

作业二：词向量训练

一、实验目的

1. 掌握课堂所讲词向量的基本概念和训练方法。
2. 加强对pytorch、tensorflow等深度学习框架的使用能力。

二、实验要求

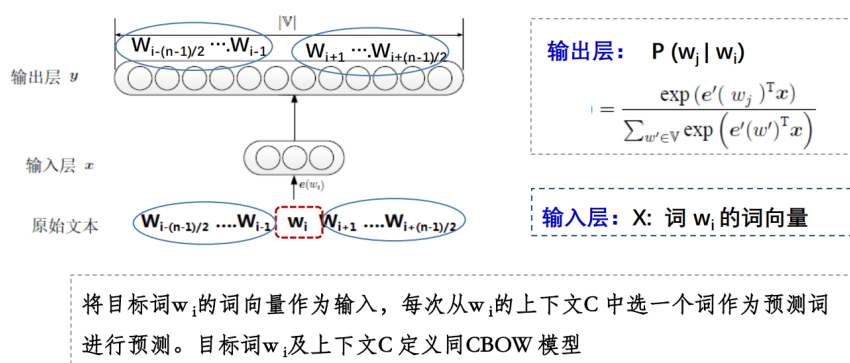
1. 任选课上讲的一种词向量模型进行实现即可，如是其他模型则请写明模型结构，作业压缩文件中也提供给大家相关的一些论文来进行参考。

三、任务说明

1. 任务时间为2周半，分数为15分。
2. 请使用任意一种深度学习框架（推荐pytorch），利用给定的语料训练词向量即可，中文和英文都需要训练，中文语料位于data文件夹下的zh.txt中，已经分好词。英文语料位于data文件夹下的en.txt中。（给定的语料规模较小，如果有充足的计算资源，也可自己选用更大的语料来训练，但需在报告中写明）。
3. 作业需提交代码部分和生成的两个词向量文件（中文、英文）（注意：不需要提交训练好的模型），同时要写一个实验报告来说明自己所使用的模型结构以及训练方式等，也可写自己的遇到的问题和思考（不强制要求）。

一、Skip-gram模型

■ Skip-gram模型



■ 模型学习

- 优化目标: 最大化 $\sum_{(w,c) \in D} \sum_{w_j \in c} \log P(w_j | w)$
- 参数训练: 梯度下降法

```
import torch
import torch.nn as nn
class SkipGram(nn.Module):
    def __init__(self, v: int, embedding_dim=100):
        """
        :param v: the size of vocabulary
        :param embedding_dim: the number of dimensions of word vector
        """
        super(SkipGram, self).__init__()
```

```

self.embedding_dim = embedding_dim
self.in_embeddings = nn.Embedding(V, embedding_dim, sparse=True)
self.out_embeddings = nn.Embedding(V, embedding_dim, sparse=True)
self.reset_parameters()

def reset_parameters(self):
    upper = 0.5 / self.embedding_dim
    self.in_embeddings.weight.data.uniform_(-upper, upper)
    self.out_embeddings.weight.data.zero_()

def forward(self, inputs, contexts, negatives):
    """
    :param inputs: (#mini_batches, 1)
    :param contexts: (#mini_batches, 1)
    :param negatives: (#mini_batches, #negatives)
    :return:
    """
    in_vectors = self.in_embeddings(inputs)
    pos_context_vectors = self.out_embeddings(contexts)
    neg_context_vectors = self.out_embeddings(negatives)

    pos = torch.sum(in_vectors * pos_context_vectors, dim=(1, 2))
    neg = torch.sum(in_vectors * neg_context_vectors, dim=2)

    return pos, neg

```

二、负采样

自然语言处理领域中，判断两个单词是不是一对上下文词（context）与目标词（target），如果是一对，则是正样本，如果不是一对，则是负样本。采样得到一个上下文词和一个目标词，生成一个正样本（positive example），生成一个负样本（negative example），则是用与正样本相同的上下文词，再在字典中随机选择一个单词，这就是负采样（negative sampling）。

不同于原本每个训练样本更新所有的权重，负采样每次让一个训练样本仅仅更新一小部分的权重，这样就会降低梯度下降过程中的计算量。

```

import numpy as np
class NegativeSampler(object):
    def __init__(
        self, frequency: np.ndarray, negative_alpha=0.,
        table_length=int(1e8), is_neg_loss=True,
    ):
        if negative_alpha == 0:
            self.table_length = len(frequency)
            self.negative_table = np.arange(self.table_length, dtype=np.int32)
        else:
            self.table_length = table_length
            z = np.sum(np.power(frequency, negative_alpha))
            negative_table = np.zeros(table_length, dtype=np.int32)
            begin_index = 0
            for index, freq in enumerate(frequency):
                c = np.power(freq, negative_alpha)
                end_index = begin_index + int(c * table_length / z) + 1
                negative_table[begin_index:end_index] = index
                begin_index = end_index

```

```

        self.negative_table = negative_table
    if not is_neg_loss:
        self.noise_dist = np.power(frequency, negative_alpha)

    def sample(self, k, rnd: np.random.RandomState, exclude_words=None):
        """
        :param k: number of negative samplings
        :param rnd: np.random.RandomState
        :param exclude_words: numpy array contains context words
        :return: negative words (#context_words, #negatives)
        """
        size = (len(exclude_words), k)
        if exclude_words is None:
            return self.negative_table[rnd.randint(low=0,
high=self.table_length, size=size)]
        else:
            negs = np.zeros(size, np.int32)
            for i, word in enumerate(exclude_words):
                negs_for_word = self.negative_table[rnd.randint(low=0,
high=self.table_length, size=k)]
                while word in negs_for_word:
                    negs_for_word = self.negative_table[rnd.randint(low=0,
high=self.table_length, size=k)]
                negs[i] = negs_for_word
            return negs

```

三、NCE损失函数

NCE loss的直观想法：把多分类问题转化成二分类。之前计算softmax的时候class数量太大，NCE损失直接把多分类缩减为二分类问题。之前的问题是计算某个类的归一化概率是多少，二分类的问题是input和label正确匹配的概率是多少。二分类问题直接通过logistic regression进行概率估算。

```

import torch
from torch.nn.functional import logsigmoid

def nce_loss(pos_dot, neg_dot, pos_log_k_negative_prob, neg_log_k_negative_prob,
size_average=True, reduce=True):
    s_pos = pos_dot - pos_log_k_negative_prob
    s_neg = neg_dot - neg_log_k_negative_prob
    loss = - (torch.mean(logsigmoid(s_pos)) + torch.sum(logsigmoid(-s_neg),
dim=1)))
    if not reduce:
        return loss
    if size_average:
        return torch.mean(loss)
    return torch.sum(loss)

def negative_sampling_loss(pos_dot, neg_dot, size_average=True, reduce=True):
    loss = - (
        logsigmoid(pos_dot) + torch.sum(logsigmoid(-neg_dot), dim=1)
    )
    if not reduce:
        return loss
    if size_average:
        return torch.mean(loss)
    return torch.sum(loss)

```

四、生成语料库

将分词好的文本文件整理成id_word字典和结构化的语料库（不去除停用词，以保留更多词间信息）

```
import numpy as np
class Dictionary(object):
    def __init__(
        self,
        replace_lower_freq_word=False,
        replace_word='<unk>'
    ):
        self.word2id = {}
        self.id2word = []
        self.word2freq = {}
        self.id2freq = None
        self.replace_lower_freq_word = replace_lower_freq_word
        self.replace_word = replace_word

    def add_word(self, word):
        if word not in self.word2id:
            self.id2word.append(word)
            self.word2id[word] = len(self.id2word) - 1
            self.word2freq[word] = 1
        else:
            self.word2freq[word] += 1

    def rebuild(self, min_count=5):
        self.id2word = sorted(self.word2freq, key=self.word2freq.get,
reverse=True)
        for new_word_id, word in enumerate(self.id2word):
            freq = self.word2freq[word]
            if freq >= min_count:
                self.word2id[word] = new_word_id
            else:
                if self.replace_lower_freq_word:
                    self.word2id[self.replace_word] = new_word_id
                    sum_unk_freq = 0
                    for word in self.id2word[new_word_id:]:
                        sum_unk_freq += self.word2freq[word]
                    del self.word2id[word]
                    self.word2freq[self.replace_word] = sum_unk_freq
                    self.id2word = self.id2word[:new_word_id]
                    self.id2word.append(self.replace_word)
                else:
                    for word in self.id2word[new_word_id:]:
                        del self.word2id[word]
                    self.id2word = self.id2word[:new_word_id]
                break
        self.id2freq = np.array([self.word2freq[word] for word in self.id2word])
        del self.word2freq

    def __len__(self):
        return len(self.id2word)

class Corpus(object):
    def __init__(
        self,
```

```

        min_count=5,
        replace_lower_freq_word=False,
        replace_word='<unk>',
        bos_word='<bos>',
        eos_word='<eos>'
    ):
        self.dictionary = Dictionary(replace_lower_freq_word, replace_word)
        self.min_count = min_count
        self.num_words = 0
        self.num_vocab = 0
        self.num_docs = 0
        self.discard_table = None
        self.replace_lower_freq_word = replace_lower_freq_word
        self.replace_word = replace_word
        self.bos_word = bos_word
        self.eos_word = eos_word

    def tokenize_from_file(self, path):
        def _add_special_word(sentence):
            return self.bos_word + ' ' + sentence + ' ' + self.eos_word

        self.num_words = 0
        self.num_docs = 0
        with open(path, encoding='utf-8') as f:
            for l in f:
                for word in _add_special_word(l.strip()).split():
                    self.dictionary.add_word(word=word)
        self.dictionary.rebuild(min_count=self.min_count)
        self.num_vocab = len(self.dictionary)

        with open(path, encoding='utf-8') as f:
            docs = []
            for l in f:
                doc = []
                for word in _add_special_word(l.strip()).split():
                    if word in self.dictionary.word2id:
                        doc.append(self.dictionary.word2id.get(word))
                    elif self.replace_lower_freq_word:
                        doc.append(self.dictionary.word2id.get(self.replace_word))
                doc.append(self.dictionary.word2id.get(self.replace_word))
                if len(doc) > 1:
                    docs.append(np.array(doc))
                    self.num_words += len(doc)
                    self.num_docs += 1

            return np.array(docs)

    def build_discard_table(self, t=1e-4):
        tf = t / (self.dictionary.id2freq / self.num_words)
        self.discard_table = np.sqrt(tf) + tf

    def discard(self, word_id, rnd):
        return rnd.rand() > self.discard_table[word_id]

```

五、基础设置

```
import logging
import numpy as np
import torch
from torch import optim
from loss import negative_sampling_loss, nce_loss
from model import SkipGram
from utils.negative_sampler import NegativeSampler
from utils.vocab import Corpus

NEGATIVE_TABLE_SIZE = int(1e8)
ws = 2
num_negatives = 5
epochs = 20
num_minibatches = 512
starting_lr = 0.001 * num_minibatches
lr_update_rate = 1000
embedding_dim = 100
seed = 7
#固定初始随机赋值，确保每次运行.py文件时，生成的随机数都是固定的
rnd = np.random.RandomState(seed)
torch.manual_seed(seed)
```

六、函数配置

```
#更新学习率
def update_lr(starting_lr, num_processed_words, epochs, num_words):
    new_lr = starting_lr * (1. - num_processed_words / (epochs * num_words + 1))
    lower_lr = starting_lr * 0.0001
    return max(new_lr, lower_lr)
```

```
def generate_words_from_doc(doc, num_processed_words, corpus, rnd):
    new_doc = []
    for word_id in doc:
        num_processed_words += 1
        if corpus.discard(word_id=word_id, rnd=rnd):
            continue
        new_doc.append(word_id)
        if len(new_doc) >= 1000:
            yield np.array(new_doc), num_processed_words
            new_doc = []
    yield np.array(new_doc), num_processed_words
```

七、批量样本训练

```
def train_on_minibatches(model, optimizer, inputs, contexts, negatives,
is_neg_loss, log_k_prob=None):
    num_minibatches = len(contexts)
    #inputs填充成一行数据
    inputs = torch.LongTensor(inputs).view(num_minibatches, 1)
    optimizer.zero_grad()

    if is_neg_loss:
        contexts = torch.LongTensor(contexts).view(num_minibatches, 1)
```

```

        negatives = torch.LongTensor(negatives)
        pos, neg = model.forward(inputs, contexts, negatives)
        loss = negative_sampling_loss(pos, neg)
    else:
        pos_log_k_negative_prob =
torch.FloatTensor(log_k_prob[contexts]).view(num_minibatches, 1)
        neg_log_k_negative_prob = torch.FloatTensor(log_k_prob[negatives])
        contexts = torch.LongTensor(contexts).view(num_minibatches, 1)
        negatives = torch.LongTensor(negatives)
        pos, neg = model.forward(inputs, contexts, negatives)
        loss = nce_loss(pos, neg, pos_log_k_negative_prob,
neg_log_k_negative_prob)
        loss.backward()
        optimizer.step()
    return loss.item()

```

八、主函数

```

def main():
    logger = logging.getLogger(__name__)
    logger.setLevel(logging.INFO)
    stream_handler = logging.StreamHandler()
    stream_handler.setLevel(logging.INFO)
    stream_handler.terminator = ''
    logger.addHandler(stream_handler)
    logger.info('Loading training corpus...\n')
    corpus = Corpus(min_count=5)
    docs = corpus.tokenize_from_file("../data/zh.txt") #en.txt
    corpus.build_discard_table(t=1e-3)
    logger.info('V:{}, #words:{}\n'.format(corpus.num_vocab, corpus.num_words))
    is_neg_loss = True
    negative_sampler = NegativeSampler(
        frequency=corpus.dictionary.id2freq,
        negative_alpha=0.75,
        is_neg_loss=is_neg_loss,
        table_length=NEGATIVE_TABLE_SIZE
    )
    if is_neg_loss:
        log_k_prob = None
        logger.info('loss function: Negative Sampling\n')
    else:
        log_k_prob = np.log(num_negatives * negative_sampler.noise_dist)
        logger.info('loss function: NCE\n')
    model = SkipGram(V=corpus.num_vocab, embedding_dim=embedding_dim)
    optimizer = optim.SGD(model.parameters(), lr=starting_lr)
    model.train()
    num_processed_words = last_check = 0
    num_words = corpus.num_words
    loss_value = 0
    num_add_loss_value = 0
    for epoch in range(epochs):
        inputs = []
        contexts = []
        for sentence in docs:
            for doc, num_processed_words in generate_words_from_doc(
                doc=sentence, num_processed_words=num_processed_words,
                corpus=corpus, rnd=rnd
            ):

```

```

):
    doclen = len(doc)
    dynamic_window_sizes = rnd.randint(low=1, high=ws + 1,
size=doclen)
    for (position, (word_id, dynamic_window_size)) in
enumerate(zip(doc, dynamic_window_sizes)):
        begin_pos = max(0, position - dynamic_window_size)
        end_pos = min(position + dynamic_window_size, doclen - 1) +
1
        for context_position in range(begin_pos, end_pos):
            if context_position == position:
                continue
            contexts.append(doc[context_position])
            inputs.append(word_id)
            if len(inputs) >= num_minibatches:
                negatives = negative_sampler.sample(k=num_negatives,
rnd=rnd, exclude_words=contexts)
                loss_value += train_on_minibatches(
                    model=model,
                    optimizer=optimizer,
                    inputs=inputs,
                    contexts=contexts,
                    negatives=negatives,
                    is_neg_loss=is_neg_loss,
                    log_k_prob=log_k_prob
                )
                num_add_loss_value += 1
                inputs.clear()
                contexts.clear()
            if len(inputs) > 0:
                negatives = negative_sampler.sample(k=num_negatives,
rnd=rnd, exclude_words=contexts)
                loss_value += train_on_minibatches(
                    model=model,
                    optimizer=optimizer,
                    inputs=inputs,
                    contexts=contexts,
                    negatives=negatives,
                    is_neg_loss=is_neg_loss,
                    log_k_prob=log_k_prob
                )
                num_add_loss_value += 1
                inputs.clear()
                contexts.clear()

        # update lr and print progress
        if num_processed_words - last_check > lr_update_rate:
            optimizer.param_groups[0]['lr'] = lr =
update_lr(starting_lr,

num_processed_words,

epochs,
num_words)

logger.info('\rprogress: {0:.7f}, lr={1:.7f}, loss=
{2:.7f}'.format(
    num_processed_words / (num_words * epochs),
    lr, loss_value / num_add_loss_value),

```



```

    )
    last_check = num_processed_words

    with open("zh.vec", 'w') as f:
        #en.vec
        f.write('{} {} \n'.format(corpus.num_vocab, embedding_dim))
        embeddings = model.in_embeddings.weight.data.numpy()
        for word_id, vec in enumerate(embeddings):
            word = corpus.dictionary.id2word[word_id]
            vec = ' '.join(list(map(str, vec)))
            f.write('{} {} \n'.format(word, vec))
if __name__ == '__main__':
    main()

```

```

D:\python\python.exe C:/Users/Vincent/Desktop/研究生/研一.上/自然语言处理/作业二: 词向量训练/pyt
Loading training corpus...
C:/Users/Vincent/Desktop/研究生/研一.上/自然语言处理/作业二: 词向量训练/pytorch_skipgram-master\
return np.array(docs)
V:3013, #words:151197
loss function: Negative Sampling
progress: 0.9996934, lr=0.0001571, loss=2.5227476

```

九、结果：词向量文件

```

3415 100 en.txt
the 0.21564382 0.035284925 0.041183706 -0.14620282 0.0064559155 0.0676749 -0.33078903 -0.3049707 -0.16561316 -0.085897654 0.27
. 0.061837245 -0.34372312 0.16705364 -0.52190405 0.20602897 0.06572465 -0.070795916 -0.087498344 -0.332679 0.31033066 0.439955
, 0.36547437 -0.083347164 -0.06673435 0.002303804 -0.0045468705 0.034068506 -0.27890727 0.084491 -0.34918955 -0.07213748 0.177
and 0.18537661 -0.23411368 0.0699319 -0.34015682 -0.06297772 0.018607156 -0.31074154 -0.050637472 -0.24128968 0.24064161 0.042
<bos> 0.54234797 0.24321789 -0.19135872 0.2746542 0.40834442 -0.10557268 -0.20992942 -0.123726666 -0.5461802 -0.34376028 0.225
<eos> 0.096013874 -0.22236064 0.29363105 -0.4422103 -0.25107554 -0.098324016 -0.37595212 0.25872388 -0.51835084 0.35575798 0.0
of 0.27275938 -0.24335167 0.14836015 -0.38446808 0.00822084 0.060084347 -0.16324922 -0.014835352 -0.46572414 0.07920102 0.0161
to 0.20388807 -0.10164469 0.17605199 -0.14334925 -0.12389737 -0.009414108 -0.20036331 0.049240954 -0.5424449 0.073200874 0.068
in 0.32006076 -0.27995902 0.120276615 -0.33252388 -0.025700964 -0.024872871 -0.25524443 -0.023323152 -0.3525632 0.18512686 0.2
" 0.19961496 0.09690796 0.28044975 -0.26272893 -0.053930365 0.19188102 -0.00444596 0.13926017 -0.47217038 0.021473395 0.058268
a 0.18451938 0.035694793 0.05047792 -0.123658314 -0.071958035 -0.0017765879 -0.1771089 -0.07886951 -0.043000147 -0.11843151 0.
is 0.3193777 0.028576596 0.26031196 -0.21164724 -0.15055352 -0.054572582 -0.24485053 0.11131566 -0.3936795 0.08885897 0.252726
that 0.4148451 0.01649326 -0.18198891 0.24950328 0.19680041 0.006631575 -0.2920904 -0.11971344 -0.41183582 -0.22391964 -0.1265
's 0.30971456 0.12987974 -0.083781235 -0.3820373 0.107208535 0.10914942 -0.2809488 -0.32475415 -0.42469764 0.23158927 0.409954
on 0.24877338 -0.0044224765 0.029989775 -0.2187489 0.03697774 0.058916554 -0.17124526 -0.17259945 -0.55596596 0.14345558 0.019
china 0.20778075 -0.11450185 -0.16136134 -0.14504845 -0.18373601 0.09854919 -0.3481709 0.11380488 -0.07327756 -0.014395433 -0.
for 0.2983933 -0.12999177 0.06933801 -0.34138763 -0.07532256 0.07378054 -0.18844858 -0.06232917 -0.4199098 0.1334416 0.0787503
has 0.5301892 0.24677365 -0.18066749 -0.1243784 -0.10642707 0.07943219 -0.5692722 -0.063128084 -0.3374401 0.35224986 -0.090295
with 0.28156123 -0.18805985 -0.00043911053 -0.25689727 0.1067133 0.001757772 -0.3932009 0.023162335 -0.4922003 0.19457823 0.14
this 0.33175176 -0.025102 0.14207202 -0.18668124 -0.14288041 -0.0842273 -0.075079486 0.01911247 -0.06683796 -0.1824764 0.26811
have 0.34185648 0.14181924 0.18221359 0.09875365 0.1368715 -0.033808302 -0.70590854 0.10235255 -0.41408935 0.29862595 -0.09135
will 0.3647775 -0.028290797 0.3968502 0.016417922 0.05494762 -0.051986616 -0.5102149 0.22876495 -0.40801445 0.13710952 0.30547
by 0.33928272 0.03329492 0.053018954 -0.11388378 0.029613523 0.05792666 -0.23117617 -0.12178087 -0.4447342 0.07739939 -0.06872
be 0.07035755 0.09791447 0.403294 0.06805238 0.08913347 -0.272834 -0.17027192 0.040760327 -0.42014188 -0.12241269 0.16243921 -
as 0.51374096 -0.13919394 0.13211957 0.108164184 -0.03808893 0.3391241 -0.360637 0.04973075 -0.39943337 0.014004858 0.04385063
are 0.26933753 0.056878276 0.23996042 0.16872504 0.14275852 -0.011378094 -0.49511296 0.1447302 -0.4819269 0.19662374 0.0239458
it 0.37735626 0.0031773627 0.2525066 -0.026985863 -0.33280507 -0.08559181 -0.123076685 0.14951791 0.061969295 -0.1867883 -0.00
people 0.32141238 -0.08276457 0.3619361 0.0028746496 0.07935139 -0.09382496 -0.25454706 0.28592518 0.028944504 -0.103864886 0.
taiwan 0.1464728 0.2872107 0.4943322 -0.28825328 0.0061819614 -0.00093184615 -0.26787147 0.10106324 -0.15397894 -0.05758993 -0
he 0.5197373 0.15289606 -0.12085856 0.29664 0.20383674 -0.20982331 -0.23444903 -0.0918253 -0.19432181 -0.4015905 0.041972112 -
development -0.009425807 -0.24528417 0.37630934 -0.33884925 -0.12508431 0.1320474 -0.24142437 0.22837739 -0.23162784 0.1092630
at 0.61427915 -0.09844158 0.08931023 -0.043148793 0.2272146 -0.14060277 -0.11137447 -0.11819305 -0.40953806 -0.11240568 0.2607
we 0.22359285 -0.0019877695 0.45066884 0.15669616 0.20397183 -0.4321785 -0.17664313 0.1089418 -0.15962729 -0.26393482 0.299120
not 0.29095638 0.16712976 0.44729465 0.03634201 -0.16791049 -0.011905071 -0.223408 0.17795216 -0.13279253 -0.04170798 -0.02266
: 0.4566082 0.056755535 -0.3076288 0.31955045 0.4534471 -0.028755493 -0.26141012 -0.22588414 -0.5672124 -0.35292482 -0.0465997
said 0.47027814 0.2166875 -0.122896604 0.13551903 0.5399753 -0.30503798 -0.24623019 0.061300103 -0.6361981 0.0098488815 0.4393

```

```
3013 100 Zh.txt
-0.063939065 -0.11157483 0.22449571 0.014153564 0.3889095 0.1808173 0.28596747 -0.18136206 0.1573282 -0.2217429 0.077
的 -0.24166724 -0.14851102 0.055817746 0.022280999 0.18301922 0.04597234 0.21612369 -0.3511076 -0.032506377 -0.09515692
<bos> 0.3037343 -0.25671867 0.11522135 0.075080365 0.0022876174 0.055194307 0.057510406 0.15240476 0.046737842 -0.14182
<eos> -0.47555372 -0.080668144 -0.047916546 0.10626558 0.48452497 0.016675388 0.13730477 -0.53229785 -0.022235718 -0.18
。 -0.53917676 0.072344735 0.1165143 0.022387428 0.43283412 -0.077131435 0.17624259 -0.53636754 -0.23559767 -0.04699373
是 -0.23703912 0.20525385 -0.10933248 -0.008668406 0.13222478 0.18846498 0.19309759 -0.06551057 -0.14185677 -0.18370312
和 -0.20767784 -0.19711863 0.16736957 0.012437382 0.08142412 0.09045763 0.29873005 -0.34506363 0.20521946 0.086689815 -
在 0.09320845 -0.16413692 0.14782165 0.004942621 0.07960903 0.2630157 0.04930717 0.0041248947 0.07709329 -0.22596002 -0
、 -0.19418256 -0.20759289 0.054487124 0.21397418 -0.026897075 -0.16780353 0.2674208 -0.2075735 0.12250108 0.27777785 -
了 -0.10167742 -0.11132382 0.09630349 0.02760773 0.2992239 0.12940888 0.002624267 -0.18888265 -0.18377763 -0.15378621 -
中国 -0.021994805 -0.021784691 0.16521007 0.1480053 0.15981492 0.12364901 0.06382677 -0.15423763 0.09270755 -0.24565591
一 -0.051938843 -0.1364368 -0.065733336 -0.13263363 0.003887372 0.13033222 0.06825052 -0.23749821 -0.03766682 -0.496716
" -0.0638178 -0.027031364 -0.22805822 0.25583586 0.018069558 0.004140055 0.043918937 -0.11928288 0.19953604 -0.21408477
" -0.49120128 -0.21062525 -0.2775807 0.12953673 -0.025184236 0.087013885 0.16213688 -0.30081967 -0.16003394 -0.28714615
对 0.077935904 -0.11497622 0.1608892 0.004217013 0.07580351 0.18487416 0.09209618 -0.011486481 0.108346455 -0.07323535
发展 -0.48558778 -0.085066505 0.360255 -0.060536858 0.30799693 0.31927788 0.33864227 -0.3294574 0.172528 0.027529925 -0
不 -0.24588144 0.40592468 -0.31915227 -0.012045806 0.20206913 0.11697612 -0.078573406 -0.0964478 -0.0756415 -0.3935221
中 -0.24378666 -0.43957698 0.40712202 0.077040076 0.4224331 0.26234955 0.21389633 -0.01515012 0.14110717 -0.06233566 -0
-0.2760177 -0.46928805 0.23148192 0.15174955 0.08710497 0.48293725 0.37503338 -0.019651981 0.4195253 -0.27570558 -0
两 0.26601675 -0.5337623 0.53237236 0.050774213 0.2656879 0.34802207 0.45381755 0.2077637 0.28659877 -0.02042705 -0.34
国 -0.022894593 -0.06334412 -0.19317275 -0.023910336 -0.15969776 0.45245016 0.32217222 0.2485943 -0.006715711 -0.051917
个 0.16116773 -0.04516994 -0.11618432 -0.0050093425 0.017797641 0.25386438 -0.04943551 0.08655507 0.26175806 -0.4618067
这 -0.10806459 0.008363062 -0.13694292 0.02598664 0.19446893 0.15117104 -0.047298353 -0.1345778 0.01334293 -0.27763733
有 -0.060332786 0.037068106 -0.024594033 0.098507546 0.23286787 0.05179506 0.039446868 -0.18160568 0.16174439 -0.187517
要 -0.14888194 -0.07367867 0.17564167 -0.036959868 0.28360543 0.2621461 0.21305755 -0.10477957 -0.03226476 -0.03048479
美国 0.112492524 0.03937188 0.030308513 0.07902801 0.10487839 0.117626555 -0.11971158 -0.07339798 0.09636201 -0.2813899
经济 -0.1753035 -0.0639842 0.23494324 -0.10314871 0.13547045 0.27946636 0.4134532 -0.44148174 0.31495222 0.058126263 -0
说 0.20686187 -0.15377885 -0.022834692 0.09477123 0.032980483 0.28772122 -0.18488295 0.12609695 0.097898714 -0.33390287
他 0.22670732 -0.17193058 0.08845341 0.16382287 0.26424137 0.33870906 -0.17143856 0.038836952 0.33661687 -0.5401711 0.2
也 -0.101876 0.23935866 -0.08757456 -0.004002464 0.27112246 0.14310394 -0.03175249 -0.16300926 0.019464837 -0.31438762
上 -0.076448634 0.21540694 -0.15559702 0.009454646 0.06304157 0.24425223 -0.04436071 -0.20266752 -0.09609127 -0.4388814
关系 -0.5254931 -0.35320154 0.40804487 -0.07933517 0.44895965 0.60237384 0.39481157 -0.12254101 0.13384424 -0.18513644
问题 -0.28693175 0.32591397 -0.16461433 0.036037102 0.26284087 0.3909239 -0.17298865 -0.106815204 -0.27738762 -0.537808
国家 -0.18943289 -0.19688399 0.18086948 0.085169755 0.0503825 0.109550446 0.15729316 -0.3086892 0.08661172 -0.00906753
将 0.031102045 -0.13758092 0.15067583 0.026567455 0.18024778 0.12843278 0.07388011 -0.07675538 0.092457816 -0.11403007
```

十、思考

1、是否需要去除停用词

训练词向量不需要去除停用词。有两个原因，一个是训练的词语字典越多，在之后训练神经网络模型做 **embedding** 词语映射时可以减少 **oov** 的情况，二则是根据具体的 **nlp** 任务，比如有的情感识别任务，停用词的作用是蛮大的。训练词向量，当然是越多语料越好，一般而言是把整个数据集都扔进去训练。

2、如何加速原始softmax层

hierarchical softmax 实质上生成一颗带权路径最小的哈夫曼树，计算量减小，让高频词搜索路劲变小，问题：每个节点用 **sigmoid** 函数，其实应该将语义相近的词汇放在一起，做一个聚类。
negative sampling 变成 **sigmoid**，对每一个样本中每一个词都进行负采样，一个样本：（中心词，正样本，负样本集合）**sigmoid** 函数，这样速度会很快，**NCE loss**。

3、skip_gram模型训练出来的词向量不能处理一词多义的问题。

4、激活函数sigmoid和tanh的区别

表面上看起来，两者的区别只是值域不同，**sigmoid** 是 (0,1)，**tanh** 是 (-1,1)。可就是这个小小的不同，造成了他们的效果其实有很大差别。**sigmoid** 函数 作为激活函数，**sigmoid** 函数存在梯度弥散的问题，即当 **x** 的取值特别大或者特别小的时候，其梯度几乎为零，这会导致参数的值无法更新。即所说的梯度消失。作为激活函数，**sigmoid** 的不是中心对称的，且总是输出正数，这会导致在做非线性变换时，其只能“同意”上一层的结果，这也会导致随着深度的增加，结点的取值会“爆炸”，这也是在 **sigmoid** 函数中引入 **batch normalization** 变换的原因，因为随着层数增加，样本的分布会从 **0-1** 高斯分布逐渐偏移，偏移至 **sigmoid** 的饱和区域，导致反向传播很难进行，所以其收敛较慢，而 **batch-normalization** 会把样本分布强行拉回到 **0-1** 高斯分布。输出总是正数也会导致其优化路径出现“**zigzag**”现象，即所有权值的优化方向总是相同的（同时增大或减小）。**tanh** 函数 作为激活函数，其同样存在梯度弥散问题。**tanh** 函数相较于 **sigmoid** 函数的优点在于其是中心对称的，均值为 **0** 的分布，其能将一个 **0-1** 高斯分布依然映射到 **0** 附近的分布，保持零均值的特性，而且由于其有正有负，所以他可以对来自上一层的结果“支持”（正）、“反对”（负）、“弃权”（**0**）。所以其收敛速度较 **sigmoid** 快一些。

