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

Mohsen Malmir, Erfan Sayyari

March 17, 2014

1 Abstract

2 Introduction

3 Design and Analysis of Algorithm

3.1 Recursive definition of meaning

In the context of deep learning to represent the semantic content of sentences, we need to represent each word by a vector, and then represent their relative relation by a tree. We use binary tree representation and each word is a leaf node of the tree, and there are $n-1$ internal nodes, where n is the length of the sentence. Each of the internal nodes shows the phrase of two or more consecutive words. To represent the meaning of each node we use a vector of $\mathbb{R}^d$.

We use a random initialization to represent the meaning of each node in $\mathbb{R}^d$. Each random vector is produced from a Gaussian of dimension d with zero mean and diagonal covariance matrix $\sigma^2 I$ independent from each other.

To obtain the meaning of each internal node (say node k), rather than leaf nodes, we use the following equation:

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

where i and j are children of node k, and W is a matrix in $\mathbb{R}^{d\times 2d}$ and b is a vector in $mathbbR^d$. The function h is a pointwise sigmoid-shaped function from $\mathbb{R}^d$ to the interval $[-1,+1]^d$.

In the training stage, we need to learn parameters W and b. Since our problem is a supervised method, we need a target value for the meaning of the sentences. We use the notation x_r to represent the predicted meaning of the sentence (of the root node r). If the true target value of the sentence is t, the loss function could be for example $E = (t - x_r)^2$, and for training we need to minimize this error. To do minimization we need to compute $\frac{\partial E}{\partial W}$ and $\frac{\partial E}{\partial b}$ and use a stochastic gradient descent (SGD) algorithm to minimize E for the all of sentences, or we can use LBFGS quasi-Newton method.

3.2 Autoencoders

In the autoencoder approach, we want to change the goal of supervised learning to find the parameters W, b automatically without training labels provided from the outside. We could consider the meaning of the node x_k is equal to:

$$x_k = h(W[x_i; x_j] + b). \tag{2}$$

We represent two other variables, z_i and z_j such that:

$$[z_i; z_j] = Ux_k + c \tag{3}$$

where U is a matrix in $\mathbb{R}^{2d\times d}$ and c is a vector in $\mathbb{R}^{2d}$, where z_i and z_j are approximation reconstructions of the inputs x_i and x_j, and U and c are additional parameters. So we can define a new loss as:

$$E = ||x_i - z_i||^2 + ||x_j - z_j||^2 = ||[x_i; x_j] - Uh(W[x_i; x_j] + b) - c||^2. \tag{4}$$

For the whole sentence the error is the sum of error of reconstruction of all non-leaf nodes. So we can use this reconstruction error to learn W, b, U, and c with no training labels provided from the outside.

This approach has two other refinements. In order not to train the system to the trivial solution which leads to zero error we need to justify the equations. The trivial solution is all the parameters equal to zero. So the error will be zero. In order to solve it we can normalize the equation for the meaning of k^th node such that:

$$x_k = \frac{h(W[x_i; x_j] + b}{||h(W[x_i; x_j] + b||}. \tag{5}$$

This leads the h function not to be pointwise anymore, and derivatives are much harder in this case.

The other refinement is based on the intuition that the error of reconstruction of the longer phrases are much more important. Therefore, the definition of the loss for node k is equal to:

$$E_1(k) = \frac{n_i}{n_i + n_j}||x_i - z_i||^2 + \frac{n_j}{n_i + n_j}||x_j - z_j||^2 \tag{6}$$

where z_i and z_j are as before, and n_i and n_j are how many words are covered by nodes i and j.

3.3 Selecting a tree structure

After training the parameters automatically, we will generate a method to reconstruct the tree automatically as well. For a given sentence, if T be the set of non-leaf nodes of its binary tree, the error of the whole tree is equal to:

$$\sum_{k \in T} E_1(k). \tag{7}$$

For a sentence of length n, there is a finite number of possible trees calculated from the Catalan number of C_{n-1}, where C_n is:

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}. \tag{8}$$

We can define an optimal tree to be the one that minimizes the total error. This optimal tree can be found by the exhaustive search or we can find it by a greedy algorithm approximately.

The greedy algorithm is as follows: first, we consider all $n-1$ pairs of neighboring words. Then we find the best pair based on reconstruction error with smallest error. Now omit their combination, and find the best pair based on remaining pairs.

A small change in the parameter values either causes no change in the optimal tree obtained by greedy algorithm or a jump to a different tree. Generally, gradient descent could cause cycling between two or more different trees, without convergence, while LBFGS would converge smoothly.

3.4 Using meaning to predict labels

We can have a target value rather than target meaning for a sentence to be predicted. For example, positivity or negativity of the sentence, which is binary.

Each node of the tree (say node k) has a meaning vector x_k, and we can add a linear model to each node to predict target value of each node. If the values of the predicted labels are binary, then linear model is a standard logistic regression classifier, and if there are more than two classes, the model is multinomial or multiclass logistic regression.

Suppose that we have r different discrete labels and x_k be the meaning of node k. Then the vector of predicted probabilities of the label values is equal to:

$$\bar{p} = softmax(V x_k) \tag{9}$$

where V is a matrix in $\mathbb{R}^{r \times d}$. If we have $\bar{t}$ to be the true binary vector of length r of node k, then we can define squared error of the prediction as $||\bar{t} - \bar{p}||^2$, or we can define log loss as:

$$E_2(k) = -\sum_{i=1}^{r} t_i \log p_i. \tag{10}$$

Algorithm 1: Constructing the Tree using Greedy Algorithm

```
# Input: Training Examples $< x_1, x_2, \ldots, x_n >$
# Output: Optimal Tree T
Nodes=$\{x_1, x_2, \ldots, x_n\}$
while $Nodes.remaining > 1$ do
    $MinErrorFound = \inf$
    $j = -1$
    ConcatenatingNode = $null$
    for from $i = 1$ to $i = Nodes.remaining - 1$ do
        Compute $p \leftarrow h(W^{(1)}[c_1; c_2] + b^{(1)})$
        Compute $[\hat{c}_1; \hat{c}_2] \leftarrow W^{(2)} p + b^2$
        $Error \leftarrow E_1([c_1; c_2], [\hat{c}_1; \hat{c}_2])$
        if $Error < MinErrorFound$ then
            MinErrorFound $\leftarrow Error$
            $j \leftarrow i$ ConcatenatingNode $\leftarrow p$
        end
    end
    ConcatenatingNode.leftChild $\leftarrow Nodes(j)$
    ConcatenatingNode.rightChild $\leftarrow Nodes(j+1)$
    Nodes $\leftarrow Nodes - \{Nodes(j), Nodes(j+1)\} + \{ConcatenatingNode\}$
end
$T.root \leftarrow$ ConcatenatingNode
```

We predict the target values for all the internal nodes except leaf nodes.

If we consider a collection S of m labeled training sentences, then we could write the loss function as:

$$J = \frac{1}{m} \sum_{<s,t> \in S} E(s, t, \theta) + \frac{\lambda}{2} ||\theta||^2 \tag{11}$$

where $\theta =< W, b, U, c, V >$ is all the parameters of the model, and λ is a regularization factor, and $E(s, t, \theta)$ is the total error for one sentence s with label t which is equal to:

$$E(s, t, \theta) = \sum_{k \in T(s)} \alpha E_1(k) + (1 - \alpha) E_2(k) \tag{12}$$

where $T(s)$ is the set of non-leaf nodes of the tree is constructed by the greedy algorithm for the sentence s.

In order to update the meaning of a node x_n, we can compute the derivative of the loss with respect to x_n and update it. More particularly, to do this task we use the error of label targets E_2. We can write the general equation as:

$$\frac{\partial}{\partial x_n} \sum_{<s,t>\in S} \sum_{k\in T(s)} E_2(k). \tag{13}$$

3.5 Back-propagation for vector-valued nodes

The complete loss function J is consisted of error of reconstruction E_{rec} or E_1 and error of prediction E_2:

$$J = \frac{1}{N}\sum_{s,t} E(s,t;\theta) + \frac{\lambda}{2}||\theta||^2 \tag{14}$$

$$= \sum_{k\in T(s)} \alpha E_1(k) + (1-\alpha)E_2(k) + \frac{\lambda}{2}||\theta||^2 \tag{15}$$

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

$$\frac{\partial J}{\partial \theta} = \frac{1}{N}\sum_{s,t} \frac{\partial E(s,t;\theta)}{\partial \theta} + \lambda\theta \tag{16}$$

We have different activation and values in the network:

$$p = h(a) = h([W^{(1)}b^{(1)}][c_1; c_2; 1]) \tag{17}$$

$$e = [\hat{c}_1; \hat{c}_2] = [W^{(2)}b^{(2)}][p; 1] \tag{18}$$

$$r = softmax(a^{output}) = softmax([W^{label}b^{label}][p_r; 1]) \tag{19}$$

where r is a vector in $\mathbb{R}^q$ (the number of classes is q), and r is the predicted probabilities for the classes. Here p_r is the meaning vector for the output node, and W^{label} is a matrix in $\mathbb{R}^{q\times d}$.

We can concatenate $W^{(1)}$ with $b^{(1)}$ and we can concatenate $W^{(2)}$ with $b^{(2)}$:

$$W^{(1*)} = [W^{(1)}b^{(1)}] \tag{20}$$

$$W^{(2*)} = [W^{(2)} b^{(2)}] \tag{21}$$

Softmax is not a point-wise function, and r_j is equal to:

$$r_j = f_j((W^{label} p_r)_1, (W^{label} p_r)_2, \ldots, (W^{label} p_r)_q) = \frac{e^{(W^{label} p_r)_j}}{\sum_{i=1}^{q} e^{(W^{label} p_r)_j}} \tag{22}$$

$$r = \begin{pmatrix} f_1(\sum_{j=1}^{d} W_{1j}^{label} p_{r_j}, \sum_{j=1}^{d} W_{2j}^{label} p_{r_j}, \ldots, \sum_{j=1}^{d} W_{qj}^{label} p_{r_j}) \\ f_2(\sum_{j=1}^{d} W_{1j}^{label} p_{r_j}, \sum_{j=1}^{d} W_{2j}^{label} p_{r_j}, \ldots, \sum_{j=1}^{d} W_{qj}^{label} p_{r_j}) \\ \vdots \\ f_q(\sum_{j=1}^{d} W_{1j}^{label} p_{r_j}, \sum_{j=1}^{d} W_{2j}^{label} p_{r_j}, \ldots, \sum_{j=1}^{d} W_{qj}^{label} p_{r_j}) \end{pmatrix} \tag{23}$$

So we could consider $a^{output} = W^{label} p_r$ as, which is a vector in $\mathbb{R}^q$ and rewrite the r as:

$$r = \begin{pmatrix} f_1(a_1^{output}, a_2^{output}, \ldots, a_q^{output}) \\ f_2(a_1^{output}, a_2^{output}, \ldots, a_q^{output}) \\ \vdots \\ f_q(a_1^{output}, a_2^{output}, \ldots, a_q^{output}) \end{pmatrix} \tag{24}$$

Now, we want $\frac{\partial J}{\partial W_{i,j}^{(1)}}$, which is not explicitly related to loss function J. We could write:

$$p_r = h(W^{(1)}[c_1; c_2] + b^{(1)}]) \tag{25}$$

where h is a pointwise highly nonlinear function. Now we could have:

We need the partial derivatives of the loss function with respect to parameters:

$$\frac{\partial J}{\partial W_{ij}^{(1*)}} = \frac{\partial J}{\partial a_i} \frac{\partial a_i}{\partial W_{ij}^{(1*)}} = \delta_i \frac{\partial a_i}{\partial W_{ij}^{(1*)}} \tag{26}$$

$$\frac{\partial a_i}{\partial W_i^{(1*)}} = [c_1; c_2; 1]^T \tag{27}$$

$$\frac{\partial J}{\partial W_{ij}^{(2*)}} = \frac{\partial J}{\partial e_i} \frac{\partial e_i}{\partial W_{ij}^{(2*)}} = \gamma_i \frac{\partial e_i}{\partial W_{ij}^{(2*)}} \tag{28}$$

$$\frac{\partial e_i}{\partial W_i^{(2*)}} = [x_p; 1]^T \tag{29}$$

$$\frac{\partial J}{\partial W_{ij}^{label}} = \frac{\partial J}{\partial a_i^{output}} \frac{\partial a_i^{output}}{\partial W_{ij}^{label}} = \zeta_i \frac{\partial a_i^{output}}{\partial W_{ij}^{label}} \tag{30}$$

$$\frac{\partial a_i^{output}}{\partial W^{label}} = [x_r; 1]^T \tag{31}$$

We want to calculate γ_i which just related to error of reconstruction, E_1:

$$\gamma_i = \frac{\partial J}{\partial e_i} = \alpha \frac{n_1}{n_1 + n_2} 2(c_1 - \hat{c}_1)^T (-\frac{\partial \hat{c}_1}{\partial e_i}) + \alpha \frac{n_2}{n_1 + n_2} 2(c_2 - \hat{c}_2)^T (-\frac{\partial \hat{c}_2}{\partial e_i}) \tag{32}$$

$$\frac{\partial [\hat{c}_1; \hat{c}_2]}{\partial e_i} = 1 \tag{33}$$

$$\gamma_i = -\alpha [\frac{n_1}{n_1 + n_2} 2(c_1 - \hat{c}_1); \frac{n_2}{n_1 + n_2} 2(c_2 - \hat{c}_2)] \tag{34}$$

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

$$\frac{\partial J}{\partial a_i^{(r)}} = \sum_{k=1}^{q} \frac{\partial J}{\partial r_k} \frac{\partial r_k}{\partial a_i^{(r)}} \tag{35}$$

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

$$\frac{\partial r_k}{\partial a_i^{(r)}} = \frac{\partial f_k}{\partial a_i^{(r)}} = \sum_{t=1}^{q} \frac{\partial f_k}{\partial a_t^{output}} \frac{\partial a_t^{output}}{\partial a_i^{(r)}} \tag{36}$$

The output node with values r, the contribution to J is with the error of classification E_2.

$$\zeta_i = \frac{\partial J}{\partial a_i^{output}} = (1 - \alpha) \frac{\partial}{\partial a_i^{output}} \sum_{l=1} q t_l \log r_l \tag{37}$$

If $l = k$:

$$\frac{\partial f_i}{\partial a_i^{output}} = r_i (1 - r_i) \tag{38}$$

and if $l \neq i$

$$\frac{\partial f_i}{\partial a_l^{output}} = -r_i r_l \tag{39}$$

So we could calculate ζ_i as:

$$\zeta_i = -(1-\alpha)\sum_{l,l\neq i}^{q} \frac{t_i}{r_i}(-r_l r_i) - (1-\alpha)\frac{r_i}{1-r_i} = (1-\alpha)(r_i - t_i) \tag{40}$$

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

$$\delta_i = \frac{\partial J}{\partial a_i} = \sum_{k=1}^{d} \frac{\partial J}{\partial a_k}\frac{\partial a_k}{\partial a_i} + \sum_{k=1}^{2d} \frac{\partial J}{\partial e_k}\frac{\partial e_k}{\partial a_i} + \sum_{k=1}^{q} \frac{\partial J}{\partial a_k^{output}}\frac{\partial a_k^{output}}{\partial a_i} \tag{41}$$

$$= \sum_{k=1}^{d} \delta_k \frac{\partial a_k}{\partial a_i} + \sum_{k=1}^{2d} \gamma_k \frac{\partial e_k}{\partial a_i} + \sum_{k=1}^{q} \zeta_k \frac{\partial a_k^{output}}{\partial a_i} \tag{42}$$

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

$$\frac{\partial a_k}{\partial a_i} = \frac{\partial}{\partial a_i}[W^{(1*)}[\ldots; h(a); \ldots; 1]]_k = V_{ki} h'(a_i) \tag{43}$$

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

$$\frac{\partial e_k}{\partial a_i} = \frac{\partial}{\partial a_i}[W^{(2*)}[h(a); 1]]_k = W_{ki}^{(2*)} h'(a) \tag{44}$$

$$\frac{\partial a_k^{output}}{\partial a_i} = \frac{\partial}{\partial a_i}[W^{label}[h(a); 1]]_k = W_{ki}^{label} h'(a) \tag{45}$$

$$\delta_i = \sum_{k=1}^{d} \delta_k V_{ki} h'(a_i) + \sum_{k=1}^{2d} \gamma_k W_{ki}^{(2*)} h'(a_i) + \sum_{k=1}^{q} \zeta_k W_{ki}^{label} h'(a_i) \tag{46}$$

$$= h'(a_i)(\sum_{k=1}^{d} \delta_k V_{ki} + \sum_{k=1}^{2d} \gamma_k W_{ki}^{(2*)} + \sum_{k=1}^{q} \zeta_k W_{ki}^{label}) \tag{47}$$

For the leaves we could write:

$$\frac{\partial J}{\partial x_i} = \delta_{x_i} \tag{48}$$

3.6 Calculating derivatives for normalized vectors

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

$$h(a) = \frac{\tanh(a)}{||\tanh(a)||} \tag{49}$$

$$h'(a) = \begin{bmatrix} sech^2(a_1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & sech^2(a_d) \end{bmatrix} \times \left[\frac{1}{||tanh(a)||} I - \frac{1}{||tanh(a)||^3} tanh(a)tanh(a)^T \right] \tag{50}$$

3.7 Summarizing Derivatives

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

$$\gamma = \frac{\partial J}{\partial e} = \left[-2\alpha \frac{n_1}{n_1+n_2}(c_1 - \hat{c}_1); -2\alpha \frac{n_2}{n_1+n_2}(c_2 - \hat{c}_2) \right] \tag{51}$$

$$\zeta = \frac{\partial J}{\partial a^{output}} = (1 - \alpha)(r - t) \tag{52}$$

For the internal nodes,p we have:

$$\delta = h'(a). \left[W^{(1*)^T} \delta_q + W^{(2*)^T} \gamma + W^{(label)^T} \zeta + \frac{\partial E_r ec(p, \hat{p})}{\partial p} \right] \tag{53}$$

For the leaf nodes, x we have:

$$\delta_x = \left[W^{(1*)^T} \delta_q + W^{(label)^T} \zeta + \frac{\partial E_r ec(x, \hat{x})}{\partial x} \right] \tag{54}$$

So the derivatives for the parameters are equal to:

$$\frac{\partial J}{\partial W^{(1*)}} = \delta[c_1; c_2; 1]^T \tag{55}$$

$$\frac{\partial J}{\partial W^{(2*)}} = \delta[p; 1]^T \tag{56}$$

$$\frac{\partial J}{\partial W^{(label)}} = \delta[p; 1]^T \tag{57}$$

The algorithm for training the model is given in algorithm 2.

Algorithm 2: Training the Recursive Autoencoder

```
# Input: Training Examples $< x_1, x_2, \ldots, x_n >$
# Output: Training Model
$\theta =< W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, W^{label}, Leaves >$
#Initialization Initialize word meaning vectors $x_i \sim \mathcal{N}(0, \sigma^2)$
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 $\bigtriangledown J \leftarrow 0$ for all $x_i$ do
        Compute $Tree = RAE_\theta(x_i)$
        Compute Gradient $\bigtriangledown J_i \leftarrow \frac{\partial J(x_i)}{\partial \theta}$
        Update Total Gradient $\bigtriangledown J \leftarrow \bigtriangledown J + \bigtriangledown J_i$
    end
    Update $\theta \leftarrow \frac{1}{N} \bigtriangledown J + \lambda\theta$
end
```

4 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,

$$\frac{\partial J}{\partial \theta} = \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} + O(\epsilon^2). \tag{58}$$

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 with tanh classification function. For each experiment, we show the norm of gradient 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 two different positive and negative classes. And finally, we show the most positive and negative words learned for the version where the leaves are updated.

5 Results

5.1 Gradient Check

Figure 1 shows the absolute value of the difference between numerical and gradients we derived in this paper. This figure shows the absolute difference value as a function of ϵ for softmax and tanh network. It is obvious that the derivatives are correct in comparison to the numerical derivatives.

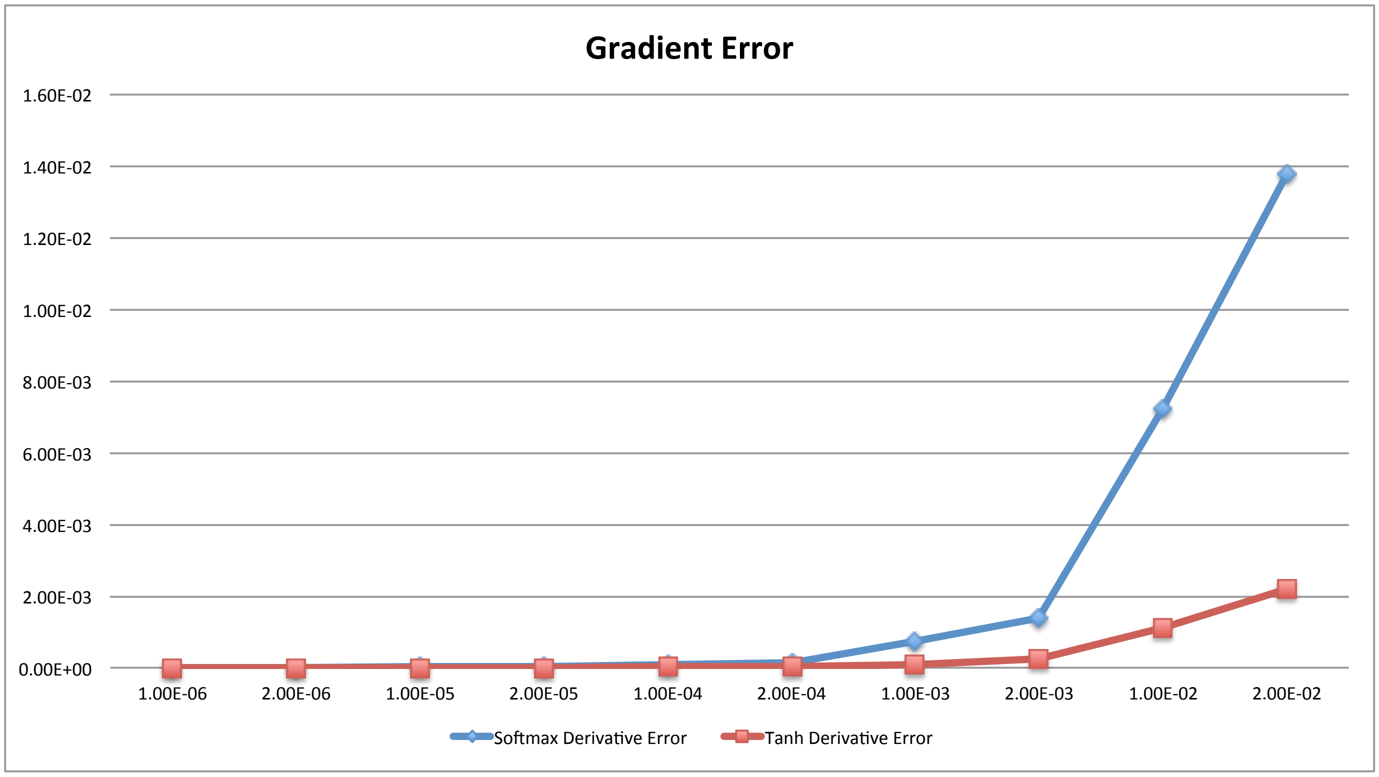

Figure 1: Absolute difference between the numerical gradient and derivatives we derived in this paper.

5.2 Convergence of Model Training

Figure 2 shows the convergence of the algorithm for different configurations of the model.

5.3 Recongition Performance on Test Set

The results of recognition are shown in table 1

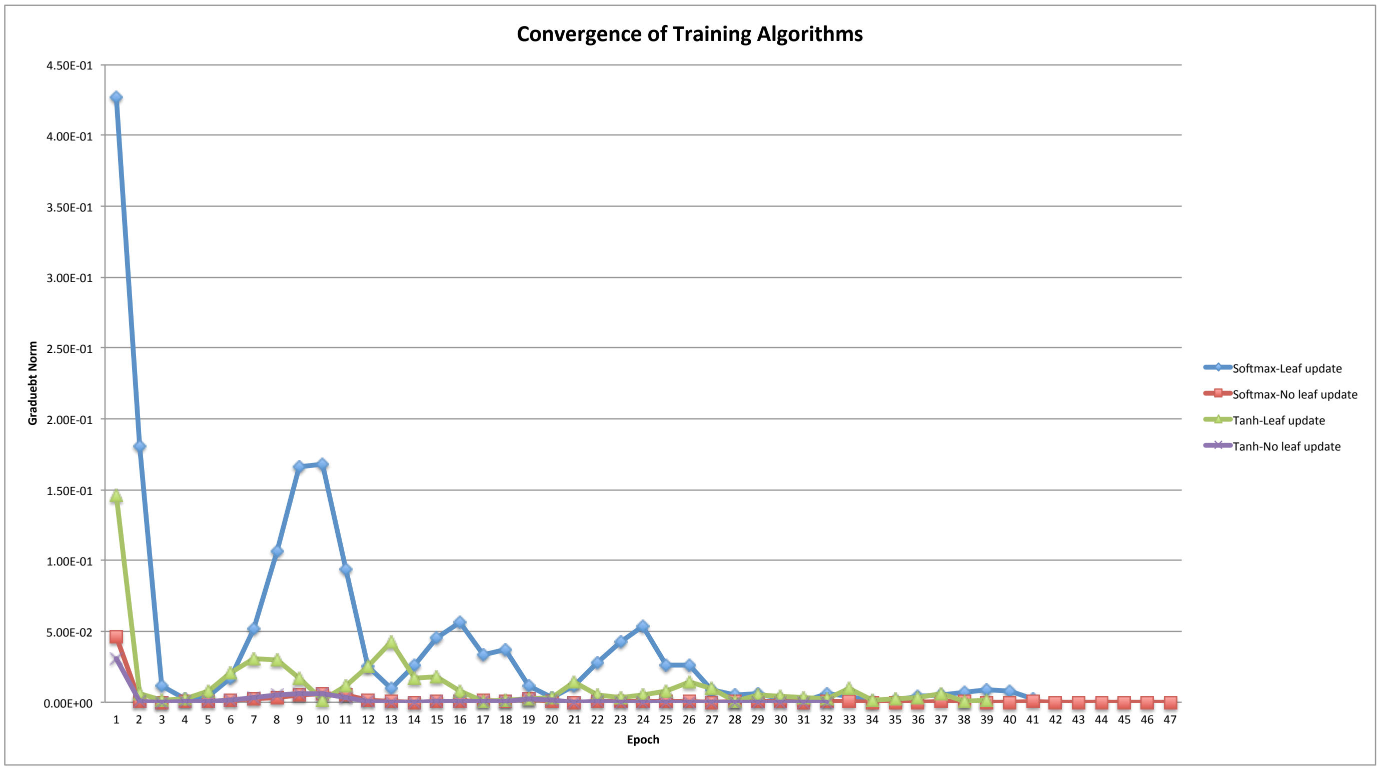

Figure 2: Convergence of different models based on the number of epochs.

Model	**Test set recognition rate**
softmax- Leaf update	0.743
softmax- no leaf update	0.514
Tanh- leaf update	0.690
Tanh- no leaf update	0.501

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

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 3 topics of Classic400. Topic 1 is scientific methods, topic 2 is aerospace-physics and topic 3 is medical.

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 3 topics of Classic400. Topic 1 is scientific methods, topic 2 is aerospace-physics and topic 3 is medical.

5.4 Most positive and most negative words

6 Discussion

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.