IDC410
A course on Image Processing and Machine Learning
(Lecture 23)

Shashikant Dugad,

IISER Mohali

Graphical Neural Network

GNN Lectures adapted from following References

1. Youtube Lectures given by Petar Velickovic on YouTube:

   https://www.youtube.com/watch?v=uF53xsT7mjc&t=1350s

   https://www.youtube.com/watch?v=8owQBFAHw7E&list=PPSV

   https://www.youtube.com/watch?v=uF53xsT7mjc&list=PPSV&t=728s

2. Other Youtube Videos

https://www.youtube.com/watch?v=fOctJB4kVIM&list=PPSV

https://www.youtube.com/watch?v=ABCGCf8cJOE&list=PPSV

https://www.youtube.com/watch?v=0YLZXjMHA-8&list=PPSV

https://www.youtube.com/watch?v=2KRAOZIULzw&list=PPSV

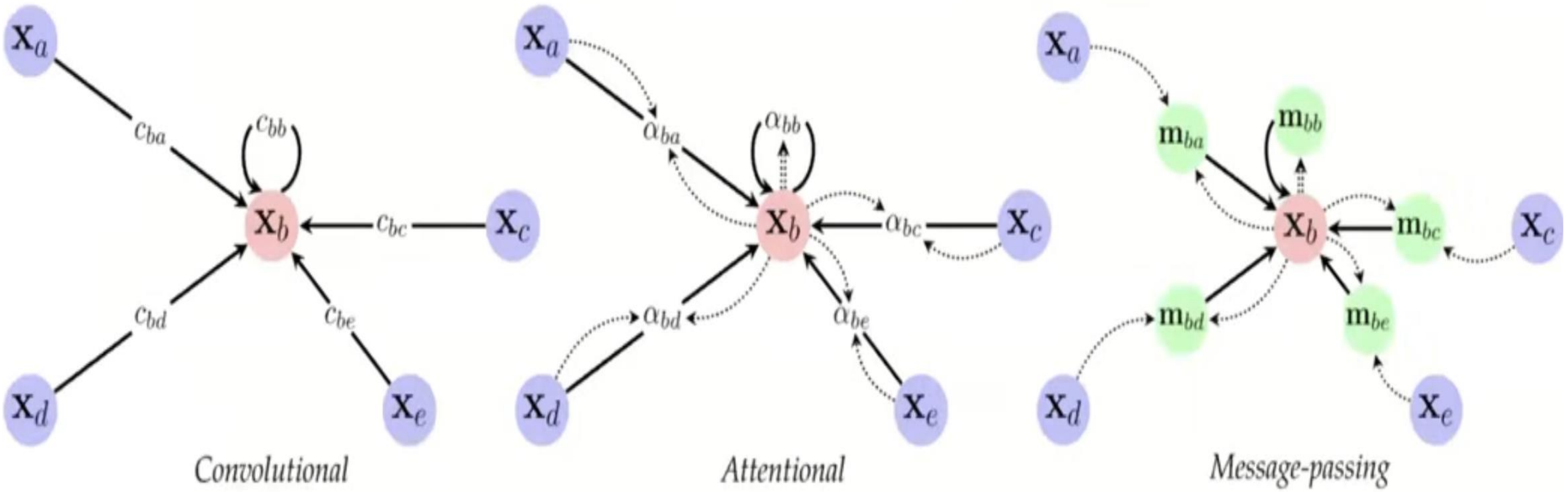
https://www.youtube.com/watch?v=wJQQFUcHO5U&list=PPSV

3. Notes:

https://distill.pub/2021/gnn-intro/

https://distill.pub/2021/understanding-gnns/

Embedding Algorithms

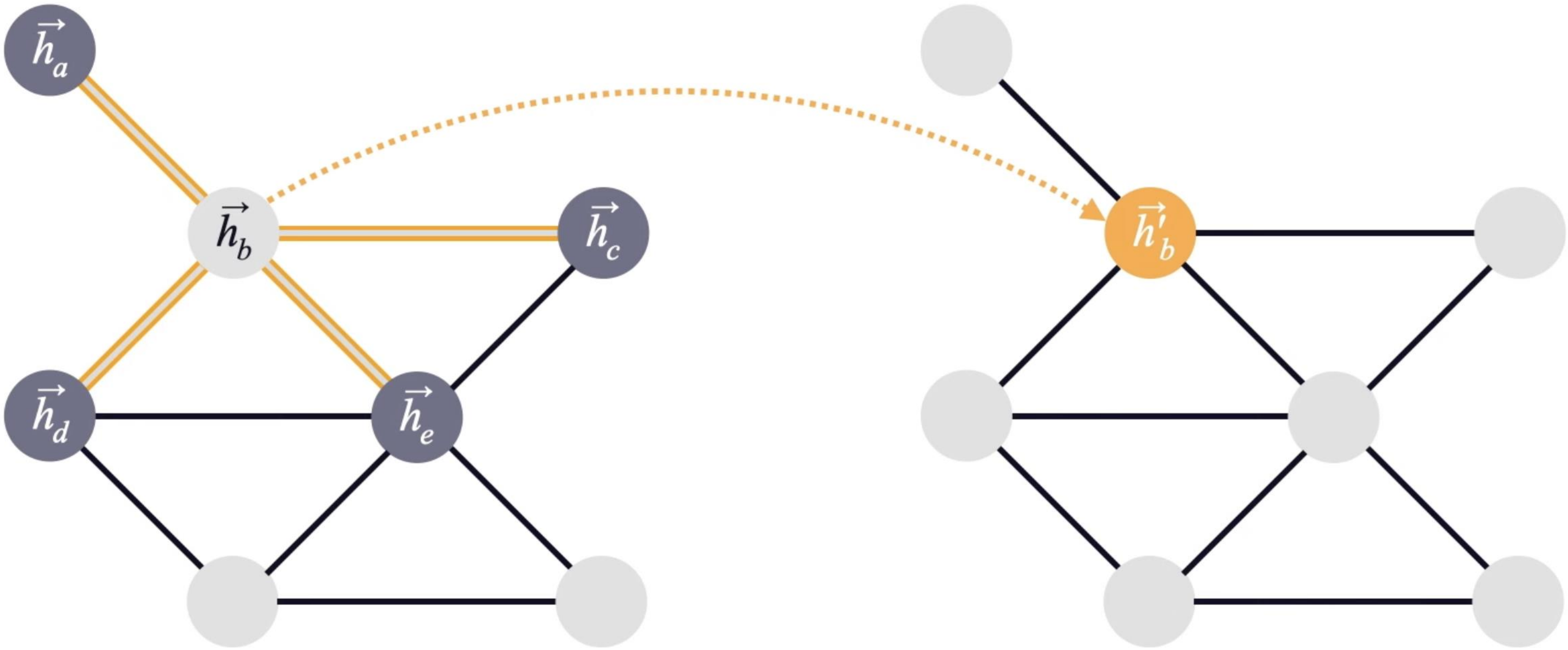


$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} a(\mathbf{x}_i,\mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} \psi(\mathbf{x}_i,\mathbf{x}_j)\right)$$

Convolutional GNN → GCN



$\vec{h}'_i = g(\vec{h}_a, \vec{h}_b, \vec{h}_c, \ldots)$ $\quad (a, b, c, \ldots \in N_i)$

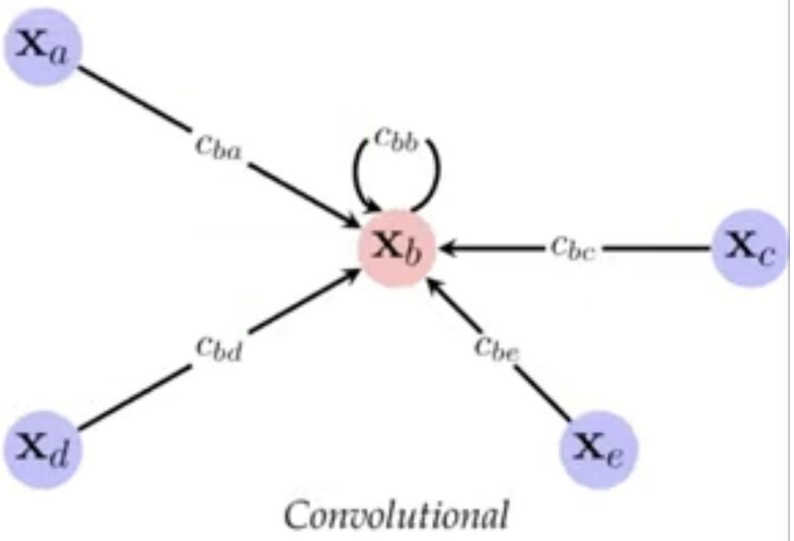
Convolutional GNN → GCN

- Features of neighbours aggregated with fixed weights, c_{ij}

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

- Usually, the weights depend directly on **A**.
  - ChebyNet (Defferrard *et al.*, NeurIPS'16)
  - GCN (Kipf & Welling, ICLR'17)
  - SGC (Wu *et al.*, ICML'19)

- Useful for **homophilous** graphs and **scaling up**
  - When edges encode *label similarity*



Homophily is a graph property describing the tendency of edges to connect similar nodes; the opposite is called heterophily.

GCN: Polynomial Filters on Graphs

- For a given Graph ***G***, Adjacency matrix ***A*** (matrix elements either ***0*** or ***1***), the degree of a ***node v*** can be obtained as:

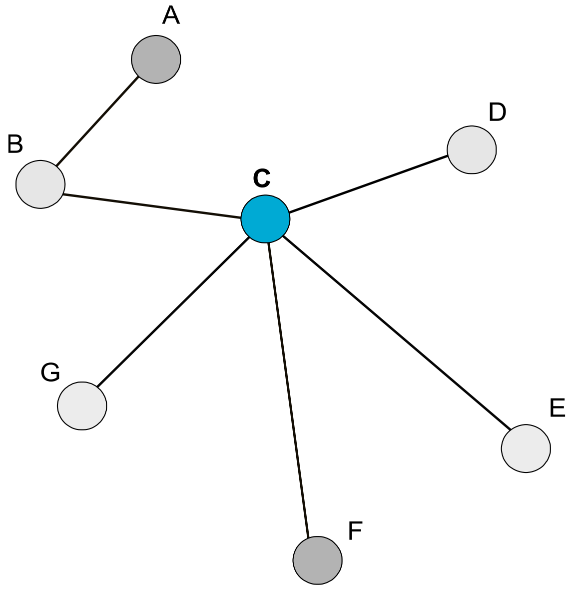
$$D_v = \sum_u A_{uv}$$

- where, A_{uv} denotes the entry in the row corresponding ***v*** to and the column corresponding to ***u*** in the adjacency matrix

- Then, the ***Laplacian L*** of graph ***G*** with ***n*** nodes is given by a ***n×n*** matrix as:
  - ***L= D− A,***
  - The name, ***Laplacian of graph*** if from being the discrete analog of the Laplacian operator from calculus.

- Note: ***D*** and ***A*** are matrices of size ***n x n***

GCN: Polynomial Filters on Graphs



Undirected Input Graph G

$$\begin{array}{c|c} & \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} & \begin{bmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 5 & -1 & -1 & -1 & -1 \\ & & -1 & 1 & & & \\ & & -1 & & 1 & & \\ & & -1 & & & 1 & \\ & & -1 & & & & 1 \end{bmatrix} \end{array}$$

Laplacian of Input Graph G

GCN: Polynomial Filters on Graphs

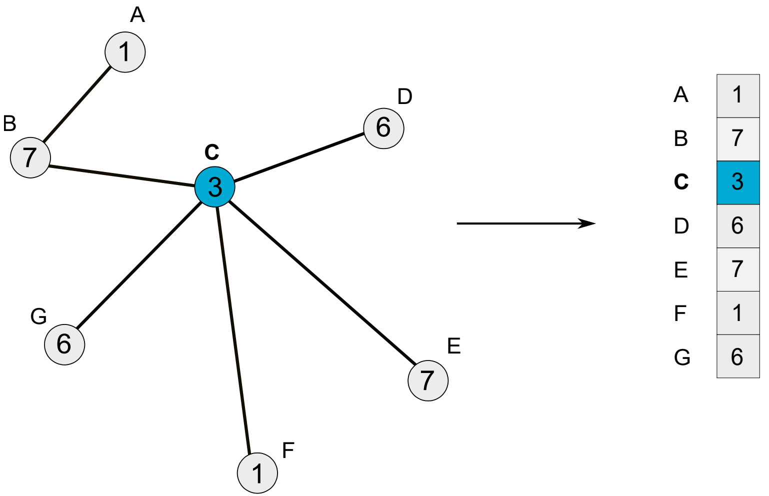
- We can build polynomials of order *d* of graph Laplacian the form:

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + w_3 L^3 + \dots + w_d L^d = \sum_{i=0}^{d} w_i L^i$$

- Polynomial of this form can be represented by its vector of coefficients:
  - $w \rightarrow [w_0, w_1, w_2, \dots, w_d]$
  - Note that for every vector w; $p_w(L)$ is an $n \times n$ matrix, just like L
- These polynomials can be thought of as the equivalent of *filters* in *CNNs*, and the coefficients *w* as the weights of the *filters*
- Having constructed the feature vector *x*, we can define its convolution with a polynomial filter p_w as: $x' = p_w(L)\, x$

GCN: Polynomial Filters on Graphs

- Le us consider the case where, nodes have one-dimensional features: each of the x_v for $v \in V$ is just a single real number. The same ideas hold when each of the x_v are higher-dimensional vectors, as well
- Using the previously chosen ordering of the nodes, we can stack all of the node features x_v to get a vector $x \in R_n$



GCN: Polynomial Filters on Graphs

- To understand how the coefficients ***w*** affect the convolution, let us begin with the ***simplest*** polynomial: when $w_0=1$ and all of the other coefficients are ***0***. In this case, ***x'*** is just ***x***

$$x'_v = p_w(L)x_v = \sum_{i=0}^{d} w_i L^i x_v = w_0 I_n x_v = x_v$$

- Consider a polynomial of ***order 1*** with $w_1=1$ and all other coefficients ***0***:
  - Note: Features at each node ***v*** are combined with the features of its immediate neighbours ***u ∈ N(v)***

$$x'_v = p_w(L)x_v = \sum_{i=0}^{d} w_i L^i x_v = w_1 L^1 x_v = (Lx)_v = L_v x$$

$$= \sum_{u \in G} (D_{vu} - A_{vu})\, x_u = D_v x_v - \sum_{u \in N(v)} x_u$$

GCN: Polynomial Filters on Graphs

- **Hops:** Refers to the ***minimum # of edges*** a node is away from another node in a graph. A ***1-hop*** neighbour is directly connected to a node, a ***2-hop*** neighbour is connected to a ***immediate*** neighbour of the node, and so on. GNNs often ***aggregates or convolutes*** information from nodes within a certain ***# of hops***, shaping the node's representation based on its local neighbourhood.

- ***# of hops*** used in polynomial filter in graph convolution is equivalent to the ***size of convolutional filter in CNN***

- How does the degree ***d*** of the polynomial influence the behaviour of the convolution?

- If ***# of hops*** between two nodes is ***k***, then it can be shown that,

$$L_{uv}^{i} = 0 \quad for\ all \quad i > k$$

GCN: Polynomial Filters on Graphs

- This implies, when we convolve ***x*** with ***p_w(L)*** of degree ***d*** then to get ***x'***:

$$x'_v = (p_w(L)x)_v = (p_w(L))_v x = \sum_{i=0}^{d} w_i L^i x = \sum_{i=0}^{d} w_i \sum_{u \in G} L^i_{uv} x_u = \sum_{i=0}^{d} w_i \sum_{\substack{u \in G, \\ distG(u,v) \le i}} L^i_{uv} x_u$$

- Effectively, the convolution at node ***v*** occurs only with nodes ***u*** which are not more than ***i hops*** away. Thus, these polynomial filters are localized. The degree of the localization is governed completely by ***d***

GCN: ChbNetPolynomial Filters on Graphs

- **Chebnet uses the polynomial of the forma:**

$$p_w(L) = \sum_{i=0}^{d} w_i T_i(\bar{L})$$

- **where, T_i is the degree-*i* Chebyshev polynomial of the first kind ($T_i(\cos\theta) = \cos(i\theta)$) and $\bar{L}$ is the *normalized* Laplacian defined using the largest eigenvalue of *L* as:**

$$\bar{L} = \frac{2L}{\lambda_{max}(L)} - I_n$$

- **$\bar{L}$ always have eigenvalues between [-1,1] whereas the eigenvalues for un-normalised L can be >1 may cause divergences at higher order of L**
- **Use of Chebyshev polynomials has certain properties that makes interpolation more numerically stable.**

GCN: Embedding Computation

- GNN is built by stacking ChebNet (or any polynomial filter) layers one after the other with non-linearities, much like a standard CNN.

- $h^{(0)}=x$ is an original feature

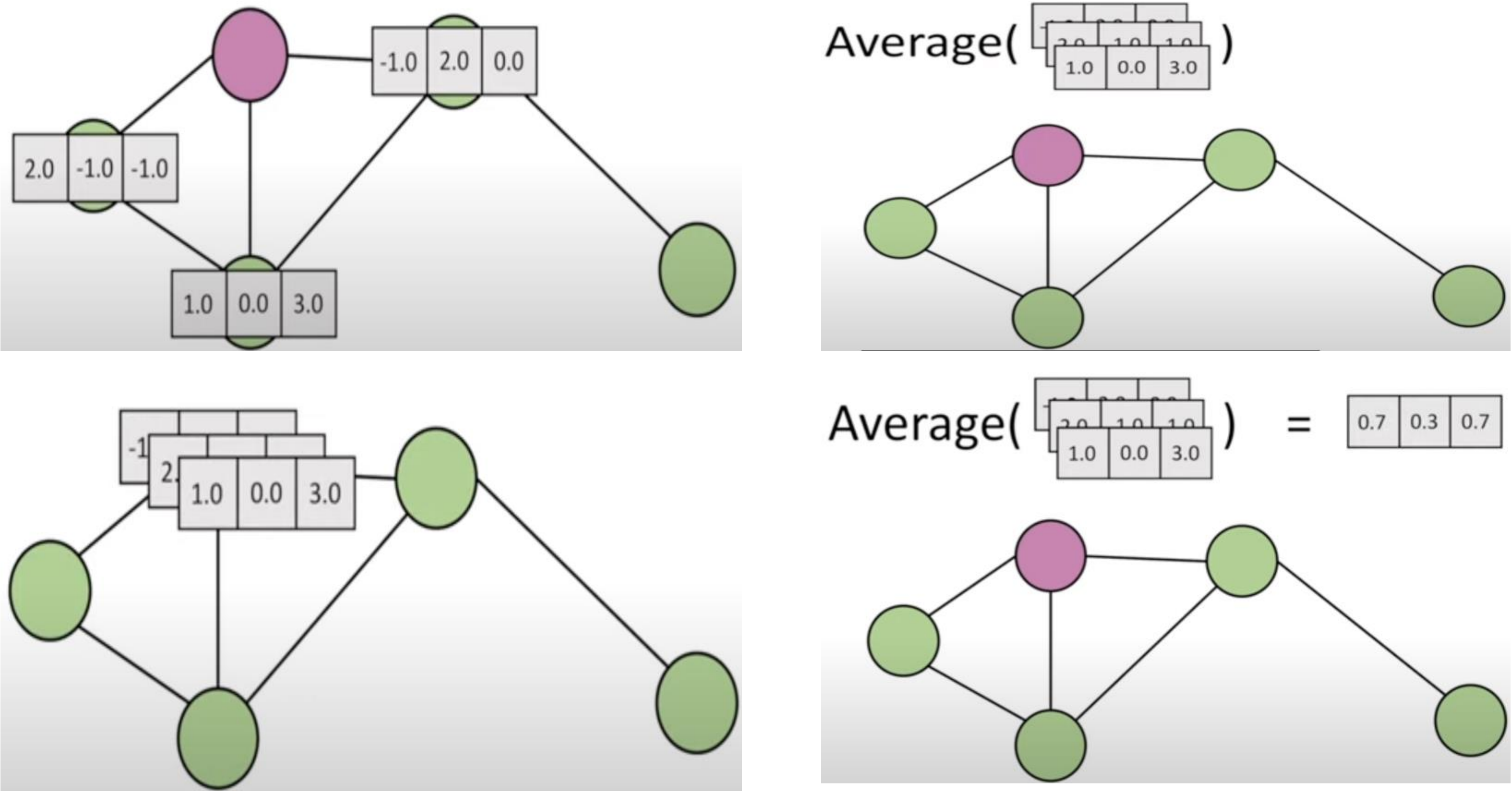
- Then iterate, for $k=1, 2, 3, ..$ *up to* K: Compute the *matrix* $p^{(k)}$ as the polynomial filter defined by the filter weights $w^{(k)}$ evaluated at L: $p^{(k)} = p_{w^{(k)}}(L)$

- Multiply $p^{(k)}$ with $h^{(k-1)}$, a standard matrix-vector multiplication operation: $g^{(k)} = p^{(k)} \times h^{(k-1)}$

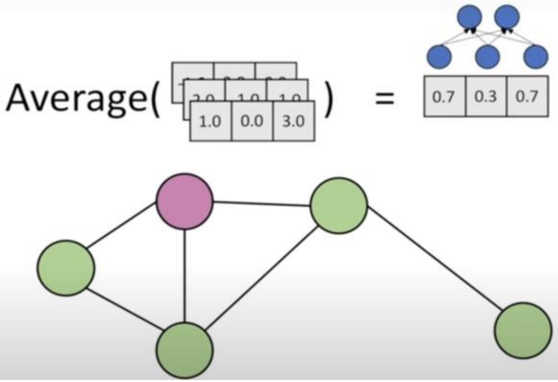
- Now apply an activation function σ and introduce non-linearity to get the next level learned features $h^{(k)}$: $h^{(k)} = \sigma(g^{(k)})$

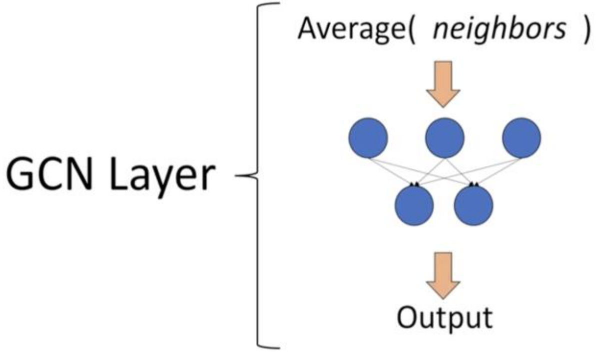
- Note that these networks uses the same filter weights across different nodes, exactly mimicking weight-sharing in CNNs

GNN Example: Predictions with Message Passing

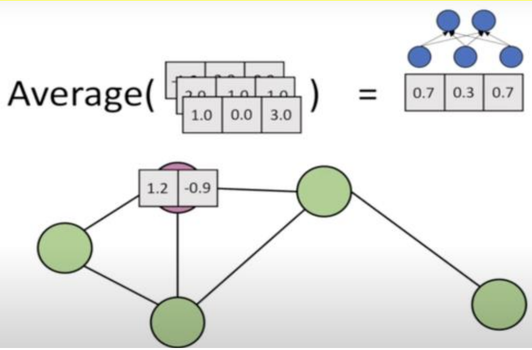


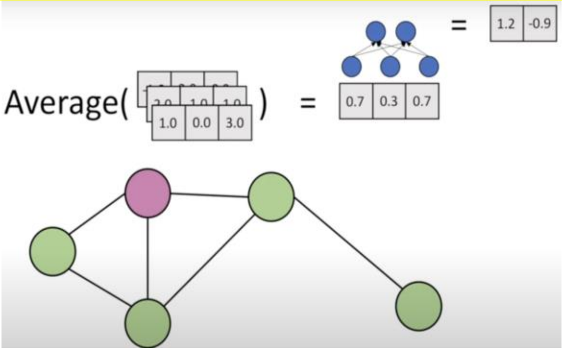
GNN Example: Predictions with Message Passing



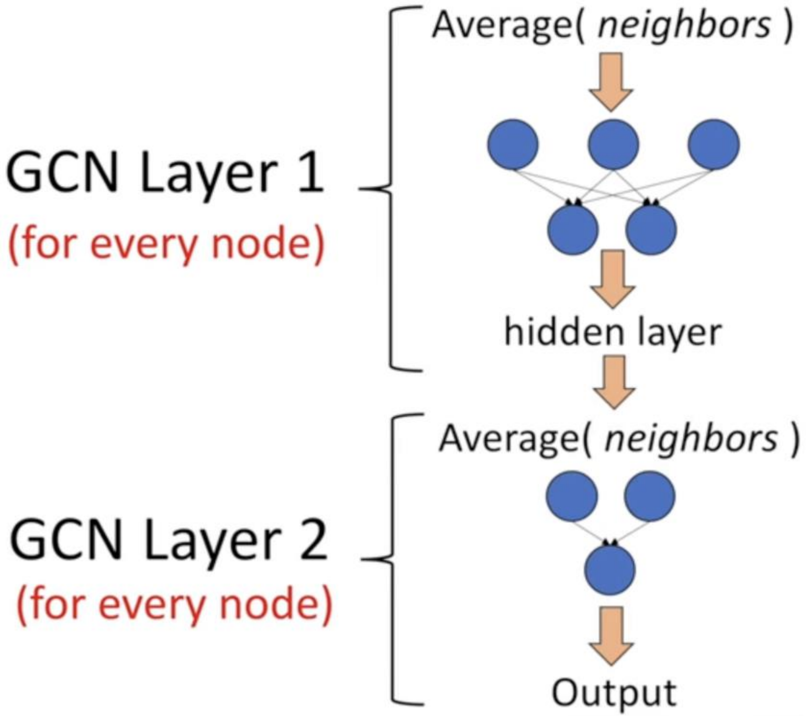


The dimensionality of each attribute is revised with the update function. The update function is generally a 1-layer MLP with a ReLU activation function and a layer norm for normalization of activations

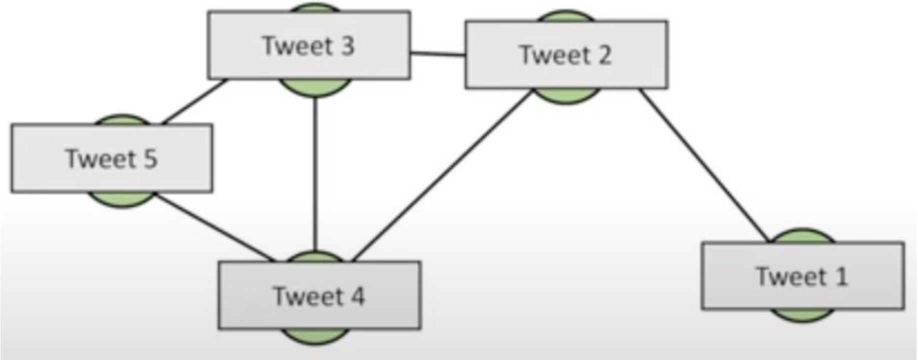


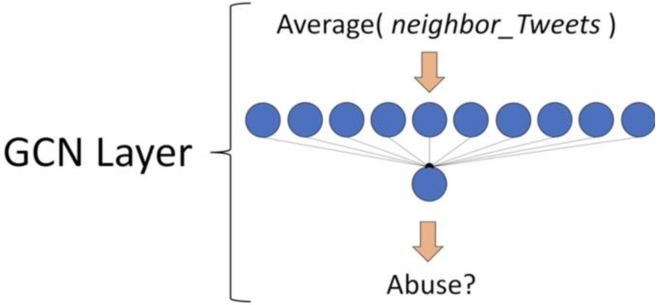


GNN Example: Predictions with Message Passing

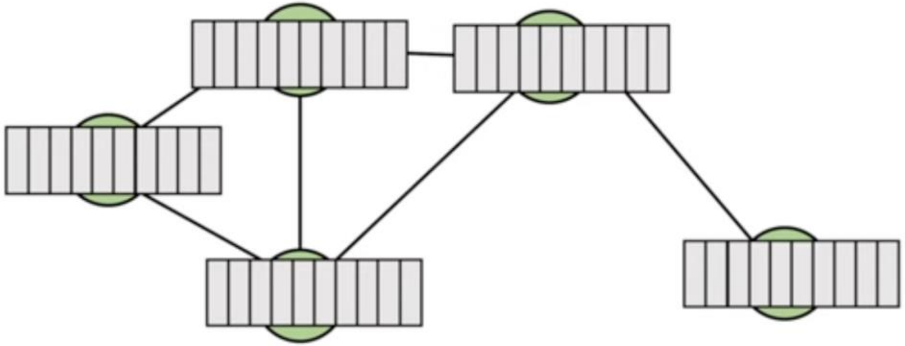


GNN Example: Predictions with Message Passing





$$h_i^{l+1} = \sigma\left(\sum_{j\in N_i} \frac{1}{c_{ij}} h_j^l W^l\right)$$



$$h_i^{l+1} = \sigma\left(W_0^l h_i^l + \sum_{r\in R}\sum_{j\in N_i^r} \frac{1}{c_{ir}} W_r^l h_j^l\right)$$

Super Adjacency Matrix

$$W_r^l = \bigoplus_{b=1}^{B} Q_{br}^l$$