《自然语言处理》实验报告

基于 PyTorch 实现 Word2Vec 词嵌入

班 级：大数据 1802

学 号：2018317230228

姓 名：刘玉莎

2020 年 11 月

实验内容： 基于 PyTorch 实现 Word2Vec 词嵌入

依据课堂讲授和实验指导书内容，完成词嵌入的代码实现。

实验要求：

基本要求：能使用 PyTorch 完成计算 Word2Vec 神经网络训练模型搭建，能按照要求对指定语料库进行训练。

进阶要求：参照其他词嵌入 package，对指定语料库中词语的嵌入质量进行比较。

数据地址：

https://hzaubionlp.com/nlp4underg/

实验模型介绍：

嵌入（embedding）是机器学习中最迷人的想法之一，嵌入技术用于神经网络模型已有相当大的发展，我们平时使用过的 Siri，Google 翻译，以及智能手机键盘进行下一个词的预测等等，都是在从自然语言处理模型的核心的想法中获益。 Word2Vec 是一种有效创建词嵌入的方法，它是从大量文本预料中以无监督方式学习语义知识的模型，这个模型为浅层双层的神经网络，用来训练以重新建构语言学之词文本。

在自然语言处理里面，最小的单位量是词语，词语组成句子，句子组成段落，段落在组成一篇文章，所以在处理自然语言的问题时，首先要对最基本的词语进行处理。词语是一个抽象的符号，所以需要把他们转换为数值形式，即可以理解为将词语嵌入到一 个数学空间里面，这样的嵌入方式就叫做词嵌入（word embedding）。 Word2Vec，是词嵌入的一种，主要思想是把一个词语转化成对应向量的表达形式，以便于让机器读取数据。所以 Word2Vec 是一组用于生成单词嵌入的相关模型，这些模型是浅层的两层神经网络，采用的是 n 元语法模型（n-gram model），即假设一个词只与周围 n 个词有关，而与文本中的其他词无关，经过训练可以重建单词的语言环境。 Word2Vec 是轻量级的神经网络，其模型仅仅包括输入层，隐藏层和输出层，模型框架根据输入输出的不同，可以分为 CBOW 模型和 skip-gram 模型，CBOW 模型是 通过上下文的内容预测中心词的可能情况，而 skip-gram 模型与其相反，它是通过中心词预测上下文词的，这两个模型在下文将会详细介绍。Word2vec 将大量文本语料库作

为输入，并产生向量空间，通常具有数百个维度，语料库中的每个唯一词都在空间中分配了相应的向量。词向量位于向量空间中，以便在语料库中共享公共上下文的词在空间中彼此靠近。

Word2Vec 词嵌入的模型介绍

1、单词的表示方法类型

①、one hot 编码（独热编码） one hot 编码就是用一个向量来表示一个词，向量的长度为词典的大小，向量 中只有一个 1 ，其他全为 0，1 的位置对应该词在词典中的位置。例如：I like NLP very much，转换成 one hot 编码就是：“I”：[1，0，0，0，0]，“like”：[0，1， 0，0，0]，“NLP”：[0，0，1，0，0]，“very”：[0，0，0，1，0]，“much”：[0， 0，0，0，1]，每个单词占一个维度。 这个编码方式不需要繁琐的计算，简单易懂，但是缺点也非常明显，比如：1、 任意两个词之间都是孤立的，因为表示这两个词的向量是正交的，根本无法表示出 在语义层面上词语之间的相关信息，这一点是致命的缺陷。2、这个编码方式会导 致矩阵非常稀疏，当语料库达到十万或者百万级别时，向量的维数太大导致内存灾难。因此介绍另外一种词的表示方式，可以把词向量的维度变小。

②、词的分布式表示（distributed representation） 词的分布式表示可以解决 one hot 编码的一系列缺陷，它的思路是通过训练， 将每个词都映射到一个较短的词向量上面来，所有的这些词就构成了一个词向量空 间，每一个向量可以视为该空间上的一个点，此时向量长度可以自由选择，与词典 规模无关，这是非常大的优势，进而可以用普通的统计学的方法来研究词与词之间 的关系。这个较短的词向量维度一般需要我们在训练时自己来指定。 还是以之前的例子来说明，现在要将“like”这个词从一个可能非常稀疏的向量 所在的空间，映射到一个更加低维的空间中，必须满足两个性质：1）、这个映射是 单射。2）、映射之后的向量不会丢失之前的那种向量所含的信息。将高维词向量嵌 入到一个低维空间的这个过程称为词嵌入。经过一系列的降维操作，就得到了用分布式表示（distributed representation） 的较短的词向量，就可以比较容易的分析词之间的关系了，假如降维到了 2 维，用词向量表示词的时候可能会出现以下结果： $\overrightarrow{NLP} + \overrightarrow{very} + \vec{I} - \overrightarrow{much} = \overrightarrow{like}$

2、CBOW（Continuous Bags of Words） 这个模型的思想是给定上下文单词的情况下来预测当前单词，要预测的当前单 词为 y，在输入层输入经过 one hot 编码过的上下文单词，再通过一个隐藏层进行 操作后输出，输出后通过函数 softmax，可以计算出每个单词的生成概率。

3、skip-gram 这个模型的思想是通过当前单词预测上下文单词，当前单词为 x，要预测的其 他单词是 y，同样也是通过一个隐藏层输入后再用 Softmax 函数来预测其它词的 概率。

本次实验选择利用 PyTorch 的功能实现 Word2Vec 的词嵌入，达到预测上下文词的目的。

实验程序代码：

1、运行代码 Bert_4_WordEmbedding.py，得到运行结果。

```python
from transformers import BertTokenizer, BertModel

# 1. Load model.
model_name = 'bert-base-uncased'
model = BertModel.from_pretrained(model_name)
tokenizer = BertTokenizer.from_pretrained(model_name)

# 2. Data preprocessing.
token = 'Harry'
token_input = tokenizer(token, return_tensors='pt')
token_decode = tokenizer.decode(token_input['input_ids'][0])
# 3. Calculate word embedding
model.eval()
token_embedding, _ = model(**token_input)
print(token_embedding.shape)

token_embedding = token_embedding.squeeze(0)
print(token_embedding.shape)

# without_speical_token
token_embedding = token_embedding[1, :]
print(token_embedding.shape)
```

```
C:\ProgramData\Anaconda3\python.exe D:/D3S/自然语言处理/实验/sy6/Tutorial_4_word2vec-main/Bert_4_WordEmbedding.py
Downloading: 100%|██████████| 433/433 [00:00<00:00, 256kB/s]
Downloading: 100%|██████████| 440M/440M [03:15<00:00, 2.25MB/s]
Downloading: 100%|██████████| 232k/232k [00:01<00:00, 150kB/s]
torch.Size([1, 3, 768])
torch.Size([3, 768])
torch.Size([768])
0.0815548524260521 -0.4707965552806854 0.06033928692340851 -0.24240435659885406 0.28813374042510986 0.2684640884399414 0.2642417550086975 0.2

Process finished with exit code 0
```

2、运行代码 Gensim_4_WordEmbedding.py，得到运行结果。

```
14410 100
the -0.3386694 -0.025847288 -0.15671359 0.15840124 -0.29532057 -0.23414786 -0.368354 -0.(
of -0.37827346 -0.017863726 -0.22813812 0.4256354 -0.038207263 -0.2104737 -0.013431226 0.
and -0.24893594 0.022421708 -0.07343479 0.19510756 0.052314036 -0.28027117 -0.16232325 -(
in -0.19232817 -0.40343913 0.35795537 0.39775774 -0.021147035 -0.07922451 -0.41706818 -0.
nbr -0.13465062 -0.02398632 -0.57653254 -0.262392 0.4592734 -0.13403545 0.43764484 0.365
rice -0.16230969 -0.04592122 -0.24000284 -0.028402433 0.19154368 -0.6811177 -0.24632183 -
a -0.18069294 -0.51771456 0.009694714 0.4279158 -0.22799323 -0.31922942 -0.13583684 0.16
to -0.40009177 0.15013747 0.14680122 0.35920876 -0.036952123 -0.41483784 -0.1401947 0.36
that -0.19549097 -0.032555506 0.38990918 0.040816236 -0.3754258 -0.68607014 -0.18241796 (
is -0.21698232 -0.3401164 0.3826472 -0.19849268 -0.4301931 -0.33188102 -0.2825801 0.1780
for -0.41218218 -0.21707517 -0.4990902 0.12921432 -0.057528205 -0.11032261 0.13755181 -0.
with -0.34736234 0.37449303 -0.017970877 0.31205812 0.44518575 0.15037537 -0.20124395 0.4
by 0.5256826 0.12579744 -0.031613342 0.11741891 -0.12177488 -0.47692674 -0.48197317 0.032
plant -0.64099723 0.30967426 0.03610428 0.46328768 0.49986663 -0.61545426 -0.083651155 0.
was 0.24201344 -0.037960533 -0.025979135 0.16593416 0.07438486 0.044404637 -0.1531373 -0.
gene -0.0049542016 0.08902293 -0.08246116 -0.29605243 -0.53279424 -0.7441946 -0.14205705
```

更换数据集，运用 Gensim 处理 Harry potter 文本，得到 embedding

```python
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence

# 1. data path
corpus_file = './Harry Potter 1.txt'
model_save_file = './hp_Gensim.model.bin'
embedding_save_file = './hp_Gensim.embedding.txt'

# 2. Define hyper-parameters
size = 100
window = 5
min_count = 5
sg = 1
alpha = 0.02
epochs = 2
batch_words = 10000

# 3. Training model
model = Word2Vec(LineSentence(corpus_file), size=size,
window=window,min_count=min_count, sg=sg, alpha=alpha,
iter=epochs,batch_words=batch_words)

# 4. Save Model and Embedding.
model.save(model_save_file)
model.wv.save_word2vec_format(embedding_save_file,
binary=False)
```

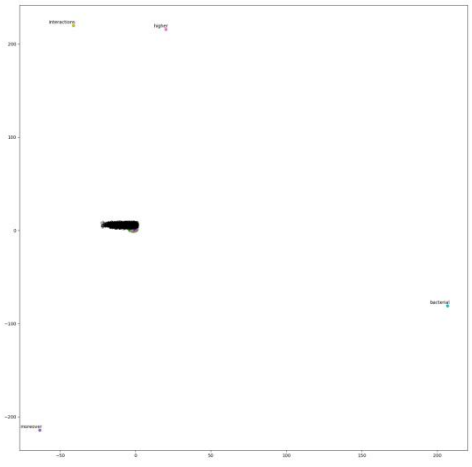
```
1975 100
the 0.04358704 -0.28804836 -0.16924095 -0.01352447 0.10736811 -0.071771756 -0.19178301 -0.19772577 0.1055984
to 0.022046598 -0.2954999 -0.17185034 -0.019857638 0.11575906 -0.08041247 -0.18102051 -0.20036937 0.11333957
and 0.04284418 -0.28312117 -0.1726792 -0.009277033 0.10690824 -0.07874148 -0.19360785 -0.19520456 0.10575507
a 0.03783274 -0.29166922 -0.1797364 -0.011280786 0.115568526 -0.08211441 -0.19017795 -0.19845359 0.11331744
of 0.0440077 -0.28596115 -0.16884303 -0.014740006 0.113318175 -0.08203987 -0.19101189 -0.19190998 0.10533615
was 0.035388023 -0.28284475 -0.17036359 -0.021854322 0.10777735 -0.07773671 -0.18462366 -0.191722 0.11025545
he 0.029381325 -0.285455 -0.17205657 -0.02012234 0.119734004 -0.0740159 -0.18021813 -0.19199893 0.11470174 -
Harry 0.034290135 -0.2875209 -0.16722669 -0.017000614 0.1141461 -0.075786404 -0.1889151 -0.19262752 0.112936
in 0.036218867 -0.28509983 -0.16727749 -0.014196918 0.11331029 -0.07972377 -0.18710485 -0.19551812 0.1153861
his 0.031505425 -0.27723688 -0.17050196 -0.008437586 0.116127014 -0.08005956 -0.17877771 -0.19377269 0.11235
had 0.03236339 -0.2816554 -0.17584823 -0.011452732 0.10798395 -0.07996137 -0.18185075 -0.1911585 0.11581372
-- 0.0251748 -0.29034802 -0.17751645 -0.012728404 0.113350525 -0.08160514 -0.1792831 -0.18750443 0.12214002
said 0.026636373 -0.2838007 -0.1734071 -0.023983743 0.119238764 -0.08839078 -0.17066842 -0.19349822 0.112071
at 0.029818809 -0.2870128 -0.16639915 -0.015451901 0.117814764 -0.08015376 -0.19036654 -0.20182651 0.1123517
you 0.014328372 -0.29562 -0.17462645 -0.015862582 0.11964747 -0.08999629 -0.17181642 -0.19007412 0.11686971
it 0.0347397 -0.2932067 -0.17370254 -0.01791184 0.11420178 -0.07537107 -0.18130633 -0.19246899 0.111854024 -
on 0.033781122 -0.28338203 -0.16438244 -0.017704174 0.10834542 -0.07691817 -0.18351687 -0.1951696 0.11389809
that 0.028849915 -0.28248852 -0.1705435 -0.020245938 0.11588402 -0.07825697 -0.18078765 -0.19422963 0.113697
I 0.018398335 -0.29834905 -0.18313387 -0.022174207 0.124966264 -0.09127677 -0.17522912 -0.19386226 0.1204250
as 0.03757158 -0.28897047 -0.17644459 -0.010013366 0.11080696 -0.08846836 -0.18262643 -0.19053973 0.11120115
He 0.03110941 -0.28703195 -0.17312112 -0.012405178 0.1176532 -0.08226536 -0.18288004 -0.19292118 0.114354014
they 0.033972513 -0.3008511 -0.17666529 -0.013908487 0.11591698 -0.08063448 -0.1911161 -0.19494918 0.1219319
with 0.037879277 -0.29330507 -0.16914697 -0.018778369 0.11359806 -0.075672686 -0.19128892 -0.20088473 0.1130
but 0.032888994 -0.28737882 -0.17996721 -0.018844984 0.115291744 -0.08635098 -0.18041946 -0.20046395 0.11240
be 0.023526816 -0.2850859 -0.17827883 -0.022907186 0.121465735 -0.08563859 -0.18027966 -0.19559358 0.1162072
for 0.034174845 -0.28600326 -0.16748358 -0.013895379 0.11669621 -0.07737763 -0.18089189 -0.19489056 0.111880
```

3、运行代码 Skip_Gram_basic.py，得到运行结果。

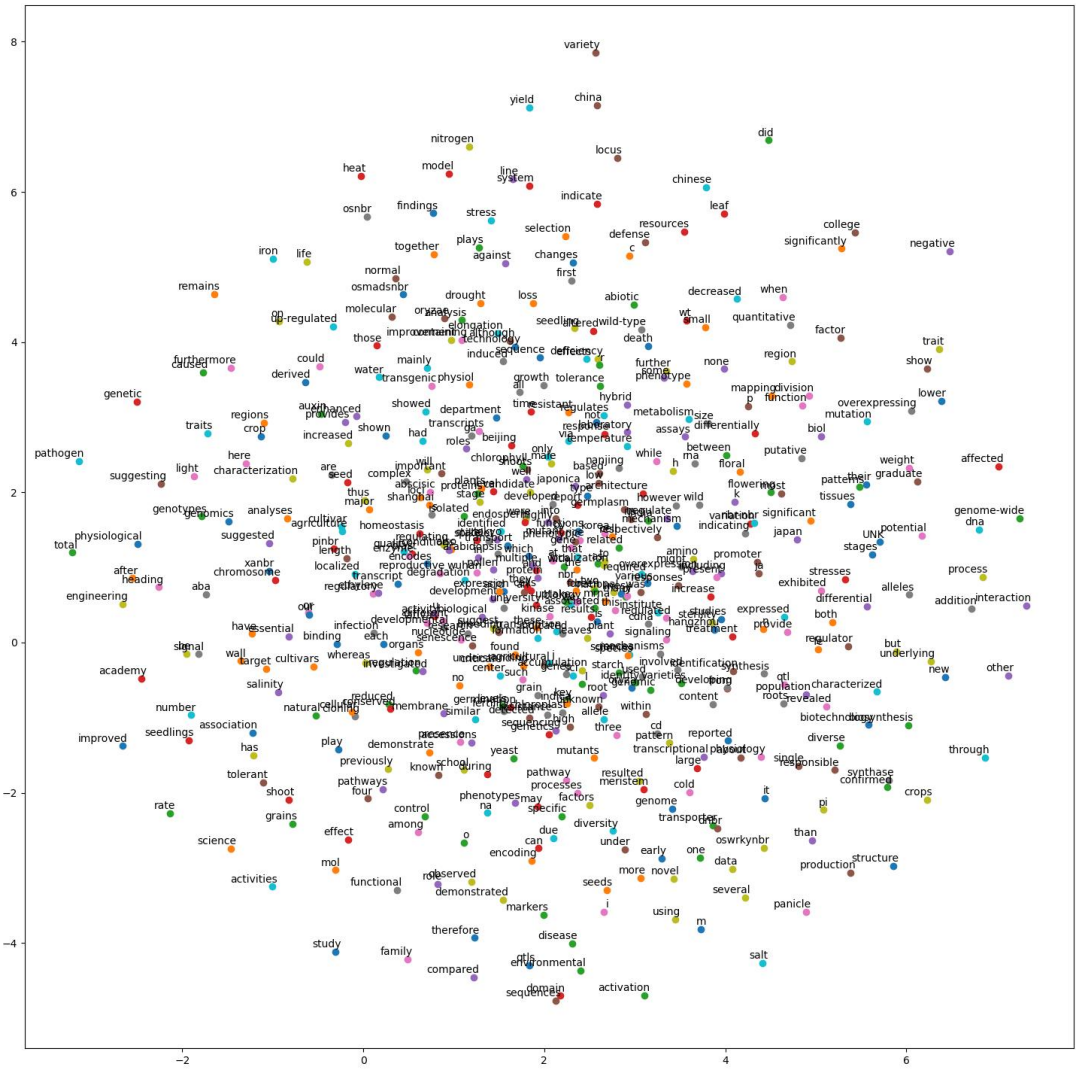
```
In[3]: final_embedding
Out[3]:
tensor([[ 0.2873, -0.0209, -0.6548,  ..., -0.0853, -0.6204,  0.5767],
        [ 0.4566, -0.2044,  0.4576,  ..., -0.2778, -0.0459, -0.7128],
        [ 0.0594,  0.0129,  0.6241,  ...,  0.1348, -0.2597,  0.1643],
        ...,
        [ 1.8498, -0.2568,  0.8975,  ..., -0.1698,  0.3875,  0.4505],
        [-0.4643,  0.2689, -1.5349,  ...,  0.8695, -0.3778,  0.8294],
        [ 1.4144, -0.6491, -0.9073,  ..., -0.4737, -1.4122, -1.2031]])
```

```
SkipGram(
  (embedding): Embedding(40000, 200)
  (output): Linear(in_features=200, out_features=40000, bias=True)
  (log_softmax): LogSoftmax()
)
--------------------------------------------------
Start training.
Average loss as step 20: 10.55, cost: 13.41s.
Average loss as step 40: 10.63, cost: 9.77s.
Average loss as step 60: 10.63, cost: 8.57s.
Average loss as step 80: 10.80, cost: 8.79s.
Training Done.
--------------------------------------------------
torch.Size([40000, 200])
Visualizing.
C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins

TSNE visualization is completed, saved in ./tsne.png.
```



发现上图可视化比较不均匀，不美观，调整参数，从新制图。



更换数据集，运用 skipgram 处理 Harry potter 文本，得到 embedding

```
# Step 1: Data preprocessing.
def str_norm(str_list: list, punc2=' ', num2='NBR', space2=' ', lower=True):
    punctuation = string.punctuation.replace('-', '')
    rep_list = str_list.copy()
    for index, row in enumerate(rep_list):
        row = row.strip()
        row = re.sub("\d+.\d+", num2, row)
        row = re.sub('\d+', num2, row)
        for pun in punctuation:
            row = row.replace(pun, punc2)
        # if lower:
        #     row = row.lower()
        rep_list[index] = re.sub(' +', space2, row)
    return rep_list
```

首先更改数据预处理函数，Harry Potter 文本中有许多人名地名，所以不对其做小写转换处理。

更换数据集，数据大小为 81,152 个词。Most common words (+UNK) [['UNK', 0], ('the', 3310), ('to', 1846), ('and', 1802), ('a', 1578), ('Harry', 1324)]

```
raw_file = './Harry Potter 1.txt'
corpus = Data_Pre(raw_file, './out.txt')
words = read_data((corpus))
print('Data size: {0} words.'.format(format(len(words), ',')))
"""
Data size: 81,152 words.
"""
```

SkipGram(

(embedding): Embedding(40000, 200)

(output): Linear(in_features=200, out_features=40000, bias=True)

(log_softmax): LogSoftmax()

)

```
# Step 3: Function to generate a training batch for the skip-
gram model.
def generate_batch(data, batch_size, num_skips, skip_window):
    global data_index
    assert batch_size % num_skips == 0
    assert num_skips <= 2 * skip_window
    batch = np.ndarray(shape=(batch_size), dtype=np.int32)
    labels = np.ndarray(shape=(batch_size, 1), dtype=np.int32)
    # total window length
    span = 2 * skip_window + 1  # [ skip_window target
skip_window ]
    buffer = collections.deque(maxlen=span)
    # 从 data 开头添加整个窗口长度的 idx
    for _ in range(span):
        buffer.append(data[data_index])
        # 防止 index 溢出
        data_index = (data_index + 1) % len(data)

    for i in range(batch_size // num_skips):
        # center 在这个窗口中的位置
        target = skip_window
        targets_to_avoid = [skip_window]
        #
print('i=',i,'target=',target,'targets_to_avoid=',targets_to_a
```

```
void,'\n')
        for j in range(num_skips):
            # 窗口中采样非 center 的单词
           while target in targets_to_avoid:
                target = random.randint(0, span - 1)
                # print(target)
            targets_to_avoid.append(target)
            # print(target,'\t',targets_to_avoid,'\n')
            # index 为 batch 中第几个数据
           batch[i * num_skips + j] = buffer[skip_window]
            labels[i * num_skips + j, 0] = buffer[target]

        buffer.append(data[data_index])
        data_index = (data_index + 1) % len(data)
    return batch, labels


data_index = 0
batch_size = 16
# left and right target number.
skip_window = 4
# how many target in window.
num_skips = 8

batch, labels = generate_batch(data=data,
batch_size=batch_size,num_skips=num_skips,
skip_window=skip_window)
# Step 4: Build a skip-gram model.
class SkipGram(nn.Module):
    def __init__(self, args):
        super().__init__()

        self.vocabulary_size = args.vocabulary_size
        self.embedding_size = args.embedding_size

        self.embedding = nn.Embedding(self.vocabulary_size,
                                      self.embedding_size)  #
W = vd lookup  [1*v']*[V*embedding_size]  -> [v*
embedding_size]
        self.output = nn.Linear(self.embedding_size,
self.vocabulary_size)  # 输出层
       self.log_softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):  # x 16
```

```
        x = self.embedding(x)  # x  16*128
        x = self.output(x)  # x  16*40000
        log_ps = self.log_softmax(x)  # x 16*40000
        return log_ps


# Step 5: Begin training.
class config():
    def __init__(self):
        self.num_steps = 1000
        self.batch_size = 128
        self.check_step = 20

        self.vocabulary_size = 40000
        self.embedding_size = 200  # Dimension of the
embedding vector.
        self.skip_window = 4  # How many words to consider
left and right.
        self.num_skips = 8  # How many times to reuse an input
to generate a label.

        self.use_cuda = torch.cuda.is_available()

        self.lr = 0.03


args = config()

model = SkipGram(args)
nll_loss = nn.NLLLoss()
adam_optimizer = optim.Adam(model.parameters(), lr=args.lr)

print('-' * 50)
print('Start training.')
average_loss = 0
start_time = time.time()
for step in range(1, args.num_steps):
    batch_inputs, batch_labels = generate_batch(
        data, args.batch_size, args.num_skips,
args.skip_window)
    batch_labels = batch_labels.squeeze()
    batch_inputs, batch_labels =
torch.LongTensor(batch_inputs), torch.LongTensor(batch_labels)
    if args.use_cuda:
```

```
        batch_inputs, batch_labels = batch_inputs.to('cuda'),
batch_labels.to('cuda')
    log_ps = model(batch_inputs)
    loss = nll_loss(log_ps, batch_labels)
    average_loss += loss
    adam_optimizer.zero_grad()
    loss.backward()
    adam_optimizer.step()
    if step % args.check_step == 0:
        end_time = time.time()
        average_loss /= args.check_step
        print('Average loss as step {0}: {1:.2f}, cost:
{2:.2f}s.'.format(step, average_loss, end_time - start_time))
        start_time = time.time()
        average_loss = 0


final_embedding = model.embedding.weight.data
最后得到训练的 embedding
```

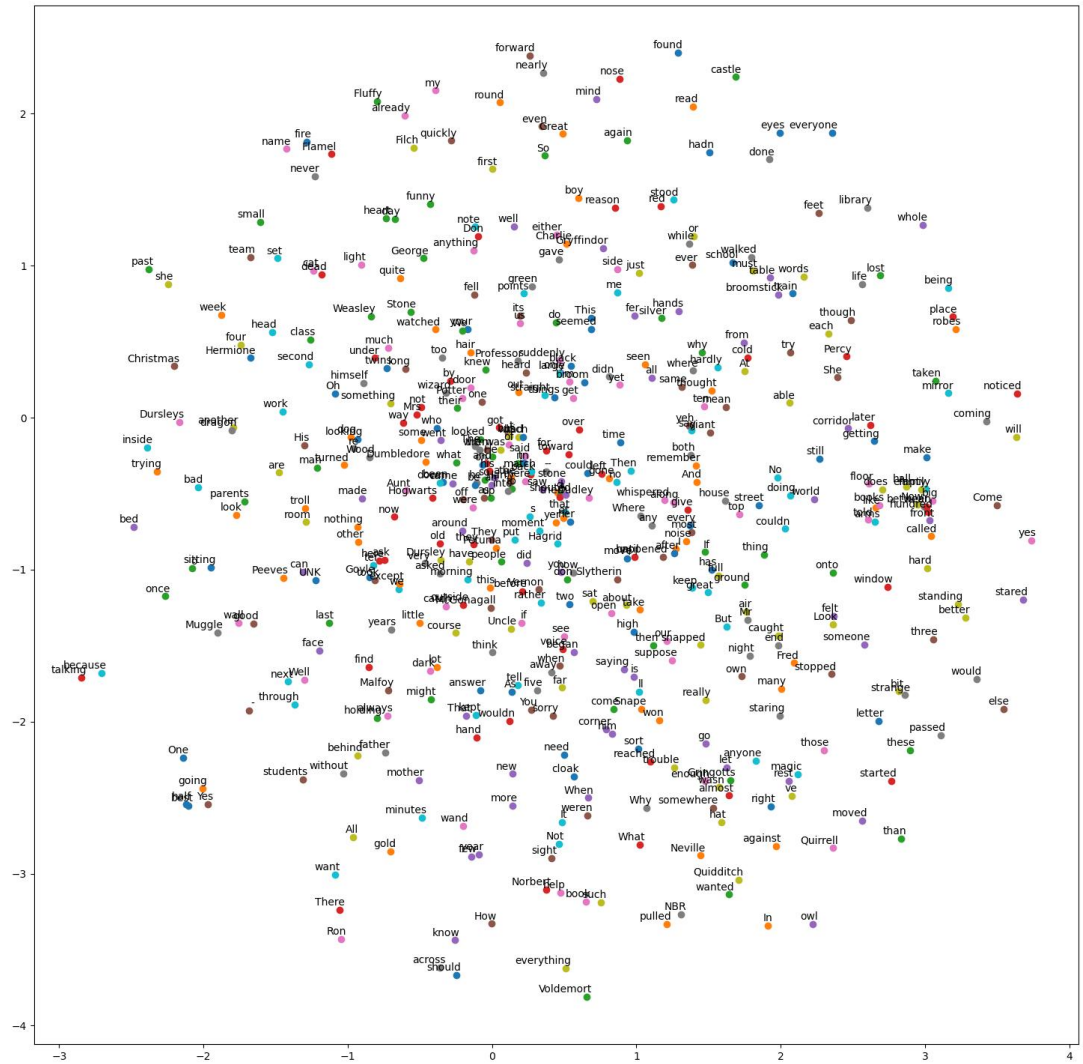
降维数据，显示 2 维特征绘图，增加控制条件，控制数据范围，只显示规定范围内的点。数据可视化如下图所示。

```
# Step 6: Visualize the embeddings.
def plot_with_labels(low_dim_embs, labels,
filename='tsne.png'):
    assert low_dim_embs.shape[0] >= len(labels), "More labels
than embeddings"
    print('Visualizing.')
    plt.figure(figsize=(18, 18))  # in inches
    for i, label in enumerate(labels):
        x, y = low_dim_embs[i, :]
        if x < -25 or x > 25 or y < -25 or y > 25:
            continue
        plt.scatter(x, y)
        plt.annotate(label,
                     xy=(x, y),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')

    plt.savefig(filename)
    plt.show()
```

```
    print('TSNE visualization is completed, saved in
{0}.'.format(filename))


matplotlib.use("Agg")
tsne = TSNE(perplexity=30, n_components=2, init='pca',
n_iter=5000)
plot_only = 500
low_dim_embs =
tsne.fit_transform(final_embedding[:plot_only, :])
labels = [idx2word[i] for i in range(plot_only)]
plot_with_labels(low_dim_embs, labels, filename='./hp1.png')
```



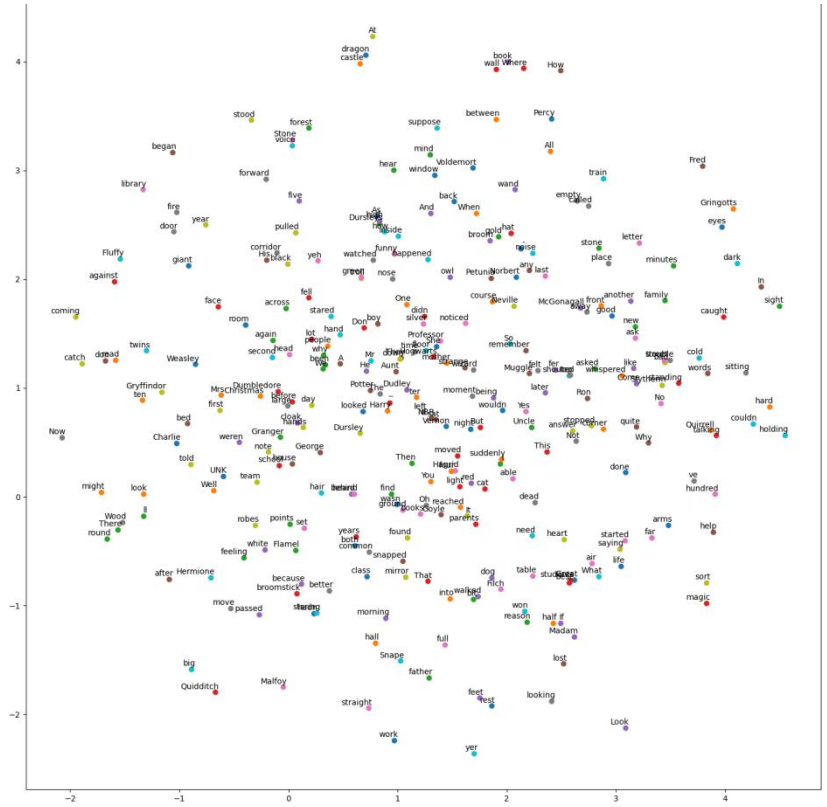
实验结果及分析：

在上图实验结果中，显示的词中有许多无用的词，导致我们可视化效果不太好，于是考虑去除停用词，这样会使有用的信息更加明显。

```
def plot_with_labels(low_dim_embs, labels, filename='tsne.png'):
    assert low_dim_embs.shape[0] >= len(labels), "More labels than embeddings"
    stop = []
    # 引入停用词
    with open('D:\\D3S\\自然语言处理\\实验\\sy2\\Tutorial_4_TFIDF-main\\teststopwords.txt','r',encoding='utf-8')as f
        for l in f:
            if l not in stop:
                stop.append(l)

    print('Visualizing.')
    plt.figure(figsize=(18, 18))  # in inches
    for i, label in enumerate(labels):
        if label in stop:
            continue
        x, y = low_dim_embs[i, :]
        if x < -25 or x > 25 or y < -25 or y > 25:
            continue
        plt.scatter(x, y)
        plt.annotate(label,
                     xy=(x, y),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')

    plt.savefig(filename)
    plt.show()
    print('TSNE visualization is completed, saved in {0}.'.format(filename))
```



去除部分不需要的停用词之后，可视化图表更加清晰，此时我们分析 Harry Potter 文本中词语的关系会更加方便。由上图可以看出“Muggle-Uncle”、“Harry-Potter-Dudely-Dursley-Dumbledore”等关系较近，说明训练结果不错，embedding 可以比较准确表示词语。

后记：

此次实验学习了 word2vec 的实现代码，每一个过程手动实现非常清晰，由于有师兄的讲解，代码理解起来事半功倍，在调试过程中，对代码理解更进一步了。Bert 实现直接调用模型，非常便捷，而且 Bert 的训练效果不错，是一个很不错的算法模型。