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 and meaning vectors of length 20, we are able to predict the sentiment of a movie review dataset with a 74.5% accuracy, which is only 3.3% less than the 76.8% accuracy reported in the 2011 paper "Semi-Supervised Recursive Autoencoders" by Soch et al.

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. To overcome this shortcoming, 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 $[-1, +1]^d$. 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

2.4 Backpropogation

Backpropogation is an efficient method for computing the derivatives required for training a neural network. Given

2.5 Goal of Training

2.6 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:

$$\frac{\partial J}{\partial \theta} = \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon} + O(\epsilon^2) \tag{5}$$

where ϵ is the grid spacing.

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.

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].

Convergence: error less than 10^{-6} (as stated in the project description)

Regularization: ?

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

Table 1: Words predicted to be the most positive and negative.

Ranking	Positive	Negative
1	beautiful	fails
2	brilliant	boring
3	thoughtful	neither
4	triump	bad
5	flaws	flat
6	beautifully	predictable
7	success	bore
8	spectacular	poorly
9	enjoyable	suffers
10	wonderful	unnecessary

Table 2: Phrases (length 2) predicted to be the most positive and negative.

Ranking	Positive	Negative
1	moving and	lack of
2	an enjoyable	boring .
3	and beautifully	how bad
4	a moving	the dullest
5	a triumph	flat ,
6	a beautiful	of bad
7	the best	it fails
8	and powerful	it isn't
9	its flaws	and predictable
10	a wonderful	a boring

3.3 Experiment 1: RAE

The full method (RAE)

d = 20: prediction accuracy = 74.5%

10-fold cross validation

same hyperparameters as Socher et al.

3.4 Results

3.4.1 Most Positive and Negative

Table 1–2 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".

3.4.2 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

x and y, can be quantified using cosine similarity:

$$\text{cosine similarity}(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} \tag{6}$$

which is bounded between -1 (opposite in meaning) and $+1$ (same in meaning). Applying this metric to Table 1–2 we find the following pairs to be most similar:

1. Positive words: "bad" and "boring"
2. Negative words: "wonderful" and "enjoyable"
3. Positive phrases (length 2): "how bad" and "of bad"
4. Negative phrases (length 2): "moving and" and "and powerful"

3.4.3 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 structure of a positive sentence, which the greedy algorithm correctly determined. Meanwhile Figure 2 shows the structure of a negative sentence which the greedy algorithm performed poorly. Specifically, the structure should have connected "right now" with "go , girls" instead of "the reality drain .". Incorrectly determine tree structures such shown in Figure 2 likely caused problems for the RAE model and decreased it's prediction accuracy.

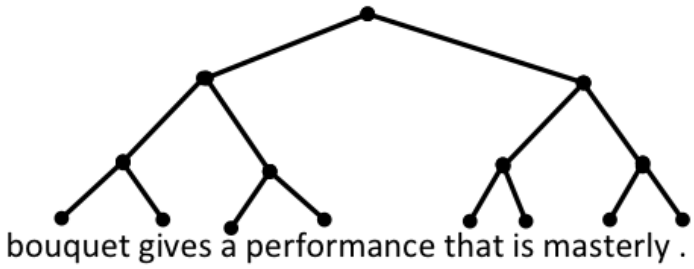

Figure 1: The tree structured determined by the greedy algorithm of a positive sentence.

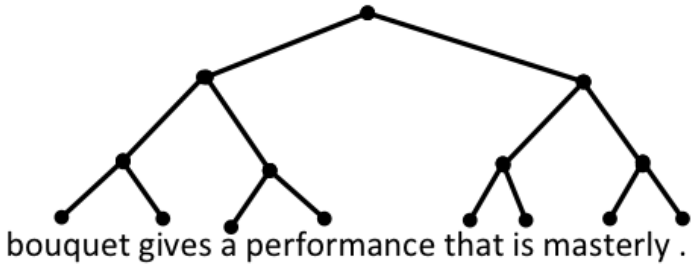

Figure 2: The tree structured determined by the greedy algorithm of a negative sentence.

4 Conclusion

Final remarks

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.