# TransE复现和实验结果汇报

## TransE复现和实验结果汇报

### 1. 伪代码以及代码流程

**Algorithm 1** Learning TransE

**input** Training set $S = \{(h, \ell, t)\}$, entities and rel. sets $E$ and $L$, margin $\gamma$, embeddings dim. $k$.

1: **initialize** $\boldsymbol{\ell} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$
2: $\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$
3: $\mathbf{e} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$
4: **loop**
5: $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$
6: $S_{batch} \leftarrow$ sample$(S, b)$ // sample a minibatch of size $b$
7: $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets
8: **for** $(h, \ell, t) \in S_{batch}$ **do**
9: $(h', \ell, t') \leftarrow$ sample$(S'_{(h,\ell,t)})$ // sample a corrupted triplet
10: $T_{batch} \leftarrow T_{batch} \cup \{((h, \ell, t), (h', \ell, t'))\}$
11: **end for**
12: Update embeddings w.r.t. $\sum_{((h,\ell,t),(h',\ell,t')) \in T_{batch}} \nabla \left[\gamma + d(\boldsymbol{h} + \boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{\ell}, \boldsymbol{t'})\right]_+$
13: **end loop**

代码流程为:

1：初始化： 对于关系按照1的初始化方式初始化即可

2：这里进行了L2范数归一化，也就是除以自身的L2范数

3：同理，也对实体进行了初始化，但是这里没有除以自身的L2范数

4：训练的循环过程中：

5：首先对实体进行了L2范数归一化

6：取一个batch的样本，这里Sbatch代表的是正样本，也就是正确的三元组

7： 初始化三元组对，应该就是创造一个用于储存的列表

8，9，10：这里的意思应该是根据Sbatch的正样本替换头实体或者尾实体构造负样本，然后把对应的正样本三元组和负样本三元组放到一起，组成Tbatch

11：完成正负样本的提取

12：根据梯度下降更新向量

13：结束循环

其中选择L2范数，梯度下降中求导为：

$$\frac{\partial loss}{\partial h} = \frac{\partial[\gamma + (h+r-t)^2 - (h^{'}+r-t^{'})^2]_+}{\partial h}$$
$$= \begin{cases} 2(h+r-t) & \text{if } \gamma + (h+r-t)^2 - (h^{'}+r-t^{'})^2 \geq 0 \\ 0 & \text{if } \gamma + (h+r-t)^2 - (h^{'}+r-t^{'})^2 < 0 \end{cases}$$

## 部分C++代码展示

```
relation_tmp=relation_vec;
entity_tmp = entity_vec;
for (int k=0; k<batchsize; k++)
{
    int i=rand_max(fb_h.size());
    int j=rand_max(entity_num);
    double pr = 1000*right_num[fb_r[i]]/(right_num[fb_r[i]]+left_num[fb_r[i]]);
    if (method ==0)
        pr = 500;
    if (rand()%1000<pr)
    {
        while (ok[make_pair(fb_h[i],fb_r[i])].count(j)>0)
            j=rand_max(entity_num);
        train_kb(fb_h[i],fb_l[i],fb_r[i],fb_h[i],j,fb_r[i]);
    }
    else
    {
        while (ok[make_pair(j,fb_r[i])].count(fb_l[i])>0)
            j=rand_max(entity_num);
        train_kb(fb_h[i],fb_l[i],fb_r[i],j,fb_l[i],fb_r[i]);
    }
    norm(relation_tmp[fb_r[i]]);
    norm(entity_tmp[fb_h[i]]);
    norm(entity_tmp[fb_l[i]]);
    norm(entity_tmp[j]);
}
```

随机提取正样本，然后随机选择实体进行代替，得到负样本，然后将正负样本一起加入Tbatch中，进行对应的求偏导和梯度下降。

```
void train_kb(int e1_a,int e2_a,int rel_a,int e1_b,int e2_b,int rel_b)
{
    double sum1 = calc_sum(e1_a,e2_a,rel_a);
    double sum2 = calc_sum(e1_b,e2_b,rel_b);
    if (sum1+margin>sum2)
    {
        res+=margin+sum1-sum2;
        gradient( e1_a, e2_a, rel_a, e1_b, e2_b, rel_b);
    }
}
```

```
void gradient(int e1_a,int e2_a,int rel_a,int e1_b,int e2_b,int rel_b)
{
    for (int ii=0; ii<n; ii++)
    {

        double x = 2*(entity_vec[e2_a][ii]-entity_vec[e1_a][ii]-relation_vec[rel_a][ii]);
        if (L1_flag)
            if (x>0)
                x=1;
            else
                x=-1;
        relation_tmp[rel_a][ii]-=-1*rate*x;
        entity_tmp[e1_a][ii]-=-1*rate*x;
        entity_tmp[e2_a][ii]+=-1*rate*x;
        x = 2*(entity_vec[e2_b][ii]-entity_vec[e1_b][ii]-relation_vec[rel_b][ii]);
        if (L1_flag)
            if (x>0)
                x=1;
            else
                x=-1;
        relation_tmp[rel_b][ii]-=rate*x;
        entity_tmp[e1_b][ii]-=rate*x;
        entity_tmp[e2_b][ii]+=rate*x;
    }
}
```

梯度下降的实现，目标是让正样本的loss越小，负样本的loss越大。

```
cout<<"epoch:"<<epoch<<' '<<res<<endl;
FILE* f2 = fopen(("relation2vec."+version).c_str(),"w");
FILE* f3 = fopen(("entity2vec."+version).c_str(),"w");
for (int i=0; i<relation_num; i++)
{
    for (int ii=0; ii<n; ii++)
        fprintf(f2,"%.6lf\t",relation_vec[i][ii]);
    fprintf(f2,"\n");
}
for (int i=0; i<entity_num; i++)
{
    for (int ii=0; ii<n; ii++)
        fprintf(f3,"%.6lf\t",entity_vec[i][ii]);
    fprintf(f3,"\n");
}
fclose(f2);
fclose(f3);
```

不断训练，让loss收敛，最后得到一个映射每个实体的向量空间

## 测试

用训练好的模型做Link prediction，预测三元组中缺失的部分，例如给定关系和尾实体，预测头实体。其本质上都是预测向量。

其中预测的指标为Mean Rank和Hit@10，分为头实体预测和尾实体预测两种。训练集也分为Raw和Filter

```
int h = fb_h[testid];
int l = fb_l[testid];
int rel = fb_r[testid];
rel_num[rel]+=1;
vector<pair<int,double> > a;
for (int i=0; i<entity_num; i++)
{
    double sum = calc_sum(i,l,rel);
    a.push_back(make_pair(i,sum));
}
sort(a.begin(),a.end(),cmp);
```

```
if (a[i].first ==h)
{
    ve.push_back((info){id2entity[h],id2entity[l],id2relation[rel],(int)a.size()-i});
    lsum+=a.size()-i;
    lsum_filter+=filter+1;
    lsum_r[rel]+=a.size()-i;
    lsum_filter_r[rel]+=filter+1;
    if (a.size()-i<=10)
    {
        lp_n+=1;
        lp_n_r[rel]+=1;
    }
    if (filter<10)
    {
        lp_n_filter+=1;
        lp_n_filter_r[rel]+=1;
    }
    break;
}
```

首先进行一次测试，预测三元组的头和尾

```
minari@MinarideMacBook-Air TransE % ./Test_TransE bern
1345 14951
left:305.626    0.476545        183.382 0.678861
right:168.717   0.555687        92.477  0.749708
```

将得分排名低于1000的三元组标记为正样本，加入到训练集中重新训练，然后再进行一次测试

```
minari@MinarideMacBook-Air TransE % ./Test_TransE bern
1345 14951
left:279.677    0.484383        165.183 0.688781
right:142.731   0.566928        69.7852 0.759984
```

将得分排名低于100的三元组标记为正样本，加入到训练集中

```
minari@MinarideMacBook-Air TransE % ./Test_TransE bern
1345 14951
left:263.301    0.527738        155.251 0.757969
right:133.601   0.6114  65.7861 0.821825
```

可以发现训练完之后，测试的效果越来越好