实验：知识图谱表示学习模型 TransE

执行 notebook 代码，完成 TODO 任务

【实验任务】完成 notebook 文件 transe_todo.ipynb 中的内容：

(1) 补全 TODO 代码

a) 初始化实体 embedding

```
24          for entity in self.entity:
25              e_emb_temp = #【TODO】初始化实体embedding
26              entity_dict[entity] = e_emb_temp / np.linalg.norm(e_emb_temp, ord=2)
27
```

【补全后代码如下】：

```
for entity in self.entity:
    e_emb_temp = np.random.uniform(-6 / math.sqrt(self.embedding_dim),
                                   6 / math.sqrt(self.embedding_dim),
                                   self.embedding_dim) #初始化实体embedding

    entity_dict[entity] = e_emb_temp / np.linalg.norm(e_emb_temp, ord=2)
```

b) 使用范数计算正负例距离

```
124             if self.L1:
125                 dist_correct = # 【TODO】使用L1范数计算正例距离
126                 dist_corrupt = # 【TODO】使用L1范数计算负例距离
127             else:
128                 dist_correct = # 【TODO】使用L2范数计算正例距离
129                 dist_corrupt = # 【TODO】使用L2范数计算正例距离
130
```

【补全后代码如下】：

```
if self.L1:
    dist_correct = np.linalg.norm(h_correct + relation - t_correct, ord=1) # 使用L1范数计算正例距离
    dist_corrupt = np.linalg.norm(h_corrupt + relation - t_corrupt, ord=1) # 使用L1范数计算负例距离
else:
    dist_correct = np.linalg.norm(h_correct + relation - t_correct, ord=2) # 使用L2范数计算正例距离
    dist_corrupt = np.linalg.norm(h_corrupt + relation - t_corrupt, ord=2) # 使用L2范数计算负例距离
```

c) 编写损失函数代码

```
181     def hinge_loss(self, dist_correct, dist_corrupt):
182         return # 【TODO】编写损失函数代码
```

【补全后代码如下】：

```
def hinge_loss(self, dist_correct, dist_corrupt):
    return max(0, self.margin + dist_correct - dist_corrupt) # 编写损失函数代码
```

d) 计算 Mean reciprocal rank（MRR）指标并打印输出

```
        print("mean rank:", mean / len(triple_batch))
        print("hit@3:", hit3 / len(triple_batch))
        print("hit@10:", hit10 / len(triple_batch))
        # 【TODO】计算Mean reciprocal rank (MRR) 指标并打印输出
```

【补全后代码如下】:

```
def calc_metrics(entity_set, triple_list):
    triple_batch = random.sample(triple_list, 100)
    # triple_batch = triple_list
    mean = 0
    hit10 = 0
    hit3 = 0
    mrr_sum = 0  # 新增MRR累加器

    for triple in triple_batch:
        dlist = []
        h = triple[0]
        t = triple[1]
        r = triple[2]
        dlist.append((t, distance(entityId2vec[h], relationId2vec[r], entityId2vec[t])))
        for t_ in entity_set:
            if t_ != t:
                dlist.append((t_, distance(entityId2vec[h], relationId2vec[r], entityId2vec[t_])))
        dlist = sorted(dlist, key=lambda val: val[1])
        for index in range(len(dlist)):
            if dlist[index][0] == t:
                mean += index + 1
                if index < 3:
                    hit3 += 1
                if index < 10:
                    hit10 += 1
                mrr_sum += 1 / (index+1)  # 计算倒数排名
                print(index)
                break

    print("mean rank:", mean / len(triple_batch))
    print("hit@3:", hit3 / len(triple_batch))
    print("hit@10:", hit10 / len(triple_batch))

    #计算Mean reciprocal rank（MRR）指标并打印输出
    mrr = mrr_sum / len(triple_batch)
    print("Mean reciprocal rank (MRR):", mrr)
```

(2) 将 notebook 每个代码段执行成功的输出截图粘贴在下面

【代码段 1 执行截图】:
导入包

```
import codecs
import copy
import math
import random
import time

import numpy as np
```
✓ 0.0s

【代码段 2 执行截图】:
定义变量

```
entity2id = {}
relation2id = {}
loss_ls = []
```
✓ 0.0s

【代码段 3 执行截图】:
定义数据装载函数

```
            entity_set.add(h_)
            entity_set.add(t_)

            relation_set.add(r_)

    return entity_set, relation_set, triple_list
```
✓ 0.0s Python

【代码段 4 执行截图】:
定义距离函数

```
def distanceL2(h, r, t):
    # 为方便求梯度，去掉sqrt
    return np.sum(np.square(h + r - t))


def distanceL1(h, r, t):
    return np.sum(np.fabs(h + r - t))
```

✓ 0.0s

【代码段 5 执行截图】:
TransE 类

```
    def hinge_loss(self, dist_correct,
    dist_corrupt):
        return max(0, self.margin + dist_correct
        - dist_corrupt) # 编写损失函数代码
```

✓ 0.0s Python

【代码段 6 执行截图】:
装载数据

```
file1 = ".\\data\\"
entity_set, relation_set, triple_list = data_loader(file1, "train.txt")
print("load file...")
print("Complete load. entity : %d , relation : %d , triple : %d" % (len(entity_set), len(relation_set), len
(triple_list)))
```

✓ 1.1s

```
load file...
Complete load. entity : 14951 , relation : 1345 , triple : 483142
```

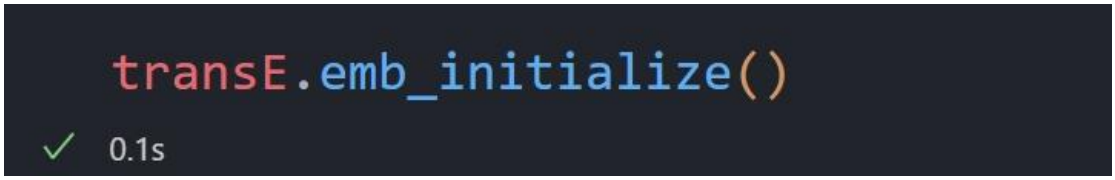
【代码段 7 执行截图】:
创建 TransE 类对象

```
transE = TransE(entity_set, relation_set,
triple_list, embedding_dim=50, learning_rate=0.
01, margin=1, L1=True)
```

✓ 0.0s Python

【代码段 8 执行截图】:

初始化 embedding



【代码段 9 执行截图】:
训练模型

```
batch size:  1207
epoch:  0 cost time: 0.376
loss:  1329.3118310186642
epoch:  1 cost time: 0.381
loss:  1294.9115550590882
epoch:  2 cost time: 0.405
loss:  1225.03822666342
epoch:  3 cost time: 0.359
loss:  1262.9021232828256
epoch:  4 cost time: 0.377
loss:  1182.7413528845952
epoch:  5 cost time: 0.362
loss:  1185.107346441835
epoch:  6 cost time: 0.371
loss:  1247.2728456526288
epoch:  7 cost time: 0.41
loss:  1258.5409538295232
epoch:  8 cost time: 0.368
loss:  1173.452238463467
epoch:  9 cost time: 0.384
loss:  1199.7264013782217
epoch:  10 cost time: 0.357
loss:  1193.5789445805137
epoch:  11 cost time: 0.338
loss:  1194.4190717544134
...
epoch:  1999 cost time: 0.195
loss:  93.33536198557704
```

```
写入文件...
写入完成
```

【代码段 10 执行截图】:
定义变量

```
entityId2vec = {}
relationId2vec = {}
```
✓ 0.0s

【代码段 11 执行截图】:
定义模型装载函数

```
def transE_loader(file):
    file1 = file + "entity_50dim1"
    # file1 = file + "entity_50dim"
    file2 = file + "relation_50dim1"
    # file2 = file + "relation_50dim"
    with codecs.open(file1, 'r') as f:
        content = f.readlines()
        for line in content:
            line = line.strip().split("\t")
            entityId2vec[line[0]] = eval(line[1])
    with codecs.open(file2, 'r') as f:
        content = f.readlines()
        for line in content:
            line = line.strip().split("\t")
            relationId2vec[line[0]] = eval(line[1])
```
✓ 0.0s

【代码段 12 执行截图】:
定义 h+r 与 t 的距离函数

```
def distance(h, r, t):
    h = np.array(h)
    r = np.array(r)
    t = np.array(t)
    s = h + r - t
    return np.linalg.norm(s)
```
✓ 0.0s

【代码段 13 执行截图】:
定义评测函数

```
    print("mean rank:", mean / len(triple_batch))
    print("hit@3:", hit3 / len(triple_batch))
    print("hit@10:", hit10 / len(triple_batch))

    #计算Mean reciprocal rank（MRR）指标并打印输出
    mrr = mrr_sum / len(triple_batch)
    print("Mean reciprocal rank (MRR):", mrr)
```

✓ 0.0s

【代码段 14 执行截图】:
装载数据

```
file1 = ".\\data\\"
print("load file...")
entity_set, relation_set, triple_list = data_loader
(file1, "test.txt")
print("Complete load. entity : %d , relation : %d ,
triple : %d" % (len(entity_set), len(relation_set), len
(triple_list)))
```

✓ 0.0s Python

```
load file...
Complete load. entity : 13584 , relation : 961 , triple : 59071
```

【代码段 15 执行截图】:
装载模型

```
print("load transE vec...")
transE_loader(".\\")
print("Complete load.")
```

✓ 2.1s

```
load transE vec...
Complete load.
```

【代码段 16 执行截图】:
进行模型评测

```
calc_metrics(entity_set, triple_list)
✓ 11.5s
```

```
80
2520
112
8
3
20
3729
97
768
992
114
5739
1862
14
68
6
0
32
3
15
16
11
15
41
1443
...
```

```
mean rank: 546.16
hit@3: 0.05
hit@10: 0.21
Mean reciprocal rank (MRR): 0.07548562788651451
```

【代码段 17 执行截图】:
绘制损失函数曲线

```python
import matplotlib.pyplot as plt
import numpy as np

loss = []
with open("loss", 'r') as f:
    content = f.readlines()[0]
    loss = eval(content)

x = np.array(loss)
y = [i for i in range(len(loss))]
plt.figure(figsize=(6, 4))
plt.plot(x, y, linewidth=1)
plt.xlabel("epochs")  # xlabel、ylabel：分别设置X、Y轴的标题文字。
plt.ylabel("loss")
plt.title("loss")  # title：设置子图的标题。
# plt.savefig('quxiantu.png',dpi=120,bbox_inches='tight')
plt.show()
# plt.close()
```

✓ 0.7s Python