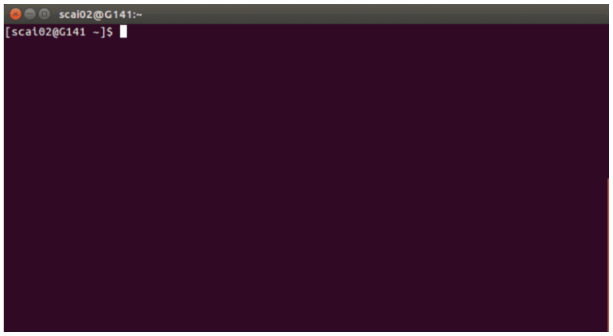
Word2vec: Introduction to word embedding

Prepared by Minghong Yao
Advisor: Prof. Jie Wang
Machine Intelligence Research and Applications (**MIRA**) Lab @ USTC
send comments to yaominghong1@gmail.com

Introduction: In nature language processing tasks, each word in the text should be coded somehow in someway. The word2vec model accomplishes this mission by assigning each word a dense vector. In this project, you will implement the famous word2vec model with python. We have prepared the skeleton code for you and the places where you need to fill in are remarked with "YOUR CODE HERE" and "END YOUR CODE".

Exercise 1: Download the Skeleton Code and Data Set(5 pts)

Please make sure that you have logged in the server. For example, if your account is "scai02", then you should be able to see this window:



You don't need to worry about the python environment but you need to download the skeleton code and data set from gitlab. You can use the following command to download these material:

git clone http://192.168.6.49/mhyao/Word2Vec.git

When you are done, please type the command "**ls**" to check whether there is a folder named "Word2Vec". Then you should type the command "**cd Word2Vec**" to get into this folder. Once again, type the command "**ls**" to list all the files in this folder and you should be able to see a file named "utils.tar.gz". Before the final step, you should type the command "**pwd**" to show the pass of current folder and you should be able to see "/home/scai02/Word2Vec" on the screen. Finally, you should type the following command to unzip the data set.

tar xzvf utils.tar.gz -C /home/scai02/Word2Vec

You can also type "**ls**" to check whether there is a new folder named "utils".

Congratulations ! Now you are ready to start your coding. The skeleton code for different questions are **Skipgram-negsam.py**, **Skipgram-SMCE.py**, **softmax.py**, and **sigmoid.py**.

You can use "vim" to edit these skeleton code and complete them. You should google how to use "vim" editor if you are not familiar with it.

Exercise 2: Some Exercises About Numpy Module(15 pts)

We design this exercise to get you familiar with the numpy module. You can google the official docs of numpy module on the internet (**https://docs.scipy.org/doc/numpy/**). When you are prepared enough, you should be able to implement the following functions by numpy. These functions are important in word2vec model.

We can start from the sigmoid function $\sigma : \mathbb{R} \to \mathbb{R}$ and it is defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Then the softmax function $SM : \mathbb{R}^n \to \mathbb{R}^n$ is defined as:

$$SM(x_i) = \frac{\exp(x_i)}{\sum_{k=1}^{n} x_k}$$

Where x_i is the i th element of input vector $\mathbf{X}$. We can interpret the result of softmax function as a probability distribution (given that each element of it's result is positive and the summation equals to 1).

However, we will modify the definition of softmax function when we implement it in the computer. If x is a large positive number, then $\exp(x)$ may overflow. For each element in vector $\mathbf{X}$, we minus it with the max element of $\mathbf{X}$ to overcome this issue. i.e. the following equality holds for all $\mathbf{X}$

$$SM(\mathbf{X}) = SM(\mathbf{X} - \mathbf{C})$$

Where $\mathbf{C}_i = \max(\mathbf{X}),\ \forall i \leq n$.
The cross entropy loss function is defined as:

$$CE(\mathbf{Y}, \hat{\mathbf{Y}}) = -\mathbf{Y} \log \hat{\mathbf{Y}}$$

Where $\mathbf{Y}, \hat{\mathbf{Y}} \in \mathbb{R}^n$ stands for two different probability distribution. Here is an example:

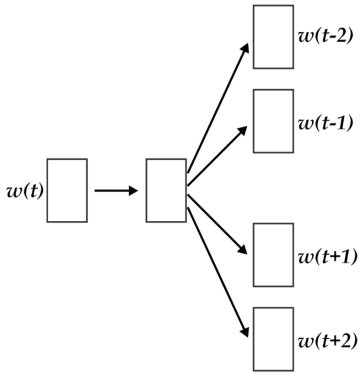
$$\begin{cases} \mathbf{Y} = [0, 0, \ldots, 1, \ldots, 0], \ \textit{where only the i'th element of } \mathbf{Y} \textit{ is } 1. \\ \hat{\mathbf{Y}} = SM(\mathbf{X}) \\ CE(\mathbf{Y}, \hat{\mathbf{Y}}) = -\log(SM(x_i)) \end{cases}$$

The questions for you are:

1. Please google the interpretation of the sigmoid function and record it in your answer. Also, please implement it using numpy module in *sigmoid.py*. If your implementation is valid, you should be able to pass the testing code in *sigmoid.py*.
2. Please prove $SM(\mathbf{X}) = SM(\mathbf{X} - \mathbf{C})$ and implement the softmax function using numpy module in $softmax.py$. If your implementation is valid, you should be able to pass the testing code in $softmax.py$.
3. Please google the interpretation of cross entropy loss function and summarize it briefly in your answer.

Exercise 3: Introduction to Word2vec Model(80 pts)

We will recall the details of Word2vec model firstly. We believe that the words appear in similar context should have similar meanings. Hence, we invent the concept of word window. The word in the center of each window is called the center word and the rest of words are called context words. In word2vec model, we either predict the center word according to the context words (CBOW model) or predict the context words according to the center word(Skip-gram model). We make predictions by employing a shallow neuron network. Here are the details.



Skip-gram

Let the word window's length to be 5 and it contains the following 5 words now:

$$"w(t-2), w(t-1), w(t), w(t+1), w(t+2)"$$

In Skip-gram model:

(1) Firstly, we assign each word an unique ont-hot vector $\mathbf{Y}$ according to its index in the vocabulary.

(2) Then, we initialize each word with two random vectors $\mathbf{v}$ and $\mathbf{u}$. We call $\mathbf{v}$ as an input vector and $\mathbf{u}$ as an output vector. We also call the collection of all $\mathbf{v}$ as the input matrix $\mathbf{V}$. i.e. $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N]$. Similarly, we define $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]$ and named it after the output matrix. N denotes the size of vocabulary.

(3) Denote the center word $w(t)$'s input vector as $\mathbf{v}_c$. The skip-gram model will calculate a score vector in the following way:

$$z = [\mathbf{v}_c \cdot \mathbf{u}_1, \mathbf{v}_c \cdot \mathbf{u}_2, \ldots, \mathbf{v}_c \cdot \mathbf{u}_N]$$

z_i can be interpreted as the similarity between $\mathbf{v}_c$ and $\mathbf{u}_i$. $\mathbf{v}_c \cdot \mathbf{u}_i$ means we take the dot product between them.

(4) Substitute z into softmax function and we get the predicted probability distribution $\hat{\mathbf{Y}}$.i.e.

$$\hat{\mathbf{Y}} = SM(z)$$

(5) For each context word in the word window, $w(t-1)$ for example, let $\mathbf{Y}$ denote its ont-hot vector and let $\mathbf{u}_o$ denote its output vector. Then, the skip-gram model will calculate the cross entropy loss of this pair of training sample $(w(t-1), w(t))$:

$$J = CE(\mathbf{Y}, \hat{\mathbf{Y}}) = -\log(\hat{y}_o) = -\log\left(\frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{w=1}^{V} \exp(\mathbf{u}_w^\top \mathbf{v}_c)}\right)$$

Where $\mathbf{u}_w$ denotes all the output vectors in vocabulary.
Then the skip-gram model will calculate the following gradient

$$\frac{\partial J}{\partial \mathbf{u}_o} \ \& \ \frac{\partial J}{\partial \mathbf{u}_w} \ \& \ \frac{\partial J}{\partial \mathbf{v}_c}$$

(6) At last, the skip-gram model will calculate the loss summation of all context words in this word window and denote this summation as $J_{skip-gram}$. Then, calculate the gradient of $J_{skip-gram}$ with respect to all input vectors and output vectors and update them accordingly.

(7) When we train this model for enough time or the loss is small enough, we take each word's output vector as its word embedding.

The questions for you are:

1. Please derive $\frac{\partial J}{\partial \mathbf{u}_o}$ & $\frac{\partial J}{\partial \mathbf{u}_w}$ & $\frac{\partial J}{\partial \mathbf{v}_c}$ precisely. [Hint: a good answer should contain the **V,U** matrix and this will help you a lot when you are coding.]

2. Please derive $\frac{\partial J_{skip-gram}}{\partial \mathbf{u}_o}$ & $\frac{\partial J_{skip-gram}}{\partial \mathbf{u}_w}$ & $\frac{\partial J_{skip-gram}}{\partial \mathbf{v}_c}$ & $\frac{\partial J_{skip-gram}}{\partial \mathbf{v}_w}$. [Hint: feel free to use the notations in problem 1.]

3. Complete the implementation of the cross entropy cost and gradients in problem 1 and 2 in $Skipgram - SMCE.py$. Specifically, you should fill in the $softmaxCosstAndGradient$ and $skipgram$ function in $Skipgram - SMCE.py$. Please use the numpy module and don't turn to other fancy tools such as tensorflow. If your coding is valid, you should be able to pass the gradient check.

However, you will find the training process of this model is very slow and the loss decreases even slower. Indeed, this model is not practical in real application. You may think about the reason and record it in your answer. We recommend you to stop the training process and turn to the next problem as long as you pass the gradient check.

4. In real application, the skip-gram model will employ the negative sampling technic to speed up the training process instead of using softmax and cross entropy function. The negative sampling technic will sample K words outside this word window and calculate the cross entropy loss for each pair training sample in the following way:

$$J = -\log(\sigma(\mathbf{u}_o^\top \mathbf{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\mathbf{u}_k^\top \mathbf{v}_c))$$

   Please repeat the problem 1 in this situation and complete the corresponding implementation in *Skipgram−negsam.py*. You should fill in the *negSamplingCostAndGradient* function and *skipgram* function in *Skipgram − negsam.py*. Please use the numpy module and don't turn to other fancy tools such as tensorflow. [Hint: a good implementation will take about 1 hour to finish the training process. Arrange your schedule reasonably.]

5. Please collect the figures generated in these problems and put them in your answer. Make an observation and record your findings in your answer.