Sentiment Classification of Movie Reviews Using Recursive Autoencoders

CSE250B, Winter 2014, Project 4

Kaushik Kalyanaraman
UC San Diego
kkalyana@ucsd.edu

Garrett Rodrigues
UC San Diego
gbrodrig@ucsd.edu

March 17, 2014

Abstract

In this paper, we present an implementation of recursive autoencoders for the purposes of correctly classifying the sentiments in movie review exerpts. We train the model in a semi-supervised manner, and it learns both the vector representations for the meaning of words and phrases as well as the sentiment of the overall sentence. After training the neural network, we achieve a 68 % accuracy on the training data.

1 Introduction

One of the major challenges in machine learning is learning from unlabeled data. Clustering and other forms of pattern recognition are often used for such unsupervised learning tasks, and while they are useful for finding patterns, they do not always provide easily interpretable results. Supervised learning, in contrast, provides a distinct learning goal, such as predicting the value of housing prices or classifying a tumor as benign or malignant, but it is often difficult, sometimes prohibitively so, to attain labeled data.

Semi-supervised learning is an attempt to learn information from a dataset without having labels for *all* the information that we are trying to learn. In this paper, we attempt to use a recursive autoencoder (RAE) to learn a vector representation of the meanings of individual words and phrases so that we can predict the overall sentiment of movie review sentences. The input dataset already has the sentences separated into those with positive sentiments and those with negative sentiment, but it does not contain any target value or label for the meanings of the individual words or phrases. Thus, the learning algorithm is only semi-supervised.

A significant advantange of recursive autoencoders in this context is their ability to learn the structure and complex semantics present within the English language. The input to the algorithm is a sentence, which can be thought of a structured sequence of words. The algorithm itself learns which words or phrases belong together in order to create a sentence with the correct meaning. Other classification algorithms that use a Bag of Words reprentation for text input lose all information about the order of the words. While the words themselves may seem more important than the ordering, consider of the following as an example of the importance of the ordering of words. The sentences "The scene's dialogue was surprisingly fresh" and "Nothing was suprising or fresh with regards to the scene's dialogue" share very similar bag of words representations but their meanings differ greatly.

In this paper, we use recursive autoencoders to learn the sentiment of the movie reviews using real valued input vectors $\in R^{20}$ as the representation of each word. We apply our RAE to a dataset of 10,662 sentences critiquing various films. The sentences are of varying length and styles, and we build a vocabulary of words used in the dataset that contains approximately 260,000 words. We represent each of these words with a 20-dimensional vector that serves to capture the word's meaning. In addition to learning the sentiment of the overall sentence, the autoencoder also learns the meaning vectors of each word and is capable of internally clustering groups of similar or related words so as to make good final

predictions about the sentiment of the sentence. Using this approach, we train a model using 4-fold cross validation that correctly predicts the sentiment of sentences from movie reviews with 68% accuracy. We compare this with a bag of words classification model which achieves a test accuracy of 66 %.

2 Design and Analysis of Algorithms

2.1 Word Representations

We represent each word in a sentence as a d-dimensional vector, initialized randomly using a Gaussian with zero mean and standard deviation σ. Such a representation yields a vocabulary representation of $|V|$ d-dimensional vectors, where each vector represents the meaning of some word within the vocabulary. In the backpropagation section, we discuss the manner in which we update the meanings of individual words.

2.2 Binary Tree Construction

In order to train our autoencoder, we first need to define the structure of the neural network. We simplify our model by considering only binary representations. An example network is given in Figure 1. In order to describe we select a tree, we must first discuss how each node in the tree is formed and the objective we seek to optimize when merging any two nodes. The leaves of the tree represent the meanings of the individuals words of the sentence, and the other nodes represent two or more words or phrases concatenated together.

In Figure 1, consider the orange node closest to the leaves of the tree. That node represents the two words "always" and "seemed" merged together. When two nodes, always of dimension d, are merged together, it is done through the formula given in (1). W and b are parameters of the model, and W $\in \mathbb{R}^{dx2d}$ and b $\in \mathbb{R}^d$, and h is an elementwise activation function that is the same for all nodes in the model. In our model we use the *tanh* function. For each node we call the input to h() the *activation* of the node. Thus, each "parent" node has the same dimensionality as its two children, and its activation is the weighted sum of the meaning vectors of its two children added to a bias term. Because of this, internal nodes can be merged the same way as leafs until the root is reached.

In a good model, the meaning vector of a parent node encompasses the meaning of both children, and it should be possible to recover the meanings of both children from its parent. The process for this recovery is shown in (2). This is the unsupervised portion of the autoencoder, and at each merge, we are able to reduce the dimensionality of the inputs by a factor of 2, and we use only the inputs to validate the performance of our model. The performance of the reconstruction is evaluated using the equation for the reconstruction error shown in (5). This is the 2-norm of the difference between the recovered inputs and the true inputs, with a scaling factor for each contribution. In (5), n_1 and n_2 represent the number of words beneath each of the respective nodes about to be merged. If one of the nodes being reconstructed consists has 5 leaf nodes beneath it and the other only has 1, the the first node node is given 5 times more weight than the second node when evaluating the reconstruction error.

$$\bar{p} = h(W[\bar{c_1}; \bar{c_2}] + b) \tag{1}$$

$$[\bar{c_1}\prime; \bar{c_2}\prime] = U\bar{p} + c \tag{2}$$

Now, we have all the definitions needed in order to construct an optimal binary tree. For a given sequence of inputs $\bar{x}$, we would like to select the tree that produces the minimal reconstruction error overall non-leaf nodes as shown in 3. However, performing such a search for the optimal sequence of merges requires a search over the $(m-1)^{th}$ Catalan number of trees and therefore is infeasible to do repeatedly when we train our model.

$$[h!]RAE_\theta(\bar{x}) = \arg\min_{y \in A(x)} \sum_{s \in T(y)} E_{rec}(c_1; c_2)_s \tag{3}$$

Instead, we use a greedy search algorithm to find a good approximation to the optimal sequence of merges. The algorithm works as follows. For a sentence with m words, begin by computing the reconstruction error associated with each of the m-1 possible merges, and

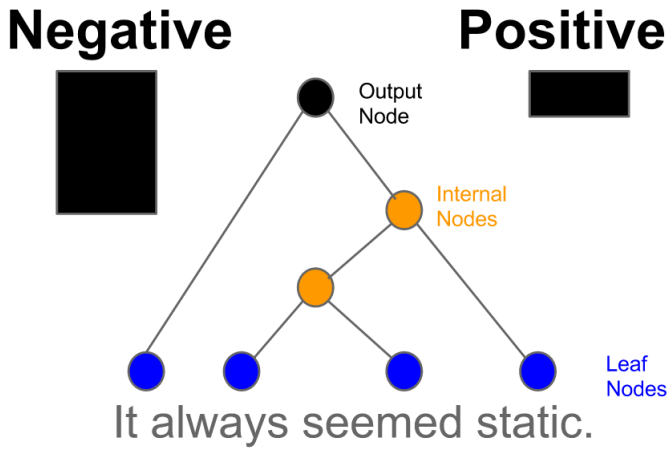


Figure 1: An example binary tree constructed from our algorithm. The four nodes representing the initial sentence are merged together until a single output node is a achieved, and the output is then passed to a softmax function to generate a sentiment prediction.

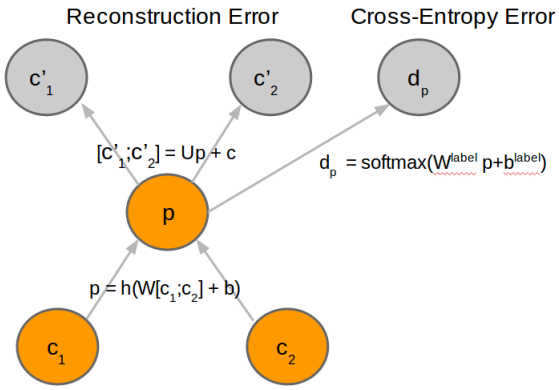


Figure 2: Illustration of a neural network with c_1 and c_2 as leaf nodes, p as internal node with reconstruction errors defined by output nodes c_1' and c_2'. Cross-entropy a.k.a. classification error indicated by output node d_p defined by the softmax function

merge the two nodes that produce the smallest error. Then replace the two nodes with the single merged node and repeat the above calculation and selection until only one node remains. This node represents the meaning of the overall sentence.

In other words, it first searches for the two nodes that most closely fit together (i.e. the two nodes that produce the reconstruction error) and then continues merging nodes until only a single root node remains, representing the entire tree.

$$E_{cE}(p,t;\theta) = -\sum_{k=1}^{K} t_k log d_k(p;\theta) \quad (4)$$

$$E_{rec}(c_1;c_2) = \frac{n_1}{n_1+n_2}||\bar{c_1}-\bar{c_1}'||^2 + \frac{n_2}{n_1+n_2}||\bar{c_2}-\bar{c_2}'||^2 \quad (5)$$

$$J = \sum_{allnodes-\{leafs\}} \alpha E_{rec}(c_1;c_2) + \sum_{allnodes} (1-\alpha)E_{cE}(p,t;\theta) + \frac{\lambda}{2}||\theta||^2 \quad (6)$$

2.3 Forward and Backward Propagation

After we have the structure of the tree defined, we must complete the prediction of target value, the movie sentiment, using the parameters W, b, W^{label} and b^{label}. To do this, we simply start at the leaves of the tree and calculate the value of $\bar{p}$ for every non-leaf node. When, we reach the root node, we use the softmax function

$$P(K|\bar{x}) = softmax(\bar{p}_{root}) \tag{7}$$

$$= \frac{exp([W^{label}\bar{p_{root}} + b^{label}]_k)}{\sum_k exp([W^{label}\bar{p_{root}} + b^{label}]_k)} \tag{8}$$

$$softmax(V_{xk}) = \frac{e^{V_{xk}}}{\sum_{i=1}^{n} e^{V_{xi}}} \tag{9}$$

where K is one of the two applicable classes, and $\bar{x}$ represents the input sequence of meaning vectors.

$$\gamma_i = \frac{\partial J}{\partial e_i} \tag{10}$$

$$= -\alpha \frac{n_1}{n_1+n_2} 2(\bar{c_1} - \bar{c_1}')^T \frac{\partial \bar{c_2}'}{\partial e_i} - \alpha \frac{n_2}{n_1+n_2} 2(\bar{c_2} - \bar{c_2}')^T \frac{\partial \bar{c_2}'}{\partial e_i} \tag{11}$$

$$= -\alpha h'(e_i)[\frac{n_1}{n_1+n_2} 2(\bar{c_1} - \bar{c_1}') + \alpha \frac{n_2}{n_1+n_2} 2(\bar{c_2} - \bar{c_2}')]^T \tag{12}$$

$$\varsigma = \frac{\partial J}{\partial \bar{g}} = (1-\alpha)(\vec{d} - \vec{t}) \tag{13}$$

Derivative of the error function J with respect to leaf nodes, $\bar{x}$ is:

$$\delta_{\bar{x}} = [V_{xt}^{1T} \delta_{\bar{q}} + W^{(label)^T} \varsigma] \tag{14}$$

The derivative of the un-normalized, pointwise hyperbolic tangent function, $tanh(\bar{v})_i = sech^2(v_i)$. The derivative with respect to the whole vector $\bar{v}$ is a matrix:

$$\frac{\partial g(\bar{v})}{\partial \bar{v}} = \begin{bmatrix} sech^2 v_1 & . & 0 \\ . & . & . \\ 0 & . & sech^2 v_d \end{bmatrix} \tag{15}$$

We use the formula from [5] to compute the derivative of the normalized transfer function, $\frac{tanh(\alpha)}{||tanh(\alpha)||}$, which is defined as follows:

$$f(\bar{v}) = \frac{g(\bar{v})}{|g^T g|^{1/2}} \tag{16}$$

The final weight updates we perform are:

$$\theta = \theta - (\frac{1}{N} \nabla J + \lambda \theta) \tag{17}$$

3 Results and Analysis

We consider three possible methods for training our RAE network, which are:

1. Including label classification error for computing the gradient of the loss function J, which is computed using the cross entropy error
2. Using a normalized transfer function $\frac{tanh(a)}{||tanh(a)||}$
3. Considering a bag-of-words model where no label classification error is computed

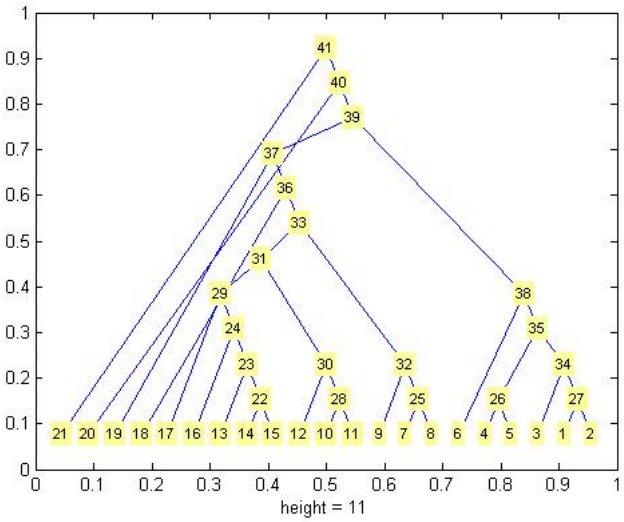
We hardcode values of $\alpha = 0.4$ and $\lambda = 0.01$ based on previously successful results, and each iteration takes about 5 minutes. Training our algorithm for approximately 70 iterations using L-BFGS takes nearly 6 hours. We evaluate the performance of the combinations of methods 1) and 2), and method 3) by itself. We obtain an accuracy of 95.43% on the training set with the combination of methods 1) and 2), using 4-fold cross-validation. For this configuration, we define the convergence condition to be when the change in the norm of the θ vector between iterations is less than 50. This model encodes the tree structure of input sentences and also considers the Jacobian matrix of the normalized transfer function to perform gradient updates.

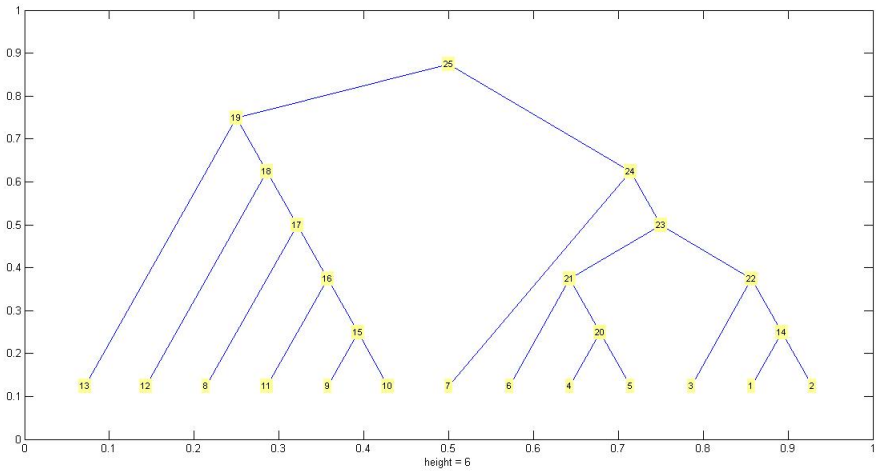
The reconstruction error is weighted by the α term in the expression of total error J and simply setting $\alpha = 0$ ensures that we consider a bag of words model, i.e., reconstruction error no longer affects the gradient but only classification or cross-entropy error at the leaf nodes. This model, in combination with the model parameters defined above, gives us an accuracy of 94.12% on the training set and 67.85% on the test set. We postulate that the bag of words model is almost as effective as the tree structure model because the sentiment of an entire sentence seems to most likely depend on the presence/absence of few influencing words and not on the specific order of these words in the sentence.

Top 10 positive	**Top 10 negative**
masterpiece	mess
compelling	tedious
promising	awful
likably	dreafdul
interesting	repellent
entertaining	entertainment
wonderful	stupid
poignant	mediocre
skillful	unsatisfying
brilliant	pretentious

Table 1: Top 10 most positive words and Top 10 most negative words.

We notice from the table above that the top 10 most positive and top 10 most negative words are intuitively and semantically correct to a human being, except for a single anomaly in the top 10 most negative. We compute the words by selecting those leaf nodes from sentences trees, which have the highest contribution to activation among all other nodes, and then comparing them across trees to obtain words in a descending order of their relative "importance".





Normalization is an essential part of recursive auto-encoder networks. When the σ function is saturated, we observe that derivative of the error function J with respect to model parameter θ is nearly zero. We end up learning a trivial neural network which does not consider label classification and merely tries to achieve a perfect input reconstruction. The zero derivatives obtained here cause no updates in the weight matrices and this model doesn't generalize at all. Forcing vectors to be normalized ensures that zero derivatives do not occur and ensures weight updation in each iteration.

We have not considered the effect of adding a slightly randomized noise to the weight update vectors, as has been done in Socher et. al.'s code. This random noise most probably leads to a more generalized approximation of the error function and hence is able to provide better accuracy.

3.1 Verifying Gradients

For each of the six parameters in the autoencoder, we analytically evaluate the gradient, and then in order to verify the correct computation of the gradient, we numerically evaluate the following definition of the derivative

$$\frac{\partial J}{\partial \theta} = \lim_{\epsilon \to \infty} \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} + O(\epsilon^2) \tag{18}$$

We then compare the value computed using our derived formulation with this numerical approximation and verify the correct implementation. For the W matrix, we initially found that we were not implementing the backpropagation algorithm correctly, and the numerical validation allowed us to correct the mistake. The final results are shown for all derivatives in agreement with their numerical approximation.

4 Findings and Lessons Learned

This paper highlighted the effectiveness of semisupervised learning algorithms. With only the positive or negative sentiments of movie sentences available, the recursive autoencoder was able to learn meaningful vector representations of each of the words in the vocabulary. Such an approach to training classifiers lends itself well to future applications, because it assumes no fixed structure for the input sentence and instead learns a good neural network structure. As a result, photographs, music and other forms of unstructured inputs might also perform well with such a model and it is something that we would like to investigate in the future.

5 References

[1] [1]Elkan, C. "Learning Meaning for Sentences" CSE250B. February 25, 2014

[2] Dongdong Zhang, Shuangzhi Wu, Nan Yang and Mu Li1, "Punctuation Prediction with Transition-based Parsing", in ACLWEB 2010

[3] Socher, R., Pennington, J. , Huang, E, et al. "Semi-Supervised Recursive Autoencoders for Predicting Distributions", SLAC National Accelerator Laboratory, 2011

[4] Schmidt, Mark. , "An Implementation of LBFG-S", in University of British Columbia, 2010

[4] Altwaijry, H., Lin Kuen-Han, Yamada, Toshiro. Mark. , "CSE 250B Project Assignment 4", in University of British Columbia, 2010