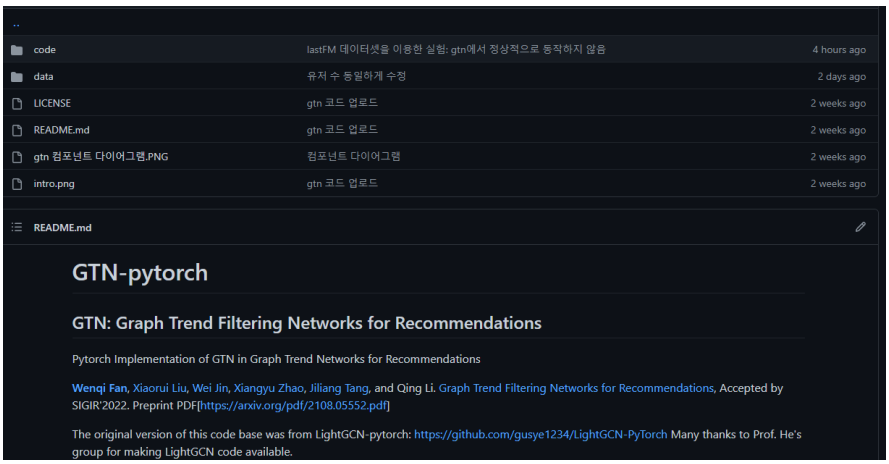
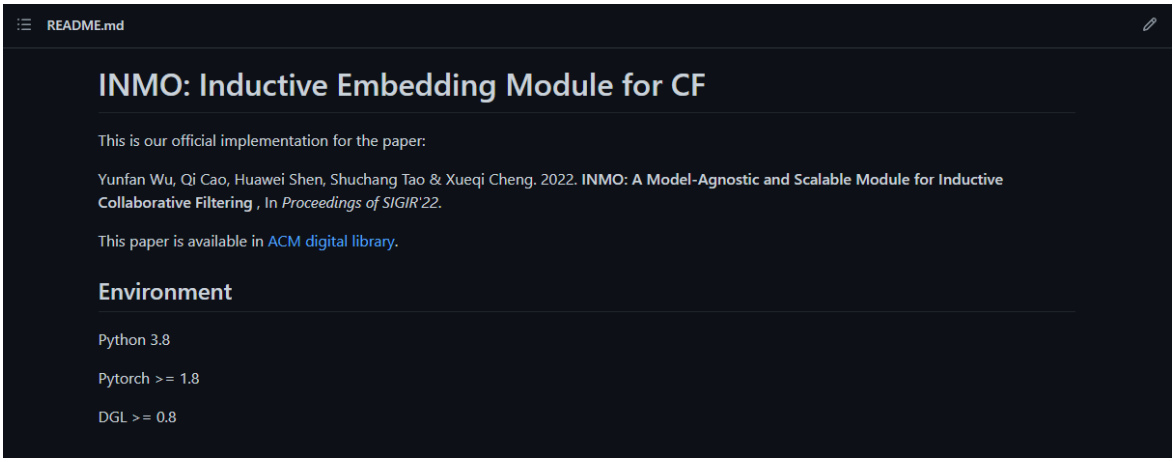
1. 논문에 사용되는 패키지 명칭 정리



[gtn_lib]

[lgcn_lib]



[igcn_lib]

2. MovieLens를 데이터셋으로 각 패키지를 구동했을 때의 결과

- 모든 패키지에서 공통적으로 적용한 config

- epochs: 100
- train_batch_size: 512,
- test_batch_size: 100
- embedding size: 64
- lr: 0.001
- decay: 0.0001
- dropout: 0

패키지 명	추천 모델	ndcg(%)
gtn_lib	GTN	44.4
	IGTN	**44.3**
lgcn_lib	LGCN	44.7
igcn_lib	LGCN	44.7
	IGCN	45.6

- gtn_lib는 lgcn_lib를 기반으로 구현됨
  - lgcn_lib와 igcn_lib에서 각각 구현된 LGCN은 성능 상 차이가 없음
  - 정규화, 스케일링 등에 의해 GTN 모델의 성능이 낮은 것이라고는 생각할 수 없음
- MovieLens 데이터셋은 논문에서 사용된 데이터셋이 아니기 때문에 GTN 모델의 유용성을 입증하지 못할 수 있음

Table 2: The comparison of overall performance.

Datasets		Gowalla		Yelp2018		Amazon-Book		LastFM	
Metrics		Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
Method	MF	0.1299	0.111	0.0436	0.0353	0.0252	0.0198	0.0725	0.0614
	NeuCF	0.1406	0.1211	0.045	0.0364	0.0259	0.0202	0.0723	0.0637
	GC-MC	0.1395	0.1204	0.0462	0.0379	0.0288	0.0224	0.0804	0.0736
	NGCF	0.156	0.1324	0.0581	0.0475	0.0338	0.0266	0.0774	0.0693
	Mult-VAE	0.1641	0.1335	0.0584	0.045	0.0407	0.0315	0.078	0.07
	DGCF	0.1794	0.1521	0.064	0.0522	0.0399	0.0308	0.0794	0.0748
	LightGCN	0.1823	0.1553	0.0649	0.0525	0.042	0.0327	0.085	0.076
	GTN (Ours)	0.187	0.1588	0.0679	0.0554	0.045	0.0346	0.0932	0.0857
Relative Improvement (%)		2.59	2.26	4.62	5.59	7.15	5.95	9.61	12.77

- 위 그림은 gtn_lib 논문에 제시된 GTN 모델의 성능 비교 표
- **LastFM** 데이터셋에 대해 LightGCN보다 12% 이상의 성능 향상을 보임

Dataset

We provide three processed datasets: Gowalla, Yelp2018 and Amazon-book and one small dataset LastFM.

see more in `dataloader.py`

```
424     4076    993
1311    1574    957
1671    410     146
997     1686    1417
1422    2304    159
121     1928    93
168     117     496
574     1057    374
833     594     5786
110     940     265
848     344     298
86      675     249
976     42      5
686     769     1986
1660    3273    3
1850    931     1517
769     2318    62
116     1889    248
1001    4084    890
1768    2250    246
1832    2878    502
314     588     450
1746    299     1977
1259    911     210
275     484     166
829     730     108
920     111     4
417     2358    5439
107     745     538
```

- lgcn_lib에서는 gowalla, yelp, amazon과 같은 정제된 데이터와 함께 작지만 정제되지는 않은 LastFM 데이터셋을 제공함

- gtn_lib의 논문 표에 제시된 바와 같이 LastFm에서 GTN 모델은 LastGCN 모델에 비해 높은 성능을 보였기 때문에 epoch, embedding size 등의 차이를 감안하더라도 유의미한 성능 향상을 보일 것이라는 가설을 세움

3. LastFm 데이터셋을 이용한 실험

```
[TEST]
{'precision': array([0.06964478]), 'recall': array([0.24958578]), 'ndcg': array([0.19013062])}
EPOCH[101/101] loss0.046-|Sample:1.69|
```

- 실험 결과 lgcn_lib의 LightGCN 모델을 이용했을 때 19.1%(ndcg)의 성능을 보임

```
EPOCH[91/101] loss 30.7243  30.7242  0.0001 - |Sample:1.71|  |  00:09mins
EPOCH[92/101] loss 32.0622  32.0621  0.0001 - |Sample:1.99|  |  00:04mins
EPOCH[93/101] loss 31.1646  31.1645  0.0001 - |Sample:1.83|  |  00:03mins
EPOCH[94/101] loss 30.9815  30.9815  0.0001 - |Sample:1.43|  |  00:03mins
EPOCH[95/101] loss 32.6042  32.6041  0.0001 - |Sample:1.79|  |  00:03mins
EPOCH[96/101] loss 31.2125  31.2124  0.0001 - |Sample:1.50|  |  00:03mins
EPOCH[97/101] loss 31.8934  31.8933  0.0001 - |Sample:1.76|  |  00:03mins
EPOCH[98/101] loss 31.2169  31.2168  0.0001 - |Sample:1.49|  |  00:03mins
EPOCH[99/101] loss 30.3663  30.3662  0.0001 - |Sample:2.05|  |  00:04mins
EPOCH[100/101] loss 30.5217  30.5216  0.0001 - |Sample:2.06|  |  00:04mins
########################
[TEST]

Testing EPOCH[101/101]  loss 30.6054  30.6053  0.0001 - |Sample:1.55| | Results Top-k (pre, recall, ndcg): 0.00347, 0.01192, 0.00
84
```

- 반면, gtn_lib의 GTN모델을 이용해 실험해본 결과 위와 같이 학습이 정상적으로 되지 않음
  - 성능(ndcg)은 0.84%
- 두 모델의 임베딩 룩업 테이블을 비교해본 결과, 완전히 동일
  - 학습 수준에서 문제가 있다고 판단함

```
'''
# ? seed값이 동일해서 임베딩 룩업 테이블은 LightGCN(lgcn_lib)과 정확히 같음
# ! tensor([[ 27.2716,  -0.9661,  15.4678,  ..., -11.3022,  24.2125,   0.0000],
# ! [ -0.2903,  -0.1734,  -0.0956,  ...,   0.0739,   0.0000,  -0.1329],
# ! [ 38.0587,  28.5872, -34.5981,  ..., -29.6428, -27.6292,  -8.7488],
# ! ...,
# ! [  1.1554,  -0.3685,  -0.7906,  ...,   0.0000,   0.7672,  -0.4871],
# ! [  0.4570,  -1.2098,   0.3067,  ...,  -0.5867,   1.4079,  -1.2274],
# ! [  1.1645,  -0.2060,   0.0000,  ...,   0.0000,  -0.9328,  -0.7533]],
'''
```

[gtn_lib의 GTN모델의 학습된 임베딩 결과]

```
'''
# ! tensor([[-0.0238,  0.0356, -0.0534,  ...,  0.0160, -0.0127,  0.0202],
# ! [ 0.0137,  0.0310, -0.0025,  ..., -0.0023, -0.0143,  0.0744],
# ! [-0.0280, -0.0281,  0.0123,  ..., -0.0215,  0.0263,  0.0219],
# ! ...,
# ! [ 0.0102, -0.0145,  0.0211,  ...,  0.0178,  0.0038, -0.0510],
# ! [-0.0097, -0.0520, -0.0216,  ...,  0.0090, -0.0209,  0.0100],
# ! [ 0.0287,  0.0105, -0.0151,  ...,  0.0376,  0.0230,  0.0104]],
'''
```

[lgcn_lib의 LightGCN모델의 학습된 임베딩 결과]

- 데이터가 정제되지 않았음을 문제로 삼고 MovieLens 데이터셋을 만들었던 것과 같은 방법으로 정제된 포맷으로 LastFM 데이터셋을 변경함

```
0 61 67 55 62 69 68 38 51 54 64 44 43 58 48 53 66 39 46 57 56 60 65 45 50 41 40
1 73 77 78 74
2 101 94 96 100 39 90 104 80 99 93 82 50 107 81 88 89 105 103 98 83 102 87 95 92 97
3 110 113 121 129 119 111 127 132 123 122 131 126 109 124 115 125 114 133 128 112
4 55 150 165 140 147 138 142 148 155 145 162 167 166 154 152 157 134 158
5 181 169 170 177 189 187 172 188 183 179 173 168 185 182 176 171 180
6 201 204 196 198 200 193 203 51 197 192 143 66 177 195 205 191 169 190 194
7 221 223 229 236 210 231 227 215 206 224 237 220 230 222 226 211 208 216 239 213 238 214 219 228 5
8 252 253 240 265 267 110 270 261 247 248 255 246 245 259 249 125 243 254 112 260 266 264 258 257 26
9 286 292 280 291 275 296 172 274 285 295 272 282 290 279 283 294 276 288 289 277 281 273
10 301 322 313 318 312 302 304 326 328 307 309 319 327 300 310 233 311 305 320 306 316 303 315 323 2
11 338 341 336 331 343 345 329 342 333 340 339 334 344 332 337
12 360 352 351 358 362 348 355 367 347 361 368 353 359 357 365 363 366
13 374 383 376 371 397 373 122 111 375 370 372 396 124 378 377 380 387 116 391 389 49 379 381 384 39
14 402 219 409 335 406 408 404 413 401 399 412 398 410 193 405 414
15 346 415 430 429 426 423 431 420 428 427 231 421 425
16 450 451 440 456 432 433 442 437 441 438 455 92 444 447 449 448 436 221 452 51 439 446 457 434
17 475 320 468 473 460 465 476 462 471 469 458 461 464 467 482 79 474 484 480 472 483 113 463
18 486 493 488 492 489 288 280 487 337 334 412 407 494 495 496 490 497 499 498
19 505 509 533 321 519 524 507 537 503 501 510 515 529 516 535 513 514 530 511 522 531 528 520 534 5
20 560 549 562 554 550 540 542 546 564 543 563 556 553 558 548 539 559 557 555 565 545 243 547 382
21 572 569 571 574 578 579 147 256 576 577 489 570 486 88 191 330 580 568 581 420 573
22 83 175 582 276 583 486 278 586 585 197 296 176 498 195 590 168 177 169 283 334 587
23 4 605 613 601 614 595 531 592 617 598 603 609 596 593 604 594 611 602 608 600 441 591 612 607 18
24 402 627 640 246 626 618 619 623 639 629 621 91 633 194 624 431 447 628 638 625 401 81 637 98 279
25 643 667 664 669 653 656 652 648 655 654 672 646 661 107 205 670 659 671 62 657 69 641 651 663 59
26 678 677 181 673 339 286 336 682 272 310 684 334 681 675 676 685 583
```

- MovieLens와 같이 processing 코드는 파일을 읽고 딕셔너리 형태로 아이템을 유저에 추가하도록 함

```python
PATH = '/home/hwiric/Internship/LastFM/lastfm/test1.txt'

f = open(PATH, 'r')
rdr = csv.reader(f)

# 유저-아이템 딕셔너리 생성
d = {}
users=[]
for l in rdr:
    s = ''.join(l).split()
    # print(s)
    if s[0] in d:
        d[s[0]].append(s[1])
    else:
        d[s[0]] = [s[1]]
    users.append(d[s[0]])
f.close()

# print(len(users))

sorted_dict=sorted(d.items(), key=lambda x: int(x[0]))

# print(sorted_dict)

# cnt=0
# txt 파일 생성
with open('/home/hwiric/Internship/LastFM/lastfm/test.txt', 'w', encoding='UTF-8') as f:
    for line in sorted_dict:
        user, item=line[0], line[1]

        user = str(int(user)-1)
        item = ' '.join(item)
        f.write(f'{user} {item}\n')
# print(cnt)
```

[reader.py]

3.1. 실험결과

```
EPOCH[91/101] loss 0.0561  0.0532  0.0029 - |Sample:1.53|  |  00:09mins
EPOCH[92/101] loss 0.0582  0.0553  0.0029 - |Sample:1.48|  |  00:03mins
EPOCH[93/101] loss 0.0547  0.0518  0.0029 - |Sample:1.55|  |  00:03mins
EPOCH[94/101] loss 0.0552  0.0522  0.0029 - |Sample:1.40|  |  00:03mins
EPOCH[95/101] loss 0.0546  0.0517  0.0030 - |Sample:1.56|  |  00:03mins
EPOCH[96/101] loss 0.0539  0.0510  0.0030 - |Sample:1.41|  |  00:02mins
EPOCH[97/101] loss 0.0535  0.0505  0.0030 - |Sample:1.40|  |  00:03mins
EPOCH[98/101] loss 0.0527  0.0497  0.0030 - |Sample:1.54|  |  00:03mins
EPOCH[99/101] loss 0.0556  0.0525  0.0030 - |Sample:1.39|  |  00:03mins
EPOCH[100/101] loss 0.0517  0.0487  0.0030 - |Sample:1.53|  |  00:03mins
##########################
[TEST]

Testing EPOCH[101/101]  loss 0.0492  0.0462  0.0031 - |Sample:1.39| | Results Top-k (pre, recall, ndcg): 0.07018, 0.24934, 0.19136
```

[gtn_lib의 GTN모델]

```
EPOCH[91/101] loss0.049-|Sample:1.58|
EPOCH[92/101] loss0.050-|Sample:1.37|
EPOCH[93/101] loss0.047-|Sample:1.37|
EPOCH[94/101] loss0.047-|Sample:1.47|
EPOCH[95/101] loss0.047-|Sample:1.36|
EPOCH[96/101] loss0.047-|Sample:1.50|
EPOCH[97/101] loss0.046-|Sample:1.36|
EPOCH[98/101] loss0.045-|Sample:1.47|
EPOCH[99/101] loss0.047-|Sample:1.36|
EPOCH[100/101] loss0.044-|Sample:1.47|
[TEST]
{'precision': array([0.06967169]), 'recall': array([0.24791962]), 'ndcg': array([0.19142373])}
EPOCH[101/101] loss0.043-|Sample:1.38|
```

[lgcn_lib의 LightGCN모델]

패키지 명	추천 모델	recall(%)	ndcg(%)
gtn_lib	GTN	24.93	19.136
lgcn_lib	LGCN	24.79	19.142

- MovieLens와 비슷한 경향을 보임

4. 고찰

- IGCN이 LGCN보다 우수한 성능을 보임은 입증했지만 GTN이 LightGCN보다 우수함은 아직까지도 입증이 되지 않은 상황

  - 초기 gowalla 데이터셋으로 실험 중

```
RuntimeError: CUDA out of memory. Tried to allocate 12.00 MiB (GPU 0; 7.80 GiB total capacity; 38.19 MiB already allocated; 15.4
4 MiB free; 46.00 MiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to a
void fragmentation.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

    - embedding size를 64로 해도 위와 같은 에러가 발생됨

      - 초기 세팅은 256 <- 불가능

      - lgcn_lib의 LightGCN **임베딩 사이즈를 32**로 변경해서 실험 진행

        - gtn_lib의 GTN도 동일한 조건으로 진행

- 실험 결과

```
# GTN-gowalla
EPOCH[91/101] loss 0.0293  0.0248  0.0045 - |Sample:30.52|  |  02:48mins
EPOCH[92/101] loss 0.0291  0.0245  0.0046 - |Sample:34.99|  |  02:24mins
EPOCH[93/101] loss 0.0293  0.0247  0.0046 - |Sample:33.12|  |  02:22mins
EPOCH[94/101] loss 0.0291  0.0245  0.0046 - |Sample:30.87|  |  02:20mins
EPOCH[95/101] loss 0.0287  0.0241  0.0046 - |Sample:34.10|  |  02:23mins
EPOCH[96/101] loss 0.0286  0.0240  0.0047 - |Sample:34.06|  |  02:24mins
EPOCH[97/101] loss 0.0285  0.0239  0.0047 - |Sample:32.06|  |  02:21mins
EPOCH[98/101] loss 0.0290  0.0243  0.0047 - |Sample:34.68|  |  02:24mins
EPOCH[99/101] loss 0.0288  0.0241  0.0047 - |Sample:29.73|  |  02:19mins
EPOCH[100/101] loss 0.0289  0.0241  0.0047 - |Sample:31.44|  |  02:21mins
#########################
[TEST]

Testing EPOCH[101/101]  loss 0.0284  0.0237  0.0048 - |Sample:36.56| | Results Top-k (pre, recall, ndcg): 0.04743, 0.15987, 0.13102
```

[GTN(gtn_lib)]

```
# LightGCN-gowalla
EPOCH[91/101] loss0.020-|Sample:32.13|
EPOCH[92/101] loss0.020-|Sample:33.01|
EPOCH[93/101] loss0.020-|Sample:32.09|
EPOCH[94/101] loss0.020-|Sample:33.06|
EPOCH[95/101] loss0.020-|Sample:32.29|
EPOCH[96/101] loss0.019-|Sample:31.71|
EPOCH[97/101] loss0.019-|Sample:32.04|
EPOCH[98/101] loss0.020-|Sample:32.19|
EPOCH[99/101] loss0.019-|Sample:31.77|
EPOCH[100/101] loss0.020-|Sample:31.76|
[TEST]
{'precision': array([0.04917777]), 'recall': array([0.16356285]), 'ndcg': array([0.13753617])}
```

[LightGCN(lgcn_lib)]

- GTN이 LightGCN의 문제를 개선한 부분에 대한 정확한 이해가 필요함

  - 어떻게 smoothness 문제를 해결했는지 막연한 상황

  - 단순히 논문에 제시된 표만을 보고 나아졌을 거라고 판단했음