Conquering vanishing gradient: Tensor Tree LSTM on aspect-sentiment classification

Shenxiu Liu
Department of Physics
Stanford University
shenxiu@stanford.edu

Qingyun Sun
Department of Mathematics
Stanford University
qysun@stanford.edu

Abstract

Our project focus on the problem of aspect specific sentiment analysis using recursive neural networks. Different from the previous studies where labels exist on every node of constituency tree, we have only one label each sentence, which is only on the root node, and it causes a severe vanishing gradient problem for both RNN and RNTN. To deal with such problem, we develop a classification algorithm specifically for data with only labels only for each sentence, instead of labels on each words. We deploy tree LSTM to solve the problem. and also reform it to a new model called ”Tensor Tree LSTM” as a combination of Tree LSTM and RNTN to conquer the vanishing gradient problem for RNTN.

1 Introduction

User reviews are generated daily on website such as Amazon, Yelp, Alibaba, etc., which are growing into a huge source of customers opinion and preference, the volumes and varieties of reviews are so huge that the demand for summarization by aspect specific sentiment analysis is urgent.

Our work is inspired by the work of Himabindu Lakkaraju, Richard Socher, Chris Manning [RNTN1], which uses recursive tensor network to generate pairs of aspect and sentiment from reviews, and also [RNTN2] which is dealing with Stanford tree bank data. Trying to understand RNTN and developing better model is the starting point of us.

The data we use is the beer review data from beeradvocate, which also appears in the paper [RNTN1]. This data is different from the Stanford tree bank data used in [RNTN2] and [TreeLSTM], which is a standard data set in this field. The data we used consists 8415 sentences, each sentence has a (aspect, rating) pair, where aspect is one of (overall, appearance, taste, palate and aroma), and rating ranges from 1 to 5. The data set is smaller, and the most important difference is that every sentence has only one label, instead of one label for every word, as will also happen to most of real world data. This will cause vanishing gradient problem, especially for RNTN, because of much more parameters, the problem is so serious that RNTN does not learn at all, but just stuck at initial parameters.

We look more closely into the previous work on RNTN to see how they solved this. In [RNTN2], vanishing gradient problem is automatically solved by all nodes on a tree have labels. Whereas in [RNTN1], where they use the same data as us with label only on the top of the tree, thus suffering greatly from vanishing gradient. They came up with a ”hacky” solution: replicate the label on the top to nodes on lower layers, and after some experiments, they decide that for a tree of depth N, they will replicate labels on the nodes of top $\log_2 N$ layers. This is certainly working but not a satisfactory enough solution. We then decided to solve the problem throughly in other ways, by looking at a new model.

Inspired by long short-term memory(LSTM) model[LSTM], we explore the idea of gate and memory to solve the vanishing gradient problem, and we went through recent works and found tree LSTM, a variant of LSTM for tree structure by Kai Sheng Tai, Richard Socher, Chris Manning [TreeLSTM]. It turns out to be a successful solution for our problem.

LSTM updated recurrent neural network, and Tree LSTM updated recursive neural network, hence it is natural to propose a LSTM version to update RNTN. So we propose Tensor Tree LSTM, a model combining ideas from RNTN and Tree LSTM.

GitHub repo: `https://github.com/shenxiul/aspect-sentiment.git`

2 Models

2.1 Vanishing gradient Problems with RNN and RNTN

Recursive neural network is a model generalizing recurrent neural network to tree structure. For each layer of the tree, the equation is

$$h_i = \mathrm{ReLu}\left(Uh_{\text{children}} + b\right).$$

It is proved to be a successful model in sentiment analysis [RNTN2]. However, a problem with RNNs is that during training, components of the gradient vector can grow or decay exponentially over long sequences. Especially if we only have label on root of the tree, the problem will sometimes be fatal. This problem with *exploding* or *vanishing gradients* makes it difficult for the RNN model to learn long-distance correlations in a sequence.

In our data sets, we only have labels on the top of the tree, when the tree depth is not too small, vanishing gradient problem is much more serious than the case when we have labels on each node of the tree.

Recursive tensor network is proposed to represent the interaction between words by a tensor, the tensor that defines multiple bilinear forms, which is like add a quadratic term in the affine transformation in RNN.

$$h = \tanh\left(h_{\text{children}}^T V^{[1:d]} h_{\text{children}} + Uh_{\text{children}} + b\right).$$

This model provides a direct, powerful multiplicative interaction between input vectors, which is useful to characterize the composition relation between words and phrases. RNTN reduce to RNN when V is set to 0, but the tensor can directly relate input vectors, each slice of the tensor is capturing a specific type of composition.

It performs especially well for “X but Y” structure, and high level negation such as “the movie was not terrible” in IMDB data set.

But RNTN suffers more from vanishing gradient problem because it has much more parameters than RNN. In our experiment below, RNTN almost does not learn at all when there is only one label for each sentence.

2.2 Tree-Structured LSTMs

Tree-LSTM unit contains input and output gates i_j and o_j, a memory cell c_j and hidden state h_j. Tree-LSTM unit contains one forget gate f_{jk} for each child k. This allows the Tree-LSTM unit to selectively incorporate information from each child. Tree-LSTM unit takes an input vector x_j, as a vector representation of a word in a sentence. The input word at each node depends on the tree structure used for the network. We only use binary tree LSTM because it suits more to constituency trees we use.

Binary Tree-LSTMs[TreeLSTM]:

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{k=1}^{2} U_k^{(i)} h_{jk} + b^{(i)}\right), \tag{1}$$

$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{\ell=1}^{2} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right), \tag{2}$$

$$o_j = \sigma\left(W^{(o)}x_j + \sum_{k=1}^{2} U_k^{(o)} h_{jk} + b^{(o)}\right), \tag{3}$$

$$z_j = \sum_{k=1}^{2} U_k^{(u)} h_{jk}, \tag{4}$$

$$u_j = \tanh\left(z_j + W^{(u)}x_j + b^{(u)}\right), \tag{5}$$

$$c_j = i_j \odot u_j + \sum_{k=1}^{2} f_{jk} \odot c_k, \tag{6}$$

$$h_j = o_j \odot \tanh(c_j), \tag{7}$$

where $k = 1, 2$ since we use binary tree, and h_j combines h_{j1}, h_{j2} as one vector. x's are inputs, in our case they are not deployed for non-input layers.

The introduction of separate parameter matrices for each child allows the binary Tree-LSTM model to learn more fine-grained conditioning on the states of a unit's children. The number of parameters scales as $O(4|h|^2)$;When the tree is simply a chain, the model reduce to the standard LSTM.

2.3 Tensor Tree LSTM

The binary tensor Tree-LSTM transition equations we proposed are the following:

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{k=1}^{2} U_k^{(i)} h_{jk} + b^{(i)}\right), \tag{8}$$

$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{\ell=1}^{2} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right), \tag{9}$$

$$o_j = \sigma\left(W^{(o)}x_j + \sum_{k=1}^{2} U_k^{(o)} h_{jk} + b^{(o)}\right), \tag{10}$$

$$z_j = h_j^T V^{[1:d]} h_j + \sum_{k=1}^{2} U_k^{(u)} h_{jk}, \tag{11}$$

$$u_j = \tanh\left(z_j + W^{(u)}x_j + b^{(u)}\right), \tag{12}$$

$$c_j = i_j \odot u_j + \sum_{k=1}^{2} f_{jk} \odot c_k, \tag{13}$$

$$h_j = o_j \odot \tanh(c_j), \tag{14}$$

where $k = 1, 2$ since we again use binary tree, and h_j combines h_{j1}, h_{j2} as one vector. Again x's are inputs.

The number of parameters scales as $O(4|h|^3 + 4|h|^2)$; this architecture is therefore useful when the train data is large.

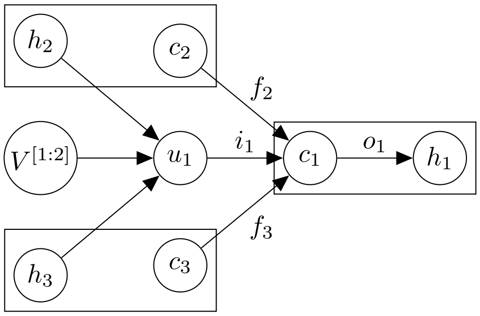


Figure 1: The hidden state h_1 is composition of the memory cell $c_1$of a Tree-LSTM unit with two children h_2 and h_3, the hidden states h_2 and h_3 are interacting multiplicatively through tensor $V^{[1:d]}$. Labeled edges correspond to gating by the indicated gating vector, with dependencies omitted for compactness.This diagram is adapted from [TreeLSTM].

2.4 Classification

In our data sets, we wish to predict labels $\hat{y}$ from a discrete set of classes $\mathcal{Y}$ for the root node in a tree. We use a softmax classifier for each tree

$$\hat{p}_\theta(y \mid \{x\}_j) = \text{softmax}\left(W^{(s)}h_j + b^{(s)}\right),$$
$$\hat{y_j} = \arg\max_y \hat{p}_\theta\left(y \mid \{x\}_j\right).$$

The cost function is the negative log-likelihood of the true class labels $y^{(k)}$ for each tree k:

$$J(\theta) = -\frac{1}{m}\sum_{k=1}^{m} \log \hat{p}_\theta\left(y^{(k)} \,\middle|\, \{x\}^{(k)}\right) + \frac{\lambda}{2}\|\theta\|_2^2,$$

where m is the number of trees in the training set.

3 Experiment results

3.1 Setup

Among 8415 different data entries, there are 8364 distinct sentences, we split them into 5854/836/1674 train/dev/test sets. In the first we want to try out the labeling method where a sentence can have more than one label, however because we only have 51 sentences with two labels, it is hard to really see any success from such approach. We treat every sentence as single labeled instead. The parameter was chosen on dev set and final performance was measured on test set.

For aspect learning, it falls into a 5-class classification problem; for sentiment learning, we round down the ratings to make it also a 5-class classification; and for aspect-sentiment pair, we treat each (aspect, rating) pair as distinct, and it turns to a 25-class classification problem.

To measure the performance, as people on BeerAdvocate are pretty generous in rating and therefore around 57% sentences fall into the group with rating 4, which makes accuracy useless in this case. Thus we use F1-measure as our metric.

3.2 Performance

The optimal performance on test throughout different models are shown in Table 1.

The baselines are set by Naive Bayes method with tf-idf and linear SVM with uni-to-tri-gram and tf-idf. For SVM, all of 1-3 gram are tested with various strengths for L2 regularization and the best performance is chosen.

mean F-1 score	aspect	rating	(aspect, rating) pair
baseline NB (tf-idf)	76.33	48.65	37.18
baseline SVM (tri-gram tf-idf)	83.2	54.03	44.64
RNN	82.28	54.12	38.76
RNTN	21.73	43.1	13.19
Tree LSTM	**85.56**	**56.08**	**47.34**
Tensor Tree LSTM	84.83	54.9	44.38

Table 1: F-1 measure for different models.

For the Neural net models, the parameters are chose by validation on dev set. For training, we use a mini-batch stochastic gradient descent with AdaGrad strategy at minibatch size = 20.

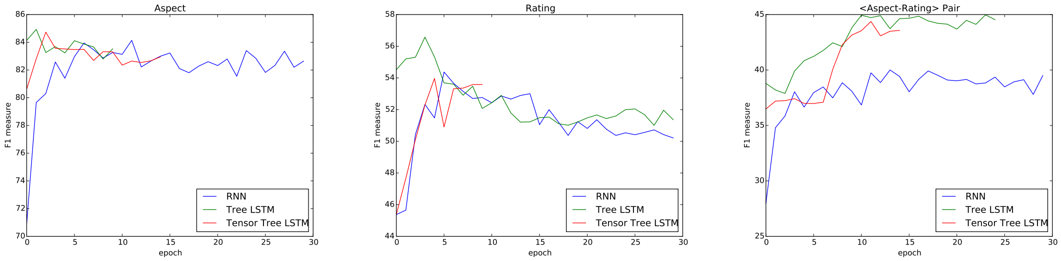
Specifically, as all our neural network models are tree-structured, we use a binarized constituency tree by Stanford CoreNLP tools [CoreNLP].

The optimal parameters we found of each model are listed as following:

1. RNN: word vector dimension 100 (with 30, 50, 100, 150 tested). Dropout at 50% for softmax layer only (no dropout, 20% dropout and 50% dropout tested).
   - Aspect learning: regularization $\rho = 10^{-4}$, learning rate $\alpha = 5 \times 10^{-2}$.
   - Sentiment learning: regularization $\rho = 10^{-5}$, learning rate $\alpha = 2 \times 10^{-2}$.
   - Aspect, sentiment pair: regularization $\rho = 10^{-5}$, learning rate $\alpha = 5 \times 10^{-2}$.
2. RNTN: word vector dimension 30 (25, 30, 35, 50 tested). Dropout at 50% for softmax layer only (no dropout, 50% dropout tested). Any regularization and learning rate are not learning at all (cost function is steady in tens of epochs during training), the parameters we used to test the final performance is regularization $\rho = 10^{-4}$ and learning rate $\alpha = 5 \times 10^{-2}$.
3. Tree-LSTM: word vector to measure performance is 50. We tested 15, 30, 50, 100, 300 dimensional word vectors, with memory dimension same as word vector, also memory dimension is half of word vector dimension. As long as word vector dimension higher than 30, the performance saturates (The performance improvement from the word vector dimension is less than 0.3%, which is almost noise). We also tested the improvement from dropout. We tried no dropout, dropout on only softmax layer and dropout on all nodes with dropout ratio 20%, 30% and 50%. It turns out that no dropout and dropout on only softmax layer with different dropout ratio have the same performance, however dropout on all nodes will decrease the performance by more than 1%. We use 50% dropout on softmax layer to test the performance. Other hyper parameters we found optimal are
   - Aspect learning: regularization $\rho = 10^{-4}$, learning rate $\alpha = 5 \times 10^{-2}$.
   - Sentiment learning: regularization $\rho = 10^{-5}$, learning rate $\alpha = 5 \times 10^{-2}$.
   - Aspect, sentiment pair: regularization $\rho = 10^{-4}$, learning rate $\alpha = 5 \times 10^{-2}$.
4. Tensor tree-LSTM: As the number of parameters in tensor tree-LSTM is large, we only have the ability to exhaust the hyper parameter space for word vector dimension = 30. From some fixed parameters we found that higher word vector dimension is not helping because of the limited data. For the dropout strategy, we still use dropout on softmax layer at 50%, which turns out to be about 1% improvement on dev set than no dropout. Other optimal hyper parameters we found are
   - Aspect learning: regularization $\rho = 10^{-4}$, learning rate $\alpha = 5 \times 10^{-2}$.
   - Sentiment learning: regularization $\rho = 10^{-4}$, learning rate $\alpha = 2 \times 10^{-2}$.
   - Aspect, sentiment pair: regularization $\rho = 10^{-5}$, learning rate $\alpha = 5 \times 10^{-2}$.

3.3 Learning curve

The dev F1 for different models in optimal hyper parameter stated above during training is shown in Fig. 2.



(a) aspect

(b) sentiment

(c) pair

Figure 2: Learning curve for different models.

We can see that all those models are learning pretty fast. If we turn down learning rate, in terms of learning speed it will be more smooth. However in terms of best performance over epochs, the result will be worse. This might be caused by our data set is relatively small and too many parameters will lead the learning into local minimums too easily. Therefore we stick on our parameter settings.

4 Comparison and conclusion

Form our experiments, RNTN suffers most from vanishing gradient problem, RNN also suffers from that a bit, especially in (aspect, rating) pair leaning. LSTM variants cure such problem for both RNN and RNTN. We can see from Table 1 that both tree-LSTM and tensor tree-LSTM can get a good performance, in contrast to RNN and RNTN.

Overall the best model is tree LSTM, which is somewhat counter intuitive because tensor tree-LSTM will reduce to tree LSTM when $V = 0$, it should have more ability in learning. This is not the case in our problem probably because of the limited data set, tensor tree-LSTM has too many parameters and it will be easy to overfit. Beer review is also tricky in overfitting, especially for sentiment analysis because it depends too much on personal experience. If we can work in a larger data set tensor tree LSTM should be less problematic.

5 Variants of model and future work

It would be useful to use Low rank approximation of the tensor, which will reduce the number of parameters, hence increase the training speed.

And to avoid overfitting when training the tensor, we can do connection dropout on the tensor, by multiple a mask tensor with iid Bernoulli entries.

Another thing is to replace softmax loss with max margin loss for multi-class classification, when the number of class grows large. We observe that when we classify aspect-sentiment pairs, we have 25 class, the performance of max margin loss could be better than softmax and cross-entropy loss.

Acknowledgement

The authors are grateful for the help of Richard Socher on the suggestion of several aspects, and also for the help of Himabindu Lakkarahu on sharing the beer review data sets.

References

[LSTM] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory;

[TreeLSTM] Kai Sheng Tai, Richard Socher, Chris Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks;

[RNTN1] Himabindu Lakkaraju, Richard Socher, Chris Manning. Aspect Specific Sentiment Analysis using Hierarchical Deep Learning;

[RNTN2] Richard Socher, etc.; Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank.

[CoreNLP] http://nlp.stanford.edu/software/corenlp.shtml