Graph Convolutional Neural Networks

* Graphs are irregular data structure, typically used to represent Hierarchal relationships

* No parent and child relationship exist

{DAG $\Longrightarrow$ Tree}

with loops & self loops and directed edges along with weights they become very general & powerful.

$G = (V, E)$

Set of Vertices $|V|$ $(v_1, v_2, v_3 \ldots)$

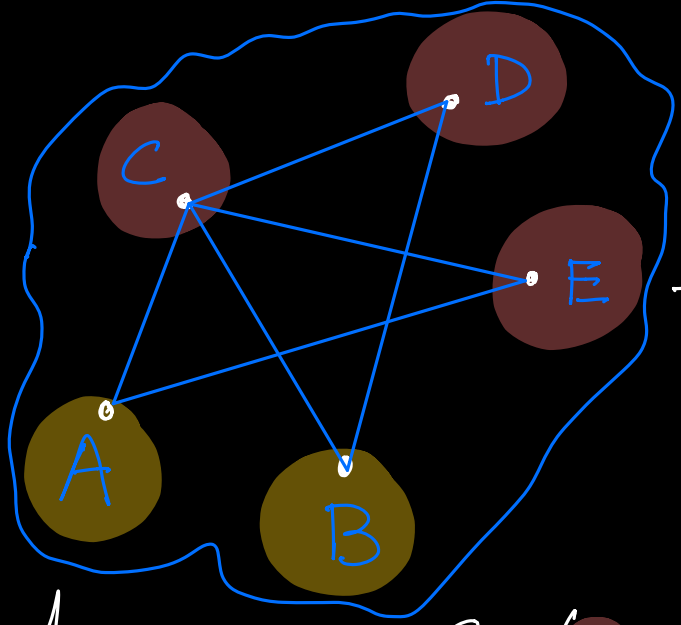
Set of Edges $|E|$ $(e_1, e_2, \ldots)$

Several problems Can be easily embedded into graphs and Can be solved using existing & efficient graph algorithms for traversal, shortest path.

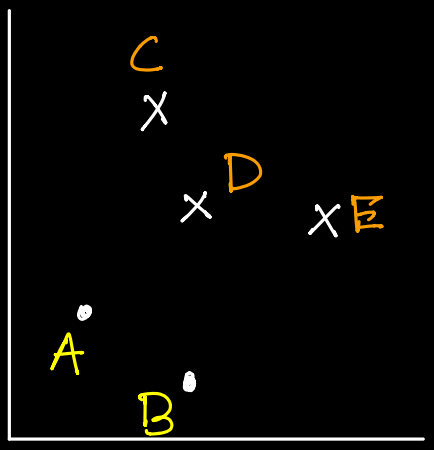
It has been observed that for learning related graph embedded problems Graph CNN can be very powerful learning tool.

Even with few randomly initiated GCN layers Can extract Meaningful node's feature representation.

2-types of GCN's Spectral & Spatial



passed into GCN



The learned node embeddings must preserve the relative nearness.

Any graph $G = (V, E)$

$V = (A, B, C, D, E)$, $|V| = 5$

$E = (AC, AE, BC, BD, CD, CE)$, $|E| = 6$

Adjacency Matrix (A): (N X N)

	A	B	C	D	E
A	0	0	1	0	1
B	0	0	1	1	0
C	1	1	0	1	1
D	0	1	1	0	0
E	1	0	1	0	0

Each node $\{A, B, C, D, E\}$ have a feature f_A (vector) associated to it. Let us assume that the size of input feature is F^0. i.e each node have a F^0 dimensional feature vector viz. f_A, f_B, f_C, f_D, f_E.

		1	2	3	----	F_0
V_1	A	f_A^1	f_A^2	f_A^3	----	$f_A^{F_0}$
V_2	B	f_B^1	f_B^2	f_B^3	----	$f_B^{F_0}$
V_3	C	-	-	-	-	
⋮	D	-	———			
V_N	E	f_E^1	f_E^2	———	----	$f_E^{F_0}$

I/P Feature Matrix (X) $\equiv$ $N \times F^0$

Hence from any given graph $G(V, E)$ we will have 2 matrices extracted as input : ($|V| = N$)

* Adjacency matrix (A) and * Feature matrix (X)

$[N \times N]$ $[N \times F^0]$

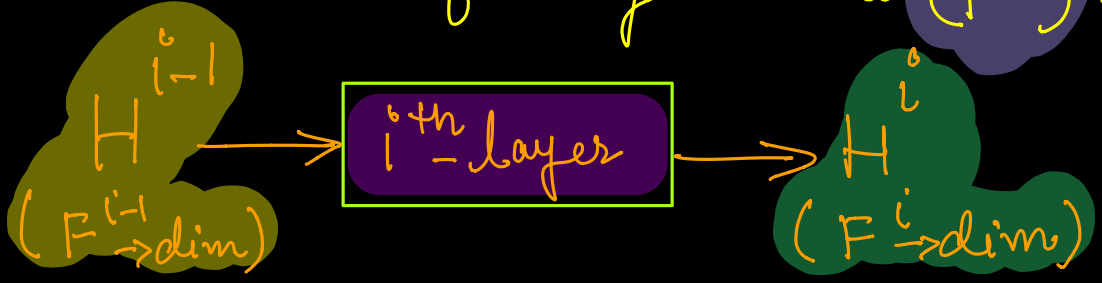
$\Rightarrow$ [Graph I/P (X, A)] fed to GCN layers $\Rightarrow$ H_1 H_2 --- H_i

$H_1 = f(X, A)$ $H_i = f(H^{i-1}, A)$

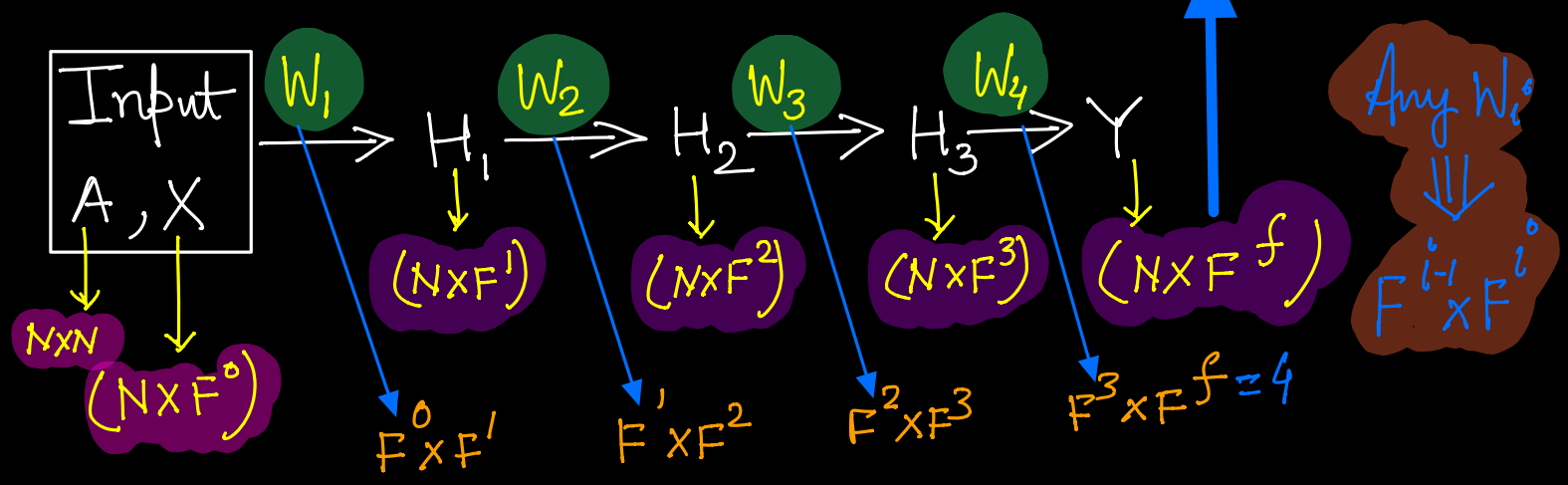
$H_0 = X$

$f \Rightarrow$ propagation function

* We have the O/P of any (i^{th}) hidden layer (H_i), taking a feature of size (F^{i-1}) corresponding to any node and transforming it into (F^i) dimensions.



Hence the feature matrix after [i] hidden layers will be of size $[N \times F^i]$

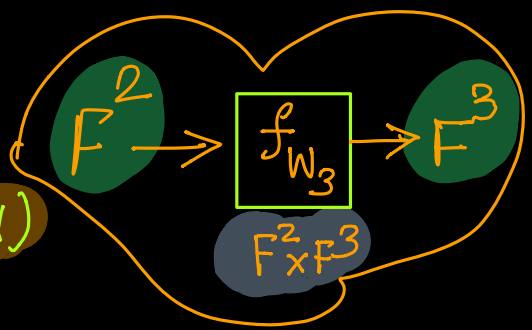


(*) At each layer the node embedding feature vector got aggregated (utilizing its neighbourhood (A)) to form the next layer features using the learned propagation rule (f) parameterized over (W_i)

$$\therefore \quad H_i = f_{W_i}(H_{i-1}, A)$$

$$\Rightarrow \text{Say} \quad H_3 = f_{W_3}(H_2, A)$$

H_3 : $(N \times F^3)$; W_3 : $(F^2 \times F^3)$; H_2 : $(N \times F^2)$; A : $(N \times N)$



The most simplest rule can be :-

$$H_i = f_{W_i}(H_{i-1}, A) = \sigma\left(A * H_{i-1} * W^i\right)$$

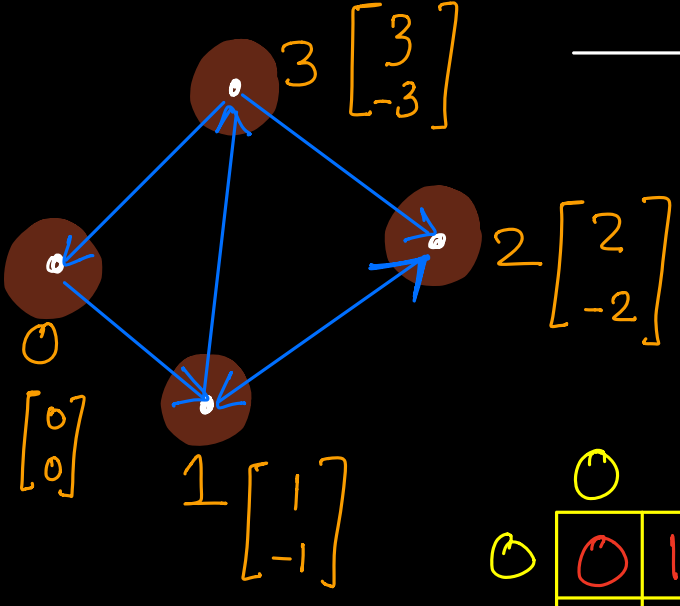
- $(N \times N)$ — Talks about node inter-connection and neighbourhood
- $(N \times F^{i-1})$ $(F^{i-1} \times F^i)$ — $(N \times F^i)$ — Filtering as done in any fully connected network.

For simplicity let us assume
σ = linear function

$$\therefore H_1 = A H_0 W_0 = A X W_0$$

$\hookrightarrow (W = \mathbb{1}$ initialization$)$

$$= AX$$



→ Considering such a simple graph with random but easy to use node inital embeadings to understand what is happing.

$A =$

	0	1	2	3
0	0	1	0	0
1	0	0	1	1
2	0	1	0	0
3	1	0	1	0

$X =$

0	0	0
1	1	-1
2	2	-2
3	3	-3

$$\therefore H_1 = f(X, A) = AX = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 2 & -2 \\ 3 & -3 \end{bmatrix}$$

Any (i^{th}) node feature becomes the addition of features of its neighbours.

$0 = [0, 0] \Rightarrow [1, -1]$
$1 = [1, -1] \Rightarrow [2, -2] + [3, -3] = [5, -5]$
$2 = [2, -2] \Rightarrow [1, -1]$
$3 = [3, -3] \Rightarrow [0, 0] + [2, -2]$

$$\begin{bmatrix} 1 & -1 \\ 5 & -5 \\ 1 & -1 \\ 2 & -2 \end{bmatrix}$$

Neighbourhood feature aggregation just as it happens in Convolutional neural network

Problem-01

$\Rightarrow$ But since there are no self loops their own features are ignored. In matrix [A] diagonals are "0".

Solution

$\Rightarrow$ Just need to add self loop. $[A] \Rightarrow [\hat{A}]$

$\Rightarrow \hat{A} = A + I$ (Identity matrix)

$$\hat{A} \cdot X = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 2 & -2 \\ 3 & -3 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 6 & -6 \\ 3 & -3 \\ 5 & -5 \end{bmatrix}$$

nodes own features are not ignored.

Problem-02

↳ for any node (i^{th}), we will be considering the (i^{th}) row of the adjacency matrix (A). Basically the feature of node (i) will be the aggregation of the neighbourhood.

$$\forall_{j=1 \text{ to } N} \; A_{ij}$$

If $A_{ij}=1$ then it belongs to the neighbourhood of (i^{th}) node

$$\text{Degree}(i) = \sum_{j=1 \text{ to } N} A_{ij}$$

* If degree is very high too many things will be added
* If degree is low then very less things will be added

This may lead to vanish/explode the absolute value of the feature

This may result in backpropagation issues

As optimizers are quite sensitive to i/p scales.

Hence the finally aggregated node feature need to be suitably normalized.

Solution ②

The node features are normalized by the degree of the node.

(Aggregated feature per unit of degree)

$$D = \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & d_3 & & \\ & & & \ddots & \\ & & & & d_N \end{bmatrix}$$

$$D^{-1} = \begin{bmatrix} 1/d_1 & & & & \\ & 1/d_2 & & & \\ & & 1/d_3 & & \\ & & & \ddots & \\ & & & & 1/d_N \end{bmatrix}$$

Hence the final update rule can be

$$f(X, A) = D^{-1} A X \quad (a)$$

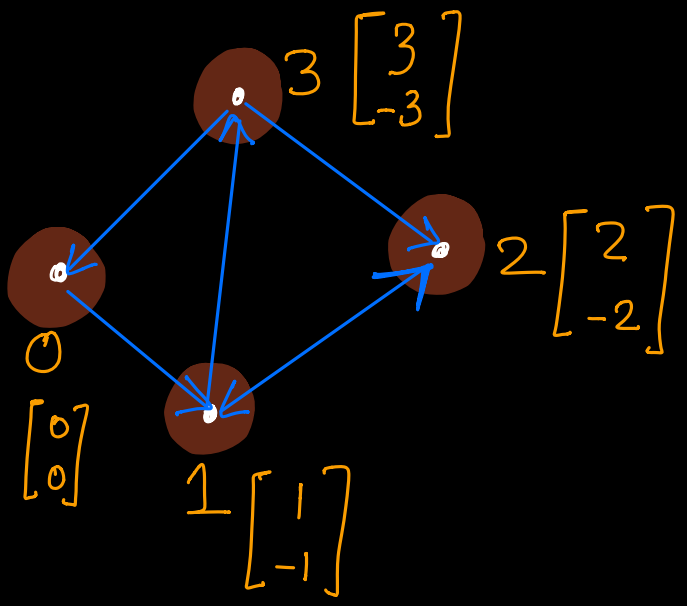
$$D^{-1} \hat{A} X \quad (b)$$

$$\left(\hat{A} = A + I\right)$$

(Solving both issues)

Therefore for our case

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$



$$X = \begin{bmatrix} 0 & 0 \\ 1 & -1 \\ 2 & -2 \\ 3 & -3 \end{bmatrix}$$

$$AX = \begin{bmatrix} 1 & -1 \\ 5 & -5 \\ 1 & -1 \\ 2 & -2 \end{bmatrix}$$

$$\hat{A} = A + I = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\hat{A}X = \begin{bmatrix} 1 & -1 \\ 6 & -6 \\ 3 & -3 \\ 5 & -5 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad D^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$$

(a) $D^{-1}AX$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.5 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 5 & -5 \\ 1 & -1 \\ 2 & -2 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 2.5 & -2.5 \\ 1 & -1 \\ 1 & -1 \end{pmatrix}$$

(b) $D^{-1}\hat{A}X$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.5 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 6 & -6 \\ 3 & -3 \\ 5 & -5 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 3 & -3 \\ 3 & -3 \\ 2.5 & -2.5 \end{pmatrix}$$

This aggregation utilizes nodes features as well as aggregated.

⊛ Now learning the transformation parameterized by (W)

$$f_W = (\hat{D}^{-1}\hat{A}X) * W$$

(a) $f_{W_1} = (\hat{D}^{-1}\hat{A}X) * W_1$ $\quad$ (b) $f_{W_2} = (\hat{D}^{-1}\hat{A}X) * W_2$

$W_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ $\quad$ $W_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$$\begin{bmatrix} 1 & -1 \\ 3 & -3 \\ 3 & -3 \\ 2.5 & -2.5 \end{bmatrix}_{4\times2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}_{2\times2} = \begin{bmatrix} 2 & -2 \\ 6 & -6 \\ 6 & -6 \\ 5 & -5 \end{bmatrix}_{(4\times2)}$$

In order to reduce the feature vector's dimensionality

$$\begin{bmatrix} 1 & -1 \\ 3 & -3 \\ 3 & -3 \\ 2.5 & -2.5 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 6 \\ 5 \end{bmatrix}_{(4\times1)}$$

⊛ After aggregation & transformations, nonlinearity has been introduced using Relu Activation (σ)

$$\text{Relu}(f_W(X, A))$$

(a) $\begin{bmatrix} 2 & 0 \\ 6 & 0 \\ 6 & 0 \\ 5 & 0 \end{bmatrix}$ $\quad$ (b) $\begin{bmatrix} 2 \\ 6 \\ 6 \\ 5 \end{bmatrix}$

These are the complete final representations.

$$H = \sigma(\hat{D}^{-1}AX * W)$$

Previous layer o/p.

∴ for Aggregation (A, X, & D) are required

Transformation (W & σ) are required

Hence one can see that any function (f_W) learned at any layer (l), can be represented as

$$f_W(H^l, A) = \text{Transform}(\text{Aggregate}(A, H^l), W)$$

$$\text{Transform}(M, W) = O(M * W)$$

Rule (1) (SUM)

$$\text{Aggregate}(A, H^l) = AH$$

Rule (2) (MEAN)

$$\text{Aggregate}(A, H) = \hat{D}^{-1}\hat{A}H$$

$$(\hat{A} = A + I)$$

Basically both aggregation rule uses weighted sum of its neighbouring features. Just weighing is different.

(A) SUM Rule (Neighbourhood Aggregation)

$$\underset{A_1}{\underset{A_2}{\underset{A_3}{\underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}}_{A}}}} \overset{3\times3}{} * \underbrace{\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}}_{3\times3} = \underbrace{\begin{bmatrix} 4a+5d+6g \\ 4b+5e+6h \\ 4c+5f+6i \end{bmatrix}}_{3\times1} \overset{(1\times3)^T}{} = 4\underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{(3\times1)} + 5\begin{bmatrix} d \\ e \\ f \end{bmatrix} + 6\begin{bmatrix} g \\ h \\ i \end{bmatrix}$$

$$\underbrace{\quad}_{(1\times3)}{}^T$$

Let us working on (i^{th}) node, hence using (i^{th}) row of adjacency matrix (A_i).

$$agg(A, X)_i = A_i X = \sum_{j=1}^{N} A_{ij} * X_j$$

A_{ij}: only included if its a neighbour

X_j: feature of (j^{th}) node

Hence Contribution of each Node depends solely on neighbourhood defined by adjacency matrix (A).

(B) Mean Rule

$$agg(A, X)_i = D^{-1} A_i X$$

$$= \sum_{K=1}^{N} D^{-1}_{i,K} * \sum_{j=1}^{N} A_{ij} * X_j$$

(i) is fixed & (K) varies from (1) to (N)

Just same as previous.

$$= \sum_{j=1}^{N} D^{-1}_{i,i} A_{i,j} X_j = \sum_{j=1}^{N} \frac{A_{ij}}{D_{ii}} * X_j$$

$\frac{A_{ij}}{D_{ii}}$: Contributions got weighted by degree of the (i^{th}) node

X_j: feature of j^{th} node

$$\sum_{j=1}^{N} \frac{A_{ij}}{D_{ii}} = \frac{\sum_{j=1}^{N} A_{ij}}{D_{ii}} = (1)$$

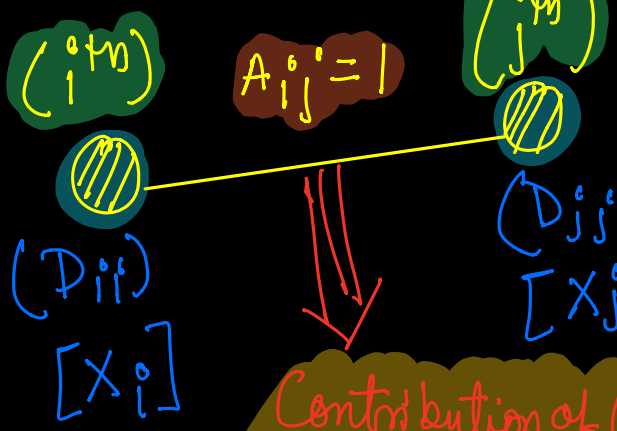
Weights are guaranted to be Summed equal to (1).
for any node (i^{th}) [Weighted Normalization]

This can be seen as the mean feature representation of neighbours of (i^{th}) node.

(C) Spectral Rule

$$agg(A, X)_i = D^{-\frac{1}{2}} A_i D^{-\frac{1}{2}} \quad X = \sum_{j=1}^{N} \frac{A_{ij}}{D_{ii} D_{jj}} X_j$$

Degree of (i^{th}) & (j^{th}) node



Contribution of (X_j) in the aggregation of (j^{th}) node can be

$$\frac{A_{ij}}{D_{ii}} * X_j$$

$$\frac{A_{ij}}{D_{ii} * D_{jj}} * X_j$$

(*) If both are highly connected
Less Contributions ↓

(*) If both are Sparsely Connected
High Contribution. ↑