1차 실험 결과

```
# IGTN-MovieLen
Testing EPOCH[91/100]  loss 0.1682  0.1654  0.0028 - |Sample:0.33| | Results Top-k (pre, recall, ndcg): 0.36277, 0.24925, 0.44152

EPOCH[91/100] loss 0.1682  0.1654  0.0028 - |Sample:0.33|  |  00:01mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[92/100] loss 0.1717  0.1689  0.0028 - |Sample:0.42|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[93/100] loss 0.1699  0.1671  0.0028 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[94/100] loss 0.1719  0.1691  0.0029 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[95/100] loss 0.1710  0.1681  0.0029 - |Sample:0.41|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[96/100] loss 0.1712  0.1684  0.0029 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[97/100] loss 0.1695  0.1666  0.0029 - |Sample:0.36|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[98/100] loss 0.1648  0.1618  0.0029 - |Sample:0.41|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[99/100] loss 0.1694  0.1664  0.0030 - |Sample:0.32|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152
EPOCH[100/100] loss 0.1660  0.1630  0.0030 - |Sample:0.32|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24925, 0.44152

# loss: 16.6%          (+0.02%)
# precision: 36.277%   (+0.065%)
# recall: 24.925%      (+0.211%)
# ndcg: 44.152%        (-0.042%)
```

```
# GTN-MovieLen
Testing EPOCH[91/100]  loss 0.1691  0.1663  0.0028 - |Sample:0.33| | Results Top-k (pre, recall, ndcg): 0.36212, 0.24714, 0.44194

EPOCH[91/100] loss 0.1691  0.1663  0.0028 - |Sample:0.33|  |  00:01mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[92/100] loss 0.1718  0.1690  0.0028 - |Sample:0.42|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[93/100] loss 0.1687  0.1659  0.0028 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[94/100] loss 0.1724  0.1696  0.0028 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[95/100] loss 0.1710  0.1682  0.0029 - |Sample:0.41|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[96/100] loss 0.1723  0.1694  0.0029 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[97/100] loss 0.1706  0.1677  0.0029 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[98/100] loss 0.1644  0.1615  0.0029 - |Sample:0.42|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[99/100] loss 0.1678  0.1649  0.0029 - |Sample:0.33|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194
EPOCH[100/100] loss 0.1658  0.1628  0.0030 - |Sample:0.34|  |  00:00mins | Results val Top-k (recall, ndcg):  0.24714, 0.44194

# loss: 16.58%
# precision: 36.212%
# recall: 24.714%
# ndcg: 44.194%
```

IGCN(1)

```python
def inductive_rep_layer(self, feat_mat):
    padding_tensor = torch.empty([max(self.feat_mat.shape) - self.feat_mat.shape[1], self.embedding_size],
                                dtype=torch.float32,
                                device=self.device)
    padding_features = torch.cat([self.embedding.weight, padding_tensor], dim=0)

    # print(padding_tensor.shape)
    # print(padding_features.shape)

    row, column = feat_mat.indices()
    g = dgl.graph((column, row), num_nodes=max(self.feat_mat.shape), device=self.device)
    x = dgl.ops.gspmm(g, 'mul', 'sum', lhs_data=padding_features, rhs_data=feat_mat.values())
    x = x[:self.feat_mat.shape[0], :]   #[ : 2063, : ]
    return x

def get_rep(self):
    feat_mat = NGCF.dropout_sp_mat(self, self.feat_mat)
    representations = self.inductive_rep_layer(feat_mat)        # ! 이거를 잘라가면 됨

    # ! ===================================================================================================
    # LightGCN으로 학습되기 전에 반환
    return representations
    # ! ===================================================================================================
```

IGCN(2)

```
all_layer_rep = [representations]
row, column = self.norm_adj.indices()
g = dgl.graph((column, row), num_nodes=self.norm_adj.shape[0], device=self.device)

# 밑에는 LightGCN 로직
for _ in range(self.n_layers):
    representations = dgl.ops.gspmm(g, 'mul', 'sum', lhs_data=representations, rhs_data=self.norm_adj.values())
    all_layer_rep.append(representations)
all_layer_rep = torch.stack(all_layer_rep, dim=0)
final_rep = all_layer_rep.mean(dim=0)

# print('\n########################################INMO EMBEDDING#########################################\n')
# # print(type(final_rep))  # <class 'torch.Tensor'>
# print(final_rep.shape)      # torch.Size([2063, 64])
# # print(final_rep.tolist())
# print('\n###############################################################################################\n')
# exit()

return final_rep
```

GTN(1)

```python
def computer(self, corrupted_graph=None):
    """
    propagate methods for lightGCN
    """

    # print('\n##########################################################################################\n')
    # # # print(type(self.embedding_user))    # <class 'torch.nn.modules.sparse.Embedding'>
    # # # # print(self.embedding_user.shape)
    # # # print('\n\n\n')
    # # # print(type(self.embedding_user.weight))     # <class 'torch.nn.parameter.Parameter'>
    # # print(self.embedding_user.weight.shape)    # torch.Size([458, 64])
    # # print(self.embedding_item.weight.shape)     # torch.Size([1605, 64])            # 총합 2063
    # print('\n##########################################################################################\n')
    # exit()


    users_emb = self.embedding_user.weight
    items_emb = self.embedding_item.weight
    all_emb = torch.cat([users_emb, items_emb])     # ! 여기에 붙이면 됨 → 아님
    # <class 'torch.Tensor'>
    # torch.Size([2063, 64])
```

GTN(2)

```python
# our GCNs
x = all_emb
rc = g_droped.indices()
r = rc[0]
c = rc[1]
num_nodes = g_droped.shape[0]
edge_index = SparseTensor(row=r,
                          col=c,
                          value=g_droped.values(),
                          sparse_sizes=(num_nodes, num_nodes))
emb, embs = self.gp.forward(x, edge_index, mode=self.args.gcn_model)
light_out = emb

# 합친 뒤 학습시키고 분리하는 것 같음
users, items = torch.split(light_out, [self.num_users, self.num_items])
return users, items
```

GTN(3)

```python
# ? ===========================================연결===========================================================
import igcn_copy

final_rep=igcn_copy.main()

all_emb=torch.split(final_rep, [self.num_users, self.num_items])
e_user, e_item=all_emb[0], all_emb[1]

emb_user=nn.Embedding.from_pretrained(e_user, freeze=False)
emb_item=nn.Embedding.from_pretrained(e_item, freeze=False)

'''
<class 'torch.nn.modules.sparse.Embedding'>
<class 'torch.nn.parameter.Parameter'>
torch.Size([458, 64])

<class 'torch.nn.modules.sparse.Embedding'>
<class 'torch.nn.parameter.Parameter'>
torch.Size([1605, 64])
'''
# print(type(emb_user))
# print(type(emb_user.weight))
# print(emb_user.weight.shape)
# print()
# print(type(emb_item))
# print(type(emb_item.weight))
# print(emb_item.weight.shape)

self.embedding_user = emb_user
self.embedding_item = emb_item
```

문제점

- **IGCN은 BPR loss + self-enhanced loss를 사용함**
  - IGTN은 GTN 코드를 이용해서 구현됐기 때문에 BPR loss로만 학습됨

- IGTN이 self-enhanced loss 역시 고려했을 때 성능이 유의미하게 향상될 경우
  - Inductive 시나리오에서 테스트 해봐야 함
  - MovieLens(작은 데이터셋)를 Inductive 시나리오에 맞게 가공해야 함