文本表征第二次实验

注：本次实验使用 enwiki8 (http://mattmahoney.net/dc/enwik8.zip) 语料库，及 https://github.com/deborausujono/word2vecpy的模型

性能结果

相似度计算使用余弦相似度

Spearman-correlation如下：

方法	CBOW	SG
HS	0.24	0.57
NS	0.23	0.36

其他超参数
- 向量维度:200
- 窗口大小:+/- 5词
- 词频过滤:5

- Number of negative examples:5

分析比较

在控制其他变量不变的情况下，SG得到的向量的相似度，比CBOW的向量更与标准数据相关，即**向量数据SG更好**，但相应的，**计算时间SG也比CBOW长**。
这可能是因为CBOW是通过上下文来预测一个词，这样学习的次数比较少，在遇到生僻词的时候效果比较差，而SG是通过一个词预测上下文，学习的次数比较多，可以更好的处理生僻词

在控制其他变量不变的情况下，HS比NS更好，但在CBOW时无明显优势。
HS使用了一个二叉树（通常是霍夫曼树）来计算词的概率，而NS通过随机抽取负样本来近似整个词汇表的概率分布。HS通常在处理大词汇表时更有效，而NS在小词汇表或者高质量负样本的情况下表现更好，这就能说明HS在SG时优势明显，而在处理CBOW时无明显优势。

实验过程

模型实现

1. **初始化**：模型首先初始化词向量。在这段代码中，`syn0` 和 `syn1` 是词向量的初始化值。`syn0` 是输入词向量，`syn1` 是输出词向量。这两个向量都是随机初始化的。

```
def init_net(dim, vocab_size):
    # Init syn0 with random numbers from a uniform distribution on the interval [-0.5, 0.5]/dim
    tmp = np.random.uniform(low=-0.5/dim, high=0.5/dim, size=(vocab_size, dim))
    syn0 = np.ctypeslib.as_ctypes(tmp)
    syn0 = Array(syn0._type_, syn0, lock=False)

    # Init syn1 with zeros
    tmp = np.zeros(shape=(vocab_size, dim))
    syn1 = np.ctypeslib.as_ctypes(tmp)
    syn1 = Array(syn1._type_, syn1, lock=False)

    return (syn0, syn1)
```

2. **构建词汇表**：模型读取训练数据，构建词汇表，并统计每个词的出现次数。
使用 `vacab` 类实现词汇表

```
class Vocab:
    def __init__(self, fi, min_count):
        vocab_items = []
        vocab_hash = {}
        word_count = 0
        fi = open(fi, 'r')

        # Add special tokens <bol> (beginning of line) and <eol> (end of line)
        for token in ['<bol>', '<eol>']:
            vocab_hash[token] = len(vocab_items)
            vocab_items.append(VocabItem(token))

        for line in fi:
            tokens = line.split()
            for token in tokens:
                if token not in vocab_hash:
                    vocab_hash[token] = len(vocab_items)
                    vocab_items.append(VocabItem(token))

                #assert vocab_items[vocab_hash[token]].word == token, 'Wrong vocab_hash index'
                vocab_items[vocab_hash[token]].count += 1
                word_count += 1

                if word_count % 10000 == 0:
                    sys.stdout.write("\rReading word %d" % word_count)
                    sys.stdout.flush()

            # Add special tokens <bol> (beginning of line) and <eol> (end of line)
            vocab_items[vocab_hash['<bol>']].count += 1
            vocab_items[vocab_hash['<eol>']].count += 1
            word_count += 2
```

3. **训练**：模型会遍历训练数据中的每一个词，对每一个词，模型会查找它的上下文词（在 CBOW 中）或预测它的上下文词（在 Skip-gram 中）。模型会计算损失函数关于词向量的梯度，然后使用这个梯度来更新词向量。这个过程使用梯度下降法来完成。在 `train_process` 中完成，对于不同Method，使用不同过程，比如参数NS=0时，使用Huffman树

> **Note**
>
> 该模型使用了并行计算来缩短计算时间，在train()中有如下代码

```
  You, 5天前 • update
def train(fi, fo, cbow, neg, dim, alpha, win, min_count, num_processes, binary):
    # Read train file to init vocab
    vocab = Vocab(fi, min_count)

    # Init net
    syn0, syn1 = init_net(dim, len(vocab))

    global_word_count = Value('i', 0)
    table = None
    if neg > 0:
        print ('Initializing unigram table')
        table = UnigramTable(vocab)
    else:
        print ('Initializing Huffman tree')
        vocab.encode_huffman()

    # Begin training using num_processes workers
    t0 = time.time()
    pool = Pool(processes=num_processes, initializer=__init_process,
                initargs=(vocab, syn0, syn1, table, cbow, neg, dim, alpha,
                          win, num_processes, global_word_count, fi))
    pool.map(train_process, range(num_processes))
    t1 = time.time()
    print
    print ('Completed training. Training took', (t1 - t0) / 60, 'minutes')

    # Save model to file
    save(vocab, syn0, fo, binary)
```

数据处理

模型接受参数，输出一个含有Vec表的文本文档model.txt

使用 `similarity.py` 处理两个word在输出模型中的相似度，采用余弦相似度，并且把余弦值从[-1,1]映射到[0,1]，以直观的相似度比较

```python
def cosine_similarity(self,word1,word2):
    '''
    计算两个词的余弦相似度
    '''
    vec1=[]
    vec2=[]
    if(self.cbow==1):
        file_name='cbow'
    else:
        file_name='sg'
    if(self.hs==1):
        file_name=file_name+'_hs_'
    else:
        file_name=file_name+'_ns_'
    file_name=file_name+'model_output'
    with open(file_name,'r') as f:
        lines=f.readlines()
        for line in lines:
            if word1+" " in line:
                vec1=[float(i) for i in line.split()[1:]]
            if word2+" " in line:
                vec2=[float(i) for i in line.split()[1:]]

    num = float(np.dot(vec1, vec2))  # 向量点乘
    denom = np.linalg.norm(vec1) * np.linalg.norm(vec2)  # 求模长的乘积
    return 0.5 + 0.5 * (num / denom) if denom != 0 else 0
```

使用 `spearman_correlation.py` 来获得wordsim_similarity_goldstandard.txt的词汇，并不断调用similarity.py来每对词汇的相似度，最终计算与标准相似度的Spearman-correlation。

```python
import similarity
import scipy

for cbow in range(0,2):
    for hs in range(0,2):
        Model=similarity.Similarity(cbow,hs)
        simModel=Model.count_words_similarity()
        simGold=[]
        with open("wordsim_similarity_goldstandard.txt",'r') as f:
            lines=f.readlines()
            for line in lines:
                simGold.append(float(line.split()[2]))
        if cbow==1:
            print("CBOW",end=' ')
        else:
            print("Skip-gram",end=' ')
        if hs==1:
            print("HS",end=' ')
        else:
            print("NS",end=' ')
        print(scipy.stats.spearmanr(simModel,simGold))
```