Recursive Auto Encoder (RAE) Method of Learning Meanings for Sentences
Learning Algorithms, Project 4

Mohsen Malmir, Erfan Sayyari

March 17, 2014

Abstract

In this paper, we develop a recursive auto encoder neural network for recognizing sentence level sentiment. A difficulty arises when input to neural networks is text sentence. In this paper, we choose a dynamic representation for sentences that is derived greedily to minimize the reconstruction error. This sentence level representation dictates the form of shape of the neural network. We also adapt a flexible representation for words that resembles the meaning of that word in a low dimensional space. The word representations update as recognition error back propagates through the auto-encoder neural network. We demonstrate the performance of the proposed model on the movie review dataset [5]. This is a dataset that contains "positive" and "negative" reviews of movies, each comprised of one or more sentences. Our model can produce comparable results on this dataset.

1 Introduction

One of the main challenges in text-related applications of machine learning is the representation of the data. In applications such as topic modeling, an input corpus is represented by a bag-of-words model, which ignores the basic property that comprises the text: order of the words. One way to deal with this is to use *n-grams*, which builds a representation of the text based on n-consecutive words in sentences. However, to avoid the curse of dimensionality, this approach is applied only with small values of n, which is insufficient in the applications such as extracting meaning of the text.

With increasing volume and importance of online reviews, predicting *sentiment* is becoming crucial for understanding the user generated content on websites. The baseline method for this task is to

use a bag-of-words representation [6]. These methods fail to capture the sentiment properly, as they ignore words order, a crucial component in analyzing text. For example in reviews about a movie, "A good movie from a bad director" has actually positive sentiment, but "A bad movie from a good director" has negative sentiment, considering the target is reviews about the movie.

In this paper, we develop a representation for text sentences that is based on individual words *meanings*. The meaning of sentence is constructed based on the meaning of individual words and their order in the sentence. This requires representation of meaning in someway, however is no standard way to do this without falling into philosophical issues. We choose a d-dimensional real valued vector for to represent meaning of parts of text. A good candidate to combine the meaning of words to get sentence level meaning is to use the sentence's parse tree. However this requires use of some external text processing package which means the results can't be generalized to other languages. In this paper, we introduce a greedy approach to combine meanings of parts of text based on the concept of auto encoder neural network. The benefits of this approach is its speed and that it can be generalized to other languages with no extra effort. We show that this greedy method is capable of producing state of the art results for sentiment prediction task on a movie review dataset.

The main contributions of this paper are introducing a compositional structure to represent meaning of text and developing a greedy method to implement this structure. A recursive auto encoder neural network is used to combine the meanings of parts of sentence by minimizing the reconstruction error. At the top level, meaning of sentence is used in a *softmax* classifier to predict the sentiment. We show that this model can achieve state-of-the-art performance on sentiment prediction of the movie review dataset.

2 Proposed Model

2.1 Overal Architecture

An exemplar architecture of the proposed model is shown in figure 1. In this figure, the input sentence has 4 words which are shown in the bottom level of the network with blue nodes. Each word is represented by a d-dimensional vector representing its meaning. The orange nodes indicate the composition nodes which are constructed by *encoding* the meaning of two words of parts of sentence. Matrix W^1 with size $2d \times d$ at each orange node indicates the meaning-composition stage of the network. For the composition of meanings to be appropriate, a *reconstruction* stage is added that is shown by violet nodes. Matrix W^2 with size $d \times 2d$ is used to reconstruct the original meaning vectors from a composite meaning. Note that the same matrices are used in all levels of encoding and reconstruction in the network. A prediction stage is added to the top level meaning node of the network, shown in green. For a target value of dimensionality k, a prediction matrix W^{label} with size $d \times k$ is used which transforms the meaning to the input of a softmax function with k target classes.

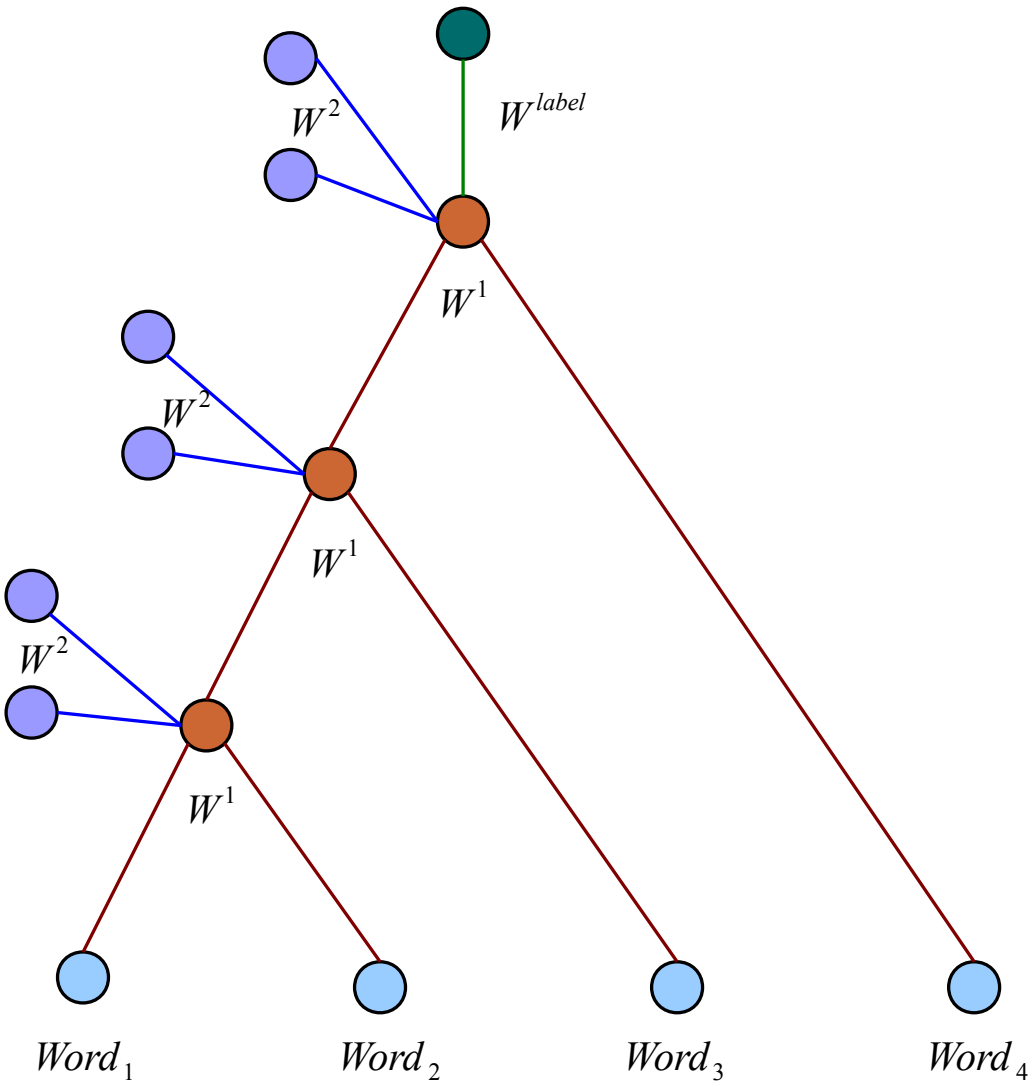

Figure 1: An exemplar schematic diagram of recursive auto-encoder network developed in this paper. Blue nodes indicate the words of the input sentence, the orange nodes indicate the intermediate coding of meaning of parts of sentence. The violet nodes are the reconstruction of input nodes, and the green node is the predicted target. The weight matrices for encoding, reconstruction and label prediction are indicated correspondingly by W^1, W^2 and W^{label}.

A more detailed view of the encoding and reconstruction stages is shown in figure 2. This is called an auto-encoder network. The goal of encoding is to produce a d-dimensional representation Z from two d-dimensional meaning vectors, shown by X and Y. The encoded representation Z is calculated by,

$$Z = h(W^1[X;Y] + b^1) \tag{1}$$

where W^1 is a $d \times 2d$ matrix, b^1 is a $d \times 1$ vector, h is a point-wise thrashing function $h : \mathbb{R}^d \to \mathbb{R}^d$ and $[X;Y]$ indicates the column-wise concatenation of vectors X and Y, that is X and Y are $d \times 1$ vectors and $[X;Y]$ is a $2d \times 1$ vector. The purpose of encoding stage is to produce a representation that is the composite of meanings of individual elements of the sentence. By using only two inputs for encoding, we are putting constraints on the model that simplifies the implementation. One should note that though the meaning of a phrase is well defined in traditional view of meaning, longer expression's or sentence's meaning that is acquired in our model is only interpretable in the context of overall goal of the network, which is in this case predicting the sentiment.

Having a composite meaning vector Z, a measure of goodness of encoding is to reconstruct the original meaning vectors by an inverse transformation. This is shown using blue lines and violet nodes in figure 2. Specifically, having vector Z, we can transform it back to reconstructions X' and Y' using,

$$[X', Y'] = W^2 Z + b^2 \tag{2}$$

where W^2 is a $2d \times d$ matrix, b^2 is a $2d \times 1$ vector and X' and Y' are $d \times 1$ vectors. The encoding and decoding stages together comprise a single orange node in figure 1. Note that the same parameters (W^1, W^2, b^1, B^2 and h) are used in different orange nodes of the tree. Starting from the lowest orange node, a compact meaning is derived from its inputs, which is then fed to the orange node in the next level to be combined with other compact meanings of the parts of sentence. This is repeated until a single d dimensional vector representation is produced for the sentence. For now, consider that the structure of the tree is given. Later in this section, we will delve into the details of calculating the structure of tree for different sentences.

2.2 Reconstruction Error

The goal of the auto-encoder network is to produce a compact representation of the inputs, which is good enough to be used to decode the original vectors. We define the *reconstruction error* of the auto-encoder as,

$$E_{rec}(X, Y, X', Y') = \frac{n_X}{n_X + n_Y}||X - X'||^2 + \frac{n_Y}{n_X + n_Y}||Y - Y'||^2 \tag{3}$$

Here, n_X and n_Y are the length of the parts of the input sentence that are encoded in X and Y correspondingly. For leaf nodes, n_X is equal to 1. For non-leaf nodes, $n_X = n_{lc} + n_{rc}$ where n_{lc} and

n_{rc} are the corresponding n for left and right child of X. We use the reconstruction error to train the parameters W^1, W^2, b^1, b^2.

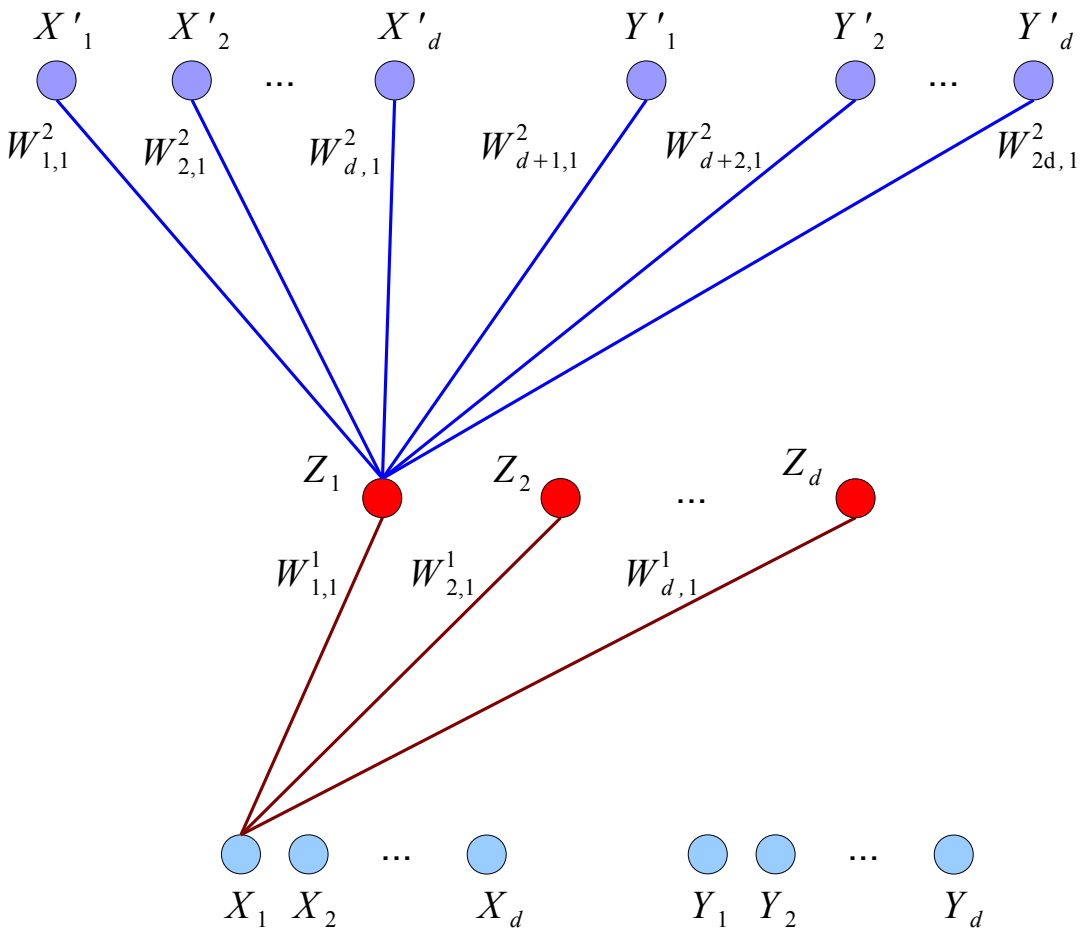

Figure 2: Encoding and Decoding stages of the recursive auto-encoder network. There are two d-dimensional inputs to the network. The encoding part which is indicated by matrix W^1 and red-colored edges, embeds the inputs to a d-dimensional representation. The decoding stage is represented by blue color edges and matrix W^2, aims to reconstruct the original inputs from the encoded representation. Only one column of W^1 and W^2 are shown for clarity.

2.3 Predicting Sentiment

We use the top level meaning to predict the sentiment of the sentence. This is indicated by the green edge and node in figure 1. Let Z denote the top-level meaning of the sentence, then

$$V = \text{softmax}(W^{label} Z) \tag{4}$$

denotes the predicted sentiment for the input sentence. W^{label} is a $K \times d$ matrix, which transforms the d-dimensional meaning vector into a K-dimensional vector. The softmax function for an input of size K is defined as,

$$V_k = \frac{e^{W^{label}_{k,:} Z}}{\sum_{i=1}^{K} e^{W^{label}_{i,:} Z}} \tag{5}$$

where $W^{label}_{j,:}$ indicates the jth row of the matrix W^{label}. Each training sentence is accompanied by a vector T of length K of target values. We use the cross entropy between the target value and the predicted value as a measure of error,

$$E_{CE}(T, V|\theta) = -\sum_{i=1}^{K} -T_i \log V_k \tag{6}$$

here, θ indicates the model. We use back propagation to update the model parameters.

2.4 Greedy Construction of Neural Network Structure

Up to now, we assumed that the tree structure of the neural network is given. In this section, we develop a greedy algorithm to construct this tree. In order to understand this step clearly, we should note that the tree structure dictates the way the meaning of sub-parts of each sentence are combined to produce the final meaning. There is no unique tree structure that can include the meaning combinations of sub-parts of all sentences. Therefore, each sentence should have a specific tree that is built based on its constituent parts. An appealing choice here is to use the parse trees of the sentences, by using some 3rd party text parsing package. However, there is a subtlety we should be aware of: the optimal combination of meaning of sub-parts of a sentence depends on the application. Therefore, a parse tree for a sentence might not be the perfect solution to our problem.

In the greedy method we propose here, the reconstruction error in (3) is used to decide what sub-parts of a sentence to combine in each step. We start by individual words at the first level, then choose the two words to be combined that have the lowest reconstruction error, as given in (3). In the next level, the same step happen, except that the two combined sup-parts are replaced by the result of encoding, as given by (1). We repeat this step until a single meaning vector is calculating for the sentence. The exact algorithm is given in 1. Note that for a tree constructed in this way for a sentence of length n, there are n leaf and $n-1$ non-leaf nodes.

Algorithm 1: Greedy Algorithm For Building the Tree for a Sentence.

```
# Input: meaning of words of the sentence < x_1, x_2, ..., x_n >
# Output: Optimal Tree R
Nodes={x_1, x_2, ..., x_n}
while Nodes.remaining > 1 do
    MinErrorFound = inf
    j = -1
    ConcatenatingNode = null
    for i = 1 to Nodes.remaining - 1 do
        Let X, Y ← x_i, x_{i+1}
        Compute p ← h(W^1[X; Y] + b^1)
        Compute [X'; Y'] ← W^2 p + b^2
        Error ← E_rec([X; Y], [X'; Y'])
        if Error < MinErrorFound then
            MinErrorFound ← Error
            j ← i
            ConcatenatingNode ← p
        end
    end
    ConcatenatingNode.leftChild ← Nodes(j)
    ConcatenatingNode.rightChild ← Nodes(j + 1)
    Nodes ← Nodes − {Nodes(j), Nodes(j + 1)} + {ConcatenatingNode}
end
R.root ← ConcatenatingNode
```

2.5 Learning Model Parameters

To learn the parameters $W^1, W^2, b^1, b^2, W^{label}$, we use the reconstruction error (3) and cross entropy error (6). We define the error of sentence x with the tree structure $R(x)$and target vector T as a combination of these,

$$E(R(x), T) = \sum_{k \in R(x)} \alpha E_{rec}(k) + (1 - \alpha) E_{CE}(k, T) \tag{7}$$

where k is over all nodes of the tree $R(x)$. For a non-leaf node k in $R(x)$ with left child X and right child Y, the reconstruction error is calculated using (2) and (3). For leaf nodes, reconstruction error is 0. Cross entropy error for all nodes is calculated using back-propagation, which is described later.

The objective function to minimize over training set $S = \{(x_1, T_1), (x_2, T_2), \ldots, (x_m, T_m)\}$ is,

$$J = \frac{1}{m} \sum_{<x,T> \in S} E(R(x), T|\theta) + \frac{\lambda}{2}||\theta||^2 \tag{8}$$

where $\theta =< W^1, W^2, b^1, b^2, W^{(label)} >$. The partial derivative of the loss function J with respect to parameters θ is,

$$\frac{\partial J}{\partial \theta} = \frac{1}{m} \sum_{<x,T> \in S} \frac{\partial E(R(x), T|\theta)}{\partial \theta} + \lambda\theta \tag{9}$$

2.6 Backpropagation

For different nodes in the network, we use the same notation to indicate the nonlinear activation function,

$$p = h(a) = h([W^{(1)}b^{(1)}][c_1; c_2; 1]) \tag{10}$$

$$e = [\hat{c}_1; \hat{c}_2] = [W^{(2)}b^{(2)}][p; 1] \tag{11}$$

$$r = softmax(a^{output}) = softmax([W^{label}b^{label}][p_r; 1]) \tag{12}$$

where r is the predicted probabilities for the target classes. Here p_r is the meaning vector for the output node of the network. We concatenate $W^{(1)}$ with $b^{(1)}$ and $W^{(2)}$ with $b^{(2)}$ to simplify the notation in the following analysis,

$$W^{(1*)} = [W^{(1)}b^{(1)}] \tag{13}$$

$$W^{(2*)} = [W^{(2)}b^{(2)}] \tag{14}$$

We can write r_j, the j-th component of r as,

$$r_j = f_j((W^{label}p_r)_1, (W^{label}p_r)_2, \ldots, (W^{label}p_r)_K) = \frac{e^{(W^{label}p_r)_j}}{\sum_{i=1}^{K} e^{(W^{label}p_r)_j}} \tag{15}$$

$$r = \begin{pmatrix} f_1(\sum_{j=1}^{d} W_{1j}^{label} p_{r_j}, \sum_{j=1}^{d} W_{2j}^{label} p_{r_j}, \ldots, \sum_{j=1}^{d} W_{Kj}^{label} p_{r_j}) \\ f_2(\sum_{j=1}^{d} W_{1j}^{label} p_{r_j}, \sum_{j=1}^{d} W_{2j}^{label} p_{r_j}, \ldots, \sum_{j=1}^{d} W_{Kj}^{label} p_{r_j}) \\ \vdots \\ f_K(\sum_{j=1}^{d} W_{1j}^{label} p_{r_j}, \sum_{j=1}^{d} W_{2j}^{label} p_{r_j}, \ldots, \sum_{j=1}^{d} W_{Kj}^{label} p_{r_j}) \end{pmatrix} \tag{16}$$

Let $a^{output} = W^{label} p_r$ be a vector in $\mathbb{R}^K$. We can rewrite r as:

$$r = \begin{pmatrix} f_1(a_1^{output}, a_2^{output}, \ldots, a_K^{output}) \\ f_2(a_1^{output}, a_2^{output}, \ldots, a_K^{output}) \\ \vdots \\ f_K(a_1^{output}, a_2^{output}, \ldots, a_K^{output}) \end{pmatrix} \tag{17}$$

We can write:

$$p_r = h(W^{(1)}[c_1; c_2] + b^{(1)}]) \tag{18}$$

where h is a pointwise nonlinear function. We need the partial derivatives of the loss function,

$$\frac{\partial J}{\partial W_{ij}^{(1*)}} = \frac{\partial J}{\partial a_i} \frac{\partial a_i}{\partial W_{ij}^{(1*)}} = \delta_i \frac{\partial a_i}{\partial W_{ij}^{(1*)}} \tag{19}$$

$$\frac{\partial a_i}{\partial W_i^{(1*)}} = [c_1; c_2; 1]^T \tag{20}$$

$$\frac{\partial J}{\partial W_{ij}^{(2*)}} = \frac{\partial J}{\partial e_i} \frac{\partial e_i}{\partial W_{ij}^{(2*)}} = \gamma_i \frac{\partial e_i}{\partial W_{ij}^{(2*)}} \tag{21}$$

$$\frac{\partial e_i}{\partial W_i^{(2*)}} = [x_p; 1]^T \tag{22}$$

$$\frac{\partial J}{\partial W_{ij}^{label}} = \frac{\partial J}{\partial a_i^{output}} \frac{\partial a_i^{output}}{\partial W_{ij}^{label}} = \zeta_i \frac{\partial a_i^{output}}{\partial W_{ij}^{label}} \tag{23}$$

$$\frac{\partial a_i^{output}}{\partial W^{label}} = [x_r; 1]^T \tag{24}$$

γ_i is related to error of reconstruction, E_{rec}:

$$\gamma_i = \frac{\partial J}{\partial e_i} = \alpha \frac{n_1}{n_1 + n_2} 2(c_1 - \hat{c}_1)^T (-\frac{\partial \hat{c}_1}{\partial e_i}) + \alpha \frac{n_2}{n_1 + n_2} 2(c_2 - \hat{c}_2)^T (-\frac{\partial \hat{c}_2}{\partial e_i}) \tag{25}$$

$$\frac{\partial [\hat{c}_1; \hat{c}_2]}{\partial e_i} = 1 \tag{26}$$

$$\gamma_i = -\alpha [\frac{n_1}{n_1 + n_2} 2(c_1 - \hat{c}_1); \frac{n_2}{n_1 + n_2} 2(c_2 - \hat{c}_2)] \tag{27}$$

where $\delta_i = \frac{\partial J}{\partial a_i^{(r)}}$ contains all nonlinearity. Now we can write:

$$\frac{\partial J}{\partial a_i^{(r)}} = \sum_{k=1}^{K} \frac{\partial J}{\partial r_k} \frac{\partial r_k}{\partial a_i^{(r)}} \tag{28}$$

We have that $\frac{\partial J}{\partial r_k} = \frac{-t_k}{r_k}$, and we want to calculate $\frac{\partial r_k}{\partial a_i^{(r)}}$:

$$\frac{\partial r_k}{\partial a_i^{(r)}} = \frac{\partial f_k}{\partial a_i^{(r)}} = \sum_{t=1}^{K} \frac{\partial f_k}{\partial a_t^{output}} \frac{\partial a_t^{output}}{\partial a_i^{(r)}} \tag{29}$$

The output node with values r, the contribution to J is with the error of classification E_{CE}.

$$\zeta_i = \frac{\partial J}{\partial a_i^{output}} = (1-\alpha) \frac{\partial}{\partial a_i^{output}} \sum_{l=1}^{K} t_l \log r_l \tag{30}$$

If $l = k$:

$$\frac{\partial f_i}{\partial a_i^{output}} = r_i(1 - r_i) \tag{31}$$

and if $l \neq i$

$$\frac{\partial f_i}{\partial a_l^{output}} = -r_i r_l \tag{32}$$

So we could calculate ζ_i as:

$$\zeta_i = -(1-\alpha) \sum_{l, l \neq i}^{K} \frac{t_i}{r_i}(-r_l r_i) - (1-\alpha)\frac{r_i}{1-r_i} = (1-\alpha)(r_i - t_i) \tag{33}$$

For the internal values p, they contribute to loss J through the $[\hat{c}_1; \hat{c}_2]$, r:

$$\delta_i = \frac{\partial J}{\partial a_i} = \sum_{k=1}^{d} \frac{\partial J}{\partial a_k} \frac{\partial a_k}{\partial a_i} + \sum_{k=1}^{2d} \frac{\partial J}{\partial e_k} \frac{\partial e_k}{\partial a_i} + \sum_{k=1}^{K} \frac{\partial J}{\partial a_k^{output}} \frac{\partial a_k^{output}}{\partial a_i} \tag{34}$$

$$= \sum_{k=1}^{d} \delta_k \frac{\partial a_k}{\partial a_i} + \sum_{k=1}^{2d} \gamma_k \frac{\partial e_k}{\partial a_i} + \sum_{k=1}^{K} \zeta_k \frac{\partial a_k^{output}}{\partial a_i} \tag{35}$$

Now we need to compute $\frac{\partial a_k}{\partial a_i}$, $\frac{\partial a_k^{output}}{\partial a_i}$, and $\frac{\partial e_k}{\partial a_i}$:

$$\frac{\partial a_k}{\partial a_i} = \frac{\partial}{\partial a_i}[W^{(1*)}[\ldots; h(a); \ldots; 1]]_k = V_{ki}h'(a_i) \tag{36}$$

where V_{ki} is the part of $W^{(1*)}$ that multiplies a.

$$\frac{\partial e_k}{\partial a_i} = \frac{\partial}{\partial a_i}[W^{(2*)}[h(a); 1]]_k = W_{ki}^{(2*)}h'(a) \tag{37}$$

$$\frac{\partial a_k^{output}}{\partial a_i} = \frac{\partial}{\partial a_i}[W^{label}[h(a); 1]]_k = W_{ki}^{label}h'(a) \tag{38}$$

$$\delta_i = \sum_{k=1}^{d} \delta_k V_{ki} h'(a_i) + \sum_{k=1}^{2d} \gamma_k W_{ki}^{(2*)} h'(a_i) + \sum_{k=1}^{K} \zeta_k W_{ki}^{label} h'(a_i) \tag{39}$$

$$= h'(a_i)(\sum_{k=1}^{d} \delta_k V_{ki} + \sum_{k=1}^{2d} \gamma_k W_{ki}^{(2*)} + \sum_{k=1}^{K} \zeta_k W_{ki}^{label}) \tag{40}$$

For the leaves we could write:

$$\frac{\partial J}{\partial x_i} = \delta_{x_i} \tag{41}$$

2.7 Calculating derivatives for normalized vectors

We derive the above equation for unnormalized vectors. For the normalized vectors and function we have:

$$h(a) = \frac{\tanh(a)}{||\tanh(a)||} \tag{42}$$

$$h'(a) = \begin{bmatrix} sech^2(a_1) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & sech^2(a_d) \end{bmatrix} \times \left[\frac{1}{||tanh(a)||} I - \frac{1}{||tanh(a)||^3} tanh(a) tanh(a)^T\right] \tag{43}$$

2.8 Summarizing Derivatives

for the output nodes $\hat{c}_1$, $\hat{c}_2$, and r:

$$\gamma = \frac{\partial J}{\partial e} = \left[-2\alpha\frac{n_1}{n_1+n_2}(c_1 - \hat{c}_1); -2\alpha\frac{n_2}{n_1+n_2}(c_2 - \hat{c}_2)\right] \tag{44}$$

$$\zeta = \frac{\partial J}{\partial a^{output}} = (1-\alpha)(r-t) \tag{45}$$

For the internal nodes,p we have:

$$\delta = h'(a).\left[W^{(1*)^T}\delta_q + W^{(2*)^T}\gamma + W^{(label)^T}\zeta + \frac{\partial E_rec(p,\hat{p})}{\partial p}\right] \tag{46}$$

For the leaf nodes, x we have:

$$\delta_x = \left[W^{(1*)^T}\delta_q + W^{(label)^T}\zeta + \frac{\partial E_rec(x,\hat{x})}{\partial x}\right] \tag{47}$$

So the derivatives for the parameters are equal to:

$$\frac{\partial J}{\partial W^{(1*)}} = \delta[c_1; c_2; 1]^T \tag{48}$$

$$\frac{\partial J}{\partial W^{(2*)}} = \delta[p; 1]^T \tag{49}$$

$$\frac{\partial J}{\partial W^{(label)}} = \delta[p; 1]^T \tag{50}$$

2.9 Training Algorithm

To train the network, we start at leaves and perform a forward propagation to calculate the predicted label for the input. Then the derivatives and δ values are calculated and back prpagated from the root to the leaves. We start with a random initialization of parameters, then we update them after one epoch. The complete algorithm for training the model is given in algorithm 2.

3 Design of Experiment

In this section, we perform a series of experiments to validate the proposed model. First, we perform test to validate the correctness of derivatives calculated in the previous section. We use the numerical gradients calculated using the following equation as the reference,

$$\frac{\partial J}{\partial \theta} = \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon} + O(\epsilon^2). \tag{51}$$

For all experiments described below, we fix a set of parameters that are used in [4] . The dimensionality of meaning in all models is $d = 20$. The α parameter is set to 0.2, and the weight decay parameter

Algorithm 2: Training the Recursive Autoencoder

```
# Input: Training Examples < x_1, x_2, ..., x_n >
# Output: Training Model
θ =< W^(1), b^(1), W^(2), b^(2), W^label, Leaves >
#Initialization Initialize word meaning vectors x_i ~ N(0, σ^2)
Initialize matrices W^(1), W^(2), W^label, b^1, b^2 randomly.
while Not Converged or Exceeds the Maximum Number of Iteration do
    Initialize Gradient ▽J ← 0 for all x_i do
        Compute Tree = RAE_θ(x_i)
        Compute Gradient ▽J_i ← ∂J(x_i)/∂θ
        Update Total Gradient ▽J ← ▽J + ▽J_i
    end
    Update θ ← (1/N) ▽ J + λθ
end
```

is set to $\lambda = 1e-5$. Maximum number of epochs is 100. Most of the reviews contain more than one sentence. For these examples, we averaged the final prediction of the network for different sentences as the output for the entire text. The minimization was done using LBFGS method to minimize the cost function given in (7).

We perform 4 series of experiments. The first two are with softmax classification function and normalized vectors, one with leaf update and the other without leaf update. The second two experiments are the same except that we use with tanh activation function to predict the labels. For each experiment, we show the norm of gradient for different epochs as a sign of convergence of the algorithm. The recognition rate of the model for each approach is demonstrated on the test set, which is 1/3 of the movie review dataset. This is a dataset containing 10662 movie reviews, equally distributed between positive and negative classes. And finally, we show the most positive and negative words learned for the version where the leaves are updated.

4 Results

4.1 Gradient Check

Figure 3 shows the absolute value of the difference between numerical derivatives and derivatives we claculated in this paper. This figure shows the absolute difference value as a function of ϵ for networks with softmax and tanh activation function. It is obvious that the derivatives are correct in comparison

to the numerical derivatives for small values of ϵ.

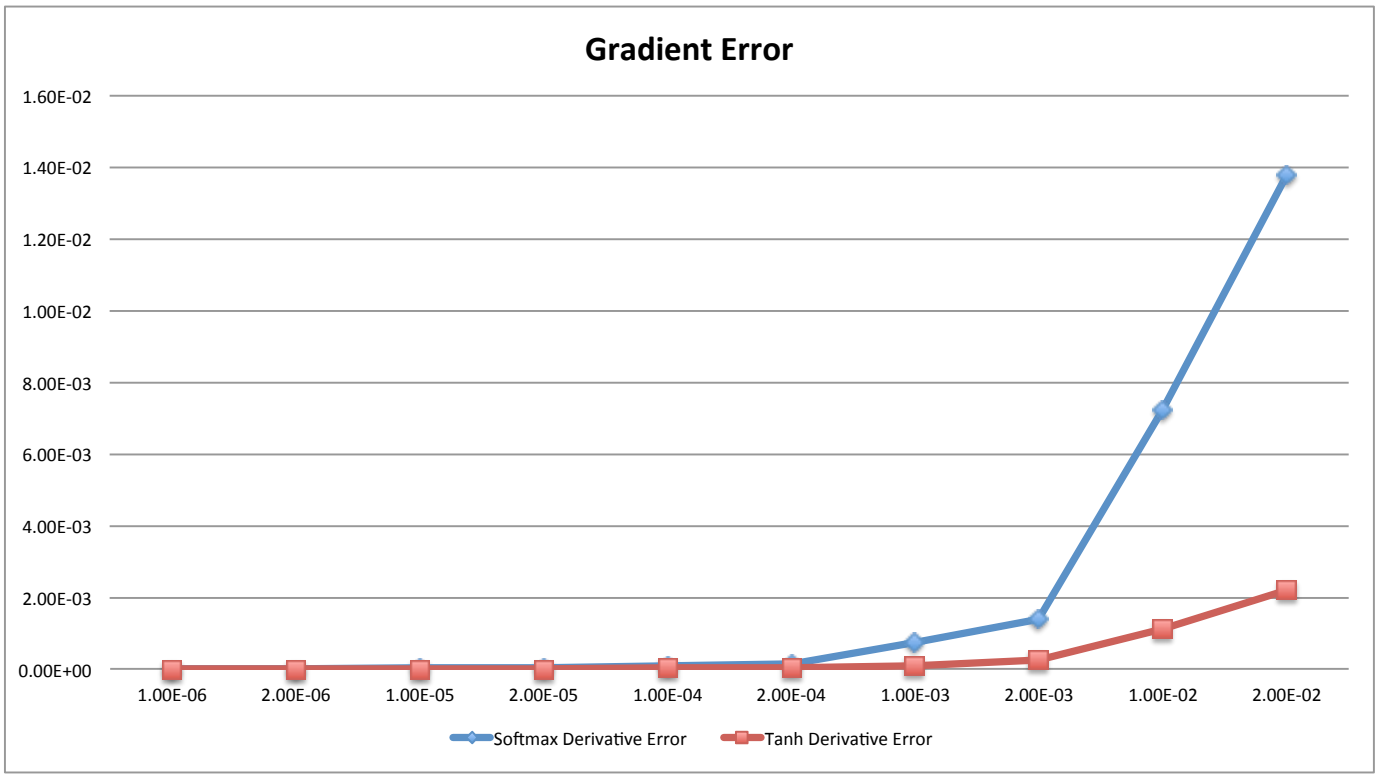

Figure 3: Absolute difference between the numerical gradient and derivatives we derived in this paper.

4.2 Convergence of Model Training

Figure 4 shows the convergence of the algorithm for different configurations of the model. It is clear that all models converge in less than 50 epochs, where the gradient falls below $1e-5$. The rate of convergence for different models is different, with the models with leaf update having some fluctuations. This is reasonable since the models with leaf update change the representation of the input and explore larger areas in the search space.

4.3 Recognition Performance on Test Set

The results of recognition are shown in table 1. As one can see, there is a large margin between models with leaf update and models without leaf update. The model with softmax activation function and leaf

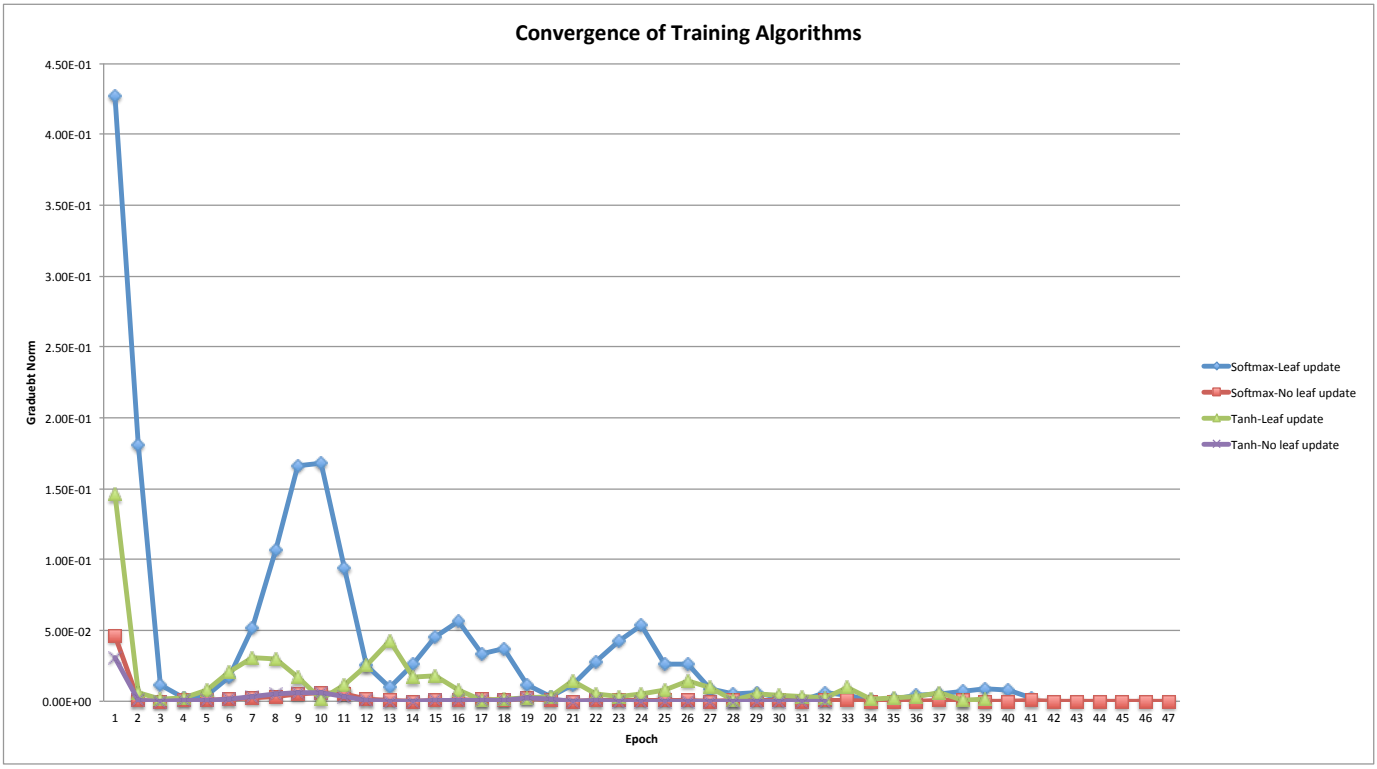

Figure 4: Convergence of different models based on the number of epochs.

update achieves the highest accuracy, while the model with tanh activation and leaf update achieves 69% accuracy on the test set. The model with softmax activation and no leaf update performs better on classification task compared to the model with tanh activation and leaf update. The lowest recognition rate is achieved by tanh activation function and no leaf update, which is about 67%.

Model	**Accuracy of test set recognition(%)**
softmax- Leaf update	74.3
softmax- no leaf update	71.4
Tanh- leaf update	69.0
Tanh- no leaf update	67.1

Table 1: Recognition rate for predicting the correct topic for each document for classic400 dataset.

4.4 Most positive and most negative words

For each word in a sentence, we can perform the classification the same way we do for the root node of the tree. Then result is two probabilities of belonging to positive and negative class. We perform this for the entire training dataset, and pick 20 words with highest probability of belonging to positive and negative classes. We do this for two models: softmax activation with leaf update and tanh activation with leaf update. The results are shown in table 2 for positive class and table 3 for negative class. The positive words mostly include cheers and expression of support, while the negative words contain words that aim to belittle movies.

Model	Top 20 Positive Words
Softmax Classifier	performances, moving, cinema, portrait, touching, powerful, best, culture, engrossing, beautiful, heart, family, solid, enjoyable, wonderful, warm, compelling, provides, rare, world
Tanh Classifier	delights, arresting, repressed, deceptively, exotic, duke, joyous, fluid, quaid, breathtakingly, reflective, infuses, immensely, expressive, uncomfortable, stark, jean, marching, indictment, 20th

Table 2: Top 20 words for positive class from movie review dataset.These words have the highest probability of belonging to positive class.

Model	Top 20 Negative Words
Softmax Classifier	bad, feels, boring, minutes, thing, nothing, script, worst, silly, or, video, was, title, only, mess, jokes, problem, fails, really, neither,
Tanh Classifier	or, only, into, if, who, characters, some, do, have, he, will, been, they, when, are, was, enough, make, their, we

Table 3: Top 20 words for negative class. These are the words that were classified to negative class with highest probabilities.

5 Discussion

In this paper, we developed a model for predicting sentiment of text. Predicting sentiment from input text is a challenging task because of many problems. It is difficult to represent individual words and

sentences in a way that is suitable for sentiment prediction. We showed that a novel *meaning*-based representation can be used to predict the text sentiment. The main contributions of this paper was the use of auto-encoder neural networks to extract the meaning of a sentence from the meaning of its constituent words. We used a dynamic structure for the network for different sentences that is derived from individual words in the sentence and their order.

One of the difficulties for the proposed model is to determine the convergence of training and to terminate early to avoid over fitting. Because of many free parameters in the model, the training can bounce forever. We used LBFGS minimization instead of stochastic gradient following to avoid such instabilities in convergence. Also there are local minima of the target function, for example where all meaning vectors are converged to 0. We used normalized vectors to represent the meaning in network nodes. Another difficulty arises when assigning meaning vectors to individual words. We solved this problem by back propagating the error to update the individual word representations.

References

[1] Porteous, Ian, et al. "Fast collapsed gibbs sampling for latent dirichlet allocation." Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.

[2] Newton, Michael A., and Adrian E. Raftery. "Approximate Bayesian inference with the weighted likelihood bootstrap." Journal of the Royal Statistical Society. Series B (Methodological) (1994): 3-48. APA

[3] Wallach, Hanna M., et al. "Evaluation methods for topic models." Proceedings of the 26th Annual International Conference on Machine Learning. ACM, 2009.

[4] Socher, Richard, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. "Semi-supervised recursive autoencoders for predicting sentiment distributions." In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 151-161. Association for Computational Linguistics, 2011.

[5] Pang and L. Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In ACL, pages 115124.

[6] Pang, L. Lee, and S. Vaithyanathan. 2002. Thumbs up? Sentiment classification using machine learning techniques. In EMNLP.