Recursive Nested Neural Network for Sentiment Analysis

Milad Sharif
Stanford University
msharif@stanford.edu

Hossein Karkeh Abadi
Stanford University
hosseink@stanford.edu

Abstract

Early sentiment prediction systems use semantic vector representation of words to express longer phrases and sentences. These methods proved to have a poor performance, since they are not considering the compositionality in language. Recently many richer models has been proposed to understand the compositionality in natural language for better sentiment predictions. Most of these algorithms are using phrase-tree-based Recursive Neural Networks (RNNs) architectures. In this project we first reproduce the results of state-of-the-art algorithms for sentiment predictions and then propose a different model called Recursive Nested Neural Networks (RN3) with a higher sentiment prediction accuracy.

1 Introduction

A basic task of sentiment predictors is classifying the polarity of a given sentence such as a movie review. The expressed opinion can be positive, negative, or neutral. Many of the early sentiment prediction systems treat each sentence as a bag of words and try to assign a positive or negative score to each word in isolation [1]. These models can make wrong prediction for sentences that although have positive words, convey a negative message. To further understand the compositional effects in natural language, richer and more complex models are required.

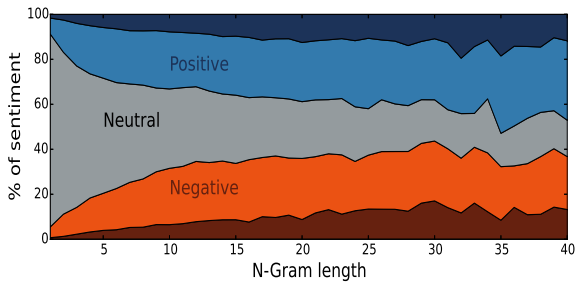
In [2], authors propose a phrase-tree-based recursive neural network to compute compositional vector representations for phrases of variable length and syntactic type. They study the Recursive Neural Tensor Networks (RNTN) which can achieve an accuracy of 45.7% for fined grain sentiment classification. Authors in [3], introduce a deep RNN constructed by stacking multiple recursive layers to utilize the hierarchical capacity of deep neural networks. Their model has achieved an accuracy of 49.8% in a very deep neural network and by employing the dropout technique in which they randomly set the entries of hidden layers to 0 with a probability called the dropout rate.

In this project, we first implement RNTN as our baseline model and then propose Recursive Nested Neural Network (RN3) to further improve the prediction accuracy. As opposed to RNTN, which directly adds the quadratic terms, RN3 tries to increase the interaction between the input vectors by incorporating additional hidden layer. We show RN3 outperforms the RNTN with a much smaller set of parameters and can be trained much faster.

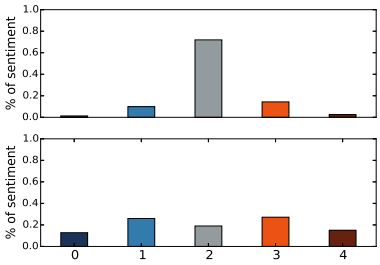
The rest of the paper is organized as follows. In section 2 we describe the dataset that we used for this project. In section 3, we introduce different recursive models including RNTN, and present RN3. In section 4, we evaluate different models and compare their prediction accuracies.

2 Dataset

We use Stanford Sentiment Treebank dataset [2] which allows us to train and evaluate compositional models. The dataset consists of 11,855 single sentences extracted from movie reviews. These re-



(a) Normalized histogram of sentiment annotations at each n-gram length



(b) Distributions of sentiment values for unigrams and full sentences

views are further parsed to extract 215,154 unique phrases and labeled into five classes: negative, somewhat negative, neutral, somewhat positive, positive. Figure 1 shows the distribution of sentiment annotations for different n-grams. As we can see in the figure, most of shorter n-grams are neutral, while full sentence semantics are well distributed.

3 Models

3.1 RNN: Recursive Neural Network

Recursive neural networks (RNNs) comprise an architecture in which the same set of weights is recursively applied within a structural setting: given a positional directed acyclic graph, it visits the nodes in topological order, and recursively applies transformations to generate further representations from previously computed representations of children. Figure (1) shows the RNN structure for phrase "*not so good*".

The prediction and recursive equations for the RNN model are as follows:

$$h_p = f(W \begin{bmatrix} h_l \\ h_r \end{bmatrix} + b),$$

$$\hat{y} = \text{softmax}(Uh + b_s)$$

where h_p, h_l, and h_r are the output vector of the hidden layer in a parent node, and its left and right child nodes, respectively, and $\hat{y}$ is the predicted probability vector for all classes. In the first equation, $f(.)$ is a nonlinear function which can be $tanh(.)$ or $ReLU(.)$.

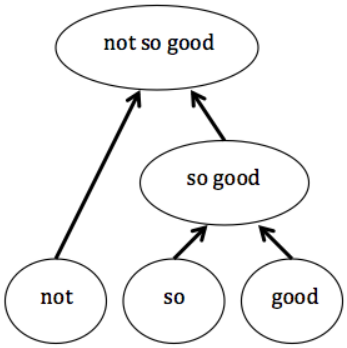


Figure 2: Recursive Neural Network structure for phrase "*not so good*"

3.2 RNTN: Recursive Neural Tensor Network

The main difference between RNTN and RNN is that in the former the model utilizes a richer compositionality function to enforce a greater interaction between children node vectors in creating the parent vector. In RNTN additional quadratic terms of children's vector representation are added to generate the parent vector representation. The equations for the RNTN model is as follows:

$$h_p = f(\begin{bmatrix} h_l \\ h_r \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} h_l \\ h_r \end{bmatrix} + W \begin{bmatrix} h_l \\ h_r \end{bmatrix} + b),$$
$$\hat{y} = \text{softmax}(Uh + b_s)$$

where $d = \dim(h)$, and V is a $d \times 2d \times 2d$ matrix.

3.3 RN3: Recursive Nested Neural Network

In order to improve the predictor's performance we propose a new model which tries to fit a more complex compositionality function by adding new features through a nested neural layer whereas the RNTN model in which quadratic terms are added manually. Figure 3 shows how the RN3 model works. In order to compute a parent vector $h_p \in \mathbb{R}^{d_w \times 1}$, the model first computes new features $h_m \in \mathbb{R}^{d_m \times 1}$ using a nested neural layer, and then using these new features along with the children's vector, it computes the parent vector representation. Then each node uses a softmax classifier on its vector representation to predict its label. The forward propagation equations for RN3 model is as follows:

$$h_m = f(W^{(1)} \begin{bmatrix} h_l \\ h_r \end{bmatrix} + b^{(1)}),$$
$$h_p = f(W^{(2)} \begin{bmatrix} h_l \\ h_r \\ h_m \end{bmatrix} + b^{(2)}),$$
$$\hat{y} = \text{softmax}(W^{(s)}h + b^{(s)})$$

In order to train our model we need to find parameters to minimize the cross-entropy error between the predicted distribution $\mathbf{y}^{(i)}$ and the true one-hot distribution $\mathbf{t}^{(i)}$ of each node i. The set of parameters for this model is $\theta = (W(1), b^{(1)}, W^{(2)}, b^{(2)}, W^{(s)}, b^{(s)}, L)$, where L is the matrix of all vector representations of words in the dictionary. So, the cost function we need to minimize is

$$J(\theta) = \sum_{i=1} CE(\mathbf{y}^{(i)}, \mathbf{t}^{(i)}) + \lambda ||\theta||^2.$$

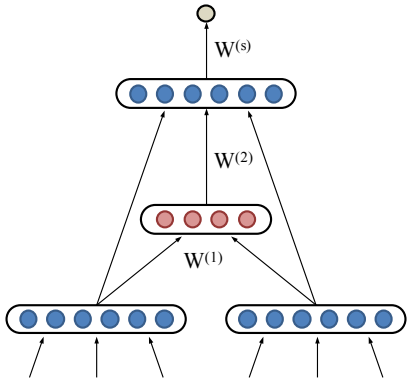


Figure 3: Computation of the parent vector from children vectors

In order to apply Stochastic Gradient Descent (SGD), we need to find the derivative of the cost function with respect to the model's parameters using back propagation method. We can easily prove the following equations for derivatives:

$$\frac{\partial J}{\partial b^{(s)}} = \sum_i (t^{(i)} - y^{(i)}),$$

$$\frac{\partial J}{\partial W^{(s)}} = \sum_i (t^{(i)} - y^{(i)}) {h^{(i)}}^T + \lambda W^{(s)},$$

$$\frac{\partial J}{\partial b^{(2)}} = \sum_i \delta_1^{(i)},$$

$$\frac{\partial J}{\partial W^{(2)}} = \sum_i \delta_1^{(i)} \begin{bmatrix} h_l \\ h_r \\ h_m \end{bmatrix}^T + \lambda W^{(2)},$$

$$\frac{\partial J}{\partial b^{(1)}} = \sum_i \delta_2^{(i)}[2d_w + 1 : 2d_w + d_m],$$

$$\frac{\partial J}{\partial W^{(1)}} = \sum_i \delta_2^{(i)}[2d_w + 1 : 2d_w + d_m] \begin{bmatrix} h_l \\ h_r \end{bmatrix}^T + \lambda W^{(1)}.$$

where $\delta_1^{(i)}$ and $\delta_2^{(i)}$ can be computed recursively from the following equations when f is the ReLU function:

$$\delta_1^{(i)} = \sum_i (h^{(i)} > 0) \text{o} (W^{(s)}(t^{(i)} - y^{(i)}) + \delta_1^{(i,up)} + \delta_2^{(i,up)})$$

$$\delta_2^{(i)} = \sum_i (h_m^{(i)} > 0) \text{o} (W^{(2)} \delta_1^{(i)})$$

where for $k \in \{1, 2\}$,

$$\delta_k^{(i,up)} = \begin{cases} \delta_k^{(p_i)}[1 : d_w] & \text{if } i \text{ is a left node} \\ \delta_k^{(p_i)}[d_w + 1, 2d_w] & \text{if } i \text{ is a tight node} \end{cases}.$$

For every leaf node i we need to add the derivative with respect to $L[w_i]$ using the following equation:

$$\frac{\partial J}{\partial L[w_i]} = W^{(s)}(t^{(i)} - y^{(i)}) + \delta_1^{(i,up)} + \delta_2^{(i,up)}.$$

4 Experiments

We implemented all three models from scratch in Python. In order to optimize different models, we use a dev set to cross-validate different hyper-parameters such as regularization of the weights (λ), word vector size (d_w), hidden layer dimension (d_m) as well as the mini-batch size for AdaGrad. Optimal performance for RN3 was achieved at word vector sizes between 30 and 40 dimensions and hidden layer dimension between 20 and 30. Figure 4 shows the accuracy of RN3 with optimized hyper-parameters over training and dev set versus the number of epochs. As shown in the figure, 15 iterations over the training set achieves the highest dev accuracy. RN3 can be trained much faster comparing to RNTN as it has a lot less parameters.

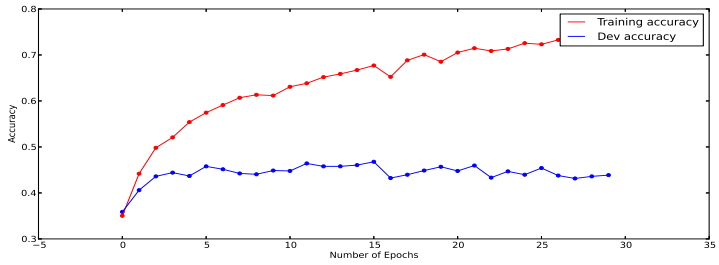
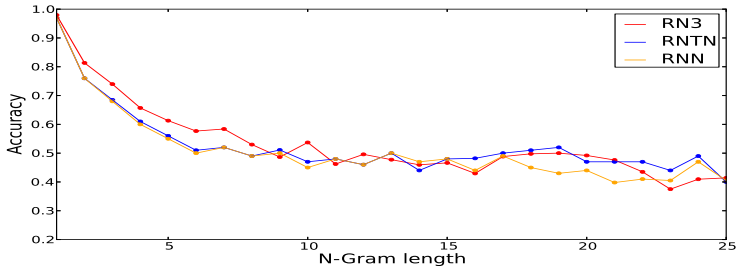


Figure 4: Training error and dev error over number of epochs

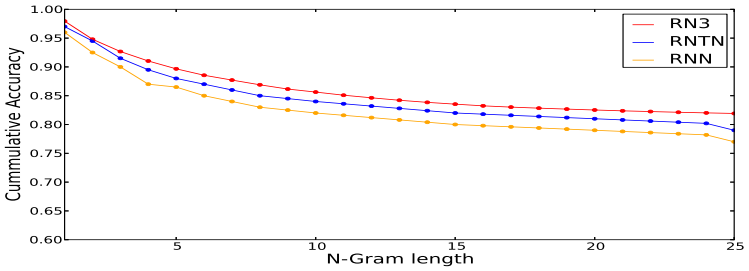
We compare RN3 to other recursive models such as RNTN. We report the accuracy on the test set which is about 20% of the sentences in Treebank dataset. Following table shows the overall accuracy numbers for fine grained prediction at all phrase lengths and full sentences. As we can see in this table, RN3 outperforms all other models with 47.2% accuracy in full sentence sentiment prediction.

Model	Fine-grained	
	All	root
VecAvg	73.3	32.7
RNN	79.0	43.2
RNTN	80.7	45.7
RN3	**81.3**	**47.2**

In order to make detail comparison of RN3 and other recursive models we looked at separate accuracy of each n-grams as well as cumulative accuracy for different models. As shown in Figure 5, RN3 works very well on shorter phrases, while have similar performance for larger n-grams.

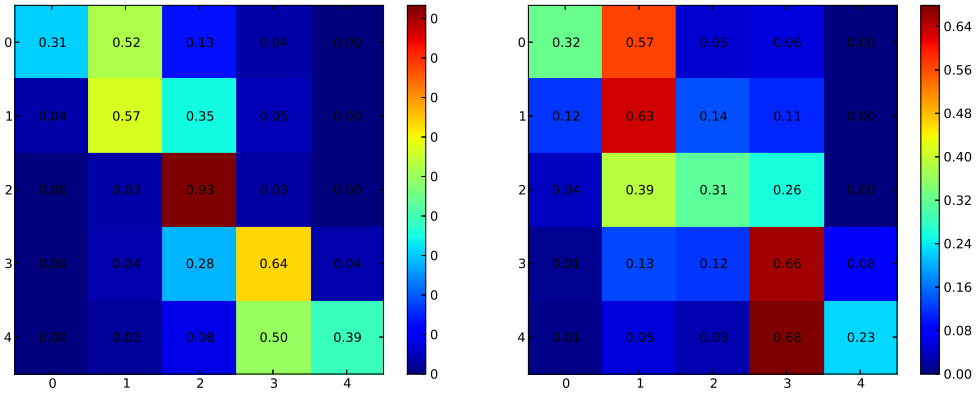


(a) Accuracy for each set of n-grams



(b) Cumulative accuracy of all $\leq$ n-grams

Figure 5: Accuracy curves for fine grained sentiment classification at each n-gram lengths



(a) All (b) root

Figure 6: Confusion matrix with truth down the y axis, and our models guess across the x axis

Figure 6 shows the confusion matrix of the resulting sentiment prediction on the test set using RN3 model for both all-phrases and full-sentence cases. As we can see, for the full-sentence case the largest error is in the prediction of "positive" reviews which are misclassified to "somewhat positive" in 68% of the cases. But for the all phrases case, the largest classification error is for 'negative' class with a misclassification error of 69%.

One of the main strength of RNTN is correctly classifying reviews with positive and negative negations. We manually investigated RN3 to see if it can correctly classify reviews with negations. Figure 7 shows an example of such a review with negation which is misclassified by RNTN but correctly classified by RN3.

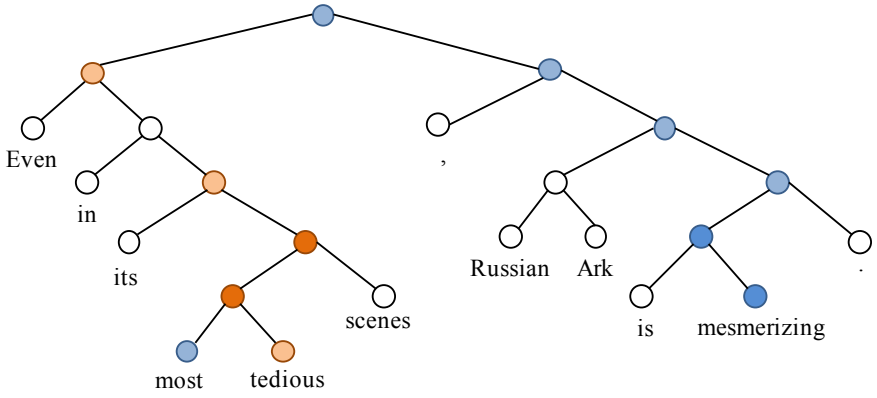


Figure 7: Example of correct prediction for a negation

We probe our model for its predictions on what the most positive or negative n-grams are, measured as the highest activation of the "negative" and "positive" classes. Table 1 shows some n-grams which RN3 their strongest sentiment.

Table 1: Examples of n-grams which RN3 predicted the most positive and most negative responses

n	Most positive n-grams	Most negative n-grams
1	bad, worst, stupid, bore, boring	powerful, brilliant, charming, excellent, pleasant
2	amazing finesse, amazing breakthrough, breathtakingly spectacular, gorgeous epic, most brilliant	phlegmatic bore, painfully unfunny, pathetic junk, 88-minute rip-off, painfully bad
3	brilliant surfing photography, excellent companion piece, beautifully detailed performances, heartfelt and hilarious, mesmerizing cinematic poem	wretchedly unfunny wannabe, barnburningly bad movie, the worst movie, this pathetic junk, worst cinematic tragedies

References

[1] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[2] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, C. Potts, Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, *Conference on Empirical Methods in Natural Language Processing* (EMNLP 2013)

[3] Irsoy, Ozan, and Claire Cardie. "Deep Recursive Neural Networks for Compositionality in Language." Advances in Neural Information Processing Systems. 2014.