第八次作业

181250035 高金比尔

1. 目的

使用transE方法，实现对FB15k的数据补全。

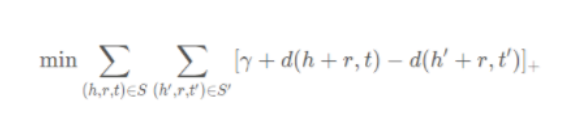
1. 数据集

FB15k

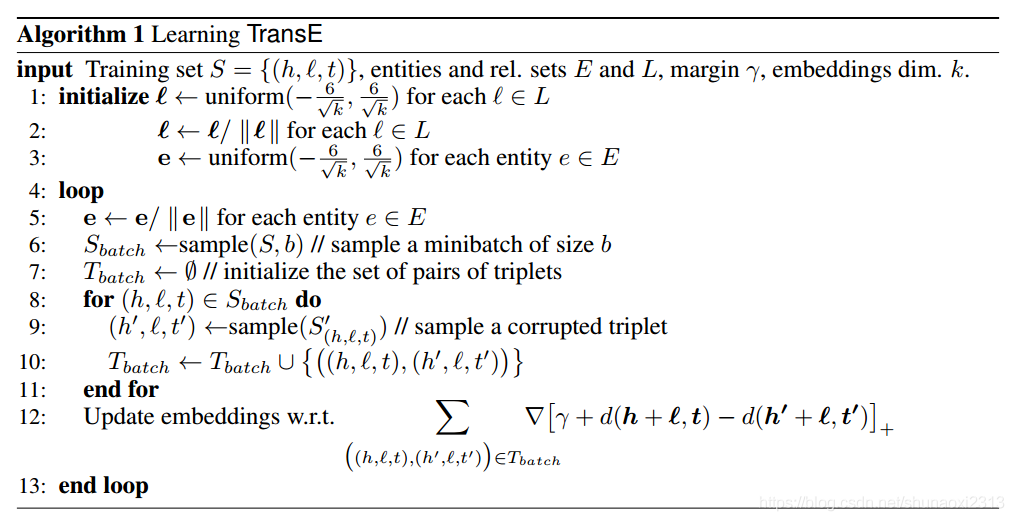
1. 方法

TransE模型认为正确的三元组，h代表头实体的向量，r代表关系的向量，t代表尾实体的向量。需要满足h+r=t，即尾实体是头实体通过关系平移得到的。

期望正确的三元组的距离越小越好，错误的三元组距离越大越好。因此目标函数为：



下面是算法的过程：



在用python实现的过程中。首先是训练transE，在update\_embeddings函数中有一个deepcopy函数，目的是为了批量更新。update\_embeddings函数中，要对correct triplet和corrupted triplet都进行更新。

对于每一次迭代，每一次的归一化约束实体长度为1，使得收敛有效，但对关系不做此要求。

通过SGD更新词向量时，会同时更新Correct triple和Corrupted triple。这两个triple中其实只有一个实体不同，因此另一个实体就需要更新两次，需要使用同一个，不然后一次更新的结果会将前一次的更新结果覆盖掉。

当优化目标取得最小值的时候，程序结束。

import codecs  
import random  
import math  
import numpy as np  
import copy  
import time  
  
entity2id = {}  
relation2id = {}  
  
  
def data\_loader(file):  
 file1 = file + "train.txt"  
 file2 = file + "entity2id.txt"  
 file3 = file + "relation2id.txt"  
  
 with open(file2, 'r') as f1, open(file3, 'r') as f2:  
 lines1 = f1.readlines()  
 lines2 = f2.readlines()  
 for line in lines1:  
 line = line.strip().split('\t')  
 if len(line) != 2:  
 continue  
 entity2id[line[0]] = line[1]  
  
 for line in lines2:  
 line = line.strip().split('\t')  
 if len(line) != 2:  
 continue  
 relation2id[line[0]] = line[1]  
  
 entity\_set = set()  
 relation\_set = set()  
 triple\_list = []  
  
 with codecs.open(file1, 'r') as f:  
 content = f.readlines()  
 for line in content:  
 triple = line.strip().split("\t")  
 if len(triple) != 3:  
 continue  
  
 h\_ = entity2id[triple[0]]  
 t\_ = entity2id[triple[1]]  
 r\_ = relation2id[triple[2]]  
  
 triple\_list.append([h\_,t\_,r\_])  
  
 entity\_set.add(h\_)  
 entity\_set.add(t\_)  
  
 relation\_set.add(r\_)  
  
 return entity\_set, relation\_set, triple\_list  
  
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))  
  
class TransE:  
 def \_\_init\_\_(self,entity\_set, relation\_set, triple\_list,  
 embedding\_dim=100, learning\_rate=0.01, margin=1,L1=True):  
 self.embedding\_dim = embedding\_dim  
 self.learning\_rate = learning\_rate  
 self.margin = margin  
 self.entity = entity\_set  
 self.relation = relation\_set  
 self.triple\_list = triple\_list  
 self.L1=L1  
  
 self.loss = 0  
  
 def emb\_initialize(self):  
 relation\_dict = {}  
 entity\_dict = {}  
  
 for relation in self.relation:  
 r\_emb\_temp = np.random.uniform(-6/math.sqrt(self.embedding\_dim) ,  
 6/math.sqrt(self.embedding\_dim) ,  
 self.embedding\_dim)  
 relation\_dict[relation] = r\_emb\_temp / np.linalg.norm(r\_emb\_temp,ord=2)  
  
 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)  
 entity\_dict[entity] = e\_emb\_temp / np.linalg.norm(e\_emb\_temp,ord=2)  
  
 self.relation = relation\_dict  
 self.entity = entity\_dict  
  
 def train(self, epochs):  
 nbatches = 400  
 batch\_size = len(self.triple\_list) // nbatches  
 print("batch size: ", batch\_size)  
 for epoch in range(epochs):  
 start = time.time()  
 self.loss = 0  
  
 for k in range(nbatches):  
 # Sbatch:list  
 Sbatch = random.sample(self.triple\_list, batch\_size)  
 Tbatch = []  
  
 for triple in Sbatch:  
 # 每个triple选3个负样例  
 # for i in range(3):  
 corrupted\_triple = self.Corrupt(triple)  
 if (triple, corrupted\_triple) not in Tbatch:  
 Tbatch.append((triple, corrupted\_triple))  
 self.update\_embeddings(Tbatch)  
  
  
 end = time.time()  
 print("epoch: ", epoch , "cost time: %s"%(round((end - start),3)))  
 print("loss: ", self.loss)  
  
 #保存临时结果  
 if epoch % 20 == 0:  
 with codecs.open("entity\_temp", "w") as f\_e:  
 for e in self.entity.keys():  
 f\_e.write(e + "\t")  
 f\_e.write(str(list(self.entity[e])))  
 f\_e.write("\n")  
 with codecs.open("relation\_temp", "w") as f\_r:  
 for r in self.relation.keys():  
 f\_r.write(r + "\t")  
 f\_r.write(str(list(self.relation[r])))  
 f\_r.write("\n")  
  
 print("写入文件...")  
 with codecs.open("entity\_50dim\_batch400", "w") as f1:  
 for e in self.entity.keys():  
 f1.write(e + "\t")  
 f1.write(str(list(self.entity[e])))  
 f1.write("\n")  
  
 with codecs.open("relation50dim\_batch400", "w") as f2:  
 for r in self.relation.keys():  
 f2.write(r + "\t")  
 f2.write(str(list(self.relation[r])))  
 f2.write("\n")  
 print("写入完成")  
  
  
 def Corrupt(self,triple):  
 corrupted\_triple = copy.deepcopy(triple)  
 seed = random.random()  
 if seed > 0.5:  
 # 替换head  
 rand\_head = triple[0]  
 while rand\_head == triple[0]:  
 rand\_head = random.sample(self.entity.keys(),1)[0]  
 corrupted\_triple[0]=rand\_head  
  
 else:  
 # 替换tail  
 rand\_tail = triple[1]  
 while rand\_tail == triple[1]:  
 rand\_tail = random.sample(self.entity.keys(), 1)[0]  
 corrupted\_triple[1] = rand\_tail  
 return corrupted\_triple  
  
 def update\_embeddings(self, Tbatch):  
 copy\_entity = copy.deepcopy(self.entity)  
 copy\_relation = copy.deepcopy(self.relation)  
  
 for triple, corrupted\_triple in Tbatch:  
 # 取copy里的vector累积更新  
 h\_correct\_update = copy\_entity[triple[0]]  
 t\_correct\_update = copy\_entity[triple[1]]  
 relation\_update = copy\_relation[triple[2]]  
  
 h\_corrupt\_update = copy\_entity[corrupted\_triple[0]]  
 t\_corrupt\_update = copy\_entity[corrupted\_triple[1]]  
  
 # 取原始的vector计算梯度  
 h\_correct = self.entity[triple[0]]  
 t\_correct = self.entity[triple[1]]  
 relation = self.relation[triple[2]]  
  
 h\_corrupt = self.entity[corrupted\_triple[0]]  
 t\_corrupt = self.entity[corrupted\_triple[1]]  
  
 if self.L1:  
 dist\_correct = distanceL1(h\_correct, relation, t\_correct)  
 dist\_corrupt = distanceL1(h\_corrupt, relation, t\_corrupt)  
 else:  
 dist\_correct = distanceL2(h\_correct, relation, t\_correct)  
 dist\_corrupt = distanceL2(h\_corrupt, relation, t\_corrupt)  
  
 err = self.hinge\_loss(dist\_correct, dist\_corrupt)  
  
 if err > 0:  
 self.loss += err  
  
 grad\_pos = 2 \* (h\_correct + relation - t\_correct)  
 grad\_neg = 2 \* (h\_corrupt + relation - t\_corrupt)  
  
 if self.L1:  
 for i in range(len(grad\_pos)):  
 if (grad\_pos[i] > 0):  
 grad\_pos[i] = 1  
 else:  
 grad\_pos[i] = -1  
  
 for i in range(len(grad\_neg)):  
 if (grad\_neg[i] > 0):  
 grad\_neg[i] = 1  
 else:  
 grad\_neg[i] = -1  
  
 # head系数为正，减梯度；tail系数为负，加梯度  
 h\_correct\_update -= self.learning\_rate \* grad\_pos  
 t\_correct\_update -= (-1) \* self.learning\_rate \* grad\_pos  
  
 # corrupt项整体为负，因此符号与correct相反  
 if triple[0] == corrupted\_triple[0]: # 若替换的是尾实体，则头实体更新两次  
 h\_correct\_update -= (-1) \* self.learning\_rate \* grad\_neg  
 t\_corrupt\_update -= self.learning\_rate \* grad\_neg  
  
 elif triple[1] == corrupted\_triple[1]: # 若替换的是头实体，则尾实体更新两次  
 h\_corrupt\_update -= (-1) \* self.learning\_rate \* grad\_neg  
 t\_correct\_update -= self.learning\_rate \* grad\_neg  
  
 #relation更新两次  
 relation\_update -= self.learning\_rate\*grad\_pos  
 relation\_update -= (-1)\*self.learning\_rate\*grad\_neg  
  
  
 #batch norm  
 for i in copy\_entity.keys():  
 copy\_entity[i] /= np.linalg.norm(copy\_entity[i])  
 for i in copy\_relation.keys():  
 copy\_relation[i] /= np.linalg.norm(copy\_relation[i])  
  
 # 达到批量更新的目的  
 self.entity = copy\_entity  
 self.relation = copy\_relation  
  
 def hinge\_loss(self,dist\_correct,dist\_corrupt):  
 return max(0,dist\_correct-dist\_corrupt+self.margin)  
  
  
if \_\_name\_\_=='\_\_main\_\_':  
 file1 = "FB15k\\"  
 entity\_set, relation\_set, triple\_list = data\_loader(file1)  
 print("load file...")  
 print("Complete load. entity : %d , relation : %d , triple : %d" % (len(entity\_set),len(relation\_set),len(triple\_list)))  
  
 transE = TransE(entity\_set, relation\_set, triple\_list,embedding\_dim=50, learning\_rate=0.01, margin=1,L1=True)  
 transE.emb\_initialize()  
 transE.train(epochs=1001)

1. 指标

isFit参数用于区分raw和filter。filter会非常慢。

在测试时，以一个三元组为例，用语料中所有实体替换当前三元组的头实体计算距离，将结果按升序排序，用正确三元组的排名情况来评估学习效果（同理对尾实体这样做）。在替换后要使用filter训练方法，检查一下新三元组是否出现在训练集中，是的话就删掉。

1. 结论

最后，我们的输出就是用两个向量，去预测另一个向量，达到了预测的效果，达到了我们的目标。

知识库中的实体关系类型可分为一对一、一对多、多对一、多对多4种类型，而复杂关系主要指的是一对多、多对一、多对多的3种关系类型。TransE最大的优点就是简单并且容易扩展，但是缺点是存在多个实体在同一个空间竞争多个点的情况，不适合处理一对多或者多对多的关系，并且没有完全考虑语义信息。TransE在复杂关系的表示上面表现比较差。