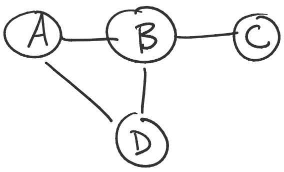
GNNs (continued)

Friday, December 9, 2022 1:23 PM



Given: $\vec{x}_A, \vec{x}_B, \vec{x}_C, \vec{x}_D$

Goal: predict y_A, y_B, y_C, y_D

Model #1: GNN with no neighbor info

$$\vec{h}_{v_j} = \sigma\left(W^T \vec{x}_{v_j} + \vec{b}\right)$$

Model #2: GNN w/ neighbors only

$$\vec{h}_{v_j} = \sigma\left(\sum_{v_i \in N(v_j)} W^T \vec{x}_{v_i} + \vec{b}\right)$$

Model #3: GNN w/ nbrs + self-loops

Assume $\vec{h}_{v_i}^{(0)} = \vec{x}_{v_i}$

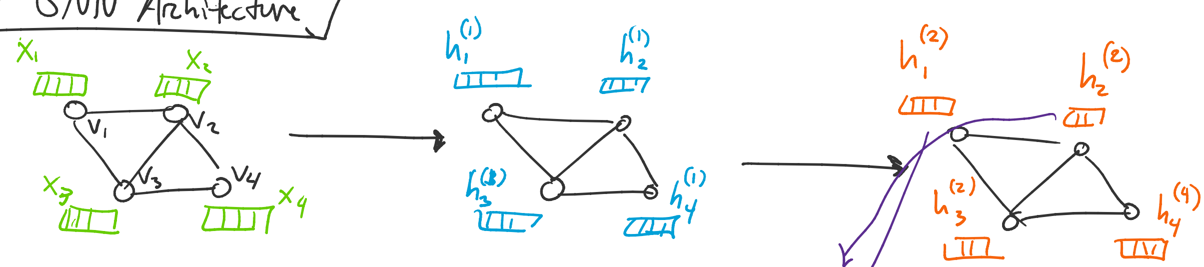
Multiple levels

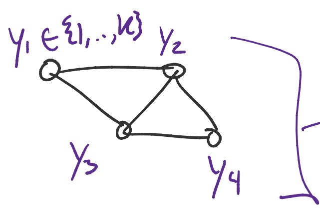
$$\vec{h}_{v_j}^{(k)} = \sigma\left(W^T h_{v_j}^{(k-1)} + \vec{b}\right)$$

$$h_{v_j}^{(k)} = \sigma\left(\underbrace{\sum_{v_i \in N(v_j)} W^T h_{v_i}^{(k-1)}}\, + \vec{b}\right)$$

$$h_{v_j}^{(k)} = \sigma\left(W_{self}^T h_{v_j}^{(k-1)} + \sum_{v_i \in N(v_j)} W_{other}^T \vec{h}_{v_i}^{(k-1)} + \vec{b}\right)$$

Full GNN Architecture





$$loss(G) = loss(y_1, softmax(linear(h_1^{(2)})))$$
$$+ \ldots$$
$$+ \ldots$$
$$+ loss(y_4, softmax(linear(h_4^{(2)})))$$

Matrix Version of Basic GNN

$$h_{v_j}^{(k)} = \sigma\left(W_{self}^T h_{v_j}^{(k-1)} + \sum W_{other}^T \vec{h}_{v_i}^{(k-1)} + \vec{b}\right)$$

D

$$H^{(k)} = \left[\, h^{(k)} \quad \right]$$

$$h_{v_j}^{(k)} = \sigma\left(W_{self}^{T} h_{v_j}^{(k-1)} + \sum_{v_i \in N(v_j)} W_{other}^{T} \vec{h}_{v_i}^{(k-1)} + \vec{b}\right)$$

$$H^{(k)} = \begin{array}{cc} & D \\ N & \begin{bmatrix} - & h_{v_1}^{(k)} & - \\ & \vdots & \\ - & h_{v_N}^{(k)} & - \end{bmatrix} \end{array}$$

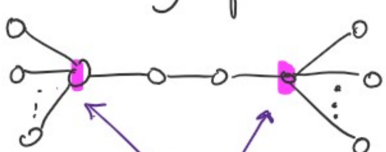
☆ use adjacency matrix, $A \in \{0,1\}^{N \times N}$

$$H^{(k)} = \sigma\left(\underbrace{I}_{N\times N}\,\underbrace{H^{(k-1)}}_{N\times D}\,\underbrace{W_{self}}_{D\times D} + \underbrace{A}_{N\times N}\,\underbrace{H^{(k-1)}}_{N\times D}\,\underbrace{W_{other}}_{D\times D}\right)$$

fold in bias term

Normalization

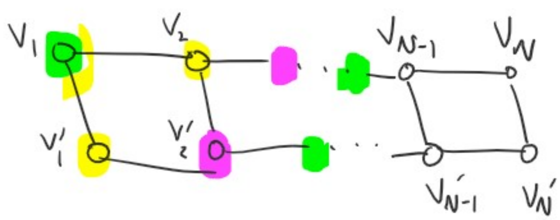
Consider a graph with nodes of varying degree:

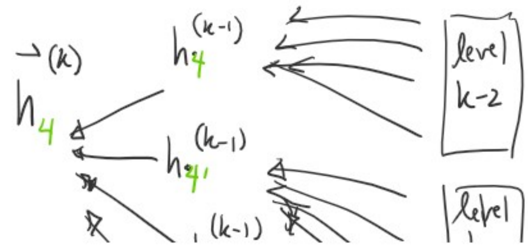


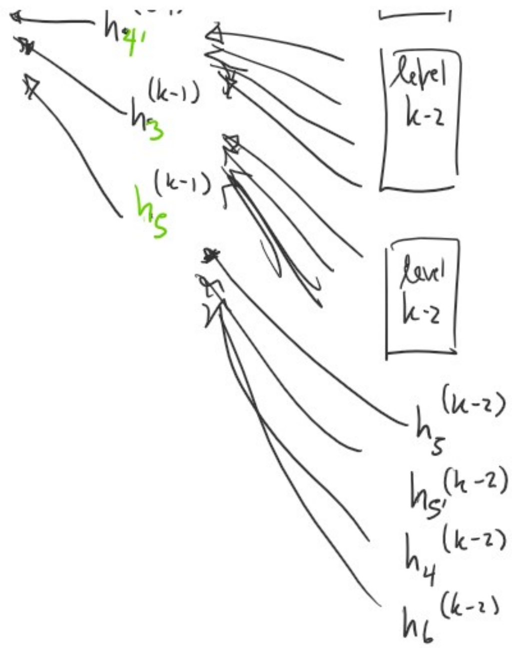
Normalized version

$$h_{v_j}^{(k)} = \sigma\left(W_{self}^{T} h_{v_j}^{(k-1)} + \frac{\sum_{v_i \in N(v_j)} W_{other}^{T} \vec{h}_{v_i}^{(k-1)} + \vec{b}}{|N(v_j)|}\right)$$

K-Hop Neighborhood







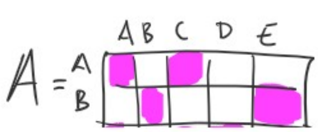
Edge-Level Representations

$$h_{e_{ij}}^{(k)} = \sigma\Big(\underbrace{W_{edge}^{(k)} h_{e_{ij}}^{(k-1)}}_{\text{information about self}} + \underbrace{W_{node}^{(k)} h_{v_i}^{(k-1)} + W_{node} h_{v_j}^{(k-1)}}_{\text{information about adj nodes}} \Big)$$

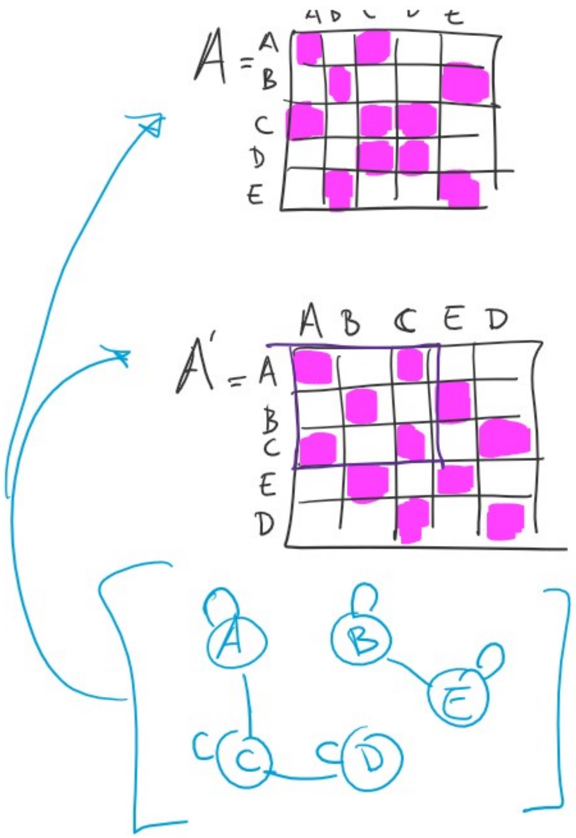
$$h_{v_j}^{(k)} = \sigma\Big(W_{self}^{(k)} h_{v_j} + \sum_{v_i \in N(v_j)} W_{other} h_{v_i}^{(k-1)} + \underbrace{\sum_{\substack{i \text{ s.t.} \\ (i,j)\in E}} W_{edge}'^{(k)} \vec{h}_{e_{ij}}^{(k-1)}}_{\text{information from edges}}$$

GNN Characterization

Strawman Model: Assume all of our graphs have exactly N nodes.



$$\vec{h}_{G} = MLP(A \,\|\, A \,\|\, \cdots \,\|\, A)$$



$$\vec{h}_G = MLP(A_{1,\cdot} \,\|\, A_{2,\cdot} \,\|\, \cdots \,\|\, A_{N,\cdot})$$

concatenation

Many adjacency matrices for the the same graph

Two desireable properties:

1. Permutation Invariance: $\vec{h}_G = f(PAP^T) = f(A)$
2. Permutation Equivariance: $\vec{h}_G = f(PAP^T) = Pf(A)$

where P is a permutation matrix (i.e one 1 in each column and row, and 0 elsewhere)

☆ key takeaway: all GNNs (usually) preserve these properties by aggregation of neighbors