CSE 250B Project Assignment 4

Hani Altwaijry
haltwaij@cs.ucsd.edu

Kuen-Han Lin
kul016@ucsd.edu

Toshiro Yamada
toyamada@ucsd.edu

Abstract

The goal of this project is to implement the Semi-Supervised Recursive Autoencoders (RAE) with random word initialization and reproduce the result of Socher et. al [1] using the Movie Reviews dataset. With a correct understanding of gradient meaning in semi-supervised RAE, our implementation achieves 76.9% accuracy with only 0.1% difference to the authors' result. The experiments also show that including supervised information for each word's representation is important for RAE to succeed with random word initialization, while applying normalization to the reconstruction output nodes makes the algorithm less sensitive to the weighting between the supervised information and the reconstruction performance.

1 Introduction

Sentiment prediction through the employment of Semi-Supervised Recursive Autoencoders (RAE) is the goal of this project. Meaning of a sentence is represented using recursive structures and a high dimensional space where points represents certain notions of meaning. Sentence understanding is useful in many applications such as automated review sentiment extraction. The difference between this method and many other is the explicit use of sentence structure. For example, if we have sentences that have the same words but different meanings, such as "Cats chase mice." and "Mice chase cats.", and if we use a non-structured model to represent the words, e.g. the Bag of Words model, then the difference between the two sentences are lost. However, the way sentence structures represented in Recursive Autoencoders allows us to differentiate between such two sentences.

In this project, we attempt to reproduce the results of Socher et al. [1]. Specifically, we attempt to recreate the results achieved on the Movie Reviews dataset on which [1] achieved an accuracy of 76.8% using randomized initialized words.

2 Datasets

We use the Movie Reviews dataset used by Socher et. al [1]. The dataset is originally provided by Pang et. al. [2]. Although the latest version of the dataset is 2.0, the one used by Socher is 1.0. The dataset contains 10662 movie reviews which are split into 9596 training examples and 1066 test probes. The reviews are described as either positive or negative and are based of snippet sentences from the actual review that was obtained from the famous website Rotten Tomatoes.

3 Recursive Autoencoders

We present an overview of the recursive autoencoder (RAE) method. A thorough description can be found in [3]. Recursive autoencoders are neural networks that represent meanings of fixed-size inputs in reduced dimensional space. For example, each word in a sentence, such as "man" and "cats" are represented using a vector $\bar{x} \in \mathbb{R}^d$, and the RAE method reduces the entire sentence to a single vector of size $\mathbb{R}^d$. Sentences are sequences of words that are understood using a binary tree structure. The words are the leaves of the tree and their combined grouping is used to get a notion of the meaning of the sentence. Figure 1 shows an example of a tree representing a sentence. The internal nodes of the tree correspond to the combined meaning of the nodes below them. Each internal node is also represented in the same manner as individual words as vectors $\bar{p} \in \mathbb{R}^d$. These internal nodes are the hidden representation of the neural network. In the RAE model, the vocabulary is stored in an embedding matrix $L \in \mathbb{R}^{d \times |V|}$ where $|V|$ is the cardinality of the vocabulary. Typically, each word $\bar{x_i} \in L$ is initialized independently following a Gaussian distribution $x_i \sim \mathcal{N}(0, \sigma^2)$.

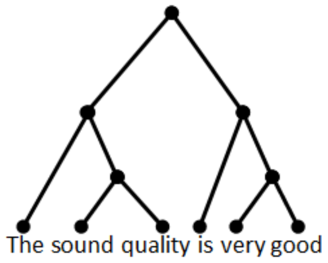


Figure 1: A binary tree structure representing a sentence "The sound quality is very good". This structure is an example of how a sentence could be represented by compositions of meanings.

3.1 Compositional Meaning

A meaning of a single internal node is a composition of two children according to the following equation:

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

where $\bar{p}$ is the combined meaning vector, $W^{(1)} \in \mathbb{R}^{d \times 2d}$ is a weight matrix governing the conversion, $[\bar{c_1}, \bar{c_2}]$ is a concatenated vector of the two children nodes, and $\bar{b}^{(1)} \in \mathbb{R}^d$ is a bias vector. Here $f(\cdot)$ is a point-wise non-linear function, such as a hyperbolic tangent. The goal of this non-linearity in the composition is to prevent the model from collapsing into a single matrix-vector linear operation.

The vectors $\bar{p}$ are constrained to $||\bar{p}|| = 1$ to avoid trivial autoencoder for the error function defined in section 3.4.

3.2 Autoencoding

When meaning is represented in a high-dimensional space, labels for meanings of words or phrases are not available at hand. Therefore, our goal would be to have words and phrases that are close in meaning be close in that high-dimensional space, and that the our combined meaning for two words or phrases can be used to reconstruct the simpler meanings from it. To achieve this, we require our model to be able to reproduce the meanings as close as possible to the original meaning (decoding) we had. The decoding operation is defined in equation (2). Therefore, the goal of the autoencoder is to minimize the difference between the originally encoded meaning and the decoded one.

$$[\bar{c_1}'; \bar{c_2}'] = f(W^{(2)}\bar{p} + \bar{b}^{(2)}) \tag{2}$$

where $[\bar{c_1}'; \bar{c_2}']$ is the reconstruction of the original meaning vectors, $W^{(2)} \in \mathbb{R}^{2d \times d}$ is a weight matrix governing the reconstruction, $\bar{p}$ is the combined meaning vector, and $\bar{b}^{(2)} \in \mathbb{R}^{2d}$ is a bias vector.

3.3 Class Distributions

We can use a semi-supervised recursive autoencoder model to predict class distributions to distinguish different topics of meaning, e.g. "positive review" or "negative review" in a "semi-supervised" fashion. Given labeled training data for K class distributions, we compute the distribution of topics at a given node $\bar{p}$ using the following equation:

$$\bar{d}(\bar{p}) = softmax(W^{(label)}\bar{p} + \bar{b}^{(label)}) \tag{3}$$

where $\bar{d} \in \mathbb{R}^K$ is the class distribution, $W^{(label)} \in \mathbb{R}^{K \times d}$ and $\bar{b}^{(label)} \in \mathbb{R}^K$.

3.4 Error Function

The error in our model is defined by two factors. First, we want our model to correctly reconstruct the original meaning vectors at node $\bar{p}$:

$$E_{rec}([\bar{c_1}, \bar{c_2}]_p, [\bar{c_1}', \bar{c_2}']_p) = \frac{n_1}{n_1 + n_2} \|\bar{c_1} - \bar{c_1}'\|^2 + \frac{n_2}{n_1 + n_2} \|\bar{c_2} - \bar{c_2}'\|^2 \tag{4}$$

in this equation, we also account for the number of nodes under each branch where n_1 is the number of nodes under $\bar{c_1}$ plus one and n_2 is similarly the number of nodes under $\bar{c_2}$ plus one.

Second, we want our model to correctly predict the distribution of K classes and we measure the error using the cross-entropy at node $\bar{p}$:

$$E_{cE}(\bar{p}) = -\sum_{k=1}^{K} t_k \log d_k(\bar{p}) \tag{5}$$

where d_k is the k-th element of $\bar{d}$, and t_k is the true distribution of label k.

3.5 Constructing the Binary Tree

We construct the binary tree representation of the sentence by selecting the tree that offers the minimum reconstruction error according to the following equation:

$$RAE_\theta(x) = \underset{y \in A(x)}{\arg\min} \sum_{s \in T(y)} E_{rec}([\bar{c_1}, \bar{c_2}]_s, [\bar{c_1}', \bar{c_2}']_s) \tag{6}$$

where $A(x)$ is the set of all possible trees for sentence x, and $T(y)$ a is the set of internal nodes in the binary tree y.

The greedy algorithm is described in [1, 3], and the following is a brief outline of the algorithm we implemented:

Algorithm 1 Constructing the Greedy Tree

```
Input: Words $\bar{x_1}, \bar{x_2}, ..., \bar{x_n}$
Output: Tree $T$

Let $Nodes = \{\bar{x_1}, \bar{x_2}, ..., \bar{x_n}\}$
while $Nodes.size > 1$ do
  $minError = \infty$
  $j = -1$
  $newNode = null$
  for $i \leftarrow 1$ to $i \leftarrow Nodes.size - 1$ do
    Compute $\bar{p} \leftarrow s(W^{(1)}[\bar{c_1}; \bar{c_2}] + \bar{b}^{(1)})$
    Compute $[\bar{c_1}'; \bar{c_2}'] \leftarrow f(W^{(2)}\bar{p} + \bar{b}^{(2)})$
    $error \leftarrow E_{rec}([\bar{c_1}, \bar{c_2}]_p, [\bar{c_1}', \bar{c_2}']_p)$
    if $error < minError$ then
      $minError \leftarrow error$
      $j \leftarrow i$
      $newNode \leftarrow p$
    end if
  end for
  $newNode.leftChild \leftarrow Nodes[j]$
  $newNode.rightChild \leftarrow Nodes[j+1]$
  $Nodes \leftarrow Nodes - \{Nodes[j], Nodes[j+1]\} + \{newNode\}$
end while
$T.root \leftarrow newNode$
```

3.6 Optimization Goal

The goal of recursive autoencoders is to minimize both errors across the entire nodes of the tree. Our objective function J for all sentences x and their respective tree t is:

$$\begin{aligned} J &= \frac{1}{N}\sum_{x,t} E(x,t;\theta) + \frac{\lambda}{2}\|\theta\|^2 \\ &= \frac{1}{N}\sum_{x,t}\{\alpha E_{rec}(x;\theta) + (1-\alpha)E_{cE}(x,t;\theta)\} + \frac{\lambda}{2}\|\theta\|^2 \end{aligned} \tag{7}$$

where θ are the parameters of the model, as follows:

$$\theta = \langle W^{(1)}, \bar{b}^{(1)}, W^{(2)}, \bar{b}^{(2)}, W^{(label)}, \bar{b}^{(label)}, L\rangle.$$

We calculate the gradient of the objective function J with respect to θ according equation (8) and using backpropagation as described in [3].

$$\frac{\partial J}{\partial \theta} = \frac{1}{N}\sum_{x,t}\frac{\partial E(x,t;\theta)}{\partial \theta} + \lambda\theta \tag{8}$$

The structure of the recursive autoencoder is shown in Figure 2. For each internal node, we have three output nodes. First, we have two reconstruction nodes $\bar{c_1}'$ and $\bar{c_2}'$. Second, a single class distribution node $\bar{d}$. In the network, we have added two artificial nodes to supply the values for the bias vectors, and to allow for the calculation of the partial derivative of the embedding matrix L. From Figure 2, we see that the weights affecting the network are exactly those of θ. We need to compute the partial derivative (8) for all the components of θ.

We have the following node values in the network:

$$\begin{aligned} \bar{p} &= f(\bar{a}) = f([W^{(1)}\ \bar{b}^{(1)}][\bar{c_1}; \bar{c_2}; 1]) \\ [\bar{c_1}'; \bar{c_2}'] &= f(\bar{e}) = f([W^{(2)}\ \bar{b}^{(2)}][\bar{p}; 1]) \\ \bar{d} &= softmax(\bar{g}) = softmax([W^{(label)}\ \bar{b}^{(label)}][\bar{p}; 1]) \end{aligned}$$

In these equations we combined $W^{(1)}$ and $\bar{b}^{(1)}$ into a single extended matrix to simplify the derivation. Similarly for $W^{(2)}$ and $\bar{b}^{(2)}$. We denote the extended matrices W^* as follows:

$$\begin{aligned} W^{1*} &= [W^{(1)}\ \bar{b}^{(1)}] \\ W^{2*} &= [W^{(2)}\ \bar{b}^{(2)}] \\ W^{label*} &= [W^{(label)}\ \bar{b}^{(label)}] \end{aligned}$$

W^{1*} affects the loss J through $\bar{a}$, and W^{2*} affects the loss J through $\bar{e}$. Similarly for W^{label*} through $\bar{g}$.

In our implementation, we used the normalized hyperbolic tangent function

$$f(\bar{v}) = \frac{tanh(\bar{v})}{\|tanh(\bar{v})\|} \tag{9}$$

In the following equations W^*_{ij} refers to element of W^* at the location $W^*(i,j)$ where i refers to the target dimension i affected by input vector dimension j.

$$\begin{aligned}
\frac{\partial J}{\partial W_{ij}^{1*}} &= \frac{\partial J}{\partial a_i}\frac{\partial a_i}{\partial W_{ij}^{1*}} = \delta_i \frac{\partial a_i}{\partial W_{ij}^{1*}} \\
\frac{\partial a_i}{\partial W_{i}^{1*}} &= [\bar{c_1}; \bar{c_2}; 1]^T \\
\frac{\partial J}{\partial W_{ij}^{2*}} &= \frac{\partial J}{\partial e_i}\frac{\partial e_i}{\partial W_{ij}^{2*}} = \gamma_i \frac{\partial e_i}{\partial W_{ij}^{2*}} \\
\frac{\partial e_i}{\partial W_{i}^{2*}} &= [\bar{p}; 1]^T \\
\frac{\partial J}{\partial W_{ij}^{(label)}} &= \frac{\partial J}{\partial g_i}\frac{\partial g_i}{\partial W_{ij}^{(label)}} = \zeta_i \frac{\partial g_i}{\partial W_{ij}^{(label)}} \\
\frac{\partial g_i}{\partial W_{i}^{(label)}} &= [\bar{p}; 1]^T
\end{aligned}$$

Output nodes with values $[\bar{c_1}'; \bar{c_2}']$ contribute to J with error function $E_{rec}(\cdot)$. For all dimensions, we find γ_i:

$$\begin{aligned}
\gamma_i &= \frac{\partial J}{\partial e_i} = \alpha\frac{n_1}{n_1+n_2}2(\bar{c_1}-\bar{c_1}')^T\left(-\frac{\partial \bar{c_1}'}{\partial e_i}\right) + \alpha\frac{n_2}{n_1+n_2}2(\bar{c_2}-\bar{c_2}')^T\left(-\frac{\partial \bar{c_2}'}{\partial e_i}\right) \\
\frac{\partial[\bar{c_1}'; \bar{c_2}']}{\partial e_i} &= f'(e_i) \\
\gamma_i &= -\alpha\left[\frac{n_1}{n_1+n_2}2(\bar{c_1}-\bar{c_1}'); \frac{n_2}{n_1+n_2}2(\bar{c_2}-\bar{c_2}')\right]^T f'(e_i)
\end{aligned}$$

For the output nodes with values $\bar{d}$, the contribution to J is with the error function $E_{cE}(\cdot)$. For dimensions $i = 1$ to K, we find ζ_i:

$$\begin{aligned}
\zeta_i &= \frac{\partial J}{\partial g_i} \\
&= (1-\alpha)\frac{\partial}{\partial g_i}\left(-\sum^{K} t_k \log d_k\right) \\
&= -(1-\alpha)\sum_{k\neq i}^{K}\frac{t_k}{d_k}(-d_i d_k) - (1-\alpha)\frac{t_i}{d_i}d_i(1-d_i) \\
&= \left((1-\alpha)\sum_{k\neq i}^{K} t_k d_i\right) - (1-\alpha)t_i(1-d_i) \\
&= \left((1-\alpha)d_i\sum_{k=1}^{K} t_k\right) - (1-\alpha)t_i \\
&= (1-\alpha)(d_i - t_i)
\end{aligned}$$

Internal nodes with values $\bar{p}$, they contribute to J through the $[\bar{c_1}'; \bar{c_2}']$, $\bar{d}$, and through further above $\bar{q}$, as we can see in Figure 2. We calculate δ_i for all dimensions:

$$\begin{aligned}
\delta_i &= \frac{\partial J}{\partial a_i} \\
&= \sum_{z=1}^{d} \frac{\partial J}{\partial a_z}\frac{\partial a_z}{\partial a_i} + \sum_{z=1}^{2d} \frac{\partial J}{\partial e_z}\frac{\partial e_z}{\partial a_i} + \sum_{z=1}^{K} \frac{\partial J}{\partial g_z}\frac{\partial g_z}{\partial a_i} \\
&= \sum_{z=1}^{d} \delta_z \frac{\partial a_z}{\partial a_i} + \sum_{z=1}^{2d} \gamma_z \frac{\partial e_z}{\partial a_i} + \sum_{z=1}^{K} \zeta_z \frac{\partial g_z}{\partial a_i} \\
\frac{\partial a_z}{\partial a_i} &= \frac{\partial}{\partial a_i}\left[W^{1*}[..; f(\bar{a}); ..; 1]\right]_z \\
&= V_{zi} f'(a_i)
\end{aligned}$$

where V_{zi} is the part of $W^{(1)}$ that multiples $\bar{a}$.

$$\begin{aligned}
\frac{\partial e_z}{\partial a_i} &= \frac{\partial}{\partial a_i}\left[W^{2*}[f(\bar{a}); 1]\right]_z \\
&= W^{(2)}_{zi} f'(a_i) \\
\frac{\partial g_z}{\partial a_i} &= \frac{\partial}{\partial a_i}\left[W^{label*}[f(\bar{a}); 1]\right]_z \\
&= W^{(label)}_{zi} f'(a_i) \\
\delta_i &= \sum_{z=1}^{d} \delta_z V_{zi} f'(a_i) + \sum_{z=1}^{2d} \gamma_z W^{(2)}_{zi} f'(a_i) + \sum_{z=1}^{K} \zeta_z W^{(label)}_{zi} f'(a_i) \\
&= f'(a_i)\left(\sum_{z=1}^{d} \delta_z V_{zi} + \sum_{z=1}^{2d} \gamma_z W^{(2)}_{zi} + \sum_{z=1}^{K} \zeta_z W^{(label)}_{zi}\right)
\end{aligned}$$

Updating the embedding matrix L is performed as well through the following partial derivative for each word $\bar{x_i}$:

$$\frac{\partial J}{\partial \bar{x_i}} = \delta_{x_i}$$

where δ_{x_i} is the partial derivative vector at the node $\bar{x_i}$.

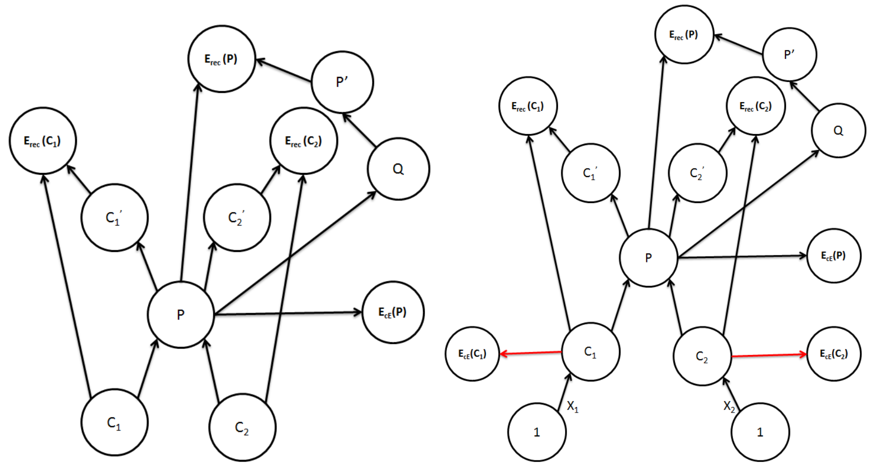
3.7 Summary of partial derivatives

We summarize the partial derivatives we derived in vector notation. We begin by noting our vector derivative for the function $f(\cdot)$:

$$\begin{aligned}
f(\bar{v}) &= \frac{tanh(\bar{v})}{\|tanh(\bar{v})\|} \\
f'(\bar{v}) &= \frac{\partial f(\bar{v})}{\bar{v}} \\
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \\
&\quad \left[\frac{1}{\|tanh(\bar{v})\|} I - \frac{1}{\|tanh(\bar{v})\|^3} tanh(\bar{v}) tanh(\bar{v})^T\right]
\end{aligned}$$

Proof: Let $g(\cdot)$ be a pointwise $tanh$ function such that:

$$g(\bar{v}) = tanh(\bar{v})$$



(a) C_1 and C_2 are internal nodes (b) C_1 and C_2 are leaf nodes

Figure 2: The structure of the nodes in the recursive autoencoder network. Q is another node up in the tree, if it exists. Nodes with (1) are artificial nodes added with edges weighted with the bias vector and words. In (b), if the label prediction error for the leaf node is considered, there will be two additional red edges from leaf nodes to label node.

The pointwise hyperbolic tangent function is applied as:

$$g(\bar{v})_i = tanh(v_i)$$

The derivative for each point is therefore:

$$g'(\bar{v})_i = sech^2(v_i)$$

Therefore, the derivative with respect the whole vector $\bar{v}$ is a matrix:

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

The derivation of the gradient follows as (for the column vector):

$$\begin{aligned}
f(\bar{v}) &= \frac{g(\bar{v})}{|g^T g|^{1/2}} \\
\frac{\partial f(\bar{v})}{\partial \bar{v}} &= \frac{\partial g(\bar{v})}{\partial \bar{v}} \times \frac{f(\bar{v})}{\partial (g(\bar{v}))} \\
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \frac{\partial}{\partial g(\bar{v})} \left(\frac{g(\bar{v})}{|g^T g|^{1/2}} \right) \\
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \left(\frac{1}{|g^T g|^{1/2}} \frac{\partial g(\bar{v})}{\partial g(\bar{v})} - \frac{1}{2} \frac{g(\bar{v})}{|g^T g|^{3/2}} \frac{\partial g^T g}{\partial g} \right) \\
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \left(\frac{1}{|g^T g|^{1/2}} I - \frac{1}{2} \frac{g(\bar{v})}{|g^T g|^{3/2}} \times 2g^T \right) \\
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \left(\frac{1}{|g^T g|^{1/2}} I - \frac{g(\bar{v})}{|g^T g|^{3/2}} \times g^T \right)
\end{aligned}$$

$$
\begin{aligned}
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \left(\frac{1}{||g(\bar{v})||} I - \frac{g(\bar{v})}{||g(\bar{v})||^3} g \times g^T \right) \\
&= \begin{bmatrix} sech^2(v_1) & & 0 \\ & \ddots & \\ 0 & & sech^2(v_d) \end{bmatrix} \times \left[\frac{1}{||tanh(\bar{v})||} I - \frac{1}{||tanh(\bar{v})||^3} tanh(\bar{v}) \times tanh(\bar{v})^T \right)
\end{aligned}
$$

End Proof

Output nodes, $\bar{c_1}', \bar{c_1}', \bar{d}$:

$$
\begin{aligned}
\gamma &= \frac{\partial J}{\partial \bar{e}} = \left[-2\alpha \frac{n_1}{n_1 + n_2} f'(\bar{e}) \times (\bar{c_1} - \bar{c_1}'); -2\alpha \frac{n_2}{n_1 + n_2} f'(\bar{e}) \times (\bar{c_2} - \bar{c_2}') \right] \\
\zeta &= \frac{\partial J}{\partial \bar{g}} = (1 - \alpha)(\bar{d} - \bar{t})
\end{aligned}
$$

Internal nodes, $\bar{p}$:

$$
\delta = f'(\bar{a}) \cdot \left[{W^{(1)}}^T \delta_{\bar{q}} + {W^{(2)}}^T \gamma + {W^{(label)}}^T \zeta + \frac{\partial E_{rec}(\bar{p}, \bar{p}')}{\partial \bar{p}} \right]
$$

where the last term in the bracket is the deviation of $\bar{p}$ effect on $E_{rec}(\bar{p}, \bar{p}')$. Since the representation of node $\bar{p}$ is not fixed, this last term has to be taken into consideration during back propagation. Note that it is the derivative of the reconstruction error $E_{rec}(\bar{p}, \bar{p}')$ respect to the node $\bar{p}$ rather then the node $\bar{p}'$.

Leaf nodes, $\bar{x}$:

$$
\delta_{\bar{x}} = \left[{W^{(1)}}^T \delta_{\bar{q}} + {W^{(label)}}^T \zeta + \frac{\partial E_{rec}(\bar{x}, \bar{x}')}{\partial \bar{x}} \right]
$$

Therefore, we have the following partial derivatives for all weight matrices:

$$
\begin{aligned}
\frac{\partial J}{\partial W^{1*}} &= \delta [\bar{c_1}; \bar{c_2}; 1]^T \\
\frac{\partial J}{\partial W^{2*}} &= \delta [\bar{p}; 1]^T \\
\frac{\partial J}{\partial W^{label*}} &= \delta [\bar{p}; 1]^T
\end{aligned}
$$

We then decompose each W^* matrix to its original components:

$$
\begin{aligned}
\frac{\partial J}{\partial W^{1*}} &= \left[\frac{\partial J}{\partial W^{(1)}} \frac{\partial J}{\partial \bar{b}^{(1)}} \right] \\
\frac{\partial J}{\partial W^{2*}} &= \left[\frac{\partial J}{\partial W^{(2)}} \frac{\partial J}{\partial \bar{b}^{(2)}} \right] \\
\frac{\partial J}{\partial W^{label*}} &= \left[\frac{\partial J}{\partial W^{(label)}} \frac{\partial J}{\partial \bar{b}^{(label)}} \right] \\
\frac{\partial J}{\partial \bar{x}} &= \delta_{\bar{x}}
\end{aligned}
$$

We now give the complete algorithm to train our model in Algorithm 2 below:

Algorithm 2 Training the Recursive Autoencoder

```
Input: Training Examples $\langle x_1, x_2, ..., x_n \rangle$
Output: Trained Model $\theta = \langle W^{(1)}, \bar{b}^{(1)}, W^{(2)}, \bar{b}^{(2)}, W^{(label)}, \bar{b}^{(label)}, L \rangle$

Get Vocabulary $V$ from all examples
Initialize word vectors $w_i \sim \mathcal{N}(0, \sigma^2) \quad \forall 1 \le i \le |V|$
Initialize matrices $W^{(1)}, W^{(2)}, W^{(label)}$ randomly.
Initialize bias vectors $\bar{b}^{(1)}, \bar{b}^{(2)}, \bar{b}^{(label)} \leftarrow 0$
while Not Converged do
  Initialize Gradient $\nabla J \leftarrow 0$
  for all $x_i \quad \forall i = 1, ..., n$ do
    Compute $Tree = RAE_\theta(x_i)$ according to equation (6)
    Compute Gradient $\nabla J_i \leftarrow \frac{\partial J(x_i)}{\partial \theta}$
    Update Total Gradient $\nabla J \leftarrow \nabla J + \nabla J_i$
  end for
  Update $\theta \leftarrow \frac{1}{N} \nabla J + \lambda \theta$
end while
```

3.8 Verifying the Gradient

To verify the correct computation of our gradient in Equation (8) we follow the recommendation in [3] and evaluate the following equation:

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

We compare the result of this computation with the gradient computed previously. Although loss functions are defined locally to each node, the tree structure make each internal node's delta value propagate to its parent nodes. Therefore full backpropagation is necessary to compute the correct gradient respect to loss functions J and parameters θ.

4 Experiments

4.1 Design

The experiments are designed to closely follow Socher's to reproduce similar results. As so, we have used the the same regularization strengths and optimization algorithm (i.e. limited-memory BFGS). The regularizations are applied to $W^{(1)}$, $W^{(2)}$, $W^{(label)}$ and L with values 10^{-5}, 10^{-4}, 10^{7} and 10^{-2} respectively. We used Mark Schmidt's `minFunc` function [4] for the L-BFGS optimization.

The accuracy is measured using a modification of Socher's `classifyWithRAE.m` so we can have a meaningful comparison between our implementation and his results. His code is modified to accommodate differences in the way we implement the learning algorithm. The same logistic regression model is used for learning the classifier. The feature of logistic regression model for each tree is the representation of the root node and the average representation of the rest of the nodes. Thus, the dimensionality of each feature is $2d$. The final accuracy is reported as the average of the ten-fold cross-validation.

We compare the method described in [1] and its modification. The modifications are 1) including leaf nodes for the loss and gradient calculations, and 2) applying normalization on the reconstruction output nodes c_1' and c_2'. The combination of the two modifications are what the author seems to implement in his code despite the fact that it is not mentioned in the paper. The error for the leaf node is calculated using the cross-entropy defined in (5). We refer to the author's proposed method in the paper [1] as Method A, the first modification as Method B, the second modification as Method C, and the combinations of the modifications as Method D. In addition, we also compare these methods with a pure supervised learning method (Method E) which only computes the supervised cross-entropy error for each leaf node without a tree structure. The methods are summarized in Table 1. Additionally, the gradient of each method is verified using Jason Rennie's `checkgrad2` [5].

Table 1: Different combination of Methods

Features	RAE tree	Normalization	Leaf node Label
Method A	v		
Method B	v		v
Method C	v	v	
Method D	v	v	v
Method E			v

We perform three experiments. First, we compare Method A through D for different error weights, α, in equation (7) ranging from 0 to 1 with 0.1 increment using word vector size of 30-dimensional space. The experiment is done on a single fold. Second, using the best performing error weight value for each method, we compare the average accuracy and standard error from the ten-fold cross-validation of all methods. The average accuracy μ is calculated as

$$\mu = \frac{1}{10} \sum_{n=1}^{10} a_n$$

where a_n is the accuracy for the nth fold, and the standard error SE is calculated as

$$SE = \sqrt{\frac{\sigma^2}{10}}$$

where σ^2 is the sample variance. This experiment also uses 30-dimensional word vector size. Last, we run Method D for various word vector size to compare accuracy and computation time.

The first two experiments uses 30-dimensional word vector size since they were conducted on multiple computers and not all computers were able to run in higher dimensions. The last experiment was done on a machine with Intel Core i7 2.8 GHz processor with 4 core and 6 GB of memory.

4.2 Results

Figure 3 shows the result of the first experiment where we vary the weights of reconstruction and supervised cross-entropy error on a single fold. We see that Method A and C have the highest accuracy when $\alpha = 0$. This means that the highest accuracy is achieved when no reconstruction error is used for the error and gradient calculations, and the training is fully supervised. On the other hand, Method B and D, which include leaf nodes for the label prediction error calculation, have overall higher accuracies than Method A and C for most weights. The highest accuracy for Method B is $\alpha = 0.1$ and for Method D is $\alpha = 0.3$. When $\alpha = 1$, the training is fully unsupervised because the weight for the label prediction error is zero. We see that Method A and B and Method C and D have the same accuracies in this case. The performance is also worse, especially for Method D, since semantically similar words cannot be updated to have similar vector representation.

When the error weight is zero, the tree structure of the sentence is irrelevant to minimizing the objective function (7) and learning is done based on only the label prediction error. Therefore, Method A and C and Method B and D have the same accuracy since the methods only differ in reconstruction error calculation. For Method A and C with error weights higher than zero, the accuracies decrease significantly compare to Method B and D. This implies that the meaning of the sentence is greatly affected by the representation of the leaf node, and that if each word does not have a meaning representation, i.e. the object function does not include t he label prediction error, then RAE can not guarantee that two words with a same meaning will have similar representations. This also explains why Method D has a sudden performance drop at fully unsupervised learning case ($\alpha = 1$).

Table 2 shows the average accuracy and standard error for all methods, using the optimal weight values found in Figure 3 for Method A-D. The result shows that, under best alpha setting, including leaf node label prediction significantly improves the accuracy compare to Method A, and Method E achieves the highest accuracy without a RAE tree structure. Adding normalization to the reconstructed nodes seems to have a negligible effect to the error weight. However, the results of Method

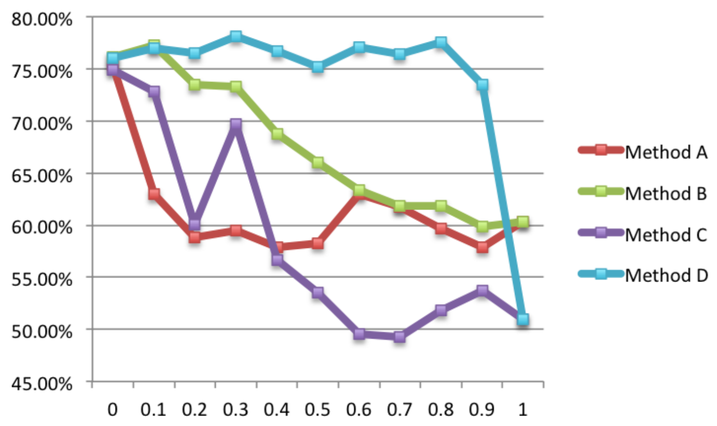


Figure 3: Accuracy of the four methods for different weightings of reconstruction and supervised cross-entropy error shown in equation (7). The highest accuracy weights for each method are 0, 0.1, 0 and 0.3 respectively.

B and Method D in Figure 3 show that this normalization makes the accuracy less sensitive to the choice of alpha. Since this normalization constrains the reconstruction error for each output node to be at most 2, it has an effect to put more weight on the label prediction error of the objective function. Thus, this explains why Method D is less sensitive to the choice of the error weight, and why Method B has the best performance at $\alpha = 0.1$ rather than $\alpha = 0.3$.

Table 2: Average accuracy and standard error for each method using 30-dimensional word vector size and optimal error weights found in Figure 3.

	Method A	Method B	Method C	Method D	Method E
Avg. Accuracy (%)	73.8	75.7	73.8	76.4	76.4
Standard Error (%)	0.43	0.60	0.44	0.51	0.52

While the first two results are done using 30-dimensional word vector size, Table 3 shows the average accuracy and computation time for Method D using higher dimension word vectors. $\alpha = 0.2$ is chosen to compare with original accuracy of 76.8% reported in [1]. The average computation time includes both training and validation. The table shows that the accuracy improves for higher word vector size and computation time increases linearly to the vector size. However, while the computation time is linear to the word vector size, the accuracy only improves 0.4% between 50 and 100 dimensions. For future experiments, it would be interesting to see if even higher word vector size may produce better result. Additionally, running Socher's code on the same computer takes over 50 minutes for word vector size of 30.

Table 3: Average accuracy and computation time for Method D using word vector size of 30, 50 and 100 and $\alpha = 0.2$.

Word vector size	30	50	100
Avg. Accuracy (%)	76.2	76.5	76.9
Avg. Computation time (minutes)	39	51	104

5 Conclusion

In this project, we implement a semi-spupervised RAE with random word initialization using the Movie Reviews dataset. Compare to the original accuracy reported in [1], our result only differed 0.1% and achieved 76.9% accuracy using 10-fold cross-validation. We found that it is beneficial to include label prediction for each leaf node and update its word representation with its corresponding gradient. Using label information for each leaf node (word) enhances the trend for words with similar meaning to have similar representations. On the other hand, adding normalization to the output nodes essentially constrains the effect of reconstruction error during optimization, thus making the algorithm less sensitive to the choice of weight between the label prediction error and the reconstruction error. Finally, the results show that the RAE tree structure is not necessary for this MR dataset to achieve the highest accuracy.

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] B. Pang and L. Lee, “Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales,” in Proceedings of the ACL, 2005, pp. 115–124.

[3] C. Elkan, “Learning the meanings of sentences,” 2012. [Online]. Available: http://cseweb.ucsd.edu/ elkan/250B/learningmeaning.pdf

[4] M. Schmidt, “minfunc,” http://www.di.ens.fr/ mschmidt/Software/minFunc.html, 2005.

[5] J. Rennie, “checkgrad2,” http://people.csail.mit.edu/jrennie/matlab/checkgrad2.m.