Semi-Supervised Recursive Autoencoders

Adrian Guthals
aguthals@cs.ucsd.edu

David Larson
dplarson@ucsd.edu

Abstract

We evaluate semi-supervised recursive autoencoders (RAE) as a method for predicting the sentiment of sentences. Using random word initialization, meaning vectors of length 20 and cross-validation, we are able to predict the sentiment of a movie review dataset with a 74.5% accuracy, which is within 3% of the 76.8% accuracy reported in the 2011 paper "Semi-Supervised Recursive Autoencoders" by Soch et al. Analysis of the words and phrases predicted to be the most positive and negative, as well as the words and phrases with the most similar, reinforce that RAE is an effective method for predicting sentence-level sentiment.

1 Introduction

Learning the meaning of text documents, including short documents such as Twitter messages, is an active field of research in computer science. Until 2011, the state-of-the-art methods for predicting sentence-level meanings relied on either bag-of-words representations or manually generated resources, both of which have difficulty in dealing with complex sentiments. Bag-of-words methods ignore the word ordering of a sentence, which means two sentences with the same words will be treated the same (e.g. "Man sees dog." versus "Dog sees man."). Meanwhile, methods that use manually generated resources are unable to deal with sentences beyond the scope of the resources, which would be a common occurance for sources that contain rapidly evolving linguistics, e.g., Twitter messages.

To overcome these shortcomings, Socher et al. introduced a new semi-supervised method based on neural networks [1]. This study will attempt to recreate and expand upon the results reported in Socher et al.

2 Recursive Autoencoders

For any English sentence of length n, the syntactic structure of the sentence can be represented by a binary tree, where each word is a leaf node and the meaning of each node (x) is represented as a vector of length d ($x \in R^d$) [2]. A node that connects two or more words is a phrase, with its meaning being a function of its two children nodes. If node k has children i and j, then the meaning of node k is:

$$x_k = h(W[x_i; x_j] + b) \tag{1}$$

where W and b are parameters to be learned while $h()$ is a pointwise sigmoid function which maps R^d to $[-1, +1]^d$ ($h() = tanh()$ for this study). As x_k, x_i, and x_j are $\in R^d$, then $W \in R^{d \times 2d}$ and $b \in R^d$. To learn W and b, the target meaning t of the sentence must be known, which is usually not the case. To get around this, we use autoencoders, whose goal is to reconstruct the input [2].

2.1 Autoencoders

For a node k where t is unknown, the inputs z_i and z_j of its children node meanings x_i and x_j can be approximated by:

$$[z_i; z_j] = Ux_k + c \tag{2}$$

where x_k is the same as in Equation 1, and U and c are parameters to be learned ($U \in R^{2d \times d}$, $c \in R^d$). Then the square loss at node k is:

$$E = ||x_i - z_i||^2 + ||x_j - z_j||^2 = |||[x_i; x_j] - Uh(W[x_i; x_j] + b) - c||^2 \tag{3}$$

and the total loss of the tree is the sum of all errors at non-leaf nodes. Importance should be placed more on reduce the error of nodes that have more children nodes, which can be done by modifying the error function to:

$$E_1(k) = \frac{n_i}{n_i + n_j}||x_i - z_i||^2 + \frac{n_j}{n_i + n_j}||x_j - z_j||^2 \tag{4}$$

where n_i and n_j are the number of children nodes of nodes i and j respectively.

2.2 Binary Tree Construction

If the tree structure of a sentence is unknown, it can be approximated using a greedy algorithm:

1. calculate $E_1(k)$ for all $n - 1$ pairs of consecutive words
2. select pair with minimum error and connect with a node
3. calculate error for all possible pairs (now $n - 2$ pairs)
4. select pair with minimum error and connect with a node
5. repeat until only one choice left for the root node

2.3 Predicting Labels using Meanings

Although the target meaning is unknown, other target labels may be known, e.g., is the sentence positive or negative. If there are r target labels, the probability of the labels at node k can be found using multiclass logistic regress:

$$\bar{p} = \text{softmax}(Vx_k) \tag{5}$$

where $V \in R^{r \times d}$ is a parameter matrix. The log loss of the predictions is therefore

$$E_2(k) = -\sum_{i=1}^{r} t_i \log p_i \tag{6}$$

where $\bar{t}$ is the true label at node k and is in $R^{r \times d}$.

To predict the target label for all internal nodes, excluding leaf nodes, we need to minimize the objective function J, which is defined as

$$J = \frac{1}{m} \sum_{(s,t) \in S} E(s, t, \theta) + \frac{\lambda}{2}||\theta||^2 \tag{7}$$

where m is the length of each labelled training sentences in the set S, $\theta =< W, b, U, c, V >$ are the parameters to learn, λ is the strength of L_2 regularization, and the total error for a sentence s with a true label t ($E(s, t, \theta)$) is

$$E(s, t, \theta) = \sum_{k \in T(s)} \alpha E_1(k) + (1 - \alpha)E_2(k) \tag{8}$$

where $T(s)$ is the set of non-leaf nodes and α is a hyperparameter weighting the importance of the two per-node errors.

2.4 Computing the Gradient

To minimize the objective junction J with with gradient following techniques, we need to compute the folowing derivative with parameters $\theta = (W, b, U, c, V, W^{(label)})$ where N is the number of training examples and $W^{(label)} \in R^{K \times d}$ is the wieght matrix governing final output labels [1].

$$\frac{\partial J}{\partial \theta} = \frac{1}{N} \sum_{(x,t)} \frac{\partial E(x, t; \theta)}{\partial \theta} + \lambda\theta \tag{9}$$

To compute this derivative efficiently we use backpropagation as described in [2], where the gradient for each node j is conditional on the node being an input or output node. First, we define the partial derivative vector at the node j as

$$\frac{\partial J}{\partial j} = \delta_k = \frac{\partial L(z_k, t_k)}{\partial z_k} h'(a_k) \tag{10}$$

Here, L defines the local loss function for node k and t_k is the known training label. Output nodes with decoded values $[c'_i; c'_j]$ only contribute to J with the error function $E_1(k)$, thus

$$\delta_k = \frac{\partial J}{\partial a_i} = -2\alpha \left[\frac{n_i}{n_i + n_j}(c_i - c'_i); \frac{n_j}{n_i + n_j}(c_j - c'_j) \right] h'(a_i) \tag{11}$$

Remaining output nodes contribute with the error function $E_2(k)$.

$$\delta_k = (1 - \alpha)(-t + softmax(a) \sum_{i=1}^{r} t_i) \tag{12}$$

Internal nodes exhibit a more complex derivative that depends on neighboring nodes:

$$\delta_k = h'(a_k)(\sum_{i=1}^{d} \delta_i V_{ik} + \sum_{i=1}^{2d} \delta_i U_{ik} + \sum_{i=1}^{K} \delta_i W_{ik}^{(label)}) \tag{13}$$

Here, the term $\sum_{i=1}^{d} \delta_i V_{ik}$ sums over all other dependent internal nodes while $\sum_{i=1}^{2d} \delta_i U_{ik}$ sums over output nodes contributing with the error function $E_1(k)$ and $\sum_{i=1}^{K} \delta_i W_{ik}^{(label)}$ sums over output nodes contributing with the error function $E_2(k)$.

2.5 Gradient Verification

It is important to verify the accuracy of the gradients calculated using backpropogation. For this study we have chosen to verify the accuracy of backpropogation by comparing against gradients calculated numerically using finite central-differences:

$$\frac{\partial J}{\partial \theta} = \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} + O(\epsilon^2) \tag{14}$$

where ϵ is the grid spacing. The computed gradient was then verified to be correct within the order of 10^6.

A downside to using numerical derivatives to verify backpropogation is time complexity. If $W \in R^{2d \times d}$ then the time complexity of computing derivatives is $O(d^2)$ using backpropgation and $O(d^4)$ using finite central-differences, which is not feasible for real-world applications. One option for reducing the time complexity of checking the derivatives is to check a subset of the derivatives, chosen randomly, and assume those selected derivatives are representative of the entire set.

3 Experiments

3.1 Datasets

We use the same movie reviews dataset as in [1], which consists of 10662 snippets from reviews posted to the Rotten Tomatoes website[1]. Each snippet is roughly equivalent to a single sentence and includes a positive/negative label, with the entire dataset containing 5331 positive and 5331 negative labelled snippets. For all experiments we randomly selected 90% of the original dataset as a training set, with the remaining 10% used as a testing set. In splitting the dataset we have taken care to prevent any snippets from existing in both sets, so as to not contaminate the results. Also, non-word modifying punctuation marks (i.e. commas, colons, semi-colons and periods) were treated as separate words in each sentence, rather than lumped together with their neighbors.

[1] `http://www.rottentomatoes.com`

Table 1: n-grams predicted to be the most positive and negative.

	$n = 1$		$n = 2$	
Ranking	**Positive**	**Negative**	**Positive**	**Negative**
1	beautiful	fails	moving and	lack of
2	brilliant	boring	an enjoyable	boring .
3	thoughtful	neither	and beautifully	how bad
4	triump	bad	a moving	the dullest
5	flaws	flat	a triumph	flat ,
6	beautifully	predictable	a beautiful	of bad
7	success	bore	the best	it fails
8	spectacular	poorly	and powerful	it isn't
9	enjoyable	suffers	its flaws	and predictable
10	wonderful	unnecessary	a wonderful	a boring

3.2 Optimization

We use limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), a well-known quasi-Newton optimization method, to learn the parameters θ. Specifically we use the Matlab-based L-BFGS function from the minFunc toolbox [3] to minimize the objective function J.

3.3 Hyperparameters

Due to time constraints, we evaluated the RAE model with $d = 20$, instead of $d = 100$ as in Socher et al. Also, we did not perform a grid search on α and λ, instead electing to use the same values as Socher et al. ($\alpha = 0.2$, $\lambda =< 10^{-5}, 10^{-4}, 10^{-7}, 10^{-2} >$).

For the purpose of this study, we arbitrarily set $\epsilon = 10^{-4}$ and the maximum absolute error between the backpropogation and numerical derivatives to 10^{-6}.

3.4 Random Word Initialization

The meaning vector of each word was initialized randomly as in Socher et al. by using a Gaussian distribution of $\mu = 0$ and $\sigma^2 = 0.05$.

3.5 Results

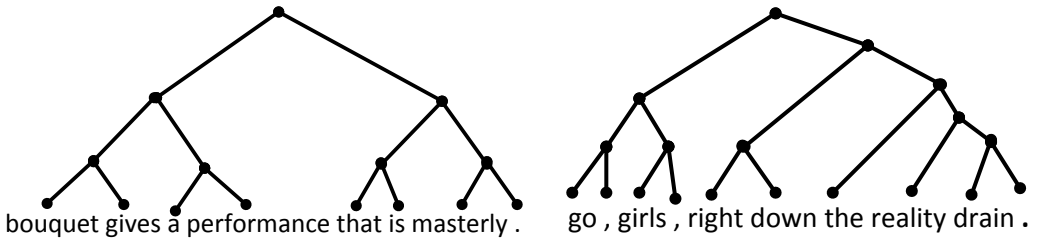
With 10-fold cross-validation and $d = 20$, we were able to achieve a prediction accuracy of 74.5% on the testing set (10% of the original dataset). Our accuracy is within 3% of the the value reported by Socher et al. (76.8%). It should be noted that the accuracy reported by Socher et al. was for $d = 100$, while we used $d = 20$, which indicates that there is a diminishing return after d exceeds some threshold value.

3.5.1 Most Positive and Negative

Table 1 shows the words and phrases predicted to be the most positive and negative. The only result that stands out as possibly an error is the word "flaws" being predicted as positive rather than negative. Although the word "flaws" may be normally associated with a negative meaning, it could be associated with a positive meaning due its usage in a phrase, e.g., "despite its flaws".

3.5.2 Similar Meanings

Comparing words and phrases with the most similar meanings is another intiutive method for evaluating the trained model. The similarity between a pair of words or phrases, with meaning vectors



(a) positive (b) negative

Figure 1: The binary tree structured of a positive sentence and negative sentence, with black solid circles to indicate nodes. The structure was determined using the greedy algorithm outlined in Section 2.2.

x and y, can be quantified using cosine similarity:

$$\text{cosine similarity}(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} \tag{15}$$

which is bounded between -1 (opposite in meaning) and $+1$ (same in meaning). Applying this metric to Table 1 we find the following pairs to be most similar:

1. Positive ($n = 1$): "bad" and "boring"
2. Negative ($n = 1$): "wonderful" and "enjoyable"
3. Positive ($n = 2$): "how bad" and "of bad"
4. Negative ($n = 2$): "moving and" and "and powerful"

3.5.3 Tree Structure of Interesting Sentences

Visually inspecting the tree structure of sentences offers on insight on the strengths and weaknesses of the greedy algorithm. Figure 1 shows the tree structure of a positive and negative sentence. The greedly algorithm correctly determined the structure of the positive sentence, but had issues with the negative sentence. Specifically, the structure should have connected "right now" with "go , girls" instead of "the reality drain .". Incorrectly determine tree structures such as the one shown in Figure 1 are likely causes of error for the RAE model.

4 Conclusion

Semi-supervised recursive autoencoders were used to predict the meanings of a movie review dataset. Using 10-fold validation, randomly generated word meaning vectors, and meaning vectors of length 20, our implementaiton of the method achieves a prediction accuracy of 74.5%. Although our accuracy is less than the amount reported (76.8%) in by Socher et al., we use smaller meaning vectors (20 versus Socher et al.'s 100). Additionally, analysis of the most positive and negative, and most similar words and phrases indicate that recursive autoencoders are an effective method for predicting the meanings of text snippets.

References

[1] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[2] C. Elkan, "Learning meanings for sentences," http://cseweb.ucsd.edu/~elkan/250B/, February 2013.

[3] M. Schmidt, "minFunc," http://www.di.ens.fr/~mschmidt/Software/minFunc.html, accessed: 03/04/2013.