all sentences as well as for sentences with ≤ 40 words).

Parser	LR	LP	LF	EX
all				
Berkeley parser	89.98	90.25	90.12	37.22
This paper	90.10	90.54	90.32	38.17
Oracle	95.94	96.21	96.08	65.56
≤ 40 words				
Berkeley parser	90.43	90.70	90.56	39.65
This paper	90.57	91.01	90.78	40.50
Oracle	96.47	96.73	96.60	68.51

Table 2: Reranking results on 50-best lists on WSJ23 (LR is labelled recall, LP is labelled precision, LF is labelled F-score, and EX is exact match.)

Table 3 shows the comparison of the three parsers that use the same hybrid reranking approach. On F-score, our method performed 0.1% lower than Socher et al. (2013), and 1.5% better than Menchetti et al. (2005). However, our method achieved the least improvement on F-score over its corresponding baseline. That could be because our baseline parser (i.e., the Berkeley parser) performs much better than the other two baseline parsers; and hence, detecting errors it makes on candidate parse trees is more difficult.

[5] www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

[6] We used the "Significance testing for evaluation statistics" software (http://www.nlpado.de/ sebastian/software/sigf.shtml) given by Padó (2006).

Parser	LF (all)	K-best parser
Menchetti et al. (2005)	88.8 (0.6)	Collins (1999)
Socher et al. (2013)	90.4 (3.8)	PCFG Stanford parser
This paper	90.3 (0.2)	Berkeley parser

Table 3: Comparison of parsers using the same hybrid reranking approach. The numbers in the blankets indicate the improvements on F-score over the corresponding baselines (i.e., the k-best parsers).

5 Conclusions

This paper described a new reranking method which uses semantics in syntactic parsing: a symbolic parser is used to generate a k-best list which is later reranked thanks to parse-correctness scores given by a connectionist compositional-semantics-based classifier. Our classifier uses a recursive neural network, like Socher et al., (2010; 2011), to not only represent phrases in a vector space given parse trees, but also identify if these parse trees are grammatically correct or not.

Tested on WSJ23, our method achieved a statistically significant improvement on F-score (0.20%) as well as on exact match (0.95%). This result, although not comparable to the results reported by Collins (2000), Charniak and Johnson (2005), and Johnson and Ural (2010), shows an advantage of using vector-based compositional semantics to support available state-of-the-art parsers.

One of the limitations of the current paper is the lack of a qualitative analysis of how learnt vector-based semantics has affected the reranking results. Therefore, the need for "compositional semantics" in syntactical parsing may still be doubted. In future work, we will use vector-based semantics together with non-semantic features (e.g., the ones of Charniak and Johnson (2005)) to find out whether the semantic features are truly helpful or they just resemble non-semantic features.

Acknowledgments

We thank two anonymous reviewers for helpful comments.

References

Rens Bod, Remko Scha, and Khalil Sima'an. 2003. *Data-Oriented Parsing*. CSLI Publications, Stanford, CA.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics*, pages 132–139. Association for Computational Linguistics.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the International Workshop on Machine Learning (then Conference)*, pages 175–182.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. 2011a. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011b. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Ronan Collobert. 2011. Deep learning for efficient discriminative parsing. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, volume 1, pages 347–352. IEEE.

Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM.

Mark Johnson and Ahmet Engin Ural. 2010. Reranking the Berkeley and Brown parsers. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 665–668. Association for Computational Linguistics.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, Volume 1*, pages 423–430. Association for Computational Linguistics.

Dong C Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Sauro Menchetti, Fabrizio Costa, Paolo Frasconi, and Massimiliano Pontil. 2005. Wide coverage natural language processing using kernel methods and neural networks for structured data. *Pattern Recogn. Lett.*, 26(12):1896–1906, September.

Sebastian Padó, 2006. *User's guide to `sigf`: Significance testing by approximate randomisation*.

Barbara Partee. 1995. Lexical semantics and compositionality. In L. R. Gleitman and M. Liberman, editors, *Language. An Invitation to Cognitive Science*, volume 1, pages 311–360. MIT Press, Cambridge, MA.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Jordan B Pollack. 1988. Recursive auto-associative memory. *Neural Networks*, 1:122.

Federico Sangati and Willem Zuidema. 2011. Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 84–95. Association for Computational Linguistics.

Libin Shen and Aravind K Joshi. 2003. An SVM based voting algorithm with application to parse reranking. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 9–16. Association for Computational Linguistics.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, Cliff C Lin, Andrew Y Ng, and Christopher D Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, volume 2.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing With Compositional Vector Grammars. In *Proceedings of the ACL conference (to appear)*.

Mark Steedman. 1999. Connectionist sentence processing in perspective. *Cognitive Science*, 23(4):615–634.

Ivan Titov and James Henderson. 2006. Loss minimization in parse reranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 560–567. Association for Computational Linguistics.

Andreas van Cranenburgh, Remko Scha, and Federico Sangati. 2011. Discontinuous Data-Oriented Parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 34–44. Association for Computational Linguistics.

Hwanjo Yu and Sungchul Kim. 2012. SVM tutorial: Classification, regression, and ranking. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, volume 1, pages 479–506. Springer.