Sentiment Analysis using Recursive Neural Network

Sida Wang
CS229 Project
Stanford University
sidaw@cs.stanford.edu

Abstract

This work is based on [1] where the recursive autoencoder (RAE) is used to predict sentiment distributions. In this project, I compare the performance of several different tree building schemes and find that greedily merging nodes with minimal autoencoder error gives the best performance, which is better than using the correct parsing tree, among others. I then apply the recursive neural network (RNN) on the newly released Yelp restaurants review dataset and obtain 88% test accuracy. Finally, I use the dual (matrix-vector) RNN model on the sentiment prediction task on both the Rotten-Tomatoes (RT) movie reviews and Yelp restaurants reviews. This report contains the first sentiment analysis result using the dual RNN and one of the first results on the Yelp review dataset.

1 Introduction

The aim of the sentiment analysis task is to predict the expressed opinion in a sentence, paragraph or document. In this work, I focus on the simpler task of polarity analysis: to determine if the opinion expressed in a review is positive or negative. Simple bag of words representation is inadequate for this task, as many of the reviews are short and the sentiment depends crucially on word and phrase order [2]. Although one can achieve reasonable performance using simple methods like Naive Bayes, voting with lexica, and bag of words features etc., much more can be done.

The examples below (the first 3 examples are random, the rest are picked from only the first 50 reviews) show the difficulty of this task. Some positive reviews can contain many negative words (the last movie review in table 1). Moreover, many reviewers use metaphor, sarcasm, and other literary devices to express themselves.

In section 2, I introduce the task of polarity analysis on the Yelp dataset [4], consisting of about 150k labeled reviews. Then I present results on this dataset.

By modeling the recursive structure of each sentence, Socher et al. [1] showed that RAE is a promising method for sentiment analysis. They achieved state-of-the-art results on the Experience Project, and the Rotten-Tomatoes (RT) datasets. However, it is not clear how much the tree structure helps. In section 3, I investigate this question using 6 distinct tree structures: degenerate (linear) trees, complete binary tree (shallowest tree), random tree, minimal reconstruction error tree and the Stanford parser tree. Having a good tree structure is important for getting the last 1-2% to achieve state-of-the-art. However, a difference of 1-2% mean that the RNN classifier does not depend crucially on the tree structure.

Lastly, I provide a brief introduction to Richard Socher's dual model, how I use it for sentiment analysis, and present some sentiment analysis results on both RT and Yelp using the dual model.

Table 1: The movie review dataset has around 10k reviews, consisting of mostly single sentences. Below is sample from the first 50 reviews.

label	Examples from movie review
neg.	simplistic, silly and tedious.
neg.	a sentimental mess that never rings true .
pos.	if this movie were a book, it would be a page-turner, you can't wait to see what happens next.
neg.	wait for it to hit cable.
pos.	in its ragged, cheap and unassuming way, the movie works.

Table 2: The Yelp dataset has around 150k reviews, about 20% of the reviews has under 150 words.

label	Examples from Yelp
neg.	yuck. nasty, greasy burgers and fries.
pos.	best pad thai ever.
pos.	when in doubt, you can't go wrong at a peets.
neg.	two words: mustard bomb.
pos.	i've teabagged all the owners at some point. they deserve the love.
neg.	the food can be described quite simply: only good when drunk.

2 The Yelp dataset

The Yelp academic dataset [4] consists of over 150k reviews with IDs identifying which user wrote each review as well as which business is being reviewed. Occasionally, reviews are also rated as helpful, funny or cool. These review ratings are quite sparse and are not used. When these ratings become more populated, another interesting task is to predict the review ratings distribution, which can be more objective. There are more 4 or 5 stars review than there are 1-3 stars reviews as is commonly observed in review datasets.

To simplify this task to polarity predictions, I take all the 4-5 stars reviews to be positive, all the 1-2 stars reviews to be negative, and ignore the ambiguous 3 stars reviews. I take 2000 reviews from each of the four remaining star ratings (1,2,4,5) for 8000 reviews in total, 10% of which is used as the test set. I cross-validate on the remaining 90% to determine the best hyper-parameters. Although it is possible to predict lengthy reviews by averaging over its component sentences, I only take those reviews that are shorter than 150 words to simplify this task.

After cross-validating to determine the best hyper-parameters, I got a test set performance of 0.881 (corresponding training set performance = 0.971). See figure 1 for a visualization of word embeddings and table 3 for some error cases. It is not surprising that the Yelp dataset is easier than the RT dataset, because most Yelp reviews are longer than those one sentence comments selected in RT. As the reviews get longer, bag-of-words based methods also performs better, resulting in a better baseline.

3 Comparing the different tree-building schemes

One interesting question for me is whether the tree structure actually help, and this question is investigated here. I repeat the authors' original experiments in [1] on the RT dataset, with different tree structures, and with all other parameters fixed. I run 10-fold cross-validation for each of the 6 different tree structures. The reason for 10-fold cross validation is that these different tree structures only make a small difference in the final performance, so some averaging is required to make this comparison more confident and meaningful.

In figure 2, the "read forward" tree is the left-branching (linear) tree where the RNN always "read" the next word to the right starting from the beginning of each sentence (i.e. the RNN behaves like a

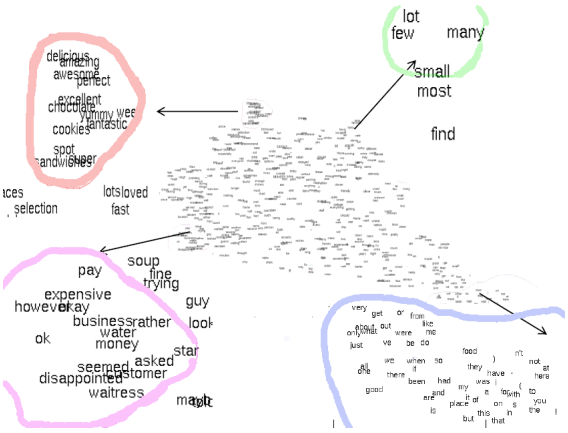

Figure 1: t-SNE embedding of learned word vectors on common restaurant review vocabulary, capturing their polarity, but also other semantics information such as the concept of quantity in green.

Table 3: Some error cases from Yelp. The first column is the output of the RNN classifier, which is different from the actual label.

lbl	Errors from Yelp
pos.	Most people only order from here their first year on campus because it is free delivery. Once they have, they usually don't return.
neg.	Good service but the inventory sucks! A lot of the stuff I wanted didn't have my size unfortunately, the Express in Memorial Mall had a better selection (*this one has 4-5 stars)*
neg.	Well curated vintage selection with some locally handmade pieces as well. There are some deals, though most things are on the pricey side. Definitely recommended for finding all the stuff you never inherited from your grandmother
pos.	At first glace, this place is very nice! The soaps are high quality, the rooms are clean and with pretty furnishings, but to me, it's the sleeping experience that really determines how great a hotel is... My sleeping experience sucked.

recurrent neural network). The "read backward" tree is the right-branching tree. Here I start at the last word in a sentence, and merge with the next word to the left.

The shallowest tree in figure 2 is the complete binary tree, which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

To generate the random tree, I start with a list of nodes to be merged, initialized to be the list of all the leaf nodes. Then a pair of consecutive nodes is picked randomly from this list and merged into a parent node p. p replaces the pair picked in the list of nodes to be merged. This process repeats until all nodes are merged. The random seed is fixed in each particular sentence for consistency.

Least reconstruction error is the scheme used in [1], where the pair of consecutive nodes that generate the least reconstruction error is merged greedily.

The Stanford parser tree is the tree given by running the Stanford PCFG parser on this dataset [3].

Results are shown in figure 2, where the error bar shows one standard deviation if the 10 CV runs are independent. In reality, since the validation set is chosen by a deterministic process, all the CV runs agree with each other, so the actual confidence is better than what the error bars indicates. That is, if tree-type A is better than tree-type B in CV run #1, it is also better in CV run #2. A particular run for all tree types use exactly the same training/validation data.

From figure 2, we see that reading forward is better than reading backward. The Stanford parser tree performs poorly, and the original least reconstruction error scheme performs the best. These

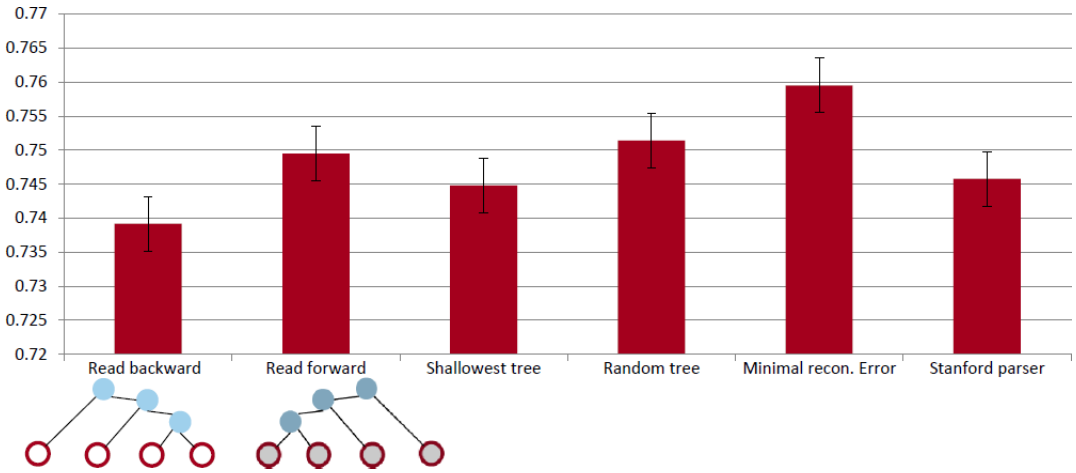

Figure 2: Testing accuracy using different tree structures.

differences are small but consistent. A more conclusive test is to use a more challenging and sensitive tasks such as paraphrase detection to test different tree structures.

I first expected that by using the correct tree (to which the Stanford parser tree is a much better approximation than the least reconstruction error tree), phrases can be more efficiently represented to achieve the best performance. But this is not the case, which means that the RNN is not using all the structural information that is given to it. However, this is not too surprising either, because the Stanford parser tree closely resembles the "read backward" (right branching) tree due to the linguistics of English. The "read backward" tree also performed poorly.

4 Extensions

4.1 The dual model

Dataset	Train	Test
Yelp 5-d	0.865	0.742
RT 5-d	0.819	0.660
RT 10-d	0.893	0.631
RT 20-d	0.813	0.662
RT 20-d overfitting	0.997	0.599
RT voting with two lexica		0.631
RT Tree-CRF [6]		0.77

Dataset	Train	Test
RT 5-d LR	0.807	0.698
RT 10-d LR	0.7670	0.629
RT 20-d LR	0.7653	0.640

Table 4: Dual model results on Yelp and RT. n in n-d is the dimension of word vectors used.

The dual model attempts to capture compositionality by modeling each word as both a vector a and a matrix A. Pairs of child vectors $a, b \in \mathbb{R}^n$, and child matrices $A, B \in \mathbb{R}^{n \times n}$ are combined to get the parent vector $p = W_{vec}[Ba; Ab]$, and the parent matrix $P = W_{mat}[A; B]$. This is designed so that each word also acts on another word to produce a modified meaning (e.g. not good $\rightarrow$ bad, very good $\rightarrow$ great).

In order to do sentiment analysis, I use a margin-based classifier on entire tree. The cost function of the classifier is $c = \max(0, 1 - ls)$, where $s = \frac{1}{|T|} \sum_{t \in T} f(t)$, $l \in \{-1, 1\}$ is the label, t is a node in tree T. The structure T is provided by the Stanford parser. $f(t) = Wt + b$ is just a linear function of each tree node $t = [a; A]$. This method can quick overfit to the modest 10k sentences RT dataset, even with 5d word vectors.

Logistic regression on individual tree nodes is an alternative to the margin-based classifier on the entire tree. The cost function here is $c = \sum_{t \in T} \mathrm{CE}(l, \sigma(Wt))$ where l is the polarity label for the entire tree T. This does not overfit as much as the margin-based method, and gives slightly better results.

As in the original RNN sentiment work [1], I extract features from sentence trees and make prediction based on logistic regression on those features, instead of just using the raw scores or the average of logistic regression on each node. This technique gives a 1-3% improvement over just using the raw scores. The results are shown in figure 4. These results are better than some simple baselines, but is still far from the state of the art. The n-dimensional model has $n^2 + n$ parameters per word. This model quickly overfits for $n = 20$ (440 parameters per word) without generalizing to the validation set. In comparison, a 50-100d vector is usually used per word in deep learning language models [7].

A preliminary comparison between the dual and original RNN shows that the naive dual model does not do well for sentiment analysis. One reason is that too many parameters are spent on the word matrix. While many words acts like functions that modifies meaning, it does not make sense to use n^2 parameters to represent this function and only use n parameters for the word vector (i.e. the meaning of the word). However, a simple linear combination like $p = \sigma(W[a; b])$ used in the original RNN probably does not have the power to represent the interactions between words. Low rank approximations to word matrices where $W_{mat} = \lambda I + UV$, fixed basis approximation to word matrices where $A_j = \sum_i \alpha_i^j M_i$ (so each word matrix can be represented just by α_i^j and a shared set of fixed basis M_i) are promising next steps for the dual model. A more obvious approach is to give up on RT, and move to a bigger dataset.

4.2 More powerful RNNs

Besides the dual model, there are many other extensions to the original "linear" RNN $p = \sigma(W[a; b])$, where word vectors can be combined more expressively. In particular, two-layer RNN where $p = \sigma(W_{ph}\sigma(W_{hc}[a; b]))$; matrix factorization modulated by the input where $p = \sigma(W_{ab}[a; b]), W_{ab} = W_{of}\,\mathrm{diag}(W_f[a; b])W_{fi}$ [5]; quadratic function where $p_i = \sigma(a^T W_i b + V_i[a; b])$ are some of the many possibilities.

Out of these, I implemented the two-layers RNN, which got a validation accuracy of 0.68 on RT without much cross-validation (this is very preliminary and can probably get much better).

Acknowledgments

I thank Richard Socher for starting this work, releasing his code, and helpful discussions. In this project, I used his released code, and dual RNN pre-training code as starting points. I thank Prof. Manning for allowing me to use the NLP clusters for this project.

References

[1] R. Socher, E. Huang, J. Pennington, A. Y. Ng, and C. D. Manning 2011 Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions *EMNLP*

[2] B. Pang, L. Lee, and S. Vaithyanathan 2002 Thumbs up? Sentiment classication using machine learning techniques. *EMNLP*

[3] D. Klein and C. Manning. 2003. Accurate Unlexicalized Parsing. *Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.*

[4] The Yelp academic dataset `http://www.yelp.com/academic_dataset`

[5] J. Martens and I. Sutskever 2011 Training Recurrent Neural Networks with Hessian-Free Optimization, ICML

[6] T. Nakagawa, K. Inui, and S. Kurohashi. 2010 Dependency tree-based sentiment classication using CRFs with hidden variables. In NAACL, HLT.

[7] R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of ICML