Learning Meanings for Sentences with Recursive Autoencoders

Tsung-Yi Lin and Chen-Yu Lee
Department of Electrical and Computer Engineering
University of California, San Diego
{tsl008, chl260}@ucsd.edu

March 18, 2013

Abstract

The objective of this project is to implement the recursive auto encoder (RAE) method to learn a model to predict sentiments for sentences and reproduce the result in [1]. To learn the weights for recursive functions, we implement forward and backward propagation algorithms. We validate the gradient computed from forward and backward algorithm by comparing it to the gradient computed from numerical approximation. Our implementation could achieve accuracy 48.5% with the basic RAE implementation. We show that the system achieves accuracy 76.0% by using normalized transfer function and adjustable meaning vectors.

1 Introduction

Learning meanings for sentences is an important application. We leverage recursive auto encoder (RAE) to learn a model to predict the meaning of sentence. At the same time, we learn the structure of sentences and meaning vector of words with semi-supervised algorithm. In this project, we implement the recursively structured function to this task and reproduce the result in [1]. We use forward propagation with greedy algorithm to predict the tree structure of a sentence, and then use backward propagation to compute all needed gradients values efficiently to update weights in our recursively structured model. To validate the correctness of gradient computation, we compare our gradient values with numerical differentiation. We can compute correct gradient values with differences between numerical differentiation under 10^{-9}.

In the experiment section, we implement RAE with four different algorithms. we first implement basic RAE tree using un-normalized transfer function and fixed meaning vectors for input words. And we produce the rest of three variations by adding different combinations of these two methods to investigate the impact of each method. The experiment result suggests using normalized feature and adjusting meaning vector achieves highest accuracy for sentiment prediction. Empirically, we demonstrate the ability of RAE model to predict the sentiment of sentences and words by showing top 10 best and worst movie review sentences and words. We find RAE can predict sentiments for whole sentence but fail to predict the word-level sentiments since we do not assign sentiments for each word while training. We also show the learned meaning vectors can be used for retrieving sentence with similar meaning. We also show that our model can predict correct tree structure of a sentence.

2 Algorithm

2.1 Unsupervised Autoencoders

The goal of using autoencoders is to do unsupervised learning through reconstruction error between two children nodes and two reconstructed nodes. Let node k has two children nodes i and j, whose meaning vector is x_i and x_j in length d. We can encode the meanings of two children nodes to generate a meaning vector for node k using:

$$c_k = h(W_1[c_i; c_j] + b_1) \tag{1}$$

where $W_1 \in \mathbb{R}^{d \times 2d}$ and $b_1 \in \mathbb{R}^d$ are weights we need to learn. We use point-wise $tanh(x)$ function for h to map each element from $\mathbb{R}$ to $[-1, 1]$. Now we can try to reconstruct the meanings of children node i and j from c_k by using:

$$[c'_i; c'_j] = W_2 \cdot c_k + b_2 \tag{2}$$

we can then use square loss to compute reconstruction error E_1 by:

$$E_1 = \frac{n_i}{n_i + n_j} \left\|c_i - c'_i\right\|^2 + \frac{n_j}{n_i + n_j} \left\|c_j - c'_j\right\| \tag{3}$$

where n_i and n_j are number of nodes covered by node i and j. Note that we do not need any true label information to compute E_1, so this is unsupervised autoencoders learning.

2.2 Forward Propagation

To get a single meaning of a sentence, we need to know the tree structure for n children nodes and encode pairs of two children nodes until we get a root node. We use greedy algorithm to find the tree structure: First, we compute $n-1$ reconstruction error for each pair of nodes, and then combine the pair of nodes with smallest error. After that, we compute $n-2$ reconstruction error for pairs of the rest nodes and the new node. Finally, we can create a root node using this procedure. The reconstruction error for the whole sentence is

$$\sum_{k \in T} E_1(k) \tag{4}$$

where T is the set of non-leaf nodes of the tree. Greedy algorithm could find the tree structure faster than exhaustive enumeration, but not always optimal.

We introduce a target value t for each sentence. In this work, we only have binary labels for each sentence. We define the sentiment p of each sentence is a binomial distribution for two states: positive and negative. We can then compute the sentiment error using log loss between target value and sentiment output

$$E_2(k) = -\sum_{i=1}^{2} t_i \log p_i \tag{5}$$

where the sentiment output p is computed from meaning vector c_k using logistic regression classifier:

$$p = \text{softmax}(W_{label} \cdot c_k) \tag{6}$$

where W_{label} is a semi-supervised auto encoder in $\mathbb{R}^{2 \times d}$ that need to be learned. Here we apply sentiment error to all non-leaf nodes.

2.3 Learning by Backward Propagation

Now we can combine reconstruction error and sentiment error and form an objective function over N labeled training sentences:

$$J = \frac{1}{N} \sum_{<s,t> \in S} E(s,t,\theta) + \frac{\lambda}{2} \|\theta\|^2 \tag{7}$$

where $\theta =< W_1, b_1, W_2, b_2, W_{label} >$ is all parameters that need to be learned, λ is the strength of the regularization term, and $E(s,t,\theta)$ is the total loss of one sentence with target value t from the training data:

$$E(s,t,\theta) = \sum_{k \in T} \alpha E_1(k) + (1-\alpha) E_2(k) \tag{8}$$

where T is the set of non-leaf nodes and α is the hyperparameter to give different weights between reconstruction error and sentiment error. Now we can learn the parameter θ using gradient descent to minimize our objective function J

$$\theta := \theta + \frac{dJ}{d\theta} \tag{9}$$

We use backward propagation algorithm to compute the gradient efficiently. The element W_{ij} is a scalar in ith row and jth column of W matrix, and it affect the total loss J through the subnode a_i of the activation a. Therefore,

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial a_i} \frac{\partial a_i}{\partial W_{ij}} = \delta_i \frac{\partial a_i}{\partial W_{ij}} \tag{10}$$

Since $a = [W, b][c_i; c_j; 1]$ and $r = h(a)$, so

$$\frac{\partial a_i}{\partial W_{ij}} = [c_i; c_j; 1]^T \tag{11}$$

To compute δ_i, we need to consider two cases. First, for each output node with vector value r and loss function L,

$$\delta_i = \frac{\partial J}{\partial a_i} = \frac{\partial L_i}{\partial a_i} = \frac{\partial L_i}{\partial r_i} \frac{\partial r_i}{\partial a_i} = \frac{\partial L_i}{\partial r_i} h'(a_i) \tag{12}$$

and for each internal node, consider its activation a feed into its parent nodes indexed by k

$$\delta_i = \frac{\partial J}{\partial a_i} = \sum_k \sum_{p=1}^{d} \frac{\partial J}{\partial a_p^k} \frac{\partial a_p^k}{\partial a_i} = \sum_k \sum_{p=1}^{d} \delta_k^p \frac{\partial a_p^k}{\partial a_i} \tag{13}$$

where δ_p^k is from its parent node, and this is the place where backpropagation happens. Note that $a^k = W[...; h(a); ...; 1]$, so

$$\frac{\partial a_p^k}{\partial a_i} = h'(a_i)V_{pi}^k \tag{14}$$

where V^k is the portion of the weight matrix W that multiplies $h(a)$. Therefore, the whole delta vector for each internal node is

$$\delta = h'(a) \circ [\sum_k (\delta^k)^T V^k]^T \tag{15}$$

where $\circ$ is point-wise multiplication.

2.4 Numerical Differentiation

We perform sanity check by comparing the gradient values computed between backpropagation and numerical differentiation. The numerical value is computed by central difference:

$$\frac{\partial J}{\partial W_{ij}} = \frac{J(w_{ij}+\epsilon) - J(w_{ij}-\epsilon)}{2\epsilon)} + O(\epsilon^2) \tag{16}$$

The time complexity of numerical differentiation is $O(d^4)$ because we have d^2 elements in W_{ij}. For each element we need to compute $J(w_{ij})$ in $O(d^2)$, so total time complexity would be $O(d^2 \cdot d^2) = O(d^4)$. However, we only need $O(d^2)$ for backpropagation because it only requires simple point-wise multiplication for d^2 elements in W_{ij}. To traverse and compare all n nodes in a tree, the complexity is $O(nd^2 + n^2)$. In our case, $d^2 >> n$ so the complexity can be approximated by $O(nd^2)$.

3 Experimental Results

In this section, we explain our experiment setup, investigate the trained model, and evaluate the testing accuracies under different parameter setting.

3.1 Experiment Setup

We use movie reviews dataset[1] mentioned in [1]. There are total 10662 sentences and 14043 words in the dictionary. We randomly select 2000 sentences for training set and 200 sentences for testing set with vector length d= 20 and α = 0.05 in our experiment.

3.2 Investigation of Trained Model

In this section we analyze our model in four different ways. First we show the ten most positive and negative words in Table 1. We can not observe the strong relationship between words and sentiments. It is probably because we do not apply sentiment error to leaf nodes, so the learning process does not try to minimize the sentiment error for leaf nodes. Therefore, input word vectors can not be mapped to target class by applying learned weight matrix.

Positive words	what; current; group; Romano; But; take; received; officer; fire; difference
Negative words	last; crashed; union; than; being; barrel; bounce; imprisoned; to; of;

Table 1: Ten most positive and ten most negative words.

[1]http://www.cs.cornell.edu/people/pabo/movie-review-data/

Second, we show the ten phrases predicted to be most positive and most negative from our testing set in Table 2 and Table 3. We can clearly see that our trained model capture correct sentiments for sentences. This is because our model try to minimize the sentiment error for sentences level instead of individual words.

Number	Positive Sentences
1	a tremendous piece of work .
2	slight but enjoyable documentary .
3	a moving and important film .
4	touch !
5	*UNKNOWN* professional but finally slight .
6	psychologically revealing .
7	*UNKNOWN* mean-spirited and wryly observant .
8	a well-executed spy-thriller .
9	sexy and romantic .
10	exciting documentary .

Table 2: Ten most positive phrases.

Number	Negative Sentences
1	i hate this movie
2	silly , loud and goofy .
3	should have been someone else
4	the action *UNKNOWN* s just pile up .
5	the script is too mainstream and the psychology too textbook to intrigue .
6	aggravating and tedious .
7	... overly melodramatic ...
8	... silly *UNKNOWN* ...
9	... unbearably lame .
10	without the *UNKNOWN* satiric humor .

Table 3: Examples of n-grams from the training data of the movie review dataset.

Third, we choose one interesting sentence:

"one big blustery movie where nothing really happens . when it comes out on video , then it 's the perfect cure for insomnia ."

and then we find the most similar sentence by selecting the smallest l_2 norm distance of the meaning vectors among all other sentences:

"it won't be long before you'll spy at a video store near you ."

It is interesting that the model could produce similar meaning vector for similar sentences. It seems that the RAE model could group words and phrases with similar meaning together in the meaning space.

Finally, we pick an interesting sentences and show the tree structure that the greedy algorithm finds for it in Figure 1. The greedy algorithm could find reasonable parsing for sentences in our implementation.

3.3 Accuracy Evaluation

We discuss whether it is necessary to adjust the vector representing each word and normalize the meaning vector for words and phrases. Table 4 shows four different experiment settings using the

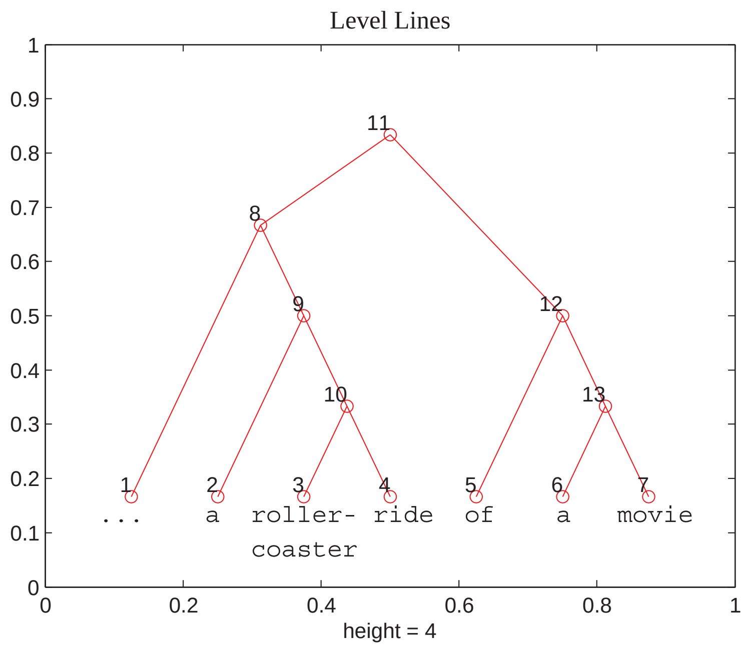

Figure 1: Tree structure found by greedy algorithm.

same training and testing set with the same initialization parameters. Setting 1 achieves the highest training and testing accuracy. However, we can not see huge difference between setting 2,3 and 4 probably because we only use meaning vector with length 20, and therefore gradient descent algorithm could be trapped in local minimum.

Setting	Normalization	Meaning Vector Learning	Training Acc	Testing Acc
1	$\vee$	$\vee$	89.0%	76.0%
2			52.8%	48.5%
3	$\vee$		50.9%	51.0%
4		$\vee$	50.5%	48.5%

Table 4: Training and testing accuracies under different setting.

4 Discussion

In this report, we apply RAE for sentiment prediction. We implement efficient forward and backward algorithms for gradient computation which has complexity $O(nd^2)$. The computed gradients are validated by comparing to the gradients computed from numerical approximation. We control two factors 1) transfer function normalization and 2) word meaning vector adjustment to investigate their effects on training and inference. We find the algorithm is more robust in avoiding local minimum and faster convergence by adding transfer function normalization and word meaning vector adjustment in the RAE algorithms.

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 *EMNLP '11 Proceedings of the Conference on Empirical Methods in Natural Language Processing.*, 2011.