知识图谱嵌入--第一阶段说明文档

1. 文档概述

1.1 文档介绍

该文档基于期末《知识图谱》课程增量包，任务为任务一的 1 号子任务，实现了两组 TransE, TransH, TransR 的训练与测试, 并撰写了以下的说明报告

1.2 结果概览（详细结果参照文档内部 4 结果部分）：

总：

F 数据集下： **6 个 Trans 平均：** **LP：0.506** **EP：0.156**

W 数据集下： **6 个 Trans 平均：** **LP：0.558** **EP：0.201**

FB15k-237 数据集下系列 A 结果：

	TransE	TransH	TransR
LP	0.4838	0.5146	0.4726
EP	0.1694	0.1627	0.2213

FB15k-237 数据集下系列 B 结果：

	TransE	TransH	TransR
LP	0.5638	0.4017	0.5314
EP	0.0674	0.2038	0.1680

WN18 数据集数据集下系列 A 结果：

	TransE	TransH	TransR
LP	0.7509	0.6966	0.3098
EP	0.3551	0.1875	0.1679

WN18 数据集数据集下系列 B 结果：

	TransE	TransH	TransR
LP	0.5638	0.5362	0.5218

EP	0.2007	0.1004	0.1108

FB15k-237 数据集下系列 A&系列 B 平均结果：

	TransE	TransH	TransR
LP	0.4823	0.5348	0.5013
EP	0.1617	0.1959	0.1954

WN18 数据集下系列 A&系列 B 平均结果：

	TransE	TransH	TransR
LP	0.643	0.6169	0.415
EP	0.268	0.194	0.139

2. 开源数据集介绍

2.1 FB15k-237 数据集介绍：

下载地址：https://github.com/TimDettmers/ConvE.git

简介：FB15k-237 是 FB15k 数据集的一个扩展版本，由 Toutanova 等人在 2015 年发布。与 FB15k 不同，FB15k-237 只包含 237 个关系，但是包含更多的三元组（共 310,116 个三元组），其中包括 272,115 个训练三元组、17,535 个验证三元组和 20,466 个测试三元组。FB15k-237 的关系是从原始的 FB15k 数据集中筛选出来的，保留了那些至少有 50 个训练三元组的关系。

图 1 数据集下载

与原始数据集对比：

关系数量不同：FB15K 包含了 15000 个三元组，涵盖了约 1400 个关系，而

FB15K-237 只包含了 237 个关系，但是有着更多的三元组（共 310,116 个三元组）。

关系的选取不同：FB15K 包含了 Freebase 数据集中出现的所有关系，而 FB15K-237 从 FB15K 中筛选出了那些至少有 50 个训练三元组的关系，保留了 237 个关系

难度程度不同：FB15K-237 在关系数量上更少但三元组数量更多

数据集展示

我在做的过程中直接选取了已经映射有 ID 的实体数据集和关系数据集，但训练与测试并无

主题词总数	14505
三元组总数	544230
关系的种类数	474
每个主题词的平均三元组数	37.5
每个主题词的平均关系数	10.3
每个关系连接的平均实体数	3.57

图 2 统计数据

数据集中的关系数远小于实体数，这也是 LP 效果比较好的原因（分析）

统计属性\集合	train	valid	test
主题词总数	13781	7652	8171
三元组总数	272115	17535	20466
关系的种类数	237	223	224
每个主题词的平均三元组数	19.75	2.29	2.50
每个主题词的平均关系数	6.78	1.58	1.68
每个关系连接的平均实体数	2.91	1.45	1.49

图 3 内部数据集对比

这个数据集最大的问题就是测试集里有很多训练集里缺失的实体，这导致在推理过程中我不得不添加过滤策略

任务	数据集	容量	缺失问题实体	缺失边	缺失答案实体
预测尾实体	test	20466	75	4806	15585
	valid	17535	44	4150	13341
预测头实体	test	20466	87	1913	18466
	valid	17535	60	1531	15944

图 4 数据问题集展示

以下是我拿到的使用的数据集的展示，头实体+尾实体+关系的组合，用制表符分割的

```
/m/027rn    /m/06cx9    /location/country/form_of_government
/m/017dcd   /m/06v8s0   /tv/tv_program/regular_cast./tv/regular_tv_appearance/actor
/m/07s9rl0  /m/0170z3   /media_common/netflix_genre/titles
/m/01sl1q   /m/044mz_   /award/award_winner/awards_won./award/award_honor/award_winner
/m/0cnk2q   /m/02nzb8   /soccer/football_team/current_roster./sports/sports_team_roster/position
/m/02_j1w   /m/01cwm1   /sports/sports_position/players./soccer/football_roster_position/team
/m/059ts    /m/03h_f4   /government/political_district/representatives./government/government_pos
/m/011yn5   /m/01pjr7   /film/film/starring./film/performance/actor
```

图 5 训练集（测试集和验证集类似）

至于实体和关系的数据表则是实体名/关系名+ID 的组合

```
/people/appointed_role/appointment./people/appointment/appointed_by 0
/location/statistical_region/rent50_2./measurement_unit/dated_money_value/currency  1
/tv/tv_series_episode/guest_stars./tv/tv_guest_role/actor   2
/music/performance_role/track_performances./music/track_contribution/contributor    3
/medicine/disease/prevention_factors    4
/organization/organization_member/member_of./organization/organization_membership/organization  5
/american_football/football_player/receiving./american_football/player_receiving_statistics/season  6
```

图 6 关系 ID 映射数据（实体集类似）

数据集总结：

我选择这个数据集主要还是因为它公开认可度高，同时数据量足够

2.2 WN18 数据集介绍

下载地址：

hhttps://download.csdn.net/download/qq_21097885/11013930

简介：它是从 WordNet 语义词典中提取的子集，包括 WordNet 中的 18 种语义关系，作为一个大型的英语词汇数据库；名词、动词、形容词和副词被分成同义词组，也称为认知同义词，每个同义词表达一个不同的上下文概念。然后，同义词集通过概念语义和词汇关系相互关联

对比特点：

重复和冗余：由于 WN18 是从 WordNet 提取的，存在许多反转关系，例如 (A, hypernym, B) 和 (B, hyponym, A)，这使得某些任务的挑战性降低。

泛化能力不足：由于三元组间的强关联性，很难预测

关系固定：18 个语义关系，和 FB15k-237 相比很固定

数据展示

类别	数量
#Relation	18
#Entity	40,943
#Train	141,442
#Valid	2,500
#Test	2,500

图 7 数据集结构展示

```
1    _member_of_domain_topic 0
2    _member_meronym 1
3    _derivationally_related_form    2
4    _member_of_domain_region    3
5    _similar_to 4
6    _hypernym   5
7    _member_holonym 6
8    _instance_hypernym  7
```

图 8 关系 ID 映射集（实体类似）

```
06845599    _member_of_domain_usage 03754979
00789448    _verb_group 01062739
10217831    _hyponym    10682169
08860123    _member_of_domain_region    05688486
02233096    _member_meronym 02233338
01459242    _part_of    01461646
```

图 9 训练集（测试集与验证集类似）

总结：这个数据集我用的还是字符串版本，和 FB15k 差不多我都看不太懂，直接用了

3. 模型说明

3.1 整体介绍

Translate 系列模型是知识图谱嵌入（Knowledge Graph Embedding, KGE）领域的重要一类方法，其核心思想是将知识图谱中的实体和关系表示为低维向量，以便用于任务如链接预测和知识图谱补全。

以下是 Translate 系列模型的基本总结：

Translate 系列的用途

知识图谱嵌入：将实体和关系映射到连续的向量空间，便于计算和推理。

链接预测：预测知识图谱中缺失的关系（例如，给定 (head,relation,?)(head,relation, ?)(head,relation,?) 或 (?,relation,tail)(?, relation, tail)(?,relation,tail)）

知识图谱补全：通过嵌入模型推断缺失的实体或关系，扩展现有的知识图谱

关系分类：预测两个实体之间的语义关系类型

其他任务支持：如实体分类、实体对齐，以及与其他机器学习任务的集成

Translate 系列模型的共同点：

基于向量表示：所有模型都将实体和关系表示为向量

目标函数：通过设计评分函数来评估三元组的合理性，最小化正确三元组与错误三元组的评分差距

优化方法：常用随机梯度下降（SGD）或其变体进行优化

嵌入学习：通过学习实体和关系的向量表示捕获知识图谱的结构和语义

Translate 系列的主要贡献

提供了一种高效的知识表示方法，使得知识图谱的推理和补全更加方便

简化了知识图谱的计算操作（从逻辑推理转化为向量操作）

为后续更复杂的知识图谱嵌入模型奠定了理论基础

将实体向量表示（Embedding）在低维稠密向量空间中，然后进行计算和推理

3.2 原理说明

TransE

TransE, NIPS2013, Translating embeddings for modeling multi-relational

data

TransE 向量空间假设：

TransE 对三元组（h， r， t）中的实体和关系映射到向量空间作了一些假设：每一个三元组（h， r， t）都能表示为（h， r， t），其中，h 是指头实体的向量表示， r 是指关系的向量表示， t 是指尾实体的向量表示

通过不断调整来构建三元组（其中三元组的实体和关系都来自源知识图谱），来发现向量空间中，头实体向量 h 加上关系 r 等于尾实体向量 t 的三元组，这个过程称为翻译，如果在向量空间这种关系成立，就说明三元组（h， r， t）所代表的知识表示可以看作是正确的，以此来发现实体间新的关系

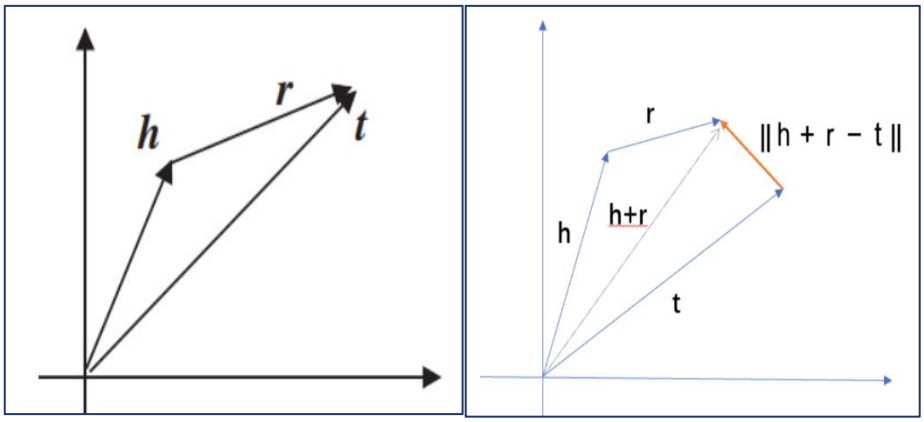


图 10 TransE 推理示意图

用我自己的理解，TransE 的功用在于：

① 通过词向量表示知识图谱中已存在的三元组(所以 TransE 可以看作知识表示方法)

② 扩大知识图谱中的关系网，扩充构建多元关系数据。其中，关系数据包括单一关系数据(single-relational data)和多元关系数据(multi-relational data)（单一关系通常是结构化的，可以直接进行简单的推理，多元关系则依赖于多种类型的实体和关系，因此需要一种通用的方法能够同时考虑异构关系）（所以 TransE 可以看作“活的”或“更高级的”的知识表示方法，因为可以做知识图谱上的链接预测，或作为媒介完善知识图谱）

$$f(\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}) = \| \boldsymbol{h}+\boldsymbol{r}-\boldsymbol{t} \|_{L_1/L_2}$$

图 11 TransE 算分公式

Algorithm 1 Learning TransE

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

initialize $\boldsymbol{\ell} \leftarrow$ uniform$(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$

$\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$

$\mathbf{e} \leftarrow$ uniform$(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$

loop

  $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$

  $S_{batch} \leftarrow$sample(S, b) // sample a minibatch of size b

  $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets

  for $(h,\ell,t) \in S_{batch}$ **do**

    $(h',\ell,t') \leftarrow$sample$(S'_{(h,\ell,t)})$ // sample a corrupted triplet

    $T_{batch} \leftarrow T_{batch} \cup \{((h,\ell,t),(h',\ell,t'))\}$

  end for

  Update embeddings w.r.t. $\sum_{((h,\ell,t),(h',\ell,t')) \in T_{batch}} \nabla[\gamma + d(\boldsymbol{h}+\boldsymbol{\ell},\boldsymbol{t}) - d(\boldsymbol{h'}+\boldsymbol{\ell},\boldsymbol{t'})]_+$

end loop

图 12 TransE 算法

TransH

TransH, AAAI2014, Knowledge graph embedding by translating on hyperplanes 头实体向量 h 和尾实体向量 t 并非投影到 xoy 平面，而是将每种关系都建模为一个超平面（这样可以使得实体及其关系之间的表示方法多元化，解决实体间无法分出语义的缺陷），将三元组中的头实体和尾实体分别映射到该超平面中，TransH 是将关系解释为超平面上的翻译操作。其中每个关系都有两个向量，超平面的范数向量 wr（用于计算映射后的实体向量）和超平面上的平移向量 dr

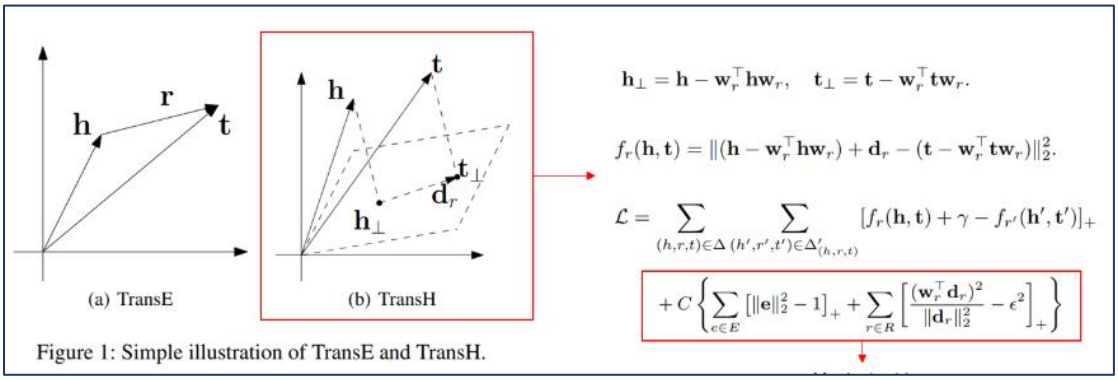


Figure 1: Simple illustration of TransE and TransH.

图 13 TransH 示意图

其中得分函数和损失函数的设计和 TransE 的形式是一样的，不同的是，是损失函数中加入了约束条件，另外负采样方式也进行了完善：

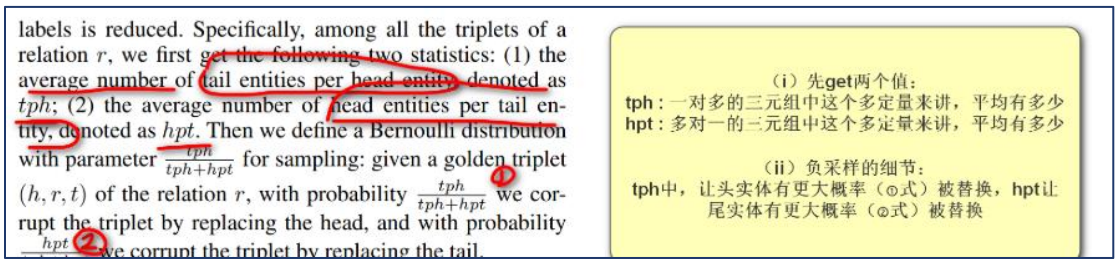


图 14 TransH 改进

上边的替换策略，比如一对多中，增加了头实体替换的概率，从而来构造负例，这样是有好处的，反过来试想，如果替换尾实体，很容易让我们文中第一个图那样，t1 替换成 t2 或 t3 或 t4，对于识别它们本身的特异性是不利的，或者这样的负采样质量不高。另一个改进是在三维中的设计，由于投影到某《关系超平面》后成立的向量，其原向量可以不同，由此改进了 TransE 的固有缺陷

TransR

TransR, AAAI2015, Learning Entity and Relation Embeddings for Knowledge Graph Completion

虽然 TransH 模型使每个实体在不同关系下拥有了不同的表示，它仍然假设实体和关系处于相同的语义空间中，这一定程度上限制了 TransH 的表示能力。TransR 模型对不同的关系建立各自的关系空间，在计算时先将实体映射到关系空间进行计算（不同的 r 用不同的投影矩阵 Mr），然后再建立从头实体到尾实体的翻译关系

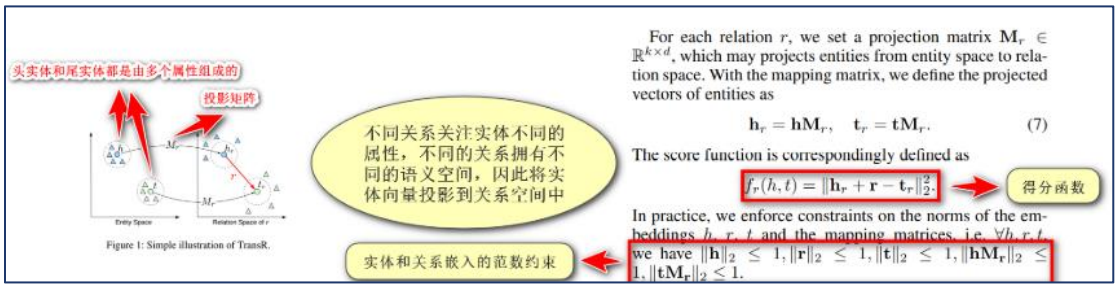


图 15 TransR 改进

它的损失函数进行了比较直观的调整

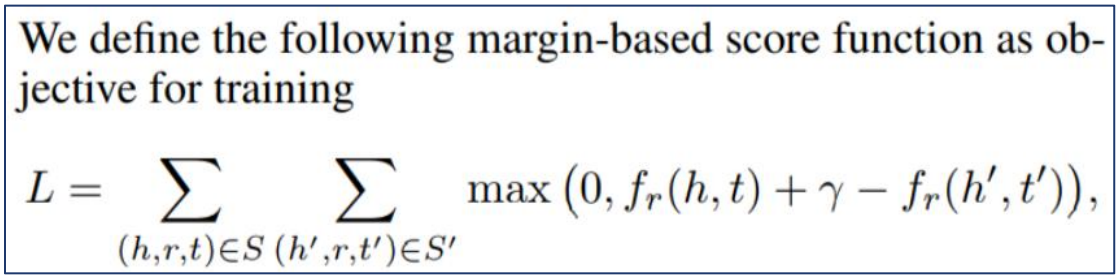


图 16 TransR 算分公式

TransR 和 TransH 的区别：

TransR 是直接转换在关系空间中探究，转换后的空间维度可能与之前不同，但更为直观

TransH 是在超平面上进行的探究，仍然在公共的语义空间进行，即实体和关系仍然都在同一三维空间中嵌入

核心代码

3.3 代码注解

我这里是两个版本的 Trans 系列，第二个版本 B 是为了弥补第一阶段训练和推理很慢的遗憾，是使用纯 python 的版本，第一个版本 A 是使用 pytorch 来完整计算的结果，主要还是为了维度而产生的。

3.4 模型代码

系列 A（使用 pytorch 来进行张量计算）

TransE

```
class TransE(nn.Module):
    def __init__(self, num_entities, num_relations, embedding_dim):
        super(TransE, self).__init__()
        self.entity_embeddings = nn.Embedding(num_entities, embedding_dim)
        self.relation_embeddings = nn.Embedding(num_relations, embedding_dim)
        self.embedding_dim = embedding_dim
        self.initialize_embeddings()
```

目的：构造模型并初始化实体和关系的嵌入向量。

num_entities: 实体的数量，定义了嵌入矩阵的大小

num_relations: 关系的数量，定义了嵌入矩阵的大小

embedding_dim: 嵌入的维度，决定了每个实体和关系的表示空间

super(TransE, self).__init__(): 调用父类的初始化方法

self.entity_embeddings: 实体嵌入矩阵，使用 nn.Embedding 来存储所有实体的嵌入向量。每个实体会被表示为一个 embedding_dim 维的向量

self.relation_embeddings: 关系嵌入矩阵，存储所有关系的嵌入向量

self.embedding_dim: 保存嵌入的维度

self.initialize_embeddings(): 调用初始化函数来随机初始化实体和关系的嵌入向量

```
def initialize_embeddings(self):
    """初始化嵌入向量"""
    nn.init.uniform_(self.entity_embeddings.weight, a=-6/np.sqrt(self.embedding_dim), b=6/np.sqrt(self.embedding_dim))
    nn.init.uniform_(self.relation_embeddings.weight, a=-6/np.sqrt(self.embedding_dim), b=6/np.sqrt(self.embedding_dim
```

目的：使用均匀分布初始化嵌入向量。

nn.init.uniform_: 使用均匀分布来初始化嵌入矩阵的权重。均匀分布的区间是 $[-6/\sqrt{d}, 6/\sqrt{d}]$，其中 d 是嵌入维度。该初始化方法有助于防止初始嵌入向量过大或过小。

self.entity_embeddings.weight: 访问并初始化实体嵌入的权重。

self.relation_embeddings.weight: 访问并初始化关系嵌入的权重

```python
def forward(self, head, relation, tail):
    """计算正样本和负样本之间的距离"""
    head_emb = self.entity_embeddings(head)
    relation_emb = self.relation_embeddings(relation)
    tail_emb = self.entity_embeddings(tail)
    return torch.norm(head_emb + relation_emb - tail_emb, p=2, dim=1)
```

目的：计算模型的输出，即正样本和负样本之间的距离

head, relation, tail: 输入的头实体、关系和尾实体。这些输入通常是正样本的头、关系和尾的 ID

head_emb, relation_emb, tail_emb: 通过查找表（self.entity_embeddings 和 self.relation_embeddings）获取头、关系和尾的嵌入向量

head_emb + relation_emb - tail_emb: 根据 TransE 模型的公式，head + relation - tail 是模型计算的一个向量表示。对于正确的三元组（正样本），该向量应该接近于零

torch.norm: 计算 head_emb + relation_emb - tail_emb 向量的 L2 范数，即欧几里得距离。该距离度量了三元组的嵌入向量之间的相似性或误差

p=2: 表示计算 L2 范数

dim=1: 按行计算范数，返回每个三元组的距离

TransH

改进的地方

self.relation_norms = nn.Embedding(num_relations, embedding_dim)

TransH 中，每个关系不仅有一个嵌入向量（relation_embeddings），还引入了一个超平面向量（relation_norms），它的作用是定义每个关系的超平面。该超平面用于约束实体在该关系下的表示。

relation_norms 是一个嵌入矩阵，其维度为 num_relations x embedding_dim，即每个关系都有一个嵌入向量和一个对应的超平面向量

```python
def project_to_relation_space(self, entity_emb, relation_emb, relation_norm):
    """将实体嵌入投影到关系超平面"""
    # 计算实体与关系超平面的投影
    norm_relation_emb = relation_norm / relation_norm.norm(p=2, dim=1, keepdim=True)  # 归一化关系的超平面
    projection = entity_emb - (entity_emb * norm_relation_emb).sum(dim=1, keepdim=True) * norm_relation_emb
    return projection
```

投影操作：为了让实体的嵌入向量适应关系的超平面，TransH 引入了投影操作。通过将实体嵌入向量投影到关系的超平面上，调整实体在该关系下的表示。这解决了 TransE 模型中，嵌入空间在所有关系中统一的限制。

norm_relation_emb：对关系的超平面向量进行归一化，使其成为单位向量。

projection：计算实体嵌入向量相对于关系超平面的投影。即通过从实体嵌入中去除与超平面向量的成分，从而得到与关系超平面垂直的部分

```
# 投影到关系的超平面
head_proj = self.project_to_relation_space(head_emb, relation_emb, relation_norm)
tail_proj = self.project_to_relation_space(tail_emb, relation_emb, relation_norm)

# 计算距离
return torch.norm(head_proj + relation_emb - tail_proj, p=2, dim=1)
```

通过 project_to_relation_space 方法，head_emb 和 tail_emb 都被投影到关系的超平面上，从而得到 head_proj 和 tail_proj。这使得实体的表示更加符合关系的几何约束

TransR

改进的地方：

添加了关系的投影矩阵

```
self.entity_embeddings = nn.Embedding(num_entities, entity_dim)
self.relation_embeddings = nn.Embedding(num_relations, relation_dim)
self.projection_matrices = nn.Embedding(num_relations, entity_dim * relation_dim)
```

在 TransR 模型中，每个关系有一个投影矩阵，它将实体嵌入从实体空间（entity_dim 维）映射到关系空间（relation_dim 维）。projection_matrices 是一个嵌入矩阵，每个关系对应一个大小为 entity_dim * relation_dim 的投影矩阵。它将实体嵌入空间与关系嵌入空间的维度进行转换。

投影矩阵的初始化

```
nn.init.uniform_(self.projection_matrices.weight, a=-6 / np.sqrt(self.entity_dim),
                 b=6 / np.sqrt(self.entity_dim))
```

projection_matrices 的初始化是通过均匀分布进行的，范围确保投影矩阵初始时的权重合理。

投影到关系空间

```
proj_matrix = self.projection_matrices(relation).view(-1, self.relation_dim, self.entity_dim)
head_proj = torch.bmm(proj_matrix, head_emb.unsqueeze(-1)).squeeze(-1)
tail_proj = torch.bmm(proj_matrix, tail_emb.unsqueeze(-1)).squeeze(-1)
```

投影操作：每个关系都有一个投影矩阵，proj_matrix 是 relation 对应的投影矩阵，通过 view 将其调整为 relation_dim x entity_dim 的形状。
使用 torch.bmm（批量矩阵乘法）将投影矩阵与实体嵌入向量相乘，从而将实体嵌入向量投影到该关系的空间中。

总结与与 TransE、TransH 的区别：
区别 1：在 TransR 中，每个关系都有一个投影矩阵，将实体嵌入从实体空间映射到关系空间。这样，实体可以在每个关系上有不同的表示，增强了模型的灵活性。
区别 2：通过矩阵乘法（torch.bmm）实现了实体的投影操作，使得每个关系的嵌入能够影响实体的表示。这与 TransH 的投影方法不同，TransH 只是将实体投影到超平面，而 TransR 通过关系特定的投影矩阵对实体进行空间转换。
这些改进使得 TransR 模型能更细致地捕捉到关系之间的差异，为每个关系分配一个不同的投影矩阵，从而可以将实体在不同关系下的表示映射到不同的空间，提高了模型的表示能力。

系列 B(未使用 pytorch 或使用非必要功能)

```python
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
```

输入参数:

entity_set: 实体的集合

relation_set: 关系的集合

triple_list: 包含知识图谱中三元组 (head, relation, tail) 的列表

embedding_dim: 嵌入向量的维度（默认为 100）

learning_rate: 学习率，用于优化（默认为 0.01）

margin: TransE 中的 margin 值，用于损失函数的约束（默认为 1）

L1: 是否使用 L1 范数计算距离（布尔值，默认为 True）

属性:

self.embedding_dim: 嵌入维度。

self.learning_rate: 学习率

self.margin: 损失函数中的 margin 值。

self.entity: 初始化时的实体集合

self.relation: 初始化时的关系集合

self.triple_list: 三元组数据

self.L1: 指示使用 L1 范数（True）或 L2 范数（False）

self.loss: 损失值，初始化为 0

```python
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
```

功能:初始化实体和关系的嵌入向量，随机生成嵌入值，并对这些向量进行归一化处理。

步骤:创建空字典:

relation_dict: 用于存储关系及其对应的嵌入向量。

entity_dict: 用于存储实体及其对应的嵌入向量。

随机初始化嵌入向量:使用 np.random.uniform 生成范围之间的均匀分布随机数

归一化处理:对生成的向量进行 L2 范数归一化，使得嵌入向量的长度为 1

存储结果:将归一化后的关系嵌入存入 relation_dict，实体嵌入存入 entity_dict。

更新类属性:将 self.relation 和 self.entity 更新为嵌入后的字典

TransH

在 E 的基础上进行了改进操作：

```python
def project_on_hyperplane(self, entity_vector, relation_vector):
    projection = entity_vector - np.dot(entity_vector, relation_vector) * relation_vector
    return projection
```

功能: 将实体嵌入向量投影到超平面上。

使用向量投影公式：projection = e - (e . r) * r，其中 e 是实体向量，r 是关系的投影向量。

该操作使得实体在关系的超平面上，确保实体的嵌入不在关系的方向上。

```
def distanceL2(self, h, r, t):
    h_proj = self.project_on_hyperplane(h, r)
    t_proj = self.project_on_hyperplane(t, r)
    return np.sum(np.square(h_proj + r - t_proj))
2 usages
def distanceL1(self, h, r, t):
    h_proj = self.project_on_hyperplane(h, r)
    t_proj = self.project_on_hyperplane(t, r)
    return np.sum(np.fabs(h_proj + r - t_proj))
```

功能：计算三元组（头实体、关系、尾实体）的距离。

L2 距离：使用平方和来度量距离。L1 距离：使用绝对值和来度量距离。

两者的区别在于距离度量的方式，L2 更敏感于大偏差，而 L1 在异常值下具有较强的鲁棒性。

```
h_correct_proj = self.project_on_hyperplane(h_correct, self.relation[triple[2] + "_proj"])
t_correct_proj = self.project_on_hyperplane(t_correct, self.relation[triple[2] + "_proj"])
h_corrupt_proj = self.project_on_hyperplane(h_corrupt, self.relation[corrupted_triple[2] + "_proj"])
t_corrupt_proj = self.project_on_hyperplane(t_corrupt, self.relation[corrupted_triple[2] + "_proj"])
```

功能：使用负采样与损失函数优化来更新实体和关系的嵌入。

细节:对每个正三元组与其腐败三元组计算距离。

使用 hinge_loss 来计算正三元组与生成三元组的损失，计算后更新嵌入向量。

计算正负样本的梯度，并更新嵌入向量。

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

功能：实现 TransH 的 hinge 损失函数。

dist_correct 是正样本的距离，dist_corrupt 是负样本的距离。

损失函数的目标是让正样本的距离小于负样本的距离，且两者之差大于一个 margin 值。

```
def corrupt(self, triple):
    corrupted_triple = copy.deepcopy(triple)
    seed = random.random()
    if seed > 0.5:
        rand_head = triple[0]
        while rand_head == triple[0]:
            rand_head = random.sample(list(self.entity.keys()), k: 1)[0]
        corrupted_triple[0] = rand_head
```

负样本生成：随机替换三元组中的头实体或尾实体，生成一个负样本三元组

TransR

改进的地方：

```
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)
    W_r = np.random.uniform(-6 / math.sqrt(self.embedding_dim),
                            6 / math.sqrt(self.embedding_dim),
                            size: (self.embedding_dim, self.embedding_dim))
    transformation_dict[relation] = W_r / np.linalg.norm(W_r, 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)
```

在 TransR 中，每个关系都有一个独立的变换矩阵（W_r），它将实体嵌入映射到关系特定的空间中。这个变换矩阵使得每个关系的表示可以在不同的空间内进行调整，从而更好地捕捉复杂的关系信息

```
h_correct_update -= self.learning_rate * grad_pos
t_correct_update -= (-1) * self.learning_rate * grad_pos
```

在更新嵌入时，TransR 会根据每个关系的变换矩阵调整实体和关系的表示：

grad_pos 和 grad_neg 是计算得到的正负梯度，用于更新实体和关系嵌入。

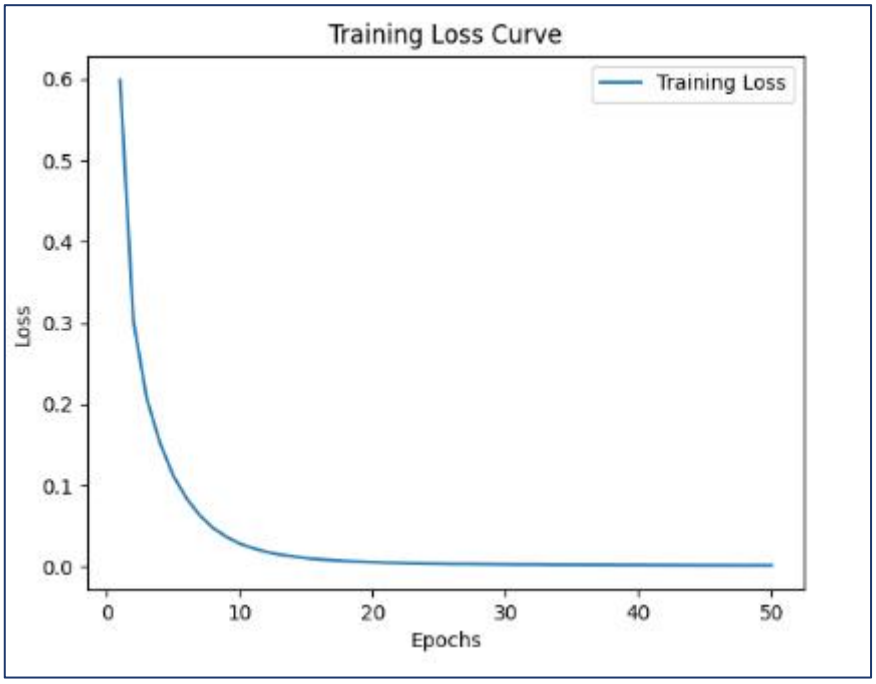
这些梯度更新的过程中，TransR 通过 relation_update 以及变换矩阵 W_r 的更新，调整了关系的嵌入表示

3.5 模型训练

系列 A 训练 EX：

毕竟是自己 DIY 了一些，看起来还是很舒服的：

```
|████████████████████████| 472/472 [00:01<00:00, 263.19it/s]
: 0.0028
|████████████████████████| 472/472 [00:01<00:00, 254.48it/s]
: 0.0027
|████████████████████████| 472/472 [00:01<00:00, 249.51it/s]
: 0.0028
|████████████████████████| 472/472 [00:01<00:00, 265.85it/s]
: 0.0026
|████████████████████████| 472/472 [00:01<00:00, 255.50it/s]
: 0.0026
|████████████████████████| 472/472 [00:01<00:00, 270.49it/s]
: 0.0023
```



系列 B 训练 EX：

我觉得是因为 Batch 和这个维度的问题，导致这个代码效果并不好

```
loss:  396.0294920538926
epoch:  39 cost time: 0.577
loss:  371.52983795247206
epoch:  40 cost time: 0.575
loss:  329.6276674638313
```

这个开源没动过，没加损失图

3.6 测试评估

系列 A 是 model+json+eval

系列 B 是 model+test+eval

Eval 的逻辑我一点都没有调整，原汁原味

代码已经完整分区和管理了，一目了然

稍微改了的是把 json 做成这种，因为实在太多了，原来那种要一个一个改路径


```
[
    {
        "head": 12577,
        "relation": 1142,
        "true_tail": 7744,
        "predicted_tails": [
            12577,
            3529,
            8678,
            4171,
            3843
        ]
    },
```

4. 结果展示

先直接放结果：

FB15k 数据集：

2 个 TransE 平均：

LP：0.482

EP：0.161

2 个 TransH 平均：

LP：0.534

EP：0.112

2 个 TransR 平均：

LP：0.501

EP：0.195

6 个 Trans 平均：

LP：0.506

EP：0.156

WN18 数据集：

2 个 TransE 平均：

LP：0.643

EP：0.268

2 个 TransH 平均：

LP：0.616

EP：0.194

2 个 TransR 平均：

LP：0.415

EP：0.139

6 个 Trans 平均：

LP：0.558

EP：0.201

图像注解：

半自制：系列 A

开源：系列 B

E：TransE

H：TransH

R：TransR

F：FB15K 数据集

W：WN18 数据集

底下双图的都是：

第一幅是我改进前，第二幅是我改进后，主要改进还是负样本生成，另外还尝试了把距离计算，在测试的时候尝试了余弦相似度，效果不错

单图的是直接拿改完的跑的：

半自制 E-F

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransE1/eval.py
Link Prediction Average Hit Score: 0.483823421532
Entity Prediction Average Hit Score: 0.000009028683

进程已结束，退出代码为 0
```

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransE1/eval.py
Link Prediction Average Hit Score: 0.483823421532
Entity Prediction Average Hit Score: 0.169415618493

进程已结束，退出代码为 0
```

半自制 E-W

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransE1/eval.py
Link Prediction Average Hit Score: 0.750966666667
Entity Prediction Average Hit Score: 0.000200000000

进程已结束，退出代码为 0
```

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransE1/eval.py
Link Prediction Average Hit Score: 0.114613333333
Entity Prediction Average Hit Score: 0.355090000000

进程已结束，退出代码为 0
```

半自制 H-F

轮 100-维 100-B1024

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH1/eval.py
Link Prediction Average Hit Score: 0.514697849481
Entity Prediction Average Hit Score: 0.000000000000

进程已结束，退出代码为 0
```

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH1/eval.py
Link Prediction Average Hit Score: 0.514697849481
Entity Prediction Average Hit Score: 0.162711821368

进程已结束，退出代码为 0
```

半自制 H-F

轮 50-维 300-B1024

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH1/eval.py
Link Prediction Average Hit Score: 0.462841044393
Entity Prediction Average Hit Score: 0.000003385756

进程已结束，退出代码为 0
```

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH1/eval.py
Link Prediction Average Hit Score: 0.462841044393
Entity Prediction Average Hit Score: 0.160165732762

进程已结束，退出代码为 0
```

半自制 H-W

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH1/eval.py
Link Prediction Average Hit Score: 0.696606666667
Entity Prediction Average Hit Score: 0.187513333333

进程已结束，退出代码为 0
```

半自制 R-F

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransR1/eval.py
Link Prediction Average Hit Score: 0.472629021190
Entity Prediction Average Hit Score: 0.221372809485

进程已结束，退出代码为 0
```

半自制 R-W

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransR1/eval.py
Link Prediction Average Hit Score: 0.309869057213
Entity Prediction Average Hit Score: 0.167934814962

进程已结束，退出代码为 0
```

开源 E-F

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransE3/eval.py
Link Prediction Average Hit Score: 0.510081653152
Entity Prediction Average Hit Score: 0.067477273112

进程已结束，退出代码为 0
```

开源 E-W

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransE3/eval.py
Link Prediction Average Hit Score: 0.563870000000
Entity Prediction Average Hit Score: 0.200716666667

进程已结束，退出代码为 0
```

开源 H-F

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH2/eval.py
Link Prediction Average Hit Score: 0.401712628306
Entity Prediction Average Hit Score: 0.203853554762

进程已结束，退出代码为 0
```

开源 H-W

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransH2/eval.py
Link Prediction Average Hit Score: 0.536250000000
Entity Prediction Average Hit Score: 0.200040000000

进程已结束，退出代码为 0
```

开源 R-F

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransR2/eval.py
Link Prediction Average Hit Score: 0.531455649416
Entity Prediction Average Hit Score: 0.168028304921

进程已结束，退出代码为 0
```

开源 R-W

```
/data_disk/libh/anaconda3/envs/KEL/bin/python /data_disk/libh/KRL/TransR2/eval.py
Link Prediction Average Hit Score: 0.521836666667
Entity Prediction Average Hit Score: 0.110820000000

进程已结束，退出代码为 0
```

5. 总结

TransE、TransH 和 TransR 是基于不同假设和策略的知识图谱嵌入模型，每个模型都有其优缺点。

1. TransE (Translating Embeddings)

优点：简单高效：TransE 的设计简单，计算效率高。它直接基于向量加法和欧几里得距离计算，易于理解和实现。

适用于大规模数据：由于计算量较小，TransE 能够处理大规模的知识图谱，适合在实际应用中使用。

效果不错：在许多标准任务（如链接预测、实体分类）上，TransE 能够取得不错的效果。缺点：处理复杂关系困难：TransE 假设实体和关系之间的嵌入空间是统一的，无法处理一些复杂的关系。特别是在处理具有多样化几何结构的关系时，TransE 会遇到困难。例如，TransE 不能很好地捕捉如一对多、多对一和多对多关系。无法处理关系的方向性：TransE 假设所有关系都可以通过简单的向量加法来建模，忽略了关系方向性和复杂性。

2. TransH (Translating Hyperplanes)

优点：引入超平面更灵活：通过为每个关系引入一个超平面，TransH 能够有效地处理复杂的关系，尤其是在处理方向性或某些特定几何约束时，比 TransE 更具灵活性。更好的泛化能力：相比 TransE，TransH 能更好地泛化到不同类型的

关系，尤其是在多对多、多对一、一对多等复杂的关系模式下表现更好。

缺点：计算复杂度增加：为了处理超平面，TransH 需要额外的计算步骤，导致训练时间和计算量比 TransE 增加。嵌入空间更加复杂：虽然超平面使模型更灵活，但也使得模型的理解和实现更加复杂，可能在训练过程中引入更多的超参数调整。

3. TransR (Translating Relations)

优点：每个关系有独立的投影矩阵：TransR 通过为每个关系定义一个投影矩阵，将实体嵌入映射到不同的关系空间。这种做法允许模型对不同关系进行更细粒度的建模，因此能处理更复杂的关系。高表示能力：通过不同关系的独立空间，TransR 能够为每个关系学习专属的表示，从而提高了模型的表现力，尤其是在复杂的多种类关系场景下效果更佳。较强的灵活性：由于每个关系都有独特的投影矩阵，TransR 能更好地处理包含多样性和异质性的关系信息。

缺点：计算开销大：每个关系都有一个投影矩阵，导致模型需要存储和计算更多的参数。尤其是在大规模数据集上，计算和存储成本可能会显著增加。

难以训练：由于关系投影矩阵数量庞大，模型的训练过程较为复杂，可能需要更多的优化和调参技巧。

总结对比：

TransE 是一个高效、简单的模型，但它无法有效处理复杂关系和多样化的几何结构。适合于关系较为简单的场景。TransH 通过引入超平面来增加灵活性，能够处理复杂的关系，但其计算开销相较 TransE 更大，适用于具有更复杂几何结构的关系。TransR 通过为每个关系引入独立的投影矩阵，能够更加精细地捕捉复杂关系的多样性，适合大规模和复杂的知识图谱任务，但其计算复杂度和训练难度也大幅增加。