Lecture 19

Graph Neural Network- Part 2

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:

$$x * g_\theta = \sum_{i=0}^{k} \theta_i T_i(\tilde{L})x$$

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

First-order approximation

- Let’s find the first-order approximation of ChebNet.

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

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

First-order approximation

- Let’s find the first-order approximation of ChebNet.

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

$$= \theta_0 \; x + \theta_1 \tilde{L}x$$

$T_0(x) = 1$

$T_1(x) = x$

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

First-order approximation

- To restrain the number of parameters and avoid over-fitting, GCN further assume $\theta = \theta_0 = \theta_1$
- $x * g_\theta = \sum_{i=0}^{k} \theta_i T_i(\tilde{L})x = \theta_0 T_0(\tilde{L})x + \theta_1 T_1(\tilde{L})$

$$= \theta_0 x + \theta_1 \tilde{L}x$$

assume $\theta = \theta_0 = \theta_1$

$$= \theta x + \theta \tilde{L}x$$

First-order approximation

- To restrain the number of parameters and avoid over-fitting, GCN further assume $\theta = \theta_0 = \theta_1$
- $x * g_\theta = \sum_{i=0}^{k} \theta_i T_i(\tilde{L})x = \theta_0 T_0(\tilde{L})x + \theta_1 T_1(\tilde{L})$

$$= \theta_0 x + \theta_1 \tilde{L}x$$
$$= \theta x + \theta \tilde{L}x$$
$$= \theta(I + \tilde{L})x$$

First-order approximation

- To restrain the number of parameters and avoid over-fitting, GCN further assume $\theta = \theta_0 = \theta_1$
- $x * g_\theta = \sum_{i=0}^{k} \theta_i T_i(\tilde{L})x = \theta_0 T_0(\tilde{L})x + \theta_1 T_1(\tilde{L})$

$$
\begin{aligned}
&= \theta_0 x + \theta_1 \tilde{L} x \\
&= \theta x + \theta \tilde{L} x \\
&= \theta\left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x
\end{aligned}
$$

First-order approximation

- To restrain the number of parameters and avoid over-fitting, GCN further assume $\theta = \theta_0 = \theta_1$
- $x * g_\theta \; = \sum_{i=0}^{k} \theta_i \, T_i \left(\tilde{L}\right) x \; = \; \theta_0 \, T_0\left(\tilde{L}\right) x + \theta_1 T_1\left(\tilde{L}\right)$

$$= \theta_0 \, x + \theta_1 \tilde{L} x$$
$$= \theta \, x + \theta \tilde{L} x$$
$$= \theta \left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

- 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

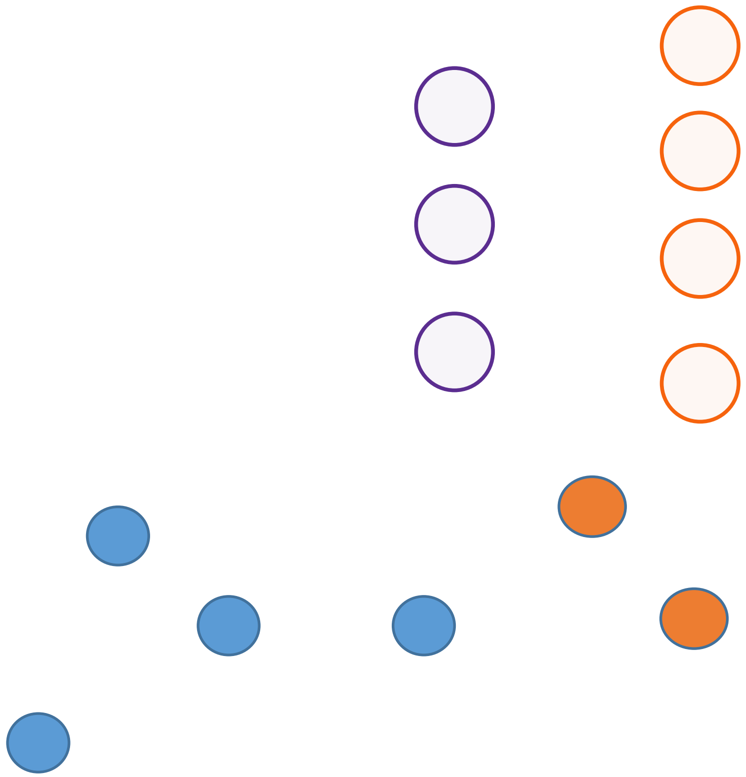
- $\tilde{A} = A + I$
- $\widetilde{D}_{ij} = \sum_j \tilde{A}_{ij}$
- $\bar{A} = \widetilde{D}^{-\frac{1}{2}} \tilde{A} \, \widetilde{D}^{-\frac{1}{2}}$

- a compositional layer can be defined as:

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

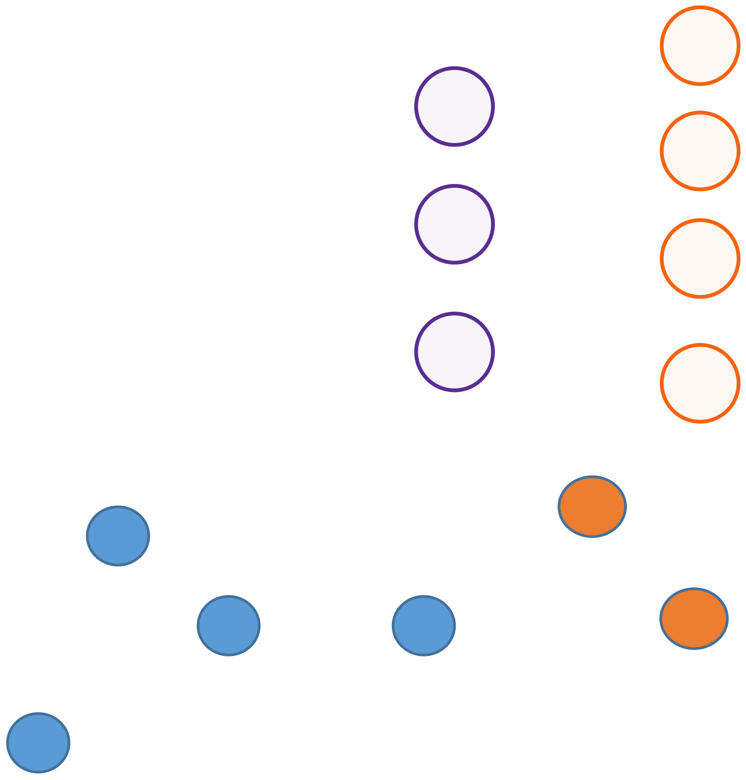
Aggregation of information

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

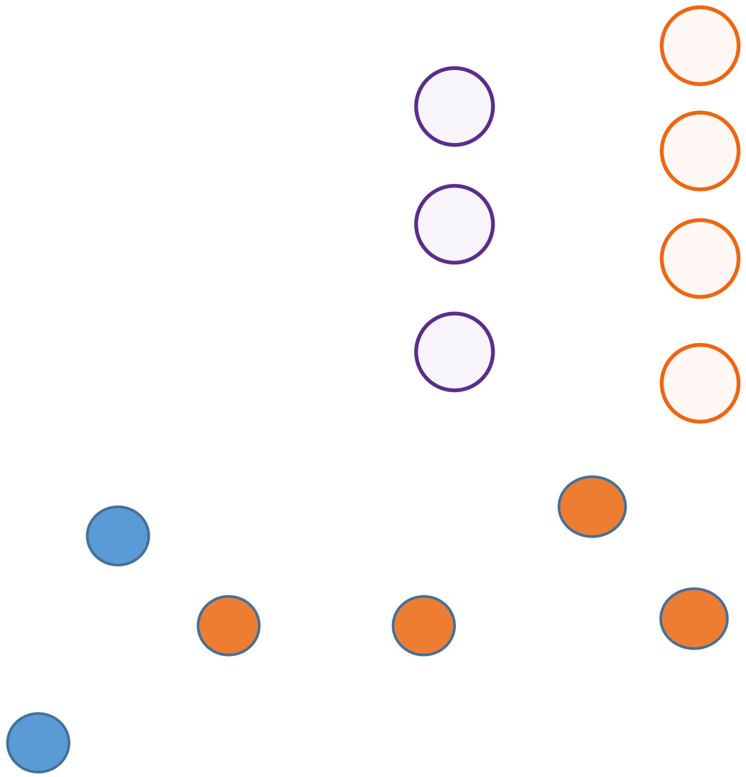


- $H' = \sigma(H\Theta)$

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



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

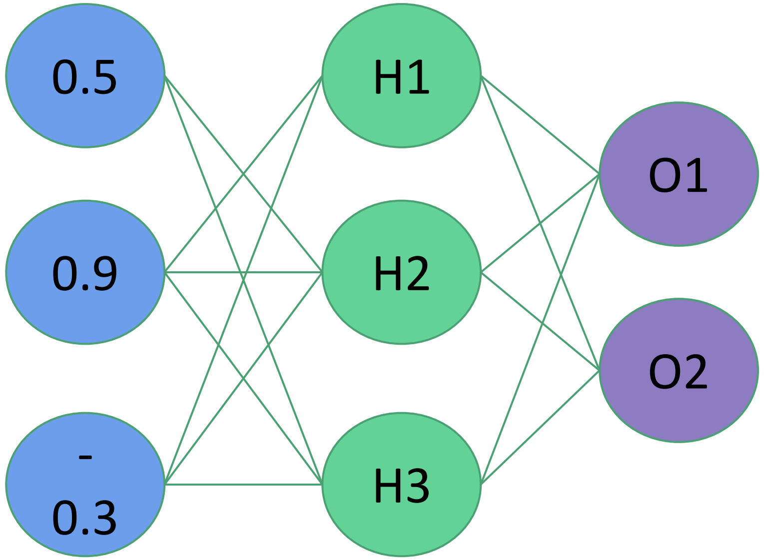


- $H' = \sigma(\bar{A}H\Theta)$

Towards more general frameworks

MLPs

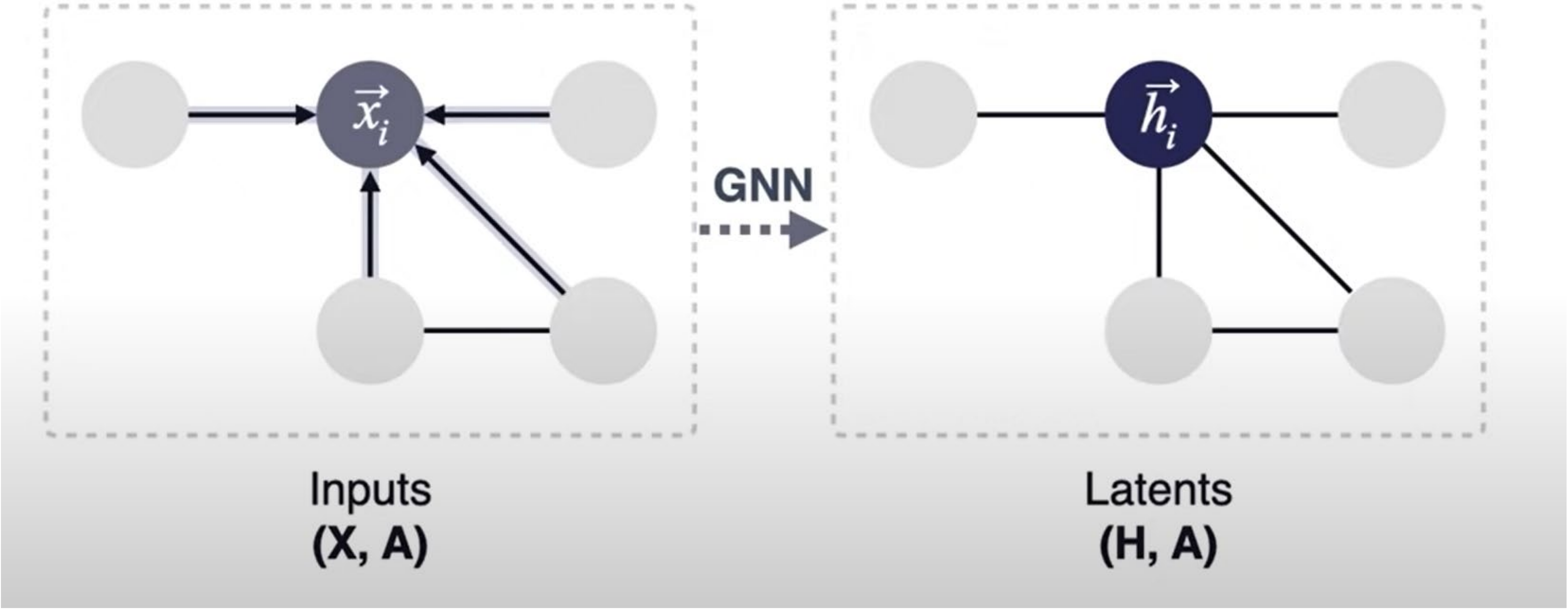
$Feed\ forward\ :\quad H' = \sigma(H\Theta)$



GNNs

We can aggregate neighbourhoods by multiplying the adjacency matrix.

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



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 $\widetilde{D}^{-1}$ where $\widetilde{D}_{ij} = \sum_j \tilde{A}_{ij}$

Mean-pooling

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

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

Mean-pooling

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

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

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

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

Graph Convolutional Networks (GCNs)

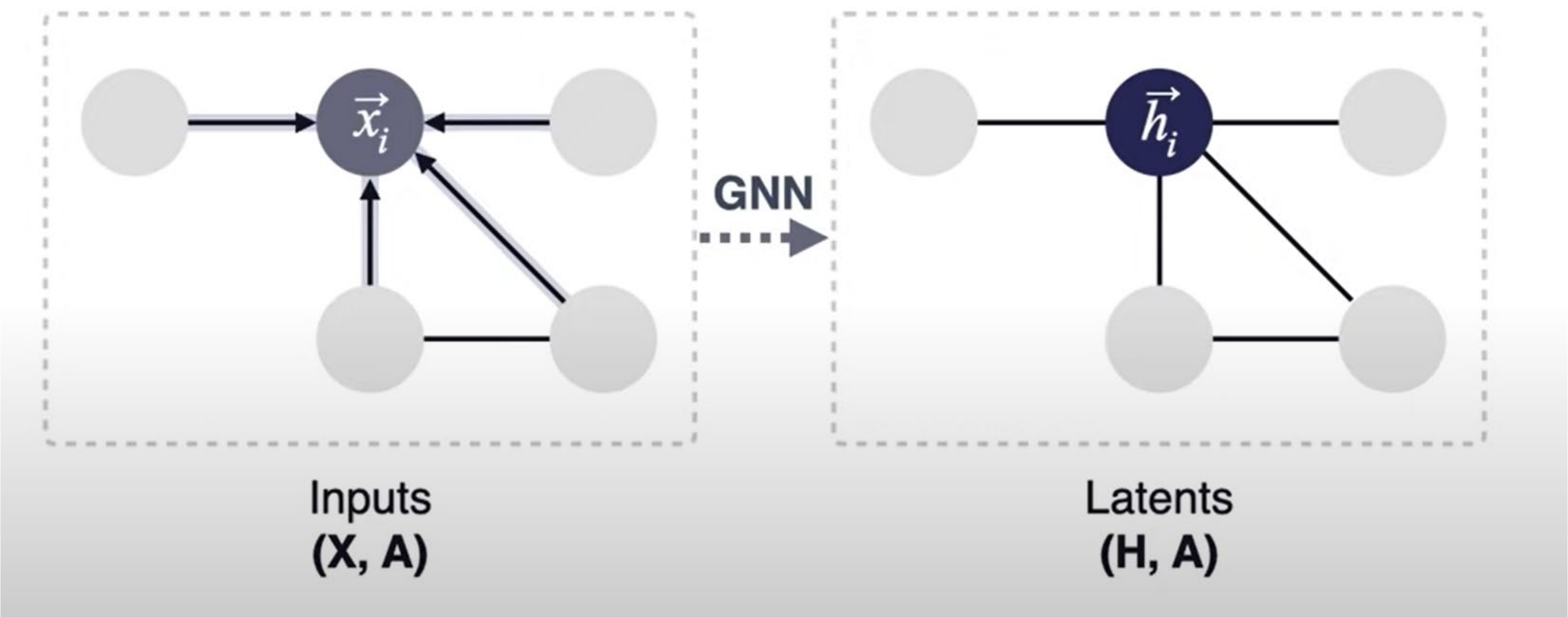
- Use symmetric normalization

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

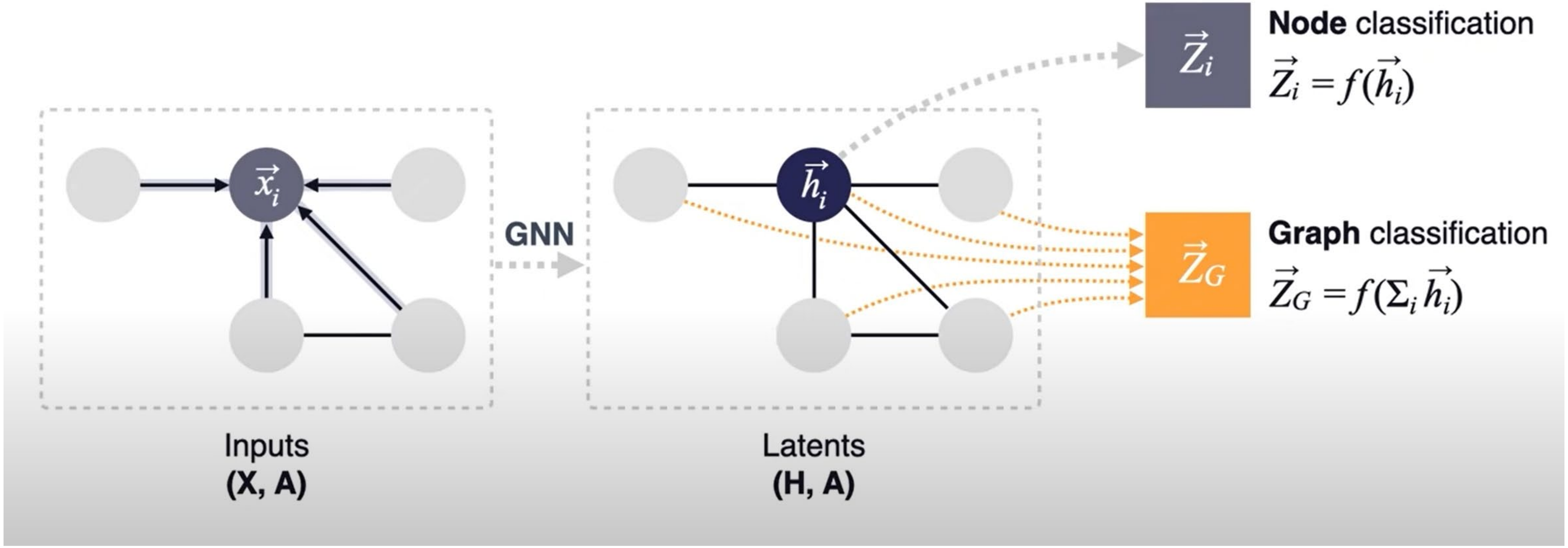
- The node-wise update rule can be written as:

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

Node Classification



Graph Classification



Link Classification

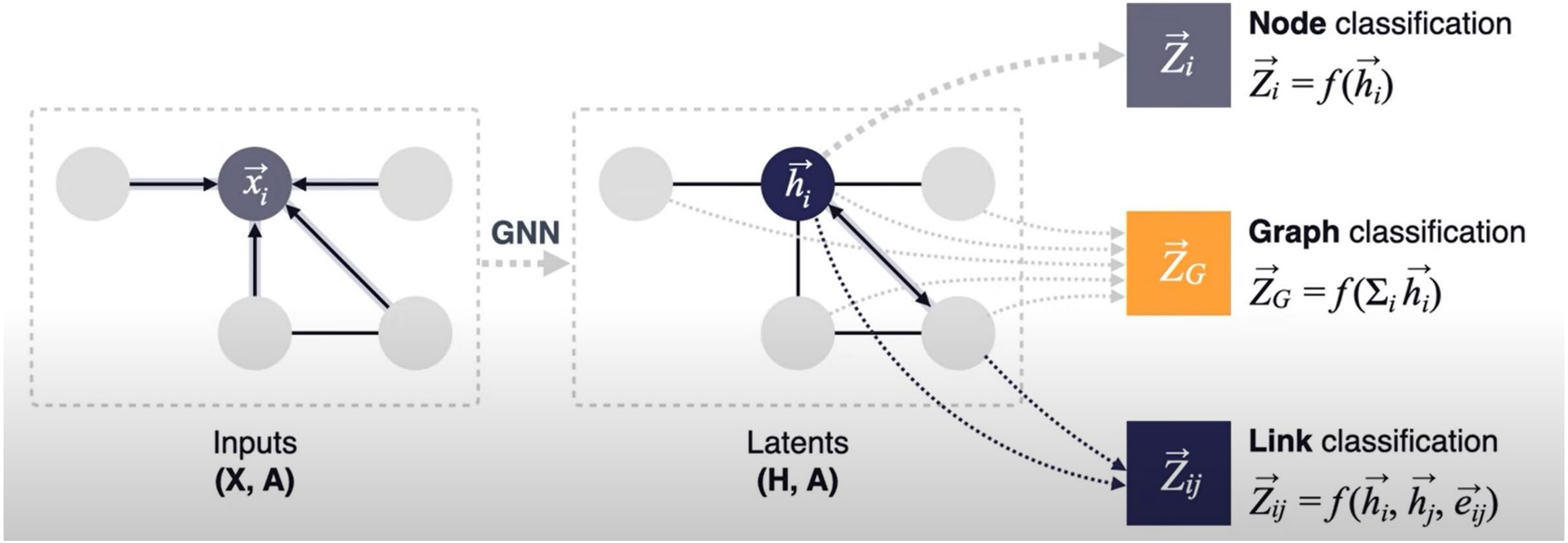


Image by Petar Veličković

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

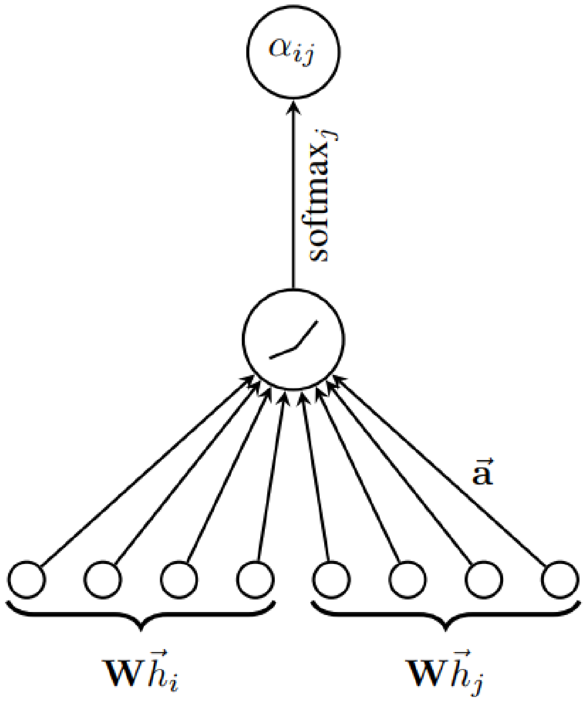
$$a_{ij} = a(h_i, h_j, e_{ij})$$

- a is a single-layer feedforward neural network.

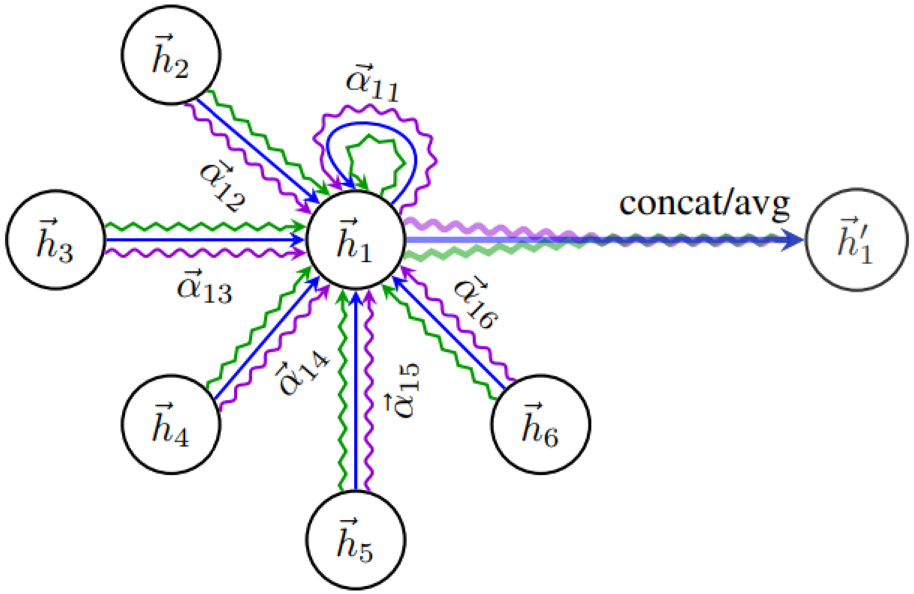
- a can be other functions for example a Transformer.

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

Multi-head attention in a single GAT step



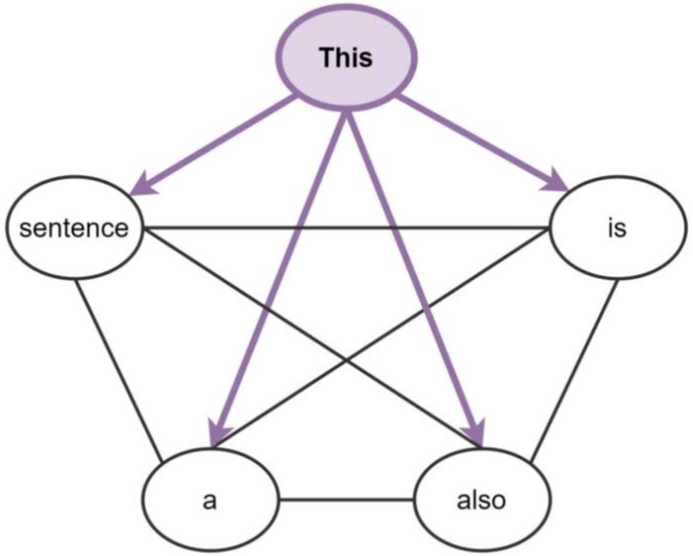
The attention mechanism



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



GAT

- 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.

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

Transformer

- The attention weight can be computed as follows:
  $$a_{ij} = a(q_i, k_j)$$

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

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