

研究生《自然语言处理》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **词向量** |
| **姓 名：** | **王昭** |
| **学 号：** | **24140617** |
| **日 期：** | **2024.10.06** |

## 一、实验目的

学会使用 word2vec 词向量算法，对给定的语料进行分词和词向量表示。

## 二、实验环境

硬件环境：

* 笔记本：CPU：R7-8845H、内存：32G
* Google Colab Jupyter NoteBook：GPU：NVIDIA L4 24GB、内存：53G

TPU：TPU V2-8 RAM：224.6G

软件环境：

* 操作系统：Windows 11
* Python版本：3.9
* Python环境管理器：Anaconda
* 依赖库：TensorFlow 2.17、Gensim 4.3.3、jieba 0.42.1、OpenCC 1.1.9、Matplotlib 3.9.2

## 三、实验原理

### 3.1 模型原理

Word2Vec是一种把语言文字转换为向量形式表达的模型，该模型利用大规模语料库中词语的上下文信息生成词向量，并且这些词向量可以捕获词语之间的语义相似性。

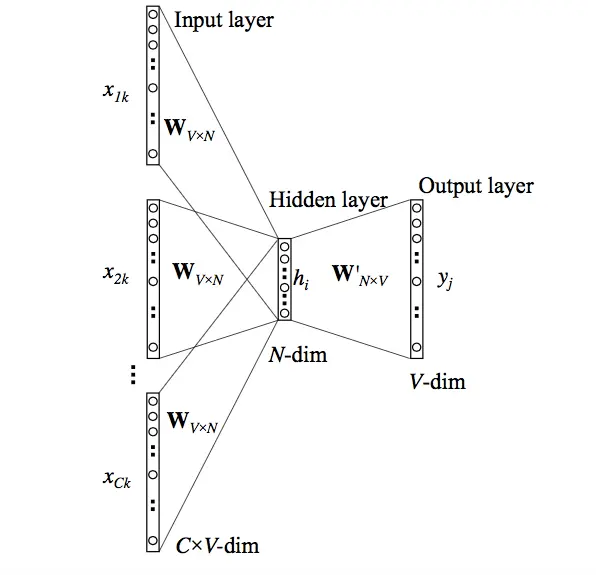
Word2Vec分为两种模型：

1. CBOW（Continuous Bag of Words）模型

该模型通过上下文来预测目标词（中心词）。CBOW 模型对小型数据比较合适。

CBOW模型训练基本步骤如下：

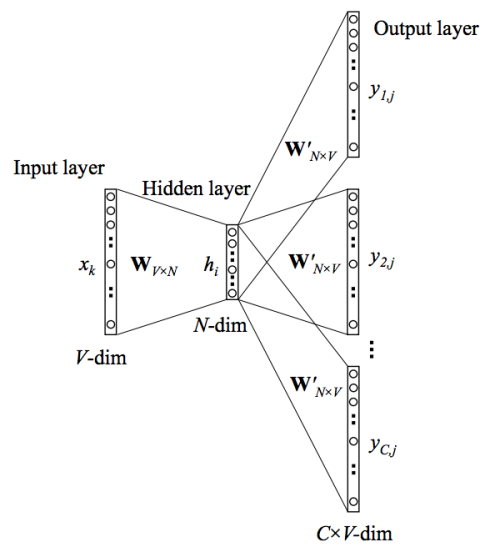
1. 将上下文词汇使用 one-hot 编码表示为模型的输入，词汇表的维度为 V，上下文单词数量为C；
2. 将所有上下文词的 one-hot 向量与输入层到隐层的权重矩阵 W 相乘，得到各个词对应的隐层表示；
3. 将这些隐层向量相加并取平均值，生成一个最终的隐藏层表示；
4. 将该隐藏层向量与隐层到输出层的权重矩阵 W' 相乘，得到一个隐藏层输出向量；
5. 对输出向量进行 softmax 激活，生成一个 V 维的概率分布，选择概率最大的词对应的索引作为预测的目标词。



1. Skip-Gram模型

该模型通过给定目标词（中心词），预测上下文词语。Skip-Gram在大型预料中表现更好。

Skip-Gram模型的训练步骤与CBOW模型几乎一致，只是输入的数据为目标词的one-hot向量。



### 3.2 实验步骤

#### 3.2.1 语料库提取与预处理

首先将中文维基百科的语料库下载到本地，然后使用Gensim库读取该语料并转换为txt格式。然后读取处理后的txt文本，对数据进行预处理，具体如下：① 将文本中的繁体中文使用OpenCC库转换为简体中文；② 将文本里的非简体中文字符、空格符和换行符进行剔除，保留较为“干净”的中文文本，再将其保存为txt文件。

#### 3.2.2 语料分词与停止词过滤

使用jieba分词库对预处理后的语料进行分词，此外对多余的空格以及停止词进行过滤，保留更有是实际意义的词汇。

#### 3.2.3 CBOW模型与Skip-Gram模型的训练

1. CBOW模型：

首先设定好模型的相关参数如batch\_size、embedding\_size、half\_window\_size等，然后读取先前保存的分词数据即完整的词表。

然后对词表进行处理，词表中共有3103064个不同的词，模型只处理完整词表的前80W个高频词，其余的词均作为UNK（unknown）处理。对这些高频词建立索引，生成词-索引表和索引-词表，以便快速进行查找。

之后进行训练集的生成，根据指定的上下文窗口生成batch（上下文）和label（目标词）集合。

接着进行模型的建立，明确模型的输入：模型的输入是目标词的所有上下文词的one-hot向量表示其中为滑动窗口的大小。

隐藏层的输入是输入层的每个向量与输入层与隐藏层之间的权重相乘再加上一个偏置项，得到每个向量的输出结果，然后再相加取平均得到向量。其中权重是随机初始化的，偏置项初始化为0向量。

输出层的输入是与隐藏层和输出层之间的权重相乘，再加上一个偏置，然后把结果经过Softmax之后得到模型的输出，该输出是一个概率分布，表示对于给的上下文，模型预测出的目标词的概率。

1. Skip-Gram模型：

Skip-Gram模型与CBOW模型非常相似，只是他们的输入和输出是反过来的。

对于输入层，模型的输入是每个目标词的one-hot向量表示。

隐藏层的输入是与输入层与隐藏层之间的权重相乘再加上一个偏置项就得到了隐藏层的最终向量表示。

输出层的输入是与个隐藏层与输出层之间的权重相乘，得到个，即目标词的上下文。

#### 3.2.4 模型训练效果检验

按照以上原理进行模型的训练，当模型训练完毕之后，将其保存下来，然后选择十个词，输出其的向量表示，并且使用余弦相似度计算与其最相近的词语。

再挑选不同类型的词如水果、任务、动物等，对他们的词向量进行二位可视化，观察学习到的词向量的好坏。

探索类比实验，计算V(王子) - V(男) + V(女)最近的词向量是不是V(公主)。

## 实验内容

### 4.1 bz2文件转txt文件

为了便于使用，将bz2文件转为txt文件的python代码被封装main()函数中，这样便可以直接在命令行进行调用。

wiki\_xml\_to\_txt.py文件：

1. import sys
2. import logging  *# 打印日志包*
3. from gensim.corpora import WikiCorpus  *# 导入gensim库中用于处理维基百科语料库的WikiCorpus类*
4. def main():
5. """
6. 该函数主要在命令行中运行，需要读取命令行参数，接收的参数为：执行的py文件名 原始的bz2语料
7. @return: Nones
8. """
9. *# 如果命令参数有误，给予提示*
10. if len(sys.argv) != 2:
11. print("命令输入有误！命令格式: python 执行的python文件名 wiki预料库路径")
12. exit()
13. *# 设置日志格式*
14. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
15. *# 读取只读、流式、内存高效的wiki语料*
16. wiki\_corpus = WikiCorpus(sys.argv[1], dictionary={})
17. *# 语料库统计*
18. text\_num = 0
19. *# 创建获取完毕的语料库文件 wiki\_text.txt, 读取数据并保存到该文件中*
20. file\_path = 'data/txt/wiki\_text.txt'
21. logging.info('==>开始处理文件...')
22. with open(file\_path, 'w', encoding='utf-8') as out\_put:
23. *# WikiCorpus.get\_texts() 方法遍历语料库获取 token(模型输入单元) 列表*
24. for text in wiki\_corpus.get\_texts():
25. *# 将分词后的文本写入到文件中，每行一个文本*
26. out\_put.write(' '.join(text) + '\n')
27. text\_num += 1
28. if text\_num % 10000 == 0:
29. logging.info('已处理 ' + str(text\_num) + ' 文本')
30. logging.info('处理的文本总数为：' + str(text\_num))
31. if \_\_name\_\_ == '\_\_main\_\_':
32. main()

在命令行执行该脚本时，需要确保安装有Python（推荐使用9.3版本），并且已经安装了相应的库。推荐使用Anaconda对Python环境进行管理，可以使用

1. conda activate env\_name

命令切换到相应的环境，其中env\_name指本地的Anaconda管理的Python环境名称。再使用

1. python wiki\_xml\_to\_txt.py bz2\_file\_path

命令进行语料库的读取和写入到txt文件中，其中bz2\_file\_path本地的bz2文件的路径。最终显示处理的文本总数如下：

1. 2024-10-07 14:01:41,858 : INFO : finished iterating over Wikipedia corpus of 484683 documents with 112669081 positions (total 4535261 articles, 132840473 positions before pruning articles shorter than 50 words)
2. 2024-10-07 14:01:41,928 : INFO : 处理的文本总数为：484683

处理完成后的文件保存在项目的data/txt/wiki\_text.txt。

### 4.2 txt语料库预处理

转换后的txt语料里有很多繁体中文、英文以及其他字符，因此还需要进一步的预处理操作。具体的操作内容如下：

1. 先将繁体中文转换为简体中文。
2. 去除杂乱的引用内容。
3. 将过滤后的简体中文文本除了简体中文字符、空格、换行符以外的所有字符，需要注意去掉了多个连续的空格只保留了一个空格。

text\_pre\_progress.py文件：

1. import os
2. import re
3. import logging
4. import opencc
5. def filter\_quote(text):
6. """
7. 过滤掉正文中的 注释、参考、参考资料、参考文献、参考书目、脚注、扩展阅读、参见、延伸阅读、概述、研究书目、引用
8. :param text: 输入文本
9. :return: 过滤后的文本
10. """
11. *# 定义要过滤的关键词*
12. keywords = [
13. "注释", "参考", "参考资料", "参考文献", "参考书目",
14. "脚注", "扩展阅读", "参见", "延伸阅读", "概述",
15. "研究书目", "引用"
16. ]
17. *# 创建正则表达式模式*
18. pattern = r'(' + '|'.join(keywords) + r').\*'
19. *# 去除包含关键词及其后的内容*
20. cleaned\_text = re.sub(pattern, '', text, flags=re.MULTILINE)
21. return cleaned\_text.strip()
22. def filter\_text(text):
23. """
24. 过滤掉非中文字符及多个连续的空格，保留中文字符、单个空格、换行符
25. :param text: 输入文本
26. :return: 过滤后的文本
27. """
28. *# 保留：简体中文、空格、换行符，过滤掉其他内容*
29. pattern = r'[^\u4e00-\u9fff\s\n]'
30. *# 使用正则表达式将匹配到的非指定字符替换为空字符*
31. filtered\_text = re.sub(pattern, '', text)
32. *# 使用正则表达式替换多个空格为单个空格*
33. filtered\_text = re.sub(r'\s+', ' ', filtered\_text)
34. return filtered\_text
35. def main():
36. *# 设置日志格式*
37. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
38. text\_file = os.path.join(os.getcwd(), 'data', 'txt', 'wiki\_text.txt')
39. pre\_pro\_wiki\_text\_file = os.path.join(os.getcwd(), 'data', 'txt', 'pre\_simp\_wiki\_text.txt')
40. out\_put = open(pre\_pro\_wiki\_text\_file, 'w', encoding='utf-8')
41. with open(text\_file, 'r', encoding='utf-8') as wiki\_text:
42. *# 遍历 wiki\_text 里的每行数据，分别对其进行处理：*
43. *# 1. 先将繁体中文转换为简体中文*
44. *# 2. 去除杂乱的引用内容*
45. *# 3. 将过滤后的简体中文文本除了简体中文字符、空格、换行符以外的所有字符，*
46. *#    需要注意去掉了多个连续的空格只保留了一个空格*
47. count = 0
48. logging.info('==>开始处理文本...')
49. for line in wiki\_text:
50. *# 先把繁体中文转换为简体中文，创建 OpenCC 对象，指定繁体转换为简体*
51. converter = opencc.OpenCC('t2s')  *# 't2s' 表示繁体转简体*
52. *# 将文本中的简体中文转换为简体中文*
53. simplified\_text = converter.convert(line)
54. *# 过滤引用内容*
55. *# 过滤非中文字符，只保留简体中文字符、单个空格符以及换行符*
56. simplified\_text = filter\_text(filter\_quote(simplified\_text))
57. *# 把处理后的文本写入到 out\_put 中*
58. out\_put.write(simplified\_text + '\n')
59. count += 1
60. if count % 500 == 0:
61. logging.info('==>已经处理了 ' + str(count) + ' 数据...')
62. logging.info('==>处理完成，共计处理 ' + str(count) + ' 条数据。')
63. if \_\_name\_\_ == '\_\_main\_\_':
64. main()

处理完成后的文件保存在项目的data/txt/pre\_simp\_wiki\_text.txt。

### 4.3 对语料库进行分词

实验数据预处理完成后，便可以进行语料库分词操作。本次实验使用jieba分词工具进行分词，分词时对分词结果进行判断，如果分词的结果是停止词，那么就跳过该内容。

word\_segment.py文件：

1. import logging
2. import os
3. import jieba
4. *# 加载停用词词表，使用的是哈工大停用词词表*
5. def load\_stop\_words(path):
6. """
7. 加载停用词词表，使用停用词可以提高效率
8. @ stop\_words\_path: 停用词路径
9. @return: 停用词词表的列表
10. """
11. with open(path, 'r', encoding='utf-8') as stop\_words\_file:
12. logging.info("加载停用词词表...")
13. return stop\_words\_file.read().split('\n')
14. def main():
15. *# 设置日志格式*
16. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
17. *# 设置输出缓冲区*
18. wiki\_seg\_file = os.path.join('data', 'txt', 'wiki\_seg.txt')
19. out\_put = open(wiki\_seg\_file, 'w', encoding='utf-8')
20. wiki\_text\_file = os.path.join('data', 'txt', 'pre\_simp\_wiki\_text.txt')
21. stop\_words\_path = os.path.join('data', 'stopwords', 'hit\_stopwords.txt')
22. stop\_words = load\_stop\_words(stop\_words\_path)
23. logging.info("开始分词...")
24. with open(wiki\_text\_file, 'r', encoding='utf-8') as wiki\_text:
25. for count, line in enumerate(wiki\_text):
26. *# 启动paddle模式, 延迟加载。 0.40版之后开始支持*
27. *# jieba.enable\_paddle()*
28. seg\_list = []
29. *# 开始分词，使用的是精确模式*
30. seg\_list = jieba.cut(line, cut\_all=False)
31. *# 去掉空格和空字符串*
32. strip\_seg\_list = [word for word in seg\_list if word.strip()]
33. *# 去掉停用词*
34. strip\_seg\_list = [word for word in strip\_seg\_list if word not in stop\_words]
35. *# 把生成的数据保存到 out\_put 中*
36. for word in strip\_seg\_list:
37. out\_put.write(word + ' ')
38. out\_put.write('\n')
39. if count % 100 == 0 and count > 0:
40. logging.info("已经处理的分词数据：" + str(count))
41. logging.info("分词结束, 处理的数据总量为：" + str(count))
42. if \_\_name\_\_ == '\_\_main\_\_':
43. main()

处理完毕后，分词后的文件保存在data\txt\wiki\_sge.txt。

### 4.4 CBOW模型编写与训练

以下内容均为一个py文件即CBOW\_TF.py，为了便于讲解，将代码分开提取，可根据代码行判断代码位置。

首先是相关参数的设置，如分词后的txt文件路径、每次迭代使用的样本个数batch\_size、词嵌入的维度、滑动窗口的大小、模型训练时验证集的构造等。

1. import collections
2. import math
3. import os
4. import logging
5. import random
6. import numpy as np
7. import tensorflow.compat.v1 as tf
8. from importlib import reload
9. from sklearn.manifold import TSNE
10. import pickle
11. """----------参数----------"""
12. colab\_cwd = '/content'
13. if os.getcwd() == colab\_cwd:
14. *# 处理好的文本存储在谷歌云盘*
15. file\_path = os.path.join(os.getcwd(), "drive", "MyDrive", "data", "txt", "wiki\_seg.txt")
16. else:
17. *# 本地的文件路径*
18. file\_path = os.path.join(os.getcwd(), "data", "txt", "wiki\_seg.txt")
19. *# 每次迭代将使用 256 个样本进行参数更新*
20. batch\_size = 256
21. *# 单词转为稠密向量的维度*
22. embedding\_size = 350
23. *# 滑动窗口的大小*
24. half\_window\_size = 4
25. *# 只从前1w个高频词里找*
26. valid\_window = 10000
27. *# 用来抽取的验证单词数*
28. valid\_size = 10
29. *# 需要验证的单词*
30. valid\_has\_word = ['父亲', '中国', '电脑', '手机', '书籍', '公里', '还', '部分', '年', '之后']
31. *# valid\_examples = np.array(random.sample(range(valid\_window), valid\_size))*
32. *# valid\_examples = np.random.choice(valid\_window, valid\_size, replace=False)*
33. *# 负采样的噪声单词的数量*
34. num\_sampled = 5

然后是logging输出的配置：

1. """----------logging配置----------"""
2. *# 使用 logging.info 打印信息，colab 需要 reload() 函数，否则无法打印*
3. reload(logging)
4. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

接下来是数据的读取：读取分词后的文件，将所有的词存入一个列表中，此外，为了便于查看读取的信息，设置了一定的logging信息。

1. """----------数据处理----------"""
2. def read\_data(filename):
3. """
4. 从本地文件中读取数据，
5. @param filename:
6. @return:
7. """
8. logging.info("开始读取文件：" + filename)
9. with open(filename, "r", encoding="utf-8", errors='ignore') as f:
10. *# 一次性读取所有的行，返回一个列表*
11. seg\_content = f.readlines()
12. seg\_words = []
13. *# 将单词都存放到seg\_words列表中，不去重，方便下一步建立以词频为基础的词表*
14. for i in range(len(seg\_content)):
15. *# 每 5000 行打印一次信息*
16. if i % 5000 == 0:
17. logging.info("当前读取到第 " + str(i) + " 行，部分数据为: " + str(seg\_content[i][:50]))
18. *# 获取当前行词的列表，使用strip()函数过滤空格包括换行符或者制表符*
19. seg\_words.extend(seg\_content[i].strip().split(" "))
20. return seg\_words
21. all\_words\_list = read\_data(file\_path)

然后是建立词表数据集，具体来说是高频词词表、转换后的索引词表、单词-索引表和索引-单词表，这样可以加快查询的速度。使用collections.Counter()可以统计各个词的词频，得到{单词：次数}格式的字典\_count，most\_common(size)函数可以返回出现频率最高的size个元素及其计数。然后根据按照词频的顺序，生成{单词：索引}的字典\_dictionary。然后根据生成的字典，将原来大的词表转换成索引表，对于不在高频词典中的词，当作UNK（unknown）处理。最后再构建一个{索引：单词}的反转字典。最后根据得到的索引，得到验证单词的索引列表。

创建词表之后使用logging.info()打印一下结果，然后根据得到的词表，完善指定的验证集的索引。

1. def build\_dataset(words, size):
2. """
3. 创建词表，包括原始词表、转换后的索引词表、单词-索引表和索引-单词表
4. @param words: 原始词表
5. @param size: 高频词词表长度
6. @return: 索引词表、原始词表、单词-索引词表、索引-单词词表
7. """
8. *# 初始化 count 列表，其中 UNK 代表 unknown, 即高频词表以外的词*
9. \_count = [['UNK', -1]]
10. *# 统计词频，只取前 vocabulary\_size 个高频词, 格式为：('词', 词频)*
11. \_word\_collection = collections.Counter(words)
12. logging.info('所有词的数量为：' + str(len(\_word\_collection)))
13. *# 构建高频词表*
14. \_count.extend(\_word\_collection.most\_common(size - 1))
15. logging.info("高频词表前 20 个数据为数据为: " + str(\_count[:20]))
16. *# 构建字典，将词转化为索引, 词典顺序为高频词顺序, 格式为('词', 索引), 其中索引从 0 开始*
17. \_dictionary = dict()
18. for \_word, \_ in \_count:
19. \_dictionary[\_word] = len(\_dictionary)
20. *# 将词转化为索引存储到\_data中, 词在高频词词典的话，索引为高频词词典的索引, 如果词不在字典中, 则转化为 UNK*
21. *# 此时 \_data 就是原来词的列表的索引列表*
22. \_data = list()
23. unk\_count = 0
24. for \_word in words:
25. if \_word in \_dictionary:
26. index = \_dictionary[\_word]
27. else:
28. index = 0
29. unk\_count += 1
30. \_data.append(index)
31. \_count[0][1] = unk\_count
32. *# 构建反向字典, 可以快速从词索引转化为词即 (索引-'词')*
33. \_reverse\_dictionary = dict(zip(\_dictionary.values(), \_dictionary.keys()))
34. return \_data, \_count, \_dictionary, \_reverse\_dictionary
35. *# 高频词词表大小, 对于 4w 的切片， 10w 词表的结果较好*
36. vocabulary\_size = 800000
37. *# vocabulary\_size = 50000*
38. *# 构建词表*
39. data, count, dictionary, reverse\_dictionary = build\_dataset(all\_words\_list, vocabulary\_size)
40. logging.info('所有词的数量为：' + str(len(count)))
41. logging.info('含 UNK 的前 20 个高频词' + str(count[: 20]))
42. logging.info('含 UNK 的倒数 20 个高频词' + str(count[-20:]))
43. logging.info('查看索引与词的映射：')
44. logging\_str = ''.join(
45. [f"{idx} : {word}, " for idx, word in zip(data[:10], [reverse\_dictionary[i] for i in data[:10]])])
46. logging.info(logging\_str)
47. *# 完善验证单词的索引*
48. valid\_examples = []
49. *# 如果待验证词在此表中，那么直接获得索引，否则指定为 unknown*
50. for word in valid\_has\_word:
51. if dictionary.get(word, None) is not None:
52. valid\_examples.append(dictionary[word])
53. else:
54. valid\_examples.append(0)
55. *# 删除原始词表，节省内存*
56. del all\_words\_list

接着是生成训练所使用的batch的函数generate\_batch()：根据给定的batch\_size，half\_window\_size生成batch（batch\_size × 2 \* half\_window\_size维）和label（batch\_size×1维）矩阵，构建的时候需要注意忽略目标词。

1. data\_index = 0
2. def generate\_batch(\_batch\_size, \_half\_window\_size):
3. """
4. 生成训练用的 batch 数据以及标签数据，其中 batch 是 上下文， label 是 目标词
5. @param \_batch\_size: batch 的大小
6. @param \_half\_window\_size: 目标词的上下文窗口大小（只有一半）
7. @return: 生成的 batch 数据 和 label 数据
8. """
9. global data\_index
10. *# 初始化二维数组, 行数为 \_batch\_size，列数为 2 \* half\_window\_size, 数据类型为 int32*
11. \_batch = np.ndarray(shape=(\_batch\_size, 2 \* \_half\_window\_size), dtype=np.int32)
12. \_labels = np.ndarray(shape=(\_batch\_size, 1), dtype=np.int32)
13. len\_data = len(data)
14. *# 生成每个样本*
15. for i in range(\_batch\_size):
16. *# 当前样本的索引*
17. index = data\_index
18. *# 目标词作为标签*
19. \_labels[i] = data[(index + \_half\_window\_size) % len\_data]
20. *# 获取上下文词*
21. for k in range(2 \* \_half\_window\_size + 1):
22. *# 当 k 不是忽略词*
23. if k != \_half\_window\_size:
24. *# 调整索引以适应数组存储*
25. t = (k if k < \_half\_window\_size else k - 1)
26. \_batch[i, t] = data[(index + k) % len\_data]
27. *# 更新数据索引*
28. data\_index = (data\_index + 1) % len\_data
29. return \_batch, \_labels

接着是神经网络的构建：

第一步是创建一个计算图graph，然后为模型的输入层：train\_dataset与train\_labels创建占位符，以便在后续传入数据。valid\_dataset存放用于后面验证的数据。

接着是嵌入层的设置：包括词嵌入矩阵embeddings、初步嵌入后的embed向量以及求取的均值向量avg\_embed。

最后是输出层的配置：softmax层的权重矩阵softmax\_weights与偏置项softmax\_biases，此外还有损失函数loss，用来计算softmax的损失。

因为本次实验参数较大，因此选择的损失函数为sampled\_softmax\_loss()，该方法为了减少计算量，会对负样本（非目标类作为负样本）进行采样，仅计算采样到的那些类别的概率。每次使用的负样本的数量是5，多分类的类别为整个高频词表的大小。

优化器选择的是AdaGradOptimizer()，该优化器学习率会自动调整，并且对于稀疏数据和稀疏特征有着很好的适应性。通过对每个参数单独维护一个学习率，从而使整个模型能够找到合适的权重。

模型训练的同时，计算指定词最相似的单词，用于后面的训练验证。

1. """----------训练 & 验证----------"""
2. *# 创建默认的 graph*
3. graph = tf.Graph()
4. with graph.as\_default():
5. '''-----输入-----'''
6. *# 创建一个 TensorFlow 占位符的，类型为 32 位整数，大小为 batch\_size 的一维张量*
7. train\_dataset = tf.placeholder(tf.int32, shape=(batch\_size, 2 \* half\_window\_size))
8. *# 创建一个 TensorFlow 占位符的，类型为 32 位整数，大小为 batch\_size x 1 的二维张量*
9. train\_labels = tf.placeholder(tf.int32, shape=(batch\_size, 1))
10. *# 创建常量张量, 值为 valid\_examples 列表中的元素, 制定了数据类型, 用于后续作为固定输入*
11. valid\_dataset = tf.constant(valid\_examples, dtype=tf.int32)
12. '''-----变量-----'''
13. *# 词嵌入矩阵, 形状为 [vocabulary\_size, embedding\_size], 通过训练，嵌入矩阵会学习到每个词的分布式表示（即词向量）*
14. *# tf.random.uniform() 用于从均匀分布中随机初始化嵌入向量，范围是 [-1.0, 1.0]。给每个词的嵌入向量的初始值是在 -1 到 1 之间的随机数。*
15. embeddings = tf.Variable(tf.random.uniform([vocabulary\_size, embedding\_size], -1.0, 1.0))
16. *# softmax 层的权重矩阵，形状为 [vocabulary\_size, embedding\_size], 用于在通过模型训练时，将词嵌入向量映射到词汇表中每个词的概率分布*
17. *# tf.random.truncated\_normal() 用于从截断的正态分布中随机初始化权重。*
18. *# 标准差 stddev=1.0 / math.sqrt(embedding\_size) 是一种常见的初始化方式，它有助于在训练开始时使网络中的权重分布更加合理，避免梯度消失或梯度爆炸问题。*
19. softmax\_weights = tf.Variable(
20. tf.random.truncated\_normal([vocabulary\_size, embedding\_size],
21. stddev=1.0 / math.sqrt(embedding\_size)))
22. *# softmax\_biases 是 softmax 层的偏置项，形状为 [vocabulary\_size]，表示每个词汇都有一个对应的偏置*
23. *# tf.zeros([vocabulary\_size]) 使用全零初始化偏置项。这是常见的初始化方式，偏置值会在训练过程中更新*
24. softmax\_biases = tf.Variable(tf.zeros([vocabulary\_size]))
25. '''-----模型-----'''
26. embed = tf.nn.embedding\_lookup(embeddings, train\_dataset)
27. *# 将嵌入矩阵每行元素求和得到一个向量, 与skip-gram不同, cbow的输入是上下文向量的均值, 因此需要做相应变换*
28. context\_embeddings = []
29. for i in range(2 \* half\_window\_size):
30. *# 获取每个词的嵌入向量加入到数组中, train\_dataset[:, i]指提取第i列的数据*
31. context\_embeddings.append(tf.nn.embedding\_lookup(embeddings, train\_dataset[:, i]))
32. *# tf.stack() 函数将多个张量按照给定的维度堆叠起来，这里将上下文词向量堆叠成一个新的张量, 这里指的是沿着0维进行堆叠*
33. *# tf.reduce\_mean() 计算沿着第 axis: 0维（即行）对堆叠后的张量进行平均。 keep\_dims=False  表示在结果中不保留被平均的维度*
34. avg\_embed = tf.reduce\_mean(tf.stack(axis=0, values=context\_embeddings), 0, keep\_dims=False)
35. *# 计算 softmax 损失，每次使用负标签样本, 使用sampled\_softmax\_loss()来简化计算*
36. loss = tf.reduce\_mean(
37. tf.nn.sampled\_softmax\_loss(
38. weights=softmax\_weights, biases=softmax\_biases, labels=train\_labels, inputs=avg\_embed,
39. num\_sampled=num\_sampled,
40. num\_classes=vocabulary\_size
41. )
42. )
43. '''-----Optimizer-----'''
44. *# 使用 AdaGradOptimizer 进行优化，学习率会自动调整，因此初始学习率设置为 1 即可*
45. *# minimize()这个方法用于定义要最小化的目标函数。在这里, loss 是之前计算得到的损失值*
46. optimizer = tf.train.AdagradOptimizer(1.0).minimize(loss)
47. *# 计算验证集的相似度*
48. valid\_embed = tf.nn.embedding\_lookup(embeddings, valid\_dataset)
49. *# 计算小批量示例和所有嵌入向量之间的相似性, 使用余弦距离:*
50. similarity = tf.matmul(valid\_embed, tf.transpose(softmax\_weights)) + softmax\_biases
51. *# 计算嵌入向量 embeddings 的 L2 范数 norm*
52. norm = tf.sqrt(tf.reduce\_sum(tf.square(embeddings), 1, keep\_dims=True))
53. *# 再将 embeddings 除以 L2 范数，得到标准化后的 normalized\_embeddings*
54. normalized\_embeddings = embeddings / norm
55. *# 计算 softmax\_weights 的 L2 范数 norm*
56. norm\_ = tf.sqrt(tf.reduce\_sum(tf.square(softmax\_weights), 1, keep\_dims=True))
57. *# 再将 softmax\_weights 除以 L2 范数，得到标准化后的 normalized\_softmax\_weights*
58. normalized\_softmax\_weights = softmax\_weights / norm\_
59. *# 计算 (normalized\_softmax\_weights + normalized\_embeddings)^2 的 L2 范数 norm*
60. norm\_ = tf.sqrt(tf.reduce\_sum(tf.square(normalized\_softmax\_weights + normalized\_embeddings), 1, keep\_dims=True))
61. *# 计算 normalized\_softmax\_weights 和 normalized\_embeddings 的平均值, 并将结果除以 norm\_ 得到 normalized\_embeddings\_2*
62. normalized\_embeddings\_2 = (normalized\_softmax\_weights + normalized\_embeddings) / 2.0 / norm\_

接下来开始模型的训练。首先指定模型训练的轮次为20W，接着在我们上面指定的计算图上创建并管理一个会话，执行计算图里的操作。然后根据TensorFlow的版本匹配一下初始化的方法。声明平均损失变量用于计算每次迭代的平均损失，每次迭代先生成所需要的训练数据feed\_dict，然后把数据输入给session.run()方法执行计算图并且获取结果。这里还需要指定优化器以及相应的损失函数，这样便可以根据指定的损失函数，使用优化器进行权重的自动优化。方法返回的结果为第一个是一个占位符，指的是optimizer操作用来更新权重，我们并不需要该操作的返回值；第二个结果则是loss操作的返回值，这里需要进行累加求和计算该次迭代的平均损失。然后每2000词迭代我们计算并输出一下平均损失、每10000次迭代计算并输出指定单词的最相近的十个词来查看训练的结果。

模型训练结束之后，使用eval()方法返回张量的实际值，也就是最终的词嵌入矩阵用于之后的输出和进一步操作。

1. num\_steps = 200001
2. with tf.Session(graph=graph) as session:
3. if int(tf.version.VERSION.split('.')[1]) > 11:
4. tf.compat.v1.global\_variables\_initializer().run()
5. else:
6. tf.initialize\_all\_variables().run()
7. logging.info("初始化完成...")
8. average\_loss = 0
9. for step in range(num\_steps):
10. batch\_inputs, batch\_labels = generate\_batch(
11. \_batch\_size=batch\_size,
12. \_half\_window\_size=half\_window\_size
13. )
14. feed\_dict = {train\_dataset: batch\_inputs, train\_labels: batch\_labels}
15. \_, loss\_val = session.run([optimizer, loss], feed\_dict=feed\_dict)
16. average\_loss += loss\_val
17. if step % 2000 == 0:
18. if step > 0:
19. average\_loss /= 2000
20. logging.info("在第 %d 次循环的平均损失为: %f" % (step, average\_loss))
21. average\_loss = 0
22. if step % 10000 == 0:
23. logging.info("验证单词与最相似的 10 个单词：")
24. sim = similarity.eval()
25. for i in range(valid\_size):
26. valid\_word = reverse\_dictionary[valid\_examples[i]]
27. top\_k = 10  *# 取最相似的 10 个单词*
28. nearest = (-sim[i, :]).argsort()[1:top\_k + 1]
29. log\_str = "单词 %s 的最相似的 10 个单词为：" % valid\_word
30. for k in range(top\_k):
31. close\_word = reverse\_dictionary[nearest[k]]
32. log\_str = "%s %s," % (log\_str, close\_word)
33. logging.info(log\_str)
34. final\_embeddings = normalized\_embeddings.eval()
35. final\_embeddings\_2 = normalized\_embeddings\_2.eval()  *# 更好的结果*

最后是关于模型的保存：首先声明需要保存的点的数量，后面实验需要词向量的二维展示，因此需要将得到的词嵌入矩阵降为二维、最后使用Python的持久化存储方案，把词向量、词典保存为本地的pkl文件、方便后期的进一步操作和验证。

1. """----------结果----------"""
2. *# 后续需要处理的点的数量*
3. num\_points = 10000
4. *# 创建了 t-SNE 对象 tsne，n\_components 指定降维后的维度为2，init 指定了初始化方法为 pca，n\_iter 指定了最大迭代次数为 5000*
5. tsne\_2 = TSNE(perplexity=30, n\_components=2, init='pca', n\_iter=5000)
6. *# 对 final\_embeddings 中的前 50000 个嵌入（从索引 1 到 50000）进行 t-SNE 降维，并将结果存储在 two\_d\_embeddings 中*
7. two\_d\_embeddings = tsne\_2.fit\_transform(final\_embeddings[1:num\_points + 1, :])
8. two\_d\_embeddings\_2 = tsne\_2.fit\_transform(final\_embeddings\_2[1:num\_points + 1, :])
9. if os.getcwd() == colab\_cwd:
10. *# 处理好的文本存储在谷歌云盘*
11. output\_path = os.path.join(os.getcwd(), "drive", "MyDrive", "data", "embedding\_cbow.pkl")
12. else:
13. *# 本地的文件路径*
14. output\_path = os.path.join(os.getcwd(), "data", "embedding\_cbow.pkl")
15. with open(output\_path, 'wb') as f:
16. *# 使用 pickle 库将一个列表（包含 final\_embeddings, final\_embeddings\_2, two\_d\_embeddings, two\_d\_embeddings\_2 和 reverse\_dictionary ）序列化并写入到文件 f 中*
17. pickle.dump(
18. [final\_embeddings[:800000, :], final\_embeddings\_2[:800000, :], two\_d\_embeddings, two\_d\_embeddings\_2,
19. reverse\_dictionary], f)

### 4.5 Skip-Gram模型编写与训练

同样的，Skip-Gram模型的编写与CBOW模型类似，不同的地方仅在于训练所用数据生成函数有所不同、由于Skip-Gram使用的损失函数为nce\_loss，因此负采样的样本数量也有所不同为64、具体的graph定义不用再进行向量均值的计算、迭代轮次为100W次外，其余代码并没有很大的改变。

1. import collections
2. import math
3. import os
4. import logging
5. import random
6. import numpy as np
7. import tensorflow.compat.v1 as tf
8. from importlib import reload
9. from sklearn.manifold import TSNE
10. import pickle
11. """----------参数----------"""
12. colab\_cwd = '/content'
13. if os.getcwd() == colab\_cwd:
14. *# 处理好的文本存储在谷歌云盘*
15. file\_path = os.path.join(os.getcwd(), "drive", "MyDrive", "data", "txt", "wiki\_seg.txt")
16. else:
17. *# 本地的文件路径*
18. file\_path = os.path.join(os.getcwd(), "data", "txt", "wiki\_seg.txt")
19. *# 每次迭代将使用 256 个样本进行参数更新*
20. batch\_size = 256
21. *# 单词转为稠密向量的维度*
22. embedding\_size = 350
23. *# 左右考虑多少个单词*
24. skip\_window = 4
25. *# 重复使用输入以生成标签的次数*
26. num\_skips = 8
27. *# 用来抽取的验证单词数*
28. valid\_size = 10
29. *# 验证单词只从频数最高的 100 个单词中抽取*
30. valid\_window = 100
31. *# 需要验证的单词*
32. valid\_has\_word = ['父亲', '中国', '电脑', '手机', '书籍', '公里', '还', '部分', '年', '之后']
33. *# valid\_examples = np.array(random.sample(range(valid\_window), valid\_size))*
34. *# valid\_examples = np.random.choice(valid\_window, valid\_size, replace=False)*
35. *# 负采样的噪声单词的数量*
36. num\_sampled = 64
37. """----------logging配置----------"""
38. *# 使用 logging.info 打印信息，colab 需要 reload() 函数，否则无法打印*
39. reload(logging)
40. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
41. """----------数据处理----------"""
42. def read\_data(filename):
43. """
44. 从本地文件中读取数据，
45. @param filename:
46. @return:
47. """
48. logging.info("开始读取文件：" + filename)
49. with open(filename, "r", encoding="utf-8") as f:
50. *# 一次性读取所有的行，返回一个列表*
51. seg\_content = f.readlines()
52. seg\_words = []
53. *# 将单词都存放到seg\_words列表中，不去重，方便下一步建立以词频为基础的词表*
54. for i in range(len(seg\_content)):
55. if i % 5000 == 0:
56. logging.info("当前读取到第 " + str(i) + " 行，部分数据为: " + str(seg\_content[i][:50]))
57. *# 获取当前行词的列表，使用strip()函数过滤空格包括换行符或者制表符*
58. seg\_words.extend(seg\_content[i].strip().split(" "))
59. return seg\_words
60. all\_words\_list = read\_data(file\_path)
61. def build\_dataset(words, size):
62. """
63. 创建词表，包括原始词表、转换后的索引词表、单词-索引表和索引-单词表
64. @param words: 原始词表
65. @param size: 高频词词表长度
66. @return: 索引词表、原始词表、单词-索引词表、索引-单词词表
67. """
68. *# 初始化 count 列表，其中 UNK 代表 unknown, 即高频词表以外的词*
69. \_count = [['UNK', -1]]
70. *# 统计词频，只取前 vocabulary\_size 个高频词, 格式为：('词', 词频)*
71. \_word\_collection = collections.Counter(words)
72. logging.info('所有词的数量为：' + str(len(\_word\_collection)))
73. \_count.extend(\_word\_collection.most\_common(size - 1))
74. logging.info("高频词表前 20 个数据为数据为: " + str(\_count[:20]))
75. *# 构建字典，将词转化为索引, 词典顺序为高频词顺序, 格式为('词', 索引), 其中索引从 0 开始*
76. \_dictionary = dict()
77. for \_word, \_ in \_count:
78. \_dictionary[\_word] = len(\_dictionary)
79. *# 将词转化为索引存储到\_data中, 词在高频词词典的话，索引为高频词词典的索引, 如果词不在字典中, 则转化为 UNK*
80. *# 此时 \_data 就是原来词的列表的索引列表*
81. \_data = list()
82. unk\_count = 0
83. for \_word in words:
84. if \_word in \_dictionary:
85. index = \_dictionary[\_word]
86. else:
87. index = 0
88. unk\_count += 1
89. \_data.append(index)
90. \_count[0][1] = unk\_count
91. *# 构建反向字典, 可以快速从词索引转化为词即 (索引-'词')*
92. \_reverse\_dictionary = dict(zip(\_dictionary.values(), \_dictionary.keys()))
93. return \_data, \_count, \_dictionary, \_reverse\_dictionary
94. *# 高频词词表大小, 对于 4w 的切片， 10w 词表的结果较好*
95. vocabulary\_size = 800000
96. *# 构建词表*
97. data, count, dictionary, reverse\_dictionary = build\_dataset(all\_words\_list, vocabulary\_size)
98. logging.info('所有词的数量为：' + str(len(count)))
99. logging.info('含 UNK 的前 20 个高频词' + str(count[: 20]))
100. logging.info('查看索引与词的映射：')
101. logging\_str = ''.join(
102. [f"{idx} : {word}, " for idx, word in zip(data[:10], [reverse\_dictionary[i] for i in data[:10]])])
103. logging.info(logging\_str)
104. *# 完善验证单词的索引*
105. valid\_examples = []
106. *# 如果待验证词在此表中，那么直接获得索引，否则指定为 unknown*
107. for word in valid\_has\_word:
108. if dictionary.get(word, None) is not None:
109. valid\_examples.append(dictionary[word])
110. else:
111. valid\_examples.append(0)
112. *# 删除原始词表，节省内存*
113. del all\_words\_list
114. data\_index = 0
115. def generate\_batch(\_batch\_size, \_num\_skips, \_skip\_window):
116. """
117. 生成训练用的 batch 数据以及标签数据，其中 batch 是 上下文， label 是 目标词
118. @param \_batch\_size: batch 的大小
119. @param \_num\_skips: 目标词的上下文窗口大小（只有一半）
120. @param \_skip\_window: 目标词的上下文窗口大小（只有一半）
121. @return: 生成的 batch 数据 和 label 数据
122. """
123. global data\_index
124. assert \_batch\_size % \_num\_skips == 0
125. assert \_num\_skips <= 2 \* \_skip\_window
126. *# 初始化二维数组, 行数为 \_batch\_size，列数为 2 \* half\_window\_size, 数据类型为 int32*
127. \_batch = np.ndarray(shape=(\_batch\_size), dtype=np.int32)
128. \_labels = np.ndarray(shape=(\_batch\_size, 1), dtype=np.int32)
129. span = 2 \* \_skip\_window + 1
130. len\_data = len(data)
131. buffer = collections.deque(maxlen=span)
132. for \_ in range(span):
133. buffer.append(data[data\_index])
134. data\_index = (data\_index + 1) % len\_data
135. for i in range(\_batch\_size // \_num\_skips):
136. target = \_skip\_window  *# target label at the center of the buffer*
137. targets\_to\_avoid = [\_skip\_window]
138. for j in range(\_num\_skips):
139. while target in targets\_to\_avoid:
140. target = random.randint(0, span - 1)
141. targets\_to\_avoid.append(target)
142. \_batch[i \* \_num\_skips + j] = buffer[\_skip\_window]
143. \_labels[i \* \_num\_skips + j, 0] = buffer[target]
144. buffer.append(data[data\_index])
145. data\_index = (data\_index + 1) % len\_data
146. return \_batch, \_labels
147. """----------训练 & 验证----------"""
148. *# 创建默认的 graph*
149. graph = tf.Graph()
150. with graph.as\_default():
151. '''-----输入-----'''
152. *# 创建一个 TensorFlow 占位符的，类型为 32 位整数，大小为 batch\_size 的一维张量*
153. *# train\_inputs = tf.placeholder(tf.int32, shape=[batch\_size])*
154. train\_inputs = tf.placeholder(tf.int32, shape=[batch\_size])
155. *# 创建一个 TensorFlow 占位符的，类型为 32 位整数，大小为 batch\_size x 1 的二维张量*
156. train\_labels = tf.placeholder(tf.int32, shape=[batch\_size, 1])
157. *# 创建常量张量, 值为 valid\_examples 列表中的元素, 制定了数据类型, 用于后续作为固定输入*
158. valid\_dataset = tf.constant(valid\_examples, dtype=tf.int32)
159. '''-----变量-----'''
160. *# 词嵌入矩阵, 形状为 [vocabulary\_size, embedding\_size], 通过训练，嵌入矩阵会学习到每个词的分布式表示（即词向量）*
161. *# tf.random.uniform() 用于从均匀分布中随机初始化嵌入向量，范围是 [-1.0, 1.0]。给每个词的嵌入向量的初始值是在 -1 到 1 之间的随机数。*
162. embeddings = tf.Variable(tf.random.uniform([vocabulary\_size, embedding\_size], -1.0, 1.0))
163. *# softmax 层的权重矩阵，形状为 [vocabulary\_size, embedding\_size], 用于在通过模型训练时，将词嵌入向量映射到词汇表中每个词的概率分布*
164. *# tf.random.truncated\_normal() 用于从截断的正态分布中随机初始化权重。*
165. *# 标准差 stddev=1.0 / math.sqrt(embedding\_size) 是一种常见的初始化方式，它有助于在训练开始时使网络中的权重分布更加合理，避免梯度消失或梯度爆炸问题。*
166. softmax\_weights = tf.Variable(
167. tf.random.truncated\_normal([vocabulary\_size, embedding\_size],
168. stddev=1.0 / math.sqrt(embedding\_size)))
169. *# softmax\_biases 是 softmax 层的偏置项，形状为 [vocabulary\_size]，表示每个词汇都有一个对应的偏置*
170. *# tf.zeros([vocabulary\_size]) 使用全零初始化偏置项。这是常见的初始化方式，偏置值会在训练过程中更新*
171. softmax\_biases = tf.Variable(tf.zeros([vocabulary\_size]))
172. '''-----模型-----'''
173. embed = tf.nn.embedding\_lookup(embeddings, train\_inputs)
174. *# 转化变量输入，适配NCE*
175. nce\_weights = tf.Variable(
176. tf.truncated\_normal([vocabulary\_size, embedding\_size],
177. stddev=1.0 / math.sqrt(embedding\_size)))
178. nce\_biases = tf.Variable(tf.zeros([vocabulary\_size]), dtype=tf.float32)
179. loss = tf.reduce\_mean(tf.nn.nce\_loss(weights=nce\_weights,
180. biases=nce\_biases,
181. inputs=embed,
182. labels=train\_labels,
183. num\_sampled=num\_sampled,
184. num\_classes=vocabulary\_size))
185. *# # 将嵌入矩阵每行元素求和得到一个向量*
186. *# inputs = tf.reduce\_sum(embed, 1)*
187. *# # 计算 softmax 损失，每次使用负标签样本*
188. *# loss = tf.reduce\_mean(*
189. *#     tf.nn.sampled\_softmax\_loss(*
190. *#         softmax\_weights, softmax\_biases, train\_labels, embed, num\_sampled, vocabulary\_size*
191. *#     )*
192. *# )*
193. '''-----Optimizer-----'''
194. optimizer = tf.train.AdagradOptimizer(1.0).minimize(loss)
195. norm = tf.sqrt(tf.reduce\_sum(tf.square(embeddings), 1, keep\_dims=True))
196. normalized\_embeddings = embeddings / norm
197. valid\_embeddings = tf.nn.embedding\_lookup(
198. normalized\_embeddings, valid\_dataset)
199. similarity = tf.matmul(valid\_embeddings, tf.transpose(normalized\_embeddings))
200. embeddings\_2 = (normalized\_embeddings + softmax\_weights) / 2.0
201. norm\_ = tf.sqrt(tf.reduce\_sum(tf.square(embeddings\_2), 1, keep\_dims=True))
202. normalized\_embeddings\_2 = embeddings\_2 / norm\_
203. num\_steps = 1000001
204. with tf.Session(graph=graph) as session:
205. if int(tf.version.VERSION.split('.')[1]) > 11:
206. tf.compat.v1.global\_variables\_initializer().run()
207. else:
208. tf.initialize\_all\_variables().run()
209. logging.info("初始化完成...")
210. average\_loss = 0
211. for step in range(num\_steps):
212. batch\_inputs, batch\_labels = generate\_batch(
213. \_batch\_size=batch\_size,
214. \_num\_skips=num\_skips,
215. \_skip\_window=skip\_window
216. )
217. feed\_dict = {train\_inputs: batch\_inputs, train\_labels: batch\_labels}
218. \_, loss\_val = session.run([optimizer, loss], feed\_dict=feed\_dict)
219. average\_loss += loss\_val
220. if step % 2000 == 0:
221. if step > 0:
222. average\_loss /= 2000
223. logging.info("在第 %d 次循环的平均损失为: %f" % (step, average\_loss))
224. average\_loss = 0
225. if step % 10000 == 0:
226. logging.info("验证单词与最相似的 10 个单词：")
227. sim = similarity.eval()
228. for i in range(valid\_size):
229. valid\_word = reverse\_dictionary[valid\_examples[i]]
230. top\_k = 10  *# 取最相似的 10 个单词*
231. nearest = (-sim[i, :]).argsort()[1:top\_k + 1]
232. log\_str = "单词 %s 的最相似的 10 个单词为：" % valid\_word
233. for k in range(top\_k):
234. close\_word = reverse\_dictionary[nearest[k]]
235. log\_str = "%s %s," % (log\_str, close\_word)
236. logging.info(log\_str)
237. final\_embeddings = normalized\_embeddings.eval()
238. final\_embeddings\_2 = normalized\_embeddings\_2.eval()  *# 更好的结果*
239. """----------结果----------"""
240. *# 后续需要处理的点的数量*
241. num\_points = 10000
242. *# 创建了 t-SNE 对象 tsne，n\_components 指定降维后的维度为2，init 指定了初始化方法为 pca，n\_iter 指定了最大迭代次数为 5000*
243. tsne\_2 = TSNE(perplexity=30, n\_components=2, init='pca', n\_iter=5000)
244. *# 对 final\_embeddings 中的前 400 个嵌入（从索引 1 到 400）进行 t-SNE 降维，并将结果存储在 two\_d\_embeddings 中*
245. two\_d\_embeddings = tsne\_2.fit\_transform(final\_embeddings[1:num\_points + 1, :])
246. two\_d\_embeddings\_2 = tsne\_2.fit\_transform(final\_embeddings\_2[1:num\_points + 1, :])
247. if os.getcwd() == colab\_cwd:
248. *# 处理好的文本存储在谷歌云盘*
249. output\_path = os.path.join(os.getcwd(), "drive", "MyDrive", "data", "embedding\_skip\_gram.pkl")
250. else:
251. *# 本地的文件路径*
252. output\_path = os.path.join(os.getcwd(), "data", "embedding\_skip\_gram.pkl")
253. with open(output\_path, 'wb') as f:
254. pickle.dump([final\_embeddings[:800000, :], final\_embeddings\_2[:800000, :], two\_d\_embeddings, two\_d\_embeddings\_2,
255. reverse\_dictionary], f)

### 4.6 模型训练结果的验证与类比实验

将模型保存到本地之后，接下来就是相似度的计算与类比实验的验证，代码如下：

#### 4.6.1 结果验证与类比实验代码

verification.py文件：

1. import logging
2. import pickle
3. import numpy as np
4. from numpy import dot
5. from importlib import reload
6. from numpy.linalg import norm
7. from sklearn.decomposition import PCA
8. from sklearn.metrics.pairwise import cosine\_similarity
9. from sklearn.preprocessing import normalize
10. """----------logging配置----------"""
11. *# 使用 logging.info 打印信息，colab 需要 reload() 函数，否则无法打印*
12. reload(logging)
13. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
14. *# 获取给定词的最相近的词*
15. def find\_most\_similar(word\_vector, \_embeddings, top\_n=5):
16. similarities = []
17. word\_vector\_reshaped = word\_vector.reshape(1, -1)
18. for i in range(len(\_embeddings)):
19. vector\_2\_reshaped = \_embeddings[i].reshape(1, -1)
20. sim = cosine\_similarity(word\_vector\_reshaped, vector\_2\_reshaped)
21. similarities.append((reverse\_dictionary[i], sim))
22. if i > 50000:
23. break
24. *# 按相似度排序，取前n个*
25. similarities = sorted(similarities, key=lambda x: x[1], reverse=True)
26. return similarities[:top\_n]
27. file\_path\_list = ['data/embedding\_cbow\_80w.pkl', 'data/embedding\_skip\_gram\_80w.pkl']
28. *# 加载 .pkl 文件*
29. for file\_path in file\_path\_list:
30. with open(file\_path, 'rb') as f:
31. *# embeddings 为列表，reverse\_dictionary 为字典，格式为{索引：单词}*
32. embeddings, embeddings\_2, two\_d\_embeddings, two\_d\_embeddings\_2, reverse\_dictionary = pickle.load(f)
33. logging.info("读取本地模型" + file\_path + "完成...")
34. word\_index\_dict = dict(zip(reverse\_dictionary.values(), reverse\_dictionary.keys()))
35. *# 选择十个单词*
36. valid\_word = ['电脑', '书籍', '公里', '还']
37. *# valid\_word = ['父亲', '中国', '电脑', '手机', '书籍', '公里', '还', '部分', '年', '之后']*
38. *# 完善验证单词的索引*
39. valid\_word\_index = []
40. *# 如果待验证词在此表中，那么直接获得索引，否则指定为 unknown*
41. for word in valid\_word:
42. if word\_index\_dict.get(word, None) is not None:
43. valid\_word\_index.append(word\_index\_dict[word])
44. else:
45. valid\_word\_index.append(0)
46. for word in valid\_word\_index:
47. out\_str = '当前词的索引为：' + str(word) + '，词为：' + str(reverse\_dictionary[word]) + ', 词向量均值为：' + str(
48. np.mean(embeddings[word])) + ', 最相近的前十个词为：\n'
49. similar\_words = find\_most\_similar(embeddings[word], embeddings, 11)
50. for sim\_word in similar\_words:
51. out\_str += str(sim\_word[0]) + ', 词向量均值为：' + str(
52. np.mean(embeddings[word\_index\_dict[sim\_word[0]]])) + ', 相似度为：' + str(
53. sim\_word[1][0][0]) + '\n'
54. logging.info(out\_str)
55. *# break*
56. *# 类比实验*
57. logging.info('\n')
58. word\_vec\_father = embeddings[word\_index\_dict['父亲']]
59. logging.info('父亲 的词向量均值为：' + str(np.mean(word\_vec\_father)) + ', 索引为：' + str(word\_index\_dict['父亲']))
60. word\_vec\_man = embeddings[word\_index\_dict['男人']]
61. logging.info('男人 词向量均值为：' + str(np.mean(word\_vec\_man)) + ', 索引为：' + str(word\_index\_dict['男人']))
62. word\_vec\_woman = embeddings[word\_index\_dict['女人']]
63. logging.info('女人 词向量均值为：' + str(np.mean(word\_vec\_woman)) + ', 索引为：' + str(word\_index\_dict['女人']))
64. word\_vec\_sub = word\_vec\_father - word\_vec\_man + word\_vec\_woman
65. word\_vec\_res = find\_most\_similar(word\_vec\_sub, embeddings, 11)
66. logging.info('父亲-男人+女人 之间的类比实验，最相近的前十个词为:')
67. for word in word\_vec\_res:
68. logging.info(
69. word[0] + ', 词向量为：' + str(embeddings[word\_index\_dict[word[0]]][0]) + ', 相似度为：' + str(word[1][0][0]))
70. logging.info('\n')
71. word\_vec\_king = embeddings[word\_index\_dict['国王']]
72. logging.info('国王 的词向量均值为：' + str(np.mean(word\_vec\_king)) + ', 索引为：' + str(word\_index\_dict['国王']))
73. logging.info('男人 词向量均值为：' + str(np.mean(word\_vec\_man)) + ', 索引为：' + str(word\_index\_dict['男人']))
74. logging.info('女人 词向量均值为：' + str(np.mean(word\_vec\_woman)) + ', 索引为：' + str(word\_index\_dict['女人']))
75. word\_vec\_queen = embeddings[word\_index\_dict['女王']]
76. logging.info('女王 词向量均值为：' + str(np.mean(word\_vec\_woman)) + ', 索引为：' + str(word\_index\_dict['女王']))
77. word\_vec\_sub = word\_vec\_king - word\_vec\_man + word\_vec\_woman
78. word\_vec\_res = find\_most\_similar(word\_vec\_sub, embeddings, 11)
79. logging.info('国王-男人+女人 之间的类比实验，最相近的前十个词为:')
80. for word in word\_vec\_res:
81. logging.info(
82. word[0] + ', 词向量均值为：' + str(np.mean(embeddings[word\_index\_dict[word[0]]])) + ', 相似度为：' + str(
83. word[1][0][0]))
84. logging.info('\n')
85. word\_vec\_pin\_pai = embeddings[word\_index\_dict['乒乓球拍']]
86. logging.info(
87. '乒乓球拍 的词向量均值为：' + str(np.mean(word\_vec\_pin\_pai)) + ', 索引为：' + str(word\_index\_dict['乒乓球拍']))
88. word\_vec\_pin = embeddings[word\_index\_dict['乒乓球']]
89. logging.info('乒乓球 词向量均值为：' + str(np.mean(word\_vec\_pin)) + ', 索引为：' + str(word\_index\_dict['乒乓球']))
90. word\_vec\_yu = embeddings[word\_index\_dict['羽毛球拍']]
91. logging.info('羽毛球拍 词向量均值为：' + str(np.mean(word\_vec\_yu)) + ', 索引为：' + str(word\_index\_dict['羽毛球拍']))
92. word\_vec\_yu\_qiu = embeddings[word\_index\_dict['羽毛球']]
93. logging.info(
94. '羽毛球拍 词向量均值为：' + str(np.mean(word\_vec\_yu\_qiu)) + ', 索引为：' + str(word\_index\_dict['羽毛球']))
95. word\_vec\_sub = word\_vec\_pin\_pai - word\_vec\_pin + word\_vec\_yu\_qiu
96. word\_vec\_res = find\_most\_similar(word\_vec\_sub, embeddings, 11)
97. logging.info('乒乓球拍-乒乓球+羽毛球拍 之间的类比实验，最相近的前十个词为:')
98. for word in word\_vec\_res:
99. logging.info(
100. word[0] + ', 词向量为：' + str(np.mean(embeddings[word\_index\_dict[word[0]]])) + ', 相似度为：' + str(
101. word[1][0][0]))
102. logging.info('\n')
103. word\_vec\_uk = embeddings[word\_index\_dict['英国']]
104. logging.info('英国 的词向量均值为：' + str(np.mean(word\_vec\_uk)) + ', 索引为：' + str(word\_index\_dict['英国']))
105. word\_vec\_lon = embeddings[word\_index\_dict['伦敦']]
106. logging.info('伦敦 词向量均值为：' + str(np.mean(word\_vec\_lon)) + ', 索引为：' + str(word\_index\_dict['伦敦']))
107. word\_vec\_usa = embeddings[word\_index\_dict['美国']]
108. logging.info('美国 词向量均值为：' + str(np.mean(word\_vec\_usa)) + ', 索引为：' + str(word\_index\_dict['美国']))
109. word\_vec\_hua = embeddings[word\_index\_dict['华盛顿']]
110. logging.info('华盛顿 词向量均值为：' + str(np.mean(word\_vec\_hua)) + ', 索引为：' + str(word\_index\_dict['华盛顿']))
111. word\_vec\_sub = word\_vec\_uk - word\_vec\_lon + word\_vec\_hua
112. word\_vec\_res = find\_most\_similar(word\_vec\_sub, embeddings, 11)
113. logging.info('英国-伦敦+美国 之间的类比实验，最相近的前十个词为:')
114. for word in word\_vec\_res:
115. logging.info(
116. word[0] + ', 词向量为：' + str(np.mean(embeddings[word\_index\_dict[word[0]]])) + ', 相似度为：' + str(
117. word[1][0][0]))

对于给定的词，首先查出它的词向量，因为词向量维度为350，为了便于展示这里计算了词向量的均值，然后通过余弦相似度的计算该词与词表中的词的相似度，之后再根据相似度的值进行排序，输出前十个最相似的词，查看训练的结果如何。test\_skip\_gram.py文件的代码和test\_cbow.py文件几乎一致。

#### 4.6.2 词向量二维可视化代码

二维可视化主要使用的是matplotlib库，这里需要注意中文默认是无法显示的，需要使用以下代码：

plt.rcParams['font.sans-serif'] = ['SimHei']  *# 使用黑体*

设置中文的字体才可以正常显示。绘制的结果保存在本地PDF便于查看实验结果。

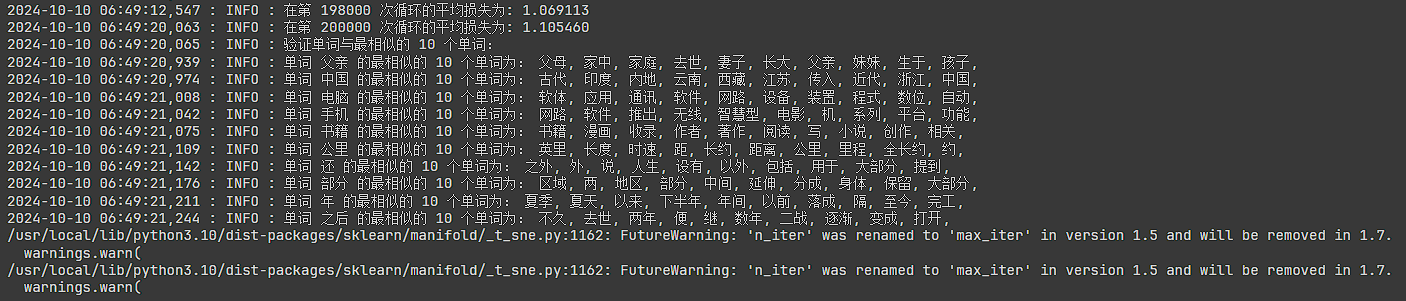
visualize.py文件：

1. import logging
2. from importlib import reload
3. *# -\*- coding: utf-8 -\*-*
4. from matplotlib import pylab as plt
5. import pickle
6. from matplotlib.backends.backend\_pdf import PdfPages
7. """----------logging配置----------"""
8. *# 使用 logging.info 打印信息，colab 需要 reload() 函数，否则无法打印*
9. reload(logging)
10. logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
11. def plot(embeddings, labels, save\_to\_pdf='embed.pdf'):
12. assert embeddings.shape[0] >= len(labels), 'More labels than embeddings'
13. pp = PdfPages(save\_to\_pdf)
14. plt.rcParams['font.sans-serif'] = ['SimHei']  *# 使用黑体*
15. plt.rcParams['font.weight'] = 'light'  *# 全局加粗字体*
16. plt.rcParams['axes.unicode\_minus'] = False  *# 解决负号显示为方块的问题*
17. plt.figure(figsize=(30, 30))  *# in inches*
18. for i, label in enumerate(labels):
19. x, y = embeddings[i, :]
20. plt.scatter(x, y)
21. plt.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points',
22. ha='right', va='bottom')
23. plt.savefig(pp, format='pdf')
24. plt.show()
25. pp.close()
26. methods = ['cbow', 'skip\_gram']
27. for method in methods:
28. filename = 'data/embedding\_%s\_80w.pkl' % method
29. with open(filename, 'rb') as f:
30. [embeddings, embeddings\_2, two\_d\_embeddings, two\_d\_embeddings\_2, reverse\_dictionary] = pickle.load(f)
31. num\_points = 1500
32. words = [reverse\_dictionary[i] for i in range(1, num\_points + 1)]
33. logging.info(method + '保存文件中...')
34. plot(two\_d\_embeddings, words, save\_to\_pdf='embeddings\_%s\_80w.pdf' % method)
35. plot(two\_d\_embeddings\_2, words, save\_to\_pdf='embeddings\_2\_%s\_80w.pdf' % method)

## 实验结果

### 5.1 模型训练结果展示

由于本机没有GPU，因此本次实验的代码在Google Colab平台运行，其中CBOW模型的运行时间约为40-60min，最后损失的均值收敛在1左右；Skip-Gram模型的运行时间约为1.5h，最后损失的均值在6-7左右；两者保存为本地的pkl文件大小在2.1G。

CBOW模型在Colab的最终结果：

可以看到，在进行了20W次的迭代之后，模型的训练效果还算让人满意，可以捕捉到部分词的相关性如父亲、电脑、公里等；但依然有一些词的训练结果较差如还、之后。

电脑萤幕画面

描述已自动生成Skip-Gram模型在Colab的最终结果：

### 5.2 CBOW模型结果验证与类比实验

CBOW模型实验结果：

1. D:\ProgramData\anaconda3\envs\NLP\_exp\python.exe D:\Files\OneDrive\Codes\PyCharmProjects\NLP\_exp\_1\test\_cbow.py
2. 2024-10-12 21:20:28,111 : INFO : 读取本地模型embedding\_cbow\_80w.pkl完成...
3. 2024-10-12 21:20:40,340 : INFO : 当前词的索引为：1438，词为：电脑, 词向量均值为：-0.0011241792, 最相近的前十个词为：
4. 电脑, 词向量均值为：-0.0011241792, 相似度为：1.0
5. 技术, 词向量均值为：-0.0019253179, 相似度为：0.27429324
6. 应用, 词向量均值为：-0.0011420895, 相似度为：0.26118135
7. 社会科学, 词向量均值为：0.003975382, 相似度为：0.25234348
8. 软体, 词向量均值为：0.0034108905, 相似度为：0.2519
9. 产品, 词向量均值为：0.00071529916, 相似度为：0.22427164
10. 开发, 词向量均值为：0.00026530403, 相似度为：0.22378248
11. 格式, 词向量均值为：-0.0008403032, 相似度为：0.22176531
12. 图曼, 词向量均值为：0.0006334874, 相似度为：0.2112391
13. 发表, 词向量均值为：-0.00019130255, 相似度为：0.2103292
14. 起家, 词向量均值为：0.0027792335, 相似度为：0.21032766
15. 2024-10-12 21:20:50,885 : INFO : 当前词的索引为：1668，词为：书籍, 词向量均值为：-0.0027298422, 最相近的前十个词为：
16. 书籍, 词向量均值为：-0.0027298422, 相似度为：1.0
17. 单行本, 词向量均值为：1.471298e-05, 相似度为：0.24198876
18. 乐曲, 词向量均值为：-0.0059630503, 相似度为：0.23195763
19. 小说, 词向量均值为：0.0029045397, 相似度为：0.22748637
20. 出刊, 词向量均值为：-0.0012178723, 相似度为：0.21572575
21. 学部, 词向量均值为：-0.002394707, 相似度为：0.21567068
22. 丛书, 词向量均值为：-0.0023043407, 相似度为：0.2123223
23. 老山, 词向量均值为：-0.0012118102, 相似度为：0.21059144
24. 介绍, 词向量均值为：0.0014630107, 相似度为：0.21020614
25. 实录, 词向量均值为：-0.0011865124, 相似度为：0.208154
26. 该书, 词向量均值为：-0.0017104819, 相似度为：0.20656857
27. 2024-10-12 21:21:01,061 : INFO : 当前词的索引为：103，词为：公里, 词向量均值为：-0.0006136574, 最相近的前十个词为：
28. 公里, 词向量均值为：-0.0006136574, 相似度为：1.0
29. 千米, 词向量均值为：-0.007788692, 相似度为：0.36377603
30. 距离, 词向量均值为：0.00065798673, 相似度为：0.31202328
31. 公路, 词向量均值为：-0.0042205877, 相似度为：0.2508559
32. 东南亚, 词向量均值为：0.0018318846, 相似度为：0.24860367
33. 公尺, 词向量均值为：-0.006814492, 相似度为：0.23382309
34. 前往, 词向量均值为：-0.0054325634, 相似度为：0.23226823
35. 东南, 词向量均值为：-0.0005565974, 相似度为：0.23125564
36. 菊科, 词向量均值为：0.0014516491, 相似度为：0.22885793
37. 南北, 词向量均值为：-0.0012936426, 相似度为：0.22668952
38. 位于, 词向量均值为：-0.0047262865, 相似度为：0.22473411
39. 2024-10-12 21:21:11,312 : INFO : 当前词的索引为：65，词为：还, 词向量均值为：-0.0015848228, 最相近的前十个词为：
40. 还, 词向量均值为：-0.0015848228, 相似度为：1.0
41. 亦, 词向量均值为：-0.0022287816, 相似度为：0.23433933
42. 横贯, 词向量均值为：-0.0030910897, 相似度为：0.21980272
43. 只, 词向量均值为：0.00025804972, 相似度为：0.21389204
44. 纸币, 词向量均值为：0.0034305286, 相似度为：0.20780483
45. 罗夫斯基, 词向量均值为：-0.0012086013, 相似度为：0.20076478
46. 经常, 词向量均值为：-0.0015510044, 相似度为：0.1985276
47. 欧尔, 词向量均值为：0.002014645, 相似度为：0.19836311
48. 教职员工, 词向量均值为：-0.00041866302, 相似度为：0.1982709
49. 崔克, 词向量均值为：-0.0037145328, 相似度为：0.19747551
50. 仍, 词向量均值为：-0.0021240471, 相似度为：0.19329679
51. 2024-10-12 21:21:11,312 : INFO :
52. 2024-10-12 21:21:11,312 : INFO : 父亲 的词向量均值为：-0.0015486331, 索引为：225
53. 2024-10-12 21:21:11,312 : INFO : 男人 词向量均值为：-0.00039755634, 索引为：1499
54. 2024-10-12 21:21:11,312 : INFO : 女人 词向量均值为：-0.0021459812, 索引为：1225
55. 2024-10-12 21:21:21,491 : INFO : 父亲-男人+女人 之间的类比实验，最相近的前十个词为:
56. 2024-10-12 21:21:21,491 : INFO : 父亲, 词向量为：-0.033893965, 相似度为：0.58650124
57. 2024-10-12 21:21:21,491 : INFO : 女人, 词向量为：0.013143478, 相似度为：0.5777545
58. 2024-10-12 21:21:21,492 : INFO : 母亲, 词向量为：-0.011030801, 相似度为：0.2827679
59. 2024-10-12 21:21:21,492 : INFO : 儿子, 词向量为：-0.060525887, 相似度为：0.2746635
60. 2024-10-12 21:21:21,492 : INFO : 搭建, 词向量为：0.040822297, 相似度为：0.22314355
61. 2024-10-12 21:21:21,492 : INFO : 女儿, 词向量为：0.018916605, 相似度为：0.22140369
62. 2024-10-12 21:21:21,492 : INFO : 努尔哈赤, 词向量为：-0.0036181263, 相似度为：0.2203364
63. 2024-10-12 21:21:21,492 : INFO : 峡湾, 词向量为：0.0792164, 相似度为：0.21961585
64. 2024-10-12 21:21:21,492 : INFO : 跨足, 词向量为：-0.08107588, 相似度为：0.21151924
65. 2024-10-12 21:21:21,492 : INFO : 新垣, 词向量为：0.024914315, 相似度为：0.20951203
66. 2024-10-12 21:21:21,492 : INFO : 领跑, 词向量为：-0.032492228, 相似度为：0.20231092
67. 2024-10-12 21:21:21,492 : INFO :
68. 2024-10-12 21:21:21,492 : INFO : 国王 的词向量均值为：0.0024870695, 索引为：490
69. 2024-10-12 21:21:21,492 : INFO : 男人 词向量均值为：-0.00039755634, 索引为：1499
70. 2024-10-12 21:21:21,492 : INFO : 女人 词向量均值为：-0.0021459812, 索引为：1225
71. 2024-10-12 21:21:21,492 : INFO : 女王 词向量均值为：-0.0021459812, 索引为：1650
72. 2024-10-12 21:21:31,581 : INFO : 国王-男人+女人 之间的类比实验，最相近的前十个词为:
73. 2024-10-12 21:21:31,581 : INFO : 国王, 词向量均值为：0.0024870695, 相似度为：0.58138704
74. 2024-10-12 21:21:31,583 : INFO : 女人, 词向量均值为：-0.0021459812, 相似度为：0.5375765
75. 2024-10-12 21:21:31,583 : INFO : 公爵, 词向量均值为：0.0039125676, 相似度为：0.26274714
76. 2024-10-12 21:21:31,583 : INFO : 一世, 词向量均值为：-0.0019017189, 相似度为：0.23277119
77. 2024-10-12 21:21:31,583 : INFO : 南方, 词向量均值为：0.0008572868, 相似度为：0.23196402
78. 2024-10-12 21:21:31,583 : INFO : 土耳其, 词向量均值为：0.0021758906, 相似度为：0.22741269
79. 2024-10-12 21:21:31,583 : INFO : 月氏, 词向量均值为：-0.0011835509, 相似度为：0.2240567
80. 2024-10-12 21:21:31,583 : INFO : 体力, 词向量均值为：-0.00093555637, 相似度为：0.21254846
81. 2024-10-12 21:21:31,583 : INFO : 加倍, 词向量均值为：-0.00023392864, 相似度为：0.20865786
82. 2024-10-12 21:21:31,584 : INFO : 王国, 词向量均值为：0.0012887603, 相似度为：0.20007038
83. 2024-10-12 21:21:31,584 : INFO : 宝塔, 词向量均值为：-0.0043801125, 相似度为：0.19912004
84. 2024-10-12 21:21:31,584 : INFO :
85. 2024-10-12 21:21:31,584 : INFO : 乒乓球拍 的词向量均值为：-0.0018855148, 索引为：569656
86. 2024-10-12 21:21:31,584 : INFO : 乒乓球 词向量均值为：0.003969798, 索引为：6196
87. 2024-10-12 21:21:31,584 : INFO : 羽毛球拍 词向量均值为：0.0033017457, 索引为：475174
88. 2024-10-12 21:21:31,584 : INFO : 羽毛球拍 词向量均值为：0.0026892521, 索引为：384
89. 2024-10-12 21:21:41,759 : INFO : 乒乓球拍-乒乓球+羽毛球拍 之间的类比实验，最相近的前十个词为:
90. 2024-10-12 21:21:41,759 : INFO : 羽毛球, 词向量为：0.0026892521, 相似度为：0.60010624
91. 2024-10-12 21:21:41,759 : INFO : 咖啡, 词向量为：0.0014812454, 相似度为：0.23321776
92. 2024-10-12 21:21:41,759 : INFO : 真君, 词向量为：-0.00051066256, 相似度为：0.21059446
93. 2024-10-12 21:21:41,760 : INFO : 李氏, 词向量为：-0.0047052572, 相似度为：0.21020365
94. 2024-10-12 21:21:41,760 : INFO : 扔进, 词向量为：-0.0019655735, 相似度为：0.20195782
95. 2024-10-12 21:21:41,760 : INFO : 魔法师, 词向量为：0.002083024, 相似度为：0.19711447
96. 2024-10-12 21:21:41,760 : INFO : 一种, 词向量为：-0.0023141487, 相似度为：0.19502565
97. 2024-10-12 21:21:41,760 : INFO : 优秀, 词向量为：0.0031026131, 相似度为：0.19052544
98. 2024-10-12 21:21:41,760 : INFO : 高生, 词向量为：0.0037019409, 相似度为：0.18839708
99. 2024-10-12 21:21:41,760 : INFO : 利沃夫, 词向量为：-0.002174166, 相似度为：0.1872812
100. 2024-10-12 21:21:41,760 : INFO : 源泉, 词向量为：0.003772194, 相似度为：0.18674247
101. 2024-10-12 21:21:41,760 : INFO :
102. 2024-10-12 21:21:41,760 : INFO : 英国 的词向量均值为：0.00073203, 索引为：75
103. 2024-10-12 21:21:41,760 : INFO : 伦敦 词向量均值为：-0.0023354378, 索引为：753
104. 2024-10-12 21:21:41,760 : INFO : 美国 词向量均值为：0.0048495266, 索引为：13
105. 2024-10-12 21:21:41,760 : INFO : 华盛顿 词向量均值为：0.006913188, 索引为：1926
106. 2024-10-12 21:21:52,017 : INFO : 英国-伦敦+美国 之间的类比实验，最相近的前十个词为:
107. 2024-10-12 21:21:52,017 : INFO : 华盛顿, 词向量为：0.006913188, 相似度为：0.6318234
108. 2024-10-12 21:21:52,017 : INFO : 英国, 词向量为：0.00073203, 相似度为：0.5415391
109. 2024-10-12 21:21:52,017 : INFO : 哈特利, 词向量为：0.0018732699, 相似度为：0.22038114
110. 2024-10-12 21:21:52,017 : INFO : 影史, 词向量为：0.0023777704, 相似度为：0.21551321
111. 2024-10-12 21:21:52,017 : INFO : 洛斯, 词向量为：0.0026530449, 相似度为：0.20685093
112. 2024-10-12 21:21:52,017 : INFO : 斯塔德, 词向量为：0.0039446508, 相似度为：0.20604959
113. 2024-10-12 21:21:52,017 : INFO : 火海, 词向量为：-0.0003327374, 相似度为：0.20253435
114. 2024-10-12 21:21:52,017 : INFO : 巴罗萨, 词向量为：-0.0041553574, 相似度为：0.1991862
115. 2024-10-12 21:21:52,017 : INFO : 伊斯兰, 词向量为：6.863892e-05, 相似度为：0.19687037
116. 2024-10-12 21:21:52,017 : INFO : 美国, 词向量为：0.0048495266, 相似度为：0.1935192
117. 2024-10-12 21:21:52,017 : INFO : 敌视, 词向量为：-0.00053562626, 相似度为：0.19234061
118. 进程已结束，退出代码为 0

### 5.3 Skip-Gram模型结果验证与类比实验

Skip-Gram模型训练结果：

1. D:\ProgramData\anaconda3\envs\NLP\_exp\python.exe D:\Files\OneDrive\Codes\PyCharmProjects\NLP\_exp\_1\test\_skip\_gram.py
2. 2024-10-12 21:20:39,306 : INFO : 读取本地模型embedding\_skip\_gram\_80w.pkl完成...
3. 2024-10-12 21:20:50,226 : INFO : 当前词的索引为：1438，词为：电脑, 词向量均值为：-0.0021989995, 最相近的前十个词为：
4. 电脑, 词向量均值为：-0.0021989995, 相似度为：1.0
5. 系统, 词向量均值为：-0.0026025374, 相似度为：0.7102532
6. 使用, 词向量均值为：-0.0030781194, 相似度为：0.6875983
7. 游戏, 词向量均值为：-0.0044427267, 相似度为：0.68595976
8. 技术, 词向量均值为：-4.1563384e-05, 相似度为：0.6846942
9. 数位, 词向量均值为：-0.0008437142, 相似度为：0.6812366
10. 提供, 词向量均值为：-0.0015751236, 相似度为：0.67405844
11. 设备, 词向量均值为：-0.0041751657, 相似度为：0.6711073
12. 开发, 词向量均值为：-0.0010847494, 相似度为：0.67097783
13. 电子, 词向量均值为：-0.0029326433, 相似度为：0.6705973
14. 功能, 词向量均值为：-0.0022916722, 相似度为：0.6662872
15. 2024-10-12 21:21:00,512 : INFO : 当前词的索引为：1668，词为：书籍, 词向量均值为：-0.0057701273, 最相近的前十个词为：
16. 书籍, 词向量均值为：-0.0057701273, 相似度为：1.0
17. 出版社, 词向量均值为：-0.0015778303, 相似度为：0.74536854
18. 出版, 词向量均值为：-0.00092858035, 相似度为：0.73476493
19. 著作, 词向量均值为：-0.0011161483, 相似度为：0.71203136
20. 小说, 词向量均值为：0.0010071212, 相似度为：0.70676047
21. 书, 词向量均值为：-0.0041834596, 相似度为：0.7037387
22. 作者, 词向量均值为：-0.0021234227, 相似度为：0.67562675
23. 写作, 词向量均值为：-0.0017604639, 相似度为：0.6703666
24. 描述, 词向量均值为：-0.0033361162, 相似度为：0.67024565
25. 文字, 词向量均值为：-0.003033033, 相似度为：0.66485775
26. 一本, 词向量均值为：-0.0025216746, 相似度为：0.6637925
27. 2024-10-12 21:21:10,911 : INFO : 当前词的索引为：103，词为：公里, 词向量均值为：0.0005090916, 最相近的前十个词为：
28. 公里, 词向量均值为：0.0005090916, 相似度为：1.0000001
29. 约, 词向量均值为：-0.0016885281, 相似度为：0.73434615
30. 距离, 词向量均值为：-0.0028089874, 相似度为：0.72414553
31. 位于, 词向量均值为：-0.0014083652, 相似度为：0.71222055
32. 距, 词向量均值为：-0.0011293874, 相似度为：0.7047797
33. 小时, 词向量均值为：-0.0041997703, 相似度为：0.7043874
34. 达, 词向量均值为：-0.004814368, 相似度为：0.6900847
35. 附近, 词向量均值为：0.0010370122, 相似度为：0.68685234
36. 千米, 词向量均值为：-0.0037396497, 相似度为：0.6797802
37. 全长, 词向量均值为：-0.00057486026, 相似度为：0.67504716
38. 英里, 词向量均值为：-0.0013346535, 相似度为：0.67455125
39. 2024-10-12 21:21:21,212 : INFO : 当前词的索引为：65，词为：还, 词向量均值为：-0.0027177008, 最相近的前十个词为：
40. 还, 词向量均值为：-0.0027177008, 相似度为：0.9999999
41. 时, 词向量均值为：-0.0010907032, 相似度为：0.8364912
42. 都, 词向量均值为：-0.0049164477, 相似度为：0.83030534
43. 没有, 词向量均值为：-0.0037912424, 相似度为：0.82623875
44. 更, 词向量均值为：-0.0006193883, 相似度为：0.825203
45. 不, 词向量均值为：-0.0035975394, 相似度为：0.8235947
46. 上, 词向量均值为：-0.0026225033, 相似度为：0.82358205
47. 中, 词向量均值为：-0.0031690155, 相似度为：0.8133216
48. 亦, 词向量均值为：-0.0035864157, 相似度为：0.8106366
49. 得到, 词向量均值为：-0.0005069693, 相似度为：0.8092233
50. 之后, 词向量均值为：-0.00066590705, 相似度为：0.8021041
51. 2024-10-12 21:21:21,212 : INFO :
52. 2024-10-12 21:21:21,212 : INFO : 父亲 的词向量均值为：-0.00014007952, 索引为：225
53. 2024-10-12 21:21:21,212 : INFO : 男人 词向量均值为：-0.0029882386, 索引为：1499
54. 2024-10-12 21:21:21,212 : INFO : 女人 词向量均值为：-0.0017167834, 索引为：1225
55. 2024-10-12 21:21:31,291 : INFO : 父亲-男人+女人 之间的类比实验，最相近的前十个词为:
56. 2024-10-12 21:21:31,291 : INFO : 父亲, 词向量均值为：-0.00014007952, 相似度为：0.76335853
57. 2024-10-12 21:21:31,291 : INFO : 女人, 词向量均值为：-0.0017167834, 相似度为：0.6608846
58. 2024-10-12 21:21:31,291 : INFO : 母亲, 词向量均值为：-0.0002011414, 相似度为：0.6311818
59. 2024-10-12 21:21:31,291 : INFO : 父母, 词向量均值为：-0.00021188574, 相似度为：0.5741208
60. 2024-10-12 21:21:31,291 : INFO : 女儿, 词向量均值为：-0.00095231127, 相似度为：0.5714824
61. 2024-10-12 21:21:31,291 : INFO : 出生, 词向量均值为：0.00019837354, 相似度为：0.57037747
62. 2024-10-12 21:21:31,291 : INFO : 弟弟, 词向量均值为：-0.00086725166, 相似度为：0.5639205
63. 2024-10-12 21:21:31,291 : INFO : 兄长, 词向量均值为：-0.00011869754, 相似度为：0.5591598
64. 2024-10-12 21:21:31,291 : INFO : 哥哥, 词向量均值为：-0.0030214288, 相似度为：0.55151737
65. 2024-10-12 21:21:31,291 : INFO : 妻子, 词向量均值为：0.0035279607, 相似度为：0.54979616
66. 2024-10-12 21:21:31,291 : INFO : 家中, 词向量均值为：-0.0005679878, 相似度为：0.54789877
67. 2024-10-12 21:21:31,291 : INFO :
68. 2024-10-12 21:21:31,291 : INFO : 国王 的词向量均值为：0.00063817843, 索引为：490
69. 2024-10-12 21:21:31,291 : INFO : 男人 词向量均值为：-0.0029882386, 索引为：1499
70. 2024-10-12 21:21:31,291 : INFO : 女人 词向量均值为：-0.0017167834, 索引为：1225
71. 2024-10-12 21:21:31,291 : INFO : 女王 词向量均值为：-0.0017167834, 索引为：1650
72. 2024-10-12 21:21:41,543 : INFO : 国王-男人+女人 之间的类比实验，最相近的前十个词为:
73. 2024-10-12 21:21:41,543 : INFO : 国王, 词向量均值为：0.00063817843, 相似度为：0.7692371
74. 2024-10-12 21:21:41,543 : INFO : 女人, 词向量均值为：-0.0017167834, 相似度为：0.5963687
75. 2024-10-12 21:21:41,543 : INFO : 三世, 词向量均值为：0.00014838789, 相似度为：0.54159576
76. 2024-10-12 21:21:41,543 : INFO : 二世, 词向量均值为：-0.0011258753, 相似度为：0.5311095
77. 2024-10-12 21:21:41,543 : INFO : 威廉, 词向量均值为：0.0016174868, 相似度为：0.5220322
78. 2024-10-12 21:21:41,543 : INFO : 教宗, 词向量均值为：0.0010263507, 相似度为：0.52091706
79. 2024-10-12 21:21:41,543 : INFO : 瑞典, 词向量均值为：-1.2835434e-05, 相似度为：0.51141906
80. 2024-10-12 21:21:41,543 : INFO : 斯, 词向量均值为：-0.00023690189, 相似度为：0.5044966
81. 2024-10-12 21:21:41,543 : INFO : 一世, 词向量均值为：0.001294487, 相似度为：0.50297356
82. 2024-10-12 21:21:41,543 : INFO : 之子, 词向量均值为：-0.0011262777, 相似度为：0.50112206
83. 2024-10-12 21:21:41,543 : INFO : 十四, 词向量均值为：0.0014167309, 相似度为：0.49227214
84. 2024-10-12 21:21:41,543 : INFO :
85. 2024-10-12 21:21:41,543 : INFO : 乒乓球拍 的词向量均值为：-0.00087083434, 索引为：569656
86. 2024-10-12 21:21:41,543 : INFO : 乒乓球 词向量均值为：-0.0019211336, 索引为：6196
87. 2024-10-12 21:21:41,543 : INFO : 羽毛球拍 词向量均值为：-0.0011700729, 索引为：475174
88. 2024-10-12 21:21:41,543 : INFO : 羽毛球拍 词向量均值为：-0.0025415819, 索引为：384
89. 2024-10-12 21:21:51,744 : INFO : 乒乓球拍-乒乓球+羽毛球拍 之间的类比实验，最相近的前十个词为:
90. 2024-10-12 21:21:51,745 : INFO : 羽毛球, 词向量均值为：-0.0025415819, 相似度为：0.4504831
91. 2024-10-12 21:21:51,745 : INFO : 大岛, 词向量均值为：-0.0015852838, 相似度为：0.38347578
92. 2024-10-12 21:21:51,745 : INFO : 加冷, 词向量均值为：-0.0010758436, 相似度为：0.38329715
93. 2024-10-12 21:21:51,745 : INFO : 坏蛋, 词向量均值为：-0.0014784639, 相似度为：0.3782618
94. 2024-10-12 21:21:51,745 : INFO : 全津, 词向量均值为：0.0027513085, 相似度为：0.37705964
95. 2024-10-12 21:21:51,745 : INFO : 海棠, 词向量均值为：6.6745283e-06, 相似度为：0.3726871
96. 2024-10-12 21:21:51,745 : INFO : 支厅, 词向量均值为：-0.0013647513, 相似度为：0.37101766
97. 2024-10-12 21:21:51,745 : INFO : 左耳, 词向量均值为：-0.0021096813, 相似度为：0.37017915
98. 2024-10-12 21:21:51,745 : INFO : 查理五世, 词向量均值为：0.0006472431, 相似度为：0.365286
99. 2024-10-12 21:21:51,745 : INFO : 髹, 词向量均值为：-0.0003573646, 相似度为：0.36256403
100. 2024-10-12 21:21:51,745 : INFO : 读作, 词向量均值为：-0.0026518262, 相似度为：0.35804343
101. 2024-10-12 21:21:51,745 : INFO :
102. 2024-10-12 21:21:51,745 : INFO : 英国 的词向量均值为：-0.0012930289, 索引为：75
103. 2024-10-12 21:21:51,745 : INFO : 伦敦 词向量均值为：-0.0016702918, 索引为：753
104. 2024-10-12 21:21:51,745 : INFO : 美国 词向量均值为：0.0012863753, 索引为：13
105. 2024-10-12 21:21:51,745 : INFO : 华盛顿 词向量均值为：-0.0013833777, 索引为：1926
106. 2024-10-12 21:22:01,433 : INFO : 英国-伦敦+美国 之间的类比实验，最相近的前十个词为:
107. 2024-10-12 21:22:01,433 : INFO : 华盛顿, 词向量均值为：-0.0013833777, 相似度为：0.784351
108. 2024-10-12 21:22:01,433 : INFO : 英国, 词向量均值为：-0.0012930289, 相似度为：0.67100793
109. 2024-10-12 21:22:01,433 : INFO : 美国, 词向量均值为：0.0012863753, 相似度为：0.59554976
110. 2024-10-12 21:22:01,433 : INFO : 第一, 词向量均值为：-0.0024473292, 相似度为：0.53430593
111. 2024-10-12 21:22:01,433 : INFO : 年, 词向量均值为：-0.0002158796, 相似度为：0.52131855
112. 2024-10-12 21:22:01,433 : INFO : 第五, 词向量均值为：-0.00017589531, 相似度为：0.50604415
113. 2024-10-12 21:22:01,433 : INFO : 成为, 词向量均值为：-0.0022449684, 相似度为：0.5043506
114. 2024-10-12 21:22:01,433 : INFO : 首位, 词向量均值为：-0.002357846, 相似度为：0.5001466
115. 2024-10-12 21:22:01,433 : INFO : 人, 词向量均值为：-0.004396792, 相似度为：0.49803913
116. 2024-10-12 21:22:01,433 : INFO : 成功, 词向量均值为：0.001319847, 相似度为：0.49583063
117. 2024-10-12 21:22:01,433 : INFO : 当时, 词向量均值为：-0.0035531, 相似度为：0.49354762
118. 进程已结束，退出代码为 0

### 5.4 相似度计算以及类比实验结果分析

Skip-Gram模型的结果比CBOW模型的结果稍微好一些，这可能得益于Skip-Gram在大语料库更优性能的表现，也可能因为训练轮次的显著增加，导致模型学习的效果更好。

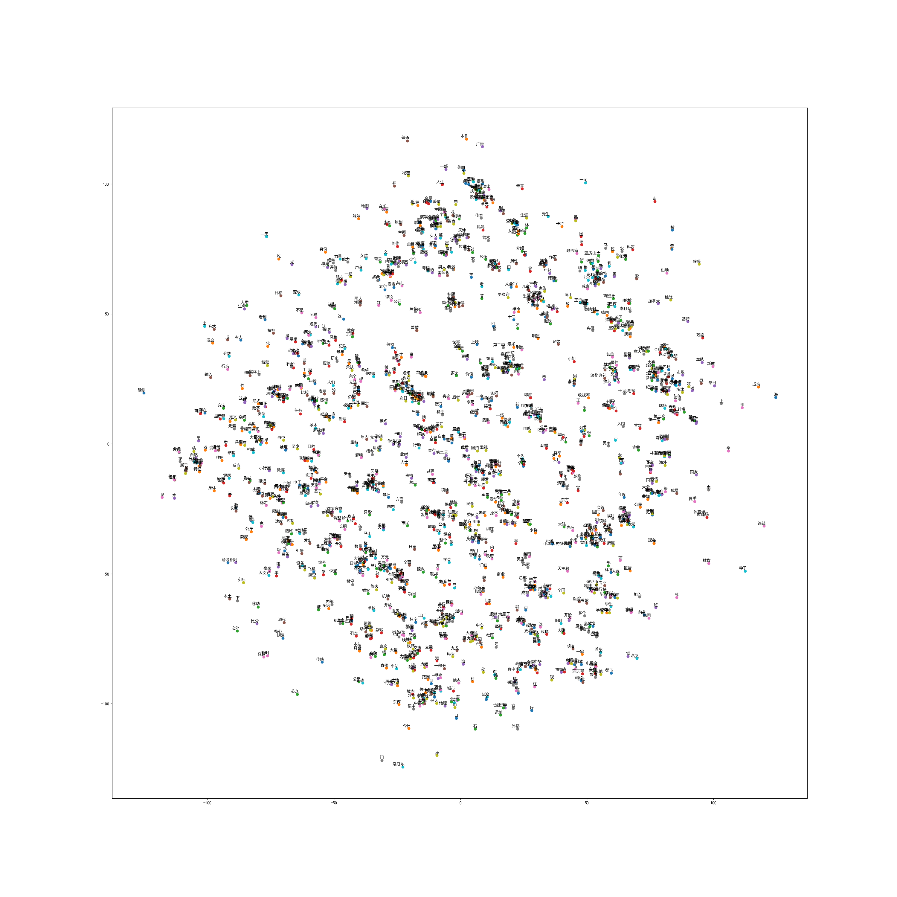
从实验的结果来看，经过训练的词向量还是可以显示出一定程度的相关性，尤其是书籍、电脑、公里这种名词，相关性学习的比较好。但是对于还这种可能是副词或动词也就是具有多种词性的词，学习的效果还是不太让人满意。

这样的结果其实也是很容易理解的，因为名词的上下文一致性较强，它们在不同上下文中的含义变化较小。这种语义上的稳定性使得模型更容易捕捉到名词的共现关系。例如，书籍可能与“小说”、“单行本”等词语频繁共现，电脑可能与“技术”、“应用”等词共现。因此，经过训练后，这类名词的词向量会显示出较好的相关性。而对于像“还”这样的词是典型的多义词，其词义依赖于上下文。它既可以是副词，表示“仍然”或“另外”，也可以是动词，表示“归还”或“偿还”。这种多义性使得模型更难以为“还”学习到一个统一的词向量，因为它的上下文多变，可能出现在不同的语境中，比如“他还在学习”和“他还了钱”，模型可能会将这两种不同的用法混为一谈。而且相比于名词，副词或动词的上下文更为广泛且灵活。例如，副词“还”可以修饰动词、形容词等，出现在非常多样的句子结构中。由于“还”出现在不同的句子中，其上下文变化很大，导致模型在训练时难以找到一致的模式来学习它的表示。

同样的，在类比实验中，对于英国-伦敦+美国的计算结果，能够很好的得到华盛顿这样的结果；但对于国王-男人+女人便没有很好的得到女王这样的结果。

这同样也是解释的通的。“英国-伦敦+美国=华盛顿”的类比成功得益于国家与首都之间明确而高频的共现关系，而“国王-男人+女人=女王”类比失败的原因主要在于：①词汇的多义性和复杂性，特别是性别相关词汇在语料中的偏差；②词向量模型的静态性无法处理复杂的语义和社会关系。

### 5.5 CBOW模型词向量二维可视化

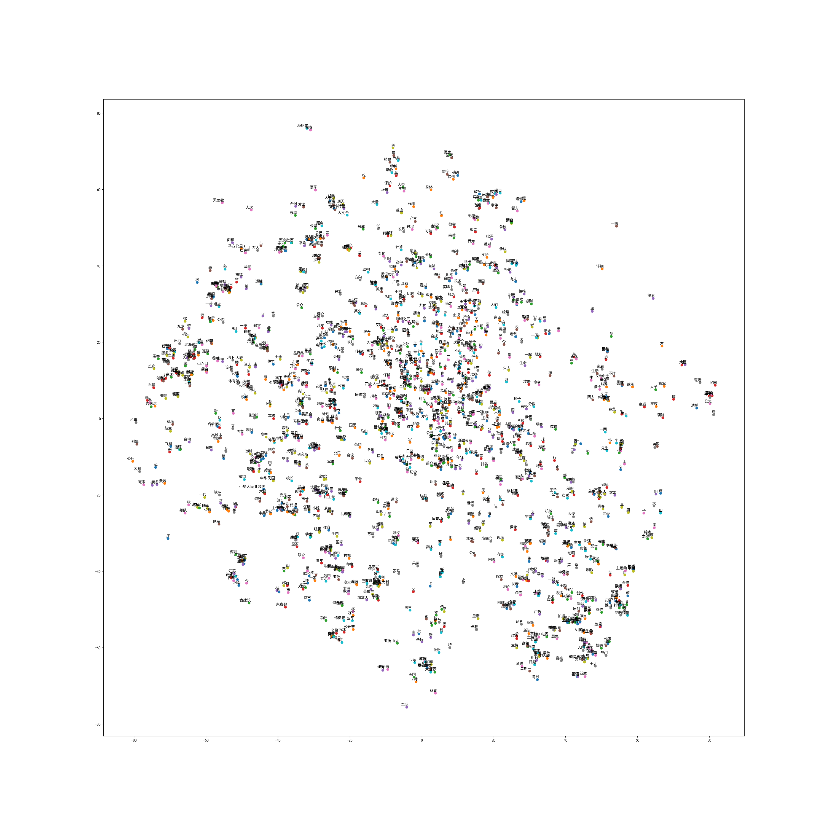
 词表（按词频）的前1500二维可视化：

具体细节：

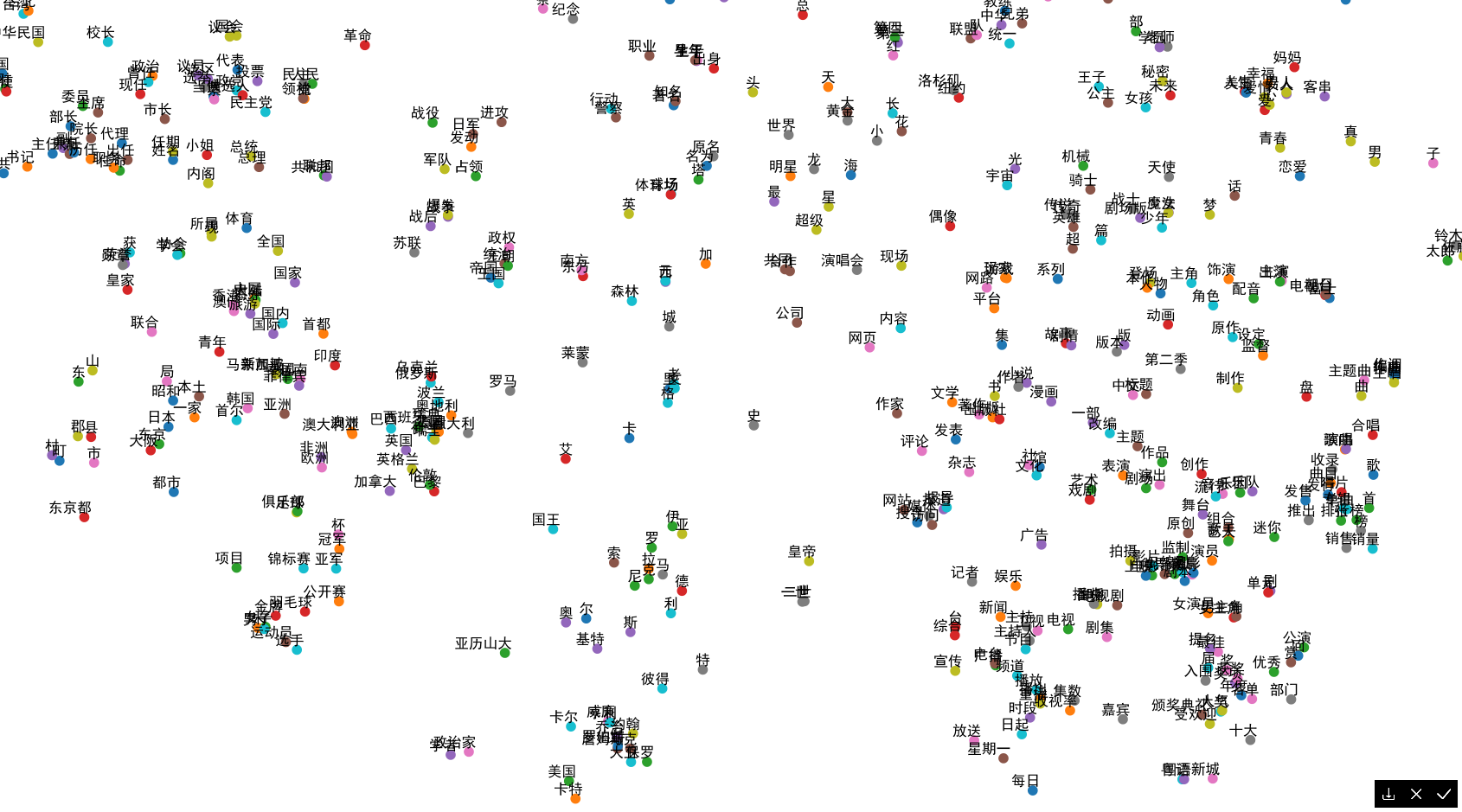
图表, 散点图

描述已自动生成

### 5.6 Skip-Gram模型词向量二维可视化

词表（按词频）的前1500二维可视化：

具体细节：



### 5.7 二维可视化结果分析

从二维可视化的结果来看，两个模型能够体现出词向量在二维空间中的分布存在着一定的聚类现象。某些词聚集在一起，形成了明显的团簇如各个国家、体育赛事名词、娱乐电视相关名词。这代表这些词在高维空间中也具有相似性，可能在语义或词性上相近，很明显这是在逻辑上是解释的通的，而且这种现象也是词向量的训练目标。

此外，部分词分布在图的外围或相对孤立，往往是一些特殊的词如洛厄尔，或者是语料中出现较少的词。而且由于只绘制了1500词，这些词里它们的词向量可能没有很强的语义相似性，因此在降维后与其他词的距离较远。

实验的过程中设计了两个embedding矩阵，其中第二个的效果理论上会好一些，但是从二维展示的结果来看，还是认为第一个的embedding效果要更好。

## 实验总结

本次实验过程总体算是一波三折。最初我使用的是PyCharm的venv虚拟环境，但是遇到opencc库如何都下载不了的情况。于是转而使用Anaconda作为Python环境的管理器，在相应的conda虚拟环境中成功完成实验的配置。

其次，一开始没有理解老师说不允许调包的含义，以为只是不给用Word2Vec()这样的函数，于是使用PyTorch实现，调用nn.Embedding()等方法，后面得知不可以调这些包，因此学习了各种资料以及源码查看该方法的代码实现，但因无法找到不调包的解决方案，从而放弃使用PyTorch转而使用TensorFlow。在学习TensorFlow的过程中，找到了黄文坚老师的《TensorFlow实战》，而里面刚好有Skip-Gram模型的实现代码，于是便一边学习一边进行编码。

但是在模型的训练时，虽然是同样的参数，但是这些参数对于我的中文词向量的训练来讲，结果是很差的。于是在网上又开始寻找相关的资料：B站上找到了实现讲解的视频、Github找到了更多使用TensorFlow实现Word2Vec的源码。于是参考这些内容对我的模型进行优化。由于是第一次实现机器学习模型，代码我几乎是完全不明白的，看到一行代码就需要写一行注释，导致注释内容甚至超过了代码的内容，在代码的理解上花了很多的时间。此外对于我最开始进行预处理的步骤也进行了优化，如我去除了相对来讲格式较为混乱的引文。但现在考虑的话，其实应该解压对应的bz2文件，读取原始的xml数据，只读取对于标签内的正文得到的语料可能更加干净一些。

本次实验模型参数的调整是漫长的，花了数天时间。从128维的词嵌入矩阵到256维、到300维、到500维、到400维最后到350维；从窗口大小3到1到2到4、从高频词词表大小5W到10W到20W到100W到50W到80W；训练轮次也从1W到5W到20W，Skip-Gram模型的训练轮次最终定为100W。虽然只是尝试了这些参数的不同配置，也是耗尽付费购买的100个计算单元，于是再次购买100个计算单元，选择了较为理想的也是本次实验所使用的参数配置进行训练，得到还算让人满意的结果。

最后验证的环节也出现了问题，Colab上验证的结果还是很不错的，但是我使用本地保存的pkl文件进行验证，结果反而是十分差的。经过一系列尝试才发现原因：①出版本的代码并没有保存我最终的350维的词嵌入向量，只是保存了用于二维展示的降维后的向量，在降维的过程中，损失了众多信息，导致一些不相关的词也显示出了较强的相关性，因此修改了代码保存了最终的词嵌入矩阵。②最初保存的点的数量只有400个，二维展示的结果不是很理想，于是增加了保存点的数量到1W个，同时绘制二维可视化的时候将点的数量也增加到了1500个，这样绘制出来的结果可以呈现出较为理想的聚类效果。在绘制二维图的时候也遇到了中文只能显示成一个方框的问题，这里需要指定一下字体便可以正常显示中文。

总的来讲，本次实验横跨整个国庆假期，花了很多时间查询资料、编辑代码与理解代码，同时也花费了一定金钱辅助实验的进行。但是在这段实验的过程中，对于完成整个机器学习任务有了更进一步的理解：从实验数据的预处理到实验数据的整理、神经网络模型的搭建、神经网络的训练、模型参数的调整都得到了进一步的训练。整个实验过程收获很大。

## 实验分工

模型设计、调优；实验报告、汇报PPT均由我一人独立完成。