Table of Contents

知识图谱中的知识表示学习

1 知识图谱与知识表示学习

1.1 知识图谱

知识图谱将现实世界中的具象事物与抽象概念表示为实体，将实体之间的联系表示为关系，并最终用以（头实体，关系，尾实体）三元组为基本元素结构来表示知识。

知识图谱中，$\mathcal{E}$表示实体集合，$\mathcal{R}$表示关系集合，$\mathcal{T}$表示三元组集合。对于知识三元组中的任意知识，我们使用$(h,r,t) \in \mathcal{T}$进行表示，其中$h \in \mathcal{E}$代表头实体，$t \in \mathcal{E}$代表尾实体，$r \in \mathcal{R}$代表头尾实体之间的关系。

知识图谱的(h,r,t)三元组符号表示面临的挑战：

- 计算效率低下。知识图谱的(h,r,t)三元组符号表示需要图算法进行计算。这些图算法往往计算复杂度较高，在目前大规模知识图谱上难以快速运行，且难以拓展至其他情况。
- 数据稀疏性强。大规模知识图谱中的实体与关系存在着长尾分布，有很多实体只存在着极少数的关系与之相连。对这些稀疏的实体和关系，往往很难有效理解与推理。

1.2 知识表示学习

知识表示学习基于分布式表示的思想，将实体（或关系）的语义信息映射到低维、稠密、实值的向量空间中，使得语义相似的两个对象之间的距离也相近。用粗体的符号$\mathbf{h}, \mathbf{t}, \mathbf{r}$表示头尾实体与关系对应的表示向量。

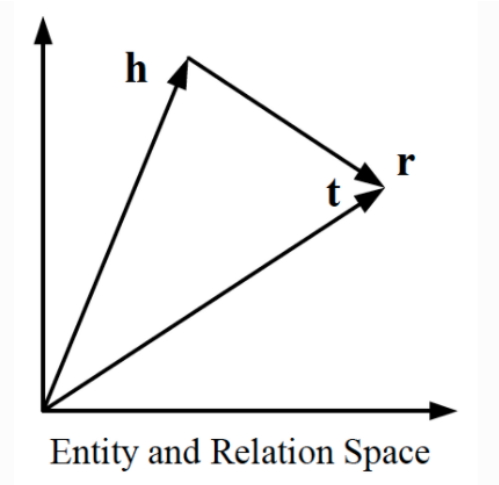
知识的分布式表示特点：

- 分布式表示学习到的是低维向量。这使得实体与关系之间的语义联系能够在低维空间中得以高速计算，显著提高计算效率。
- 传统独热表示基于所有对象的相互独立假设，所有向量之间两两正交，丢失了大量对象之间的相似及关联信息。而分布式表示则能通过稠密低维向量之间的相似度计算表达对象之间的关系，较好地缓解了数据稀疏带来的问题。
- 分布式表示能够将多源异质信息映射到同一语义空间中，建立多源跨模态的信息交互，且分布式表示也能更便捷地融入深度学习的模型框架中。

2 知识表示学习的典型模型及训练

2.1 TransE模型

TransE模型将知识三元组$(h,r,t) \in \mathcal{T}$中的实体$h,t \in \mathcal{E}$和关系$r \in \mathcal{R}$映射至同一个低维实值向量空间$\mathbf{h},\mathbf{t},\mathbf{r} \in \mathbb{R}^d$，将实体与实体之间的关系表示为实体向量之间的平移操作。



具体地，对于给定的三元组(h,r,t)，TransE模型将关系r看作从头实体h到尾实体t的平移向量。基于以上平移假设，平移模型得到一个三元组的实体与关系向量$\mathbf{h},\mathbf{t},\mathbf{r} \in \mathbb{R}^d$之间存在$\mathbf{h}+\mathbf{r} \approx \mathbf{t}$的关系。

形式化地，TransE模型对三元组(h,r,t)定义如下评分函数：

$$E(h,r,t) = \|\mathbf{h}+\mathbf{r}-\mathbf{t}\|_{L1/L2}$$

实际中，TransE模型使用最大间隔方法，定义目标函数

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{T}} \sum_{(h',r,t')\in\mathcal{T}'} \max\left(\gamma + E(h,r,t) - E(h',r,t'), 0\right)$$

其中，$\mathcal{T}$为正例三元组集合，$\mathcal{T}' = \left\{\left(h',r,t\right) | h' \in \mathcal{E}\right\} \cup \left\{\left(h,r,t'\right) | t' \in \mathcal{E}\right\}$为负例三元组集合，$\gamma > 0$为正负例三元组得分的间隔距离。TransE模型通过最大化正负例三元组之间的得分差来优化知识表示。

2.2 TransR模型

TransR模型将知识三元组$(h,r,t) \in \mathcal{T}$中的实体$h,t \in \mathcal{E}$映射到实体向量空间$\mathbf{h},\mathbf{t} \in \mathbb{R}^d$，关系$r \in \mathcal{R}$映射到关系向量空间$\mathbf{r} \in \mathbb{R}^k$，且$k \neq d$。并且使用不同的映射矩阵$\mathbf{M}_r$定义从实体空间到各个关系空间的映射。

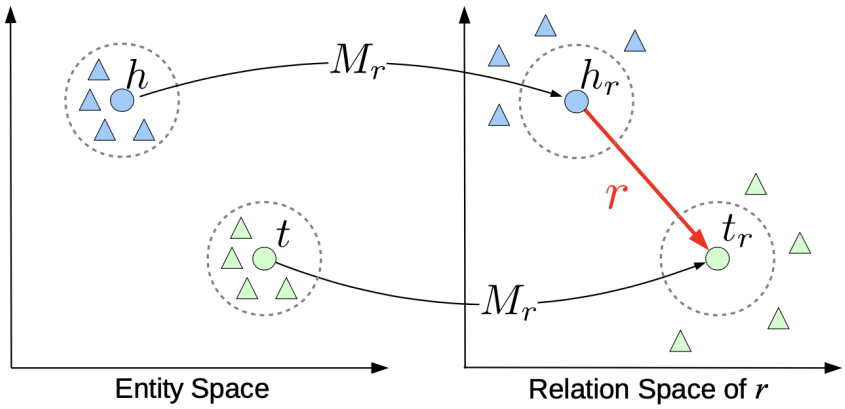


Figure 1: Simple illustration of TransR.

具体地，对于给定的三元组(h, r, t)，TransR模型使用特定关系r下的映射矩阵$\mathbf{M}_r \in \mathbb{R}^{k\times d}$将实体向量$\mathbf{h}$和$\mathbf{t}$从实体空间映射到关系r所在的关系空间中，得到$\mathbf{h}_r$和$\mathbf{t}_r$。

$$\mathbf{h}_r = \mathbf{M}_r\mathbf{h} \quad \mathbf{t}_r = \mathbf{M}_r\mathbf{t}$$

在关系r所在的关系空间中，$\mathbf{h}_r, \mathbf{t}_r, \mathbf{r} \in \mathbb{R}^k$之间存在$\mathbf{h}_r + \mathbf{r} \approx \mathbf{t}_r$的关系。

形式化地，TransR模型对三元组(h, r, t)定义如下评分函数：

$$\begin{aligned} f_r(h, t) &= \|\mathbf{h}_r + \mathbf{r} - \mathbf{t}_r\|_2^2 \\ &= \|\mathbf{M}_r\mathbf{h} + \mathbf{r} - \mathbf{M}_r\mathbf{t}\|_2^2 \end{aligned}$$

实际中，TransR模型使用最大间隔方法，定义目标函数

$$\mathcal{L} = \sum_{(h,r,t)\in\mathcal{T}} \sum_{(h',r,t')\in\mathcal{T}'} \max\left(\gamma + f_r(h, t) - f_r(h', t'), 0\right)$$

其中，$\mathcal{T}$为正例三元组集合，$\mathcal{T}'$为负例三元组集合，γ为正负例三元组得分的间隔距离。TransR模型通过在关系空间最大化正负例三元组之间的得分差来优化知识表示。

2.3 RESCAL模型

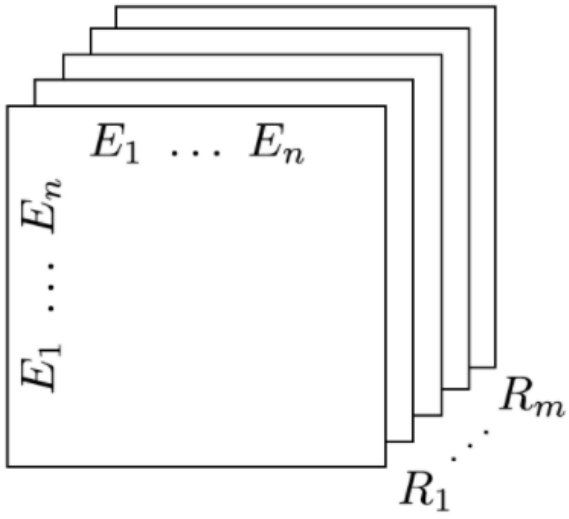
RESCAL模型是基于矩阵分解的方式进行知识表示学习。将三元组(h, r, t)表示为三阶的实体关系张量$\mathbf{X}^{n\times n\times m}$中的一个元素，

$$(h, r, t) \in \mathbf{X}^{n\times n\times m}$$

其中，$n = |\mathcal{E}|$为实体的数量，$m = |\mathcal{R}|$为关系的数量。

实体关系张量$\mathbf{X}$中，通过$\mathbf{X}_{ijk}$表示第i个实体和第j个实体之间具有第k个关系

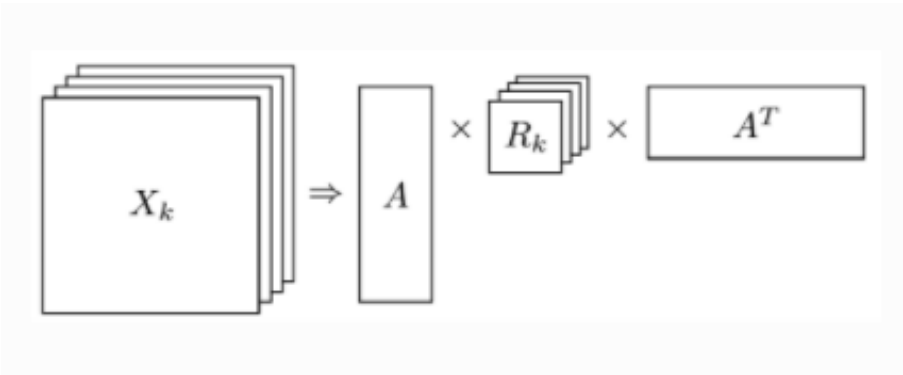
$$\mathbf{X}_{ijk} = \begin{cases} 1, & \text{如果}(h_i, r_k, t_j) \in \mathcal{T} \\ 0, & \text{如果}(h_i, r_k, t_j) \notin \mathcal{T} \end{cases}$$



由于实体关系张量$\mathbf{X}$趋于稀疏，RESCAL模型采用二元分解法，以实体向量表示和关系方阵的形式捕获张量的固有结构。设实体关系张量$\mathbf{X} = \{\mathbf{X}_1, \cdots, \mathbf{X}_m\}$，则有矩阵分解

$$\mathbf{X}_i \approx \mathbf{A}\mathbf{R}_i\mathbf{A}^\top, \quad i = 1, \cdots, m$$

其中，矩阵$A \in \mathbb{R}^{n \times d}$实体向量组成的矩阵，矩阵$R_i \in \mathbb{R}^{d \times d}$为第i个关系方阵。



RESCAL模型对三元组(h, r, t)定义如下评分函数：

$$f_r(h, t) = \mathbf{h}^\top \mathbf{M}_r \mathbf{t}$$

其中，关系方阵$\mathbf{M}_r = \mathbf{R}_i \in \mathbb{R}^{d \times d}$。

2.4 DistMult模型

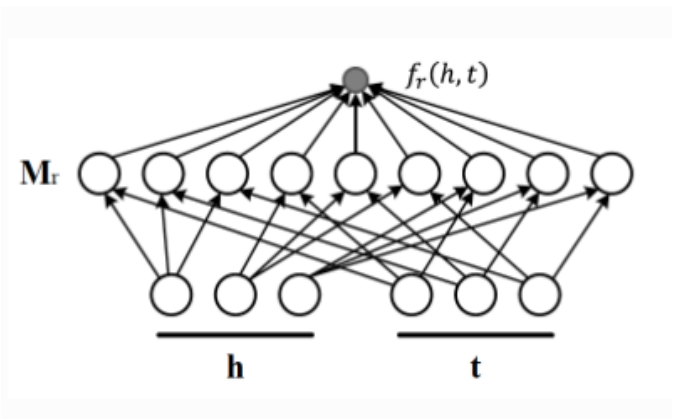
在RESCAL模型中，令关系方阵$\mathbf{M}_r$为关系r的对角矩阵

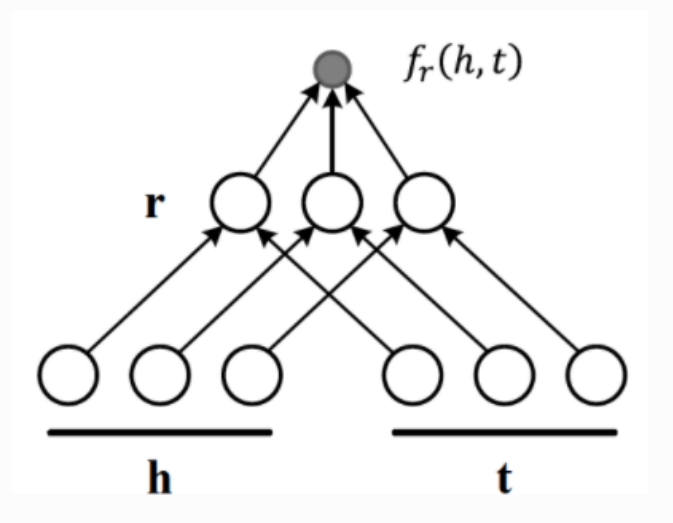
$$\mathbf{M}_r = diag(r)$$

DistMult模型对三元组(h, r, t)定义如下评分函数：

$$f_r(h, t) = \mathbf{h}^\top diag(r)\mathbf{t}$$

其中，$diag(r)$为关系r的对角矩阵。





2.5 优化策略

极大似然估计的logistic优化

$$\min \sum_{(h,r,t)\in\mathbb{D}^+\cup\mathbb{D}^-} \log\left(1+e^{-y\times f(h,r,t)}\right)$$

其中，$\mathbb{D}^+$为正样本数据，$\mathbb{D}^-$为负样本数据，$y\in+1,-1$为正、负三元组标签。

基于间隔的优化

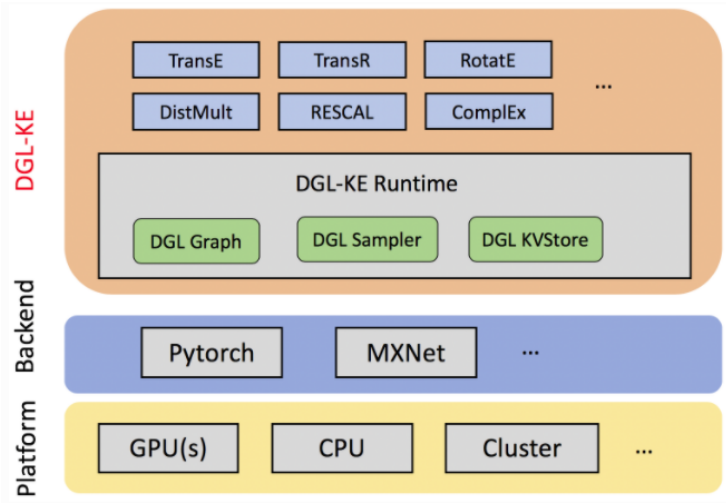
$$\min \mathcal{L} = \sum_{(h,r,t)\in\mathbb{D}^+} \sum_{(h,r,t)\in\mathbb{D}^-} \max\left(\gamma + f(h,r,t) - f(h',r',t'),0\right)$$

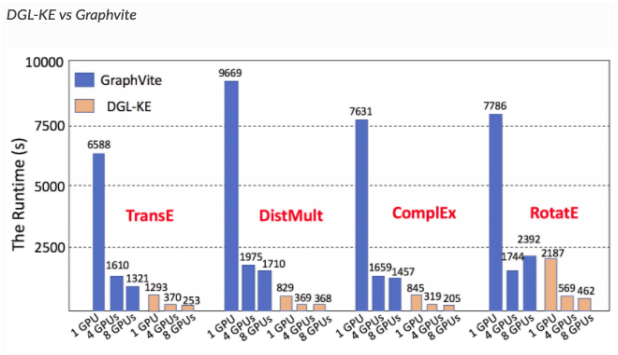
其中，$\mathbb{D}^+$为正样本数据，$\mathbb{D}^-$为负样本数据。

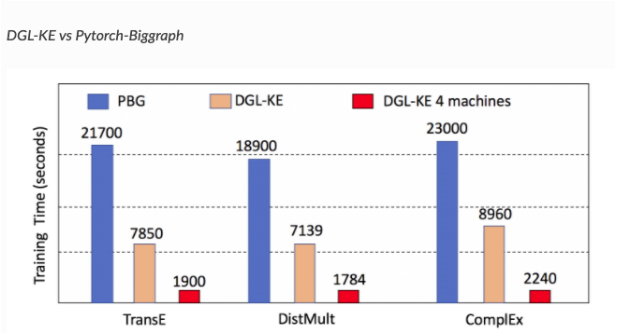
3 DGL-KE大规模知识表示学习框架

3.1 DGL-KE简介及特性

DGL-KE是一个高性能，易于使用且可扩展的软件包，用于学习大规模知识表示学习。该程序包在Deep Graph Library（DGL）的顶部实现，开发人员可以在CPU机器，GPU机器以及集群上运行DGL-KE。DGL-KE包括一系列流行的模型，包括TransE，TransR，RESCAL，DistMult，ComplEx，和RotatE。



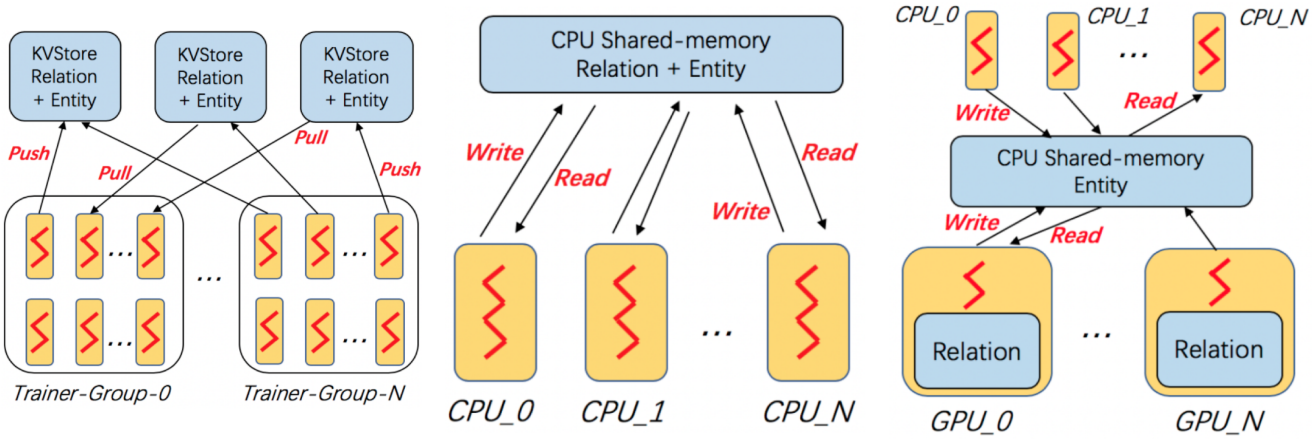




DGL-KE专为大规模表示学习而设计。它包含了各种新颖的优化，可加速对具有数百万个节点和数十亿条边的知识图的训练。对包含超过86M个节点和338M个关系的知识图的基准测试表明，DGL-KE可以在具有8个GPU的EC2实例上在100分钟内计算分布式表示，而在具有4台计算机（48核/计算机）的EC2集群上可以在30分钟内计算分布式表示。这些结果代表了最佳竞争方法的2倍～5倍加速。

DGL-KE特性：

- 基于METIS图分割算法的分布式训练；
- 基于共享内存的单机多进程训练；
- CPU-GPU混合训练；



3.2 DGL-KE安装及内建知识图谱基准

系统要求：

- Ubuntu 16.04或更高版本
- macOS x
- Python 3.5 或更高版本
- Torch 1.5.0
- DGL 0.4.3

- DGL-KE 0.1.1

In [8]:

```
1 ! pip3 install torch==1.5
```

executed in 1.64s, finished 16:18:55 2020-11-01

```
Requirement already satisfied: torch==1.5 in /Users/anch3or/opt/anacon
da3/envs/DGL-KE/lib/python3.6/site-packages (1.5.0)
Requirement already satisfied: future in /Users/anch3or/opt/anaconda3/
envs/DGL-KE/lib/python3.6/site-packages (from torch==1.5) (0.18.2)
Requirement already satisfied: numpy in /Users/anch3or/opt/anaconda3/e
nvs/DGL-KE/lib/python3.6/site-packages (from torch==1.5) (1.19.2)
```

In [2]:

```
1 ! pip3 install dgl==0.4.3
```

executed in 881ms, finished 14:52:21 2020-11-01

```
Requirement already satisfied: dgl==0.4.3 in /Users/anch3or/opt/anacon
da3/envs/DGL-KE/lib/python3.6/site-packages (0.4.3)
Requirement already satisfied: requests>=2.19.0 in /Users/anch3or/opt/
anaconda3/envs/DGL-KE/lib/python3.6/site-packages (from dgl==0.4.3)
(2.24.0)
Requirement already satisfied: scipy>=1.1.0 in /Users/anch3or/opt/anac
onda3/envs/DGL-KE/lib/python3.6/site-packages (from dgl==0.4.3) (1.5.
3)
Requirement already satisfied: numpy>=1.14.0 in /Users/anch3or/opt/ana
conda3/envs/DGL-KE/lib/python3.6/site-packages (from dgl==0.4.3) (1.1
9.2)
Requirement already satisfied: networkx>=2.1 in /Users/anch3or/opt/ana
conda3/envs/DGL-KE/lib/python3.6/site-packages (from dgl==0.4.3) (2.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /Users/anch3or/op
t/anaconda3/envs/DGL-KE/lib/python3.6/site-packages (from requests>=2.
19.0->dgl==0.4.3) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /Users/anch3or/opt/anac
onda3/envs/DGL-KE/lib/python3.6/site-packages (from requests>=2.19.0->
dgl==0.4.3) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /Users/anch3or/op
t/anaconda3/envs/DGL-KE/lib/python3.6/site-packages (from requests>=2.
19.0->dgl==0.4.3) (2020.6.20)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /Users/anch3or/opt/anaconda3/envs/DGL-KE/lib/python3.6/site-package
s (from requests>=2.19.0->dgl==0.4.3) (1.25.11)
Requirement already satisfied: decorator>=4.3.0 in /Users/anch3or/opt/
anaconda3/envs/DGL-KE/lib/python3.6/site-packages (from networkx>=2.1-
>dgl==0.4.3) (4.4.2)
```

In [3]:

```
1 ! pip3 install dglke
```

executed in 855ms, finished 14:52:22 2020-11-01

```
Requirement already satisfied: dglke in /Users/anch3or/opt/anaconda3/e
nvs/DGL-KE/lib/python3.6/site-packages (0.1.1)
Requirement already satisfied: numpy in /Users/anch3or/opt/anaconda3/e
nvs/DGL-KE/lib/python3.6/site-packages (from dglke) (1.19.2)
Requirement already satisfied: setuptools in /Users/anch3or/opt/anacon
da3/envs/DGL-KE/lib/python3.6/site-packages (from dglke) (50.3.0.post2
0201006)
```

DGL-KE提供五个内建知识图谱：

Dataset	#nodes	#edges	#relations
FB15k	14951	592213	1345
FB15k-237	14541	310116	237
wn18	40943	151442	18
wn18rr	40943	93003	11
Freebase	86054151	338586276	14824

用户可以在任务中使用 `--dataset` 选项指定其中一个数据集。

3.3 DGL-KE训练与推理

训练命令：

- `dglke_train` 在单台计算机上的CPU或GPU上训练知识图谱分布式表示，并将经过训练的节点分布式表示和关系分布式表示保存在磁盘上。
- `dglke_dist_train` 在机器集群上训练知识图谱分布式表示。此命令启动一组进程以自动执行分布式训练。

In [2]:

```
! DGLBACKEND=pytorch dglke_train --model_name TransE_l2 --dataset FB15k --batch_
--neg_sample_size 200 --hidden_dim 400 --gamma 19.9 --lr 0.25 --max_step 500 --l
--batch_size_eval 16 -adv --regularization_coef 1.00E-09 --test --num_thread 1 -
```

executed in 4m 60s, finished 16:05:59 2020-11-01

```
Reading train triples....
Finished. Read 483142 train triples.
Reading valid triples....
Finished. Read 50000 valid triples.
Reading test triples....
Finished. Read 59071 test triples.
|Train|: 483142
random partition 483142 edges into 2 parts
part 0 has 241571 edges
part 1 has 241571 edges
/Users/anch3or/opt/anaconda3/envs/DGL-KE/lib/python3.6/site-packages/d
gl/base.py:25: UserWarning: multigraph will be deprecated.DGL will tre
at all graphs as multigraph in the future.
  warnings.warn(msg, warn_type)
|valid|: 50000
|test|: 59071
Total initialize time 2.491 seconds
[proc 1][Train](100/500) average pos_loss: 0.9561189103126526
[proc 1][Train](100/500) average neg_loss: 1.4461995524168014
[proc 1][Train](100/500) average loss: 1.2011592334508896
[proc 1][Train](100/500) average regularization: 0.000658620201478044
[proc 1][Train] 100 steps take 7.313 seconds
[proc 1]sample: 0.483, forward: 3.894, backward: 2.358, update: 0.577
[proc 0][Train](100/500) average pos_loss: 0.9840711963176727
[proc 0][Train](100/500) average neg_loss: 1.5231017434597016
[proc 0][Train](100/500) average loss: 1.2535864716768266
[proc 0][Train](100/500) average regularization: 0.0006627218283119518
[proc 0][Train] 100 steps take 7.343 seconds
[proc 0]sample: 0.462, forward: 3.790, backward: 2.557, update: 0.533
[proc 1][Train](200/500) average pos_loss: 0.5287754726409912
[proc 1][Train](200/500) average neg_loss: 0.6716524624824524
[proc 1][Train](200/500) average loss: 0.6002139669656753
[proc 1][Train](200/500) average regularization: 0.0008687257225392386
[proc 1][Train] 100 steps take 8.056 seconds
[proc 1]sample: 0.467, forward: 4.417, backward: 2.575, update: 0.596
[proc 0][Train](200/500) average pos_loss: 0.5213128805160523
[proc 0][Train](200/500) average neg_loss: 0.644518364071846
[proc 0][Train](200/500) average loss: 0.5829156213998794
[proc 0][Train](200/500) average regularization: 0.0008710341912228615
[proc 0][Train] 100 steps take 8.176 seconds
[proc 0]sample: 0.442, forward: 4.413, backward: 2.723, update: 0.597
[proc 1][Train](300/500) average pos_loss: 0.45286581456661223
[proc 1][Train](300/500) average neg_loss: 0.583562252521515
[proc 1][Train](300/500) average loss: 0.5182140344381332
[proc 1][Train](300/500) average regularization: 0.0009724267537239939
[proc 1][Train] 100 steps take 8.213 seconds
[proc 1]sample: 0.539, forward: 4.424, backward: 2.639, update: 0.610
[proc 0][Train](300/500) average pos_loss: 0.4545183122158051
[proc 0][Train](300/500) average neg_loss: 0.588814621269703
[proc 0][Train](300/500) average loss: 0.5216664677858353
[proc 0][Train](300/500) average regularization: 0.0009743505890946835
[proc 0][Train] 100 steps take 8.217 seconds
[proc 0]sample: 0.481, forward: 4.368, backward: 2.780, update: 0.586
```

```
[proc 1][Train](400/500) average pos_loss: 0.4154008600115776
[proc 1][Train](400/500) average neg_loss: 0.5459030738472939
[proc 1][Train](400/500) average loss: 0.48065196692943574
[proc 1][Train](400/500) average regularization: 0.0010316338716074825
[proc 1][Train] 100 steps take 8.662 seconds
[proc 1]sample: 0.468, forward: 4.660, backward: 2.866, update: 0.665
[proc 0][Train](400/500) average pos_loss: 0.4139659160375595
[proc 0][Train](400/500) average neg_loss: 0.5454588994383812
[proc 0][Train](400/500) average loss: 0.4797124093770981
[proc 0][Train](400/500) average regularization: 0.0010311792802531273
[proc 0][Train] 100 steps take 8.752 seconds
[proc 0]sample: 0.528, forward: 4.781, backward: 2.827, update: 0.615
[proc 1][Train](500/500) average pos_loss: 0.38365421295166013
[proc 1][Train](500/500) average neg_loss: 0.5210811349749566
[proc 1][Train](500/500) average loss: 0.45236767441034315
[proc 1][Train](500/500) average regularization: 0.0010722533194348217
[proc 1][Train] 100 steps take 7.739 seconds
[proc 1]sample: 0.495, forward: 4.056, backward: 2.641, update: 0.545
proc 1 takes 39.982 seconds
[proc 0][Train](500/500) average pos_loss: 0.37971875697374347
[proc 0][Train](500/500) average neg_loss: 0.494854032099247
[proc 0][Train](500/500) average loss: 0.43728639334440234
[proc 0][Train](500/500) average regularization: 0.0010723784647416322
[proc 0][Train] 100 steps take 7.618 seconds
[proc 0]sample: 0.472, forward: 4.123, backward: 2.463, update: 0.559
proc 0 takes 40.107 seconds
training takes 40.14898729324341 seconds
Save model to ckpts/TransE_l2_FB15k_1
-------------- Test result --------------
Test average MRR : 0.31762194747952616
Test average MR : 201.96148702408965
Test average HITS@1 : 0.20915508455925919
Test average HITS@3 : 0.36826022921568957
Test average HITS@10 : 0.5210594030911954
-----------------------------------------
testing takes 247.009 seconds
```

推理命令：

- `dglke_predict` 使用预训练的分布式表示来预测三元组中的缺失实体/关系。
- `dglke_emb_sim` 计算实体分布式表示或关系分布式表示的相似性得分。

In [1]:

```
! DGLBACKEND=pytorch dglke_predict --model_path ckpts/TransE_l2_FB15k_1/ --forma
--data_files head.list rel.list tail.list --score_func logsigmoid --topK 5 --exe
```

executed in 6.98s, finished 16:29:12 2020-11-02

```
ckpts/TransE_l2_FB15k_1/config.json
{'dataset': 'FB15k', 'model': 'TransE_l2', 'emb_size': 400, 'max_train
_step': 500, 'batch_size': 1000, 'neg_sample_size': 200, 'lr': 0.25,
'gamma': 19.9, 'double_ent': False, 'double_rel': False, 'neg_adversar
ial_sampling': True, 'adversarial_temperature': 1.0, 'regularization_c
oef': 1e-09, 'regularization_norm': 3, 'emap_file': 'entities.dict',
'rmap_file': 'relations.dict'}
/Users/distiller/project/conda/conda-bld/pytorch_1587428077029/work/at
en/src/ATen/native/BinaryOps.cpp:81: UserWarning: Integer division of
tensors using div or / is deprecated, and in a future release div will
perform true division as in Python 3. Use true_divide or floor_divide
(// in Python) instead.
Inference Done
The result is saved in result.tsv
```

In [7]:

```
! DGLBACKEND=pytorch dglke_emb_sim --emb_file ckpts/TransE_l2_FB15k_1/FB15k_Trar
--format 'l_r' --data_files head.list tail.list --topK 5
```

executed in 1.32s, finished 16:16:34 2020-11-01

```
/Users/distiller/project/conda/conda-bld/pytorch_1587428077029/work/at
en/src/ATen/native/BinaryOps.cpp:81: UserWarning: Integer division of
tensors using div or / is deprecated, and in a future release div will
perform true division as in Python 3. Use true_divide or floor_divide
(// in Python) instead.
Inference Done
The result is saved in result.tsv
```

4 参考资料

- 《知识图谱与深度学习》，刘志远等，清华大学出版社，2020；
- DGL-KE Documentation https://dglke.dgl.ai/doc/ (https://dglke.dgl.ai/doc/)；
- transE: Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, JasonWeston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Advances in Neural Information Processing Systems 26. 2013.
- TransR: Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
- RESCAL: Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, 2011.
- Survey paper: Q. Wang, Z. Mao, B. Wang and L. Guo, "Knowledge Graph Embedding: A Survey of Approaches and Applications," in IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 12, pp. 2724-2743, 1 Dec. 2017.
- DistMult: Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Proceedings of the International Conference on Learning Representations (ICLR) 2015, May 2015.

In [ ]:

```
1
```