WebInfo-exp2 实验报告

组员：

廖佳怡 PB19151776

谢新格 PB19081644

实验内容：

在给定的带有文本描述的知识图谱数据集上，设计一种知识图谱补全算法。在这个给定的测试集上，预测出缺失的三元组的尾实体。

逆哈希:

我们很早就发现了原始数据集是FB15k-237，我们先使用了多种逆哈希方法还原测试集，先基于纵向频率统计，能完全还原关系编号，确定50%的实体编号，再使用横向的迭代匹配，经多次迭代，并缩小可能范围，我们确认了绝大多数实体编号，逆哈希准确率达到了98%

2021-12-19 10:18:41	192_1639880321_inverse_hash.txt	0.980993	0.980993

注：我们本着学术诚信的原则，在以下模型的训练中，均**未使用逆哈希的结果，完全只使用训练集**，这个逆哈希结果只用做了测试时准确率的估计。

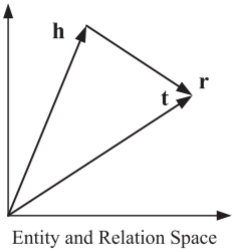
实验方案：

我们一共进行了三次模型迭代，接下来分别展示。

模型一：OpenKE中使用transE

设计思路

transE原理图：



Entity and Relation Space

(a) TransE.

$$f_r(h,t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{1/2}.$$

使用OpenKE进行transE模型搭建和下游任务的完成，此设计只利用了图结构信息。难点在于配置环境，使用OpenKE接口进行训练及预测（其文档中描述很不清晰）和数据预处理。在此使用了搭载好gpu_pytorch的docker环境，每次训练大约历时45min。

方案实现

数据预处理代码如DataProcessing.py文件，其中需对实体和关系进行连续编号。

- 训练代码：train.py

```
import config
from  models import *
import json
import os
con = config.Config()
con.set_use_gpu(True)
con.set_in_path("./benchmarks/our/")
con.set_work_threads(8)
con.set_train_times(1000)
con.set_nbatches(100)
con.set_alpha(0.001)
con.set_bern(0)
con.set_dimension(100)
con.set_margin(1.0)
con.set_ent_neg_rate(1)
con.set_rel_neg_rate(0)
con.set_opt_method("SGD")
con.set_save_steps(100)
con.set_valid_steps(100)
con.set_early_stopping_patience(10)
con.set_checkpoint_dir("./checkpoint")
con.set_result_dir("./result")
con.set_test_link(False)
con.set_test_triple(False)
con.init()
print("after init")
con.set_train_model(TransE)
con.train()
```

- 预测：predict.py

```
import config
from models import *
import json
import os
con = config.Config()
con.set_use_gpu(True)
con.set_in_path("./benchmarks/ours/")
con.set_result_dir("./result")
con.init()
con.set_test_model(TransE)
h = []
r = []
k = 5
l = []
with open(".//benchmarks//ljy//test2id.txt","r") as f_read:
    NotFirst = False
    for line in f_read.readlines():
        line = line.split()
        if NotFirst:
            h.append(int(line[0]))
```

```
            r.append(int(line[2]))
        NotFirst = True
tail_predict_list = con.predict_tail(h,r,k)

with open(".//result//map.json",'r') as f_map:
    json_map=json.load(f_map)
    for tail5 in tail_predict_list:
        flag = False
        print(tail5)
        for t in tail5:
            if flag:
                l.append(',')
            flag = True
            l.append(json_map[str(t)])
        l.append("\n")
f_write = open(".//result//res.txt","w")
f_write.writelines(l)
f_write.close()
```

训练展示：

```
Epoch 98 | loss: 4462.103687
Epoch 99 | loss: 4385.521080
Epoch 99 has finished, saving...
Epoch 99 has finished, validating...
Best model | hit@10 of valid set is 0.409296
Epoch 100 | loss: 4298.214315
Epoch 101 | loss: 4435.649178
Epoch 102 | loss: 4261.547884
Epoch 103 | loss: 4279.898436
Epoch 104 | loss: 4316.596909
Epoch 105 | loss: 4185.584116
Epoch 106 | loss: 4129.010689
Epoch 107 | loss: 4140.194132
Epoch 108 | loss: 4213.862492
Epoch 109 | loss: 4098.292675
Epoch 110 | loss: 4068.615349
Epoch 111 | loss: 4027.511238
Epoch 112 | loss: 3979.041046
Epoch 113 | loss: 3993.783363
Epoch 114 | loss: 3963.516233
Epoch 115 | loss: 3961.648005
```

结果分析：

2021-11-28 15:36:34	214_1638084993_res.txt	0.153963	0.314082

我们的hit@5达到了31.4%，这并不能算是一个很高的数，后来，我们分析了数据集的分布，发现了这个数据集的分布其实非常不均匀，一些实体在训练集中只出现个位数次甚至只出现一次，而有的实体却出现了上千次，甚至在测试集中，有的实体根本没在训练集中出现，那么关于这个实体，只基于图结构的话，是根本无法预测的，所以，由于图结构本身的高度不均衡，预测的结果会相对较低。

模型二：transE + 实体相似度

设计思路：

在一开始的模型一中，我们，我们直接使用的transE模型是利用知识图谱的图结构进行预测的，但并没有用上文本信息。接下来我们期望将文本信息利用上。由于OpenKE封装的比较好，我们无法直接对其改动，所以我们选择了对训练数据先进行预处理的方式。

关于这个数据集中，如模型一中的结果分析，训练集的不平衡。接下来我们希望通过数据预处理的方式，利用文本信息，将这个训练集添加更多的三元组，进行平衡化处理。

方案实现：

我们观察到，所给的两个文本信息，`entity_with_text.txt`，`relation_with_text.txt`，其实包含在这两个文本之间的信息量，`entity_with_text.txt`是远大于`relation_with_text.txt`的，这不仅因为实体的个数更多，也因为对一个实体的文本描述的字数更多，即描述的更详细。

我们现在只利用`entity_with_text.txt`，计算实体之间的两两相似度，通过`word2vec`的语义计算，并且进行排序，将相似度高的两个实体，增添一个新的关系，形成一个新的三元组，而这个新的关系区别于原有的237个关系，实际上，这个关系是有实际意义的，在这里我们理解为两个实体之间具有语义相似度。

另外，我们不仅只添加一个关系，而是添加了多级相似度关系，以区分高相似度的级别，并且我们可以在代码中设计阈值来控制添加进训练集中的三元组个数。

相关代码如下：

```
count = 0
rel_base = 237
rel_max = 237
total_append = 0
f = open('./data/train.txt', 'a')
for sig in map:
    rel = rel_base
    for ent in map:
        if int(sig) >= int(ent):
            continue
        s = wv.n_similarity(map[sig], map[ent])
        if s > 0.975:
            rel += 1
            if rel > rel_max:
                rel_max = rel
            f.write(sig + '\t'+ str(rel) + '\t' + ent + '\n')
            f.write(ent + '\t'+ str(rel) + '\t' + sig + '\n')

    count += 1
    total_append += 2*(rel - rel_base)
    print(str(count) + '\t' +'cur: ' + str(rel-rel_base) +'\t' +'total: ' + str(total_append))
```

最终我们将原来训练集的272115个三元组，扩展到了292131个三元组，并再次基础上使用OpenKE的transE方法进行训练，得到结果如下：

2021-12-13 17:15:41	214_1639386941_res.txt	0.148246	0.304700

结果分析：

然而最终结果没有获得提升，反而下降了了1%，这种方式并不理想，可能是因为单单用实体之间的文本相似度不能很好的反应到含有关系的三元组上，后来我们观察到，在对实际排序过程中，相似度的分布其实是比较连续的，因此如果设定一个阈值，只取阈值以上的相似度，这既减少了信息量，又不具有很好的区分效果。

而且这个方法存在的一个问题是，由于我们计算了实体直接的两两相似度，而如果我们拿一个稠密矩阵或者稠密的上三角矩阵去保存这个相似度数据，估计将有以GB计数的内存开销，所以我们采用了用时间换空间的方法，不保存这个矩阵，而是采用用到时再计算方式，虽然也和矩阵一样两两间各计算一次，但每次生成新的训练集，都有经历这个过程，我们在vlab上进行这个生成，单一次就需要花费36h，并且是已经进行了优化的情况，这种时间开销是难以承受的，因此我们寻找其他方案。

模型三：改进transE

设计思路：

之前也已经提到，OpenKE封装过好而难以直接修改，我们从github上找到了未用机器学习框架的transE模型，在此基础上进行修改。

我们注意到，在transE模型中，它的loss函数是：

$$\sum[\gamma + d(h + l, t) - d(h^{'} + l, t^{'})]_{+}$$

其中的γ是margin，是一个参数，但对于不同的三元组来说，这都是一个常数，我们可以修改这个常数，使它和正在运算中的三元组的文本相似度有关。

直观的认为，正例的文本相似度应当高于负例，并且如果这个高的程度越高，则负例应该应该越负，即在embedding中有着更大的距离，即增加γ。

另外，需要注意的是，在transE模型中，它将非正例的三元组都视为了负例，但实际上，测试集中的正例是出于这个负例的，为了减弱这点的影响，有的时候，当正例的文本相似度低于负例的时候，这个时候我们会认为，这个负例更可能是一个正例，因此我们减小γ。

首先，和模型二中一样，我们用word2vec的方法可以得到两个句子直接的文本相似度，记a, b 两个实体或关系的相应的句子的文本相似度为$s(a, b)$

下面，我们的设计是，引入三元组(h, l, t)的文本相似度记为$D(h, l, t)$，则

$$D(h, l, t) = s(h, l) \times s(t, l)$$

这样我们修正γ为

$$\gamma \leftarrow \gamma + D(h, l, t) - D(h^{'}, l, t^{'})$$

这样我们就成功改造了transE，引入了文本信息。

方案实现：

不同于模型二中的实体-实体间的相似度，我们这里只用记录实体-关系相似度，因此设计到的空间远远减小，可以直接用一个稠密矩阵记录，以提高时间效率。

实验步骤如下：

1. 使用word2vec模型，训练出word2vec.model，保存在output中
2. 使用word2vec.model，计算每个实体和每个关系的句子相似度，并形成实体或关系到其句子的映射表，再生成相似度的稠密矩阵，导出到`ent_rel_sim.npy`，大小为几十MB的数量级。
3. 修改transE中的模型，使其支持我们的设计，并且参数化。

4. 再一次改进模型，可手动设置断点，使训练和测试能从断点处继续，这样可以将训练和测试分开到不同的时间段，而每个段使用不同的学习率lr，这样能在不使用GPU的情况下加快loss下降，减少总的epoch数，减少训练时间，来方便我们实验。

对于设计中进一步的，我们进一步参数化，引入一个新的超参数，记为μ，则

$$\gamma \leftarrow \gamma + \mu \times (D(h,l,t) - D(h^{'},l,t^{'}))$$

这样我们可以控制文本信息的利用程度，相关代码如下：

```
if self.use_w2v:
    w2v_correct = self.ent_rel_sim[int(triple[0])][int(triple[2])] \
                * self.ent_rel_sim[int(triple[1])][int(triple[2])]
    w2v_corrupt = self.ent_rel_sim[int(corrupted_triple[0])][int(triple[2])] \
                * self.ent_rel_sim[int(corrupted_triple[1])][int(triple[2])]
    w2v_revise = w2v_correct - w2v_corrupt
```

```
def hinge_loss(self,dist_correct,dist_corrupt, w2v_revise):
        return max(0,dist_correct-dist_corrupt+self.margin + self.mu*w2v_revise)
```

另外，我们可以设置了每5轮epoch，就将结果向量保存到temp文件里，方便下次继续运行

并且在训练的时候，我们需要手动调整学习率，从0.01->0.001->0.0001，这样才能将loss收敛到较低的值，而调整学习率这一点，不像openKE里面一样那么智能，故我们引入训练中断是必要的。

```
#保存临时结果
if epoch % 5 == 3:
    with open(temp_path + "entity_temp.pkl", "wb") as f_e:
        pickle.dump(self.entity, f_e)
    with open(temp_path + "relation_temp.pkl", "wb") as f_r:
        pickle.dump(self.relation, f_r)
    print('*****write temp file in epoch: ', epoch)
```

同样，在测试过程中，我们也设置了20次step，就将结果保存下来，方便中断的时候继续。

训练展示：

```
load file...
Complete load. entity : 14541 , relation : 237 , triple : 272115
batch size:  2721
epoch:  0 cost time: 92.903
loss:  1095833.5039867663
epoch:  1 cost time: 89.597
loss:  870173.9873089968
epoch:  2 cost time: 88.469
loss:  706507.3198931415
epoch:  3 cost time: 86.721
loss:  563774.0724071298
*****write temp file in epoch:  3
epoch:  4 cost time: 84.381
loss:  445092.6523262239
epoch:  5 cost time: 81.6
loss:  360658.9558460762
epoch:  6 cost time: 80.211
loss:  298000.7306288589
epoch:  7 cost time: 79.377
loss:  253784.61551095077
epoch:  8 cost time: 78.864
loss:  218447.1445466604
*****write temp file in epoch:  8
epoch:  9 cost time: 76.343
loss:  192551.99488075095
epoch:  10 cost time: 76.462
```

最佳实验结果：

参数$\gamma = 4.0, \mu = 12, dim = 200$

实验结果hit@5

2021-12-18 19:00:39	192_1639825239_tail_predict.txt	0.096208	0.323121

消融实验：

为了对比我们的新方法相比transE的提升，我们这里直接使用transE，而先不用文本信息

第一次测试：

使用和模型一中openKE里的数据，$\gamma = 5.0, dim = 200$

实验结果hit@5

2021-12-14 10:48:20	192_1639450100_tail_predict.txt	0.111551	0.308170

第二次测试：

在参数γ保持不变的情况下，即$\gamma = 4.0, dim = 200$

实验结果hit@5

2021-12-18 18:59:42	192_1639825182_tail_predict.txt	0.049350	0.294049

第三次测试：

再将一二测试中的γ取个平均，即$\gamma = 4.5, dim = 200$

实验结果hit@5

2021-12-18 19:01:18	192_1639825278_tail_predict.txt	0.092837	0.312030

结果分析：

- 可以看到在γ相同时，我们的方法能将准确率提高近3%
- 即使γ变化时，也有1%以上的提升，可以认为我们的改进将transE的正确率提高了1%~3%
- 使用和openKE同样的参数，并不使用文本信息时，模型三的准确率略小于直接使用openKE，这是因为openKE能够使用GPU，且进行了1000次epoch迭代，而我们的模型三因为时间原因，只能进行100epoch以内，收敛效果自然没epoch好。也就是说，我们的最佳结果0.323，如果能直接在openKE上改进的话，应该会有更高的准确率

总结

我们引入了实体-关系的文本信息，通过修改transE中的loss函数，在增加很小的时间和空间开销基础上，提高了transE的预测准确率，提高有1%~3%，且最佳结果达到了0.323。

[参考资料]

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Translating Embeddings for Modeling Multi-relational Data