To Attend or not to Attend: A Case Study on Syntactic Structures for Semantic Relatedness

Amulya Gupta
Iowa State University
guptaam@iastate.edu

Zhu Zhang
Iowa State University
zhuzhang@iastate.edu

Abstract

With the recent success of Recurrent Neural Networks (RNNs) in Machine Translation (MT), attention mechanisms have become increasingly popular. The purpose of this paper is two-fold; firstly, we propose a novel attention model on Tree Long Short-Term Memory Networks (Tree-LSTMs), a tree-structured generalization of standard LSTM. Secondly, we study the interaction between attention and syntactic structures, by experimenting with three LSTM variants: bidirectional-LSTMs, Constituency Tree-LSTMs, and Dependency Tree-LSTMs. Our models are evaluated on two semantic relatedness tasks: semantic relatedness scoring for sentence pairs (SemEval 2012, Task 6 and SemEval 2014, Task 1) and paraphrase detection for question pairs (Quora, 2017).[1]

1 Introduction

Recurrent Neural Networks (RNNs), in particular Long Short-Term Memory Networks (LSTMs) (Hochreiter and Schmidhuber, 1997), have demonstrated remarkable accomplishments in Natural Language Processing (NLP) in recent years. Several tasks such as information extraction, question answering, and machine translation have benefited from them. However, in their vanilla forms, these networks are constrained by the sequential order of tokens in a sentence. To mitigate this limitation, structural (*dependency* or *constituency*) information in a sentence was exploited and witnessed partial success in various tasks (Goller and Kuchler, 1996; Yamada and Knight, 2001; Quirk et al., 2005; Socher et al., 2011; Tai et al., 2015).

On the other hand, alignment techniques (Brown et al., 1993) and attention mechanisms (Bahdanau et al., 2014) act as a catalyst to augment the performance of classical Statistical Machine Translation (SMT) and Neural Machine Translation (NMT) models, respectively. In short, both approaches focus on sub-strings of source sentence which are significant for predicting target words while translating. Currently, the combination of linear RNNs/LSTMs and attention mechanisms has become a *de facto* standard architecture for many NLP tasks.

At the intersection of sentence encoding and attention models, some interesting questions emerge: Can attention mechanisms be employed on tree structures, such as Tree-LSTMs (Tai et al., 2015)? If yes, what are the possible tree-based attention models? Do different tree structures (in particular constituency vs. dependency) have different behaviors in such models? With these questions in mind, we present our investigation and findings in the context of semantic relatedness tasks.

2 Background

2.1 Long Short-Term Memory Networks (LSTMs)

Concisely, an LSTM network (Hochreiter and Schmidhuber, 1997) (Figure 1) includes a *memory cell* at each time step which controls the amount of information being penetrated into the cell, neglected, and yielded by the cell. Various LSTM networks (Greff et al., 2017) have been explored till now; we focus on one representative form. To be more precise, we consider a LSTM memory cell involving: an *input gate* i_t, a forget gate f_t, and an output gate o_t at time step t. Apart from

[1] Our code for experiments on the SICK dataset is publicly available at https://github.com/amulyahwr/acl2018

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 2116–2125
Melbourne, Australia, July 15 - 20, 2018.

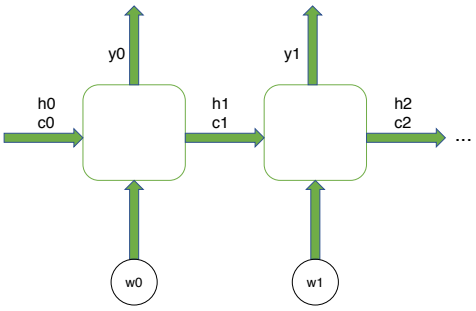

Figure 1: A linear LSTM network. w_t is the word embedding, h_t is the hidden state vector, c_t is the memory cell vector and y_t is the final processed output at time step *t*.

the hidden state h_{t-1} and input embedding w_t of the current word, the recursive function in LSTM also takes the previous time's *memory cell state*, c_{t-1}, into account, which is not the case in simple RNN. The following equations summarize a LSTM memory cell at time step *t*:

$$i_t = \sigma(w_t W^i + h_{t-1} R^i + b^i) \quad (1)$$

$$f_t = \sigma(w_t W^f + h_{t-1} R^f + b^f) \quad (2)$$

$$o_t = \sigma(w_t W^o + h_{t-1} R^o + b^o) \quad (3)$$

$$u_t = \tanh(w_t W^u + h_{t-1} R^u + b^u) \quad (4)$$

$$c_t = i_t \odot u_t + f_t \odot c_{t-1} \quad (5)$$

$$h_t = o_t \odot \tanh(c_t) \quad (6)$$

where:

- $(\boldsymbol{W^i}, \boldsymbol{W^f}, \boldsymbol{W^o}, \boldsymbol{W^u}) \in \mathbb{R}^{\mathrm{D\ x\ d}}$ represent input weight matrices, where $\boldsymbol{d}$ is the dimension of the hidden state vector and $\boldsymbol{D}$ is the dimension of the input word embedding, $\boldsymbol{w_t}$.
- $(\boldsymbol{R^i}, \boldsymbol{R^f}, \boldsymbol{R^o}, \boldsymbol{R^u}) \in \mathbb{R}^{\mathrm{d\ x\ d}}$ represent recurrent weight matrices and $(\boldsymbol{b^i}, \boldsymbol{b^f}, \boldsymbol{b^o}, \boldsymbol{b^u}) \in \mathbb{R}^{\mathrm{d}}$ represent biases.
- $\boldsymbol{c_t} \in \mathbb{R}^{\mathrm{d}}$ is the new memory cell vector at time step *t*.

As can be seen in Eq. 5, the input gate i_t limits the new information, u_t, by employing the element wise multiplication operator $\odot$. Moreover, the forget gate f_t regulates the amount of information from the previous state c_{t-1}. Therefore, the current memory state c_t includes both new and previous time step's information but partially.

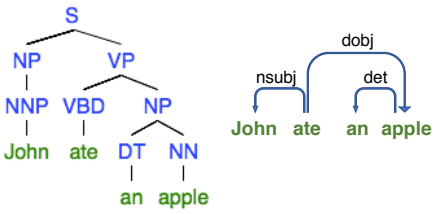

Figure 2: a. **Left**: A constituency tree; b. **Right**: A dependency tree

A natural extension of LSTM network is a *bidirectional* LSTM (bi-LSTM), which lets the sequence pass through the architecture in both directions and aggregate the information at each time step. Again, it strictly preserves the sequential nature of LSTMs.

2.2 Linguistically Motivated Sentence Structures

Most computational linguists have developed a natural inclination towards hierarchical structures of natural language, which follow guidelines collectively referred to as *syntax*. Typically, such structures manifest themselves in *parse trees*. We investigate two popular forms: *Constituency* and *Dependency* trees.

2.2.1 Constituency structure

Briefly, constituency trees (Figure 2:a) indicate a hierarchy of syntactic units and encapsulate phrase grammar rules. Moreover, these trees explicitly demonstrate groups of phrases (e.g., Noun Phrases) in a sentence. Additionally, they discriminate between terminal (lexical) and non-terminal nodes (non-lexical) tokens.

2.2.2 Dependency structure

In short, dependency trees (Figure 2:b) describe the syntactic structure of a sentence in terms of the words (lemmas) and associated grammatical relations among the words. Typically, these dependency relations are explicitly *typed*, which makes the trees valuable for practical applications such as information extraction, paraphrase detection and semantic relatedness.

2.3 Tree Long Short-Term Memory Network (Tree-LSTM)

Child-Sum Tree-LSTM (Tai et al., 2015) is an epitome of structure-based neural network which explicitly capture the structural information in a sentence. Tai et al. demonstrated that information at

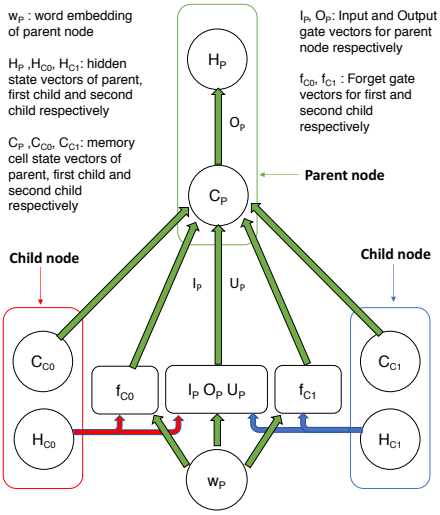

Figure 3: A compositional view of parent node in Tree-LSTM network.

a parent node can be consolidated selectively from each of its child node. Architecturally, each gated vector and memory state update of the head node is dependent on the hidden states of its children in the Tree-LSTM. Assuming a good tree structure of a sentence, each node *j* of the structure incorporates the following equations.:

$$\tilde{h_j} = \sum_{k \in C(j)} h_k \tag{7}$$

$$i_j = \sigma(w_j W^i + \tilde{h_j} R^i + b^i) \tag{8}$$

$$f_{jk} = \sigma(w_j W^f + h_k R^f + b^f) \tag{9}$$

$$o_j = \sigma(w_j W^o + \tilde{h_j} R^o + b^o) \tag{10}$$

$$u_j = \tanh(w_j W^u + \tilde{h_j} R^u + b^u) \tag{11}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{12}$$

$$h_j = o_j \odot \tanh(c_j) \tag{13}$$

where:

- $\boldsymbol{w_j} \in \mathbb{R}^{\mathrm{D}}$ represents word embedding of all nodes in Dependency structure and only terminal nodes in Constituency structure. [2]
- $(\boldsymbol{W^i}, \boldsymbol{W^f}, \boldsymbol{W^o}, \boldsymbol{W^u}) \in \mathbb{R}^{\mathrm{D\ x\ d}}$ represent input weight matrices.
- $(\boldsymbol{R^i}, \boldsymbol{R^f}, \boldsymbol{R^o}, \boldsymbol{R^u}) \in \mathbb{R}^{\mathrm{d\ x\ d}}$ represent recurrent weight matrices, and $(\boldsymbol{b^i}, \boldsymbol{b^f}, \boldsymbol{b^o}, \boldsymbol{b^u}) \in \mathbb{R}^{\mathrm{d}}$ represent biases.

[2] $\boldsymbol{w_j}$ is ignored for non-terminal nodes in a Constituency structure by removing the $\boldsymbol{wW}$ terms in Equations 8-11.

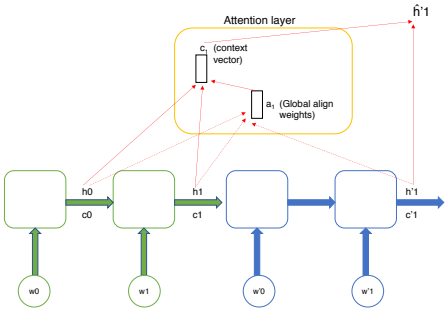

Figure 4: Global attention model

- $\boldsymbol{c_j} \in \mathbb{R}^{\mathrm{d}}$ is the new memory state vector of node *j*.
- $\boldsymbol{C(j)}$ is the set of children of node *j*.
- $\boldsymbol{f_{jk}} \in \mathbb{R}^{\mathrm{d}}$ is the forget gate vector for child *k* of node *j*.

Referring to Equation 12, the new memory cell state, c_j of node *j*, receives new information, u_j, partially. More importantly, it includes the partial information from each of its direct children, set *C(j)*, by employing the corresponding forget gate, f_{jk}.

When the Child-Sum Tree model is deployed on a dependency tree, it is referred to as *Dependency Tree-LSTM*, whereas a constituency-tree-based instantiation is referred to as *Constituency Tree-LSTM*.

2.4 Attention Mechanisms

Alignment models were first introduced in statistical machine translation (SMT) (Brown et al., 1993), which connect sub-strings in the source sentence to sub-strings in the target sentence.

Recently, attention techniques (which are effectively soft alignment models) in neural machine translation (NMT) (Bahdanau et al., 2014) came into prominence, where *attention* scores are calculated by considering words of source sentence while decoding words in target language. Although effective attention mechanisms (Luong et al., 2015) such as Global Attention Model (GAM) (Figure 4) and Local Attention Model (LAM) have been developed, such techniques have not been explored over Tree-LSTMs.

3 Inter-Sentence Attention on Tree-LSTMs

We present two types of tree-based attention models in this section. With trivial adaptation, they can

be deployed in the sequence setting (degenerated trees).

3.1 Modified Decomposable Attention (MDA)

Parikh et al. (2016)'s original decomposable inter-sentence attention model only used word embeddings to construct the attention matrix, without any structural encoding of sentences. Essentially, the model incorporated three components:

Attend: Input representations (without sequence or structural encoding) of both sentences, L and R, are soft-aligned.

Compare: A set of vectors is produced by separately comparing each sub-phrase of L to sub-phrases in R. Vector representation of each sub-phrase in L is a non-linear combination of representation of word in sentence L and its aligned sub-phrase in sentence R. The same holds true for the set of vectors for sentence R.

Aggregate: Both sets of sub-phrases vectors are summed up separately to form final sentence representation of sentence L and sentence R.

We decide to augment the original decomposable inter-sentence attention model and generalize it into the tree (and sequence) setting. To be more specific, we consider two input sequences: L = (l_1, l_2....l_{len_L}), R = (r_1, r_2....r_{len_R}) and their corresponding input representations: $\bar{L}$ = ($\bar{l}_1$, $\bar{l}_2$....$\bar{l}_{len_L}$), $\bar{R}$ = ($\bar{r}_1$, $\bar{r}_2$....$\bar{r}_{len_R}$); where len_L and len_R represents number of words in L and R, respectively.

3.1.1 MDA on dependency structure

Let's assume sequences L and R have dependency tree structures D_L and D_R. In this case, len_L and len_R represents number of nodes in D_L and D_R, respectively. ***After*** using a Tree-LSTM to encode tree representations, which results in: $D_L^{'}$ = ($\bar{l}_1^{'}$, $\bar{l}_2^{'}$....$\bar{l}_{len_L}^{'}$), $D_R^{'}$ = ($\bar{r}_1^{'}$, $\bar{r}_2^{'}$....$\bar{r}_{len_R}^{'}$), we gather unnormalized attention weights, e_{ij} and normalize them as follows:

$$e_{ij} = \bar{l}_i^{'}(\bar{r}_j^{'})^T \tag{14}$$

$$\beta_i = \sum_{j=1}^{len_R} \frac{exp(e_{ij})}{\sum_{k=1}^{len_R} exp(e_{ik})} * \bar{r}_j^{'} \tag{15}$$

$$\alpha_j = \sum_{i=1}^{len_L} \frac{exp(e_{ij})}{\sum_{k=1}^{len_L} exp(e_{kj})} * \bar{l}_i^{'} \tag{16}$$

From the equations above, we can infer that the attention matrix will have a dimension len_L x len_R. In contrast to the original model, we compute the final representations of the each sentence by concatenating the LSTM-encoded representation of root with the attention-weighted representation of the root [3]:

$$h_L^{''} = G([\bar{l}_{root_L}^{'}; \beta_{root_L}]) \tag{17}$$

$$h_R^{''} = G([\bar{r}_{root_R}^{'}; \alpha_{root_R}]) \tag{18}$$

where G is a feed-forward neural network. $h_L^{''}$ and $h_R^{''}$ are final vector representations of input sequences *L* and *R*, respectively.

3.1.2 MDA on constituency structure

Let's assume sequences L and R have constituency tree structures C_L and C_R. Moreover, assume C_L and C_R have total number of nodes as N_L ($> len_L$) and N_R ($> len_R$), respectively. As in 3.1.1, the attention mechanism is employed **after** encoding the trees C_L and C_R. While encoding trees, terminal and non-terminal nodes are handled in the same way as in the original Tree-LSTM model (see 2.3).

It should be noted that we collect hidden states of all the nodes (N_L and N_R) individually in C_L and C_R during the encoding process. Hence, hidden states matrix will have dimension N_L x *d* for tree C_L whereas for tree C_R, it will have dimension N_R x *d*; where *d* is dimension of each hidden state. Therefore, attention matrix will have a dimension N_L x N_R. Finally, we employ Equations 14-18 to compute the final representations of sequences L and R.

3.2 Progressive Attention (PA)

In this section, we propose a novel attention mechanism on Tree-LSTM, inspired by (Quirk et al., 2005) and (Yamada and Knight, 2001).

3.2.1 PA on dependency structure

Let's assume a dependency tree structure of sentence L = (l_1, l_2....l_{len_L}) is available as D_L; where len_L represents number of nodes in D_L. Similarly, tree D_R corresponds to the sentence R = (r_1, r_2....r_{len_R}); where len_R represents number of nodes in D_R.

In PA, the objective is to produce the final vector representation of tree D_R conditional on the hidden state vectors of *all* nodes of D_L. Similar to

[3] In the sequence setting, we compute the corresponding representations for the *last* word in the sentence.

the encoding process in NMT, we encode R by *attending* each node of D_R to all nodes in D_L. Let's name this process *Phase1*. Next, *Phase2* is performed where L is encoded in the similar way to get the final vector representation of D_L.

Referring to Figure 5 and assuming Phase1 is being executed, a hidden state matrix, H_L, is obtained by concatenating the hidden state vector of every node in tree D_L, where the number of nodes in D_L = 3. Next, tree D_R is processed by calculating the hidden state vector at every node. Assume that the current node being processed is n_{R2} of D_R, which has a hidden state vector, h_{R2}. Before further processing, normalized weights are calculated based on h_{R2} and H_L. Formally,

$$H_{pj} = stack[h_{pj}] \tag{19}$$

$$con_{pj} = concat[H_{pj}, H_q] \tag{20}$$

$$a_{pj} = softmax(tanh(con_{pj}W_c + b) * W_a) \tag{21}$$

where:

- $\boldsymbol{p}, \boldsymbol{q} \in \{\boldsymbol{L}, \boldsymbol{R}\}$ and $\boldsymbol{q} \neq \boldsymbol{p}$
- $\boldsymbol{H_q} \in \mathbb{R}^{\boldsymbol{x}\text{ x d}}$ represents a matrix obtained by concatenating hidden state vectors of nodes in tree $\boldsymbol{D_q}$; $\boldsymbol{x}$ is $\boldsymbol{len_q}$ of sentence $\boldsymbol{q}$.
- $\boldsymbol{H_{pj}} \in \mathbb{R}^{\boldsymbol{x}\text{ x d}}$ represents a matrix obtained by stacking hidden state, $\boldsymbol{h_{pj}}$, vertically $\boldsymbol{x}$ times.
- $\boldsymbol{con_{pj}} \in \mathbb{R}^{\boldsymbol{x}\text{ x 2d}}$ represents the concatenated matrix.
- $\boldsymbol{a_{pj}} \in \mathbb{R}^{\boldsymbol{x}}$ represents the normalized attention weights at node *j* of tree $\boldsymbol{D_p}$; where $\boldsymbol{D_p}$ is the dependency structure of sentence $\boldsymbol{p}$.
- $\boldsymbol{W_c} \in \mathbb{R}^{\text{2d x d}}$ and $\boldsymbol{W_a} \in \mathbb{R}^{\text{d}}$ represent learned weight matrices.

The normalized attention weights in above equations provide an opportunity to align the subtree at the current node, n_{R2}, in D_R to sub-trees available at all nodes in D_L. Next, a gated mechanism is employed to compute the final vector representation at node n_{R2}.

Formally,

$$h'_{pj} = \sum_{0}^{(x-1)} ((1 - a_{pj}) * H_q + (a_{pj}) * H_{pj}) \tag{22}$$

where:

- $\boldsymbol{h'_{pj}} \in \mathbb{R}^{\text{d}}$ represents the final vector representation of node *j* in tree $\boldsymbol{D_p}$
- $\sum_{\boldsymbol{0}}^{\boldsymbol{(x-1)}}$ represents column-wise sum

Assuming the final vector representation of tree D_R is h'_R, the exact same steps are followed for Phase2 with the exception that the entire process is now conditional on tree D_R. As a result, the final vector representation of tree D_L, h'_L, is computed.

Lastly, the following equations are applied to vectors h'_L and h'_R, before calculating the *angle* and *distance* similarity (see Section 4).

$$h''_L = tanh(h'_L + h_L) \tag{23}$$

$$h''_R = tanh(h'_R + h_R) \tag{24}$$

where:

- $\boldsymbol{h_L} \in \mathbb{R}^{\text{d}}$ represents the vector representation of tree $\boldsymbol{D_L}$ without attention.
- $\boldsymbol{h_R} \in \mathbb{R}^{\text{d}}$ represents the vector representation of tree $\boldsymbol{D_R}$ without attention.

3.2.2 PA on constituency structure

Let C_L and C_R represent constituency trees of L and R, respectively; where C_L and C_R have total number of nodes N_L ($> len_L$) and N_R ($> len_R$). Additionally, let's assume that trees C_L and C_R have the same configuration of nodes as in Section 3.1.2, and the encoding of *terminal* and *non-terminal* nodes follow the same process as in Section 3.1.2. Assuming we have already encoded all N_L nodes of tree C_L using Tree-LSTM, we will have the hidden state matrix, H_L, with dimension N_L x d. Next, while encoding any node of C_R, we consider H_L which results in an attention vector having shape N_L. Using Equations 19-22 [4], we retrieve the final hidden state of the current node. Finally, we compute the representation of sentence R based on attention to sentence L. We perform Phase2 with the same process, except that we now condition on sentence R.

In summary, the *progressive attention* mechanism refers to all nodes in the other tree ***while*** encoding a node in the current tree, instead of waiting till the end of the structural encoding to establish cross-sentence attention, as was done in the *decomposable attention* model.

[4] At this point, we will consider $\boldsymbol{C_q}$ and $\boldsymbol{C_p}$ instead of $\boldsymbol{D_q}$ and $\boldsymbol{D_p}$, respectively, in Equations 19-22. Additionally, $\boldsymbol{x}$ will be equal to total number of nodes in the constituency tree.

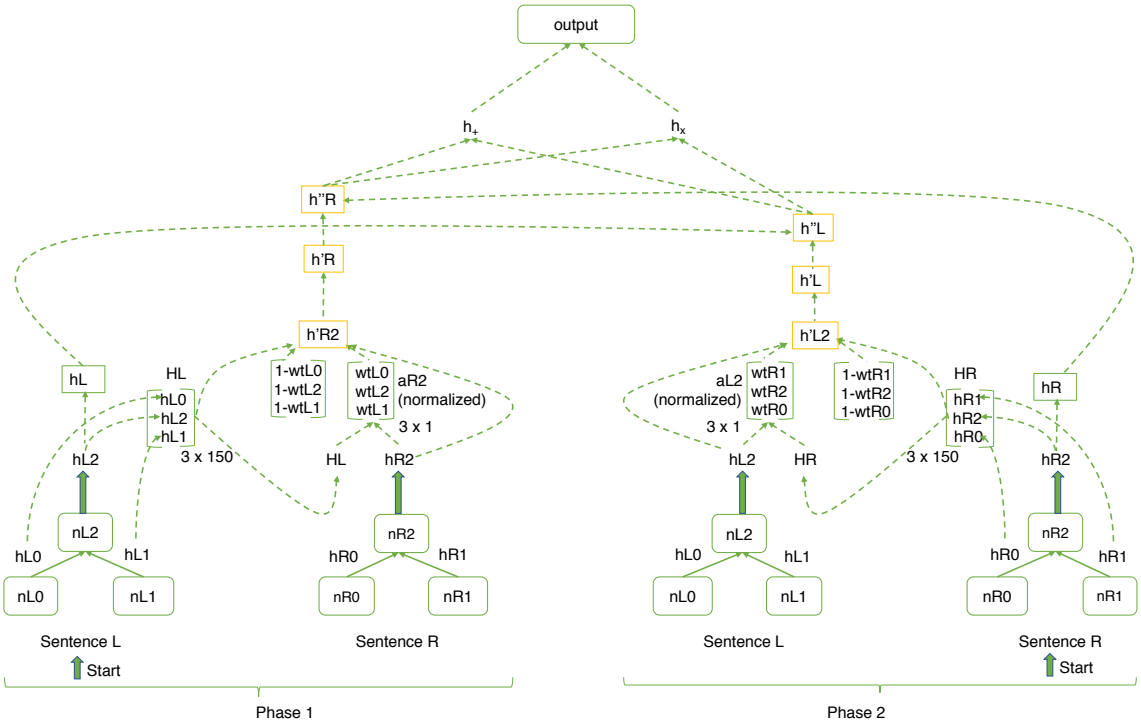

Figure 5: Progressive Attn-Tree-LSTM model

4 Evaluation Tasks

We evaluate our models on two tasks: (1) semantic relatedness scoring for sentence pairs (SemEval 2012, Task 6 and SemEval 2014, Task 1) and (2) paraphrase detection for question pairs (Quora, 2017).

4.1 Semantic Relatedness for Sentence Pairs

In SemEval 2012, Task 6 and SemEval 2014, Task 1, every sentence pair has a real-valued score that depicts the extent to which the two sentences are semantically related to each other. Higher score implies higher semantic similarity between the two sentences. Vector representations $h_L^{''}$ and $h_R^{''}$ are produced by using our *Modified Decomp-Attn* or *Progressive-Attn* models. Next, a similarity score, $\hat{y}$ between $h_L^{''}$ and $h_R^{''}$ is computed using the same neural network (see below), for the sake of fair comparison between our models and the original *Tree-LSTM* (Tai et al., 2015).

$$h_x = h_L^{''} \odot h_R^{''} \tag{25}$$

$$h_+ = |h_L^{''} - h_R^{''}| \tag{26}$$

$$h_s = \sigma(h_x W^x + h_+ W^+ + b^h) \tag{27}$$

$$\hat{p_\theta} = softmax(h_s W^p + b^p) \tag{28}$$

$$\hat{y} = r^T \hat{p_\theta} \tag{29}$$

where:

- $r^T = [1, 2..S]$
- $h_x \in \mathbb{R}^d$ measures the sign similarity between $h_L^{''}$ and $h_R^{''}$
- $h_+ \in \mathbb{R}^d$ measures the absolute distance between $h_L^{''}$ and $h_R^{''}$

Following (Tai et al., 2015), we convert the regression problem into a soft classification. We also use the same sparse distribution, p, which was defined in the original Tree-LSTM to transform the gold rating for a sentence pair, such that $y = r^T p$ and $\hat{y} = r^T \hat{p_\theta} \approx y$. The loss function is the KL-divergence between p and $\hat{p}$:

$$J(\theta) = \frac{\sum_{k=1}^{m} KL(p^k || \hat{p}_\theta^k)}{m} + \frac{\lambda ||\theta||_2^2}{2} \tag{30}$$

- m is the number of sentence pairs in the dataset.
- λ represents the regularization penalty.

4.2 Paraphrase Detection for Question Pairs

In this task, each question pair is labeled as either paraphrase or not, hence the task is binary classification. We use Eqs. 25 - 28 to compute the

predicted distribution $\hat{p}_\theta$. The predicted label, $\hat{y}$, will be:

$$\hat{y} = \arg\max_y \hat{p}_\theta \tag{31}$$

The loss function is the negative log-likelihood:

$$J(\theta) = -\frac{\sum_{k=1}^{m} y^k \log \hat{y}^k}{m} + \frac{\lambda ||\theta||_2^2}{2} \tag{32}$$

5 Experiments

5.1 Semantic Relatedness for Sentence Pairs

We utilized two different datasets:

- The Sentences Involving Compositional Knowledge (SICK) dataset (Marelli et al. (2014)), which contains a total of 9,927 sentence pairs. Specifically, the dataset has a split of 4500/500/4927 among training, dev, and test. Each sentence pair has a score $S \in$ [1,5], which represents an average of 10 different human judgments collected by crowd-sourcing techniques.
- The MSRpar dataset (Agirre et al., 2012), which consists of 1,500 sentence pairs. In this dataset, each pair is annotated with a score $S \in$ [0,5] and has a split of 750/750 between training and test.

We used the Stanford Parsers (Chen and Manning, 2014; Bauer) to produce dependency and constituency parses of sentences. Moreover, we initialized the word embeddings with 300-dimensional Glove vectors (Pennington et al., 2014); the word embeddings were held fixed during training. We experimented with different optimizers, among which AdaGrad performed the best. We incorporated a learning rate of 0.025 and regularization penalty of 10^{-4} without dropout.

5.2 Paraphrase Detection for Question Pairs

For this task, we utilized the Quora dataset (Iyer; Kaggle, 2017). Given a pair of questions, the objective is to identify whether they are semantic duplicates. It is a binary classification problem where a duplicate question pair is labeled as 1 otherwise as 0. The training set contains about 400,000 labeled question pairs, whereas the test set consists of 2.3 million unlabeled question pairs. Moreover, the training dataset has only 37% positive samples; average length of a question is 10 words. Due to hardware and time constraints, we extracted 50,000 pairs from the original training while maintaining the same positive/negative ratio. A stratified 80/20 split was performed on this subset to produce the training/test set. Finally, 5% of the training set was used as a validation set in our experiments.

We used an identical training configuration as for the semantic relatedness task since the essence of both the tasks is practically the same. We also performed pre-processing to clean the data and then parsed the sentences using Stanford Parsers.

6 Results

6.1 Semantic Relatedness for Sentence Pairs

Table 1 summarizes our results. According to (Marelli et al., 2014), we compute three evaluation metrics: Pearson's *r*, Spearman's ρ and Mean Squared Error (MSE). We compare our attention models against the original Tree-LSTM (Tai et al., 2015), instantiated on both constituency trees and dependency trees. We also compare earlier baselines with our models, and the best results are in bold. Since Tree-LSTM is a generalization of Linear LSTM, we also implemented our attention models on Linear *Bidirectional* LSTM (Bi-LSTM). All results are average of 5 runs. It is witnessed that the *Progressive-Attn* mechanism combined with *Constituency Tree-LSTM* is overall the strongest contender, but PA failed to yield any performance gain on Dependency Tree-LSTM in either dataset.

6.2 Paraphrase Detection for Question Pairs

Table 2 summarizes our results where best results are highlighted in bold within each category. It should be noted that Quora is a new dataset and we have done our analysis on only 50,000 samples. Therefore, to the best of our knowledge, there is no published baseline result yet. For this task, we considered four standard evaluation metrics: Accuracy, F1-score, Precision and Recall. The *Progressive-Attn + Constituency Tree-LSTM* model still exhibits the best performance by a small margin, but the *Progressive-Attn* mechanism works surprisingly well on the linear bi-LSTM.

6.3 Effect of the Progressive Attention Model

Table 3 illustrates how various models operate on two sentence pairs from SICK test dataset. As we can infer from the table, the first pair demonstrates an instance of the active-passive voice phenomenon. In this case, the linear LSTM and vanilla Tree-LSTMs really struggle to perform.

Table 1: Results on test dataset for SICK and MSRpar semantic relatedness task. Mean scores are presented based on 5 runs (standard deviation in parenthesis). Categories of results: (1) Previous models (2) Dependency structure (3) Constituency structure (4) Linear structure

Dataset	Model	Pearson's r	Spearman's ρ	MSE
SICK	Illinois-LH (2014)	0.7993	0.7538	0.3692
	UNAL-NLP (2014)	0.8070	0.7489	0.3550
	Meaning factory (2014)	0.8268	0.7721	0.3224
	ECNU (2014)	0.8414	-	-
	Dependency Tree-LSTM (2015)	**0.8676 (0.0030)**	**0.8083 (0.0042)**	**0.2532 (0.0052)**
	Decomp-Attn (Dependency)	0.8239 (0.0120)	0.7614 (0.0103)	0.3326 (0.0223)
	Progressive-Attn (Dependency)	0.8424 (0.0042)	0.7733 (0.0066)	0.2963 (0.0077)
	Constituency Tree-LSTM (2015)	0.8582 (0.0038)	0.7966 (0.0053)	0.2734 (0.0108)
	Decomp-Attn (Constituency)	0.7790 (0.0076)	0.7074 (0.0091)	0.4044 (0.0152)
	Progressive-Attn (Constituency)	**0.8625 (0.0032)**	**0.7997 (0.0035)**	**0.2610 (0.0057)**
	Linear Bi-LSTM	0.8398 (0.0020)	0.7782 (0.0041)	0.3024 (0.0044)
	Decomp-Attn (Linear)	0.7899 (0.0055)	0.7173 (0.0097)	0.3897 (0.0115)
	Progressive-Attn (Linear)	**0.8550 (0.0017)**	**0.7873 (0.0020)**	**0.2761 (0.0038)**
MSRpar	ParagramPhrase (2015)	0.426	-	-
	Projection (2015)	0.437	-	-
	GloVe (2015)	0.477	-	-
	PSL (2015)	0.416	-	-
	ParagramPhrase-XXL (2015)	0.448	-	-
	Dependency Tree-LSTM	**0.4921 (0.0112)**	**0.4519 (0.0128)**	**0.6611 (0.0219)**
	Decomp-Attn (Dependency)	0.4016 (0.0124)	0.3310 (0.0118)	0.7243 (0.0099)
	Progressive-Attn (Dependency)	0.4727 (0.0112)	0.4216 (0.0092)	0.6823 (0.0159)
	Constituency Tree-LSTM	0.3981 (0.0176)	0.3150 (0.0204)	0.7407 (0.0170)
	Decomp-Attn (Constituency)	0.3991 (0.0147)	0.3237 (0.0355)	0.7220 (0.0185)
	Progressive-Attn (Constituency)	**0.5104 (0.0191)**	**0.4764 (0.0112)**	**0.6436 (0.0346)**
	Linear Bi-LSTM	0.3270 (0.0303)	0.2205 (0.0111)	0.8098 (0.0579)
	Decomp-Attn (Linear)	0.3763 (0.0332)	0.3025 (0.0587)	0.7290 (0.0206)
	Progressive-Attn (Linear)	**0.4773 (0.0206)**	**0.4453 (0.0250)**	**0.6758 (0.0260)**

Table 2: Results on test dataset for Quora paraphrase detection task. Mean scores are presented based on 5 runs (standard deviation in parenthesis). Categories of results: (1) Dependency structure (2) Constituency structure (3) Linear structure

Model	Accuracy	F-1 score (class=1)	Precision (class=1)	Recall (class=1)
Dependency Tree-LSTM	**0.7897 (0.0009)**	0.7060 (0.0050)	**0.7298 (0.0055)**	0.6840 (0.0139)
Decomp-Attn (Dependency)	0.7803 (0.0026)	0.6977 (0.0074)	0.7095 (0.0083)	0.6866 (0.0199)
Progressive-Attn (Dependency)	0.7896 (0.0025)	**0.7113 (0.0087)**	0.7214 (0.0117)	**0.7025 (0.0266)**
Constituency Tree-LSTM	0.7881 (0.0042)	0.7065 (0.0034)	0.7192 (0.0216)	0.6846 (0.0380)
Decomp-Attn (Constituency)	0.7776 (0.0004)	0.6942 (0.0050)	0.7055 (0.0069)	0.6836 (0.0164)
Progressive-Attn (Constituency)	**0.7956 (0.0020)**	**0.7192 (0.0024)**	**0.7300 (0.0079)**	**0.7089 (0.0104)**
Linear Bi-LSTM	0.7859 (0.0024)	0.7097 (0.0047)	0.7112 (0.0129)	0.7089 (0.0219)
Decomp-Attn (Linear)	0.7861 (0.0034)	0.7074 (0.0109)	0.7151 (0.0135)	0.7010 (0.0315)
Progressive-Attn (Linear)	**0.7949 (0.0031)**	**0.7182 (0.0162)**	**0.7298 (0.0115)**	**0.7092 (0.0469)**

However, when our progressive attention mechanism is integrated into syntactic structures (dependency or constituency), we witness a boost in the semantic relatedness score. Such desirable behavior is consistently observed in multiple active-passive voice pairs. The second pair points to a possible issue in data annotation. Despite the presence of strong negation, the gold-standard score is 4 out of 5 (indicating high relatedness). Interestingly, the *Progressive-Attn + Dependency Tree-LSTM* model favors the negation facet and outputs a low relatedness score.

7 Discussion

In this section, let's revisit our research questions in light of the experimental results.

First, can attention mechanisms be built for Tree-LSTMs? Does it work? The answer is yes. Our novel progressive-attention Tree-LSTM model, when instantiated on constituency trees,

Table 3: Effect of the progressive attention model

ID	Test Pair	Gold	BiLSTM		Const. Tree		Dep. Tree	
			(no attn)	(PA)	(no attn)	(PA)	(no attn)	(PA)
1	S1: The badger is burrowing a hole. S2: A hole is being burrowed by the badger.	4.9	2.60	3.02	3.52	4.34	3.41	4.63
2	S1: There is no man screaming. S2: A man is screaming.	4	3.44	3.20	3.65	3.50	3.51	2.15

significantly outperforms its counterpart without attention. The same model can also be deployed on sequences (degenerated trees) and achieve quite impressive results.

Second, the performance gap between the two attention models is quite striking, in the sense that the progressive model completely dominate its decomposable counterpart. The difference between the two models is the *pacing* of attention, i.e., when to refer to nodes in the other tree while encoding a node in the current tree. The progressive attention model garners it's empirical superiority by *attending while encoding*, instead of waiting till the end of the structural encoding to establish cross-sentence attention. In retrospect, this may justify why the original decomposable attention model in (Parikh et al., 2016) achieved competitive results without any LSTM-type encoding. Effectively, they implemented a naive version of our progressive attention model.

Third, do structures matter/help? The overall trend in our results is quite clear: the tree-based models exhibit convincing empirical strength; linguistically motivated structures are valuable. Admittedly though, on the relatively large Quora dataset, we observe some diminishing returns of incorporating structural information. It is not counter-intuitive that the sheer size of data can possibly allow structural patterns to emerge, hence lessen the need to explicitly model syntactic structures in neural architectures.

Last but not least, in trying to assess the impact of attention mechanisms (in particular the progressive attention model), we notice that the extra mileage gained on different structural encodings is different. Specifically, performance lift on Linear Bi-LSTM $>$ performance lift on Constituency Tree-LSTM, and PA struggles to see performance lift on dependency Tree-LSTM. Interestingly enough, this observation is echoed by an earlier study (Gildea, 2004), which showed that tree-based alignment models work better on constituency trees than on dependency trees.

In summary, our results and findings lead to several intriguing questions and conjectures, which call for investigation beyond the scope of our study:

- Is it reasonable to conceptualize attention mechanisms as an implicit form of structure, which complements the representation power of explicit syntactic structures?
- If yes, does there exist some trade-off between the modeling efforts invested into syntactic and attention structures respectively, which seemingly reveals itself in our empirical results?
- The marginal impact of attention on dependency Tree-LSTMs suggests some form of saturation effect. Does that indicate a closer affinity between dependency structures (relative to constituency structures) and compositional semantics (Liang et al., 2013)?
- If yes, *why* is dependency structure a better stepping stone for compositional semantics? Is it due to the strongly lexicalized nature of the grammar? Or is it because the dependency relations (grammatical functions) embody more semantic information?

8 Conclusion

In conclusion, we proposed a novel *progressive attention* model on syntactic structures, and demonstrated its superior performance in semantic relatedness tasks. Our work also provides empirical ingredients for potentially profound questions and debates on syntactic structures in linguistics.

References

Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 385–393. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *ICLR'2015*.

John Bauer. Shift-reduce constituency parser.

Johannes Bjerva, Johan Bos, Rob Van der Goot, and Malvina Nissim. 2014. The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 642–646.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

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)*, pages 740–750.

Daniel Gildea. 2004. Dependencies vs. constituents for tree-based alignment. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2017. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*.

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

Csernai Iyer, Dandekar. First quora dataset release: Question pairs.

Sergio Jimenez, George Duenas, Julia Baquero, and Alexander Gelbukh. 2014. Unal-nlp: Combining soft cardinality features for semantic textual similarity, relatedness and entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 732–742.

Kaggle. 2017. Quora question pairs.

Alice Lai and Julia Hockenmaier. 2014. Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334.

Percy Liang, Michael I. Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Comput. Linguist.*, 39(2):389–446.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, Roberto Zamparelli, et al. 2014. A sick cure for the evaluation of compositional distributional semantic models. In *LREC*, pages 216–223.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

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.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 271–279. Association for Computational Linguistics.

Richard Socher, Eric H Huang, Jeffrey Pennington, Andrew Y Ng, and Christopher D Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*, volume 24, pages 801–809.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 523–530. Association for Computational Linguistics.

Jiang Zhao, Tiantian Zhu, and Man Lan. 2014. Ecnu: One stone two birds: Ensemble of heterogenous measures for semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 271–277.