Linguistically Rich Vector Representations of Supertags for TAG Parsing

Dan Friedman[*1] **Jungo Kasai**[*1] **R. Thomas McCoy**[*1]
Robert Frank[1] **Forrest Davis**[2] **Owen Rambow**[3]
[1]Department of Linguistics, Yale University
[2] Columbia University
[3] DSI, Columbia University
{dan.friedman,jungo.kasai,richard.mccoy,robert.frank}@yale.edu
fld2111@columbia.edu
rambow@ccls.columbia.edu

Abstract

In this paper, we explore several techniques for producing vector representations of TAG supertags that can be used as inputs to a neural network-based TAG parser. In the simplest case, the supertag is encoded as a 1-hot vector that is projected to a dense vector. Secondly, we use a tree-recursive neural network that is given as input the structure of the elementary tree. Thirdly, we use hand-crafted feature vectors that describe the syntactic features of each supertag, and project these to a dense vector. These three representations are learned during the training of a neural network TAG parser with a layer that embeds supertags in a low-dimensional space. Finally, we consider an embedding that is trained only on patterns of linear co-occurrence among supertags. By testing the resulting vector representations on the task of completing syntactic analogies, we show that these vector representations capture syntactically relevant information. While our linguistically-informed embeddings outperform atomic embeddings on the syntactic analogy task, we find that the same embeddings lead to only a slight improvement on the task of TAG parsing, indicating that the neural parser is able to induce useful representations of supertags from the data alone.

1 Introduction

In a Tree Adjoining Grammar (TAG), the set of elementary trees can be thought of as the possible lexical grammatical category assignments, much like the part of speech tags in a Context Free Grammar (CFG). However, the number of elementary trees is considerably larger than the category set typically found in other formalisms. In the TAG that is extracted from the Penn Treebank by Chen (2001), there are more than 4,700 distinct elementary trees, as compared to the 48 POS tags found in the Penn Treebank or even the 1,286 categories found in the Combinatory Categorial Grammar (CCG) bank (Hockenmaier and Steedman, 2007). While this is indeed a large number, the set of elementary trees in a linguistically adequate TAG is finely structured, with systematic relationships holding between elementary trees. Past work on grammar development in TAG has explored a variety of methods for capturing the relationships among and within so-called tree families (Vijay-Shanker and Schabes, 1992; Evans et al., 1995; XTAG Research Group, 1998; Becker, 2000), where all members of a tree family have the same basic argument structure (or have the same value for some other syntactic dimension) but differ from each other based on transformations such as passivization or *wh*-movement.

Under the approach suggested by Bangalore and Joshi (1999), the first step of TAG parsing, called supertagging, involves the assignment of elementary trees to lexical items. Given the high degree of supertag ambiguity and the fact that state-of-the-art TAG supertagging accuracy is only around 90% (using the bi-LISTM supertagger reported in Kasai et al. (2017)) as compared to 95% for CCG supertagging (Lewis et al., 2016), it is useful indeed if the parser can be made sensitive to the relationships between elementary trees. Certain errors in the supertagger might then not prove fatal to a parser if a single supertag can be interpreted as related to other elementary trees, providing potentially useful derivational options. Furthermore, if relations among supertags are encoded, problems of data sparsity during training

[*]Equal Contribution.

Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+13), pages 122–131,
Umeå, Sweden, September 4–6, 2017.

might be overcome; nearly half of the supertags present in the PTB WSJ Sections 1-22 appear only once, but they may be related to other supertags that occur more frequently.

Previous work by Chung et al. (2016) has proposed a way of exploiting relationships among supertags in a transition-based parser, by adding a series of hand-coded linguistic features that characterize properties of the elementary trees in the grammar. Such features had a beneficial effect on parser performance when used in conjunction with lexical identity, supertag identity, and POS tag.

In this paper, we demonstrate how the use of a neural network TAG parsing model, proposed by Kasai et al. (2017), facilitates the representation of similarity among supertags. The input to this parser is a sequence of 1-best supertags output by a bidirectional LSTM supertagger. As the first step in computation, the parsing network maps each supertag into a vector via an embedding matrix. Given a set of supertag vectors, we can study the similarity relations among them using methods similar to those that have been applied to lexical vectors by Mikolov et al. (2013a,b). For example, we can consider analogies between elementary trees that correspond to an operation of detransitivization, by asking whether an elementary tree representing a clause headed by a transitive verb (t27) stands in the same relationship to an elementary tree headed by an intransitive verb (t81) that a subject relative clause elementary tree headed by a transitive verb (t99) stands in to a subject relative headed by an intransitive verb (t109). By interpreting these elementary trees as vectors, we can express this analogy by $t27 - t81 + t109 \approx t99$. As we will demonstrate below, this formalization allows us to study the degree to which a representational scheme successfully captures a wide range of linguistic relationships among elementary trees.

Our discussion will compare four alternatives for constructing supertag embeddings. Three of these are trained in conjunction with parser, and differ only in the representation of the supertag input to the parser: atomic encodings of supertag identity, recursive encoding of the structure of the elementary tree, and the hand-coded linguistic features from Chung et al. (2016). The fourth derives embeddings via a GloVe-type model of distributional similarity (Pennington et al., 2014).

Recent work by McMahan and Stone (2016) has proposed a method to embed TAG supertags in the context of natural langauge generation. They utilize structural information of elementary trees through convolutional neural networks. Our recursive encoding is similar to their approach in that the embedding procedure respects tree structure of supertags. It should be noted, however, that our induction process runs in the opposite direction from that in McMahan and Stone (2016). In their application to natural language generation, the objective is surface realization from (unlabeled) dependency trees, whereas the problem of interest in this paper is derivation of dependency trees from surface realization.

In the next section, we briefly describe the parsing model that is the foundation of our experiments, along with our four methods for constructing supertag embeddings. Section 3 lays out our experimental set-up and Section 4 explains how we perform evaluation for supertag similarity and for parsing. Section 5 reports the results and discusses their implications.

2 Parsing Model and Embedding Construction

2.1 Neural Network TAG Parser

In our experiments, we make use of the neural network TAG parser from Kasai et al. (2017). This is an arc-eager shift-reduce parser that uses a feed-forward neural network as an oracle. At each time step, the oracle takes as input the configuration of the parser, which consists of a fixed number of cells from the top of the stack and the front of the buffer, each containing a supertag. The parser's task is to construct a derivation tree from the individual supertags. This derivation is constructed in the usual way for transition-based parsers, namely through a series of actions (shift, reduce, left-arc, and right-arc). Left-arc and right-arc create links in the derivation tree, and these operations are further specified by the type of operation (substitution and adjoining) as well as the node within the elementary tree to which the operation applies (specified for substitution as 0-4, encodings of the deep grammatical role of the substitution site). The output of the network is a softmax layer, whose activations can be interpreted as a probability distribution over actions and labels. The parser is unlexicalized; the only information that the parser uses to determine its action is the supertags in the relevant cells of the stack and buffer. For the current work, we make use of the bi-LSTM supertag-

ger discussed in Kasai et al. (2017), which is pre-trained on the WSJ Penn Treebank and does not vary across the experiments reported here. This supertagger provides as output a distribution over possible supertags for each word in a sentence.

Our focus in this paper is the first layer of the neural network, which maps each supertag in a parser configuration to a vector in a low-dimensional space. The input to the subsequent feed-forward layer is the concatenation of the dense vectors associated with the relevant cells in the stack and buffer. In our experiments, we vary the representation of the supertag that is provided as input to the parser, and retrain. Our hypothesis is that using a more linguistically informed embedding function will produce linguistically interpretable vector representations and improve parsing accuracy.

2.2 Atomic Embedding

The first type of embedding function we consider is the one from Chen and Manning (2014) (POS tags), Lewis et al. (2016); Xu (2016) (CCG supertags), and Kasai et al. (2017) (TAG supertags): here, each supertag in a parse configuration is represented as a one-hot column vector in $\mathbb{R}^{|V|}$, where $|V|$ is the total number of supertags. That is, each supertag is represented as a vector in which all entries are 0 except for a single entry, which is 1, corresponding to the integer ID of the supertag. The supertags are then projected into a lower-dimensional space by multiplying the one-hot vectors with an embedding matrix $L \in M_{d\times|V|}$. Thus each supertag t is associated with a distinct vector $x_t \in \mathbb{R}^d$, corresponding to a column in the embedding matrix L. The embeddings are then concatenated and passed to the feed-forward network. The embedding matrix L is trained jointly with the parser via the back-propagation algorithm to optimize the negative log-likelihood of outputting the correct parser actions. As a result, although the 1-hot encoding does not convey any information about relationships between supertags, the optimization process may favor an embedding matrix where the vector encoding of the supertag does convey such information to the degree to which similar supertags are best treated similarly by the parser.

2.3 Recursive Tree-Based Embedding

There are several theoretical drawbacks to using an atomic representation of supertags for parsing. First, the parser makes no use of the linguistic meaning of supertags. Each supertag is considered to be a distinct entity, and the only way for the parser to associate similar supertags is to learn associations in the process of optimizing the training objective. Second, supertag data is sparse: of the 4,724 supertags in the TAG-annotated version of the Penn Treebank, 2,165 occur only once (Chen, 2001; Kasai et al., 2017). Because each supertag is considered to be distinct in the input layer, the parser will learn little information about nearly half of all supertags.

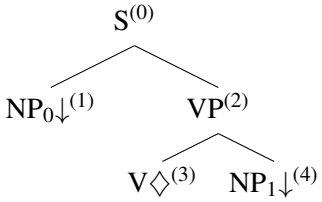
For these reasons we next consider a tree-recursive embedding layer that associates each supertag with a low-dimensional vector by passing the corresponding elementary tree through a recursive neural network. A recursive neural network (RNN)[1] in a bottom-up fashion, by using a neural network layer to combine the hidden states of the node's constituents (Goller and Kuchler, 1996; Socher et al., 2011). This recursive model has the advantage of both encoding linguistic information about supertags and also making efficient use of sparse data: even if a supertag appears infrequently in the corpus, the parser learns the parameters for embedding that supertag from encountering other, structurally similar supertags.

We use a syntactically untied RNN, similar to the model described in Socher et al. (2014). First, for each leaf node j in an elementary tree, the hidden state h_j is obtained by taking the j^{th} column of an embedding matrix $E \in M_{d\times|L|}(\mathbb{R})$, where $|L|$ is the size of the vocabulary of labels used in TAG trees. Then, for each non-terminal node i, let $C(i)$ denote the set of children of node i and let $\text{rel}(i,j)$ denote the relation between node i and its child j, defined by the label of i and the left-to-right position of j. For example, if i is a VP node and j is its leftmost child, then $\text{rel}(i,j)$ is VP_0. The hidden state of node i is then

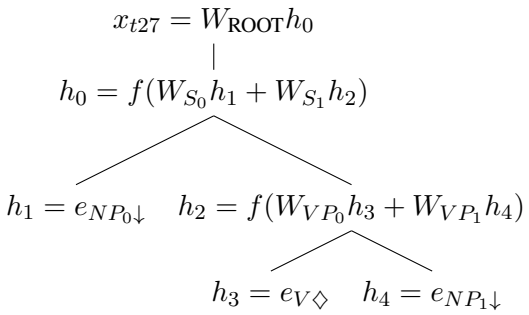
$$h_i = f\left(\sum_{j\in C(i)} W_{\text{rel}(i,j)} h_j\right),$$

where f is a nonlinear activation function. We use tanh as the activation function in all of our experiments. $W_{\text{rel}(i,j)}$ is a square matrix in $M_{d\times d}(\mathbb{R})$, so the resulting h_i is a vector in $\mathbb{R}^d$.

[1] Note that throughout this paper we use the abbreviation RNN to refer to recursive neural networks, not *recurrent* neural networks.



(a) The elementary tree for t27, a transitive verb. Nodes are indexed by their position in the breadth-first traversal of the tree.



(b) The recursive activation states for t27. e_l is the column in the embedding matrix E corresponding to label l.

Figure 1: Example of an RNN structure for generating a vector for a supertag.

Once the hidden state for each node in the tree has been computed, we obtain the final vector representation of the supertag by multiplying the hidden state of the root node by a special weight matrix $W_{\text{ROOT}} \in M_{d\times d}(\mathbb{R})$ and applying the activation function. Figure 1 gives an example of how the RNN is used to generate a vector representation for a transitive verb.

As with the Atomic Embeddings, the weight matrices W's and E can be learned during training of the parser via backpropogation.

2.4 Feature-based Embeddings

Our third representation of supertags and corresponding embedding derives from the hand-selected features associated with elementary trees in the grammar used in MICA (Bangalore et al., 2009). Each elementary tree is associated with a set of dimensions describing phrase structure, interpretation (e.g., subcategorization frame), and linguistic transformations (e.g., dative shift). The features include binary- and ternary-valued dimensions, whose values can be "yes", "no", or "NA," as well as dimensions whose values are a part of speech tag or a list of part of speech tags. See Chung et al. (2016) for the complete list of features.

To feed these feature vectors to a neural network, and to allow us to compare the vectors directly with our recursive embeddings, we convert each feature vector into a d-dimensional real-valued vector. We use two approaches to encoding features. First, we encode binary- and ternary-valued features as one-hot vectors. Since the other fields take on a much larger range of possible values, we choose to embed those features in a low-dimensional space. Specifically, we randomly initialize an embedding matrix $E \in M_{k\times|L|}(\mathbb{R})$, where $|L|$ is the number of part of speech labels, and associate each part of speech label with a column in the matrix. We set k equal to 8. For fields with list values–for example, the list of adjunction nodes–we pad the list with zeros to ensure a fixed width representation. We concatenate the one-hot vectors and the label embeddings to obtain an n-dimensional feature representation of the supertag. In order to compare these vectors more directly with our d-dimensional recursive embeddings, we then multiply each vector by a weight matrix $W \in M_{n\times d}(\mathbb{R})$ to obtain a final, d-dimensional vector for each supertag. Once again, these embedding matrices are trained in conjunction with training of the parser.

2.5 GloVe Model

As a final point of comparison, we generate vector representations of supertags by training a GloVe model on our training corpus of supertags. The GloVe model is a widely used method for generating dense representations of words from corpus co-occurrence counts (Pennington et al., 2014). A GloVe model is trained on a co-occurrence table $X \in M_{|V|\times|V|}(\mathbb{R})$, where X_{ij} is the number of times word i appears in the context of word j. Context is determined by a hyperparameter c: an occurrence of word i is in the context of an occurrence of word j if that occurrence of i appears within the window of c words on either side of that occurrence of j. The cost function is

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log W_{ij})^2.$$

f is a weighting function defined $f(x) = \min(1, (x/x_{max})^\alpha)$, where x_{max} and α are hyperparameters to be tuned. w_i and $\tilde{w}_j$ are the word

vector and the context vector for words i and j, respectively. Similarly, b_i and $\tilde{b}_j$ are word and context biases for words i and j. Intuitively, the GloVe model attempts to learn a dense representation of words that will allow it to estimate the likelihood that any pair of words co-occur.

We train a GloVe model to convergence on the training portion of the TAG-annotated PTB training corpus, from which we were able to extract a co-occurrence table for supertags. We then use the resulting vectors to initialize an embedding matrix for the parser.

3 Experimental Setups

The experiments we conducted employ the same experimental setups as Kasai et al. (2017). Specifically, we follow the protocol from Chung et al. (2016) and Bangalore et al. (2009), and use the grammar and the TAG-annotated WSJ Penn Tree Bank extracted by Chen (2001). 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.

We use a publicly available string representation of supertags to associate each supertag with an elementary tree.[2] We consider the label of a node in an elementary tree to consist of the part of speech tag as well as the deep argument position and the node type, if relevant. For example, an NP node marked for substitution with a deep argument position of 0 will be labeled "NP0s" and will be considered distinct from, for example, an NP foot node ("NPf"). The relation between a node i and a child j ($\mathrm{rel}(i, j)$) is then considered to be the label of i subscripted by the index of j in the ordered list of children of i.

We implement the recursive neural network in TensorFlow and TensorFlow Fold, a library for creating TensorFlow models that take dynamically sized, structure inputs, like trees (Looks et al., 2017).[3] For the sake of comparison with the results in Kasai et al. (2017), we use an embedding size of 50 in all of our experiments and set all of the hyperparameters of the parser to be the same as the best performing ones. Specifically, we use two fully-connected layers with 200 hidden units each, dropout rates of 0.2 for the input and 0.3 for the hidden layer, and 3 for the stack and buffer scope. We train stochastically using the Adam optimization algorithm and minibatches of size 100.

We use a publicly available TensorFlow implementation of the GloVe model (available at https://github.com/GradySimon/tensorflow-glove) training for 50 iterations. The hyperparameters are the same as those reported in Pennington et al. (2014). We use a context size of 5, which we found performed better than larger context windows.

4 Methods of Evaluation

4.1 Supertag Similarity and the Analogy Task

In the literature on word embeddings (e.g., Mikolov et al. (2013a) and Pennington et al. (2014)), word analogies are often used to evaluate whether the word embeddings have captured relevant semantic relationships between words. Similarly, we use syntactic analogies to evaluate whether our supertag embeddings have captured relevant syntactic relationships between supertags. To create our test set, we considered 9 different syntactic transformations that can relate two supertags either within or across tree families. These transformations are (i) subject relativization, (ii) object relativization, (iii) subject *wh*-movement, (iv) object *wh*-movement, (v) transitivization, (vi) infinitivization,[4] (vii) passivization with a *by* phrase, (viii) passivization without a *by* phrase, and (ix) dative shift.

For each of these transformations, we identified all pairs of supertags $(tag1, tag2)$ such that applying the transformation to $tag1$ yields $tag2$. For example, if the transformation in question is subject relativization, an example of such a pair would be $t27$ (the supertag for the verb heading a transitive clause, such as *found* in *the boy found the treasure*) and $t99$ (the supertag for the verb heading a transitive subject relative clause, such as *found* in *the boy who found the treasure*). We then generated syntactic analogies by matching up two such pairs that were both generated based on the same transformation; for example, the pairs $(t27, t99)$ and $(t81, t109)$ were both generated based on subject relativization, so combining these pairs gives

[2] The grammar is available here: http://mica.lif.univ-mrs.fr

[3] https://github.com/tensorflow/fold

[4] Infinitivization is the process of turning a verbal tree with a substitution node in subject position into a tree with an empty category in its subject position. It is called infinitivization because typically trees with empty subjects are anchored on infinitival verbs.

the analogy "$t27$ is to $t99$ as $t81$ is to $t109$", since conducting subject relativization on $t27$ generates $t99$ and conducting subject relativization on $t81$ generates $t109$. These trees are shown in Figure 2.

We wish to test if the linear relationships between our supertag embeddings properly express the syntactic relationships between the elementary trees these embeddings represent. To test if this is the case, we represent each of our analogies in the form of an equation; for example, the analogy "$t27$ is to $t99$ as $t81$ is to $t109$" would be written as $t27 - t99 \approx t81 - t109$. We then rearrange the equation so that there is only one term on the right-hand side to get $t27 - t99 + t109 \approx t81$. Each such equation becomes one test in the test set, and to test the equation we perform the arithmetic on the left hand side (here, $t27 - t99 + t109$) using the embeddings we have generated and evaluate how similar the resulting vector is to the desired right hand side (here $t81$).

We use two basic metrics for evaluating performance. The first metric (*% correct* in Table 1) is the proportion of analogies in the test set for which the most similar vector to the result of the computation is the desired one as measured by cosine distance. The second metric (*Avg. position* in Table 1) is the average rank of the correct answer within the ranked list of the embeddings that have the smallest cosine distance from the result of the computation.

In addition, it may be the case that the embeddings we generate are better for more frequent supertags due to a larger number of training examples on which to train the embedding. In order to ascertain the effects of frequency, we also compute both of the aforementioned metrics upon various restricted test sets consisting of only those analogies for which all four supertags in the analogy are among the n most common supertags in the training set for some value n, and where the ranked list of nearest neighbors is filtered to only include the n most common supertags. These metrics are listed as *% correct (top n)* and *Avg. position (top n)* in Table 1.

4.2 Parsing

We also present parsing results for models trained using each embedding scheme. We use the same neural TAG parser in each experiment, varying only the embedding layer. In all cases the parameters for the embedding layer are trained jointly with the parser. We give labeled and unlabeled attachment scores given gold supertags and predicted supertags from the supertagger in Kasai et al. (2017), using first a greedy decoding strategy and then beam search with a beam size of 16.

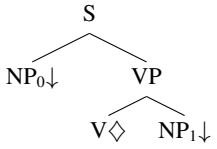
5 Results and Discussion

5.1 Analogy Task

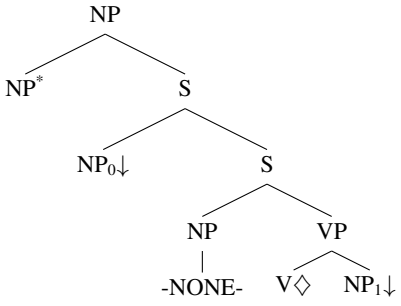
Our results for the analogy task are given in Table 1. The GloVe vectors perform significantly worse than any other embedding, suggesting that co-occurrence information alone in the absence of information about grammatical structure is not enough to learn the kind of syntactic information tested in the analogy task.

The atomic embeddings trained with the parser learn a surprising amount of syntactic information about supertags. Despite initially having no information about the underlying syntactic meaning of supertags, the parser is able to discover syntactic features in the process of optimizing parsing performance on the training set. As we would expect, however, the performance of the atomic embeddings degrades quickly as n increases. The atomic embeddings get around two thirds of the analogy tests correct when the tests and answers are restricted to the top 300 supertags, but the embeddings perform considerably worse with larger values of n; the embeddings get only 4.62% accuracy on analogy tests drawn from the full set of supertags. This is consistent with what we would expect: the atomic embedding scheme treats each supertag as distinct from the others, and the majority of supertags are very sparse in the training data, so the parser has scarce information with which to learn better representations for rare supertags.

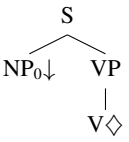
In contrast, both the recursive embeddings and the feature-based embeddings achieve very high accuracy on the analogy tests, with hardly any degradation with larger values of n. Of particular note is average position of the correct supertag in the ranked list of possible answers. For both the recursive and the feature-based embeddings, the target supertag is, on average, within the top 2 supertags most similar to the result of the analogy tests, even using tests drawn from the full set of supertags. This is not surprising given that both embedding schemes encode syntactic information about supertags by construction, so, unlike atomic embeddings, they produce meaningful representations of supertags even when the su-



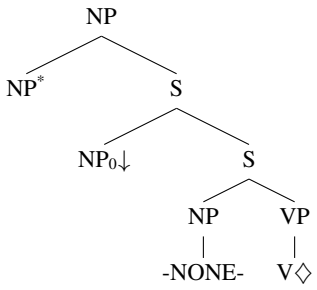
(a) The elementary tree for t27.



(b) The elementary tree for t99.



(c) The elementary tree for t81.



(d) The elementary tree for t109.

Figure 2: The supertags involved in the subject-relativization-based analogy $t27 - t99 + t109 \approx t81$.

Embedding	n	# equations	% correct	% correct (top n)	Avg. position	Avg. position (top n)
GloVe	300	246	0.00	0.00	71.21	101.30
Atomic	300	246	50.40	67.07	7.98	2.36
RNN	300	246	83.33	93.50	1.83	1.08
Features	300	246	**97.56**	**100**	**1.02**	**1.00**
GloVe	500	776	0.13	0.13	128.04	158.16
Atomic	500	776	36.47	41.62	22.15	5.67
RNN	500	776	82.09	91.36	1.68	1.15
Features	500	776	**95.62**	**99.87**	**1.05**	**1.00**
GloVe	4724	57220	0.02	–	2086.35	–
Atomic	4724	57220	4.62	–	289.48	–
RNN	4724	57220	83.34	–	1.61	–
Features	4724	57220	**94.14**	–	**1.10**	–

Table 1: Analogy task results.

pertags appear infrequently in the training data. The feature-based vectors in particular directly encode many of the dimensions we inspect with the analogy tests since several of the transformations used in the analogy tests, such as dative shift and passivization, are among the features derived from Chung et al. (2016) that we used.

The feature-based embeddings outperform the recursive embeddings. In particular, the feature-based embeddings seem to be even more robust with larger values of n. The recursive embeddings achieve above 90% accuracy when the space of possible answers is limited to the n most common supertags, but accuracy decreases by around 10% when we are allowed to consider all possible supertags. Using feature-based embeddings, however, accuracy remains above 94% for all tests, and is 100% for low values of n. One reason for this disparity might be that the recursive neural network, like the atomic embedding matrix, contains parameters that might be updated rarely; the RNN includes a weight matrix W for each parent-child relation, and some relations (e.g., VP_6) appear in only a few rare elementary trees. By contrast, the parameters trained to produce the feature-based embeddings are shared across all elementary trees, so the information learned by the parser generalizes better to rare or even unseen supertags.

Figures 3 and 4 provide a visual representation of certain atomic embeddings by plotting their first two principal components. They show how even the comparatively syntactically uninformed

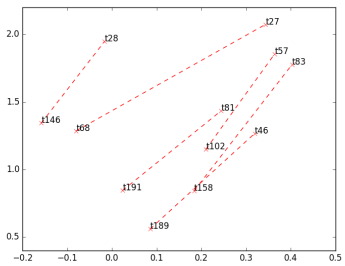


Figure 3: Graph of the first two principal components of the atomic embeddings for supertag pairs related by infinitivization.

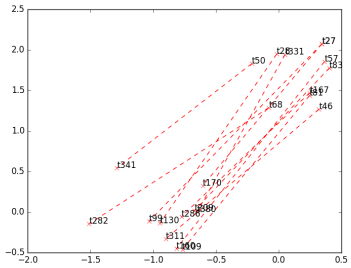


Figure 4: Graph of the first two principal components of the atomic embeddings for supertag pairs related by subject relativization.

method of using atomic embeddings still creates relatively consistent linear structure between pairs of vectors related by the same syntactic transformations.

5.2 Parsing

The parsing results are in Table 2. Although both of our linguistically informed representations of supertags (the RNN embeddings and the feature-based embeddings) significantly outperform the atomic embeddings at the analogy task, these large increases in performance do not carry over to the task of parsing. When parsing with gold supertags, the atomic embeddings outperform both the RNN embeddings and the feature-based embeddings. With predicted supertags, the feature-based embeddings do slightly overtake the atomic embeddings, but the increase in performance is quite small, while the RNN embeddings continue to achieve lower scores than the atomic embeddings.[5]

The fact that the parser achieves state-of-the-art performance even with the atomic embeddings suggests that the parser may be able to induce meaningful linguistic relationships from the data alone. If this is the case, it could explain why the linguistically informed embeddings do not make much of a difference in parsing: If the parser can learn the syntactic relationships between supertags that are most relevant to parsing solely from the data, then it does not require the additional linguistic information that the RNN-based and feature-based embeddings provide.

The fact remains that the linguistically informed embeddings do perform much better at the analogy task than the atomic embeddings. This may be because the analogies contain some syntactic constructions (such as relativization of an indirect object) that are relatively uncommon. Since these constructions are uncommon, it may be possible to attain high parsing performance without being able to properly handle such constructions. Thus, the atomic embeddings are able to yield high parsing performance despite having poorer analogy task performance. It would be interesting to parse only those sentences containing uncommon syntactic constructions to see if the linguistically informed embeddings outperform the atomic embeddings in such cases, since we would expect that the greatest improvements in performance from linguistically informed embeddings would come with the parsing of infrequent supertags, since the parser would not have much training data with which to learn representations of these supertags.

6 Conclusions and Future Work

We presented two techniques for computing real-valued vector representations of TAG supertags and applied these vector representations to the shift-reduce parsing system. We showed that the vectors produced by all but the non-syntactic GloVe embeddings performed reasonably well on the syntactic analogy task. The two representations which were built on the basis of explicitly represented linguistic structure, namely recursive tree-based embedding and feature-based embedding, on average produced the correct answer to an analogy question as one of its top two guesses, thereby outperforming both the distribution-based embeddings trained using the GloVe method and the atomic embeddings. On the parsing task, our two linguistically-rich embedding methods per-

[5] To see if the two types of linguistically informed embeddings are complementary, we also trained a model combining both RNN embeddings and feature-based embeddings, but there was no significant improvement over either of the models with only one form of embedding.

Parsing Model	Beam size	Dev Results				Test Results			
		Gold Stags		Predicted Stags		Gold Stags		Predicted Stags	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
bi-LSTM Stagger + MICA Parser	–	97.60	97.30	90.05	88.32	96.97	96.59	90.20	88.66
GloVe	1	93.01	91.97	88.35	86.46	–	–	–	–
Atomic	1	**96.82**	**96.45**	89.48	88.00	–	–	–	–
RNN	1	95.53	95.21	89.55	88.05	–	–	–	–
Features	1	94.90	94.74	**89.63**	**88.19**	–	–	–	–
GloVe	16	93.91	92.96	89.14	87.32	94.10	93.16	88.83	87.15
Atomic	16	**97.67**	**97.45**	90.23	88.77	**97.87**	**97.64**	90.25	88.90
RNN	16	96.39	96.10	90.30	88.81	96.73	96.46	90.24	88.85
Features	16	95.78	95.62	**90.36**	**88.91**	96.42	96.21	**90.31**	**88.96**

Table 2: Parsing results on the development and test sets. In each cell, shown is the mean of 5 trials with different initialization; the standard deviation for these values ranges from 0.01 to 0.26.

formed comparably with the atomic embedding method; the fact that the linguistically informed embeddings did not significantly improve performance compared to the atomic embeddings is an interesting indication that the parser can learn most relevant syntactic information purely from its training data. In the future, we will explore the application of these linguistically-rich embeddings in different systems of parsing such as the graph-based parsing system that has recently been successful in dependency grammar parsing.

Acknowledgments

We thank the anonymous reviewers for their helpful suggestions on this work.

References

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.

Tilman Becker. 2000. Patterns in metarules for TAG. In *Tree Adjoining Grammars*, page 331342.

Danqi Chen and Christopher 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)*. Association for Computational Linguistics, Doha, Qatar, pages 740–750. http://www.aclweb.org/anthology/D14-1082.

John Chen. 2001. *Towards efficient statistical parsing using lexicalized grammatical information*. Ph.D. thesis, Ph. D. thesis, University of Delaware.

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.

Roger Evans, Gerald Gazdar, and David Weir. 1995. Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. pages 77–84.

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.

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.

Jungo Kasai, Robert Frank, R. Thomas McCoy, Owen Rambow, and Alexis Nasr. 2017. TAG Parsing with Neural Networks and Vector Representations of Supertags. In *Proceedings of EMNLP*.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. Lstm ccg parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 221–231. http://www.aclweb.org/anthology/N16-1026.

Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. 2017. Deep learning with dynamic computation graphs. *Proceedings of the International Conference on Learning Representations* https://openreview.net/pdf?id=ryrGawqex.

Brian McMahan and Matthew Stone. 2016. Syntactic realization with data-driven neural tree grammars. In *Proceedings of COLING 2016, the 26th In-*

ternational Conference on Computational Linguistics: Technical Papers. The COLING 2016 Organizing Committee, Osaka, Japan, pages 224–235. http://aclweb.org/anthology/C16-1022.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013a. 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.

Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2013)*. Atlanta, page 746751.

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)*. pages 1532–1543.

Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. 2014. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics* 2:207–218.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. pages 129–136.

K Vijay-Shanker and Yves Schabes. 1992. Structure sharing in lexicalized tree-adjoining grammars. In *Proceedings of the 14th conference on Computational linguistics*. Association for Computational Linguistics, volume 1, pages 205–211.

XTAG Research Group. 1998. A lexicalized tree adjoining grammar for English. Technical report, Department of Computer and Information Sciences, University of Pennsylvania.

Wenduan Xu. 2016. LSTM shift-reduce CCG parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1754–1764. https://aclweb.org/anthology/D16-1181.