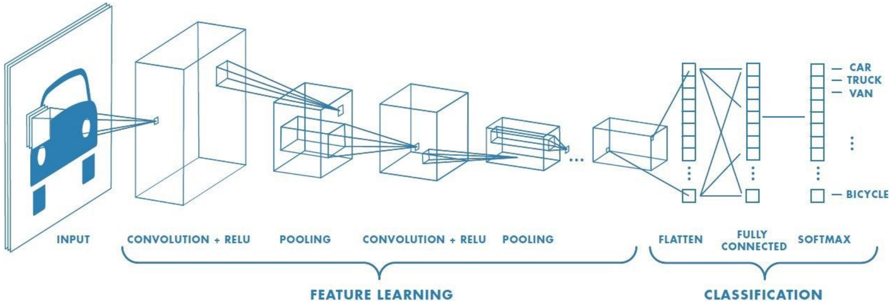
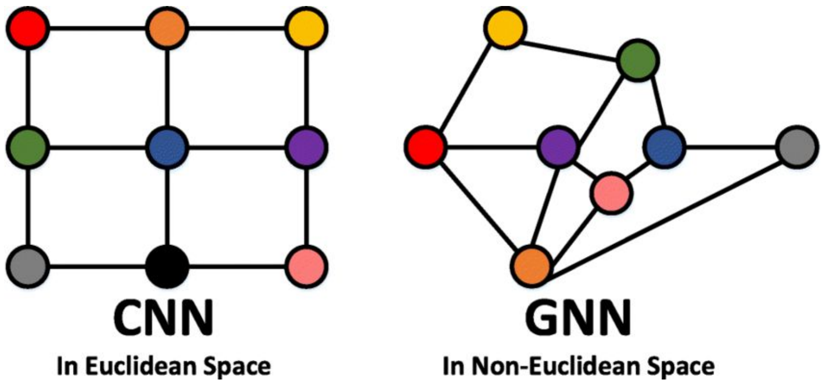
Graph Convolutional Network
Neural Graph Collaborative Filtering

Graph Neural Network



- Applicable to non-euclidean space



※ Convolutional Neural Network

$$x_{ij}^{\ell} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} \, y_{(i+a)(j+b)}^{\ell-1}$$

Lin et al., (2020) A Survey on Deep Learning-Based Vehicular Communication Applications

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

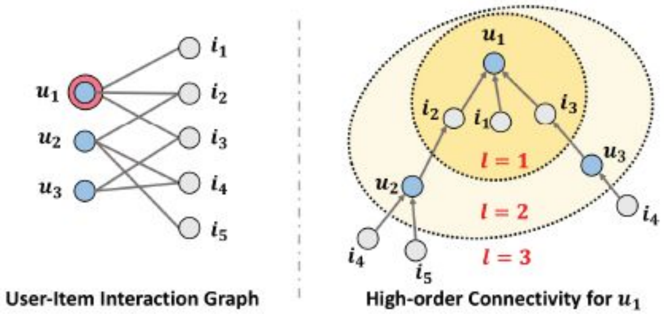
Graph Neural Network

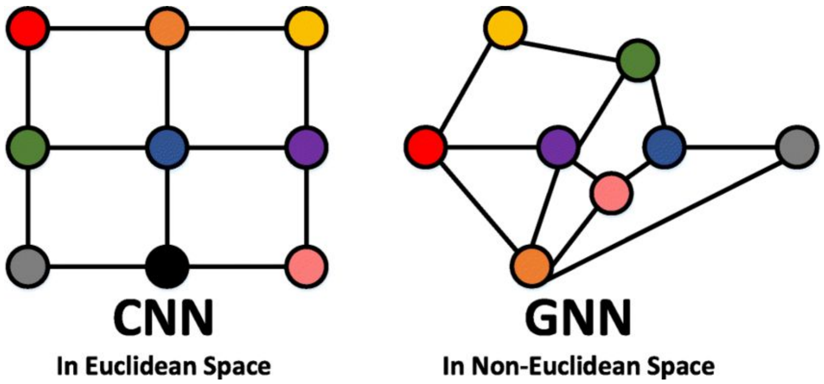
- Applicable to non-euclidean space

※ Graph Convolutional Neural Network

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Work of art:
showing that the network following this equation works well!





Lin et al., (2020) A Survey on Deep Learning-Based Vehicular Communication Applications

Prior Works

대충 설명할 예정 ㅋㅋ

Regularization Term

Complex Eigenvalue Computation

Not Transductive

⇒ GCN!

⇒ GAT! (consider feature nodes as well)

FAST APPROXIMATE CONVOLUTIONS ON GRAPHS

1. Graph Laplacian
 - Normalized: $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$
 - Graph Fourier Transform ***(convolution!!)***
2. Chebyshev Polynomials

   T1(x)=x, T2(x)=x, … ⇒ $g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x$

3. Renormalization Trick

$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x \quad I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}, \quad \tilde{A} = A + I_N \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

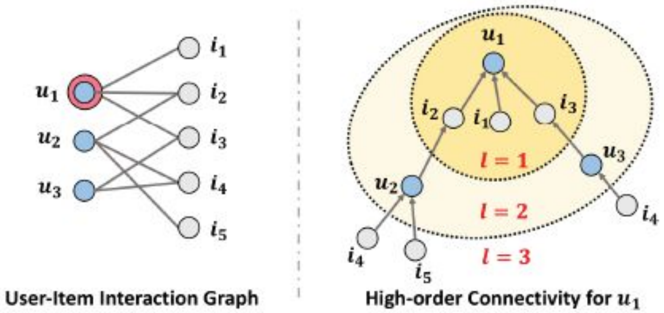
코드 구현

깃허브, 노션 보면서 대충 할 예정 ㅋㅋ

- Pytorch로 구현 언젠가는 할 예정
- Torch-geo에 GCN layer로 predefined 되어있긴 함
- CSR 형태에 대해 이해해야함!
    - sparse matrix의 경우, N x N 형태가 아니라 값이 있는 행, 열 index와 value만 P x 3 형태로 저장한다.

Neural Graph Collaborative FIltering

- GNN에게는 CF signal 정보를 담는 structural benefit 존재
- 그림은 얼핏 보면 트리 구조 같지만 (실제로 트리 형태로 propagation 되기도 하고)
- GCN 형태로 학습시키면 효율적이다.



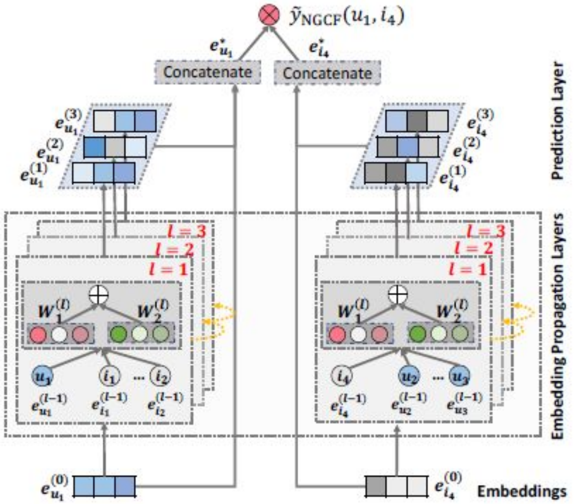
- Architecture
  - Embedding Layer
  - Embedding Propagation Layer
    - Message Construction
    - Message Aggregation
- Training details

Architecture

1. *Message Propagation / Aggregation (+ Layer Orders)*

$$e_u^{(l)} = \text{LeakyReLU}\Big(m_{u\leftarrow u}^{(l)} + \sum_{i\in\mathcal{N}_u} m_{u\leftarrow i}^{(l)}\Big),$$

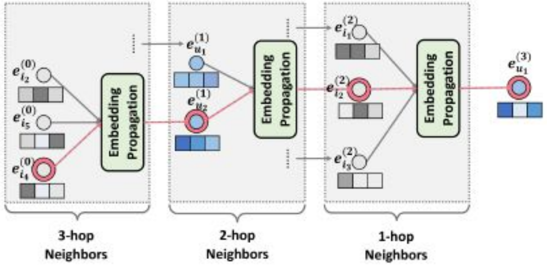
$$\begin{cases} m_{u\leftarrow i}^{(l)} = p_{ui}\big(W_1^{(l)} e_i^{(l-1)} + W_2^{(l)}(e_i^{(l-1)} \odot e_u^{(l-1)})\big) \\ m_{u\leftarrow u}^{(l)} = W_1^{(l)} e_u^{(l-1)}, \end{cases}$$



$$E^{(l)} = \text{LeakyReLU}\Big((\mathcal{L}+I)E^{(l-1)}W_1^{(l)} + \mathcal{L}E^{(l-1)} \odot E^{(l-1)}W_2^{(l)}\Big),$$

2. *Graph Formulation (<u>items/users are linked only to vice versa</u>)*

$$\mathcal{L} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \text{ and } A = \begin{bmatrix} 0 & R \\ R^\top & 0 \end{bmatrix}$$

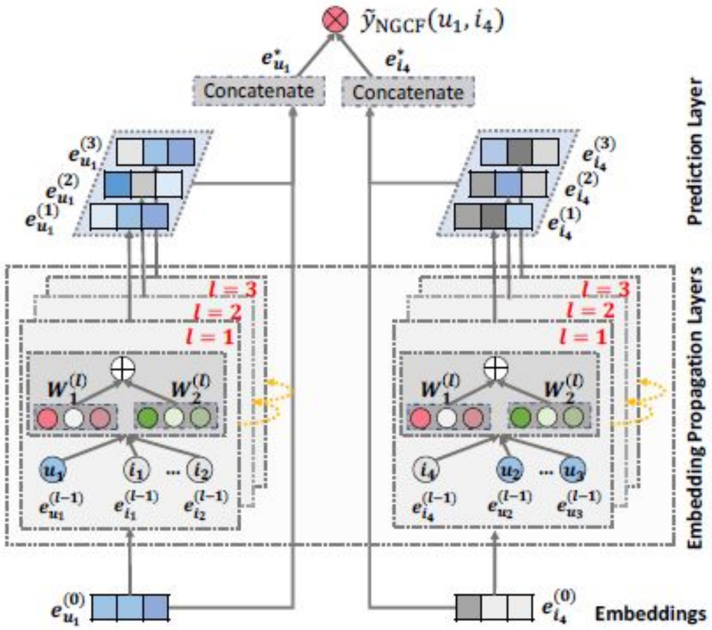


Training Detail

1. Concatenate every order of representation.
2. Inner product

---

3. Pairwise BPR loss
4. Randomly sampled triples



코드 구현

아마 이쯤 되면 조원들 모두 지쳐있을 것이라 예상.

설명 블로그와 원저자 깃허브 슬랙에 올려두었으며, 심화 과제가 따로 있으니 턴을 넘기기로 함.

추신

앞서 언급한 **Graph Attention Network (GAT)** 계열의 **MultiSAGE**가 성능이 조금 더 잘나오는 듯 하다.

다음엔 **GAT**랑 **MultiSAGE** 공부해서 오도록 하겠다.