TAG Parsing with Neural Networks and Vector Representations of Supertags

Jungo Kasai
Dept. of Computer Science
Yale University
jungo.kasai@yale.edu

Robert Frank
Dept. of Linguistics
Yale University
robert.frank@yale.edu

R. Thomas McCoy
Dept. of Linguistics
Yale University
richard.mccoy@yale.edu

Owen Rambow
DSI
Columbia University
rambow@ccls.columbia.edu

Alexis Nasr
LIF
Université Aix Marseille
Alexis.Nasr@lif.univ-mrs.fr

Abstract

We present supertagging-based models for Tree Adjoining Grammar parsing that use neural network architectures and dense vector representation of supertags (elementary trees) to achieve state-of-the-art performance in unlabeled and labeled attachment scores. The shift-reduce parsing model eschews lexical information entirely, and uses only the 1-best supertags to parse a sentence, providing further support for the claim that supertagging is "almost parsing." We demonstrate that the embedding vector representations the parser induces for supertags possess linguistically interpretable structure, supporting analogies between grammatical structures like those familiar from recent work in distributional semantics. This dense representation of supertags overcomes the drawbacks for statistical models of TAG as compared to CCG parsing, raising the possibility that TAG is a viable alternative for NLP tasks that require the assignment of richer structural descriptions to sentences.

1 Introduction

Recent work has applied Combinatory Categorial Grammar (CCG, Steedman and Baldridge (2011)) to the problem of broad-coverage parsing in order to derive grammatical representations that are sufficiently rich to support tasks requiring deeper representation of a sentence's meaning (Lewis et al., 2015; Reddy et al., 2016; Nadejde et al., 2017). Yet CCG is only one of a number of mildly context-sensitive grammar formalisms that can provide such rich representations, and each has distinct advantages. In this paper we explore the applicability of another formalism, Tree Adjoining Grammar (TAG, Joshi and Schabes (1997)), to the task of broad-coverage parsing.

TAG and CCG share the property of lexicalization: words are associated with elementary units of grammatical structure which are composed during a derivation using one of a small set of operations to produce a parse tree. The task of parsing involves the construction of a derivation tree that encodes the application of this set of actions to a set of elementary lexically-associated objects. TAG differs from CCG in having an even richer set of lexical units, so that the identification of these units in a derivation could be even more informative for subsequent tasks involving semantic interpretation, translation and the like, which have been the focus of CCG-based work.

The elementary units of CCG and TAG (categories for CCG, and elementary trees for TAG) determine a word's combinatory potential, in a way that is not the case for the usual part-of-speech tags used in parsing. Indeed, the assignment of elementary objects to the words in a sentence almost determines the possible parse for a sentence. The near uniqueness of a parse given a sequence of lexical units motivated Bangalore and Joshi (1999) to decompose the parsing problem into two phases: *supertagging*, where elementary objects, or supertags, are assigned to each word, and *stapling*, where these supertags are combined together. They claim that given a perfect supertagger, a parse of a sentence follows from syntactic features provided by the supertags, and therefore, supertagging is "almost parsing." This claim has been confirmed in subsequent work: it has been shown that the task of parsing given a gold sequence of supertags can achieve high accuracy (TAG: (Bangalore et al., 2009; Chung et al., 2016), CCG: (Lewis et al., 2016)). However, it has also been revealed that the difficulty of supertagging, because of the large set of possible supertags, re-

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1713–1723
Copenhagen, Denmark, September 7–11, 2017.

sults in inaccuracies that prevent us from effectively utilizing syntactic information provided by the imperfect set of supertags that are assigned. This problem is even more severe for TAG parsing. TAG differs from CCG in having a smaller set of combinatory operations, but a more varied set of elementary objects: the TAG-annotated version of the Penn Treebank that we use (Chen, 2001) includes 4727 distinct supertags (2165 occur once) while the CCG-annotated version (Hockenmaier and Steedman, 2007) includes 1286 distinct supertags (439 occur once). As a result, building a robust, broad-coverage TAG parser has proven difficult.

In this work, we show that robust supertagging-based parsing of TAG is indeed possible by using a dense representation of supertags that is induced using neural networks. In the first half of the paper, we present a neural network supertagger based on a bi-directional LSTM (BLSTM) architecture, inspired by the work of Xu (2015) and Lewis et al. (2016) in CCG, and we make crucial use of synchronized dropout (Gal and Ghahramani, 2016). This supertagger achieves the state-of-the-art accuracy on the WSJ Penn Treebank. When combined with an existing TAG chart parser (Bangalore et al., 2009), the LSTM-based supertagger already yields state-of-the-art unlabeled and labeled attachment scores.

In the second half of the work, we present a shift-reduce parsing model based on a feed-forward neural network that makes use of dense supertag embeddings. Although this approach has much in common with the approach to shift-reduce CCG parsing taken by Zhang and Clark (2011), it differs in its additive structures in supertag embeddings. When a CCG operation combines two supertags (categories), it yields a resulting category that is typically distinct from the two that are combined, and CCG shift-reduce parsers (e.g. Xu (2015)) make use of this result to guide subsequent actions. When the resulting category is the same as some lexical category assignment (for example when function application over $(S\backslash NP)/NP\ NP$ yields $S\backslash NP$, the same as an intransitive verb), the parser will benefit from sharing statistics across these contexts. For TAG however, substitution or adjoining of one elementary tree into another does not change the nature of the elementary tree into which the operation has taken place. Consequently, the results of partial derivations are not identified with atomic lexical entries, resulting in sparser data. We propose a solution to this problem for TAG by introducing vector representations that are added to the supertag embedding when an operation has been applied to an elementary tree. Not only does this result in a TAG-parser with the best known performance over the WSJ Penn Treebank, but the resulting supertag embeddings turn out to contain linguistically sensible linear structure that we illustrate.

2 TAG Parsing and Dependency Parsing

TAG is a tree-rewriting system, and typically the elementary structures of a TAG are phrase structure trees. Thus, TAG-derived structures are also phrase structure trees. In addition, a TAG derivation also yields a record of the derivational operations (substitutions, adjunctions) used to produce the derived tree. Since these operations are context-free, this record also forms a tree, called the *derivation tree*, whose nodes are the elementary objects and the edges are combinatory operations. If we assume the TAG is lexicalized (i.e., each elementary structure is anchored by at least one terminal symbol), then we can label the nodes of the derivation tree with the tree names and also the anchors of the elementary trees, and we obtain what is *formally* a dependency tree.[1] Since each node is also labeled with an elementary tree from the grammar, we can associate rich linguistic structure with that node, such as passive voice or empty subject.

In addition, it has long been observed that the derivation tree can also be interpreted *linguistically* as a dependency tree (Rambow and Joshi, 1997), if certain assumptions are made about the shape of the elementary trees in the grammar (Frank, 2001). The substitution operation corresponds to the obligatory addition of an argument, and adjunction is used to add adjuncts, as well as function words to a lexical head. The one exception is the treatment of long distance *wh*-movement in TAG. Here, a matrix clause is represented by a *predicative auxiliary tree* which is adjoined into the embedded clause, so that the *wh*-element moved from the embedded clause can still be substituted locally into the tree headed by its verb. As a result, the dependency between the matrix and embedded verbs is inverted relative to the

[1] For the difference between formal and linguistic dependency, see Rambow (2010).

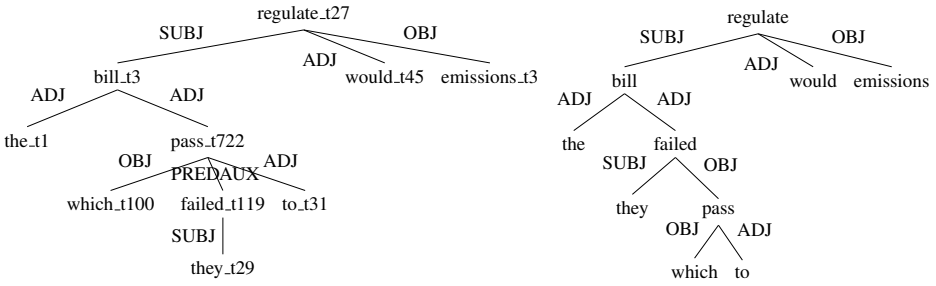


Figure 1: TAG derivation tree (left) and closely related dependency tree (right) for *The bill, which they failed to pass, would regulate emissions*. Substitution edges are labeled SUBJ or OBJ, predicative auxiliary edges are labeled PREDAUX, while all other adjoining edges are labeled ADJ. We use the same edge labels in the dependency tree. The derivation tree also carries the name of the elementary tree used during the derivation, which can be used to look up rich syntactic information about that word in context.

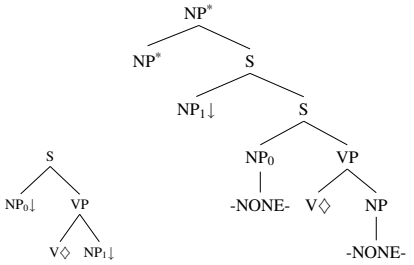


Figure 2: The elementary trees for t27 (left) and t722 (right).

normally assumed dependency. This can be seen in Figure 1, where in the linguistically motivated dependency tree (right) *pass* depends on *failed* as the latter's object, while in the TAG derivation tree (left), *failed* depends on *pass*, linked by an arc marked PREDAUX for predicative auxiliary. These cases can be detected automatically because of the trees used; as a result of this inversion, there is almost no non-projectivity in English. In summary, TAG parsing into derivation trees is very closely related to dependency parsing. In this paper, we are interested in extracting derivation trees, not the derived trees (which can be recovered from the derivation trees).

The corpus we use is obtained by extracting a TAG grammar from the WSJ part of the Penn Treebank corpus, resulting in a grammar and derivation trees labeled with the grammar Chen (2001). For example, in Figure 1, t27 is the basic tree for a transitive verb (*regulate*), while t722 is the tree for a transitive verb which forms an object relative clause with an overt relative pronoun but an empty subject (Figure 2).[2] The corpus and grammar were iteratively refined to obtain linguistically plausible derivation trees which could serve as input for a generation task (Bangalore and Rambow, 2000). As a result, the dependency structure is similar to Universal Dependency (Nivre et al., 2016), apart from the different treatment of long-distance *wh*-movement noted above: the primary dependencies are between the core meaning-bearing lexical words, while function words (auxiliaries, determiners, complementizers) depend on their lexical head and have no dependents.[3] We label verbal argument arcs with deep dependency labels: Subject, Object, and Indirect Object normalized for passive and dative shift. All other arcs are labeled as Adjuncts. This means that our label set is small, but determining the argument labels requires detection of voice alternations and dative shift.

3 TAG Supertagging

3.1 Long-Distance Dependencies and LSTMs

In CCG, a transitive verb is uniformly associated with the category $(S\backslash NP)/NP$, and variation in the word order of a clause is addressed through the use of different combinatory operations. This results in greater parsing ambiguity given a sequence of categories. In TAG, the set of operations is more restricted. While this has the positive effect of reducing parsing ambiguity given a sequence of elementary trees, it necessitates a proliferation in the number of elementary trees. For example, a TAG will associate different elementary trees for the same transitive verb in order to derive canonical clauses, subject and object relatives, and subject and object questions. Not only does this lead to a larger number of supertags, it

[2] Our full grammar is shown at http://mica.lif.univ-mrs.fr/d6.clean2-backup.pdf

[3] One difference should be noted: UD considers prepositions always to be function words, while our TAG grammar treats them as core words unless the Penn Treebank marks them as closely related to the verb.

also means that the determination of the correct supertag requires sensitivity to long-distance dependencies. For example, in the question *Who does Bill think Harry likes?*, the category of the verb *like* requires sensitivity to the first word of the sentence. To address this problem, we make use of a supertagging model that is based on a recurrent network architecture, the Long Short-Term Memory (LSTM, Hochreiter and Schmidhuber (1997)), which is constructed so that its update rule avoids the vanishing/exploding gradient problem.

3.2 Supertagger Model

The model architecture we adopt is depicted in Figure 3, a BLSTM. The input for each word is represented via the concatenation of a 100-dimensional embedding of the word, a 5-dimensional embedding of a predicted part of speech tag, and a 10-dimensional embedding of a suffix vector (which encodes the presence of 1 and 2 character suffixes of the word). We initialize the word embeddings to be the pre-trained GloVe vectors (Pennington et al., 2014); for words which do not have a corresponding GloVe vector, we initialize their embedding to a zero vector. The other embeddings are randomly initialized. Features for each word are fed into the BLSTMs. To produce an output for the network, we concatenate the output vectors from the two LSTM directions and apply an affine transformation before the softmax function to obtain a probability distribution over the 4727 supertags. We train this network, including the embeddings, by optimizing the negative log-likelihood of the observed sequences of supertags in a mini-batch stochastic fashion with the Adam optimization algorithm with $l = 0.001$ (Kingma and Ba, 2015).

Since neural networks have numerous parameters, regularization plays a key role in training. This is typically accomplished by using dropout (Srivastava et al., 2014). Although dropout training has been successful on feed-forward neural networks, performing dropout on recurrent neural networks has been problematic (Gal and Ghahramani, 2016). Armed with a novel interpretation of dropout based on variational inference on parameters, Gal and Ghahramani (2016) propose that dropout noise should be shared across the time steps. We apply this technique to the training of our LSTM network, and achieve an improvement of approximately 2% in accuracy.

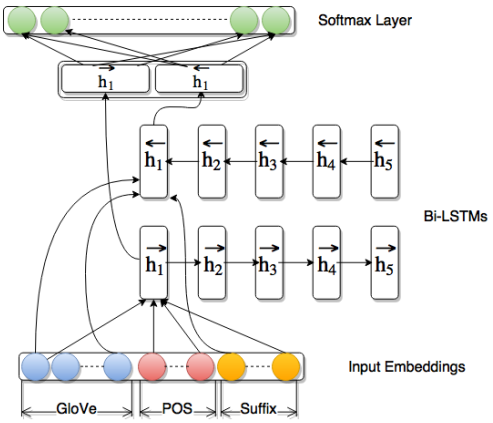


Figure 3: BLSTM Supertagger Architecture.

4 Transition-based Parsing for TAG

As discussed in Section 2, TAG parsing into derivation trees is closely related to dependency parsing; it is natural to make use of techniques from dependency parsing to reconstruct a TAG derivation tree. We make use of this approach here, eschewing complete chart-based parsing algorithms in favor of greedy or beam-search-based explorations of possible parses.

4.1 Shift-Reduce Parsing Algorithm

We employ the arc-eager system of shift-reduce parsing, familiar from the MALT parser (Nivre et al., 2006). In this system, an oracle is trained to predict a sequence of transition operations from an initial state to a terminal state for each sentence. Each state is represented by $c = (s, b, A)$ where s, b, and A denote the stack, buffer and set of dependency relations derived so far. Therefore, our objective is to predict a transition operation given the configuration set c. The initial configuration is defined as $s = [ROOT]$, $b = [w_1, \cdots w_n]$, and $A = \emptyset$ where n is the number of tokens in the sentence $w_1 w_2 \cdots w_n$. At a particular state, denote the top ith element of the stack and the buffer by s_i and b_i respectively. The arc-eager system defines four types of operations with corresponding preconditions: LEFT-ARC, RIGHT-ARC, SHIFT and REDUCE. For the present parser, the LEFT-ARC and RIGHT-ARC operations are each further divided into seven different types depending on the derivational operation involved and the location: Substitution 0-4, Adjoining, and Co-anchor attachment. Substitution n represents an instance of substitution into an argument slot of an elementary tree that is uniquely annotated with the num-

ber n (we discuss the interpretation of such numbers below). Adjoining represents an application of the adjoining operation. It is not further subdivided, as the current parser does not distinguish among different loci of adjoining within an elementary tree. Co-anchor attachment represents the substitution into a node that is construed as a co-head of an elementary tree. An example of this is the insertion of a particle into a verbally headed tree associated with a verb-particle construction, such as the insertion of *up* into the *pick*-headed tree to generate *I picked up the book*. The transitions terminate when the buffer is empty.

This system will fail to capture non-projective TAG derivation structures. However, as noted in Section 2, there is almost no non-projectivity in TAG derivation structures of English. Concretely, we find that WSJ Sections 01-22 contain only 26 non-projective sentences (0.065%), and those sentences are discarded during training. WSJ Section 00 does not have any non-projective sentences.

On the other hand, WSJ Sections 01-22 contain 0.6% of non-projective sentences in dependency grammar (Chen and Manning, 2014), an order of magnitude more than non-projectivity for TAG. This suggests that the problem of TAG parsing is more compatible with standard shift-reduce parsing than dependency grammar parsing is.[4]

4.2 Parser Model

In this work, we use a non-lexicalized parser, which does not have access to the identities of the words of the sentence to be parsed.[5] Instead, the parser's decisions will be guided by the supertags of the top k elements from the stack and the first k elements of the buffer. Using these features as input, we build a two-layer feed-forward network to predict the action for the parser to take. As noted above, the identity of the supertag does not allow the parser to encode whether a particular node in an elementary tree has already been targeted by a substitution operation. In order to overcome this deficiency, we augment the parser's state with *substitution memory*, which encodes for each possible substitution site (from 0 to 4) in a supertag T whether that substitution has already applied in T.

Each supertag is mapped to a d-dimensional vector by an embedding matrix $E \in \mathbb{R}^{d\times(N+2)}$ where N denotes the number of supertags; we also have additional vectors representing the empty state and $ROOT$. Substitution memory is similarly transformed, with a *substitution memory embedding matrix* $M \in \mathbb{R}^{d\times 5}$, to a d-dimensional vector that encodes in a distributed manner where substitution has applied. Each column in M is the vector corresponding to a specific substitution type. Each element from the stack and buffer is then represented by adding the supertag T embedding to the embedding associated with each vector from M corresponding to the substitution operations already performed on T, if any. Mathematically, suppose that we are at the configuration $c = (s, b, A)$, and $p^{(i)} \in \mathbb{R}^5$ denotes the substitution history of s_i. $p^{(i)}$ is an indicator vector that $p^{(i)}_j = 1$ if and only if we have already performed substitution j into s_i in the parser, and 0 otherwise. Define $p^{(k+1)}$ in the same way for b_1.[6] Then, the input vector to the network can be expressed as

$$[Es_1 + Mp^{(1)}; \cdots ; Es_k + Mp^{(k)}; \\ Eb_1 + Mp^{(k+1)}; \cdots ; Eb_k]$$

This model with the additive substitution memory has several conceptual advantages. First, the additive structure gives us an unbounded scope of the past transition, avoiding making decisions that lead to substitution collisions without a computationally expensive architecture such as an ensemble of LSTMs (Xu, 2015). Moreover, as TAG supertags encode rich syntactic features, the parsing data for some supertags tend to become scarce. The most common 300 supertags in the Penn Tree Bank WSJ Sections 01-22 cover 96.8% of the data. In a situation of such data sparsity, it becomes crucial to link, for example, the behaviors of intransitive verbs with those of transitive verbs. With substitution memory, the network can develop representations under which addition of appropriate substitution vectors serves to transform

[4] We recognize alternatives to shift-reduce parsing. For instance, Dozat and Manning (2017) propose a graph-based parser that accommodates non-projectivity. It remains open whether such alternatives will work for TAG parsing, and we leave this for the future. We emphasize, however, that because of the nature of TAG derivations, the issue of non-projectivity is much less severe than dependency parsing.

[5] We have tried adding word embeddings as inputs to the parser with different choices of hyperparameters (e.g., the number of embedding dimensions). Unfortunately, our experiments yielded degraded performance. It should be noted, however, that TAG supertags typically provide enough information for deriving correct parses; the only cases that supertags cannot disambiguate are ambiguous attachments to identical nonterminals (e.g. *The picture of my friend with green eyes*).

[6] Notice that no substitution should have happened on $b_2, b_3, \ldots$ by construction.

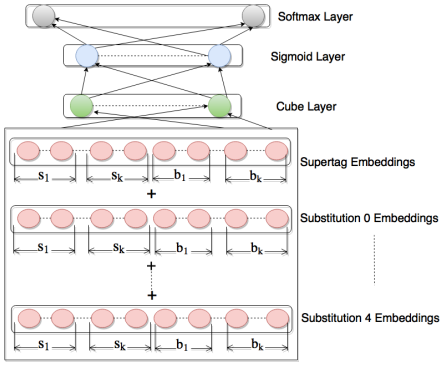


Figure 4: Shift-Reduce Parser Neural Network Architecture.

one supertag into another, allowing the generalization across these contexts. Indeed, as we will show in a later section, the substitution memory embeddings and supertag embeddings turn out to yield interpretable and linguistically sensible structures.

Finally, we concatenate the vectors associated with the relevant elements from the stack and buffer into a $2dk$ dimensional vector and feed it to the network to obtain a probability distribution over the possible transition actions. The architecture is visualized in Figure 4. Following Chen and Manning (2014), we use the cube activation function for the first layer, which could better capture interactions. We, again, optimize the negative log-likelihood in a mini-batch stochastic fashion with the Adam optimization algorithm with $l = 0.001$ (Kingma and Ba, 2015). With regards to decoding, we consider both greedy parsing as well as a beam search algorithm, where we keep transition action hypotheses at each time step, in the experiments we report below.

4.3 Supertag Input to the Parser

We consider three types of supertag inputs to the neural network parser: gold supertags, 1-best supertags from the BLSTM supertagger, and 1-best supertags from the MICA chart parser (Bangalore et al., 2009). MICA searches through n-best supertags with their corresponding probabilities and produces a full parse forest that abides by the TAG grammar. To generate the 1-best supertags from MICA, we first feed 10-best supertags from the BLSTM supertagger to the MICA chart parser, and retain only the supertags of the best parse. These supertags have the special property that there exists a feasible parse in the TAG grammar for every sentence, which does not necessarily hold for the 1-best supertags from the BLSTM supertagger.

5 Experiments

5.1 Experimental Setups

In order to ensure comparability with past work on TAG parsing, we follow the protocol of Bangalore et al. (2009) and Chung et al. (2016), and use the grammar and the TAG-annotated WSJ Penn Tree Bank described in Section 2. Following that work, we use Sections 01-22 as the training set, Section 00 as the development set, and Section 23 as the test set. The training, development, and test sets comprise 39832, 1921, and 2415 sentences, respectively. The development set contains 177 sentences with at least one supertag that was absent from the training set. We implement the networks in TensorFlow (Abadi et al., 2015). During training, we shuffle the order of the sentences in the training set to form mini-batches. Each mini-batch consists of 100 sentences, except the last which contains 32 sentences.

For supertagging, we first generate predicted POS tags for both the training set and the development set. The POS-tagger architecture is similar to that of the supertagger shown in Figure 3, except that, obviously, we do not feed it POS embeddings. The BLSTMs each contain 128 units, and we do not apply dropout at this stage. To derive predicted POS tags for the supertagger training set, we perform 10-fold jackknife training over the training set. For the supertagger, each direction of LSTM computation involves two layers, and each LSTM contains 512 units. The hidden units, layer-to-layer, and input units dropout rates are 0.5, 0.5, and 0.2 respectively. After each training epoch, we test the parser on the development set. When classification accuracy does not improve on two consecutive epochs, we end the training.

For the parser, we initialize the supertag embedding matrix E and the substitution memory embedding matrix M according to $\mathit{Uniform}(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$. For all of the experiments reported here, we fix the hyper-parameters as follows: the embedding dimensions d for the supertag and substitution memory embeddings are 50, the number of units is 200 on both of the two hidden layers, and the input dropout rate is 0.2 and the hidden dropout rate is 0.3. We choose $k = 3$ or 5 for the stack/buffer scope. After each training epoch, we test the parser on the development set, and when the greedy accuracy

fails to improving on two consecutive epochs, we terminate the training.

5.2 Supertagging Results

We achieve on Section 00 supertagging accuracy of 89.32%, 90.67% if we disregard the 177 sentences that contain an unseen supertag. This performance surpasses previous results on this task: Bangalore et al. (2009) report 88.52% accuracy using a maxent supertagger combined with a chart parser (MICA), which is the best result over a tag set of this complexity, though better results are reported for considerably smaller tag sets (on the order of 300 supertags). The n-best and β pruning accuracy (Clark and Curran, 2007) are given in Figure 5. In the β pruning scheme, we pick supertags whose probabilities are greater than β times the probability of the most likely supertag. We show the results for $\beta \in [0.075, 0.03, 0.01, 0.005, 0.001, 0.0001]$. It is noteworthy that with $\beta = 0.005$, the average number of supertags picked for each token (ambiguity level) is about 2, but the accuracy surpasses 98%, suggesting that incorporating the β pruning method in the *stapling* phase of TAG parsing will enhance the parser. We also obtain comparable accuracy of 89.44% on Section 23.

As discussed above, TAG supertags alone provide rich syntactic information. In order to understand how much such information our supertagger sucessfully captures, we analyze the 1-best supertag results on the basis of the syntactic properties of the elementary trees defined in Chung et al. (2016). Extending the notion of binary precision and recall, we define the macro-averaging precision and recall as the simple average over precision or recall corresponding to each class (Sokolova and Lapalme, 2009). We also compute accuracy, which is simply the ratio of correctly classified examples to the entire number of examples. Table 1 shows the results along with those for the maxent supertagger (Bangalore et al., 2009). Recall tends to be lower than precision; we can attribute this pattern to the nature of the macro-averaging scheme that equally treats each class regardless of the size; poor recall performance on a small class, such as the class of dative shift verbs, influences the overall recall as much as performance on a large class. Observe, however, that the BLSTM supertagger yields significantly better performance on recall in general, and it outperforms the maxent supertagger by a large margin in handling long dependencies of *wh*-movement and relativization.

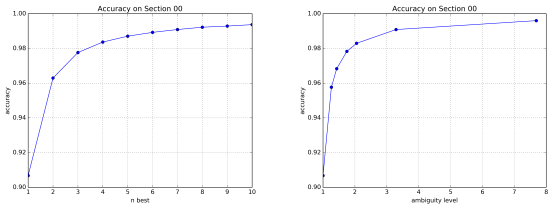


Figure 5: Section 00 n-best accuracy (left), and β pruning accuracy (right). Sentences with unseen supertags are disregarded.

		MICA			BLSTM		
Property	#	Prec	Rec	Acc	Pre.	Rec	Acc
root	42	**88.8**	68.3	95.4	80.4	**72.9**	**95.9**
coanc	4	**89.7**	64.0	**99.2**	65.3	**64.7**	**99.2**
modif	28	**82.2**	57.3	92.4	73.6	**63.6**	**93.7**
dir	3	95.9	95.9	96.0	**96.7**	**96.6**	**96.7**
predaux	2	80.0	67.6	**100.0**	**83.3**	**85.3**	**100.0**
pred	2	**93.3**	90.6	99.6	**93.3**	**93.1**	**99.7**
comp	3	92.9	61.4	99.7	**95.9**	**63.3**	**99.8**
particle	3	**94.0**	92.1	97.1	92.8	**92.5**	**97.5**
particleShift	3	**89.3**	**77.7**	**99.9**	77.5	77.0	**99.9**
voice	4	**94.6**	92.7	**99.4**	94.4	**94.7**	**99.4**
wh	4	93.0	79.6	97.1	**94.5**	**84.7**	**97.6**
rel	6	68.4	71.2	96.5	**88.9**	**73.6**	**97.2**
esubj	3	94.0	94.0	96.9	**95.4**	**95.3**	**97.4**
datshift	3	92.8	45.3	**99.9**	**96.9**	**53.3**	**99.9**

Table 1: 1-best Supertag Analysis on Section 00. # indicates the number of possible classes in a property. The Prec and Rec columns show macro-averaging precision and recall. The Acc columns indicate simple accuracy. For a complete description of the properties, see Chung et al. (2016).

Lastly, we interpret our supertagging performance in the context of prepositional phrase (PP) attachment ambiguity. Normally, in dependency parsing, PP attachment is resolved by the parser. However, in our case, it can be resolved before parsing, during the supertagging step. This is because the supertags for prepositions vary depending on the type of constituent modified by the PP containing the preposition; for example, $t4$ is the supertag for a preposition whose PP modifies an NP, while $t13$ is the supertag for a preposition whose PP modifies a VP.

To test how well our supertagger resolves PP attachment ambiguity, we used the dataset from Ratnaparkhi et al. (1994) (derived from the PTB WSJ) to extract a test set of sentences with PPs that are ambiguous between attaching to a VP or to an NP.[7]

[7]We were unable to use the full test set because, in order to run the supertagger on the test set, we had to map the test examples back to their full sentences, but some of those original sentences are no longer available in PTB3.

We then supertagged these sentences and checked whether the supertag for the preposition in the ambiguous PP is a VP modifier or an NP modifier. Of our test set of 1951 sentences, 1616 had supertags modifying the correct part of speech, to give an accuracy of 0.826. Table 2 compares this result to past work. The supertagger outperforms all other models besides the Word Vector model. Since this Word Vector model (like the MaxEnt model) is specifically trained for this task, and given that our supertagger is not trained for this particular task, the accuracy is reasonably encouraging. This result suggests that TAG supertagging is a reasonable intermediate level between only resolving PP attachment and conducting full parsing.

System	PP Attachment Accuracy
Malt (Nivre et al., 2006)	79.7*
MaxEnt (Ratnaparkhi et al., 1994)	81.6*
Word Vector (Belinkov et al., 2014)	**88.7***
Parsey McParseface (Andor et al., 2016)	82.3
BLSTM Supertagger	82.6

Table 2: Various PP attachment results. * denotes the results on a different dataset.

5.3 Parsing Results

Parsing results and comparison with prior models are summarized in Tables 3, 4 (Section 00), and 5 (Section 23). From Table 4, we see that the combination of the BLSTM supertagger, MICA chart parser, and the neural network parser achieves state-of-the-art performance, even compared to parsers that make use of lexical information, POS tags, and hand-engineered features. With gold supertags, the neural network parser with beam size 16 performs slightly better than the chart parser. As shown in Table 5, our supertag-based parser outperforms SyntaxNet (Andor et al., 2016) with the computationally expensive global normalization. This suggests that, besides providing the grammars and linguistic features that can be used in downstream tasks in addition to derivation trees (Semantic Role Labeling: (Chen and Rambow, 2003), Textual Entailments: (Xu et al., 2017)), supertagging also improves parsing performance.

5.4 Learned Vector Representation

We motivated the use of embeddings in the parser to encode properties of the supertags and the substitution operations performed on them. We can examine their structure in a way similar to what Mikolov et al. (2013) did for word embeddings by performing analogy tests on the learned supertag embeddings. Consider, for example, the analogy that an elementary tree representing a clause headed by a transitive verb (t27) is to a clause headed by an intransitive verb (t81) as a subject relative clause headed by a transitive verb (t99) is to a subject relative headed by an intransitive verb (t109). Following Mikolov et al. (2013), we can express this analogy with the equation $t27 - t81 \approx t99 - t109$, which can be rearranged as $t27 - t81 + t109 \approx t99$. By seeing if this approximate equality holds when the embeddings of the relevant supertags have been added and subtracted, we can test how well the embeddings capture syntactic properties of the supertags.

To create a set of such analogies, we extracted all pairs (stag1, stag2) such that stag2 is the result of excising exactly one substitution node from stag1. The idea here is that, once a substitution node is filled within a supertag, the result behaves like a supertag without that substitution node; for example, a transitive verb with its object filled behaves like an intransitive verb. We then create analogies by choosing two such pairs, (stag1, stag2) and (stag3, stag4), chosen so that stag1 and stag2 are related in the same way that stag3 and stag4 are related. From these two pairs we then form an equation of the form stag1 $-$ stag2 $+$ stag4 $\approx$ stag3.

We considered three different criteria for choosing which pairs of pairs can form analogies: **A-1**, where both pairs must have the same deep syntactic role (Drole) for the excised substitution node; **A-2**, where both pairs must have the same Drole and POS for the excised substitution node; and **A-3**, where both pairs must have the same Drole and same POS for the excised substitution node, and the heads of all supertags in the analogy must have the same POS. For each analogy generated, we computed the left hand side by adding and subtracting the relevant supertag embeddings and used cosine similarity to determine the most similar embeddings to the result and whether the intended right hand side was among the closest neighbors. We used four metrics for evaluation: *Acc*, the proportion of analogies for which the closest neighbor was the correct supertag; *Acc-300*, the proportion of analogies for which the closest neighbor amongst the 300 most common supertags was the correct supertag; *Avg Rank*, the average position of the correct choice in the ranked list of the closest neighbors; and *Avg Rank-300*,

k	B	Gold Stags UAS	Gold Stags LAS	BLSTM UAS	BLSTM LAS	BLSTM+Chart UAS	BLSTM+Chart LAS
3	1	$96.74_{\pm 0.06}$	$96.47_{\pm 0.06}$	$89.54_{\pm 0.03}$	$88.06_{\pm 0.04}$	$90.03_{\pm 0.02}$	$88.56_{\pm 0.02}$
3	16	$97.62_{\pm 0.06}$	$97.42_{\pm 0.07}$	$90.31_{\pm 0.04}$	$88.85_{\pm 0.04}$	$90.85_{\pm 0.02}$	$89.38_{\pm 0.02}$
5	1	$96.96_{\pm 0.19}$	$96.67_{\pm 0.20}$	$89.63_{\pm 0.03}$	$88.12_{\pm 0.04}$	$90.07_{\pm 0.06}$	$88.60_{\pm 0.06}$
5	16	$\mathbf{97.68}_{\pm 0.06}$	$\mathbf{97.46}_{\pm 0.05}$	$\underline{90.38}_{\pm 0.05}$	$\underline{88.92}_{\pm 0.04}$	$\mathbf{90.88}_{\pm 0.06}$	$\mathbf{89.39}_{\pm 0.06}$

Table 3: Parsing Results on Section 00. k is # of elements from stack and buffer used as input, B is the beam size. We show mean and standard deviation over 5 trials with different initialization for each configuration. BLSTM+Chart shows results obtained by feeding the 1-best supertag inputs from the MICA chart parser discussed in Section 4.3.

Parser	Features	Gold Stags UAS	Gold Stags LAS	Predicted Stags Stag Acc	Predicted Stags UAS	Predicted Stags LAS
MALT-Stag	Words, POS, Stags (1-best)	97.20*	96.90*	88.52	88.50*	86.80*
Maxent+Chart (MICA)	Stags (10-best)	97.60	97.30	88.52	89.32	85.80
P3	Words, POS, Stags (1-best), Stag features	97.46*	96.51*	87.88	89.96*	87.86*
BLSTM+Chart	Stags (10-best)			**89.32**	90.05	88.32
BLSTM+NN	Stags (1-best)	$\mathbf{97.68}_{\pm 0.06}$	$\mathbf{97.46}_{\pm 0.05}$	**89.32**	$90.38_{\pm 0.05}$	$88.92_{\pm 0.04}$
BLSTM+Chart+NN	Stags (1-best)	–	–	89.31	$\mathbf{90.88}_{\pm 0.06}$	$\mathbf{89.39}_{\pm 0.06}$

Table 4: Section 00 Performance Comparison with Prior Models. The P3 results are from Chung et al. (2016). P3 is based on the model described in Nivre et al. (2004). * denotes the results with gold POS tags. For the NN parser, k=5 and B=16.

Model	Stag Acc	UAS	LAS
SyntaxNet	–	$90.47_{\pm 0.05}$	$88.99_{\pm 0.06}$
Maxent+Chart	86.85	86.66	84.90
BLSTM+Chart	89.44	90.20	88.66
BLSTM+NN	89.44	$90.31_{\pm 0.03}$	$88.98_{\pm 0.03}$
BLSTM+Chart+NN	**89.71**	$\mathbf{90.97}_{\pm 0.03}$	$\mathbf{89.68}_{\pm 0.03}$

Table 5: Supertagging and Parsing Results on Section 23. For the NN parser, k=5 and B=16 throughout. We trained Syntaxnet (Andor et al., 2016) with global normalization beam size 16 using the TensorFlow toolkit.

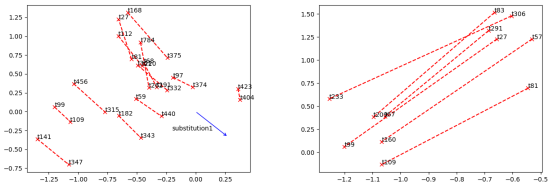
the average position of the correct choice in the ranked list of the closest neighbors amongst the 300 most common supertags.

We expect that the embeddings for common supertags would be better representations than embeddings for rare supertags. Thus, we restricted our experiment to analogies between supertags among the 300 most common ones in the training set. (Indeed, experiments that included rare supertags in the analogies produced poor results.)

Table 6 provides the results for the 3 types of analogies, which are very promising, particularly type A-3. We can visualize these results by performing PCA on the embedding vectors. Figure 6a shows the first 2 PCA components of A-3 analogies involving supertags containing transitive and intransitive predicates across a variety of structures. We see that virtually all pairs differ from one another by a similar vector, and in fact this difference is essentially the vector associated with substitution 1 in the substitution embedding memory (shown in blue). Figure 6b shows the case of pairs of canonical sentence elementary trees (*read* in *I read the book*) and their subject relative analogs (*read* in *the guy who read the book*). This again shows a systematic mapping between grammatically related embeddings, suggesting that the embeddings encode relevant structural properties.

Type	Acc	Acc-300	Avg Rank	Avg Rank-300
A-1	0.20	0.28	49.5	10.2
A-2	0.44	0.60	17.4	3.68
A-3	0.61	**0.81**	2.26	**1.38**

Table 6: Analogy Task Results.



(a) Transitive/intransitive (b) Declarative/relative

Figure 6: Embedding vector alignments.

6 Conclusions and Future Work

We presented a state-of-the-art TAG supertagger and parser, the former based on a BLSTM architecture, and the latter on a non-lexicalized shift-reduce parser using a feed-forward network. The parser makes crucial use of supertag embeddings that provide linguistically interpretable vector representations of the supertags. These positive results suggest that TAG can provide the foundation of NLP systems for tasks requiring deeper analysis than current dependency parsers provide, and we will apply our parser to such tasks in the future. Nonetheless, a large discrepancy remains in parser performance with gold supertags and predicted supertags, indicating that supertagging is still a bottleneck. We will explore ways to leverage our supertagger's high β-pruning accuracy in parsing.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. http://tensorflow.org/.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of Association for Computational Linguistics*.

Srinivas Bangalore, Pierre Boullier, Alexis Nasr, Owen Rambow, and Benoît Sagot. 2009. MICA: A Probabilistic Dependency Parser Based on Tree Insertion Grammars. In *NAACL HLT 2009 (Short Papers)*.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics* 25:237–266.

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*. Saarbrücken, Germany.

Yonatan Belinkov, Tao Lei, Regina Barzilay, and Amir Globerson. 2014. Exploring compositional architectures and word vector representations for prepositional phrase attachment. *Transactions of the Association for Computational Linguistics* (2):561–572.

Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

John Chen. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

John Chen and Owen Rambow. 2003. Use of Deep Linguistics Features for the Recognition and Labeling of Semantic Arguments. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Wonchang Chung, Suhas Siddhesh Mhatre, Alexis Nasr, Owen Rambow, and Srinivas Bangalore. 2016. Revisiting supertagging and parsing: How to use supertags in transition-based parsing. In *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12)*. pages 85–92.

Stephen Clark and James R. Curran. 2007. Wide-coverage semantic representations from a CCG parser. *Computational Linguistics* 4.

Timothy Dozat and Christopher Manning. 2017. Deep biaffine attention for neural dependency parsing. In *ICLR*.

Robert Frank. 2001. *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, Cambridge, Mass.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pages 1019–1027. http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

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.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, Springer, New York, pages 69–124.

Diederik P. Kingma and Jimmy Lei Ba. 2015. ADAM: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint a* CCG parsing and semantic role labeling". In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG Parsing. In *Proceedings of NAACL-HLT 2016*. pages 221–231.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, Curran Associates, Inc., pages 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Maria Nadejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch. 2017. Syntax-aware neural machine translation using CCG. ArXiv Preprint 1702.01147v1.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association (ELRA), Paris, France.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*. pages 49–56.

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

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 1532–1543.

Owen Rambow. 2010. The simple truth about dependency and phrase structure representations: An opinion piece. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Los Angeles, California, pages 337–340. http://www.aclweb.org/anthology/N10-1049.

Owen Rambow and Aravind Joshi. 1997. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In Leo Wanner, editor, *Recent Trends in Meaning-Text Theory*, John Benjamins, Amsterdam and Philadelphia, pages 167–190.

Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. 1994. A maximum entropy model for prepositional phrase attachment. In *ARPA Human Language Technology Workshop*. pages 250–255.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics* 4:127–140.

Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing and Management* 45:427–437.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15* (15):1929–1958.

Mark Steedman and Jason Baldridge. 2011. Combinatory categorial grammar. In Robert Borsley and Kersti Börjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, Wiley-Blackwell.

Pauli Xu, Robert Frank, Jungo Kasai, and Owen Rambow. 2017. TAG parsing evaluation using textual entailments. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+13)*.

Wenduan Xu. 2015. LSTM shift-reduce CCG parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1754–1764.

Yue Zhang and Stephen Clark. 2011. Shift-Reduce CCG Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 683–692.