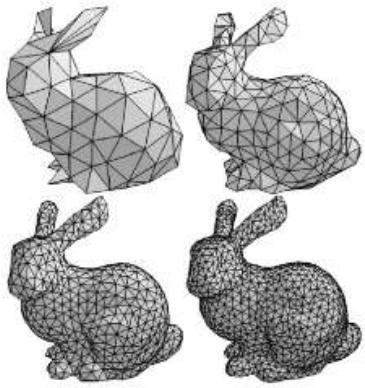
고급딥러닝 발표계획서

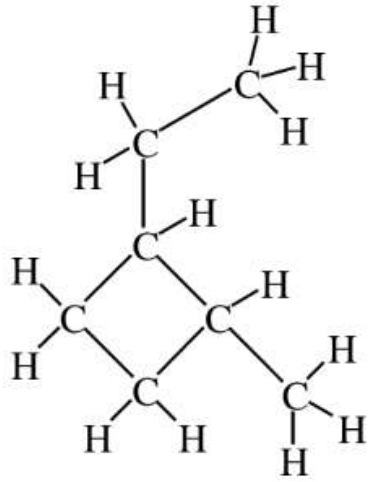
5조 김보람

1. GNN에 대한 간단한 설명

Social Graph



3D Mesh



Molecular Graph

Graph는 node와 edge로 이루어져 있고, node는 input data를 의미하고 edge는 두 data간의 관계를 의미한다.

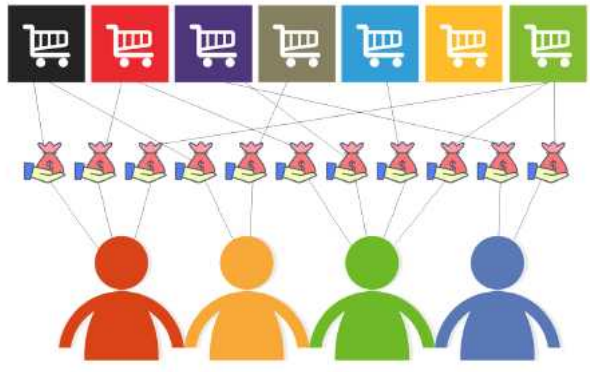
Social Graph에서 node는 한 명의 사람을 뜻하고 edge는 두 사람간의 관계 정보를 뜻하는 것과 비슷하다.

node간의 관계를 표현하기 위해 Adjacency matrix를 활용한다.

node의 feature(matrix)을 이용해 노드의 특성 정보를 나타내기도 한다.

2. 문제정의

신용카드 사기거래 분석을 위해서 사기거래의 금액을 활용하여 진행하는 경우가 많다. 사기거래 탐지를 위해서 그래프 기반 분석을 활용하려 한다. 사기 거래에 일반적으로 사용되는 그래프는 이분형 그래프와 삼분형 그래프가 있다.

이분형 그래프	삼분형 그래프
	

이분형 그래프는 고객과 상점을 노드로 잡고, 거래가 있으면 edge를 1로 표현한다. edge의 weight는 amt이다. 이분형 그래프는 개별 거래에서 이상형 탐지를 효율적으로 탐지하기 어렵다는 단점이 있다.
삼분형 그래프는 고객과 상점, 거래를 노드로 표현한다. 이분형 그래프보다 개별 거래의 이상형 탐지를 찾을 순 있지만 분석 시간이 오래걸리고 효율적이지 않다.

3. 데이터 소개

2013년 9월의 유럽의 신용카드 거래 데이터이다.
해당 데이터는 23개의 설명변수를 가지고 있지만, 모델에서 활용하는 변수는 다음과 같다.

변수	설명
index	행 번호
cc_num	고객 카드 넘버
trans_date_and_time	거래 시간
amt	거래 금액
is_fraud	사기거래여부(0:정상거래, 1:사기거래)

4. 사용하는 방법

pytorch에서 제공하는 PyG모델 중 GCNConv을 활용하여 해당 데이터를 분석하려 한다.
(참고: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GCNConv.html)
사기거래에 일반적으로 사용되는 이분형 그래프와 삼분형 그래프의 단점을 보완한 모델을 이용한다.
각 고객의 사기 거래를 더 효율적으로 분석하기 위해 각 고객별 거래 시간의 차이를 계산하는 weight matrix를 정의하고 학습시킨다.
시간 t, s를 활용해 (t ≠ s) exp(-|t-s| /theta)를 적용한 weight matrix

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & 0 & 0 & \dots & 0 \\ 0 & \mathbf{W}_2 & 0 & \dots & 0 \\ 0 & 0 & \mathbf{W}_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \mathbf{W}_{|\mathcal{I}|} \end{bmatrix}$$

5. 기대 결과

사기여부를 찾기 위해 amt에 의존하던 기존 방식들과는 달리 amt에 크게 의존하지 않아도 사기거래를 찾을 수 있고, 상점의 노드를 정의하지 않아 오버피팅이 일어나지 않고 더 효율적일 수 있다. 계산이 빠르다.

6. 입출력 형식/네트워크

(참고: https://boram-coco.github.io/DL/posts/%EB%B0%9C%ED%91%9C.html)

```
class GCN1(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(1, 32)
        self.conv2 = GCNConv(32,2)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)

X = (data.x[data.train_mask]).numpy()
XX = (data.x[data.test_mask]).numpy()
y = (data.y[data.train_mask]).numpy()
yy = (data.y[data.test_mask]).numpy()

model = GCN1()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
model.train()
for epoch in range(400):
    optimizer.zero_grad()
    out = model(data)
    loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
model.eval()
```