10/12

1. GNN_DOVE - Learning Data

Dockground dataset 1.0중에서 29개 그룹 데이터를 사용하기로 했습니다

To train and test GNN-DOVE, we first used the Dockground dataset 1.0

resulted in 29 groups (Table 1). In Table 1, complexes (PDB IDs) of the same group are shown in lower case in a parenthesis followed by the PDB ID of the representative

Fold	PDB ID
1	1A2K, 1E96 (1he1, 1he8, 1wq1), 1F6M, 1MA9 (2btf), 1G20, 1KU6, 1T6G, 1UGH, 1YVB, 2CKH, 3PRO
2	1AKJ (1p7q, 2bnq), 1DFJ, 1NBF (1r4m, 1xd3, 2bkr), 1GPW, 1HXY, 1U7F, 1UEX, 1ZY8, 2GOO, 1EWY
3	1AVW (1*b*th, 1bui, 1cho, 1ezu, 1ook, 1oph, 1ppf, 1tx6, 1xx9, 2fi4, 2kai, 1r0r, 2sni, 3sic)
4	1BVN (1tmq), 1F51, 1FM9, 1A2Y (1g6v, 1gpq, 1jps, 1wej, 1l9b, 1s6v), 1W1I, 2A5T, 3FAP

There are in total 29 representative targets shown in the upper case; targets in the lower case in a parenthesis indicate that they belong to the same group.

2. GNN_DOVE - Learning code

GNN model에
train model이라는
함수가 있어서
활용을
해보았습니다.

```
def train_model(self,data,device):
        #get data
        c_hs, c_adjs1, c_adjs2, c_valid, num_atoms = data
        c_hs = self.embede(c_hs)
        c_adjs2=self.Formulate_Adj2(c_adjs2,c_valid,num_atoms,device)
        #then do the gate
        c_hs=self.embede_graph((c_hs,c_adjs1,c_adjs2))
        #if self.params['debug']:
        #    print("embedding size",c_hs.size())
        #sum based on the atoms
        c_hs=self.Get_Prediction(c_hs,num_atoms)
        c_hs = self.fully_connected(c_hs)
        c_hs = c_hs.view(-1)
        return c_hs
```

2. GNN_DOVE - train_model

train_model로 데이터들을 통과를 시키고, 그 후에 test_model로 평가를 했을 때의 결과

평가를 한 값이 들쑥날쑥 해서 같은 데이터를 30번 돌려봤습니다.

[0.14237323, 0.007679435, 0.6854278, 0.4819477, 0.0007678783, 0.013903144, 0.0049358886, 0.4356004, 0.050293483, 0.88183063, 0.5500397, 0.00011896027, 0.9871889, 0.000858775, 0.019888029, 0.9953654, 0.8974936, 0.7231499, 0.947176, 5.2751643e-06, 0.008831843, 0.34686822, 0.13666049, 0.0006793438, 0.23550095, 0.99934775, 0.20501481, 0.5591756, 0.019033348, 0.024969764]

train을 안하고 30번을 돌렸을때,

4.5064746e-05, 3.537028e-05, 0.006264966, 0.05788756, 0.0016862344, 0.114675775, 0.0031173918, 0.43245026, 0.00011937559, 0.96945524, 0.0018846215, 0.1354497, 0.003498039, 1.0285755e-05, 0.017617103, 0.030966245, 0.07696375, 0.16977786, 0.9969073, 5.973352e-06, 0.2710371, 5.9755752e-05, 0.0012471189, 0.0033148793, 0.53525823, 0.033314835, 0.0009806397, 0.00022628697, 0.2055349, 0.00022294078

2. GNN_DOVE - 결과

triain을 하고 한 결과에서 0.5를 넘는 평가 값의 갯수는 10개
train안하고 돌린 결과에서 0.5를 넘는 평가 값의 개수는 3개

train을 하면 들쑥날쑥 한 값이 그나마 나아지지만, 정확하게 학습이 되고 있는것 같지는 않습니다.

3. GNN_DOVE - 개선

논문에 나온 Traing Network 부분에 보면 어떻게 train을 했는지 나와있어서 해볼려고 했습니다.

loss = CrossEntorpyLoss
optimizer = Adam
dropout = 0.3 # for every layer
epochs = 100
batch_size = 32
Glorot uniform
bias = 0
learning_rates = [0.2, 0.02, 0.002, 0.0002]
Adam_decay = [0.1, 0.01, 0.001, 0.0001, 0.00001]

3. GNN_DOVE - 개선

개선을 할려고 pytorch에 익숙하지 않아서 keras로 모델도 만들고, learning을 해볼려고 했는데, 제대로 안돼서 멈췄습니다.

기존 모델은 있으니까 Pytorch를 익혀서 learing을 시도하는게 더 빠를거 같아서 torch 열심히 배우고 있습니다.