Graph Neural Network- Part 2

- Definition of graph
- Adjacency matrix , Laplacian and normalized Laplacian
- Convolution on graphs using eigenvectors of Laplacian
- Graph Fourier transform and inverse.

- Vanilla Spectral GCN,
- Chebyshev polynomial and ChebNet

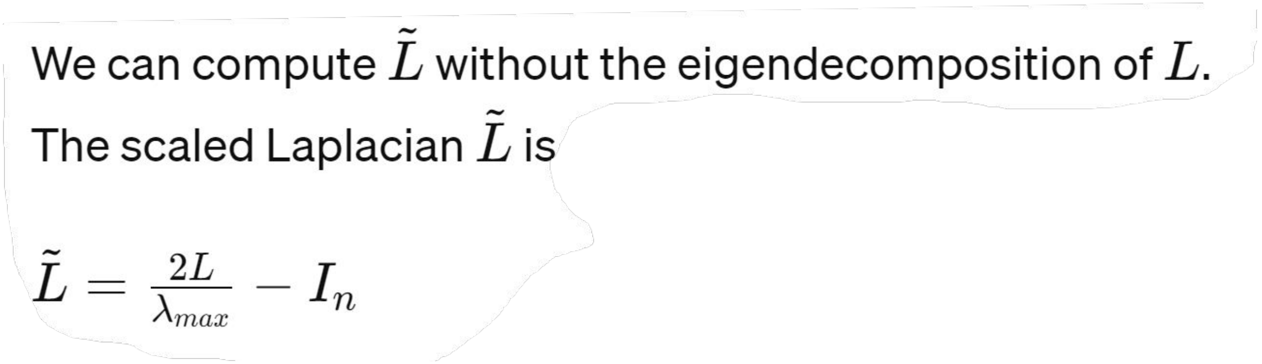
ChebNet Graph Convolution

- $x * g = U g_\theta U^T x$
- $x * g = \sum_i \theta_i U T_i(\tilde{\Lambda}) U^T x$

$$x * g_\theta = U(\sum_i \theta_i T_i(\tilde{\Lambda}))U^T x$$

- It is equivalent to:

$$x * g_\theta = \sum_{i=1}^{k} \theta_i T_i(\tilde{L}) x$$



We can compute $\tilde{L}$ without the eigendecomposition of L.

The scaled Laplacian $\tilde{L}$ is

$$\tilde{L} = \frac{2L}{\lambda_{max}} - I_n$$

Eigenvalues: [0,1,2]

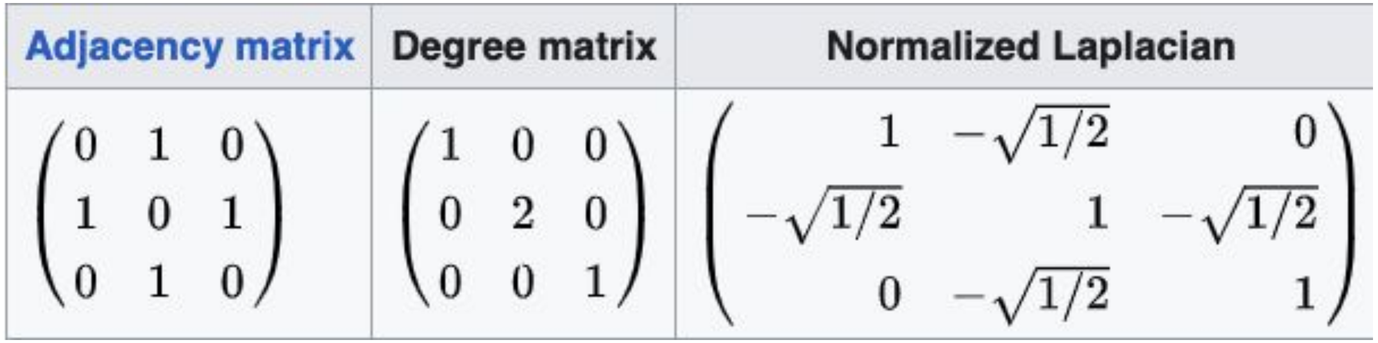
Given signal x=[1,2,3]’

$g_\theta \star x \approx \theta_0 T_0(L\sim)x + \theta_1 T_1(L\sim)x$

$L\sim = 2L/\lambda_{max} - I$

L is normalized Laplacian

$$L\sim = 2L/2 - I \quad \begin{matrix} 0 & -0.7071 & 0 \\ -0.7071 & 0.0000 & -0.7071 \\ 0 & -0.7071 & 0 \end{matrix}$$



Adjacency matrix	Degree matrix	Normalized Laplacian
$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -\sqrt{1/2} & 0 \\ -\sqrt{1/2} & 1 & -\sqrt{1/2} \\ 0 & -\sqrt{1/2} & 1 \end{pmatrix}$

$T_0(L{\sim})x=x$ $x=[1,2,3]'$

$T_1(L{\sim})x=L{\sim}x = -1.4142$

-2.8284

-1.4142

$y=\theta_0 x+\theta_1 L{\sim}x =$

0.2929

0.5858

2.2929

$\theta_0=1$ and $\theta_1=0.5$

Graph Convolutional Network (GCN)

- First-order approximation of ChebNet.

- T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. of ICLR, 2017

ChebNet and Chebyshev polynomials

- ChebNet takes the form:

$$x * g_\theta = \sum_{i=1}^{k} \theta_i T_i(\tilde{L})x$$

Where T_i is Chebyshev polynomials.

- $T_0(x) = 1$
- $T_1(x) = x$
- $T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x)$

First-order approximation

- Let’s find the first-order approximation of

$$x * g_\theta = \sum_{i=0}^{k} \theta_i \, T_i(\tilde{L})x = \theta_0 \, T_0(\tilde{L})x + \theta_1 T_1(\tilde{L})x = \theta_0 \, x + \theta_1 \tilde{L} x$$

$$\tilde{L} = \frac{2L}{\lambda_{max}} - I_n$$

Lambda_max is approximated to be 2

$$\approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

Normalized Laplacian $D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$ $I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

First-order approximation

- To restrain the number of parameters and avoid over-fitting, GCN further assume $\theta = \theta_0 = -\theta_1$

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

- This empirically causes numerical instability to GCN.

Numerical trick

- This empirically causes numerical instability to GCN. To address this problem, GCN applies a normalization trick to replace

$$\tilde{A} = A + I$$

$$\widetilde{D}_{ij} = \sum\nolimits_j \tilde{A}_{ij}$$

$$\bar{A} = \widetilde{D}^{-\frac{1}{2}} \tilde{A}\, \widetilde{D}^{-\frac{1}{2}}$$

a compositional layer can be defined as:

$$H' = X * g_\theta = \sigma(\bar{A} H \Theta)$$

One-step GCN normalization

$$A = \begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}, \quad X = \begin{bmatrix}1\\ 2\\ 3\end{bmatrix},$$

$\tilde{A} = A + I$, $\tilde{D} = \mathrm{diag}\big(row\ sums\ of\, \tilde{A}\big)$, and compute

$$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}, \quad \hat{X} = \hat{A}X.$$

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

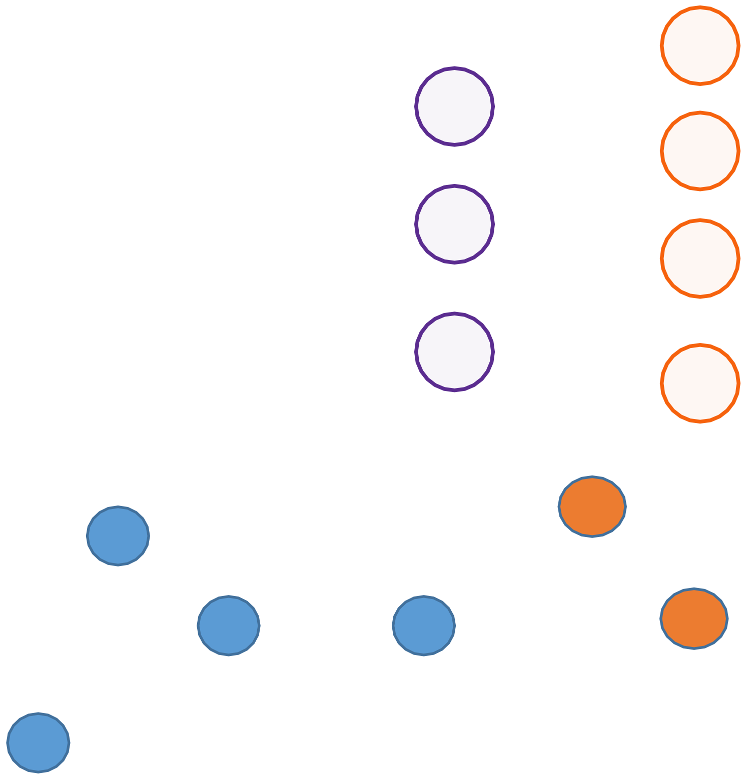
D~=diag(2,3,2)

$$\hat{A} \approx \begin{bmatrix}0.5 & 0.408 & 0\\ 0.408 & 0.333 & 0.408\\ 0 & 0.408 & 0.5\end{bmatrix} \Rightarrow \hat{X} \approx \begin{bmatrix}1.316\\ 2.298\\ 2.684\end{bmatrix}$$

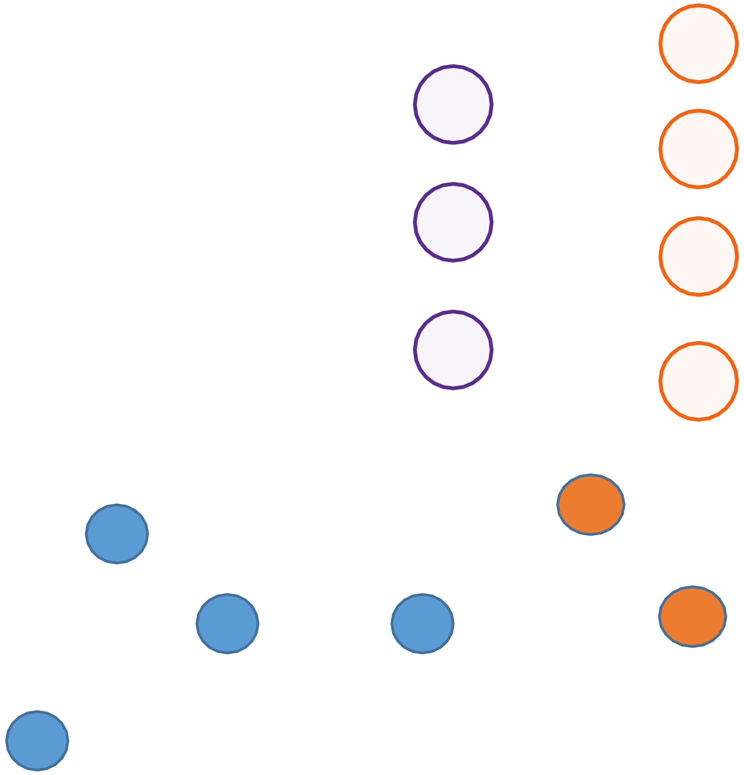
Aggregation of information

- $H' = \sigma \quad (H. \quad)$

  Θ



- $H' = \sigma(H\Theta)$

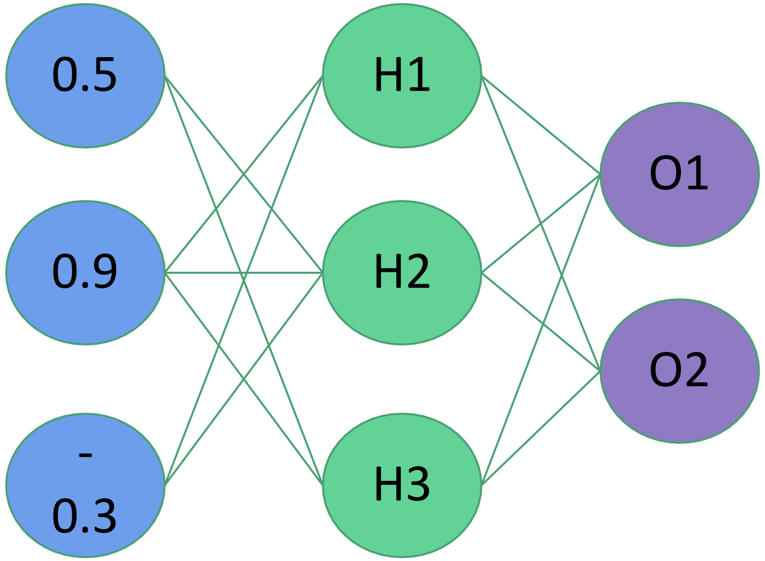


- $H' = \sigma(\bar{A}H\Theta)$

Towards more general frameworks

MLPs

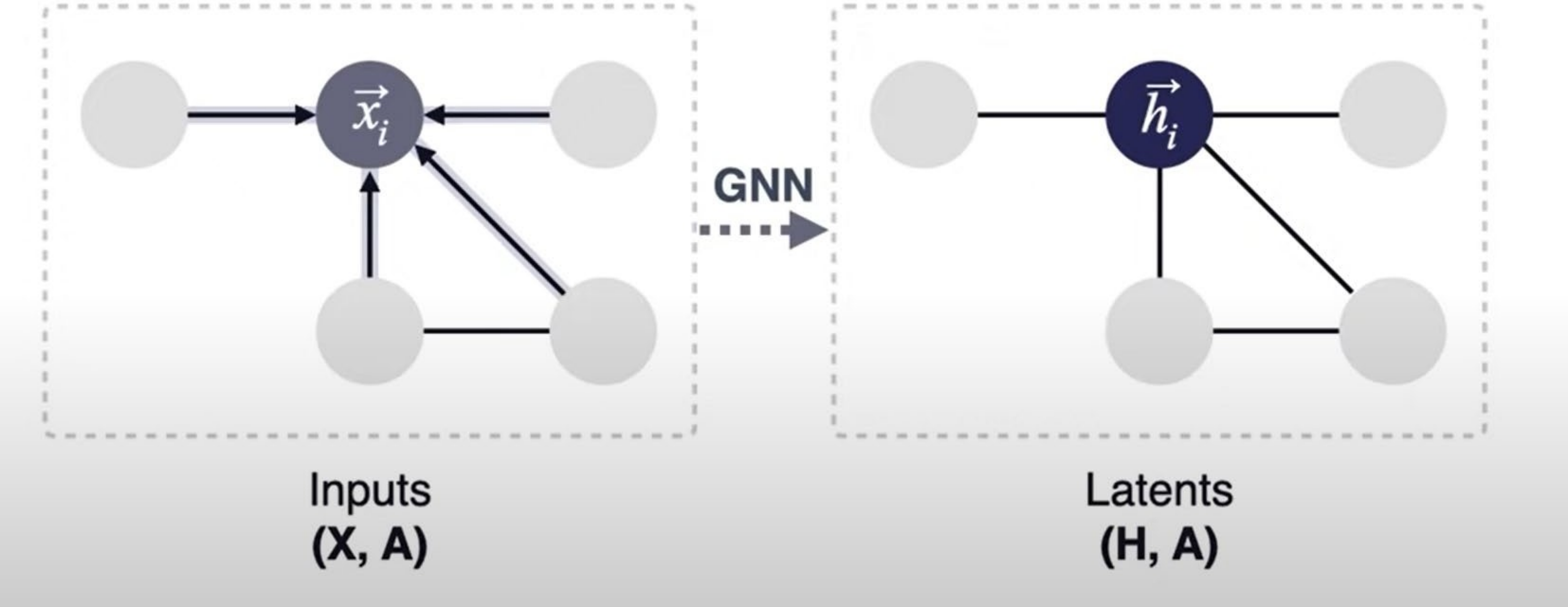
Feed forward : $H' = \sigma\, H\Theta\ (\qquad)$



GNNs

We can aggregate neighbourhoods by multiplying the adjacency matrix.

Graph neural network: $H' = \sigma\left(AH\Theta\right)$



Sum-pooling

- This update rule discards the central node.

Sum-pooling

- This update rule discards the central node.

- This can be fixed simply by

$$\tilde{A} = A + I$$

Sum-pooling

- This update rule discards the central node.

- This can be fixed simply by

$$\tilde{A} = A + I$$

- The node-wise update rule can be written as:

$$h_i' = \sigma\left(\sum_{j \in N_i} \Theta h_j\right)$$

Mean-pooling

- Summing the contents of the neighbouring nodes will increase the scale of the output feature.

Mean-pooling

- Summing the contents of the neighbouring nodes will increase the scale of the output feature
- We can normalize by $\widetilde{D}^{-1}$ $\widetilde{D}_{ij} = \sum_j \tilde{A}_{ij}$

$$h_i' = \sigma\left(\sum_{j \in N_i} \widetilde{D}^{-1} \Theta h_j\right)$$

- The node-wise update rule can b

$$h_i' = \sigma\left(\sum_{j \in N_i} \frac{1}{|N_i|} \Theta h_j\right)$$

Graph Convolutional Networks (GCNs)

- Use symmetric normalization

$$H' = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A}\, \tilde{D}^{-\frac{1}{2}}\ H\Theta\right)$$

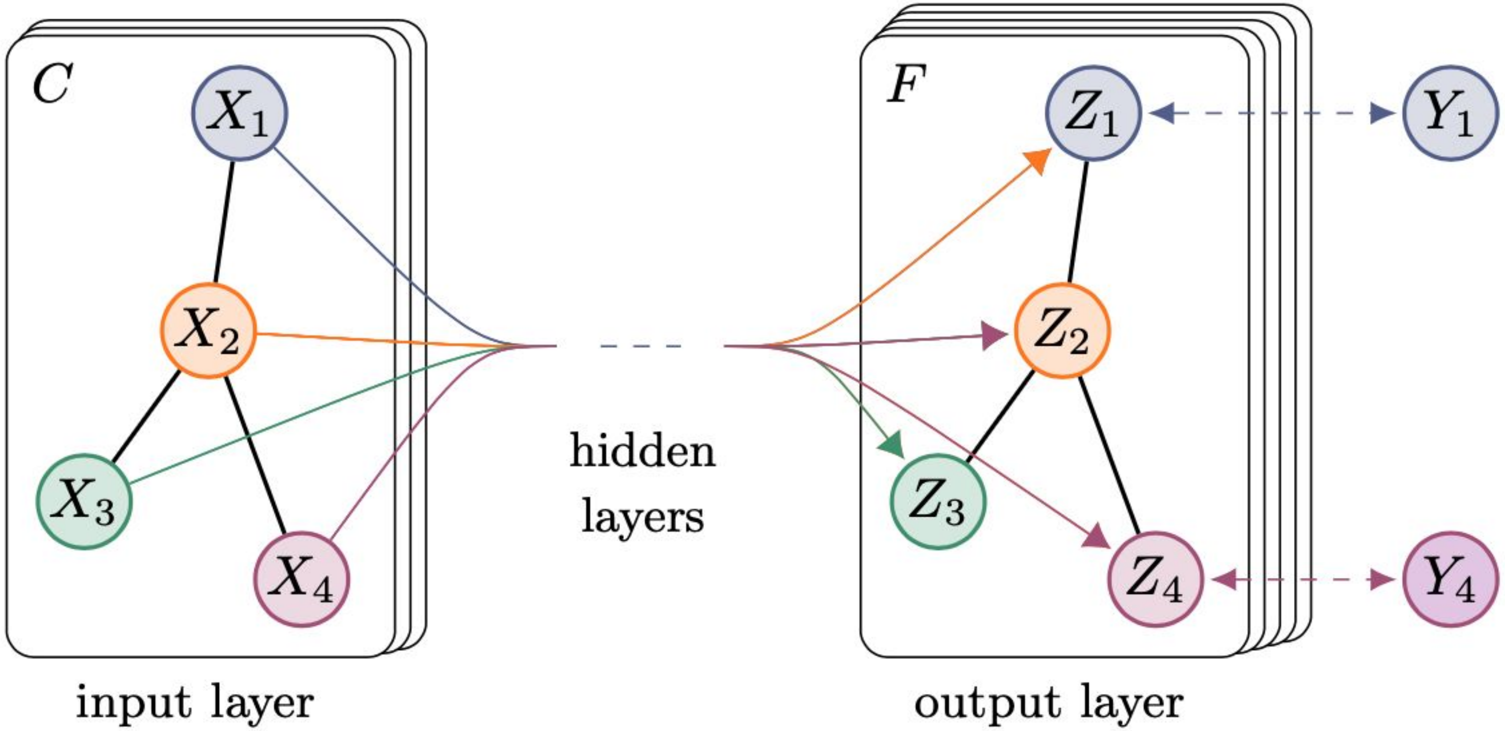
- The node-wise update rule can be written as:

$$h_i' = \sigma\left(\sum_{j \in N_i} \frac{1}{\sqrt{|N_i||N_j|}} \Theta h_j\right)$$

GCN architecture

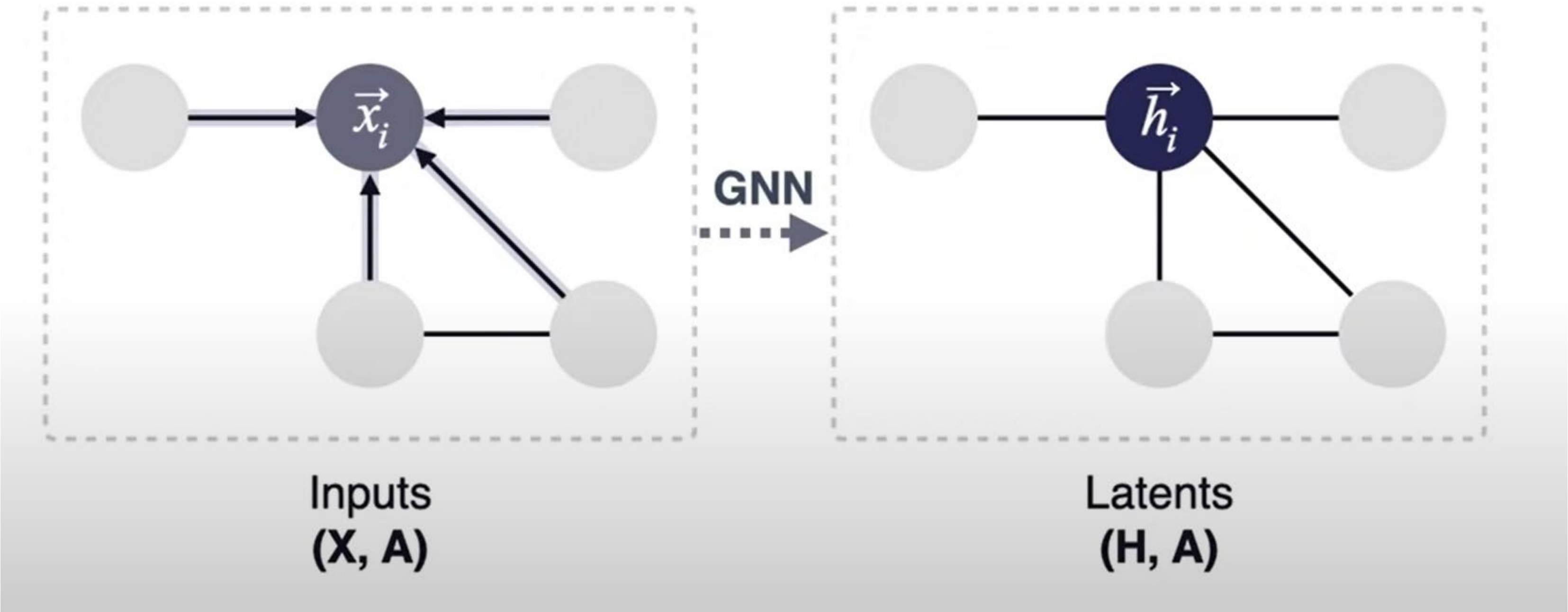
we consider a two-layer GCN $Z = f(X, A)$ = sigmoid $\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)$

W(0) $\in \mathrm{R}^{C\times d}$ W(1) $\in \mathrm{R}^{d\times 1}$

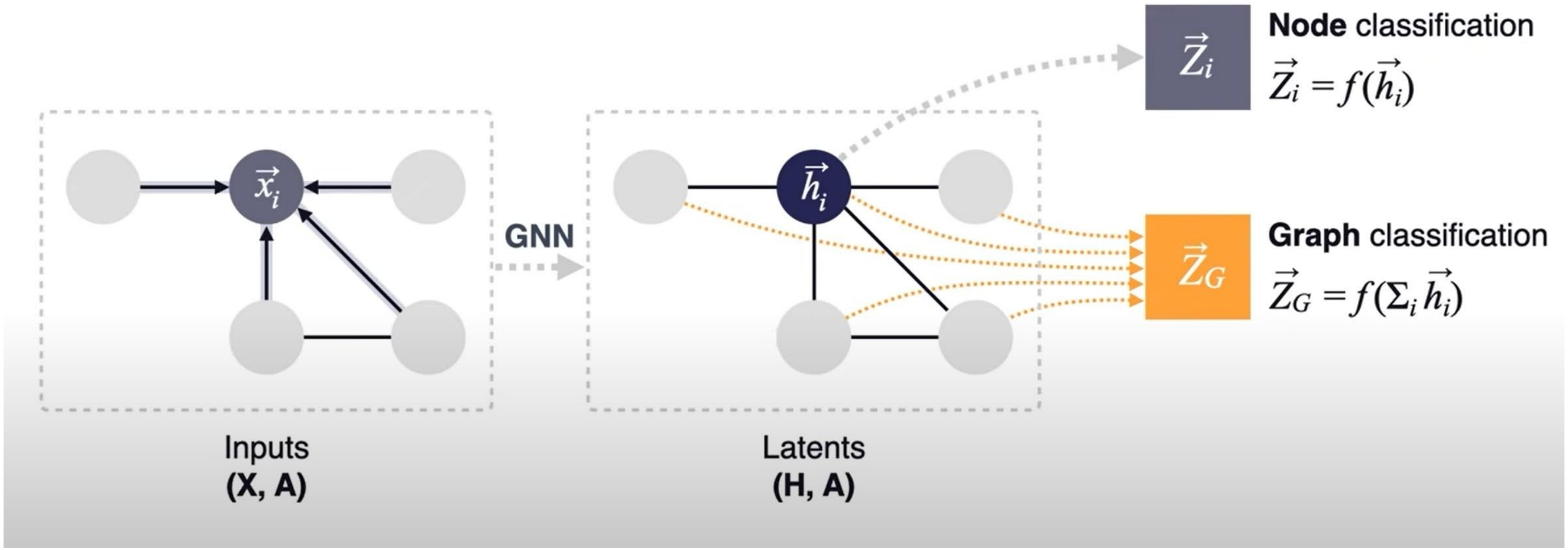


C is dimensionality of node

Node Classification



Graph Classification



Link Classification

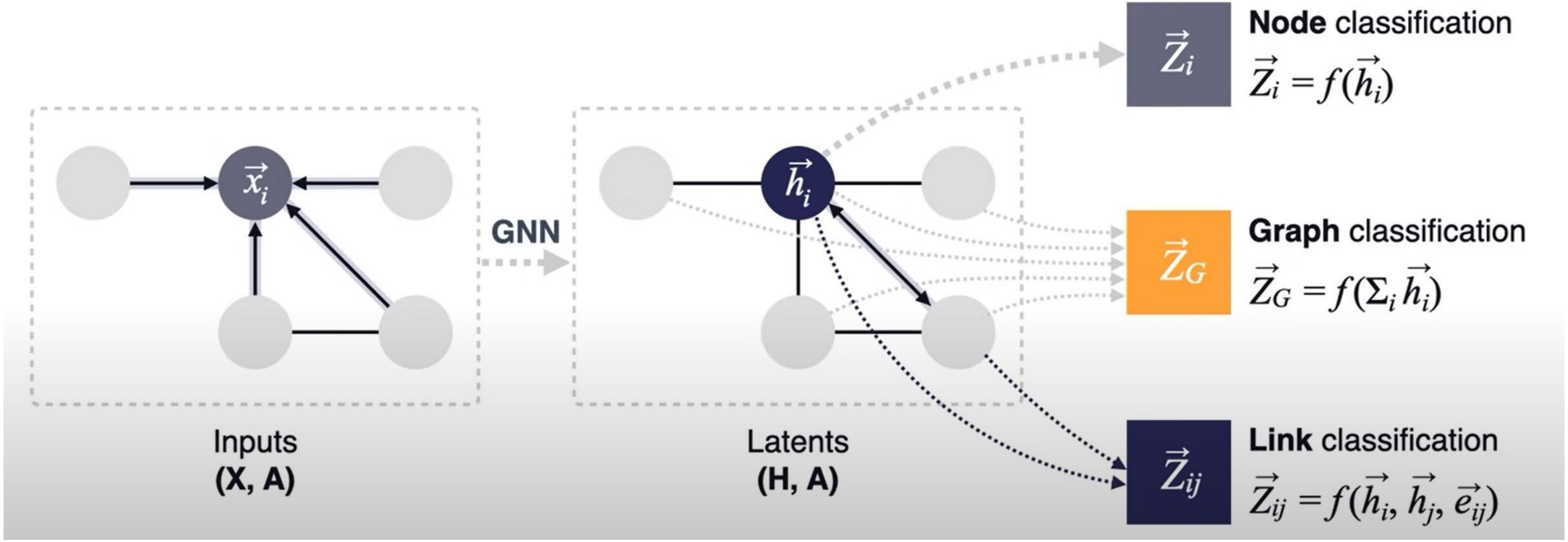


Image by Petaí Veličković

- Sum pooling $h_i' = \sigma\left(\sum_{j \in N_i} \Theta h_j\right)$

- Mean pooling $h_i' = \sigma\left(\sum_{j \in N_i} \frac{1}{|N_i|} \Theta h_j\right)$

Can you think of alternatives?

- GCNs $h_i' = \sigma\left(\sum_{j \in N_i} \frac{1}{\sqrt{|N_i||N_j|}} \Theta h_j\right)$

Graph Attention Network (GAT)

- GAT adopts attention mechanisms to learn the relative weights between two connected nodes.

$$h_i' = \sigma\left(\sum_{j\in N_i} \alpha_{ij}\, \Theta h_j\right)$$

The attention weight measures the influence of node *j* to node *i*

P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” in Proc. of ICLR, 2018

The attention weight

- The attention weight can be computed as follows:

$$a_{ij} = a(h_i, h_j\ )$$

or

$$a_{ij} = a(h_i, h_j, e_{ij})$$

- a is a single-layer feedforward neural network.

Attention Computation:

- ► $Z := \text{attention}(Q, K, V) = V\text{softmax}\left(\frac{1}{\sqrt{p}} Q^T K\right)$

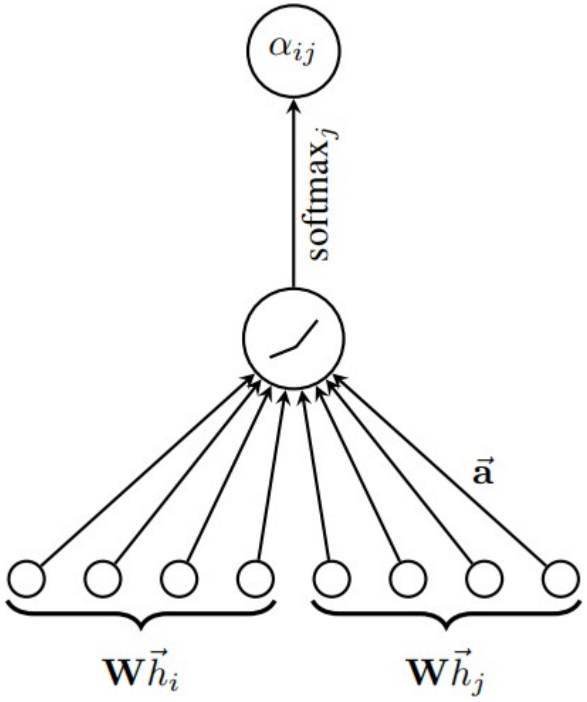
- a can be other functions for example a Transformer.

$$\alpha_{ij} = \frac{e^{a_{ij}}}{\sum_{k \in N_i} e^{a_{ik}}}$$

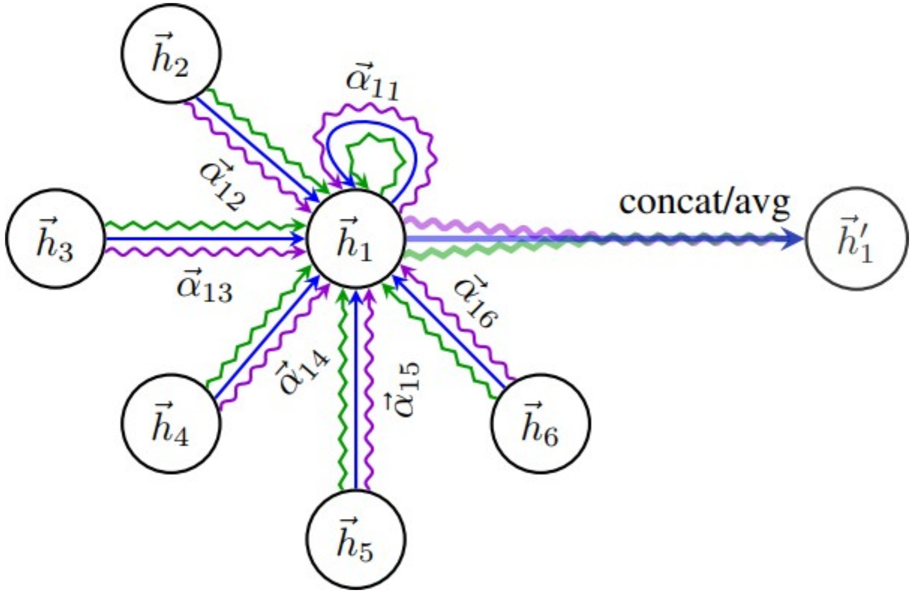
Multi-head attention in a single GAT

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

3 heads – 3 colors



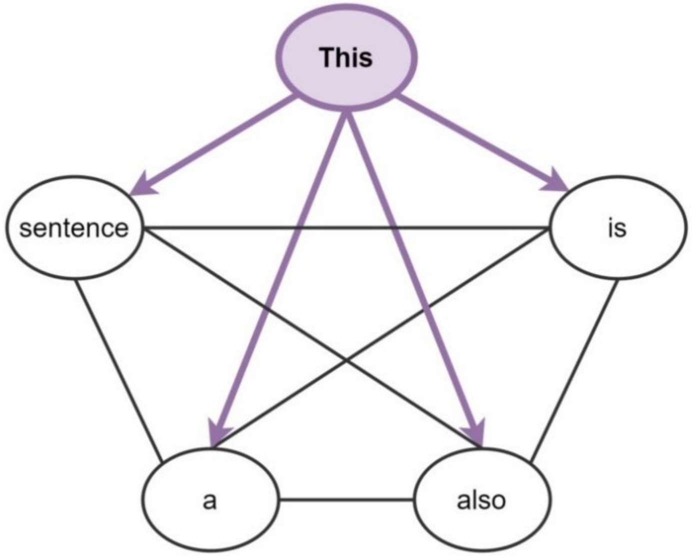
The attention mechanism



The multi-head attention. Different arrow styles and colours denote independent attention computations. The aggregated features from each head are concatenated or averaged.

Transformers are Graph Neural Networks

This is also a sentence



GA

- The attention weight can be computed as follows:

$$a_{ij} = a(h_i, h_j, e_{ij})$$

- a is a single-layer feedforward neural network.

- $\alpha_{ij} = \frac{e^{a_{ij}}}{\sum_{k \in N_i} e^{a_{ik}}}$

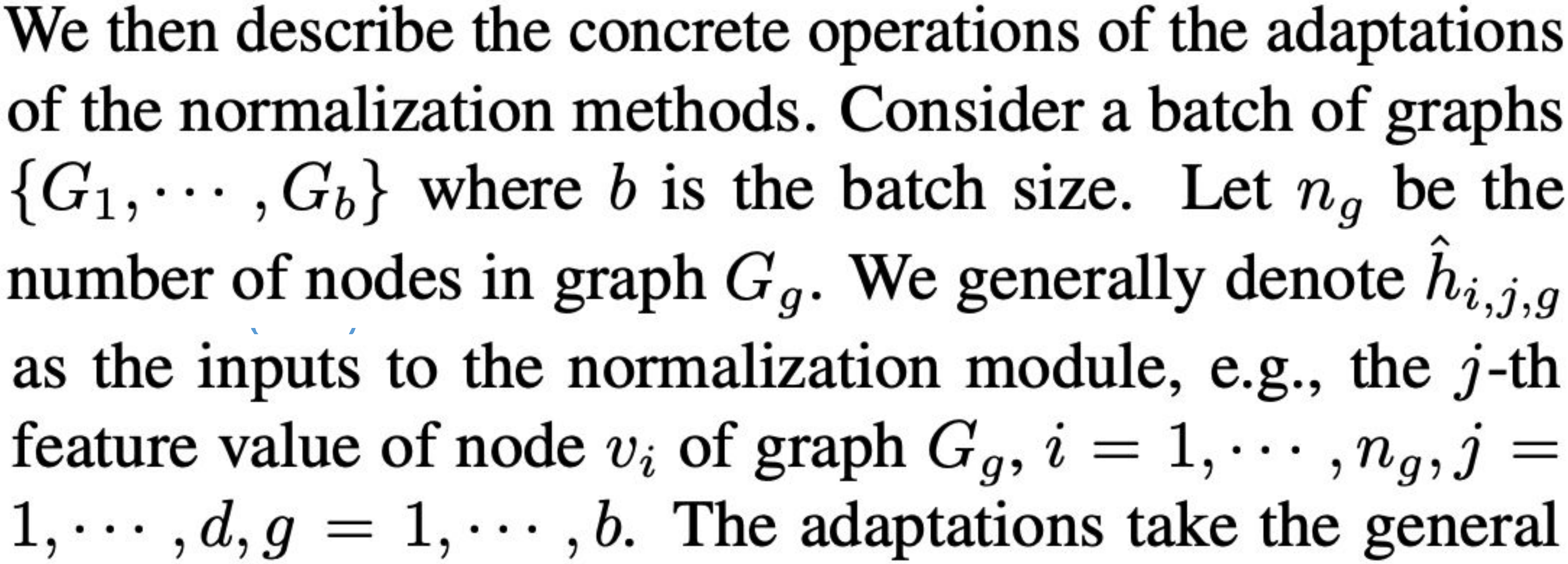
Transforme

- The attention weight can be computed as follows:

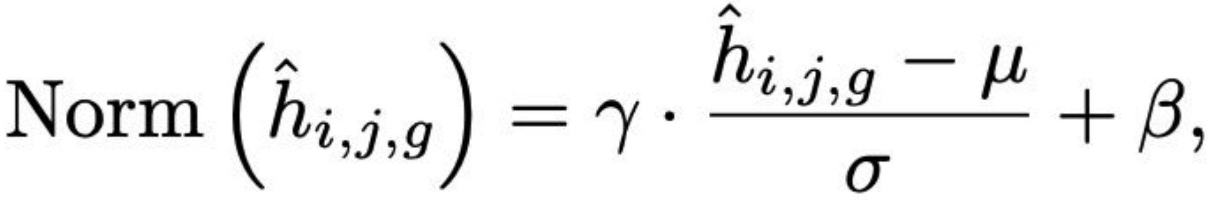
$$a_{ij} = a(q_i, k_j)$$

- $a = \frac{1}{\sqrt{p}}(q_i^T k_j)$

- $\alpha_{ij} = \frac{e^{a_{ij}}}{\sum_k e^{a_{ik}}}$ $\quad v_i = \sum_j \alpha_{ij} v_j$



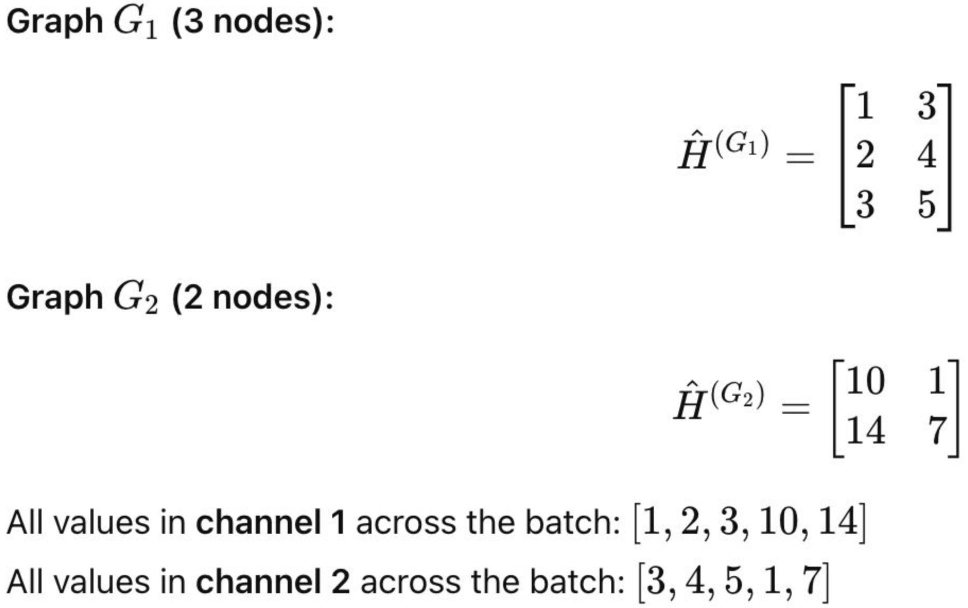
We then describe the concrete operations of the adaptations of the normalization methods. Consider a batch of graphs $\{G_1, \cdots, G_b\}$ where b is the batch size. Let n_g be the number of nodes in graph G_g. We generally denote $\hat{h}_{i,j,g}$ as the inputs to the normalization module, e.g., the j-th feature value of node v_i of graph G_g, $i = 1, \cdots, n_g$, $j = 1, \cdots, d$, $g = 1, \cdots, b$. The adaptations take the general



$$\mathrm{Norm}\left(\hat{h}_{i,j,g}\right) = \gamma \cdot \frac{\hat{h}_{i,j,g} - \mu}{\sigma} + \beta,$$

GraphNorm, ICML 2021

For BatchNorm, normalization and the computation of μ and σ are applied to all values in the same feature dimension across the nodes of all graphs in the batch as in Xu et al. (2019), i.e., over dimensions g, i of hˆ i,j,g



Graph G_1 (3 nodes):

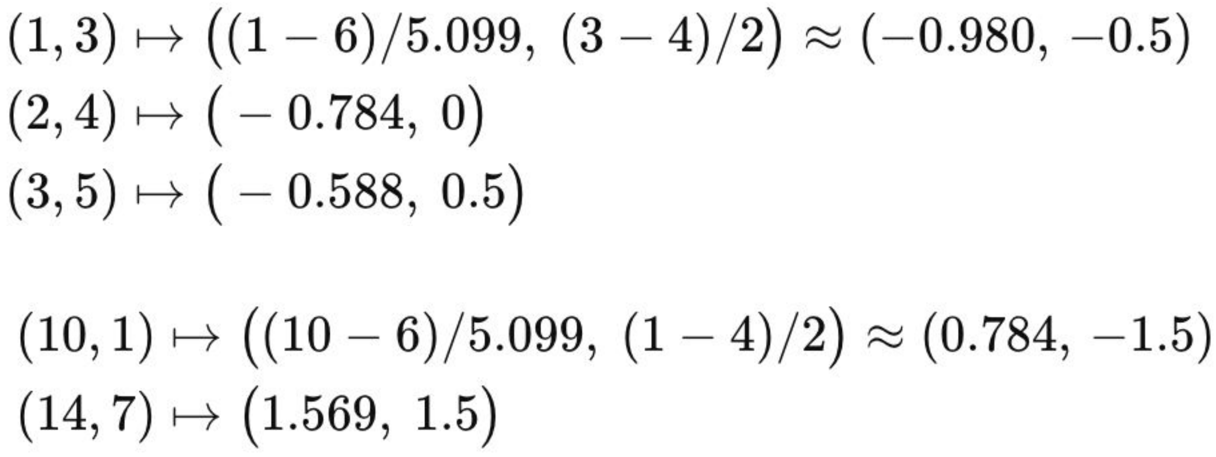
$$\hat{H}^{(G_1)} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix}$$

Graph G_2 (2 nodes):

$$\hat{H}^{(G_2)} = \begin{bmatrix} 10 & 1 \\ 14 & 7 \end{bmatrix}$$

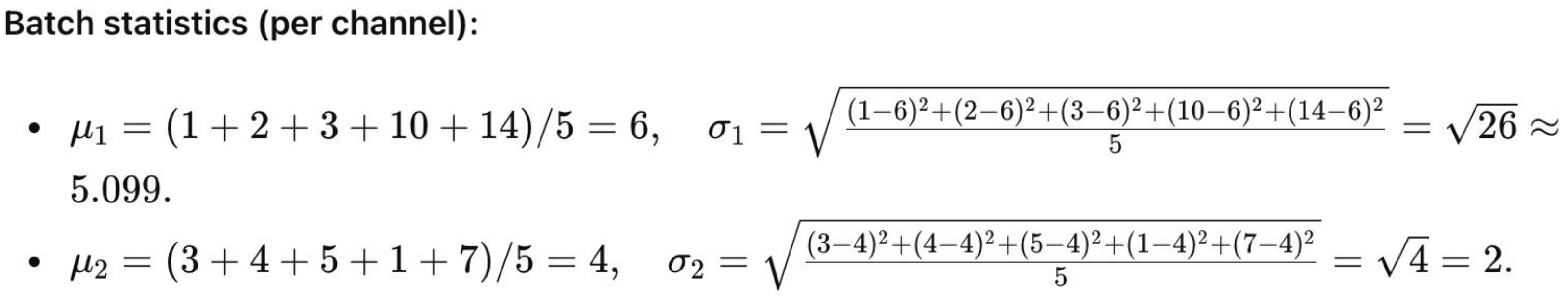
All values in **channel 1** across the batch: $[1, 2, 3, 10, 14]$

All values in **channel 2** across the batch: $[3, 4, 5, 1, 7]$



$$(1,3) \mapsto \big((1-6)/5.099,\ (3-4)/2\big) \approx (-0.980,\ -0.5)$$
$$(2,4) \mapsto \big(-0.784,\ 0\big)$$
$$(3,5) \mapsto \big(-0.588,\ 0.5\big)$$

$$(10,1) \mapsto \big((10-6)/5.099,\ (1-4)/2\big) \approx (0.784,\ -1.5)$$
$$(14,7) \mapsto \big(1.569,\ 1.5\big)$$



Batch statistics (per channel):

- $\mu_1 = (1+2+3+10+14)/5 = 6, \quad \sigma_1 = \sqrt{\frac{(1-6)^2+(2-6)^2+(3-6)^2+(10-6)^2+(14-6)^2}{5}} = \sqrt{26} \approx 5.099.$
- $\mu_2 = (3+4+5+1+7)/5 = 4, \quad \sigma_2 = \sqrt{\frac{(3-4)^2+(4-4)^2+(5-4)^2+(1-4)^2+(7-4)^2}{5}} = \sqrt{4} = 2.$

For BatchNorm, normalization and the computation of μ and σ are applied to all values in the same feature dimension across the nodes of all graphs in the batch as in Xu et al. (2019), i.e., over dimensions g, i of hˆ i,j,g

To adapt LayerNorm to GNNs, we view each node as a basic component, resembling (words) in a sentence, and apply normalization to all feature values across different dimensions of each node, i.e., over dimension j of hˆ i,j,g.

For InstanceNorm, we regard each graph as an instance. The normalization is then applied to the feature values across all nodes for each individual graph, i.e., over dimension i of hˆ i,j,g.

$$\text{GraphNorm}\left(\hat{h}_{i,j}\right) = \gamma_j \cdot \frac{\hat{h}_{i,j} - \alpha_j \cdot \mu_j}{\hat{\sigma}_j} + \beta_j, \quad \mu_j = \frac{\sum_{i=1}^{n} \hat{h}_{i,j}}{n}, \hat{\sigma}_j^2 = \frac{\sum_{i=1}^{n} \left(\hat{h}_{i,j} - \alpha_j \cdot \mu_j\right)^2}{n},$$

InstanceNorm

$$\hat{H}^{(G_1)} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix}, \qquad \mu^{(G_1)} = [2,\ 4]. \qquad \hat{H} - \mu = \begin{bmatrix} -1 & -1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}.$$

Var.
2/3

(   )

$$\frac{\hat{H} - \mu}{\sigma} \approx \begin{bmatrix} -1.2247 & -1.2247 \\ 0 & 0 \\ 1.2247 & 1.2247 \end{bmatrix}.$$

GraphNorm

α=[1, 0.5]

$$\hat{H} - \alpha \odot \mu = \begin{bmatrix} 1-2 & 3-2 \\ 2-2 & 4-2 \\ 3-2 & 5-2 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & 2 \\ 1 & 3 \end{bmatrix}$$

Var. 2/3, 14/3

$$\frac{\hat{H} - \alpha\mu}{\sigma} \approx \begin{bmatrix} -1.2247 & 0.4629 \\ 0 & 0.9258 \\ 1.2247 & 1.3887 \end{bmatrix}$$