Parsing with Compositional Vector Grammars

Richard Socher **John Bauer** **Christopher D. Manning** **Andrew Y. Ng**
Computer Science Department, Stanford University, Stanford, CA 94305, USA
richard@socher.org, horatio@gmail.com, manning@stanford.edu, ang@cs.stanford.edu

Abstract

Natural language parsing has typically been done with small sets of discrete categories such as NP and VP, but this representation does not capture the full syntactic nor semantic richness of linguistic phrases, and attempts to improve on this by lexicalizing phrases or splitting categories only partly address the problem at the cost of huge feature spaces and sparseness. Instead, we introduce a Compositional Vector Grammar (CVG), which combines PCFGs with a syntactically untied recursive neural network that learns syntactico-semantic, compositional vector representations. The CVG improves the PCFG of the Stanford Parser by 3.8% to obtain an F1 score of 90.4%. It is fast to train and implemented approximately as an efficient reranker it is about 20% faster than the current Stanford factored parser. The CVG learns a soft notion of head words and improves performance on the types of ambiguities that require semantic information such as PP attachments.

1 Introduction

Syntactic parsing is a central task in natural language processing because of its importance in mediating between linguistic expression and meaning. For example, much work has shown the usefulness of syntactic representations for subsequent tasks such as relation extraction, semantic role labeling (Gildea and Palmer, 2002) and paraphrase detection (Callison-Burch, 2008).

Syntactic descriptions standardly use coarse discrete categories such as NP for noun phrases or PP for prepositional phrases. However, recent work has shown that parsing results can be greatly improved by defining more fine-grained syntactic categories, which better capture phrases with similar behavior, whether through manual feature engineering (Klein and Manning, 2003a) or automatic learning (Petrov et al., 2006). However, subdividing a category like NP into 30 or 60 subcategories can only provide a very limited representation of phrase meaning and semantic similarity. Two strands of work therefore attempt to go further. First, recent work in discriminative parsing has shown gains from careful engineering of features (Taskar et al., 2004; Finkel et al., 2008). Features in such parsers can be seen as defining effective dimensions of similarity between categories. Second, lexicalized parsers (Collins, 2003; Charniak, 2000) associate each category with a lexical item. This gives a fine-grained notion of semantic similarity, which is useful for tackling problems like ambiguous attachment decisions. However, this approach necessitates complex shrinkage estimation schemes to deal with the sparsity of observations of the lexicalized categories.

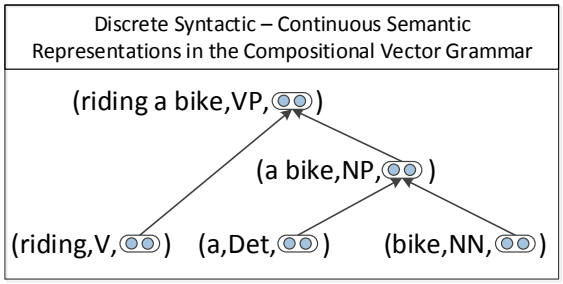

Figure 1: Example of a CVG tree with (category,vector) representations at each node. The vectors for nonterminals are computed via a new type of recursive neural network which is conditioned on syntactic categories from a PCFG.

In many natural language systems, single words and n-grams are usefully described by their distributional similarities (Brown et al., 1992), among many others. But, even with large corpora, many

n-grams will never be seen during training, especially when n is large. In these cases, one cannot simply use distributional similarities to represent unseen phrases. In this work, we present a new solution to learn features and phrase representations even for very long, unseen n-grams.

We introduce a Compositional Vector Grammar Parser (CVG) for structure prediction. Like the above work on parsing, the model addresses the problem of representing phrases and categories. Unlike them, it jointly learns how to parse and how to represent phrases as both discrete categories and continuous vectors as illustrated in Fig. 1. CVGs combine the advantages of standard probabilistic context free grammars (PCFG) with those of recursive neural networks (RNNs). The former can capture the discrete categorization of phrases into NP or PP while the latter can capture fine-grained syntactic and compositional-semantic information on phrases and words. This information can help in cases where syntactic ambiguity can only be resolved with semantic information, such as in the PP attachment of the two sentences: *They ate udon with forks.* vs. *They ate udon with chicken.*

Previous RNN-based parsers used the same (tied) weights at all nodes to compute the vector representing a constituent (Socher et al., 2011b). This requires the composition function to be extremely powerful, since it has to combine phrases with different syntactic head words, and it is hard to optimize since the parameters form a very deep neural network. We generalize the fully tied RNN to one with syntactically untied weights. The weights at each node are conditionally dependent on the categories of the child constituents. This allows different composition functions when combining different types of phrases and is shown to result in a large improvement in parsing accuracy.

Our compositional distributed representation allows a CVG parser to make accurate parsing decisions and capture similarities between phrases and sentences. Any PCFG-based parser can be improved with an RNN. We use a simplified version of the Stanford Parser (Klein and Manning, 2003a) as the base PCFG and improve its accuracy from 86.56 to 90.44% labeled F1 on all sentences of the WSJ section 23. The code of our parser is available at `nlp.stanford.edu`.

2 Related Work

The CVG is inspired by two lines of research: Enriching PCFG parsers through more diverse sets of discrete states and recursive deep learning models that jointly learn classifiers and continuous feature representations for variable-sized inputs.

Improving Discrete Syntactic Representations

As mentioned in the introduction, there are several approaches to improving discrete representations for parsing. Klein and Manning (2003a) use manual feature engineering, while Petrov et al. (2006) use a learning algorithm that splits and merges the syntactic categories in order to maximize likelihood on the treebank. Their approach splits categories into several dozen subcategories. Another approach is lexicalized parsers (Collins, 2003; Charniak, 2000) that describe each category with a lexical item, usually the head word. More recently, Hall and Klein (2012) combine several such annotation schemes in a factored parser. We extend the above ideas from discrete representations to richer continuous ones. The CVG can be seen as factoring discrete and continuous parsing in one model. Another different approach to the above generative models is to learn discriminative parsers using many well designed features (Taskar et al., 2004; Finkel et al., 2008). We also borrow ideas from this line of research in that our parser combines the generative PCFG model with discriminatively learned RNNs.

Deep Learning and Recursive Deep Learning

Early attempts at using neural networks to describe phrases include Elman (1991), who used recurrent neural networks to create representations of sentences from a simple toy grammar and to analyze the linguistic expressiveness of the resulting representations. Words were represented as one-on vectors, which was feasible since the grammar only included a handful of words. Collobert and Weston (2008) showed that neural networks can perform well on sequence labeling language processing tasks while also learning appropriate features. However, their model is lacking in that it cannot represent the recursive structure inherent in natural language. They partially circumvent this problem by using either independent window-based classifiers or a convolutional layer. RNN-specific training was introduced by Goller and Küchler (1996) to learn distributed representations of given, structured objects such as logical terms. In contrast, our model both predicts the structure and its representation.

Henderson (2003) was the first to show that neural networks can be successfully used for large scale parsing. He introduced a left-corner parser to estimate the probabilities of parsing decisions conditioned on the parsing history. The input to Henderson's model consists of pairs of frequent words and their part-of-speech (POS) tags. Both the original parsing system and its probabilistic interpretation (Titov and Henderson, 2007) learn features that represent the *parsing history* and do not provide a principled linguistic representation like our phrase representations. Other related work includes (Henderson, 2004), who discriminatively trains a parser based on synchrony networks and (Titov and Henderson, 2006), who use an SVM to adapt a generative parser to different domains.

Costa et al. (2003) apply recursive neural networks to re-rank possible phrase attachments in an incremental parser. Their work is the first to show that RNNs can capture enough information to make correct parsing decisions, but they only test on a subset of 2000 sentences. Menchetti et al. (2005) use RNNs to re-rank different parses. For their results on full sentence parsing, they re-rank candidate trees created by the Collins parser (Collins, 2003). Similar to their work, we use the idea of letting discrete categories reduce the search space during inference. We compare to fully tied RNNs in which the same weights are used at every node. Our syntactically untied RNNs outperform them by a significant margin. The idea of untying has also been successfully used in deep learning applied to vision (Le et al., 2010).

This paper uses several ideas of (Socher et al., 2011b). The main differences are (i) the dual representation of nodes as discrete categories and vectors, (ii) the combination with a PCFG, and (iii) the syntactic untying of weights based on child categories. We directly compare models with fully tied and untied weights. Another work that represents phrases with a dual discrete-continuous representation is (Kartsaklis et al., 2012).

3 Compositional Vector Grammars

This section introduces Compositional Vector Grammars (CVGs), a model to jointly find syntactic structure and capture compositional semantic information.

CVGs build on two observations. Firstly, that a lot of the structure and regularity in languages can be captured by well-designed syntactic patterns. Hence, the CVG builds on top of a standard PCFG parser. However, many parsing decisions show fine-grained semantic factors at work. Therefore we combine syntactic and semantic information by giving the parser access to rich syntactico-semantic information in the form of distributional word vectors and compute compositional semantic vector representations for longer phrases (Costa et al., 2003; Menchetti et al., 2005; Socher et al., 2011b). The CVG model merges ideas from both generative models that assume discrete syntactic categories and discriminative models that are trained using continuous vectors.

We will first briefly introduce single word vector representations and then describe the CVG objective function, tree scoring and inference.

3.1 Word Vector Representations

In most systems that use a vector representation for words, such vectors are based on co-occurrence statistics of each word and its context (Turney and Pantel, 2010). Another line of research to learn distributional word vectors is based on neural language models (Bengio et al., 2003) which jointly learn an embedding of words into an n-dimensional feature space and use these embeddings to predict how suitable a word is in its context. These vector representations capture interesting linear relationships (up to some accuracy), such as $king - man + woman \approx queen$ (Mikolov et al., 2013).

Collobert and Weston (2008) introduced a new model to compute such an embedding. The idea is to construct a neural network that outputs high scores for windows that occur in a large unlabeled corpus and low scores for windows where one word is replaced by a random word. When such a network is optimized via gradient ascent the derivatives backpropagate into the word embedding matrix X. In order to predict correct scores the vectors in the matrix capture co-occurrence statistics.

For further details and evaluations of these embeddings, see (Turian et al., 2010; Huang et al., 2012). The resulting X matrix is used as follows. Assume we are given a sentence as an ordered list of m words. Each word w has an index $[w] = i$ into the columns of the embedding matrix. This index is used to retrieve the word's vector representation a_w using a simple multiplication with a binary vector e, which is zero everywhere, except

at the ith index. So $a_w = Le_i \in \mathbb{R}^n$. Henceforth, after mapping each word to its vector, we represent a sentence S as an ordered list of (word,vector) pairs: $x = ((w_1, a_{w_1}), \ldots, (w_m, a_{w_m}))$.

Now that we have discrete and continuous representations for all words, we can continue with the approach for computing tree structures and vectors for nonterminal nodes.

3.2 Max-Margin Training Objective for CVGs

The goal of supervised parsing is to learn a function $g : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is the set of sentences and $\mathcal{Y}$ is the set of all possible labeled binary parse trees. The set of all possible trees for a given sentence x_i is defined as $Y(x_i)$ and the correct tree for a sentence is y_i.

We first define a structured margin loss $\Delta(y_i, \hat{y})$ for predicting a tree $\hat{y}$ for a given correct tree. The loss increases the more incorrect the proposed parse tree is (Goodman, 1998). The discrepancy between trees is measured by counting the number of nodes $N(y)$ with an incorrect span (or label) in the proposed tree:

$$\Delta(y_i, \hat{y}) = \sum_{d \in N(\hat{y})} \kappa \mathbf{1}\{d \notin N(y_i)\}. \quad (1)$$

We set $\kappa = 0.1$ in all experiments. For a given set of training instances (x_i, y_i), we search for the function g_θ, parameterized by θ, with the smallest expected loss on a new sentence. It has the following form:

$$g_\theta(x) = \arg\max_{\hat{y} \in Y(x)} s(\texttt{CVG}(\theta, x, \hat{y})), \quad (2)$$

where the tree is found by the Compositional Vector Grammar (CVG) introduced below and then scored via the function s. The higher the score of a tree the more confident the algorithm is that its structure is correct. This max-margin, structure-prediction objective (Taskar et al., 2004; Ratliff et al., 2007; Socher et al., 2011b) trains the CVG so that the highest scoring tree will be the correct tree: $g_\theta(x_i) = y_i$ and its score will be larger up to a margin to other possible trees $\hat{y} \in \mathcal{Y}(x_i)$:

$$s(\texttt{CVG}(\theta, x_i, y_i)) \geq s(\texttt{CVG}(\theta, x_i, \hat{y})) + \Delta(y_i, \hat{y}).$$

This leads to the regularized risk function for m training examples:

$$\begin{aligned} J(\theta) =& \frac{1}{m} \sum_{i=1}^{m} r_i(\theta) + \frac{\lambda}{2} \|\theta\|_2^2, \text{ where} \\ r_i(\theta) =& \max_{\hat{y} \in Y(x_i)} \left(s(\texttt{CVG}(x_i, \hat{y})) + \Delta(y_i, \hat{y}) \right) \\ & - s(\texttt{CVG}(x_i, y_i)) \end{aligned} \quad (3)$$

Intuitively, to minimize this objective, the score of the correct tree y_i is increased and the score of the highest scoring incorrect tree $\hat{y}$ is decreased.

3.3 Scoring Trees with CVGs

For ease of exposition, we first describe how to score an existing fully labeled tree with a standard RNN and then with a CVG. The subsequent section will then describe a bottom-up beam search and its approximation for finding the optimal tree.

Assume, for now, we are given a labeled parse tree as shown in Fig. 2. We define the word representations as (vector, POS) pairs: $((a, A), (b, B), (c, C))$, where the vectors are defined as in Sec. 3.1 and the POS tags come from a PCFG. The standard RNN essentially ignores all POS tags and syntactic categories and each non-terminal node is associated with the same neural network (i.e., the weights across nodes are fully tied). We can represent the binary tree in Fig. 2 in the form of branching triplets $(p \rightarrow c_1 c_2)$. Each such triplet denotes that a parent node p has two children and each c_k can be either a word vector or a non-terminal node in the tree. For the example in Fig. 2, we would get the triples $((p^1 \rightarrow bc), (p^2 \rightarrow ap^1))$. Note that in order to replicate the neural network and compute node representations in a bottom up fashion, the parent must have the same dimensionality as the children: $p \in \mathbb{R}^n$.

Given this tree structure, we can now compute activations for each node from the bottom up. We begin by computing the activation for p^1 using the children's word vectors. We first concatenate the children's representations $b, c \in \mathbb{R}^{n \times 1}$ into a vector $\begin{bmatrix} b \\ c \end{bmatrix} \in \mathbb{R}^{2n \times 1}$. Then the composition function multiplies this vector by the parameter weights of the RNN $W \in \mathbb{R}^{n \times 2n}$ and applies an element-wise nonlinearity function $f = \tanh$ to the output vector. The resulting output $p^{(1)}$ is then given as input to compute $p^{(2)}$.

$$p^{(1)} = f\left(W \begin{bmatrix} b \\ c \end{bmatrix}\right), \; p^{(2)} = f\left(W \begin{bmatrix} a \\ p^1 \end{bmatrix}\right)$$

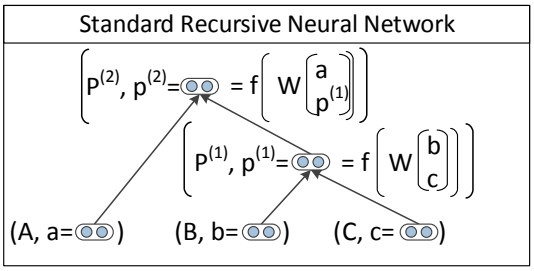

Figure 2: An example tree with a simple Recursive Neural Network: The same weight matrix is replicated and used to compute all non-terminal node representations. Leaf nodes are n-dimensional vector representations of words.

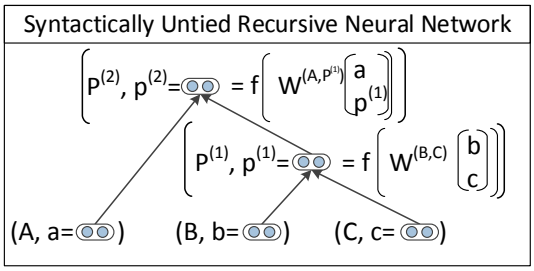

Figure 3: Example of a syntactically untied RNN in which the function to compute a parent vector depends on the syntactic categories of its children which we assume are given for now.

In order to compute a score of how plausible of a syntactic constituent a parent is the RNN uses a single-unit linear layer for all i:

$$s(p^{(i)}) = v^T p^{(i)},$$

where $v \in \mathbb{R}^n$ is a vector of parameters that need to be trained. This score will be used to find the highest scoring tree. For more details on how standard RNNs can be used for parsing, see Socher et al. (2011b).

The standard RNN requires a single composition function to capture all types of compositions: adjectives and nouns, verbs and nouns, adverbs and adjectives, etc. Even though this function is a powerful one, we find a single neural network weight matrix cannot fully capture the richness of compositionality. Several extensions are possible: A two-layered RNN would provide more expressive power, however, it is much harder to train because the resulting neural network becomes very deep and suffers from vanishing gradient problems. Socher et al. (2012) proposed to give every single word a matrix and a vector. The matrix is then applied to the sibling node's vector during the composition. While this results in a powerful composition function that essentially depends on the words being combined, the number of model parameters explodes and the composition functions do not capture the syntactic commonalities between similar POS tags or syntactic categories.

Based on the above considerations, we propose the Compositional Vector Grammar (CVG) that conditions the composition function at each node on discrete syntactic categories extracted from a PCFG. Hence, CVGs combine discrete, syntactic rule probabilities and continuous vector compositions. The idea is that the syntactic categories of the children determine what composition function to use for computing the vector of their parents. While not perfect, a dedicated composition function for each rule RHS can well capture common composition processes such as adjective or adverb modification versus noun or clausal complementation. For instance, it could learn that an NP should be similar to its head noun and little influenced by a determiner, whereas in an adjective modification both words considerably determine the meaning of a phrase. The original RNN is parameterized by a single weight matrix W. In contrast, the CVG uses a syntactically untied RNN (SU-RNN) which has a set of such weights. The size of this set depends on the number of sibling category combinations in the PCFG.

Fig. 3 shows an example SU-RNN that computes parent vectors with syntactically untied weights. The CVG computes the first parent vector via the SU-RNN:

$$p^{(1)} = f\left(W^{(B,C)} \begin{bmatrix} b \\ c \end{bmatrix}\right),$$

where $W^{(B,C)} \in \mathbb{R}^{n \times 2n}$ is now a matrix that depends on the categories of the two children. In this bottom up procedure, the score for each node consists of summing two elements: First, a single linear unit that scores the parent vector and second, the log probability of the PCFG for the rule that combines these two children:

$$s\left(p^{(1)}\right) = \left(v^{(B,C)}\right)^T p^{(1)} + \log P(P_1 \rightarrow B \;\; C), \quad (4)$$

where $P(P_1 \rightarrow B\ C)$ comes from the PCFG. This can be interpreted as the log probability of a discrete-continuous rule application with the following factorization:

$$P((P_1, p_1) \rightarrow (B, b)(C, c)) \quad (5)$$
$$= P(p_1 \rightarrow b\ c | P_1 \rightarrow B\ C) P(P_1 \rightarrow B\ C),$$

Note, however, that due to the continuous nature of the word vectors, the probability of such a CVG rule application is not comparable to probabilities provided by a PCFG since the latter sum to 1 for all children.

Assuming that node p_1 has syntactic category P_1, we compute the second parent vector via:

$$p^{(2)} = f\left(W^{(A,P_1)} \begin{bmatrix} a \\ p^{(1)} \end{bmatrix}\right).$$

The score of the last parent in this trigram is computed via:

$$s\left(p^{(2)}\right) = \left(v^{(A,P_1)}\right)^T p^{(2)} + \log P(P_2 \rightarrow A\ P_1).$$

3.4 Parsing with CVGs

The above scores (Eq. 4) are used in the search for the correct tree for a sentence. The goodness of a tree is measured in terms of its score and the CVG score of a complete tree is the sum of the scores at each node:

$$s(\texttt{CVG}(\theta, x, \hat{y})) = \sum_{d \in N(\hat{y})} s\left(p^d\right). \quad (6)$$

The main objective function in Eq. 3 includes a maximization over all possible trees $\max_{\hat{y} \in Y(x)}$. Finding the global maximum, however, cannot be done efficiently for longer sentences nor can we use dynamic programming. This is due to the fact that the vectors break the independence assumptions of the base PCFG. A (category, vector) node representation is dependent on all the words in its span and hence to find the true global optimum, we would have to compute the scores for all binary trees. For a sentence of length n, there are Catalan(n) many possible binary trees which is very large even for moderately long sentences.

One could use a bottom-up beam search, keeping a k-best list at every cell of the chart, possibly for each syntactic category. This beam search inference procedure is still considerably slower than using only the simplified base PCFG, especially since it has a small state space (see next section for details). Since each probability look-up is cheap but computing SU-RNN scores requires a matrix product, we would like to reduce the number of SU-RNN score computations to only those trees that require semantic information. We note that labeled F1 of the Stanford PCFG parser on the test set is 86.17%. However, if one used an oracle to select the best tree from the top 200 trees that it produces, one could get an F1 of 95.46%.

We use this knowledge to speed up inference via two bottom-up passes through the parsing chart. During the first one, we use only the base PCFG to run CKY dynamic programming through the tree. The $k = 200$-best parses at the top cell of the chart are calculated using the efficient algorithm of (Huang and Chiang, 2005). Then, the second pass is a beam search with the full CVG model (including the more expensive matrix multiplications of the SU-RNN). This beam search only considers phrases that appear in the top 200 parses. This is similar to a re-ranking setup but with one main difference: the SU-RNN rule score computation at each node still only has access to its child vectors, not the whole tree or other global features. This allows the second pass to be very fast. We use this setup in our experiments below.

3.5 Training SU-RNNs

The full CVG model is trained in two stages. First the base PCFG is trained and its top trees are cached and then used for training the SU-RNN conditioned on the PCFG. The SU-RNN is trained using the objective in Eq. 3 and the scores as exemplified by Eq. 6. For each sentence, we use the method described above to efficiently find an approximation for the optimal tree.

To minimize the objective we want to increase the scores of the correct tree's constituents and decrease the score of those in the highest scoring incorrect tree. Derivatives are computed via backpropagation through structure (BTS) (Goller and Küchler, 1996). The derivative of tree i has to be taken with respect to all parameter matrices $W^{(AB)}$ that appear in it. The main difference between backpropagation in standard RNNs and SU-RNNs is that the derivatives at each node only add to the overall derivative of the specific matrix at that node. For more details on backpropagation through RNNs, see Socher et al. (2010)

3.6 Subgradient Methods and AdaGrad

The objective function is not differentiable due to the hinge loss. Therefore, we generalize gradient ascent via the subgradient method (Ratliff et al., 2007) which computes a gradient-like direction. Let $\theta = (X, W^{(\cdot\cdot)}, v^{(\cdot\cdot)}) \in \mathbb{R}^M$ be a vector of all M model parameters, where we denote $W^{(\cdot\cdot)}$ as the set of matrices that appear in the training set. The subgradient of Eq. 3 becomes:

$$\frac{\partial J}{\partial \theta} = \sum_i \frac{\partial s(x_i, \hat{y}_{\max})}{\partial \theta} - \frac{\partial s(x_i, y_i)}{\partial \theta} + \theta,$$

where $\hat{y}_{\max}$ is the tree with the highest score. To minimize the objective, we use the diagonal variant of AdaGrad (Duchi et al., 2011) with minibatches. For our parameter updates, we first define $g_\tau \in \mathbb{R}^{M\times 1}$ to be the subgradient at time step τ and $G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T$. The parameter update at time step t then becomes:

$$\theta_t = \theta_{t-1} - \alpha \left(\text{diag}(G_t)\right)^{-1/2} g_t, \qquad (7)$$

where α is the learning rate. Since we use the diagonal of G_t, we only have to store M values and the update becomes fast to compute: At time step t, the update for the i'th parameter $\theta_{t,i}$ is:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}. \qquad (8)$$

Hence, the learning rate is adapting differently for each parameter and rare parameters get larger updates than frequently occurring parameters. This is helpful in our setting since some W matrices appear in only a few training trees. This procedure found much better optima (by ≈3% labeled F1 on the dev set), and converged more quickly than L-BFGS which we used previously in RNN training (Socher et al., 2011a). Training time is roughly 4 hours on a single machine.

3.7 Initialization of Weight Matrices

In the absence of any knowledge on how to combine two categories, our prior for combining two vectors is to average them instead of performing a completely random projection. Hence, we initialize the binary W matrices with:

$$W^{(\cdot\cdot)} = 0.5[I_{n\times n} I_{n\times n} 0_{n\times 1}] + \epsilon,$$

where we include the bias in the last column and the random variable is uniformly distributed: $\epsilon \sim \mathcal{U}[-0.001, 0.001]$. The first block is multiplied by the left child and the second by the right child:

$$W^{(AB)} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = \left[W^{(A)} W^{(B)} bias\right] \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = W^{(A)} a + W^{(B)} b + bias.$$

4 Experiments

We evaluate the CVG in two ways: First, by a standard parsing evaluation on Penn Treebank WSJ and then by analyzing the model errors in detail.

4.1 Cross-validating Hyperparameters

We used the first 20 files of WSJ section 22 to cross-validate several model and optimization choices. The base PCFG uses simplified categories of the Stanford PCFG Parser (Klein and Manning, 2003a). We decreased the state splitting of the PCFG grammar (which helps both by making it less sparse and by reducing the number of parameters in the SU-RNN) by adding the following options to training: '-noRightRec -dominatesV 0 -baseNP 0'. This reduces the number of states from 15,276 to 12,061 states and 602 POS tags. These include split categories, such as parent annotation categories like VP^S. Furthermore, we ignore all category splits for the SU-RNN weights, resulting in 66 unary and 882 binary child pairs. Hence, the SU-RNN has 66+882 transformation matrices and scoring vectors. Note that any PCFG, including latent annotation PCFGs (Matsuzaki et al., 2005) could be used. However, since the vectors will capture lexical and semantic information, even simple base PCFGs can be substantially improved. Since the computational complexity of PCFGs depends on the number of states, a base PCFG with fewer states is much faster.

Testing on the full WSJ section 22 dev set (1700 sentences) takes roughly 470 seconds with the simple base PCFG, 1320 seconds with our new CVG and 1600 seconds with the currently published Stanford factored parser. Hence, increased performance comes also with a speed improvement of approximately 20%.

We fix the same regularization of $\lambda = 10^{-4}$ for all parameters. The minibatch size was set to 20. We also cross-validated on AdaGrad's learning rate which was eventually set to $\alpha = 0.1$ and word vector size. The 25-dimensional vectors provided by Turian et al. (2010) provided the best

Parser	dev (all)	test$\leq$ 40	test (all)
Stanford PCFG	85.8	86.2	85.5
Stanford Factored	87.4	87.2	86.6
Factored PCFGs	89.7	90.1	89.4
Collins			87.7
SSN (Henderson)			89.4
Berkeley Parser			90.1
CVG (RNN)	85.7	85.1	85.0
CVG (SU-RNN)	91.2	91.1	90.4
Charniak-SelfTrain			91.0
Charniak-RS			92.1

Table 1: Comparison of parsers with richer state representations on the WSJ. The last line is the self-trained re-ranked Charniak parser.

Error Type	Stanford	CVG	Berkeley	Char-RS
PP Attach	1.02	0.79	0.82	0.60
Clause Attach	0.64	0.43	0.50	0.38
Diff Label	0.40	0.29	0.29	0.31
Mod Attach	0.37	0.27	0.27	0.25
NP Attach	0.44	0.31	0.27	0.25
Co-ord	0.39	0.32	0.38	0.23
1-Word Span	0.48	0.31	0.28	0.20
Unary	0.35	0.22	0.24	0.14
NP Int	0.28	0.19	0.18	0.14
Other	0.62	0.41	0.41	0.50

Table 2: Detailed comparison of different parsers.

performance and were faster than 50-,100- or 200-dimensional ones. We hypothesize that the larger word vector sizes, while capturing more semantic knowledge, result in too many SU-RNN matrix parameters to train and hence perform worse.

4.2 Results on WSJ

The dev set accuracy of the best model is 90.93% labeled F1 on all sentences. This model resulted in 90.44% on the final test set (WSJ section 23). Table 1 compares our results to the two Stanford parser variants (the unlexicalized PCFG (Klein and Manning, 2003a) and the factored parser (Klein and Manning, 2003b)) and other parsers that use richer state representations: the Berkeley parser (Petrov and Klein, 2007), Collins parser (Collins, 1997), SSN: a statistical neural network parser (Henderson, 2004), Factored PCFGs (Hall and Klein, 2012), Charniak-SelfTrain: the self-training approach of McClosky et al. (2006), which bootstraps and parses additional large corpora multiple times, Charniak-RS: the state of the art self-trained and discriminatively re-ranked Charniak-Johnson parser combining (Charniak, 2000; McClosky et al., 2006; Charniak and Johnson, 2005). See Kummerfeld et al. (2012) for more comparisons. We compare also to a standard RNN 'CVG (RNN)' and to the proposed CVG with SU-RNNs.

4.3 Model Analysis

Analysis of Error Types. Table 2 shows a detailed comparison of different errors. We use the code provided by Kummerfeld et al. (2012) and compare to the previous version of the Stanford factored parser as well as to the Berkeley and Charniak-reranked-self-trained parsers (defined above). See Kummerfeld et al. (2012) for details and comparisons to other parsers. One of the largest sources of improved performance over the original Stanford factored parser is in the correct placement of PP phrases. When measuring only the F1 of parse nodes that include at least one PP child, the CVG improves the Stanford parser by 6.2% to an F1 of 77.54%. This is a 0.23 reduction in the average number of bracket errors per sentence. The 'Other' category includes VP, PRN and other attachments, appositives and internal structures of modifiers and QPs.

Analysis of Composition Matrices. An analysis of the norms of the binary matrices reveals that the model learns a soft vectorized notion of head words: Head words are given larger weights and importance when computing the parent vector: For the matrices combining siblings with categories VP:PP, VP:NP and VP:PRT, the weights in the part of the matrix which is multiplied with the VP child vector dominates. Similarly NPs dominate DTs. Fig. 5 shows example matrices. The two strong diagonals are due to the initialization described in Sec. 3.7.

Semantic Transfer for PP Attachments. In this small model analysis, we use two pairs of sentences that the original Stanford parser and the CVG did not parse correctly after training on the WSJ. We then continue to train both parsers on two similar sentences and then analyze if the parsers correctly transferred the knowledge. The training sentences are *He eats spaghetti with a fork.* and *She eats spaghetti with pork.* The very similar test sentences are *He eats spaghetti with a spoon.* and *He eats spaghetti with meat.* Initially, both parsers incorrectly attach the PP to the verb in both test sentences. After training, the CVG parses both correctly, while the factored Stanford parser incorrectly attaches both PPs to *spaghetti*. The CVG's ability to transfer the correct PP attachments is due to the semantic word vector similarity between the words in the sentences. Fig. 4 shows the outputs of the two parsers.

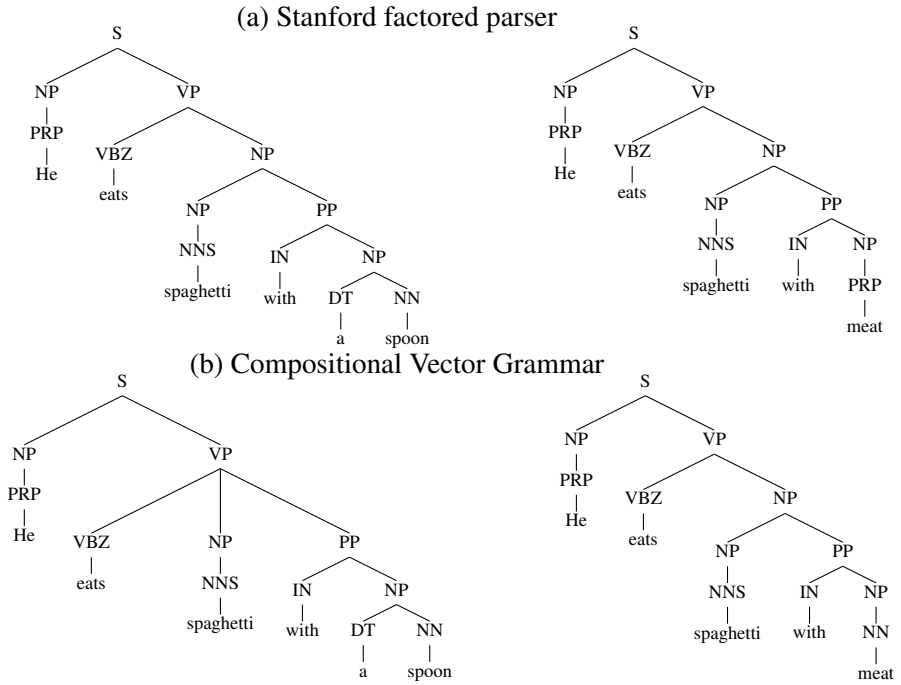

Figure 4: Test sentences of semantic transfer for PP attachments. The CVG was able to transfer semantic word knowledge from two related training sentences. In contrast, the Stanford parser could not distinguish the PP attachments based on the word semantics.

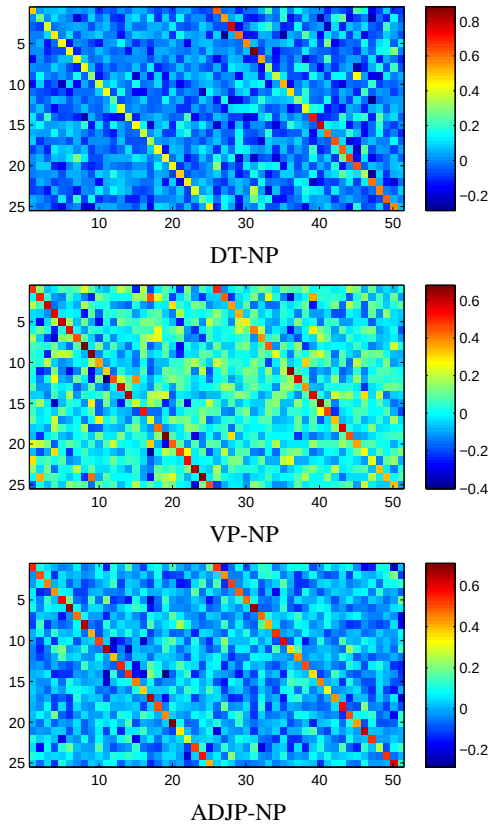

Figure 5: Three binary composition matrices showing that head words dominate the composition. The model learns to not give determiners much importance. The two diagonals show clearly the two blocks that are multiplied with the left and right children, respectively.

5 Conclusion

We introduced Compositional Vector Grammars (CVGs), a parsing model that combines the speed of small-state PCFGs with the semantic richness of neural word representations and compositional phrase vectors. The compositional vectors are learned with a new syntactically untied recursive neural network. This model is linguistically more plausible since it chooses different composition functions for a parent node based on the syntactic categories of its children. The CVG obtains 90.44% labeled F1 on the full WSJ test set and is 20% faster than the previous Stanford parser.

Acknowledgments

We thank Percy Liang for chats about the paper. Richard is supported by a Microsoft Research PhD fellowship. The authors gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Filtering of Text (DEFT) Program under Air Force Research Laboratory (AFRL) prime contract no. FA8750-13-2-0040, and the DARPA Deep Learning program under contract number FA8650-10-C-7020. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA, AFRL, or the US government.

References

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18.

C. Callison-Burch. 2008. Syntactic constraints on paraphrases extracted from parallel corpora. In *Proceedings of EMNLP*, pages 196–205.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of ACL*, pages 132–139.

M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL*.

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

R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of ICML*, pages 160–167.

F. Costa, P. Frasconi, V. Lombardo, and G. Soda. 2003. Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence*.

J. Duchi, E. Hazan, and Y. Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12, July.

J. L. Elman. 1991. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2-3):195–225.

J. R. Finkel, A. Kleeman, and C. D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL*, pages 959–967.

D. Gildea and M. Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proceedings of ACL*, pages 239–246.

C. Goller and A. Küchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of the International Conference on Neural Networks*.

J. Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, MIT.

D. Hall and D. Klein. 2012. Training factored pcfgs with expectation propagation. In *EMNLP*.

J. Henderson. 2003. Neural network probability estimation for broad coverage parsing. In *Proceedings of EACL*.

J. Henderson. 2004. Discriminative training of a neural network statistical parser. In *ACL*.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT 2005)*.

E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng. 2012. Improving Word Representations via Global Context and Multiple Word Prototypes. In *ACL*.

D. Kartsaklis, M. Sadrzadeh, and S. Pulman. 2012. A unified sentence space for categorical distributional-compositional semantics: Theory and experiments. *Proceedings of 24th International Conference on Computational Linguistics (COLING): Posters*.

D. Klein and C. D. Manning. 2003a. Accurate unlexicalized parsing. In *Proceedings of ACL*, pages 423–430.

D. Klein and C.D. Manning. 2003b. Fast exact inference with a factored model for natural language parsing. In *NIPS*.

J. K. Kummerfeld, D. Hall, J. R. Curran, and D. Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *EMNLP*.

Q. V. Le, J. Ngiam, Z. Chen, D. Chia, P. W. Koh, and A. Y. Ng. 2010. Tiled convolutional neural networks. In *NIPS*.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic cfg with latent annotations. In *ACL*.

D. McClosky, E. Charniak, and M. Johnson. 2006. Effective self-training for parsing. In *NAACL*.

S. Menchetti, F. Costa, P. Frasconi, and M. Pontil. 2005. Wide coverage natural language processing using kernel methods and neural networks for structured data. *Pattern Recognition Letters*, 26(12):1896–1906.

T. Mikolov, W. Yih, and G. Zweig. 2013. Linguistic regularities in continuous spaceword representations. In *HLT-NAACL*.

S. Petrov and D. Klein. 2007. Improved inference for unlexicalized parsing. In *NAACL*.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of ACL*, pages 433–440.

N. Ratliff, J. A. Bagnell, and M. Zinkevich. 2007. (Online) subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*.

R. Socher, C. D. Manning, and A. 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*.

R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning. 2011a. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *NIPS*. MIT Press.

R. Socher, C. Lin, A. Y. Ng, and C.D. Manning. 2011b. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *ICML*.

R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. 2012. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *EMNLP*.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proceedings of EMNLP*, pages 1–8.

I. Titov and J. Henderson. 2006. Porting statistical parsers with data-defined kernels. In *CoNLL-X*.

I. Titov and J. Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *ACL*.

J. Turian, L. Ratinov, and Y. Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394.

P. D. Turney and P. Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37:141–188.