6.864 Project Report: Semi-supervised Compositional Vector Parsing with a Skip-gram Model

David Gaddy

December 14, 2015

1 Abstract

The goal of this project is to explore a method for training a compositional vector parser in a semi-supervised setting. Our method includes an additional objective to guide and regularize the model by encouraging phrase vectors at each level of the compositional model to be able to predict their context. Each constituent vector is used in a Skip-gram model to predict the words surrounding the phrase. This objective is optimized over a large set of trees labeled by a PCFG parser - the same parser which is used by the compositional vector parser to generate candidates for reranking. While we hope this method will be able to eventually boost parsing performance, at the time of this writing, our implementation is unable to increase accuracy. In this paper, we describe the model we tried and analyze what is being learned.

2 Motivation

Compositional vector grammars are models that operate over a parse tree generating a dense vector at each constituent by combining and condensing the representation for its children. Socher et al. [2] introduced a compositional vector grammar for use in parsing that uses a neural network at each level to compose the representation from the children. The composition operation must be learned so that as words are combined together in larger and larger phrases, the most important information about a phrase is propagated up into the vector representation so the parser can make an accurate decision.

The goal of this work is to try to improve the quality of the composition by utilizing information outside the annotated training corpus. To do this, we use a Skip-gram model [1], a model typically used for learning representations of words. We hypothesize that the primary objective for the Skip-gram - determining a vector representation based on context - could also be useful with phrases. Just as a word's context determines the content of it's vector, the context surrounding a phrase could give clues as to what information should be propagated into the phrase representation.

We combine the Skip-gram model for phrases with the parser objective so that each gradient update to the neural network is an average of these two objectives. The Skip-gram objective is trained on sentences in an unlabeled corpus with the compositional tree structure determined by a PCFG parser. This provides additional information about what might make a good composition operation and can be viewed as a type of regularization for the model. Socher's paper [2] claims larger vectors didn't work as well for their model hypothesizing 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." By providing an additional source of information, we might hope to train the larger number of parameters and benefit from the advantage larger vectors give to neural models.

3 Model and Training

The composition vector parser we used is based on of Socher's syntactically untied compositional model [2]. We used the version of the parser included in the current version of the Stanford Core NLP package [1]. In this

[1] https://github.com/stanfordnlp/CoreNLP

model, a vector for a phrase (with a binary production rule) is calculated from the vectors for its children b and c as

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

where $W^{(B,C)}$ is a matrix that depends on the labels of the two children. Each vector is then scored by

$$s(p) = \left(v^{(B,C)}\right)^T p$$

, where the vector $v^{(B,C)}$ also depends on the children. Each word has a learned vector that is used at leaves of trees where that word occurs. One aspect of the parser that appears to be different from the original paper is that the original paper added a score term for each production rule $\log P(P_1 \rightarrow B\ C)$, while the current parser does not.

The parser is trained with a max-margin structured prediction objective that encourages the correct tree score to be larger than all other trees by a margin based on the number of incorrect constituents.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} r_i(\theta) + \frac{\lambda}{2}||\theta||_2^2$$

$$r_i(\theta) = \max_{\hat{y} \in Y(x_i)} (s(x_i, \hat{y}) + \Delta(y_i, \hat{y})) - s(x_i, y_i)$$

The margin $\Delta(y_i, \hat{y})$ is based on counting nodes $N(y)$ with an incorrect span or label.

$$\Delta(y_i, \hat{y}) = \sum_{d \in N(\hat{y})} (0.1)\mathbf{1}(d \notin N(y_i))$$

Since dynamic programming cannot work with the compositional vector method, this scoring is instead used to rerank 200 candidates from a PCFG parser.

We are adding another objective to the model that uses a Skip-gram model with negative sampling [1] where each constituent vector is predicting the words immediately adjacent to it. We are trying to maximize the objective

$$\log \sigma(x_{wO}^T v) + \sum_{i=1}^{k} E_{w_i \sim P(w)} \left[\log \sigma(-x_{w_i}^T v)\right]$$

for each phrase with vector v and for each of its context words (defined below) with vector x_{wi}. The second term is an expectation over samples from the word unigram distribution raised to the 3/4 power. Like Mikolov et al., we sample a window size within a particular range and use all words within the window as context. We use the words immediately adjacent to the left and right of a phrase for this context window. We optimize the Skip-gram objective over a separate training corpus which does not have gold trees available. To do this, we label trees using the same PCFG used for creating reranking candidates. We select a parse tree uniformly at random from the top 200 parses of the PCFG and optimize the Skip-gram over this tree.

We use batch Adagrad to optimize these two objectives. We get the score gradient for a batch of labeled sentences and the Skip-gram gradient for a set of PCFG trees in a text corpus. We combine the tree score gradient g_t and the Skip-gram gradient g_s with the following gradient step, putting weight w on the Skip-gram gradient:

$$\theta_i = \theta_{i-1} - \alpha \left(\frac{1}{\sqrt{\sum_{j=1}^{i} g_{t,j}}} g_{t,i} + \frac{w}{\sqrt{\sum_{j=1}^{i} g_{s,j}}} g_{s,i}\right)$$

.

4 Experimental Setup and Results

We used the WSJ Penn Treebank for parsing evaluation with the standard train and test split (2-21 for training, evaluate on 23). We used section 22 as a development set. For the Skip-gram objective, we used

	Development	Test
Original	90.05	89.19
With Skip-gram weight=1	88.80	88.15
With Skip-gram weight=.1	90.23	89.09

Table 1: Results on Penn Treebank parsing task.

	Development	Test
Original	85.04	84.69
Skip-gram weight 0.1	85.55	84.68
Skip-gram weight 0.25	85.63	84.83
Skip-gram weight 0.5	85.48	84.06
Skip-gram weight 0.75	84.98	84.13
Skip-gram weight 1.0	85.28	84.01

Table 2: Results on Penn Treebank parsing task with reduced training data.

the sentences of the New York Times section of the Gigaword 5 corpus. Word vectors were pre-trained on the entire NYT Gigaword using the Google word2vec tool and the tool was modified to also output the context vectors which we used as initialization. The training was run for 20 iterations and periodically checked on development data to select the best model over the training. The batch size was 25 and the supervised learning rate α was set to 0.1 following the original implementation.

We tried various values for the Skip-gram weight w and evaluated on the development set. Results for two values of w are shown in table 1.

Aside: You may note here that the performance of the original code is lower than the numbers reported in Socher et al. We are using the code provided by the group who wrote the paper for this baseline, but several possible reasons for the difference are the difference in model scoring from the original paper (see Model section), different word vectors, or a different random initialization.

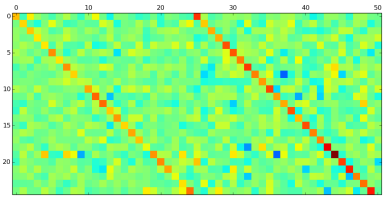
We also attempted a different setup with a reduced training set (WSJ section 2; 1989 trees). To increase the amount of unlabeled data used for the Skip-gram without overwhelming the supervised objective, we first trained the compositional model on only the Skip-gram objective then used this as an initialization for training with the combined objective. In the low data setting, we would expect the semi-supervised objective to help more, but it did not improve performance. Results are shown in table 2. Although one run did show a very slight improvement over the baseline, it is not clear that this is not just a random fluctuation. Overall, there is no score increase from the Skip-gram objective.

All code and detailed setup instructions will be made available at https://github.mit.edu/dgaddy/6.864-Project.

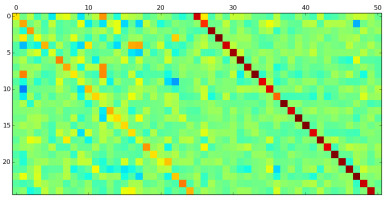
5 Analysis

After observing no increase in parsing performance, we began analyzing the model to see what the Skip-gram objective learns.

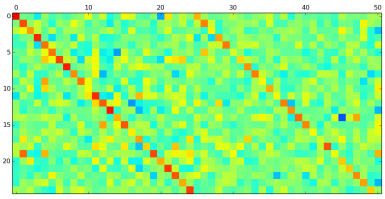
Figure 1 compares a visualization of the learned composition matrices for the fully supervised parsing model with a model trained with only the Skip-gram objective. The Skip-gram does have a few features that indicates it is going in the right direction, such as learning something like the notion of headedness where weights from one of the two children dominates. There are some slight differences, though. The first difference is along the diagonals, which represent the weights propagating information from an element in a child vector to the same dimension in the parent. The diagonals in the matrices learned with Skip-gram are more uniform than those from the original model. The other difference is in the off-diagonal elements, which generally have higher weights in the original than in the Skip-gram. These differences make the Skip-gram composition behave more like a weighted averaging of the children vectors compared to the original which does more complex transformations.



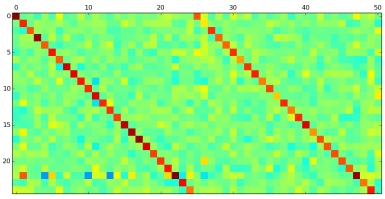
(a) DT-NP



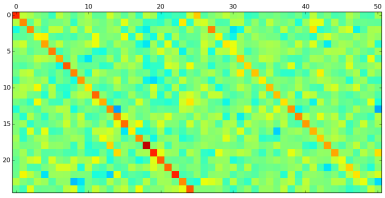
(d) DT-NP



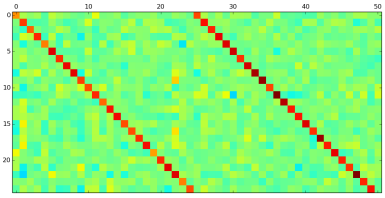
(b) VP-NP



(e) VP-NP



(c) ADJP-NP



(f) ADJP-NP

Figure 1: The composition matrices learned from the parsing objective on the left and the matrices from the Skip-gram are on the right. Red indicates the highest weight, followed by orange, yellow, and light blue. Darker blues are negative weights.

It is not yet clear what the cause of the differences is or why one is better than the other. One hypothesis that accounts for some differences seen is that using syntactically untied scoring vectors in the original, where a different scoring vector is used for different phrase types, allows it flexibility to change the vector spaces for different phrase types. The Skip-gram uses the same context vectors for all phrases, constraining all vectors to be in the same space. This hypothesis could possibly explain the differences seen since using different vector spaces at each level would use more off-diagonal elements. One possible future test would be to use different context vectors depending on the phrase types to allow for more flexibility.

Another possible improvement would be to increase the amount of training data used for the Skip-gram objective. The matrices displayed in this analysis were trained with 13 iterations through 500,000 sentences. In future experiments we will try larger numbers of sentences for the Skip-gram. To make larger training data sizes more feasible, we have implemented a version of the model with sparse matrices, however other training overheads will need to be reduced to fully optimize training speed.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space”

[2] Richard Socher, John Bauer, Christopher Manning, and Andrew Ng. “Parsing with Compositional Vector Grammars”