Word2Vec & TransE

实验报告

姓名:王雨静

班级 : 1期NLP训练营

日期:2021年2月28日

目录

一、案例简介

Word2Vec是词嵌入的经典模型，它通过词之间的上下文信息来建模词的相似度。TransE是知识表示学习领域的经典模型，它借鉴了Word2Vec的思路，用“头实体+关系=尾实体”这一简单的训练目标取得了惊人的效果。本次任务要求在给定的框架中分别基于Wikipedia和Wikidata数据集实现Word2Vec和TransE，并用具体实例体会词向量和实体/关系向量的含义。

二、Word2Vec实现

2.0 Word2Vec 任务介绍

在这个部分，你需要基于给定的代码实现Word2Vec，在Wikipedia语料库上进行训练，并在给定的WordSim353数据集上进行测试。

我们提供了一份基于gensim的Word2Vec实现，请同学们阅读代码并在Wikipedia语料库上进行训练。

运行`word2vec.py` 后，模型会保存在`embedding/word2vec_gensim`中，可以使用以下代码加载模型并衡量两个单词的相似性

```python
from gensim.models import Word2Vec
en_wiki_word2vec_model = Word2Vec.load('embeddings/word2vec_gensim')
sim = en_wiki_word2vec_model.similarity('woman', 'man')

```

在WordSim353数据集中，表格的第一、二列是一对单词，第三列中是该单词对的人工打分，分值范围为0-10之间。同学们需要在数据集上用Spearman\[1][1]相关系数衡量词向量的质量。关于gensim的Word2Vec模型更多接口和用法，请参考\[2][2]

[1] \[1] <https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient>
[2] \[2] <https://radimrehurek.com/gensim/models/word2vec.html>

具体任务

* 运行`word2vec.py`训练Word2Vec模型并保存，在WordSim353上衡量词向量的质量。
* 探究Word2Vec中各个参数对模型的影响，例如词向量维度、窗口大小、最小出现次数。如果wiki数据太大，可以在我们提供的text8语料或你找到的其他语料上进行实验。
*（选做）对Word2Vec模型进行改进，改进的方法可以参考\[3] 包括加入词义信息、字向量和词汇知识等方法。请详细叙述采用的改进方法和实验结果分析。

2.1 在wiki语料库的训练与测试

2.1.1 在Wikipedia语料库上训练模型

步骤：运行`word2vec.py`训练Word2Vec模型并保存

训练开始时间：2021-01-25 19:45:21

训练结束时间：2021-01-26 08:56:33

2.1.2 在WordSim353上衡量词向量的质量

代码实现：

a. 读取数据

```
data = pd.read_csv('wordsim353/combined.csv', sep = ',')
```

b. 用训练好的模型衡量两个单词的相似性，如果单词确实，记0

```
for i in range(data.shape[0]):
    try:
        data.loc[i, 'score'] =
en_wiki_word2vec_model.wv.similarity(data.loc[i, 'Word
1'],data.loc[i, 'Word 2'])
    except:
        data.loc[i,  'score'] = 0
```

c. 去除相似性为0的行（无效数据）

[3] \[3] A unified model for word sense representation and disambiguation. in Proceedings of EMNLP, 2014.

```
data = data.loc[data['score']!=0]
```

d. 将人工打分和模型打分分别排序

```
data['x_rank'] = data['score'].rank(method = 'first')
data['y_rank'] = data['Human (mean)'].rank(method = 'first')
```

e. 计算Spearman相关系数

```
def spearman(r1, r2): #r1, r2是两组rank的series
    cov = r1.cov(r2)
    std1 = r1.std()
    std2 = r2.std()
    r = cov / (std1 * std2)
    return r
n = data.shape[0]
r = spearman(data['x_rank'], data['y_rank'])
print("Spearman's rank corr coef: ", r)
```

f. 输出：

```
Spearman's rank corr coef:  0.6653371295787955
```

实验结果：

Spearman相关系数0.6653。

2.2 参数对模型的影响

2.2.0 评价方法：

1. 模型本身的loss
2. WordSim353上词向量的质量
    a. 斯皮尔曼相关系数
    b. 均方误差（mean square error）

        先对人工打分用MinMax归一化，然后和预测值求MSE。

为什么使用均方误差？和wiki大数据集相比，txt8数据集很小，最终训练出来的模型效果不佳，斯皮尔曼相关系数会很低，甚至为负，因此引进了另一种评价指标——均方误差——来衡量模型的效果。均方误差越小，表示模型预测的相似度越接近人的预测值，模型效果越佳。

2.2.1 词向量维度

控制变量： window = 10

min_count = 10

自变量： sizes = [4, 8, 16, 32, 64, 128, 256, 512, 1024]

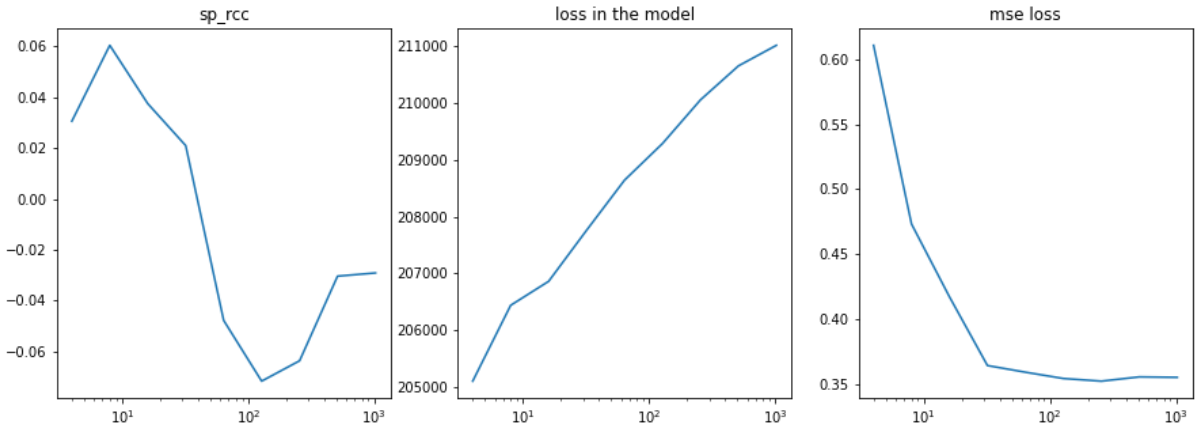

Fig 2.1. 词向量维度对模型效果的影响

分析：

从图中可以发现，随着词向量维度的增加，模型的损失逐渐提升，但是在WorSim353上的MSE逐渐下降。

模型本身的损失，可能是由于维度越大，模型越复杂，导致其损失较大。还有可能是对于较为复杂（词向量维度）的模型，5个epoch能梯度下降的损失更为有限。

随着词向量维度的增加，MSE总体成下降趋势，由此可见更大的词向量维度能够使词向量更好地表示每个词的含义。前期MSE下降较快，这说明适当增大词向量维度能获得效果更好的模型。后期基本保持早0.35左右不变，这可能是因为100左右的维度在这个数据集上已经基本足够了；而模型的整体效果可能受限于其他变量，如模型本身、参数（窗口大小等）、数据集大小等。

此外，可以发现斯皮尔曼相关系数在0附近浮动，且变化不明显，因此不能作为很好地评价指标。

2.2.2 窗口大小

控制变量： size = 128

min_count = 10

自变量：windows = [3, 5, 10, 15, 20, 25]

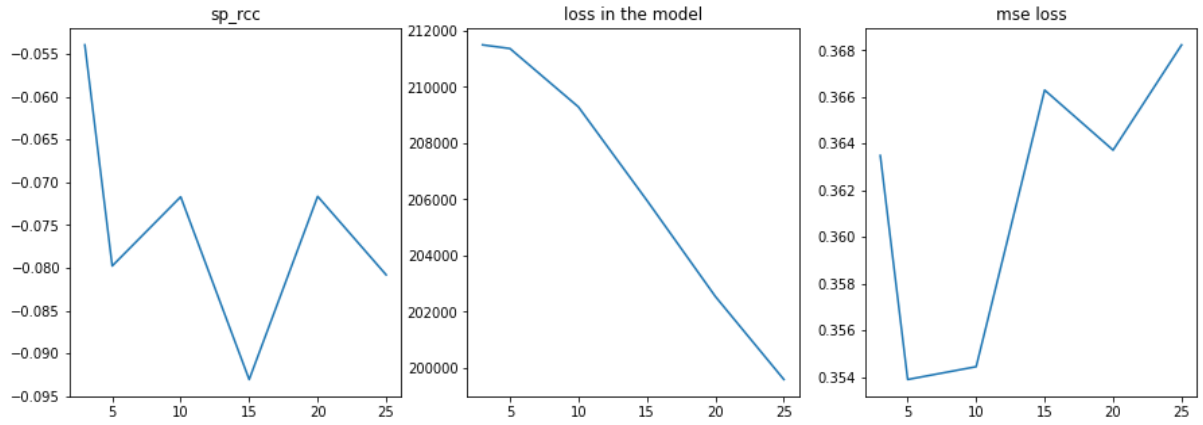

Fig 2.2. 窗口大小对模型效果的影响

分析:

同样的，斯皮尔曼相关系数在这里的参考价值不大。

模型本身的loss呈下降趋势。

MSE先下降，后上升。过于小的窗口，使得每个词向量表示时用到的其他词数量很少，不能很好地体现这个词本身的含义。而过于大的窗口，相关词数量太大，也不能很好得体现这个词的含义。模型效果在窗口大小约为5-10时达到最优。

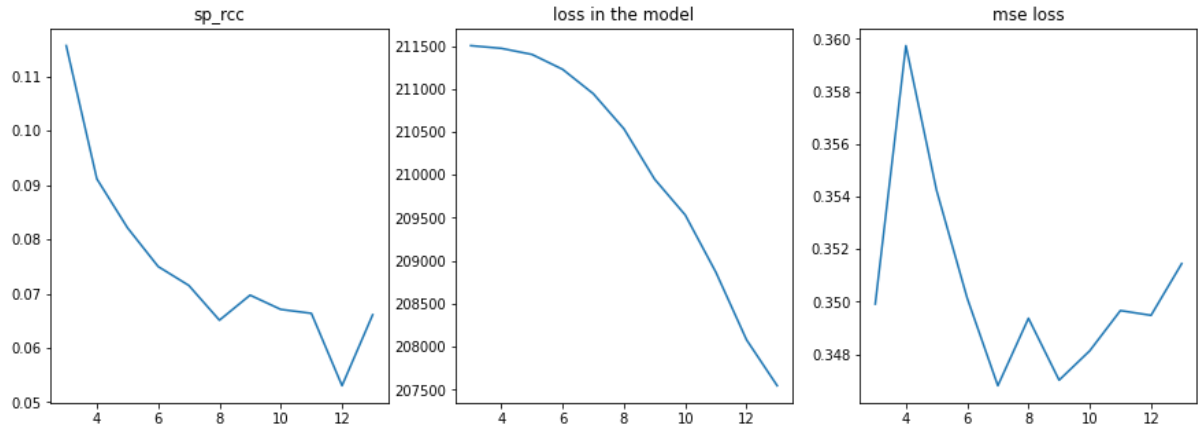

Fig 2.3. 窗口大小对模型效果的影响

进一步缩小窗口大小的范围，可以发现模型在7-10范围内效果较好，但并不显著。（只有300个左右的数据，误差可能较大）

2.2.3 最小出现次数

控制变量： size = 128

windows = 10

自变量： min_count = [2, 4, 6, 8, 10, 16, 32, 64, 128, 256, 512]

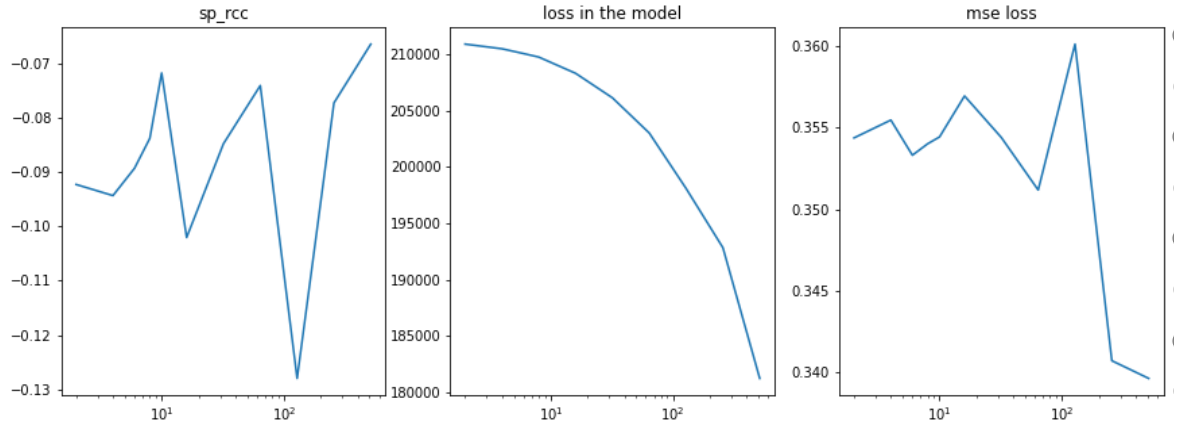

Fig 2.4.最小出现次数对模型效果的影响

随着最小出现次数的增加，模型loss降低，mse基本保持不变，但在超过100以后，有降低，这可能是因为算法中将WordSim353中没有出现在模型里的词去掉了，而留下的词向量的质量更好。

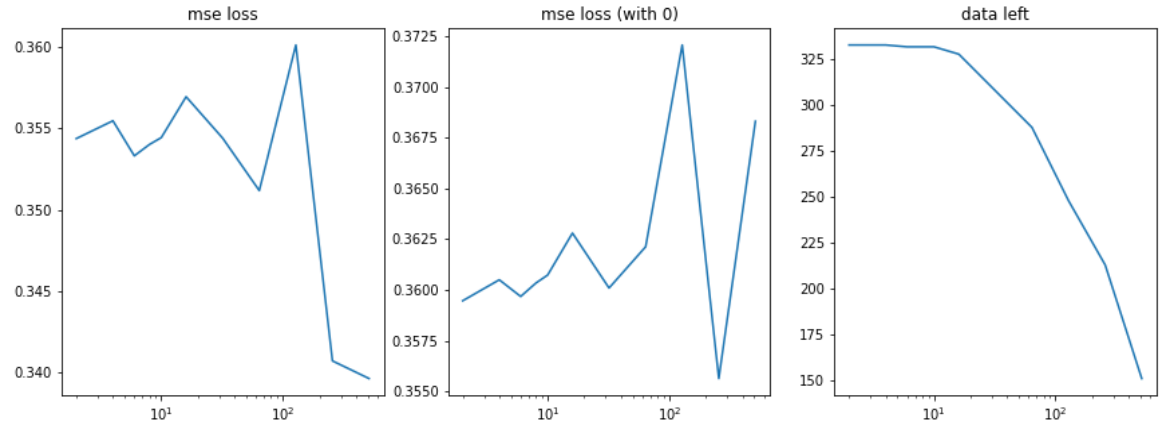

Fig 2.5.最小出现次数对模型效果的影响2

随着最小出现次数的增加，模型中留下的词数量会减少，相对应，留下的WordSim数据集中的词也会边上，正如Fig 2.5.（右一）所示。

如果不去掉模型中没有出现的词，mse明显会大一些，而相对较大的最小出现次数，会是的剩下的词的数量显著降低，总体mse较大。而图中在min_count = 256的那个点，可能是噪声，可能是那些被舍去的词的相似度比较接近于0（当模型中词没有出现时，算法将相似度填上0），所以mse反而较低。

所以，综合模型复杂度，所表示出的向量的质量考虑，可以选择min_count = 10左右。而如果数据集较大的话，直觉上可以考虑使用更大的min_count。

三、TransE实现

3.0 TransE 任务介绍

这个部分中，你需要根据提供的代码框架实现TransE，在wikidata数据集训练出实体和关系的向量表示，并对向量进行分析。

具体任务

* 在文件`TransE.py`中，你需要补全`TransE`类中的缺失项，完成TransE模型的训练。需要补全的部分为：
  * `_calc()`：计算给定三元组的得分函数(score function)
  * `loss()`：计算模型的损失函数(loss function)
* 完成TransE的训练，得到实体和关系的向量表示，存储在`entity2vec.txt`和`relation2vec.txt`中。
* 给定头实体Q30，关系P36，最接近的尾实体是哪些？
* 给定头实体Q30，尾实体Q49，最接近的关系是哪些？
* 在 <https://www.wikidata.org/wiki/Q30> 和 <https://www.wikidata.org/wiki/Property:P36> 中查找上述实体和关系的真实含义，你的程序给出了合理的结果吗？请分析原因。
*（选做）改变参数`p_norm`和`margin`，重新训练模型，分析模型的变化。

3.1 缺失项补全

3.1.1 _calc()

对embeddings进行l2 normalization后，计算（头实体+关系）和尾实体之间的基于p_norm的距离。

代码实现：

```
def _calc(self, h, t, r):
        # TO DO: implement score function

        if self.norm_flag: # normalize embeddings with l2 norm
          h = F.normalize(h)
          t = F.normalize(t)
          r = F.normalize(r)

        d = h + r -t
        score = torch.norm(d, p = self.p_norm, dim = -1)
        return score
```

3.1.2 loss()

根据公式，取(margin + pos_score - neg_score)和0之间的较大值，然后求和。

（备注：由于这里用的是求和，不是求平均，所以最后的loss会比较大，且和训练集的大小相关）

```
    def loss(self, pos_score, neg_score):
      # TO DO: implement loss function
      loss = self.margin + pos_score - neg_score
      loss[loss<0] = 0
      loss = loss.sum()
      return loss
```

3.2 模型训练

3.2.1 调整

参数调整：

```
trainTimes = 1000
learningRate = 0.03
```

保存调整：

每训练50个epoch，保存一次实体和关系的向量表示。用numpy矩阵的格式保存，更节省空间。

保存部分代码实现：

```
    if times%50==0:
      ent_path = "/content/drive/MyDrive/Colab
Notebooks/xtzx/nlp1/TransE/entity2vec" + str(times) + ".npy"
      ent = transe.ent_embeddings.weight.data.numpy()
      np.save(ent_path, ent)
      rel_path = "/content/drive/MyDrive/Colab
Notebooks/xtzx/nlp1/TransE/relation2vec" + str(times) + ".npy"
      rel = transe.rel_embeddings.weight.data.numpy()
      np.save(rel_path,rel)
```

3.2.2 训练结果1

Q30（美国）+ P36（首都）

正确答案为Q61，华盛顿。

测试代码：

```
def top5(ent, rel):
  res = ent[118] + rel[20]
  dis = res - ent
  dis_norm = np.linalg.norm(dis, ord = 1, axis = 1)
  index = np.argsort(dis_norm)[:5]
  print("dis_norm:", dis_norm[index])
  print("index:", index)

  for i in index:
    print(entity.loc[entity['50000'] == i])

  return dis_norm

def rankQ61(dis_norm): #正确答案Q61 在 1822
  rank = np.argwhere(dis_norm<dis_norm[1822]).shape[0]
  per = rank / 50000
  print("rank: ", rank, "/50000")
  return rank
```

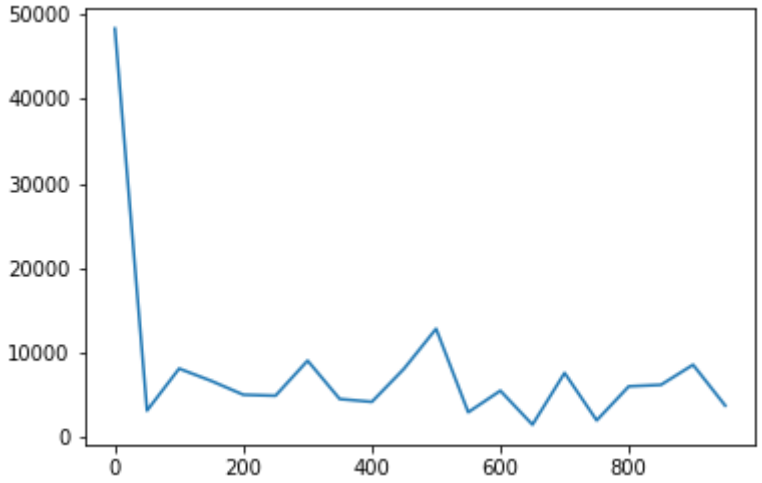

Fig 3.1. 训练次数对Q61 rank的影响

从第50个epoch往后，rank就在5000（前10%）左右抖动，其中最优在epoch = 650时达到1510（前3%左右）。由此可见，虽然随着模型的继续训练，loss依然在下降，但很可能过拟合训练集了。（特别是训练集数量仅为2992）

在epoch = 650时，排名前三的实体是：

Q775：Gelderland, province of the Netherlands

Q1101: North Brabant, province in the Netherlands

Q701: North Holland, province in the Netherlands

都是荷兰的省份，都和华盛顿同属省份名。

3.2.3 参数调整&训练结果

```
self.trainTimes = 300
self.learningRate = 0.01
```

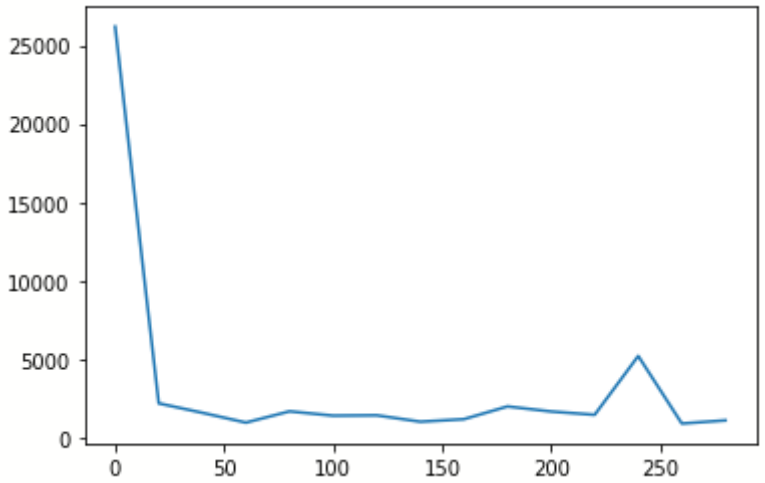

Fig 3.2. 训练次数对Q61 rank的影响_2

将训练次数减少到300个epoch后，最有效的训练还是前60词。后面虽然loss还在不断下降，但是模型的效果却可能并没有提升。这可能是因为模型在训练集上过拟合了。由于测试集只选用了一个三元组，也不能全面地反应模型的效果。

```
self.trainTimes = 60
self.learningRate = 0.01
```

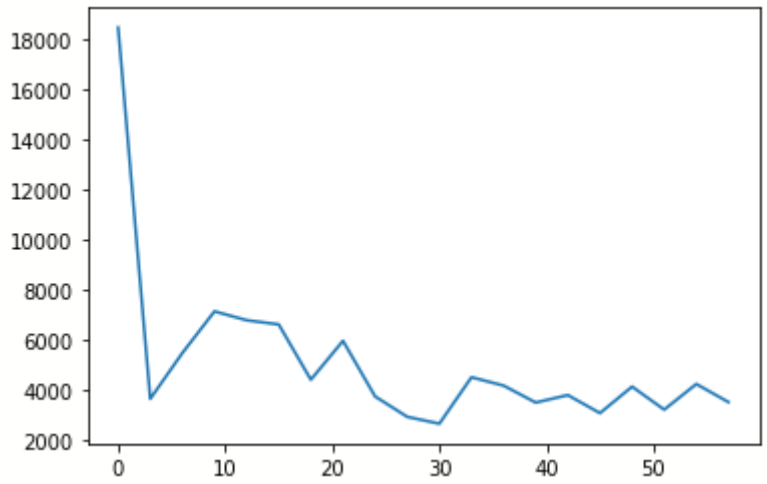

Fig 3.3. 训练次数对Q61 rank的影响_3

训练次数减少到60个epoch后，Q61的rank也稳定在相似的区间内，大体上是令人满意的。排名最考前的实体也是Q1101和Q701。

3.2.4 Q49 - Q30预测结果分析

目标：给定头实体Q30，尾实体Q49，找到最接近的关系是哪些。

代码实现：

```
def top5rel(ent, rel):
  res = ent[26878] - ent[118]  #Q49 - Q30
  dis = res - rel
  dis_norm = np.linalg.norm(dis, ord = 1, axis = 1)
  index = np.argsort(dis_norm)[:5]
  print("dis_norm:", dis_norm[index])
  print("index:", index)

  for i in index:
    print(relation.loc[relation['378'] == i])

  return dis_norm
```

```
def rankP30(dis_norm): #正确答案P30 在 19
  rank = np.argwhere(dis_norm<dis_norm[19]).shape[0]
  print("rank: ", rank, "/378")
  return rank

time_list = list(range(0, 60, 3))
rank_list = []
for times in time_list:
  ent_path = "/content/drive/MyDrive/Colab
Notebooks/xtzx/nlp1/TransE/entity2vec"
  rel_path = "/content/drive/MyDrive/Colab
Notebooks/xtzx/nlp1/TransE/relation2vec"

  ent_file = ent_path + str(times) + ".npy"
  ent = np.load(ent_file)
  rel_file = rel_path + str(times) + ".npy"
  rel = np.load(rel_file)

  dis_norm = top5rel(ent, rel)
  rank = rankP30(dis_norm)
  rank_list.append(rank)
```

排名最靠前的关系：

P376: located on astronomical body, 所在天体

P500: exclave of, territory is legally or politically attached to a main territory, 飞地从属于

P885: origin of the watercouse，源头

P853: CERO rating，电脑娱乐分级机构评级，日本的电玩评级系统

P479: input method, 输入设备

可以发现，排名靠前的关系都带有地理上的从属、源头的意思，和正确的答案——所在的大洲——较为接近，但也存在一些看上去并不相近的关系，例如点完评级系统、输入设备等。

更令人意外，令人沮丧的是，正确答案的排名在378个关系中总是排在370左右。可见对关系的预测并不好。

3.2.5 预测结果不好的可能原因

模型并没有给出理想的预测结果，可能的原因有：

1. 训练集较小：
    a. 训练集中共有299291对三元组，这相比较于50000个实体，378条关系来说，是相对较少的，可能并不能训练出较好的对实体和关系的表示。
    b. 测试的实体和关系在训练集中出现次数较少：
        i. Q30 美国这个实体在训练集中作为头实体出现了135次
        ii. P20 首都这条关系在训练集出现了1138次
        iii. Q61 华盛顿这个实体在训练集中作为尾实体出现了118次
        iv. Q49 北美洲这条实体在训练集中作为尾实体出现了66次
        v. P30 所在的大洲这条关系在训练集中出现了66次

这些实体出现的次数已经远大于了每个实体的平均出现次数（5.8次），但是题目中涉及到的关系在训练集中的出现次数却远低于每条关系的平均出现次数（767次）。这为模型为什么能更好得预测实体，却不能给出较好的关系预测提供了一种解释。

2. 参数的选择问题
    a. 由于训练集较小，较复杂的模型，较多的训练次数可能导致模型在训练集上过拟合。这从训练的loss在不断下降，甚至到了1000个epoch都没有达到最低值，但是所提供的预测效果却没有提升甚至不是最优可以体现一二。
    b. 其他参数可能不是最优。这条尚未探索。
3. 模型本身的限制

    TransE将实体关系都用高维向量的方式来表达，这能一定程度上代表词的意思，但是可能还是缺少了很多信息，或者说单凭一个只有方向和距离的向量不能很好地刻画出实体/关系。而一些非线性的结构，如多个环形或树状的结构，可能无法由简单的向量来刻画。举个例子，美国、加拿大这些实体的向量是不同的，但是加上“所在的大洲”后正确的答案应该是北美洲，但由于头实体的向量不同，计算出的尾实体也会相应有所不同。

3.3 p_norm 对模型的影响

将p_norm设置为2，其他参数保持不变，重新训练模型：

在训练过程中发现，相比p_norm = 1时，loss可以从26万较快下降到10万左右，p_norm = 2时，loss从29万仅仅下降到26万。但是，测试结果却给出了另一种可能：

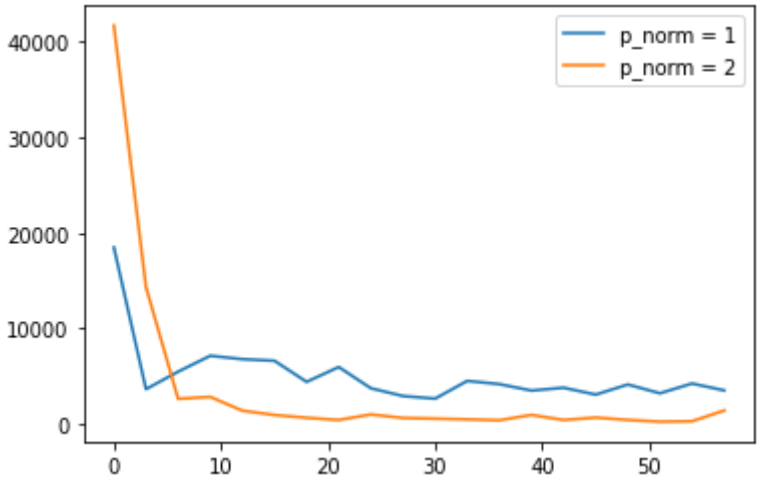

Fig 3.4. p_norm对模型的影响

从图中可以发现，使用p_norm = 2后，前期模型的效果可能没有p_norm = 1好，但是中后期却稳定优于p_norm = 2，且更平稳，抖动更少。

但是，预测出的前几名的实体，却与正确答案相差甚远：

Q1424515: charcoal 木炭

Q5460604：Vital articles, 基础条目

Q34740: genus, 属，生物分类法中的一个层级

3.4 margin对模型的影响

```
self.p_norm = 1
self.trainTimes = 31
self.learningRate = 0.01
margins = [0,  0.1, 0.3, 1.0, 3.0, 10.0]
```

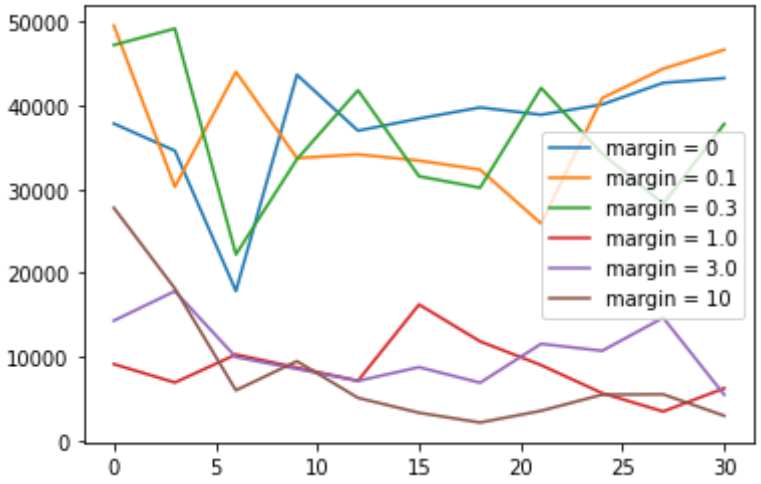

Fig 3.5. margin对模型的影响

从图中可以发现，在margin的范围是0-10之间时，margin达到1或者更大时，即模型的容忍度越高时，模型的效果越好。

此外，前6个epoch的训练都基本提升模型的效果。

四、反思

在这一次的实验中，学习和实践了Word2Vec和TransE算法。虽然实践过程中困难重重，包括硬件条件上的限制使得跑模型的速度很慢，自身编程基础薄弱，但是一步一步将模型跑通，还是很快乐的，在和同学、助教讨论的过程中开拓了思路，学到了很多。