Inducing Grammar from Long Short-Term Memory Networks by Shapley Decomposition

Allen Nie[*], **Yuhui Zhang**[*]
{anie, yuhuiz}@stanford.edu

Abstract

The principle of compositionality has deep roots in linguistics: the meaning of an expression is determined by its structure and the meanings of its constituents. However, modern neural network models such as long short-term memory networks process expressions in a linear fashion and do not seem to incorporate more complex compositional patterns. In this work, we show that we can explicitly induce grammar by tracing the computational process of a long short-term memory network. We show: (i) the multiplicative nature of long short-term memory network allows complex interaction beyond sequential linear combination; (ii) we can generate compositional trees from the network with minimal assumptions; (iii) we evaluate the syntactic difference between the generated trees, randomly generated trees and gold reference trees produced by constituency parsers; (iv) we evaluate whether the generated trees contain the rich semantic information.

1 Introduction

Recurrent neural network has demonstrated surprising performance on processing natural language data, surpassing traditional n-gram or hand-engineered features on a variety of tasks. Naturally, curiosity about whether these models capture aspects of linguistic knowledge increases. Recent works proposed different probing tests on whether a model learns a set of linguistic properties (Conneau et al., 2018) such as subject-verb agreement (Linzen et al., 2016), syntax-sensitive dependencies (Kuncoro et al., 2018), whether a neuron learns to recognize a group of words with special properties (such as date) (Dalvi et al., 2019), or by dropping the word in the context far away vs nearby and trace perplexity to see how neural networks leverage context (Khandelwal et al., 2018). However, there are two major flaws of the probing tests: i) probing test is limited in the scope of their claim; ii) probing test often treats the model as a blackbox, reaching conclusions by directly altering the testing stimuli and observing the change in the outcome. This type of research often does not yield satisfactory conclusion about the underlying complex mechanism of the blackbox model (Jonas and Kording, 2017).

More holistic approach has been explored to study whether modern neural networks understand sentences by implicitly inducing recursive structures that match the semantics and syntactic theories in linguistics (Williams et al., 2018). However, Williams et al. (2018) studied a specific type of models that explicitly build tree representations of each sentence, which are far from common text processing models such as long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997). In the end, the question of whether common text processing model assumes implicit linguistic structures is left unanswered.

In this work, we draw inspirations from the field of deep learning model interpretations to provide a glimpse into how LSTM networks process a sentence, and extract a tree structure that LSTM networks implicitly create. Using the techniques from contextual decomposition (Murdoch et al., 2018), we propose a tree building algorithm that mimics construction grammar in that the grammar we induce is conditionally dependent on the task and the sentence. We extend Williams et al. (2018)'s analysis on the trees generated from the LSTM networks. We evaluate whether the induced tree structures syntactically resemble constituency grammar, and we evaluate whether training a recursive neural network on the induced structure will provide performance gain over recursive neural network on the constituency grammar.

[*] indicate equal contributions.

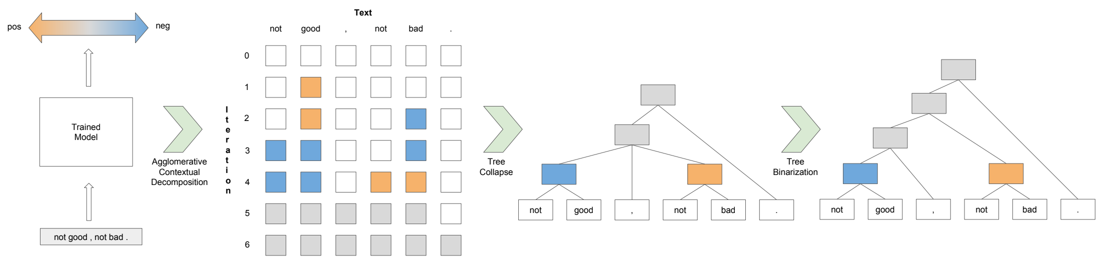

Figure 1: The architecture of the tree generation. We pretrain our model on sentiment classification (SST-2) dataset. The model outputs an overall score indicating the sentiment for the given span of text. We use Agglomerative Contextual Decomposition algorithm (ACD) for hierarchical sentiment interpretation. For each iteration, ACD selects one most salient word with the highest absolute contextual score from unselected words (shown in white), and update scores of all remaining unselected words. Blocks with sentiment scores (shown in blue for negative or orange for positive) are formed during iterations. We build the tree with sentiments based on these blocks and binarize the tree for further evaluation and analysis.

2 Method

2.1 Long Short-Term Memory Networks

Long Short-Term Memory Network is a recurrent neural network composed of a cell, an input gate, an output gate and a forget gate (Hochreiter and Schmidhuber, 1997). The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. This type of network processes input from left to right, with the same cell weights.

$$\begin{aligned}
o_t &= \sigma(W_o x_t + V_o h_{t-1} + b_o) \\
f_t &= \sigma(W_f x_t + V_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + V_i h_{t-1} + b_i) \\
g_t &= \tanh(W_g x_t + V_g h_{t-1} + b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{1}$$

2.2 Shapley Value

Given a function f and variables $F = \{z_1, ..., z_n\}$, and a subset $S \subseteq F \backslash \{z_i\}$, we can define the Shapley value ϕ_i of a given variable z_i as:

$$\phi_i(f) = \sum_{S \subseteq F \backslash \{z_i\}} \frac{1}{Z}(f(S \cup z_i) - f(S)) \tag{2}$$

Intuitively, Shapley value computes the contribution of a term for the final outcome by executing the function with the term z_i and without the term z_i in all possible permutations (enumerating over the presence and absence of all other variables), and then take the average over the number of such permutations Z. Shapley value has been shown as the unifying framework that subsumes many other deep learning interpretation methods (Lundberg and Lee, 2017).

Shapley value has some desirable properties. For example, Shapley values are locally accurate, which means $f(z_1, ..., z_n) = \sum_{i=1}^{n} \phi_i(f)$. We obtain an additive linear combination of Shapley values ϕ_i that will produce the same output as the original model f. Murdoch et al. (2018) proposed to use Shapley decomposition to linearize the non-linear activation functions in the LSTM networks.

Let $f(a, b) = \tanh(a + b)$, we can linearize $\tanh$ activation by calculating the Shapley values of variable a and b (Eq 3).

$$\begin{aligned}
\phi_a(f) &= \frac{1}{2}(\tanh(a) + (\tanh(a+b) - \tanh(b))) \\
\phi_b(f) &= \frac{1}{2}(\tanh(b) + (\tanh(b+a) - \tanh(a)))
\end{aligned} \tag{3}$$

Analogously, we can linearize σ activation as well. We use $L_{\tanh}$ and L_σ to denote this linearization process, and let $L_{\tanh}(a) = \phi_a(\tanh(a + b + ...))$ and $L_\sigma(a) = \phi_a(\sigma(a + b + ...))$. It is worth noting that since Shapley value is computed numerically, the value will change with respect to input. Also, by decomposing LSTM into a summation of Shapley values, we still retain the original output value.

2.3 The Linearly Decomposed LSTM

Murdoch et al. (2018) proposed a method to linearize the LSTM computation by computing the Shaley value of each term. We can use this linearized LSTM to understand how LSTM processes through all time steps, and why it is very powerful in terms of representing a sequence of input. By linearizing the activation functions, we can rewrite the LSTM computation in Eq 4.

$$
\begin{aligned}
o_t &= L_\sigma(W_o x_t) + L_\sigma(V_o h_{t-1}) + L_\sigma(b_o) \\
f_t &= L_\sigma(W_f x_t) + L_\sigma(V_f h_{t-1}) + L_\sigma(b_f) \\
i_t &= L_\sigma(W_i x_t) + L_\sigma(V_i h_{t-1}) + L_\sigma(b_i) \\
g_t &= L_{\tanh}(W_g x_t) + L_{\tanh}(V_g h_{t-1}) + L_{\tanh}(b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
\tag{4}
$$

Since all nonlinear computations are now linearized, we can apply distributive law of multiplciation for these additive terms and trace the computation. We note that the Hadamard product enables an efficient mixing of all additive terms.

If we trace the computation, assuming that h_0 and c_0 are initialized with $\mathbf{0}$ vector, and input (x_1, x_2, x_3), we can collect the number of terms that are associated with input by symbolic computation. We verify that each of these terms are in fact different and can be understood as the output of a function that can take a subset of $\{x_1, x_2, x_3\}$ as input. These are interaction terms among different time steps, creating features that are mixings of these steps. We remove the bias term so that the symbolic tracing is still tractable. We provide a few examples of such mixing terms in Figure 2.

$$
\boxed{
\begin{array}{l}
L_\sigma(W_o x_2) * L_{\tanh}((L_\sigma(W_f x_2) * (L_\sigma(W_i x_1) * L_{\tanh}(W_g x_1)))) \\
L_\sigma(W_o x_2) * L_{\tanh}((L_\sigma(W_i x_2) * L_{\tanh}(W_g x_2)))
\end{array}
}
$$

Figure 2: We show a few terms from the symbolic tracer's output when the LSTM has processed both x_1 and x_2.

We count the statistics of terms that are associated with each input at the first three time steps. Each term is a unique feature computation of the input from the sequence (guaranteed by the uniqueness of Shapley value). We present the result of tracing in Table 1. This shows that LSTM is implicitly mixing inputs to allow interactions, and the final hidden state h_n, assuming the sequence is of length n, can be decomposed to many terms that contain combinations of $x_1, ..., x_n$.

Terms	x_1 Step	x_2 Step	x_3 Step
x_1	1	2	16
x_2	—	1	2
$x_1 x_2$	—	9	2,574
x_3	—	—	1
$x_2 x_3$	—	—	9
$x_1 x_3$	—	—	28
$x_1 x_2 x_3$	—	—	581
Total	1	12	3,211

Table 1: Number of unique terms that are associated with inputs when the LSTM progresses. We observe an exponential increase of terms as LSTM progresses.

We show that the Hadamard product provides the much needed mixing of time steps, and each time step's feature is processed using existing weight matrices but through different ways — enabled by nonlinearity. Our result shows an alternative explanation on why LSTM is so effective at creating representations of an entire sequence — by creating interaction terms of time steps implicitly. Previous work hypothesized that the advantage of the LSTM comes from the addition in the cell state computation: $f_t \odot c_{t-1} + i_t \odot g_t$, which resembles skip-connections between time steps, or improves the effectiveness on the gradient flows (Chung et al., 2014). However, our analysis shows that the high expressivity brought by the Hadamard product $\odot$ might contribute to the overall effectiveness of the LSTM network as well.

2.4 Contextual Decomposition

Murdoch et al. (2018) proposed the contextual decomposition algorithm to interpret which part of the text sequence contributes most to the LSTM prediction. Given a range of sequence $x_i, ..., x_j$, $1 \leq i < j \leq T$, contextual decomposition rearranges the terms at every time step t, such that each hidden and cell state can be decomposed into a relevant part associated with $x_i, ..., x_j$, denoted by β, and an irrelevant part, denoted by γ (Eq 5).

$$
\begin{aligned}
c^t &= c^t_\beta + c^t_\gamma \\
h^t &= h^t_\beta + h^t_\gamma
\end{aligned}
\tag{5}
$$

Since the recurrent computation is fully linear and additive, the rearrangements of Shapley values will produce the same hidden and cell state as the original computation. At the final step of LSTM recurrence, h^T is used as the feature representation of the entire sentence. In a binary classification setting, the probability for label y can be computed by the dot product between hidden state h^T and the weight W. We can easily calculate the contextual decomposition score (contribution score) s for a given range $x_i, ..., x_j$ by calculating dot product between relevant hidden state h^T_β and the weight W.

$$\begin{aligned} \hat{y} &= Wh^T = Wh^T_\beta + Wh^T_\gamma \\ s &= Wh^T_\beta \end{aligned} \tag{6}$$

2.5 Agglomerative Contextual Decomposition

As we discussed in Section 2.3, tracing all interactive terms of all time steps is intractable. The problem of how to find out which combinations of input in a given sequence contributed the most to the final label prediction remains. Singh et al. (2018) proposed a hierarchical clustering method to discover sub-sequences that contribute the most to the final prediction, where the contribution score calculated by contextual decomposition algorithm is used as the metric to determine which clusters to join at each step.

We explain the procedure in Figure 1. We describe a simplified version of their algorithm:

- **Initialize**: Compute a contribution score for each word using the contextual decomposition algorithm and add these words to a priority queue with their scores.
- **Select**: Dequeue and obtain the word with the highest absolute contribution score.
- **Update**: Update contribution scores of other unselected words by adjusting the range of contextual decomposition algorithm to include the adjacent words.
- **Finalize**: Repeat Select and Update until the queue is empty.

2.6 Tree Generation

As the agglomerative contextual decomposition algorithm progesses, text blocks will be formed during iterations. By tracing how the merge happens at every step, we can create a tree-like structure that is the phrase-structure grammar of the sentence. We explain the procedure in Figure 1. The merging will stop when all regions are merged together. We binarize the trees by using left chomsky normal form for further evaluation and analysis.

Connection to Construction Grammar We note that by selecting and merging text spans that have the highest contribution scores, we are letting the classifier that maps a sentence to a semantic attribute (such as sentiment) to define the structure of the sentence. We leave to future work to examine possible connection between the structure induction through machine interpretation algorithm and construction grammar (Goldberg, 1995) and whether model interpretation algorithm such as $CD(x)$ that maps a text span to a real-valued semantic score has connection to the classic interpretation function $[\![\cdot]\!]$ proposed by Lewis (1970).

3 Experiments

3.1 Generation

3.1.1 Model Training

We trained a simple 1-layer unidirctional LSTM sentence classification model on Stanford Sentiment Treebank (SST) (Socher et al., 2013). We use pre-trained 300d GloVe embedding (Pennington et al., 2014). We use Adam optimizer (Kingma and Ba, 2014) with learning rate 0.001 to optimize the algorithm. We obtain 82.2% and 85.3% accuracy with hidden state dimension 50 and 500 on the binary classification task of positive and negative sentiment on the test dataset.

3.1.2 Tree Generation

We generate tree structures by tracing the selections made by the agglomerative contextual decomposition (ACD) algorithm, and binarize the final tree. We find that this algorithm becomes very inefficient for any sequence longer than 20 words, so we focus on generating structures from SST sequences that are shorter than 20 words.

3.2 Evaluation

We are interested in two aspects: i) Syntactic: how does our generated trees compare with gold trees constructed by Stanford CoreNLP parser (Manning et al., 2014)? ii) Semantic: do our generated trees contain rich semantic information? We show an overview of the syntactic and semantic evaluation in Figure 3.

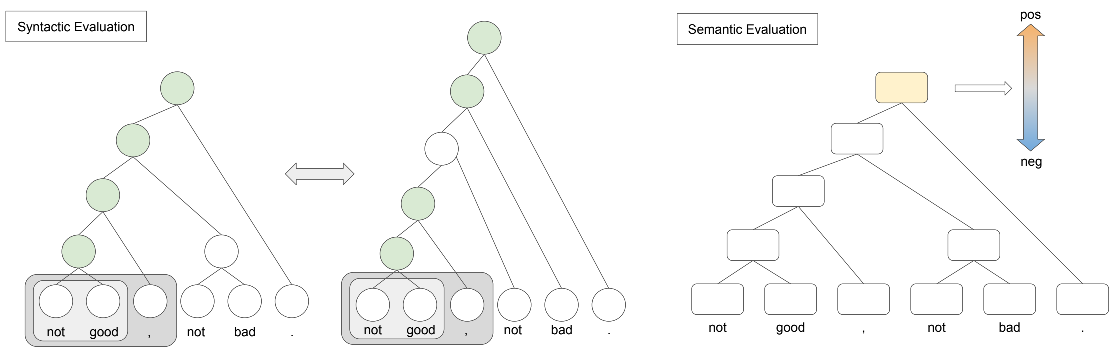

Figure 3: Syntactic and semantic evaluation of our results. For syntactic evaluation, we compare our trees with left-leaning trees, right-leaning trees and gold reference trees. An example of syntactic similarity evaluation is shown in the left part. Four out of five intermediate nodes (shown in green) represent same text span (e.g., the leftmost node represents 'not good', the second leftmost node represents 'not good ,'), thus the similarity of the two trees are 0.8. For semantic evaluation, we train a tree recursive neural network on our generated trees with sentiment labels. Each node is embedded into a low-dimensional continuous space and can represent the sentiment of the node by computing the dot product with the weight matrix. We report the sentiment classification accuracy of only the root node or all nodes.

3.2.1 Syntactic Evaluation

We compare the generated tree structures with three types of trees: always left-leaning trees (**LS**), always right-leaning trees (**RS**), and gold reference trees (**GS**) produced by Stanford CoreNLP parser. We also compute the result of randomly generated trees to compare with trees generated from ACD algorithm. Results are reported in Table 2. We use the same script from Williams et al. (2018) that computes the Jaccard distance between set representation of two trees.

Compared with randomly generated trees, here we see that LSTM does capture structures that more closely resembles the gold reference, but there are still significant differences. LSTM with 500 dimension hidden state performed better on the original sentiment classification task (85.3% vs 82.2% accuracy), and generated trees are more balanced than LSTM with 50 dimension hidden state. This is also a phenomenon discovered by Williams et al. (2018) that balanced trees are often implicitly produced by the machine learning algorithms.

3.2.2 Semantic Evaluation

We also train a recursive neural network on these generated structures. We use the contribution score s for each phrase as the intermediate labels and we allow the recursive computation step to be either a normal RNN or an LSTM. We evaluate the label accuracy on all nodes (All) or only on the root node (Root). The generated structure under-performed gold reference trees by a large margin, and is also below the original LSTM's performance, indicating that structures recovered by ACD are not equivalent to the true LSTM sequence computing process.

	LS	RS	GS	AD
Random	29.3	29.2	27.6	4.19
LSTM-50d	36.9	25.7	29.7	5.53
LSTM-500d	33.7	32.5	30.2	5.91

Table 2: The set overlap similarity between generated trees and gold trees. **LS** means left-leaning trees. **RS** means right-leaning trees. **GS** means gold parse trees. **AD** means average tree depth.

Model	RNN		LSTM	
	All	Root	All	Root
LSTM-50d	72.7	60.4	74.8	63.3
LSTM-500d	75.1	53.6	75.9	58.0
Gold Trees	75.9	74.5	78.2	78.1

Table 3: The accuracy on the SST test set. Gold tree set is also composed of sequences shorter than 20 words.

4 Conclusion

In this work, we extract trees from LSTM by an interpretation algorithm — agglomerative contextual decomposition. We show empirically that the generated trees are not similar to the trees produced from formal syntactic theory. The generated trees also do not seem to provide more computational improvement when we train a recursive neural network leveraging the structure to predict the final label. We conclude with encouragement for the community to look deeper into interpretation-based methods and their connections with semantic and syntactic theory in linguistics.

References

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. *arXiv preprint arXiv:1805.01070*.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, D Anthony Bau, and James Glass. 2019. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 7.

Adele E Goldberg. 1995. *Constructions: A construction grammar approach to argument structure*. University of Chicago Press.

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

Eric Jonas and Konrad Paul Kording. 2017. Could a neuroscientist understand a microprocessor? *PLoS computational biology*, 13(1):e1005268.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. *arXiv preprint arXiv:1805.04623*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. Lstms can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436.

David Lewis. 1970. General semantics. *Synthese*, 22(1-2):18–67.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *arXiv preprint arXiv:1611.01368*.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

W James Murdoch, Peter J Liu, and Bin Yu. 2018. Beyond word importance: Contextual decomposition to extract interactions from lstms. *arXiv preprint arXiv:1801.05453*.

Jeffrey Pennington, Richard Socher, and Christopher 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.

Chandan Singh, W James Murdoch, and Bin Yu. 2018. Hierarchical interpretations for neural network predictions. *arXiv preprint arXiv:1806.05337*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Adina Williams, Andrew Drozdov*, and Samuel R Bowman. 2018. Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association of Computational Linguistics*, 6:253–267.