Visualizing and Understanding Neural Models in NLP

Jiwei Li[1], **Xinlei Chen**[2], **Eduard Hovy**[2] **and Dan Jurafsky**[1]
[1]Computer Science Department, Stanford University, Stanford, CA 94305, USA
[2]Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{jiweil,jurafsky}@stanford.edu {xinleic,ehovy}@andrew.cmu.edu

Abstract

While neural networks have been successfully applied to many NLP tasks the resulting vector-based models are very difficult to interpret. For example it's not clear how they achieve *compositionality*, building sentence meaning from the meanings of words and phrases. In this paper we describe four strategies for visualizing compositionality in neural models for NLP, inspired by similar work in computer vision. We first plot unit values to visualize compositionality of negation, intensification, and concessive clauses, allow us to see well-known markedness asymmetries in negation. We then introduce three methods for visualizing a unit's *salience*, the amount it contributes to the final composed meaning: (1) gradient back-propagation, (2) the variance of a token from the average word node, (3) LSTM-style gates that measure information flow. We test our methods on sentiment using simple recurrent nets and LSTMs. Our general-purpose methods may have wide applications for understanding compositionality and other semantic properties of deep networks , and also shed light on why LSTMs outperform simple recurrent nets,

1 Introduction

Neural models match or outperform the performance of other state-of-the-art systems on a variety of NLP tasks. Yet unlike traditional feature-based classifiers that assign and optimize weights to varieties of human interpretable features (parts-of-speech, named entities, word shapes, syntactic parse features) the behavior of deep learning models is much less easily interpreted. Deep learning models mainly operate on word embeddings (low-dimensional, continuous, real-valued vectors) through multi-layer neural architectures, each layer of which is characterized as an array of hidden neuron units. It is unclear how deep learning models deal with *composition*, implementing functions like negation or intensification, or combining meaning from different parts of the sentence, filtering away the informational chaff from the wheat, to build sentence meaning

In this paper, we explore multiple strategies to interpret meaning composition in neural models. We employ traditional methods like representation plotting, and introduce 3 simple strategies for measuring how much a neural unit contributes to meaning composition, its 'salience' or importance: (1) using first derivatives (2) computing how much a unit differs from the average unit for a sentence, and (3) introducing a LSTM-inspired model that uses gates to measure the flow of information from unit to unit.

To simplify our explanation of the models, we focus on phrase-based sentiment analysis from the Stanford Sentiment Treebank dataset (Socher et al., 2013). Sentiment analysis uses a relatively unidirectional aspect of meaning that is much simpler than for general semantic parsing or understanding, making visualizing much simpler. However, that means our findings may not extend to other tasks. Our methods and results should therefore be considered preliminary ones subject to replication on other domains.

Nonetheless, though our attempts only touch superficial points in neural models, and each method has its pros and cons, together they may offer some insights into the behaviors of neural models in language based tasks, marking one initial step toward understanding how they achieve meaning composition in natural language processing.

The next section describes some visualization models that have inspired this work. In section 3 we describe the dataset and task. Different visualization strategies are presented in Sections 4, 5, 6, 7, followed by a brief conclusion.

2 A Brief Review of Neural Visualization

Similarity is commonly visualized graphically, generally by projecting the embedding space into two dimensions and observing that similar words tend to be clustered together (e.g., Elman (1989), Ji and Eisenstein (2014), Faruqui and Dyer (2014)). But methods for interpreting and visualizing neural models have been much more significantly explored in vision, especially for Convolutional Neural Networks (CNNs or ConvNets) (Krizhevsky et al., 2012), multi-layer neural networks in which the original matrix of image pixels is convolved and pooled as it is passed on to hidden layers. ConvNet visualizing techniques consist mainly in mapping the different layers of the network (or other features like SIFT (Lowe, 2004) and HOG (Dalal and Triggs, 2005)) back to the initial image input, thus capturing the human-interpretable information they represent in the input, and how units in these layers contribute to any final decisions (Simonyan et al., 2013; Mahendran and Vedaldi, 2014; Nguyen et al., 2014; Szegedy et al., 2013; Girshick et al., 2014; Zeiler and Fergus, 2014). Such methods include:

(1) Inversion: Inverting the representations by training an additional model to project outputs from different neural levels back to the initial input images (Mahendran and Vedaldi, 2014; Vondrick et al., 2013; Weinzaepfel et al., 2011). The intuition behind reconstruction is that the pixels that are reconstructable from the current representations are the content of the representation. The inverting algorithms allow the current representation to align with corresponding parts of the original images.

(2) Back-propagation (Erhan et al., 2009; Simonyan et al., 2013) and Deconvolutional Networks (Zeiler and Fergus, 2014): Errors are back propagated from output layers to each intermediate layer and finally to the original image inputs. Deconvolutional Networks work in a similar way by projecting outputs back to initial inputs layer by layer, each layer associated with one supervised model for projecting upper ones to lower ones These strategies make it possible to spot active regions or ones that contribute the most to the final classification decision.

(3) Generation: This group of work generates images in a specific class from a sketch guided by already trained neural models (Szegedy et al., 2013; Nguyen et al., 2014). Models begin with an image whose pixels are randomly initialized and mutated at each step. The specific layers that are activated at different stages of image construction can help in interpretation.

While the above strategies inspire the work we present in this paper, there are fundamental differences between vision and NLP. In NLP words function as basic units, and hence (word) vectors rather than single pixels are the basic units. Sequences of words (e.g., phrases and sentences) are also presented in a more structured way than arrangements of pixels.

3 Dataset and Models to be Visualized

We chose the Stanford Sentiment Treebank dataset (Socher et al., 2013), a benchmark dataset widely used for neural model evaluations. The dataset contains gold-standard sentiment labels for every parse tree constituent, from sentences to phrases to individual words, for 215,154 phrases in 11,855 sentences. The task is to perform both fine-grained (very positive, positive, neutral, negative and very negative) and coarse-grained (positive vs negative) classification at both the phrase and sentence level. For more details about the dataset, please refer to (Socher et al., 2013).

While many studies on this dataset use recursive parse-tree models, in this work we employ only standard sequence models (RNNs and LSTMs) since these are the most widely used current neural models, and sequential visualization is more straightforward. We therefore first transform each parse tree node to a sequence of tokens. The sequence is first mapped to a phrase/sentence representation and fed into a softmax classifier. Phrase/sentence representations are built with the following three models:

1. Standard recurrent sequence models with TANH activation functions
2. LSTMs.
3. Bidirectional LSTMs

For details about the three models, please refer to Appendix.

Training AdaGrad with mini-batch was used for training, with parameters ($L2$ penalty, learning rate, mini batch size) tuned on the development set. The number of iterations is treated as a variable to tune and parameters are harvested based on the best performance on the dev set. The number of dimensions for the word and hidden layer are set to 60 with 0.1 dropout rate. Parameters are tuned on the dev set.

The standard recurrent model achieves 0.429 (fine grained) and 0.850 (coarse grained) accuracy at the sentence level; LSTM achieves 0.469 and 0.870, and Bidirectional LSTM 0.488 and 0.878, respectively.

4 Representation Plotting

Local Composition We begin with simple plots of representations to shed light on local compositions, showing a 60d heat-map vector for the representation of selected phrases/sentences. Figure 1 illustrates extent modifications (adverbial and adjectival) and negation.

The intensification part of Figure 1 shows suggestive patterns where values for a few dimensions are strengthened by modifiers like "a lot" (the red bar in the first example) "so much" (the red bar in the second example), and "incredibly". Though the patterns for negations are not as clear, there is still a consistent reversal for some dimensions, visible as a shift between blue and red for dimensions boxed on the left.

We then visualize words and phrases using t-sne (Van der Maaten and Hinton, 2008) in Figure 2, deliberately adding in some random words for comparative purposes. As can be seen, neural models nicely learn the properties of local compositionally, clustering negation+positive words ('not nice', 'not good') together with negative words. Note also the asymmetry of negation: "not bad" is clustered more with the negative than the positive words (as shown both in Figure 1 and 2). This asymmetry has been widely discussed in linguistics, for example as arising from markedness, since 'good' is the unmarked direction of the scale (3; Horn, 1989; Fraenkel and Schul, 2008). This suggests that although the model does seem to focus on certain units for negation in Figure 1, the neural model is not just learning to apply a fixed transform for 'not' but is able to capture the subtle differences in the composition of different words.

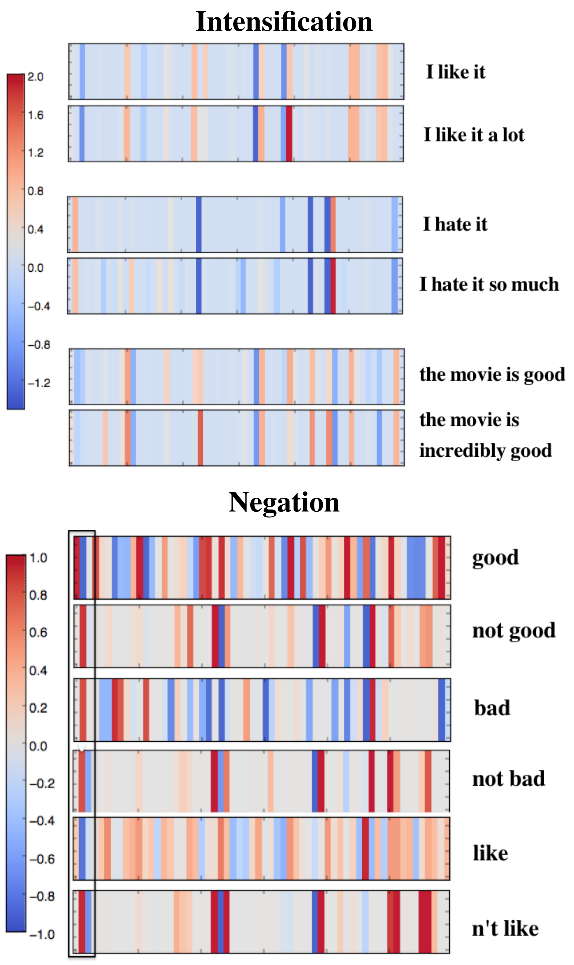

Figure 1: Visualizing intensification and negation. Each vertical bar shows the value of one dimension in the final sentence/phrase representation.

Concessive Sentences In concessive sentences, two clauses have opposite polarities, usually related by a contrary-to-expectation implicature. We plot evolving representations over time for two concessives in Figure 3. The plots suggest:

1. For tasks like sentiment analysis whose goal is to predict a specific semantic dimension (as opposed to general tasks like language model word prediction), too large a dimensionality leads to many dimensions being abandoned, causing two sentences of opposite sentiment to differ only in a few dimensions. This may explain why more dimensions don't necessarily lead to better performance on such tasks.

2. LSTMs behave differently than standard recurrent models on these abandoned dimensions. LSTMs assign 0 values to many dimensions, caused by the 0 value of the FORGET gates. Assigning a 0 value to abandoned dimensions makes

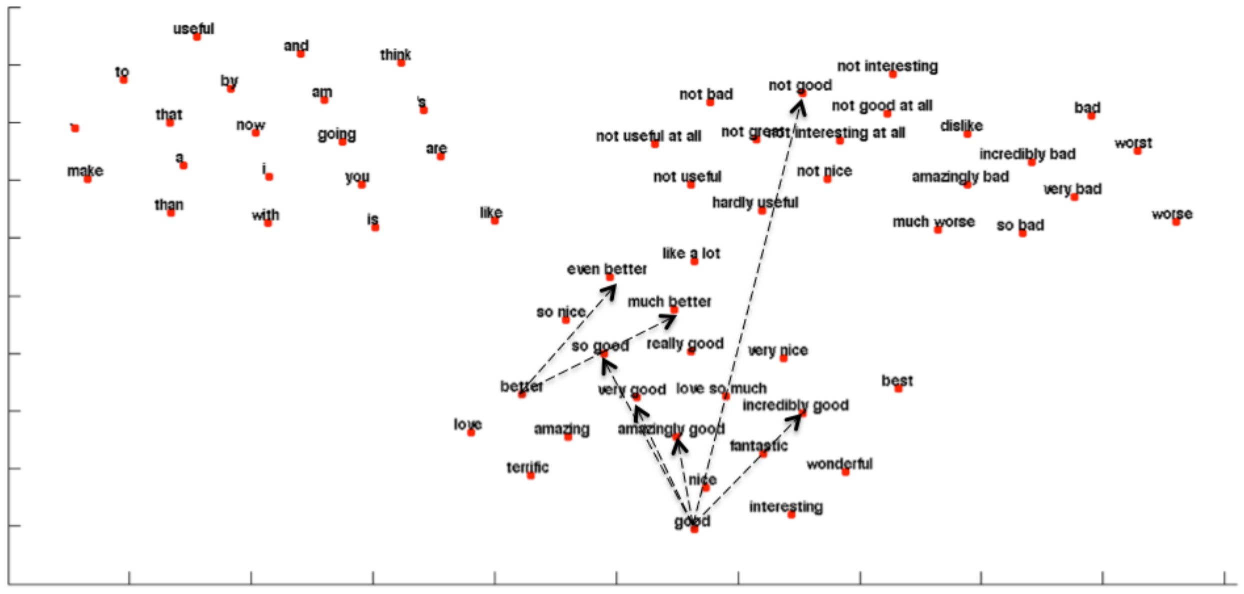

Figure 2: t-SNE Visualization on latent representations for modifications and negations.

them exert no further impact on later calculations. Recurrent models, by contrast, assign very large values to useless dimensions[1]. One possible explanation of the behavior of recurrent models is that since they contain far fewer convolutional parameters than LSTMs, they lack flexibility in controlling information flow. To deal with that, recurrent models use abandoned dimensions for buffering, but however large the original values are, numbers after tanh operation would always fall into [-1, 1]. This enables models to focus on useful dimensions without worrying too much about other dimensions. This is just a speculation, and would require further investigation.

3. Both sentences contain two clauses connected by the conjunction "though". Such two-clause sentences might work *collaboratively*—models would remember the word "though" and make the second clause share the same sentiment orientation as first—or *competitively*, with the stronger one dominating. The region within dotted line in Figure 3(a) favors the second assumption: the difference between the two sentences is diluted when the final words ("interesting" and "boring") appear.

Clause Composition In Figure 4 we explore this clause composition in more detail. Representations move closer to the negative sentiment region by adding negative clauses like "although it had bad acting" or "but it is too long" to the end of a simply positive "I like the movie". By contrast, adding a concessive clause to a negative clause does not move toward the positive; "I hate X but ..." is still very negative, not that different than "I hate X". This difference again suggests the model is able to capture negative asymmetry (3; Horn, 1989; Fraenkel and Schul, 2008).

5 Model 1: First Derivatives

Our first new model, inspired by the back-propagation strategy in vision (Erhan et al., 2009; Simonyan et al., 2013), measures how much each input unit contributes to the final decision, which can be approximated by first derivatives.

More formally, for a classification model, an input E is associated with a gold-standard class label c. (Depending on the NLP task, an input could be the embedding for a word or a sequence of words, while labels could be POS tags, sentiment labels, etc.) Given embeddings E for input words with the associated gold class label c, the trained model associates the pair (E, c) with a score $S_c(E)$. The goal is to decide which units of E make the most significant contribution to $S_c(e)$, and thus the decision, the choice of class label c.

In the case of deep neural models, the class score $S_c(e)$ is a highly non-linear function. We approximate $S_c(e)$ with a linear function of e by

[1]We adopt tanh as activation functions. Most output dimensions take the value close to 1 or -1, which means values before tanh activations are very large positive or negative values given $tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$.

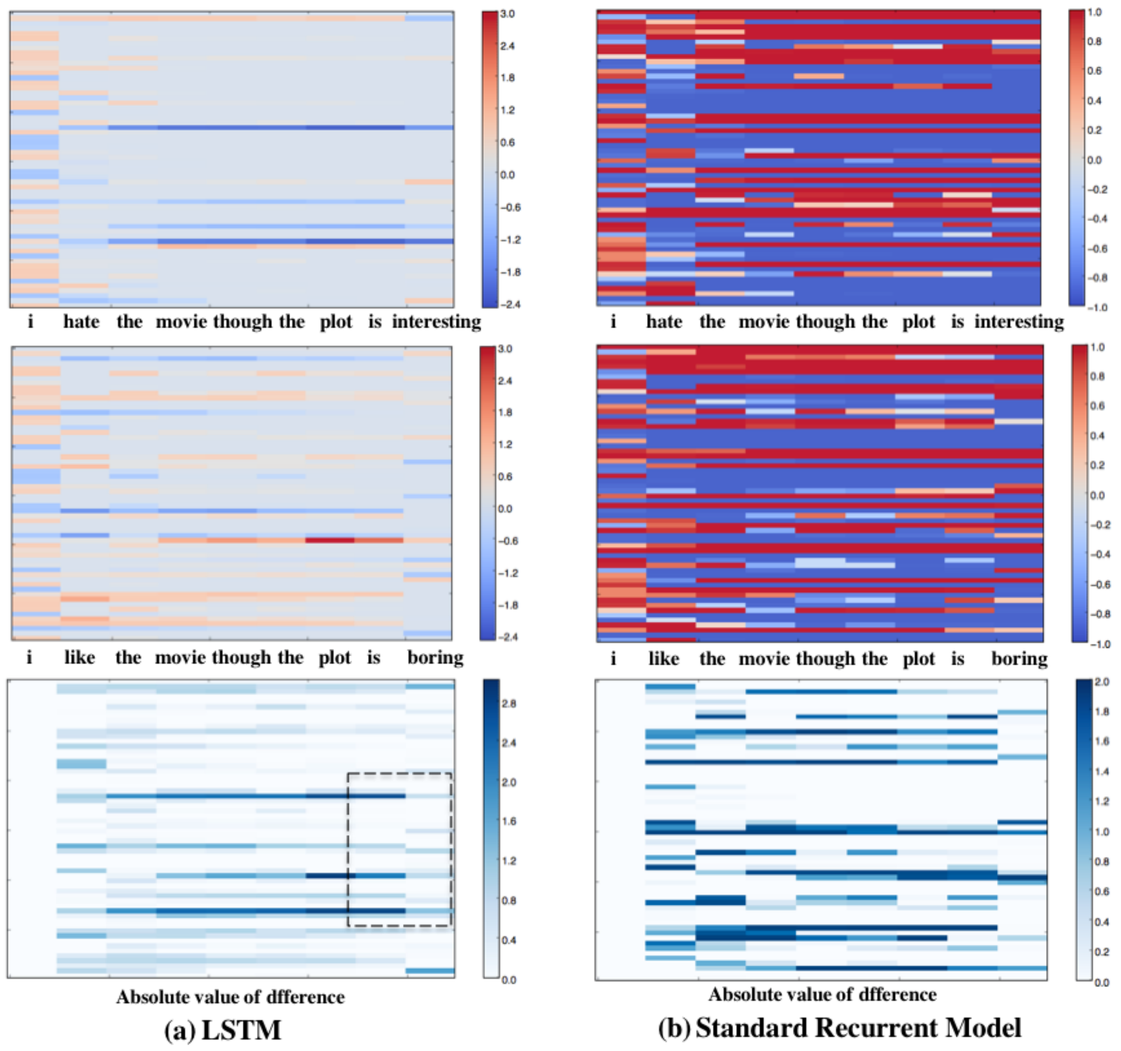

Figure 3: Representations over time from LSTMs and Recurrent Models. Each grid corresponds to outputs from LSTM/Recurrent models at each time-step. The last rows correspond to absolute differences for each time step between two sequences. To note, due to the usage of tanh activation function for recurrent models, the largest (smallest) value that vectors can take is +1 (-1).

computing the first-order Taylor expansion

$$S_c(e) \approx w(e)^T e + b \tag{1}$$

where $w(e)$ is the derivative of S_c with respect to the embedding e.

$$w(e) = \frac{\partial(S_c)}{\partial e} |_e \tag{2}$$

The magnitude (absolute value) of the derivative indicates the sensitiveness of the final decision to the change in one particular dimension, telling us how much one specific dimension of the word embedding contributes to the final decision. The saliency score is given by

$$S(e) = |w(e)| \tag{3}$$

Visualization We plot in Figures 5, 6 and 7 the saliency scores (the absolute value of the derivative of the loss function with respect to each dimension of all word inputs) for three sentences, applying the trained model to each sentence. The examples are based on the clear sentiment indicator “hate” that lends them all negative sentiment.

“I hate the movie” All three models assign high saliency to “hate” and dampen the influence of other tokens. LSTM offers a clearer focus on “hate” than the standard recurrent model, but the bi-directional LSTM shows the clearest focus, attaching almost zero emphasis on words other than “hate”. This is presumably due to the gates structures in LSTMs and Bi-LSTMs that controls information flow, making these architectures better

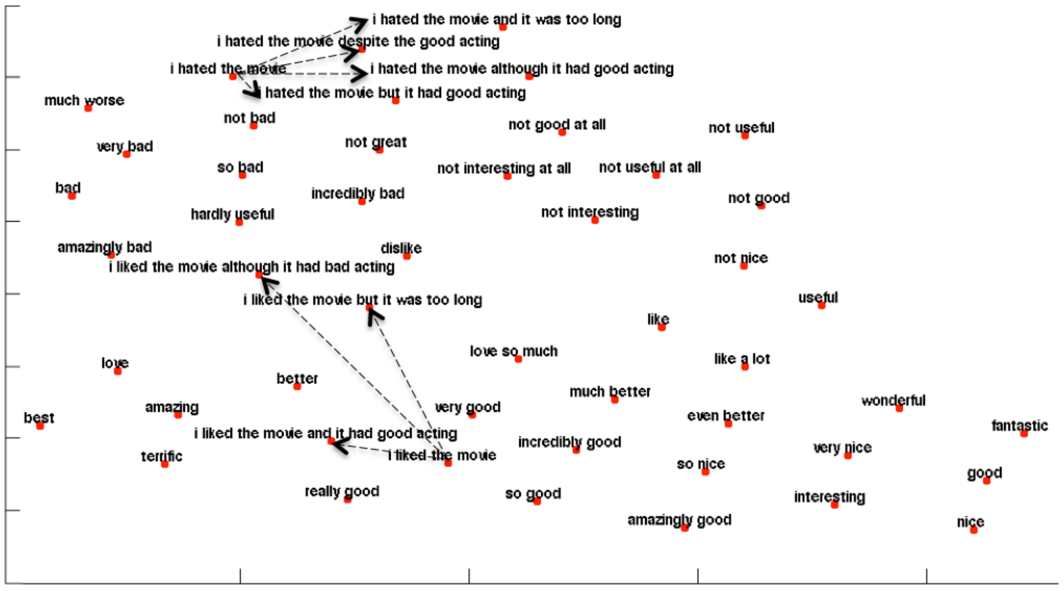

Figure 4: t-SNE Visualization for clause composition.

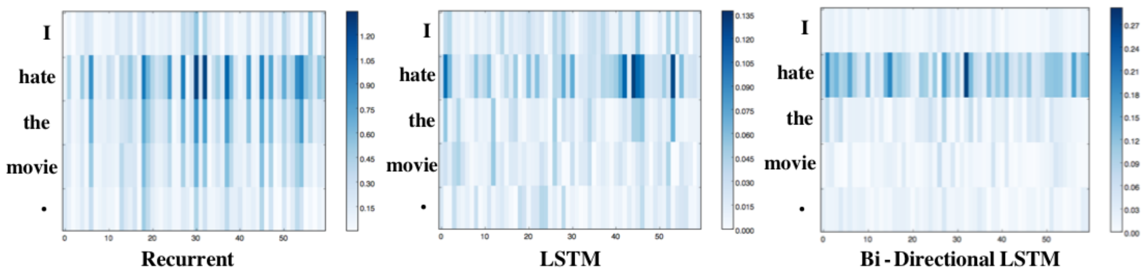

Figure 5: Visualization for “I hate the movie .” .

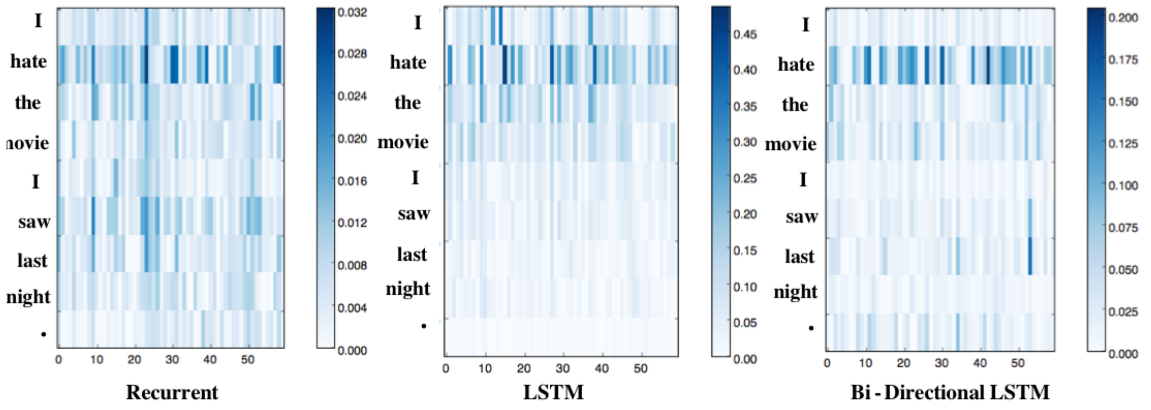

Figure 6: Visualization for “I hate the movie I saw last night .” .

at filtering out less relevant information.

“I hate the movie that I saw last night” All three models assign the correct sentiment. The simple recurrent models again do poorly at filtering out irrelevant information, assigning too much salience to words unrelated to sentiment. However none of the models suffer from the gradient vanishing problems despite this sentence being longer; the salience of “hate” still stands out after 7-8 following convolutional operations.

“I hate the movie though the plot is interesting” The simple recurrent model emphasizes only the second clause “the plot is interesting”, assigning no credit to the first clause “I hate the movie”.

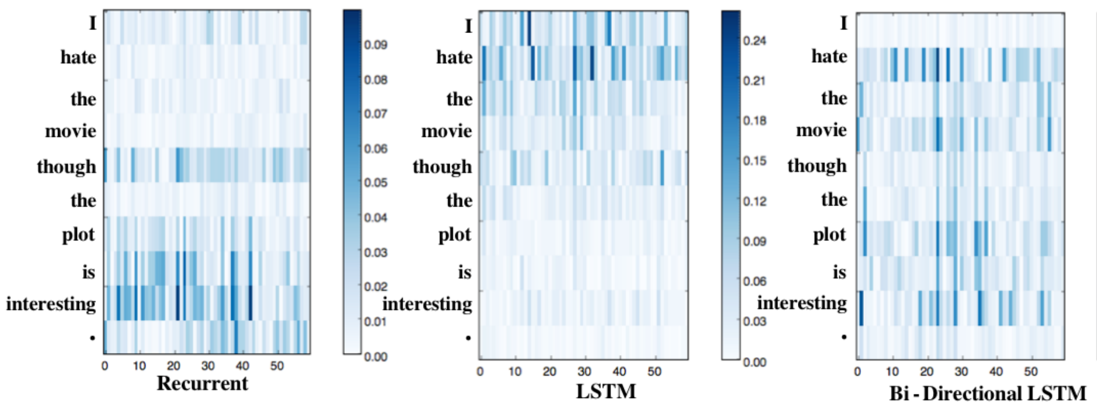

Figure 7: “I hate the movie though the plot is interesting .” .

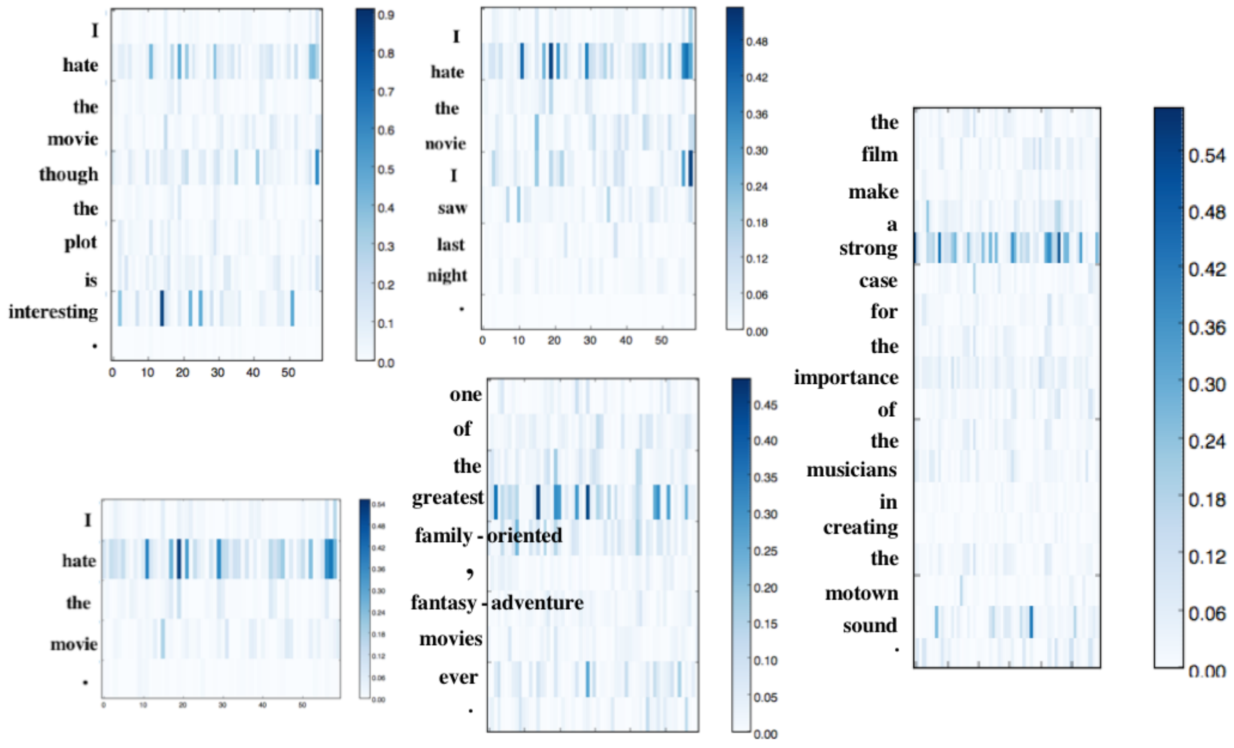

Figure 8: Variance visualization. As with the last example, the model apparently mistakes the “sound” in “motown sound” for the adjective term “sound”, meaning ‘reasonable’.

This might seem to be caused by a vanishing gradient, yet the model correctly classifies the sentence as very negative, suggesting that it is successfully incorporating information from the first negative clause. We separately tested the individual clause “though the plot is interesting”. The standard recurrent model confidently labels it as positive. Thus despite the lower saliency scores for words in the first clause, the simple recurrent system manages to rely on that clause and downplay the information from the latter positive clause—despite the higher saliency scores of the later words. This illustrates a limitation of saliency visualization. first-order derivatives don’t capture all the information we would like to visualize, perhaps because they are only a rough approximate to individual contributions and might not suffice to deal with highly non-linear cases.

By contrast, the LSTM emphasizes the first clause, sharply dampening the influence from the second clause, while the Bi-LSTM focuses on both “hate the movie” and “plot is interesting”.

6 Model 2: Average and Variance

For settings where word embeddings are treated as parameters to optimize from scratch (as opposed to using pre-trained embeddings), we propose a second, surprisingly easy and direct way to visualize important indicators. We first compute the average of the word embeddings for all the words within the sentences. The measure of salience or influence for a word is its deviation from this average. The idea is that during training, models would learn to render indicators different from non-indicator words, enabling them to stand out even after many layers of computation.

Figure 8 shows a map of variance; each grid corresponds to the value of $||e_{i,j} - \frac{1}{N_S} \sum_{i' \in N_S} e_{i'j}||^2$ where $e_{i,j}$ denotes the value for j th dimension of word i and N denotes the number of token within the sentences.

As the figure shows, the variance-based salience measure also does a good job of emphasizing the relevant sentiment words. The model does have shortcomings: (1) it can only be used in to scenarios where word embeddings are parameters to learn (2) it's clear how well the model is able to visualize local compositionality.

7 Model 3: Gate Based Models

The final strategy, gate-based modeling, is based on the intuition of LSTMs: gates are placed at each time step for controlling information flow, but modified from the LSTM gates to enable more straightforward visualization.

Let $h_{t,l}$ denote the embedding for time step t at the l^{th} layer. $h_{t,0}$ denotes the input word embeddings at the 0^{th} layer. We put a more direct control gate at each node: for layer l and time t, we associate it with a scalar gate value $M_{t,l}$ in the range [0,1].

Instead of outputting the current representation (i.e., $h_{l,t}$) for the downstream calculation as in standard recurrent models, the current node outputs the product of the gate value $M_{t,l}$ and the current embedding $h_{l,t}$.

$$h_{t,l} = \tanh(W \cdot [M_{t-1,l} \cdot h_{t-1,l}, M_{t,l-1} \cdot h_{t,l-1}]) \tag{4}$$

where $W \in \mathbb{R}^{2K*K}$. The gate values $M_{t-1,l}$ and $M_{t,l-1}$ are achieved by evaluating the relative importance between the two participant presentations $h_{t-1,l}$ and $h_{t,l-1}$ based on the assumption that the information for the degree of importance has already been encoded in the representations.

$$\begin{aligned} M_{t-1,l} &= \text{sigmoid } (U^T(W'[h_{t-1,l}, h_{t,l-1}])) \\ M_{t,l-1} &= \text{sigmoid } (U^T(W'[h_{t,l-1}, h_{t-1,l}])) \end{aligned} \tag{5}$$

where $W' \in \mathbb{R}^{K*2K}$, $U \in \mathbb{R}^{2K*1}$. $M_{t,l}$ controls the proportion of currently obtained information that should flow into later computation.

If the model thinks not too much relevant information is embedded in $h_{t,l}$, the value of $M_{t,l}$ would be small, pushing the output vector towards the other. $M_{t,l}$ gives straightforward evidence about the relative importance of each node. Models are trained similarly as described in Section 3 and gate values for each node are shown in Figure 9.

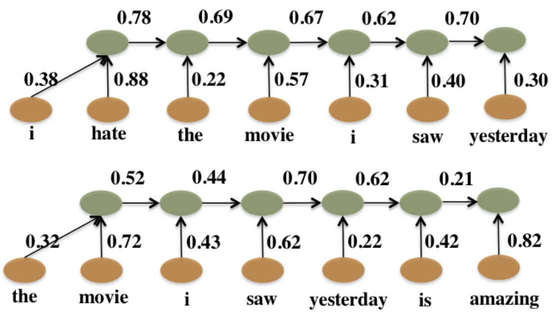

Figure 9: Visualization based on Gate models.

As the figure shows, in the first example, sequences that involve the keyword "hate" ("i hate", "i hate the", "i hate the movie", etc.) are consistently associated with higher weights than the words they are combined with ("the", "movie" and "i"). in the second example, During the last composition step, the model assigns a small weight to the sequence "the movie i saw yesterday is" and a large weight to the sentiment indicator "amazing".

One shortcoming about gate visualization techniques is that they can only provide evidence for local compositions—the relative importance between two nodes involved in the current step of convolution. Gates can't globally show individual word contributions.

8 Conclusion

In this paper, we offer several methods to help visualize how neural models are able to compose meanings, demonstrating asymmetries of negation and explain some aspects of the strong performance of LSTMs at these tasks.

References

Herbert H. Clark and Eve V. Clark. 1977. *Psychology and language: An introduction to psycholinguistics*. Harcourt Brace Jovanovich.

Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

Jeffrey L. Elman. 1989. Representation and structure in connectionist models. Technical Report 8903, Center for Research in Language, University of California, San Diego.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2009. Visualizing higher-layer features of a deep network. *Dept. IRO, Université de Montréal, Tech. Rep.*

Manaal Faruqui and Chris Dyer. 2014. Improving vector space word representations using multilingual correlation. In *Proceedings of EACL*, volume 2014.

Tamar Fraenkel and Yaacov Schul. 2008. The meaning of negated adjectives. *Intercultural Pragmatics*, 5(4):517–540.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE.

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

Laurence R. Horn. 1989. *A natural history of negation*, volume 960. University of Chicago Press Chicago.

Yangfeng Ji and Jacob Eisenstein. 2014. Representation learning for text-level discourse parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 13–24.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Aravindh Mahendran and Andrea Vedaldi. 2014. Understanding deep image representations by inverting them. *arXiv preprint arXiv:1412.0035*.

Anh Nguyen, Jason Yosinski, and Jeff Clune. 2014. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

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. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85.

Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz, and Antonio Torralba. 2013. Hoggles: Visualizing object detection features. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1–8. IEEE.

Philippe Weinzaepfel, Hervé Jégou, and Patrick Pérez. 2011. Reconstructing an image from its local descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 337–344. IEEE.

Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer.

Appendix

Recurrent Models A recurrent network successively takes word w_t at step t, combines its vector representation e_t with previously built hidden vector h_{t-1} from time $t-1$, calculates the resulting current embedding h_t, and passes it to the next step. The embedding h_t for the current time t is thus:

$$h_t = f(W \cdot h_{t-1} + V \cdot e_t) \tag{6}$$

where W and V denote compositional matrices. If N_s denote the length of the sequence, h_{N_s} represents the whole sequence S. h_{N_s} is used as input a softmax function for classification tasks.

Multi-layer Recurrent Models Multi-layer recurrent models extend one-layer recurrent structure by operation on a deep neural architecture that enables more expressivity and flexibly. The model associates each time step for each layer with a hidden representation $h_{l,t}$, where $l \in [1, L]$ denotes the index of layer and t denote the index of time step. $h_{l,t}$ is given by:

$$h_{t,l} = f(W \cdot h_{t-1,l} + V \cdot h_{t,l-1}) \tag{7}$$

where $h_{t,0} = e_t$, which is the original word embedding input at current time step.

Long-short Term Memory LSTM model, first proposed in (Hochreiter and Schmidhuber, 1997), maps an input sequence to a fixed-sized vector by sequentially convoluting the current representation with the output representation of the previous step. LSTM associates each time epoch with an input, control and memory gate, and tries to minimize the impact of unrelated information. i_t, f_t and o_t denote to gate states at time t. h_t denotes the hidden vector outputted from LSTM model at time t and e_t denotes the word embedding input at time t. We have

$$\begin{aligned}
i_t &= \sigma(W_i \cdot e_t + V_i \cdot h_{t-1}) \\
f_t &= \sigma(W_f \cdot e_t + V_f \cdot h_{t-1}) \\
o_t &= \sigma(W_o \cdot e_t + V_o \cdot h_{t-1}) \\
l_t &= \text{tanh}(W_l \cdot e_t + V_l \cdot h_{t-1}) \\
c_t &= f_t \cdot c_{t-1} + i_t \times l_t \\
h_t &= o_t \cdot m_t
\end{aligned} \tag{8}$$

where σ denotes the sigmoid function. i_t, f_t and o_t are scalars within the range of [0,1]. $\times$ denotes pairwise dot.

A multi-layer LSTM models works in the same way as multi-layer recurrent models by enable multi-layer's compositions.

Bidirectional Models (Schuster and Paliwal, 1997) add bidirectionality to the recurrent framework where embeddings for each time are calculated both forwardly and backwardly:

$$\begin{aligned}
h_t^{\rightarrow} &= f(W^{\rightarrow} \cdot h_{t-1}^{\rightarrow} + V^{\rightarrow} \cdot e_t) \\
h_t^{\leftarrow} &= f(W^{\leftarrow} \cdot h_{t+1}^{\leftarrow} + V^{\leftarrow} \cdot e_t)
\end{aligned} \tag{9}$$

Normally, bidirectional models feed the concatenation vector calculated from both directions $[e_1^{\leftarrow}, e_{N_S}^{\rightarrow}]$ to the classifier. Bidirectional models can be similarly extended to both multi-layer neural model and LSTM version.