word2vec and sentiment analysis

*Report for Project 4

GuoZhen She
15307130224
Fudan University

***Abstract*—Word2vec is a toolkit open sourced by Google in 2013, which aimed to get accessetd to word vector. But Tomas Mikolov, the author of the word2ec, do not refer to the detail of the algorithm in his published paper, which mysterify the word2vec in some extent. In this paper, we will focus on the detail code of the word2vec, and complete this model step by step. During the process of exploring the mystery of the word2vec, I read two passage([15,16]) from the CSDN, and the second paper of Tomas Mikolov[6], which tell how the word2vec can preedict the sentiment of sentences.**

***Index Terms*—Word2Vec, Sentimental Analysis, SGD, Regularization**

I. Introduction

This paper is a report for implementation and analysis of word2vec model, the following part is included in the report.
1.The mathametical elementary of the word2vec, including how the gradient of the loss function be derived.
2.The construction of the whole code framework
3. The application of the word2vec(Sentimental analysis of sentences), and the routine normalization part.

II. The Elementary of the word2vec

A. Intuition of word2vec

The intuition of the word2vec is encoding(embedding) the word as a vector, then we can tell the characteristic of the word from the embedding vector, for example, we can compare two words vector to determine the relativity of the words, and we can integrate the words into a cluster(sentence,passage), and do further analysis of the passage.

B. Framework of the skipgram method

To clarify the word2vec, I will go through the whole process of training the word2vec network.

1) Preparation and Raw Material: In this paper, we use the corpus provided by standford university[1], the corpus contains many sentences in English, and it also form a word dictionary. The ultimate goal is to assign each word a vector(the length of the vector is setted before the training).
How we can achieve this goal? The next part illustrate our steps.

2) The Idea of word2vec: What we will do is to treat the cluster of the vectors as the parameters of a loss function(also known as the weight matirx of a nueral network), and use gradient descent method to get the parameters value which minimize the loss funcition. Then we elaborate the framework step by step.
1. For each sentence in the corpus, we split it into a few sliding window, in each windows we have a center- word and a few neighbor-word. And we predict the probability of the occurence of the neighbor-word using the parameters in our system(the detail of calculation will be elaborated in next part.)
2.We use cross entropy[] to calculate the loss of prediction for this sliding window, in other word, the total loss of the corpus is to add all the sliding windows' error in all the sentences in the corpus. 3.Use gradient descent method on the loss function to modify the parameter.(The close-form fomula of the gradient for each parameter will be derived in the next part) 4. We get the word vector from the ultimate value of paramter.(In our cases, we still need a trick on parameters to get the final word vector.)

III. Mathematical Part of Word2Vec

A. The Architecture of the Word2Vec Model

First thing first, we have to claim a strange but reasonable idea of the word2vec model: Each word in dictionary contains **two vector representation** in our parameter model, the ultimate vector of our words is to add the two represention.

The first representation is treated as input($\vec{v}$), and the second representation is treated as ouput($\vec{u}$).
Before our frastrating mathematical derivation of the gradient form, we should clarify a few notation.
$\vec{v_c}$:the vector value of current center word c for skipgram model.
$\vec{y}$: the actual probability of occurence of all the word in the dictionary, and form a vector.
$\vec{y_o}$: part of the $\vec{y}$, which contains the neighbor-words in the sliding windows.
$\vec{\hat{y}}$: the predict probability of occurence of all the word in the dictionary, and form a vector.

$\vec{u_i}$: the output vector of the ith word in dictionary
U: Aggreation of $\vec{u}, namely, [\vec{u_1}, \vec{u_2}...\vec{u_w}]$
Here is the detail step of how we get the probability of the

word.

Algorithm 1 CACULATION FOR WORD PROBABILITY

INITIALIZATION: LOSS = 0
INPUT:Sliding Windows set S
for Each sliding window s_i in S **do**
  Choose the center word v_c
  for Each neigher word u_j in s_i **do**
    $\hat{y_j} = \frac{exp(u_j^T v_c)}{\sum_{w=1}^{W} exp(u_w^T v_c)}$
    $LOSS = LOSS + log(\hat{y_j})$
  end for
end for
OUTPUT: LOSS = 0

B. The Gradient of the Paramater

In this part we will discuss how to induct the gradient of each parameter based on our algorithm. Here are a few tricks that we need to stress:
1.The for loop form described above can be vectorized into matrix.
2.The paritial deriavitive to a vector is defined as vector form, and the partial deriavative to a matrix is defined as matrix form.(This will facilitate the coding style using numpy.)

1) For Each Sliding Window: Let's consider the situation of in one sliding windows(Sincely loss in each windows is independent.), note that U_o is The matrix consist of neighbor-words
The loss function is $Jce(U_o, v_c, U) = \sum_{u_i \in U_o} y_i * log(\hat{y_i})$, and here are the deriviation to two parameters.
Pay attention: Two Loss function format will result in different gradient

Normal Cross Entropy Loss Function

$a. \frac{\partial CE(\vec{y},\vec{\hat{y}})}{\partial \vec{v_c}}$
$= -\frac{\partial}{\partial \vec{v_c}} \Sigma_{w=1}^{W} (y_w log(\hat{y_w}))$
$= -\frac{\partial}{\partial \vec{v_c}} y_o log(\hat{y_o})$
$= -\frac{\partial}{\partial \vec{v_c}} y_o log(\frac{e^{\vec{u_o}^T \vec{v_c}}}{\Sigma_{m=1}^{m=W} e^{\vec{u_m}^T \vec{v_c}}})$
$= -y_o(u_o - \Sigma_{x=1}^{x=W} \hat{y_x} \vec{u_x})$

Because y_o = 1,and we can plugin in a few zero into the vector to form a matrix
$\frac{\partial CE(\vec{y},\vec{\hat{y}})}{\partial \vec{v_c}} = U^T(\hat{y} - y)$ (This form)

b.

$$\frac{\partial CE}{\partial U} = \begin{bmatrix} -\frac{\partial}{\vec{u_1}}(y_o log(\hat{y_o})) \\ | \\ -\frac{\partial}{\vec{u_o}}(y_o log(\hat{y_o})) \\ | \\ -\frac{\partial}{\vec{u_w}}(y_o log(\hat{y_o})) \end{bmatrix}$$

Choose the i th column, you will get:

$$-\frac{\partial}{\vec{u_i}}(y_i log(\hat{y_i})) = (\hat{y_i} - y_i)\vec{v_c}$$

Then combine the each column

$$\frac{\partial(CE(y,\hat{y}))}{\partial U} = \begin{bmatrix} (\hat{y_1} - y_1)\vec{v_c} \\ | \\ (\hat{y_o} - y_o)\vec{v_c} \\ | \\ (\hat{y_w} - y_w)\vec{v_c} \end{bmatrix}$$

$$. \qquad = \vec{v_c}(\vec{\hat{y}} - \vec{y})^T$$

Negative sampling

Pay attention: The parameter of the loss function is U(Composed of U_o and $U - U_o$, which are quite different when conduct deriavative),$\vec{v_c}$,

$$a. \frac{\partial J}{\partial v_c} = (\sigma(\vec{u_o}^T \vec{v_c}) - 1)\vec{u_o} + \Sigma_{k=1}^{K}(-\sigma(-\vec{u_o}^T \vec{v_c}) + 1)\vec{u_k}$$

$$b. \frac{\partial J}{\partial u_o} = (\sigma(\vec{u_o}^T \vec{v_c}) - 1)\vec{v_c}$$

$$c. \frac{\partial J}{\partial u_k} = (-\sigma(\vec{u_o}^T \vec{v_c}) + 1)\vec{v_c}$$

2) Compare the Negative Sampling And Normal Loss: Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them. Heres how it works.

Consider the word pair('fox','quick'), recall that the label or correct output of the network is a one-hot vector. That is, for the output neuron corresponding to quick to output a 1, and for all of the other thousands of output neurons to output a 0.

With negative sampling, we are instead going to randomly select just a small number of negative words (lets say 5) to update the weights for. (In this context, a negative word is one for which we want the network to output a 0 for). We will also still update the weights for our positive word (which is the word quick in our current example).

3) Intergrate For All the Sliding Windows: Pay attention to the fact that we use a function F as a wrapper in sbubstitute for the lone sliding window. The reason can be attributed to make a more clear interface.

$$\frac{\partial J_{skip-gram}(word_{c-m,\ldots c+m})}{\partial U} = \sum_{-m<j<m} \frac{\partial F(w_{\vec{w+j}}, \vec{v_c})}{\partial U}$$

$$\frac{\partial J_{skip-gram}(word_{c-m,\ldots c+m})}{\partial \vec{v_c}} = \sum_{-m<j<m} \frac{\partial F(w_{\vec{w+j}}, \vec{v_c})}{\partial \vec{v_c}}$$

$$\frac{\partial J_{skip-gram}(word_{c-m,\ldots c+m})}{\partial \vec{v_j}} = 0 \qquad All \quad j \neq c$$

C. Python Implementation of Word2vec

Thanks to numpy package in python, the matrix manipulation in python is just as easy as in matlab.

D. Visulization of Word2Vec

1) Elementary of Visualization—-PCA: Since We got the paramter of the WOrd2Vec, the next step is to visualize these words to validate the authenticity of our training, namely, we need to see the word with similar mean should be close to each other, while the word of different speech should be as far away as possible.

To visualize it on a 2-D picture, it occurs to us that the PCA is a good choice, here we give out our PCA implementation(the python code in run.py is more specific)

Algorithm 2 The Decrease of Dimension

1: Input:The words's vectors: X = $[w_1, w_2...w_n]$
2: Standardization:For Each $w_i = w_i - \frac{1}{m}\sum_{i=1}^{m} w_i$
3: Covariance Matrix: T = XX'
4: Get the eig vector of the T:W = $[w_1, w_2...w_k]$
5: Choose the 2 largest eig vector: W* = $[w_1, w_2]$
6: Project to W*: $[w_1*, w_2 * ...w_n*] = X * W*$ coordinates:$[(w_1 * (1), W * (2)), (w_2 * (1), w_2(2)), (w_3 * (1), w_3 * (2))]$

2) Analysis Of PCA Visualization: There are three paramters of the Word2Vec we can handle

- Sliding Window Size
- Iteration Times
- Word Vector Size

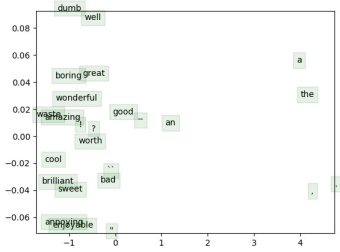

Fig. 1. Sliding Window Size = 5

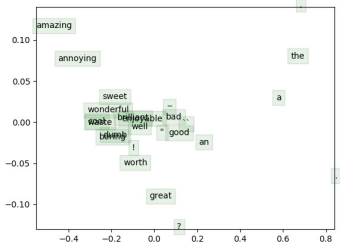

Fig. 2. Sliding Window Size = 10

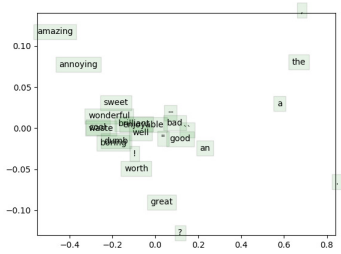

Fig. 3. Sliding Window Size = 20

E. Apllication of Word2vec: Sentimental Analysis

Sentimental Analysis is often the most popular apllication of many NLP alogrithm, the state-of-art algorithm for sentimental analysis contains LSTM, Beyes, but the interpretality of these two methods is very poor. But when we see the word vectorization model, it occurs to us that this model should be a brand-new and high-intepretable method,since we use the word2vec to get the quantative representation of each word, we can model on a sentence(passage) with the word in it, then to use the normal machine learning algorithm.

Algorithm 3 The Pipeline for sentimental analysis

1: Encode the five sentimental words into the 0-4very negative, negative, neutral, positive, very positive
2: Seperate the standford corpus into training, test set.
3: Map the word vector to sentence vector for each sentence, In our common article is use the average word vector to represent sentence vector
4: **for**
5: **do**
Choose a bias b_i for the bias setUse softmax regression(cross entropy,and add bias b_i) on training set to get the regressor. Use the regressor on test set to report the test error rate on r_i Add r_i to R
6: 9: **end for**
10: Output: choose the b_i subject to b_i = argmax$[r_1, r_2...r_n]$

Error Rate / Set / Bias	$training$	$test$
0.00001	28.405899	27.157130
0.00004	28.581461	27.157130
0.00007	28.452715	27.520436
0.0001	28.405899	26.884650

1) The Softmax Regressor Training: As mentioned in the algorithm above, we need to adjust the regularization bias weight, to get the best performance of the regularization value.I use the regularization set from [0.000001,0.00001,0.0001,0.001,0.01,0.1], The Figure.4 is used to determine the optimal regularizaion of the sofmax regressor. Then We get the optimal value is between 0.00001 and 0.0001, then we do the fine tuning in the following part. We choose the [0.00001,0.00003,0.00006,0.00009] to narrow down the range of the regularization.

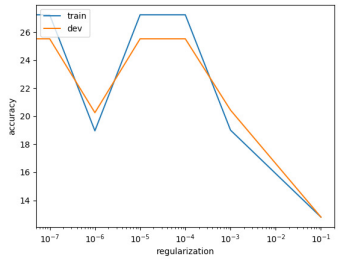

Fig. 4. Overall

2) Result of the optimal value of regularization: The optimal value of the 0.0000614, and which can be show in the Fig.5

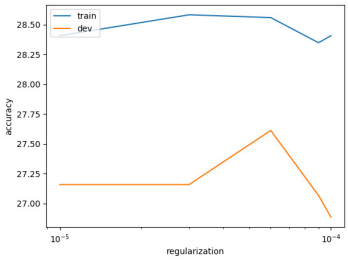

Fig. 5. Fine Tuning

F. Callback

This report derive the elementary of word2vec in our own way, which contains the close form gradient of the loss function, and the vectorization of the caculation, and in the second part, we use the word vector into the sentimental analysis task, and import the softmax regression(similar to logistic regression) to do the normal machine learning, whcih seperate the corpus into training set, and development set.