Recurrent versus Recursive Approaches Towards Compositionality in Semantic Vector Spaces

Aran Nayebi
anayebi@stanford.edu

Heather Blundell
hrblun@stanford.edu

Abstract

Semantic vector spaces have long been useful for representing word tokens; however, they cannot express the meaning of longer phrases without some notion of compositionality. Recursive neural models explicitly encode syntactic properties that combine word representations into phrases; whereas recurrent neural models attain compositionality by processing word representations sequentially. A natural question that arises is whether recursive models are strictly necessary for attaining meaningful compositionality or are recurrent models sufficient? In this paper, we demonstrate that for the task of fine-grained sentiment analysis, recurrent models augmented with neural attention can outperform a recursive model. Specifically, we introduce a new type of recurrent attention mechanism that allows us to achieve 47.4% accuracy for the root-level sentiment analysis task on the Stanford Sentiment Treebank, which outperforms the Recursive Neural Tensor Network's (RNTN) previous 45.7% accuracy on the same dataset.

1 Introduction

Semantic vector spaces for single words have been a common choice for representing word tokens, usually obtained by deep learning based methods from a large-scale data corpus (e.g. (Mikolov et al., 2013) and (Turney and Pantel, 2010)). For NLP tasks where the inputs are longer (such as phrases, sentences, or documents), compositionality in semantic vector spaces is essential in order to combine word tokens into a vector with fixed dimensionality to be used as a feature for other tasks. The models used for achieving compositionality are usually either **recurrent** or **recursive** (Li et al., 2015).

Recurrent models effectively deal with time-series data, and in the context of NLP, can be used to model a sentence as tokens to be processed sequentially, wherein each token is combined with the embeddings of the tokens that preceded it (Elman, 1990). For instance, the most basic recurrent neural network has at each time step t its hidden state h_t to be a point-wise nonlinear function of the input vector x_t that the network receives and its previous hidden state h_{t-1}, typically using the hyperbolic tangent function:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b). \qquad (1)$$

It is important to note that recurrent models usually do not consider additional linguistic structure besides word order. On the other hand, recursive models exploit the structure of syntactic parse trees to combine tokens in a bottom-up fashion starting from the leaves of a parse tree to the root. An advantage of recursive models is that they can capture long-distance dependencies between tokens that may otherwise be syntactically nearby.

However, it is an *open question* whether this "recursive advantage" is really an advantage, and if so, for which tasks. Despite their advantages, an immediate disadvantage of these recursive models is that parsing is domain-dependent and error-prone (not to mention slow), whereas recurrent models do not face this issue.

Recent work by (Li et al., 2015) has demonstrated a largely negative answer to this question, and for most tasks, the gap in performance between recurrent and recursive models is not drastic, leaving plausible room for recurrent models to improve. Motivated by these results, our aim is to construct a sequential recurrent model that attains *improved* performance over a recursive model on a given task. The task we will focus on is the fine-grained sentence-level (root) **sentiment analysis** task, and the recursive model we aim to out-

perform is the Recursive Neural Tensor Network (RNTN), first introduced by (Socher et al., 2013) for the task of sentiment analysis.

Somewhat in parallel, the concept of neural attention has gained recent popularity. Typically, the application of attention mechanisms in NLP has been used in the task of neural machine translation, where recurrent models learn alignments between different modalities to jointly translate and align words (Bahdanau et al., 2015).

Since our task is sentiment analysis, we plan on adapting this approach in order provide the recurrent network with the ability to learn to selectively focus on parts of the sentence that may be useful for predicting the output sentiment. In particular, we will augment a Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) with neural attention, whereby we will introduce a new recurrent attention model inspired by the very recent introduction of Deep Attention Fusion (Cheng et al., 2016).

2 Prior Work

In (Socher et al., 2011), a Recursive Neural Network (RNN) architecture is introduced in order to combine the deep learning representations of image segments or natural language words, resulting in their method outperforming state-of-the-art approaches (at the time) in syntactic parsing, image segmentation, and scene classification. For the application of their RNN to syntactic parsing of natural language sentences, each word was first mapped into a vector representation using a word embedding matrix $L \in \mathbb{R}^{n \times |V|}$ which captures co-occurrence statistics, where $|V|$ is the size of the vocabulary and n is the dimensionality of the semantic space. The inputs were then mapped into a semantic space to be used by the RNN, where the essential idea is: given the two vector representations of the children, denoted by $c_1 \in \mathbb{R}^d$ and $c_2 \in \mathbb{R}^d$, we compute the vector representation of the parent p as:

$$p = f\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\right), \tag{2}$$

where f is an element-wise nonlinearity (such as a sigmoid function or a tanh function), $W \in \mathbb{R}^{d \times 2d}$ is the parameter to be learned, and the input vectors c_1 and c_2 have been concatenated. Note that the parent vector p must be of the *same dimension* as each of its children to be recursively compatible. Moreover, each parent vector p is given the same softmax classifier $y = \text{softmax}(Vp)$, (where $V \in \mathbb{R}^{n \times d}$ with n denoting the number of class labels) to compute its label probabilities.

With this understanding of a basic RNN in mind, we now examine (Socher et al., 2013). Here, a more sophisticated RNN architecture is introduced, known as the Recursive Neural Tensor Network (RNTN). The motivation behind the RNTN is that in the standard RNN, the input vectors only interact through the nonlinearity f as in equation (2); instead, a more direct interaction among the input vectors might be more desirable. Briefly, the RNTN architecture is as follows: Given the two vector representations of the children, denoted by $c_1 \in \mathbb{R}^d$ and $c_2 \in \mathbb{R}^d$, we compute the vector representation of the parent p as:

$$p = f\left(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}\right), \tag{3}$$

where $V^{[1:d]} \in \mathbb{R}^{2d \times 2d \times d}$ is a tensor, and W is as defined in equation (2). Observe that the RNN is a special case of the RNTN when V is set to 0. The authors claim that the RNTN model is advantageous because it can directly relate the input vectors since each slice $V[i] \in \mathbb{R}^{2d \times 2d}$ (for $i = 1, \ldots, d$) of the tensor is essentially performing a composition. In fact, when it is compared to RNNs (as well as to Naive Bayes, bi-gram Naive Bayes, and SVM), the RNTN obtains the highest performance with **45.7% test-set accuracy** on the Stanford Sentiment Treebank when predicting fine-grained sentiment for just the root node.

The question that naturally arises is whether recursive structure is *necessary* for improved performance. Recent work by (Li et al., 2015) tries to better understand when and why recursive models can outperform simpler models. Their approach is to benchmark recursive neural models against recurrent models (specifically, LSTM and simple recurrent models) on four tasks: discourse parsing; semantic relation extraction; sentiment classification at the sentence level and phrase level; and matching questions to answer phrases. They found that for semantic relation extraction where long-distance semantic dependencies play a role, recursive models offer a significant advantage. However, they also find that one way to improve sequential recurrent models to achieve equivalent (or close to equivalent) performance would be to break long sentences on punctuation into a se-

ries of clause-like units, work on these clauses separately, and then finally join these clauses together. This suggests that one of the reasons tree-structured models help is by breaking down long sentences into manageable units. Moreover, it appears that there is potential for a recurrent model to learn these semantic dependencies from the data, indicating that the explicit recursive structure is unnecessary.

Further evidence that sequential recurrent models do have the potential to be developed to more effectively deal with recursive structure is provided by (Karpathy et al., 2016). They find that for the task of character level language modeling (where given a sequence of characters, the network is trained to predict the next character in the sequence), the memory cells of the LSTM are interpretable and can keep track of long-range dependencies including brackets, quotes, and line lengths. They also find that scaling up the number of parameters in the LSTM by a factor of 26 only provides limited gains in error rate, in turn suggesting that it may be necessary to develop new architectural improvements instead of simply adding more parameters.

3 Models and Approach

3.1 Data

We will predict the sentiment of sentences sampled from movie reviews; specifically, we use the Stanford Sentiment Treebank (Socher et al., 2013). We concentrate on the fine-grained root classification subtask over five classes (very negative, negative, neutral, positive, and very positive) with the train/dev/test split of 8544/1101/2210. The reason why we concentrate on the fine-grained subtask is that there are fewer examples for the binary subtask since neutral sentences are excluded, and to us, it would appear that compositionality plays a more important role in the fine-grained case than in the binary case.

Our cost function will be the standard cross-entropy loss:

$$CE(y, \hat{y}) = -\sum_{i=1}^{5} y_i \log(\hat{y}_i),$$

where $y \in \mathbb{R}^5$ is the one-hot label vector, and $\hat{y} \in \mathbb{R}^5$ is the predicted probability vector for all classes (which in our case represent our 5 sentiment classes).

We will now describe the four models we will use in this paper. In what follows, we will have W_* and U_* to denote weight matrices and b_* to denote biases.

3.2 LSTM

Our baseline recurrent model is the LSTM. A typical problem with the standard recurrent network as formulated in (1) is that during training, components of the gradient vector can grow or decay exponentially over long sequences (Bengio et al., 1994). As a result, this problem of exploding or vanishing gradients makes it notoriously difficult to train standard recurrent networks to learn long-distance correlations.

The LSTM, introduced by (Hochreiter and Schmidhuber, 1997), addresses this issue by using a more sophisticated activation function comprised of gating units that preserve the state over long periods of time. At each timestep t, the LSTM is a collection of vectors, consisting of an *input gate* $i_t \in [0,1]^d$, a *forget gate* $f_t \in [0,1]^d$, an *output gate* $o_t \in [0,1]^d$, a *memory cell* $c_t \in \mathbb{R}^d$, *candidate* memory cell $\tilde{c}_t \in \mathbb{R}^d$, and a *hidden state* $h_t \in \mathbb{R}^d$. The LSTM update equations are as follows:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ \tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \tanh(c_t), \end{aligned} \tag{4}$$

where $\circ$ denotes the usual Hadamard (element-wise) product.

3.3 Bidirectional LSTM

A common variant of the traditional LSTM is the Bidirectional LSTM (Graves et al., 2013), which is comprised of two LSTMs that are run in opposite directions, in order to potentially capture past and future information. At each timestep, the hidden state of the Bidirectional LSTM is the concatenation of the hidden state of the LSTM run on the input sequence and the hidden state of the LSTM run on the reverse of the input sequence.

3.4 LSTM with Global Attention

We now begin our discussion of augmenting the LSTM with a neural attention mechanism.

Our motivation for pursuing this approach was

based on the observation of (Karpathy et al., 2016) that a common error they found involved sentences that required dynamic memory. As an example, they considered the string "Jon yelled at Mary but Mary couldn't hear him". They found that if the LSTM fails to predict the characters of the first occurrence of "Mary", then it will also fail to predict the same characters of the second occurrence. However, one could imagine that a sequential recurrent network which uses some learned weighting on previously seen characters could learn that the presence of the first "Mary" would make the second occurrence more likely. Therefore, we wanted to augment a recurrent network with neural attention, whereby the model may learn that the presence of a given word can potentially make the presence of another more or less likely.

Our first use of neural attention is based on the global attentional model proposed by (Luong et al., 2015). Since we are predicting the sentiment at the end of the sentence (the last timestep, where each timestep corresponds to a token), then our approach is to compute the context vector c_T only up to the last timestep T, where c_T is defined as:

$$c_T = \sum_{t=1}^{T} a_t h_t, \tag{5}$$

where $a_t \in \mathbb{R}$ is defined to be

$$a_t = \text{softmax}(W_a h_t). \tag{6}$$

To produce our sentiment prediction $\hat{y} \in \mathbb{R}^5$, we use our context vector c_T and the hidden state at the last timestep h_T to produce the attentional hidden state $\tilde{h}_T$, defined as:

$$\tilde{h}_T = \tanh\left(W_c \begin{bmatrix} c_T \\ h_T \end{bmatrix}\right), \tag{7}$$

from which our sentiment prediction $\hat{y}$ is given by:

$$\hat{y} = \text{softmax}(W_p \tilde{h}_T). \tag{8}$$

3.5 LSTM with Recurrent Attention

Our second approach to using neural attention will be to extend the above notion of global attention, especially the production of the attentional hidden state in (7), by adding additional recurrence to the attention mechanism. Here, we gain inspiration from the recent introduction of deep attention fusion by (Cheng et al., 2016), where they replace the tanh operation in (7) by an LSTM-like attentional layer. Deep attention fusion was originally developed for neural machine translation, so we will adapt it to the setting of sentiment analysis by *not* using inter-attention (attention between the source and target sequences in the translation task) and instead primarily re-encode the sentence and adaptively transfer its representation using gating units as in the LSTM. We proceed as follows to accomplish this. Let h_t and m_t be the hidden state and the memory cell state of the LSTM at timestep t, respectively. Then, for each $t = 1, \ldots, T$, let

$$\begin{bmatrix} \tilde{h}_t \\ \tilde{m}_t \end{bmatrix} = \sum_{j=1}^{t} \alpha_j^t \begin{bmatrix} h_j \\ m_j \end{bmatrix}, \tag{9}$$

where $\alpha_j^t \in \mathbb{R}$, such that

$$\alpha_j^t = \text{softmax}\left(W_a \begin{bmatrix} h_j \\ m_j \end{bmatrix}\right).$$

Next, we adaptively transfer the representation $x_t = \begin{bmatrix} \tilde{h}_t \\ \tilde{m}_t \end{bmatrix}$ through the following gating units:

$$\begin{aligned} r_t &= \sigma\left(W_r \tilde{h}_t + b_r\right) \\ k_t &= f_t \circ k_{t-1} + i_t \circ \tilde{k}_t + r_t \circ \tilde{m}_t \\ s_t &= o_t \circ \tanh(k_t), \end{aligned} \tag{10}$$

where f_t, i_t, $\tilde{k}_t$, and o_t correspond exactly to the forget gate, input gate, candidate activation, and output gate in a traditional LSTM, defined as follows:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f s_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i s_{t-1} + b_i) \\ \tilde{k}_t &= \tanh(W_k x_t + U_k s_{t-1} + b_k) \\ o_t &= \sigma(W_o x_t + U_o s_{t-1} + b_o). \end{aligned} \tag{11}$$

The final stage of our recurrent attention mechanism involves passing s_T, where T is the last timestep, through a highway network. A highway network (Srivastava et al., 2015) is a generalization of an affine transformation that uses gating units in order to regulate information flow, defined as follows for a given input x:

$$\begin{aligned} \mathbf{H} &= H(W_H x + b_H) \\ \mathbf{T} &= T(W_T x + b_T) \\ \mathbf{C} &= C(W_C x + b_C) \\ y &= \mathbf{H} \circ \mathbf{T} + x \circ \mathbf{C}. \end{aligned}$$

where H, T, and C are non-linear activation functions, $\mathbf{T}$ is called the *transform gate* and $\mathbf{C}$ is called the *carry gate*, where usually $\mathbf{C} = 1 - \mathbf{T}$. In our implementation, our input into the highway network was s_T, and we used the following equations for our highway network:

$$\begin{aligned} \mathbf{H} &= W_H x + b_H \\ \mathbf{T} &= \sigma(W_T x + b_T) \\ \hat{s}_T &= \mathbf{H} \circ \mathbf{T} + s_T \circ (1 - \mathbf{T}). \end{aligned} \tag{12}$$

As suggested in (Srivastava et al., 2015), we initialized the transform gate bias b_T to -2 in order to bias the network initially towards *carry* behavior.

We found that using the above highway network in (12) on the last timestep s_T improved our performance. Finally, our sentiment prediction $\hat{y} \in \mathbb{R}^5$ is computed using $\hat{s}_T$, the output of the highway network, as follows:

$$\hat{y} = \text{softmax}(W_p \hat{s}_T). \tag{13}$$

4 Experiments

4.1 Hyperparameters and Training Details

We used 300-dimensional 840B Glove vectors (Pennington et al., 2014) to initialize our word representations, and we fine-tune the Glove representations during training since these vectors were not originally trained to capture sentiment[1]. Out of vocabulary words were initialized randomly using Gaussian samples with mean 0 and variance 1. We also used the Keras (Chollet, 2015) and Theano (Bastien et al., 2012) software libraries as our framework for implementing our models.

For the LSTM, the best hyperparameters we found are Dropout (Srivastava et al., 2014) of 0.2 between the Glove embedding layer and the LSTM layer (which has 150-dimensional hidden states), an L_2 regularization penalty of 0.01, and Adagrad (Duchi et al., 2011) (with the default learning rate of 0.01) as our optimizer, with a minibatch size of 128.

For the Bidirectional LSTM, the best hyperparameters we found are Dropout of 0.3 between the Glove embedding layer and each of the two LSTMs (which have 300-dimensional hidden states), an L_2 regularization penalty of 0.01, and Adagrad (with the default learning rate of 0.01) as our optimizer, with a minibatch size of 300. Additionally, for each LSTM, we have a Dropout level of 0.3 between each recurrent connection, as suggested in recent work by (Gal, 2015) that it is an effective regularizer for recurrent networks.

For the LSTM with global attention, we use a Dropout level of 0.3 between the Glove embedding layer and the LSTM layer (which has 300-dimensional hidden states), an L_2 regularization penalty of 0.01, and Adagrad (with the default learning rate of 0.01) as our optimizer, with a minibatch size of 300. Moreover, the attentional hidden state defined in (7) is 25-dimensional. Additionally, we have a Dropout level of 0.3 between each recurrent connection for the LSTM layer.

Finally, for the LSTM with recurrent attention, we use a Dropout level of 0.15 between the Glove embedding layer and the LSTM layer (which has 300-dimensional hidden states), an L_2 regularization penalty of 0.01, and a variant of Adagrad known as Adadelta (Zeiler, 2012) (with the default learning rate of 1.0) as our optimizer, with a minibatch size of 300. Additionally, we have a Dropout of 0.25 between each recurrent connection for the LSTM layer, and a Dropout level of 0.3 between each recurrent connection in the adaptive transfer layer described in (10).

In Figure 1, we have plotted the accuracy per epoch during training on both the train set and the dev set, for each of the four models. We trained our models on a Titan X GPU, and also used early stopping if the dev set accuracy did not improve after 10 epochs. To demonstrate the ease of training recurrent models versus recursive models, all of our models took 20 minutes or less to train, and started to overfit either at or before 20 epochs of training. For time comparison, we also implemented the RNTN as described in (Socher et al., 2013), and found that the RNTN was difficult to port to Theano (in order to train it on a GPU) and therefore we trained it on a laptop CPU instead, and it ultimately took around 4-5 hours to train, consistent with the training time reported in (Socher et al., 2013). Recent work by (Bowman et al., 2016) develops a shift-reduce model which supports batched computing in recursive models, though it is still much more complicated than recurrent networks.

[1] They were in fact trained on 840 billion tokens of Common Crawl data, as in `http://nlp.stanford.edu/projects/glove/`.

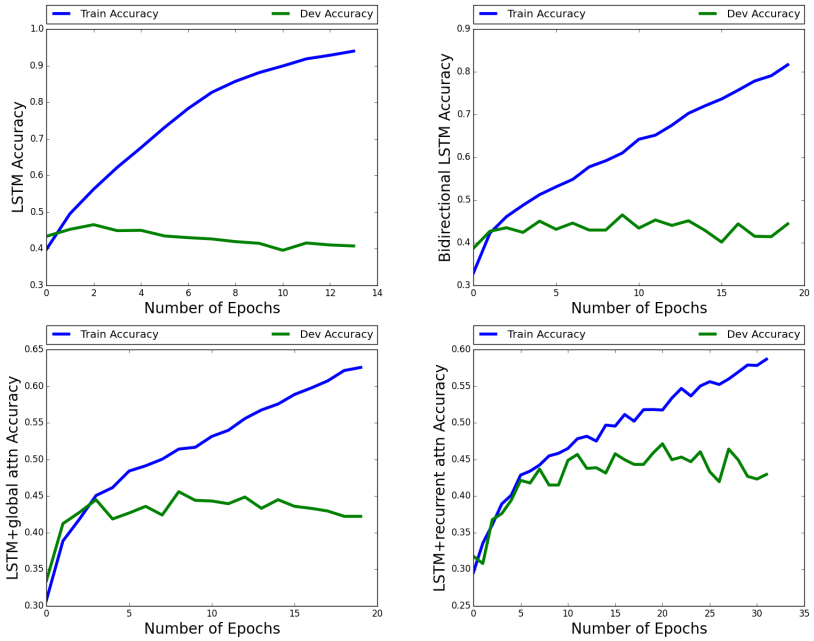

Figure 1: Accuracy per epoch on train set (blue) and dev set (green).

Model	**Test-Set Accuracy**
LSTM	45.7% (0.0113)
Bidirectional LSTM	46.4% (0.0175)
LSTM+global attn.	45.3% (0.0047)
LSTM+recurrent attn.	**47.4%** (0.0031)

Table 1: Accuracy for the fine-grained (5 class) predictions at the sentence level (root), with standard deviations in parentheses.

4.2 Results

Our results are summarized in Table 1. Although we were not able to replicate the 49.8% performance of a Bidirectional LSTM as reported in (Li et al., 2015), the *relative* improvement between our best performing model, the LSTM with recurrent attention, and the Bidirectional LSTM, appears to be statistically notable, with a two-tailed p-value of 0.04555.

Furthermore, our simplest model, the LSTM, can already achieve the same test set performance of 45.7% as the RNTN, indicating perhaps that the sophisticated gating mechanisms in the LSTM may be helpful in learning some relevant compositional structure in the data.

Interestingly, global attention does *not* appear to be useful, and in fact appears to degrade performance when compared to our baseline LSTM. One explanation could be that since we are predicting only at the last timestep, then we are only creating one context vector c_T, when in fact for many neural machine translation tasks, a context vector is created at *every* timestep and is used by the target to compute proper alignment. Thus, it appears that a single context vector that summarizes the weighting at every time point does not appear sufficient.

The recurrent attention model, on the other hand, effectively makes use of every timestep as seen in (9), and replaces the single tanh operation in (7) of the global attention model by the adaptive transfer layer in (10). We found that what provided the additional performance improvement over the Bidirectional LSTM was the inclusion of a highway network after the output of the adaptive transfer layer but prior to the final softmax layer, as previously noted in §3.5. A potential reason for the usefulness of a highway network in an attention mechanism could be that the transform gate **T** in (12) additionally allows for selectively learning portions of the adaptive transfer layer's output that are most relevant for the final sentiment prediction.

5 Analysis

To better understand the reasons behind the performance results in Table 1, we plot confusion matrices for all four models in Figure 2. It is important to note that all of the models appear to

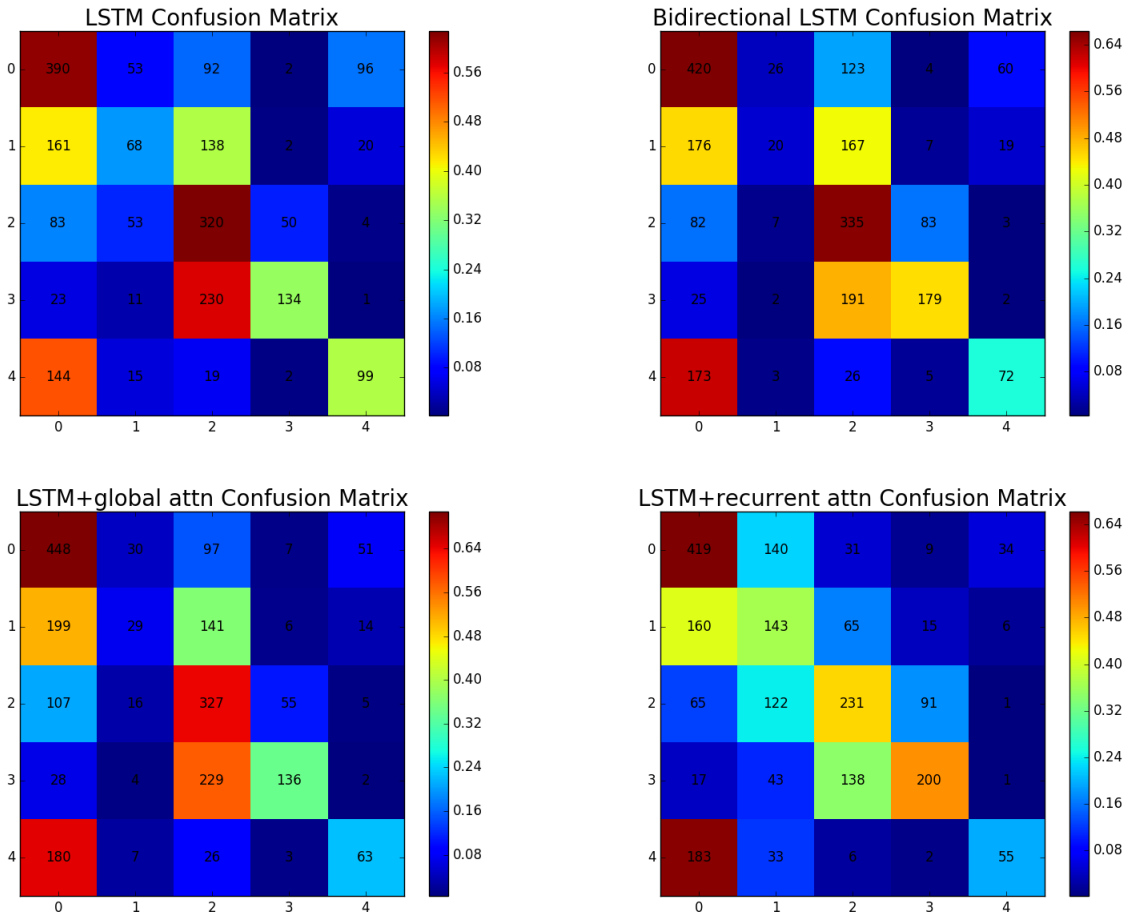

Figure 2: Confusion matrices on the test set. 0: "very negative", 1: "negative", 2: "neutral", 3: "positive", 4: "very positive".

predict the "very negative" sentiment really well, which is not entirely surprising since the Stanford Sentiment Treebank consists of movie reviews, which tend to be negative overall. However, despite this, the LSTM with recurrent attention appears to perform consistently well across the other sentiment classes, with the exception of the "very positive" class, which all the models seem to do poorly on, providing some evidence that the recurrent attention mechanism better allows the LSTM to exploit compositionality in the data. Moreover, we can see somewhat more closely the effect global attention has, which seems to help the LSTM only slightly better predict the very negative, neutral, and positive sentiment classes; however, performance on the remaining sentiment classes is noticeably worse, thereby resulting in overall decreased performance compared to the LSTM. The Bidirectional LSTM appears to predict the very negative, neutral, and positive sentiment classes better than the LSTM, and only slightly underperforms on the remaining sentiment classes, thereby resulting in overall higher performance than the LSTM.

Furthermore, all the models appear to mostly misclassify the very positive sentiment to be very negative, indicating that the models are not perfectly capturing all compositional aspects of sentiment whereby the sentiment could drastically shift. In fact, the LSTM with recurrent attention appears to confuse the two classes more than the remaining models. Another confusion among all the models is *finer grained* in that they appear to misclassify "negative" sentiment as "very negative"; however, the LSTM with recurrent attention seems to misclassify this finer grained distinction *less* than the other models.

To better understand the strengths and the weaknesses of the recurrent attention mechanism, we examined sentences that the LSTM with

Sentence	True Label
the additional storyline is interesting and entertaining, but it doesn't have the same magical quality as the beginning of the story.	negative
though everything might be literate and smart, it never took off and always seemed static.	very negative
despite what anyone believes about the goal of its makers, the show represents a spectacular piece of theater, and there's no denying the talent of the creative forces behind it.	positive

Table 2: Sentences that the LSTM with recurrent attention *classifies correctly* but the remaining models *misclassify*.

Sentence	True Label	Pred Label
a strong first quarter, slightly less so second quarter, and average second half.	negative	very negative
a journey through memory, a celebration of living, and a sobering rumination on fatality, classism, and ignorance.	positive	neutral
the editing is chaotic, the photography grainy and badly focused, the writing unintentionally hilarious, the direction unfocused, the performances as wooden.	very negative	very positive

Table 3: Sentences that the LSTM with recurrent attention *misclassifies* but the remaining models *classify correctly*. "Pred label" refers to the LSTM with recurrent attention's sentiment prediction.

recurrent attention classifies correctly but every other model misclassifies the sentiment (Table 2), and similarly, the sentences that the LSTM with recurrent attention misclassifies but every other model classifies correctly (Table 3).

As can be seen from Table 2, the LSTM with recurrent attention is able to capture contrastive conjunction, which are sentences of the form "X but Y" (as in the first sentence) as well as sentiment shifts caused by words like "despite" and "though" in both positive and negative sentences, which is the case in the last two sentences. On the other hand, as can be seen from Table 3, the LSTM with recurrent attention appears to perform poorly compared to the other models when the sentence consists entirely of a list of phrases (e.g. "X, Y, Z"). One reason could be that if a list of phrases are separated by commas, then it is difficult to relate them with attention and selectively focus on the ones that seem to be most prominent for determining the output sentiment than if we had a contrastive conjunction or a sentiment shifter to focus on.

6 Conclusions

We demonstrated a new type of recurrent attention model and demonstrated that LSTMs with recurrent attention can outperform the RNTN, a recursive model, on the task of fine-grained sentiment analysis. The motivation behind our recurrent attention model is that it provides a way of adaptively incorporating the LSTM's response at every timestep, providing a viable alternative to global attention, which we found was not as useful for this task. Moreover, our experiments demonstrated that recurrent attention is able to capture effective changes in sentiment that occur within a sentence. On the other hand, lists of phrases appear to be a noticeable shortcoming of this model.

Future work could involve potentially using our recurrent attention model with a modified LSTM that has its own *internal* attention mechanism, such as the LSTMN (Long Short-Term Memory Network) recently proposed by (Cheng et al., 2016), as a possible solution to overcoming the current shortcomings with lists of phrases. It would also be interesting to apply our proposed recurrent attention mechanism to the Stanford NLI corpus (Bowman et al., 2015), which consists of a collection of English sentence pairs manually labeled for entailment, contradiction, and neutral, supporting natural language inference, where compositionality may play a larger role.

Acknowledgments

We would like to thank Jiwei Li for his helpful advice and comments.

References

Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. 2015. "Neural Machine Translation by Jointly Learning to Align and Translate". *International Conference on Learning Representations* 2015.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. 2012. "Theano: new features and speed improvements". Presented at *Advances in Neural Information Processing Systems (NIPS) Workshop 2012*.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. "Learning long-term dependencies with gradient descent is difficult". *IEEE Transactions on Neural Networks*, 5(2):157-166.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. "Long short-term memory-networks for machine reading". arXiv:1601.06733.

François Chollet. 2015. "Keras: Deep Learning library for Theano and TensorFlow". `https://github.com/fchollet/keras/`.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. "A large annotated corpus for learning natural language inference". In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. "A Fast Unified Model for Parsing and Sentence Understanding". arXiv:1603.06021.

John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". 2011. *The Journal of Machine Learning Research*, 12: 2121-2159.

Jeffrey L. Elman. 1990. "Finding structure in time". *Cognitive science*, 14: 179-211.

Alex Graves, Navdeep Jaitly, and A-R Mohamed. 2013. "Hybrid speech recognition with deep bidirectional LSTM". In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*: 273-278.

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

Yarin Gal. 2015. "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". arXiv:1512.05287.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. "Visualizing and understanding recurrent neural networks". *International Conference on Learning Representations 2016*.

Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eduard Hovy. 2015. "When Are Tree Structures Necessary for Deep Learning of Representations?". In *Proceedings of the 2015 Conference on Empirical Methods on Natural Language Processing*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. "Effective Approaches to Attention-based Neural Machine Translation". In *Proceedings of the 2015 Conference on Empirical Methods on Natural Language Processing*.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. "Linguistic regularities in continuous space word representations". In *NAACL HLT*: 746-751.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation". In *Proceedings of the 2014 Conference on Empirical Methods on Natural Language Processing*. `http://nlp.stanford.edu/projects/glove/`.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. "Parsing Natural Scenes and Natural Language with Recursive Neural Networks". *International Conference on Machine Learning 2011*.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". *International Conference on Machine Learning 2013*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In *Journal of Machine Learning Research*, 15: 1929-1958.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. "Training Very Deep Networks". In *Advances in Neural Information Processing Systems (NIPS) 2015*.

Peter D. Turney and Patrick Pantel. 2010. "From frequency to meaning: Vector space models of semantics". *Journal of Artificial Intelligence Research* 37: 141-188.

Matthew D. Zeiler. 2012. "ADADELTA: An adaptive learning rate method". arXiv:1212.5701.