Knowledge Graph Convolutional Networks for Recommender Systems

code review

(https://github.com/hwwang55/KGCN)

I- DNA (Intelligent Data, Network, and AI) Lab.

배홍섭(Bae Hong Seob)

INDEX

- ❑ PREPROCESS
- ❑ DATA LOADER
- ❑ MODEL
- ❑ TRAIN

PREPROCESS

PREPROCESS

- read_item_index_to_entity_id_file()

- convert_rating()

- convert_kg()

PREPROCESS

- read_item_index_to_entity_id_file()
  - Item_index2entity_id.txt 파일(우리에게 주어진 데이터)
  - 첫번째 column : item_index
  - 두번째 column : satori_id (=entity_index)

```
item_index2entity_id.txt
data > movie > item_index2entity_id.txt
1	0
2	1
3	2
4	3
5	4
8	5
10	6
11	7
12	8
13	9
```

item_index_old2new	entity_id2index
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

- what that function does:
  - 무작위로 제공된 데이터(순차적인 index가 아님)에 대해 반복문을 돌면서 0부터 n(제공된 데이터 갯수)만큼 순차적으로 id를 부여한다.
    - item_index_old2new = dict()
    - item_index_old2new[item_index] = i

PREPROCESS

- convert_rating()
  - rating.csv 파일
    - Columns : userId, movieId, rating, timestamp
  - 이때 movieId(=entity)가 old2new에 존재하지 않을 경우 생략
    - Entity의 id 정보가 주어지지 않은 경우
  - rating이란 edge의 weight 값
    - Threshold(0.5 설정)에 따라 user_pos_rating / user_neg_rating 로 split
    - User_pos_rating : (key1, (value1,vaule2, … ,))
      - 하나의 user에 연결된 entity는 여러개일 수 있음.
  - Item set – user_pos_rating – user_neg_rating = unwatched set
  - Unwatched_set에서 user_pos_rating 갯수만큼 random sampling
  - User_pos_rating은 (user_index, item_index, 1)
    - (user_index1, value1, 1), (user_index1, value2, 1), (user_index1, value3, 1)
  - Unwatched_set에서 샘플링된 데이터는 (user_index, item_index, 0) 으로 변환
  - 해당 내용을 rating_final.txt 파일을 생성 및 저장

PREPROCESS

❑ convert_rating()

❑ what that function does:

- rating(=weight)을 기준으로 user가 선호하는 item에는 1, 아직 어떠한 weight도 존재하지 않는 데이터에 대해서는 0을 부과하여 dataset을 생성.
- 우리의 목적은 아직 겪지 않은 item에 대해 추천 or not

data_loader.py 1 | ratings.csv

data > ml-20m > ratings.csv

```
userId,movieId,rating,timestamp
1,2,3.5,1112486027
1,29,3.5,1112484676
1,32,3.5,1112484819
1,47,3.5,1112484727
1,50,3.5,1112484580
1,112,3.5,1094785740
1,151,4.0,1094785734
1,223,4.0,1112485573
```

data_loader.py 1 | ratings_final.txt

data > movie > ratings_final.txt

```
0	770	1
0	1795	1
0	2434	1
0	3842	1
0	775	1
0	780	1
0	141	1
0	1805	1
0	1937	1
0	1811	1
0	2708	1
0	791	1
0	1688	1
```

PREPROCESS

movieId 가
Entity_id2index 에 존재 ?

No → 건너뜀(=continue)

Yes ↓

rating >= 0.5

No → user_neg_rating 집합

Yes ↓

user_pos_rating

user_neg_rating	
user_index_old	item index set
0	(0,1,2)
1	(1,3,4,5)
2	(3,)

user_pos_rating	
user_index_old	item index set
0	(0,1,2)
1	(1,3,4,5)
2	(3,)

unwatched_set 생성(user 당)
= item_set – neg_item_set – pos_item_set

unwatched_set	
user_index_old	item index set
0	(7,8,9, … ,)
1	(2,10,11, … ,)
2	(5,6,7, … ,)

PREPROCESS

user_pos_rating	
user_index_old	item index set
0	(0,1,2)
1	(1,3,4,5)
2	(3,)

+

unwatched_set	
user_index_old	item index set
0	(7,8,9, … ,)
1	(2,10,11, … ,)
2	(5,6,7, … ,)

rating_final_txt		
user_index_old	item index set	interaction
0	0	1
0	1	1
0	2	1
1	1	1
1	3	1
1	4	1
1	5	1
2	3	1
0	7	0
0	8	0
0	9	0
1	2	0
1	10	0
1	11	0
1	12	0
2	5	0

unwawtched_set 에서 각 user 당 pos_rating item set 개수만큼
random sampling 하여 rating_final.txt 파일에 저장.

PREPROCESS

data_loader.py 1 | ratings.csv

data > ml-20m > ratings.csv

```
userId,movieId,rating,timestamp
1,2,3.5,1112486027
1,29,3.5,1112484676
1,32,3.5,1112484819
1,47,3.5,1112484727
1,50,3.5,1112484580
1,112,3.5,1094785740
1,151,4.0,1094785734
1,223,4.0,1112485573
```

data_loader.py 1 | ratings_final.txt

data > movie > ratings_final.txt

```
0	770	1
0	1795	1
0	2434	1
0	3842	1
0	775	1
0	780	1
0	141	1
0	1805	1
0	1937	1
0	1811	1
0	2708	1
0	791	1
0	1688	1
```

PREPROCESS

- convert_kg()
  - kg.txt 파일
    - Columns : head_index, relation, tail_index
  - head,tail 은 entity이므로 entity_id2index(=satori to index)에 존재.
    - 존재하지 않는다면 entity_cnt(entity의 수)를 이용하여 (head_id : entity_cnt)로 entity_id2index 에 추가.
  - relation은 0부터 새롭게 index 할당
  - kg_final.txt 파일 생성
    - (head_index, relation_index, tail_index)

- what that function does:
  - relation에 index를 부여
  - kg 생성

PREPROCESS

- convert_kg()

- what that function does:
  - relation에 index를 부여
  - kg 생성
  - 이때 relation에 적힌 내용에 관계없이 새로운 id로 할당.
  - (head_index, relation_index, tail_index) 변환

data_loader.py 1 | kg.txt

data > movie > kg.txt

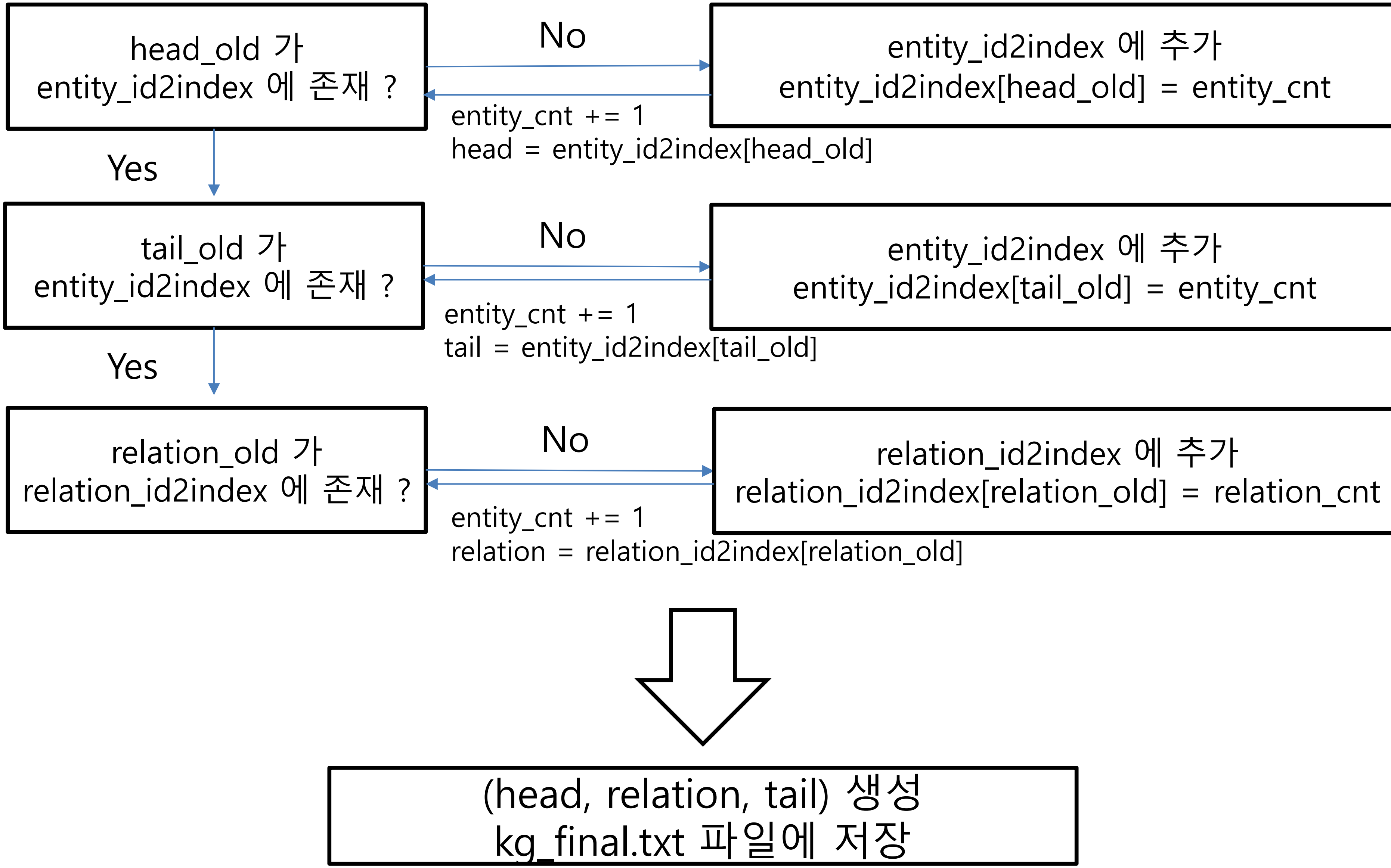
11904	film.film.producer	16954
348	film.film.actor	16955
13598	film.film.costume_designer	16956
9098	film.film.actor	16957
14187	film.film.director	16958
11504	film.film.actor	16959
8412	film.film.executive_producer	16960
1691	film.film.set_designer	16961
5018	film.film.actor	16962
12027	film.film.actor	16963

data_loader.py 1 | kg_final.txt

data > movie > kg_final.txt

11904	0	16954
348	1	16955
13598	2	16956
9098	1	16957
14187	3	16958
11504	1	16959
8412	4	16960
1691	5	16961
5018	1	16962
12027	1	16963

PREPROCESS


head_old 가
entity_id2index 에 존재 ?
No
entity_id2index 에 추가
entity_id2index[head_old] = entity_cnt
entity_cnt += 1
head = entity_id2index[head_old]
Yes
tail_old 가
entity_id2index 에 존재 ?
No
entity_id2index 에 추가
entity_id2index[tail_old] = entity_cnt
entity_cnt += 1
tail = entity_id2index[tail_old]
Yes
relation_old 가
relation_id2index 에 존재 ?
No
relation_id2index 에 추가
relation_id2index[relation_old] = relation_cnt
entity_cnt += 1
relation = relation_id2index[relation_old]
(head, relation, tail) 생성
kg_final.txt 파일에 저장

PREPROCESS

- ❑ convert_kg()

- ❑ Output 1:

data_loader.py 1 | kg.txt

data > movie > kg.txt

```
11904	film.film.producer	16954
348	film.film.actor	16955
13598	film.film.costume_designer	16956
9098	film.film.actor	16957
14187	film.film.director	16958
11504	film.film.actor	16959
8412	film.film.executive_producer	16960
1691	film.film.set_designer	16961
5018	film.film.actor	16962
12027	film.film.actor	16963
```

data_loader.py 1 | kg_final.txt

data > movie > kg_final.txt

```
11904	0	16954
348	1	16955
13598	2	16956
9098	1	16957
14187	3	16958
11504	1	16959
8412	4	16960
1691	5	16961
5018	1	16962
12027	1	16963
```

PREPROCESS

- Output 2:

item_index_old2new	entity_id2index
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

item_index_old2new	entity_id2index
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
	11
	12
	13

kg.txt 의 head, tail (=entity) 중에 item_index2entity_index 에 없던 정보들
(ex : film.producer(=item_attributes) entity 이므로 entity_id2index에 추가

DATA LOADER

DATA LOADER

- dataset_split()
- load_rating()
- load_kg()

DATA LOADER

- dataset_split()
  - eval_ratio, test_ratio에 따라 주어진 전체 dataset을 train/ valid(=eval) / test_data 으로 split

- what that function does:
  - eval_ratio, test_ratio에 따라 train/ valid(=eval) / test_data return

DATA LOADER

- load_rating()
  - rating_final.txt
    - (user_index, item_index, 0 or 1)
  - 해당 내용을 rating_final.npy에 저장

  - 첫번째 열 len = n_user
  - 두번째 열 len = n_item
  - train/vaild/test data split -> dataset_split() 사용

- what that function does:
  - rating_final.npy(numpy파일) 생성(=변환)
  - n_user, n_item return
  - train / valid / test data return

DATA LOADER

- load_kg()
  - kg_final.txt 파일
    - Columns : (head_index, relation_index, tail_index)
  - kg_final.npy 파일 생성(=변환)
  - n_entity : len(head_index와 tail_index의 합집합)
  - n_relation : len(relation_index)
  - kg 생성 by construct_kg()
  - adj_entity, adj_relation 생성 by construct_adj(kg, n_entity)

DATA LOADER

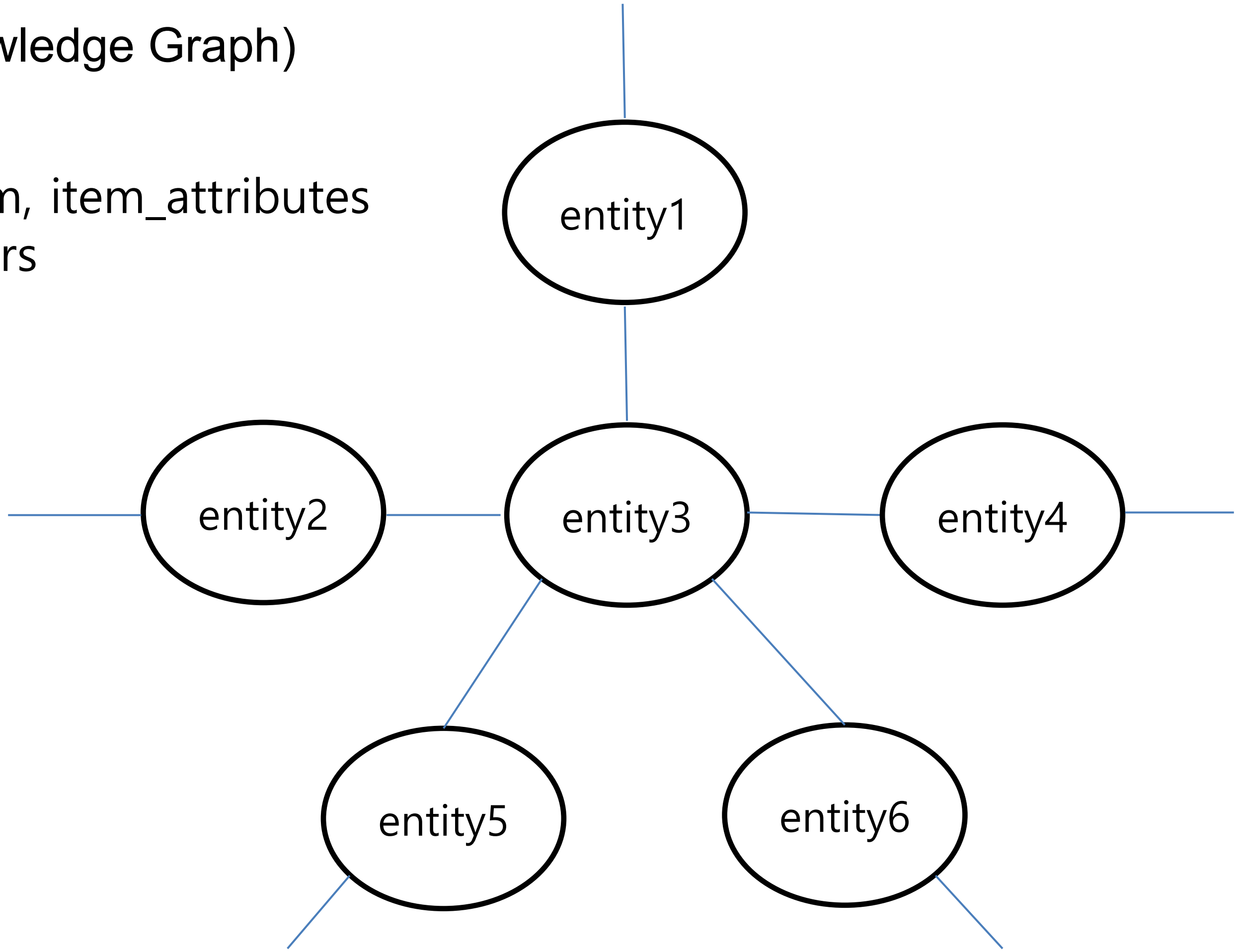
- construct_kg(kg_npy)
  - kg = dict()
  - kg[head].append((tail, relation))
  - kg[tail].append((head, relation))
    - 이때 key 값은 entity_index 따라서 head인지 tail인지 따로 표시 x (undirected)
    - entity_index : ((entity_index1, relation1), (entity_index2, relation2), … ,)

- what that function does:
  - (head_index, relation_index, tail_index) 의 npy 파일을 dictionary 구조로 변환
  - kg(Knowledge Graph)생성 -> dictionary로 표현됨

DATA LOADER

❑ KG(Knowledge Graph)

entity ∋ item, item_attributes
entity ∄ users



DATA LOADER

- construct_adj(kg_npy, n_entity)
  - adj_entity, adj_relation는 np.zeros([n_entity, neighbors_sample_size])
    - neighbors_sample_size 는 하이퍼파라미터로 main.py에서 설정
  - n_neighbors 는 실제 entity의 neighbors 수(Not hyper-parameter)
  - Neighbors_sampling 시 n_neighbors가 neighbors_sample_size보다 작다
    - 복원 추출
    - 크거나 같다 -> 비복원 추출
    - Sampled_indices에 저장.
  - adj_entity, adj_relation 생성.

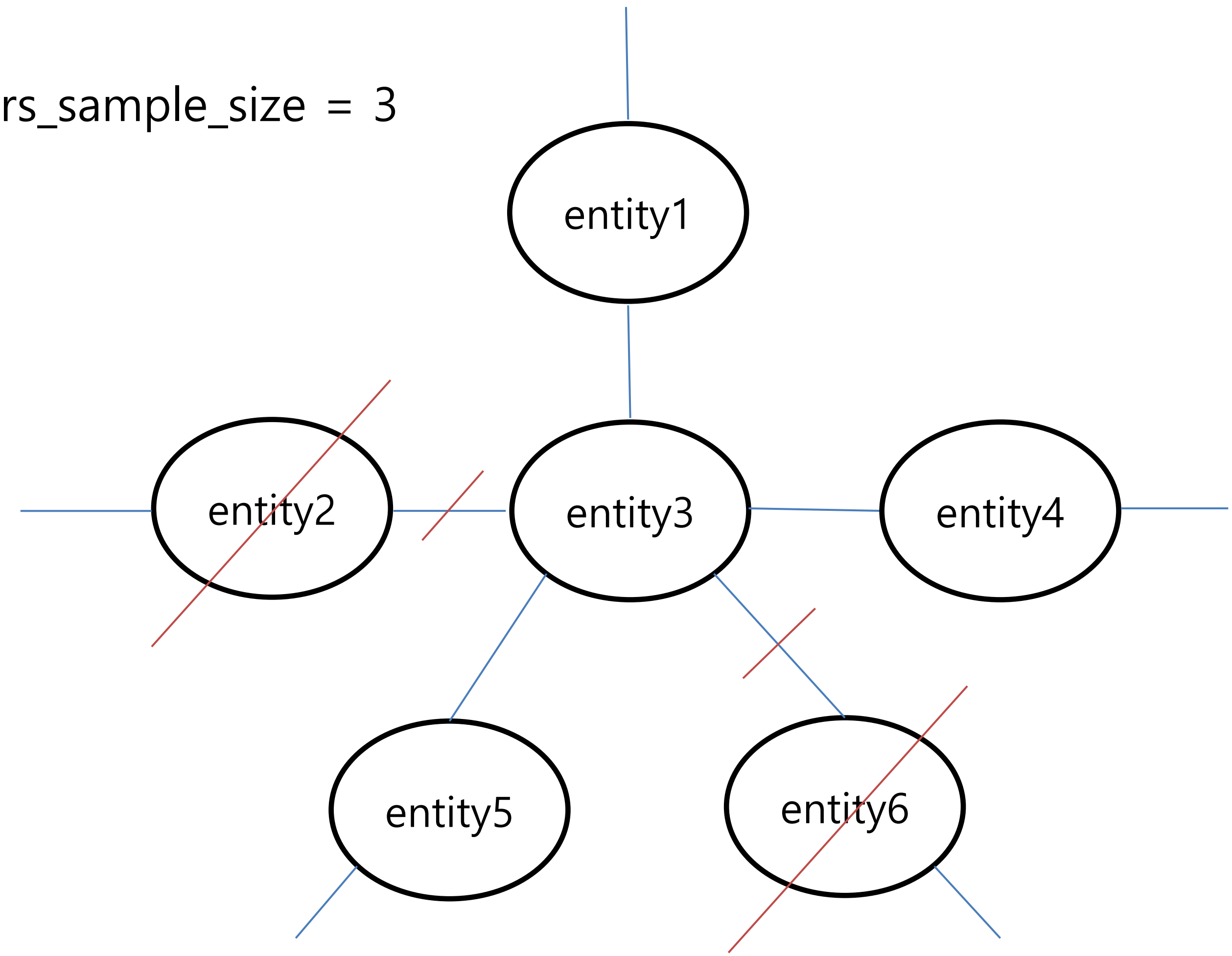
- what that function does:
  - 모든 entity에 대해 neighbors를 정의
    - 각 entity에 대응하는 adj_entity, adj_relation 생성(2차원 배열)
    - 행 길이는 전체 entity 수, 열 길이는 neighbors_sample_size

DATA LOADER

❑ KG(Knowledge Graph) adj

- Neighbors sample size(hyperparameter) 만큼 random sampling

Ex) neighbors_sample_size = 3



DATA LOADER

- data_loader()
  - load_rating()을 통해 n_user, n_item과 train / eval / test로 split된 data를 받고
  - load_kg()를 통해 n_entity, n_relation과 adj_entity, adj_relation을 받는다.

- what that function does:
  - rating_final.txt -> rating_final.npy 변환
  - train / eval / test data 로 split
  - kg_final.txt -> kg_final.npy 변환
  - Hyperparameter(neighbors_sample_size)에 맞는 adj_entity, adj_relation 생성

MODEL

MODEL

- class KGCN
  - def __init__(self, args, n_user, n_entity, n_relation, adj_entity, adj_relation):
    - self._parse_args(args, adj_entity, adj_relation)
    - self._build_inputs()
    - self._build_model(n_user, n_entity, n_relation)
    - self._build_train()
  - get_initializer() : weight 초기 설정(초기화)
    - xavier_initializer() 사용

MODEL

- class KGCN
  - def __init__(self, args, n_user, n_entity, n_relation, adj_entity, adj_relation):
    - self._parse_args(args, adj_entity, adj_relation)
    - self._build_inputs()
    - self._build_model(n_user, n_entity, n_relation)
    - self._build_train()
  - get_initializer() : weight 초기 설정(초기화)
    - xavier_initializer() 사용

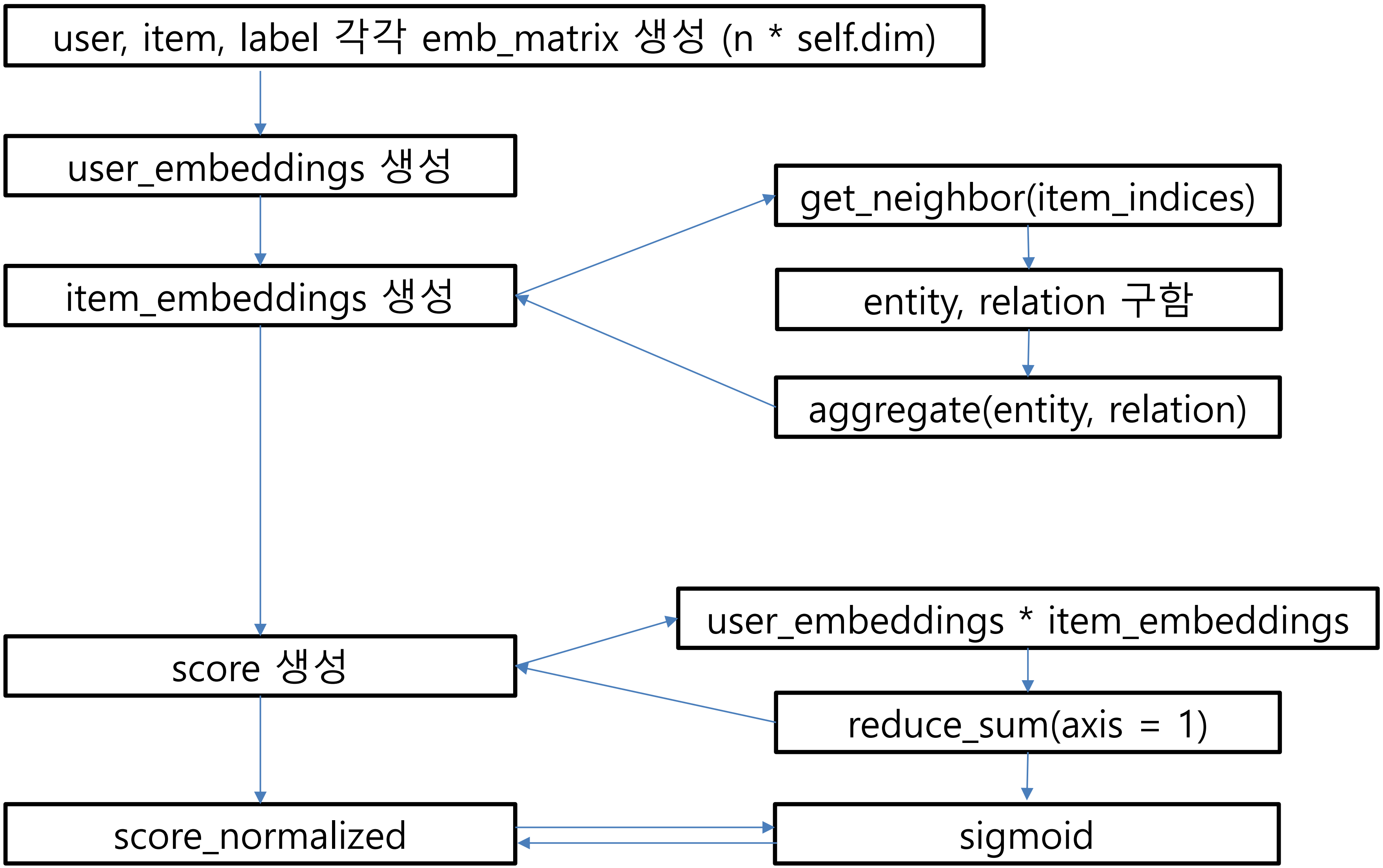
MODEL

- class KGCN
  - _bulid_inputs() :
    - tf.placeholder를 이용한 1차원 tensor 생성
      - user_indices, item_indices, labels(=y) 생성
  - _build_model() :
    - emb_matrix 생성
      - user_emb_matrix, entity_emb_matrix, relation_emb_matrix
      - shape : [n_000, self.dim] (dim은 hyperparameter, default=32)
    - entites, relations 생성 by get_neighbors(item_indices)
  - get_neighbors():
    - hyper-parameter n_iter에 근거한 neighbors_entites, neighbor_relations 생성
    - n_iter : number of iterations when computing entity representation, default=1
      - n_hope
  - _build_train():
    - base_loss : tf.nn.sigmoid_cross_entropy_with_logits()
    - L2_loss : sum of tf.nn.l2_loss
      - user_emb_matrix, entity_emb_matrix, relation_emb_matrix
    - optimizer : tf.train.AdamOptimizer.minimize

MODEL

- class KGCN
  - aggregate():
    - n_iter(=hop)만큼 반복하여 hyper-parameter aggregator로 aggregation 진행
  - train():
    - sess.run([self.optimizer, self.loss], feed_dict)
    - feed_dict 은 {model.user_indices: data[start:end, 0], model.item_indices: data[start:end, 1], model.labels: data[start:end, 2]}
    - Output : labels, loss
  - eval():
    - y_true : labels, y_score = y_pred
    - AUC, F1 score 계산

MODEL


user, item, label 각각 emb_matrix 생성 (n * self.dim)
user_embeddings 생성
item_embeddings 생성
get_neighbor(item_indices)
entity, relation 구함
aggregate(entity, relation)
score 생성
user_embeddings * item_embeddings
reduce_sum(axis = 1)
score_normalized
sigmoid

TRAIN

TRAIN

- Input:
  - feed_dict = {model.user_indices: data[start:end, 0], model.item_indices: data[start:end, 1], model.labels: data[start:end, 2]}

- Output:
  - CTR evaluation
  - Top K evaluation

TRAIN

Input : user_indices, item_indices, labels

↓

user_index – item_index score 존재

↓

Loss function : tf.nn.sigmoid_cross_entropy_with_logits

↓

Optimizer : tf.train.AdamOptimizer

TRAIN

❑ 목적 : CTR(클릭율) & TOP K recommend list

```
(KGCN) C:\Users\bhs89\KGCN-master\src>C:/pythontemp/anaconda3/envs/KGCN/python.exe c:/Users/bhs89/KGCN-master/src/main.py
reading rating file ...
splitting dataset ...
reading KG file ...
constructing knowledge graph ...
constructing adjacency matrix ...
data loaded.
neighbor_entity shape :  (65536, ?)
neighbor_entity shape :  (65536, ?)
neighbor_entity shape :  (65536, ?)
neighbor_entity shape :  (65536, ?)
WARNING:tensorflow:From c:\Users\bhs89\KGCN-master\src\aggregators.py:48: calling softmax (from tensorflow.python.ops.nn_ops) with dim is deprecated and will be removed in a future version.
Instructions for updating:
dim is deprecated, use axis instead
2022-11-15 21:29:26.356131: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
epoch 0    train auc: 0.9728  f1: 0.9194    eval auc: 0.9685  f1: 0.9146    test auc: 0.9687  f1: 0.9148
precision: 0.0900      0.1000  0.0860  0.0760  0.0605  0.0430  0.0322
recall: 0.0183  0.0380  0.0680  0.1154  0.1660  0.2838  0.4067

epoch 1    train auc: 0.9779  f1: 0.9259    eval auc: 0.9701  f1: 0.9173    test auc: 0.9702  f1: 0.9175
precision: 0.0800      0.0700  0.0760  0.0790  0.0630  0.0454  0.0339
recall: 0.0148  0.0274  0.0649  0.1212  0.1910  0.3083  0.4354

epoch 2    train auc: 0.9850  f1: 0.9420    eval auc: 0.9745  f1: 0.9264    test auc: 0.9747  f1: 0.9267
precision: 0.0800      0.0950  0.0860  0.0760  0.0605  0.0468  0.0352
recall: 0.0142  0.0250  0.0675  0.1061  0.1714  0.3046  0.4241

epoch 3    train auc: 0.9894  f1: 0.9526    eval auc: 0.9766  f1: 0.9301    test auc: 0.9766  f1: 0.9303
precision: 0.1200      0.0900  0.0880  0.0690  0.0605  0.0478  0.0355
recall: 0.0238  0.0323  0.0671  0.1066  0.1742  0.2971  0.4367

epoch 4    train auc: 0.9920  f1: 0.9594    eval auc: 0.9765  f1: 0.9294    test auc: 0.9766  f1: 0.9296
precision: 0.0800      0.0850  0.0820  0.0670  0.0545  0.0426  0.0354
recall: 0.0140  0.0248  0.0450  0.1030  0.1496  0.2795  0.4311

epoch 5    train auc: 0.9935  f1: 0.9639    eval auc: 0.9762  f1: 0.9293    test auc: 0.9762  f1: 0.9296
precision: 0.0800      0.0950  0.0860  0.0710  0.0540  0.0446  0.0326
recall: 0.0123  0.0266  0.0605  0.0991  0.1565  0.2883  0.4144

epoch 6    train auc: 0.9946  f1: 0.9676    eval auc: 0.9758  f1: 0.9290    test auc: 0.9758  f1: 0.9291
precision: 0.1100      0.0750  0.0660  0.0550  0.0520  0.0400  0.0302
recall: 0.0109  0.0226  0.0500  0.0845  0.1485  0.2604  0.3821

epoch 7    train auc: 0.9954  f1: 0.9704    eval auc: 0.9753  f1: 0.9283    test auc: 0.9752  f1: 0.9286
precision: 0.1000      0.0750  0.0700  0.0650  0.0555  0.0410  0.0305
recall: 0.0101  0.0179  0.0450  0.0925  0.1529  0.2879  0.3892

epoch 8    train auc: 0.9959  f1: 0.9724    eval auc: 0.9748  f1: 0.9279    test auc: 0.9748  f1: 0.9280
precision: 0.0700      0.0550  0.0700  0.0580  0.0520  0.0378  0.0301
recall: 0.0115  0.0168  0.0404  0.0985  0.1587  0.2576  0.3963

epoch 9    train auc: 0.9963  f1: 0.9739    eval auc: 0.9745  f1: 0.9274    test auc: 0.9745  f1: 0.9274
precision: 0.0500      0.0450  0.0680  0.0580  0.0505  0.0392  0.0303
recall: 0.0058  0.0129  0.0503  0.0976  0.1520  0.2570  0.3962
```

DISCUSSION

DISCUSSION

- tf.sess.run() 이해하기

- CTR(클릭율) & top K
  - AUC, F1 score만 기재
  - 결국에는 어떤 항목을 추천하는지 item을 알려주지 않는다..?
  - 우리가 원하는건 각 user에게 추천할 item이나 해당 item을 클릭할 확률을 구하고 싶은 건데 어떻게 표현되어 있는지 모르겠다.

- Input labels
  - user_index, item_index, 0 or 1
  - Label = 0 인 것 중에, recommendation 작업이 들어가는 줄 알았는데 label=0 을 unwatched_set에 의해 생성했다…. user_neg_rating에서 label을 0으로 해야하지 않나

THANK YOU