GraphDL4NLP

Deep Learning on Graphs for NLP

Sandya Mannarswamy – Principal Engineer, Intel India

Puneesh Khanna – Deep Learning Engineer, Intel India

Divyasree Tummalapalli – Research Scientist, Intel India

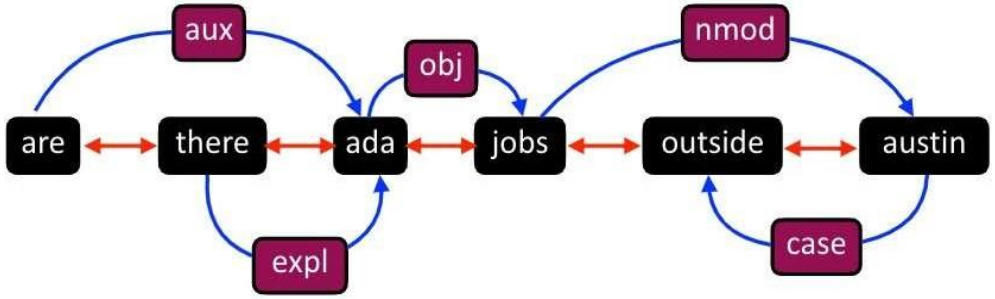
Agenda

- Background & Context
- GraphDL4NLP – Architectures
- GraphDL4NLP – Applications
- GraphDL4NLP – Libraries & Demo

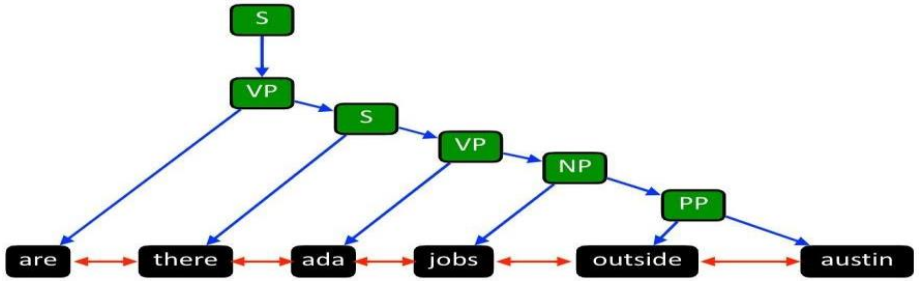
Deep Learning in NLP

- Representation
  - Word Embeddings – Word2vec/Glove
  - Sentence/Document Embeddings – Doc2vec/Para2vec
- Popular Models
  - Encoder-Decoder frameworks
    - RNNs
    - Transformers
  - Attention (Single Headed/Multi-headed)
- Increasing adoption of Pre-trained Language Models
  - BERT/GPT-3/T5

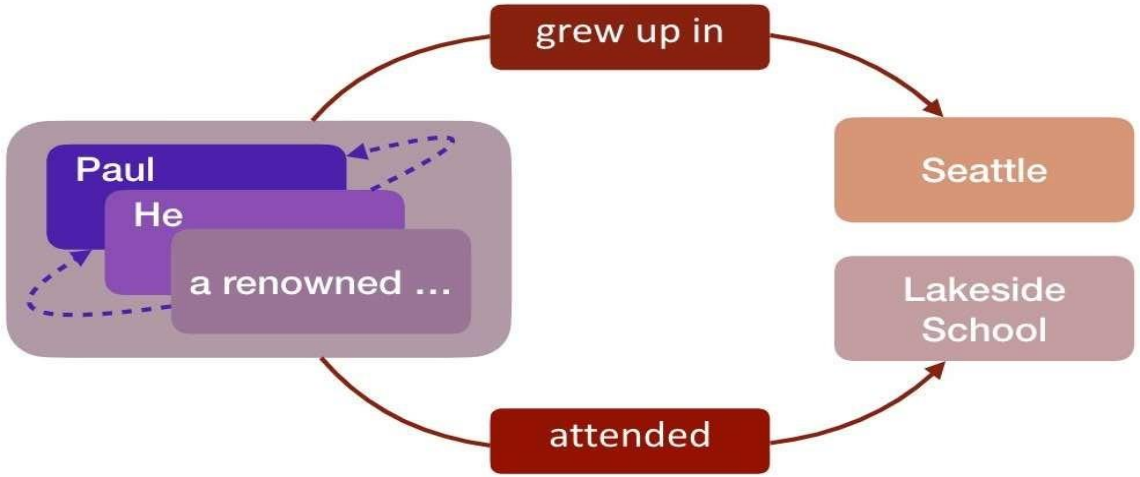
Graphs in NLP



Dependency graph



Constituency graph



IE Graph

NLP – A Graph Perspective

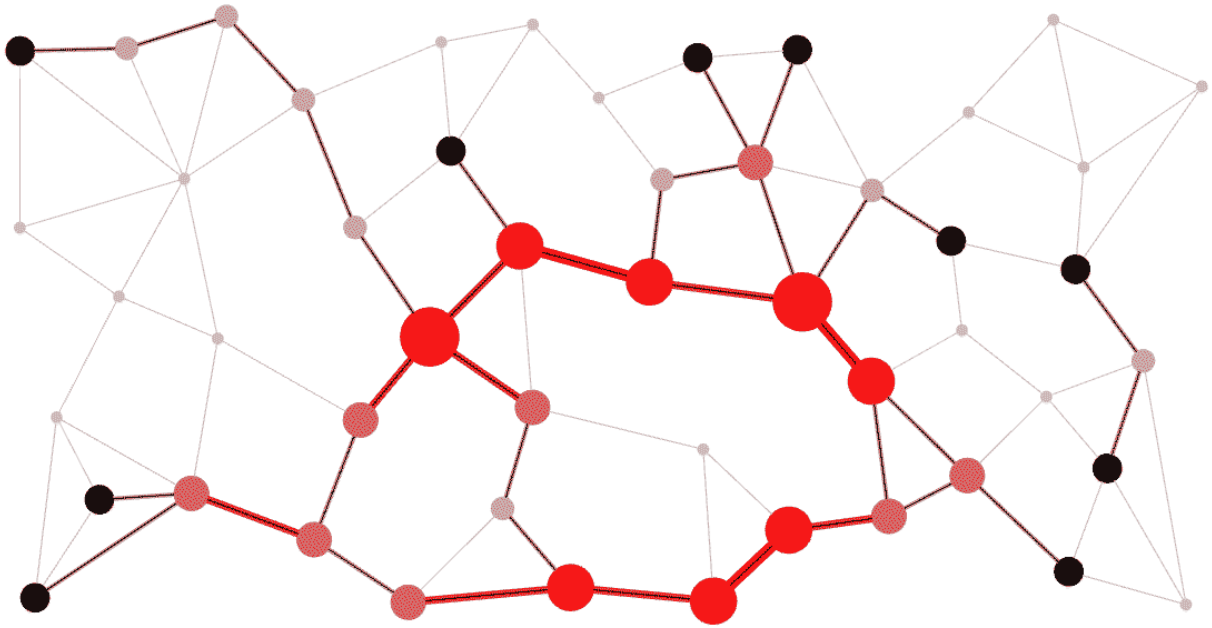
- Represent natural language as a bag of tokens
  - BOW, TF-IDF
- Represent as a linear sequence of tokens
  - Word/Sentence/Para/Doc embeddings
  - Seq2Seq models
- Represent natural language as a graph
  - Dependency graphs, constituency graphs, IE graphs
  - Text graph containing multiple hierarchies of elements, i.e. document, sentence and word

Graph Methods in NLP Tasks

- Text Document Classification – Graph Clustering
- Information Extraction – Sentence/document represented as a graph
- Neural Machine Translation – Dependency Graph as auxiliary info
- Question Answering – Graph to Sequence models

Building block for all the above is Graph Neural Networks!

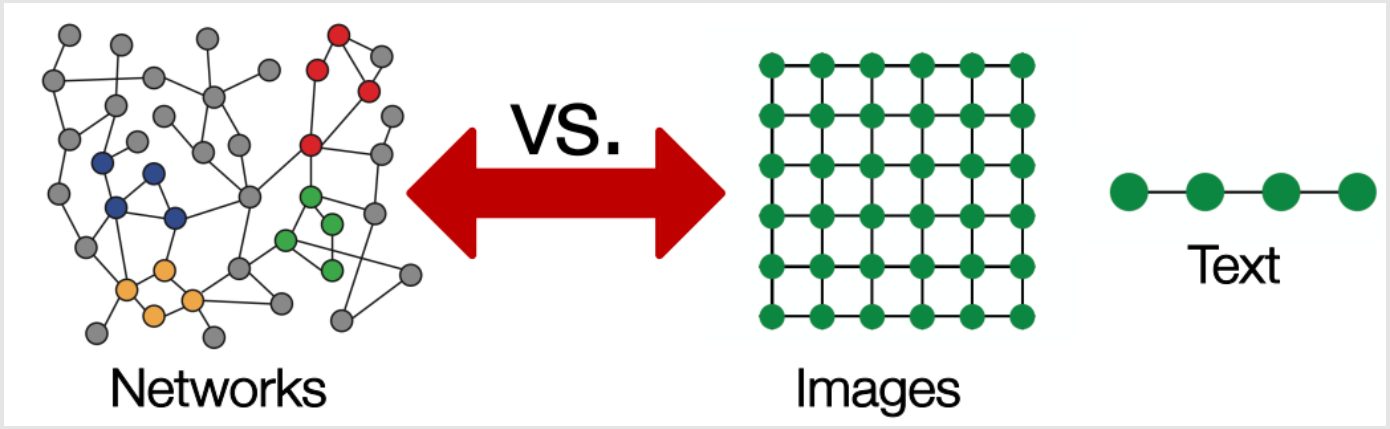
Graph Neural Networks



AGENDA

- Graph Basics
- Graph Neural Networks - Key concepts
- GCNs on standard CORA dataset
- GraphSage
- Global Pooling Layers
- Graph Neural Network Design
- Graph Attention Networks
- Graph Transformers

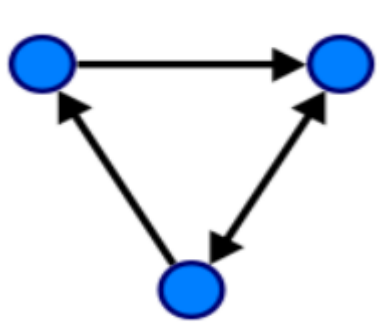
Graphs are far more complex



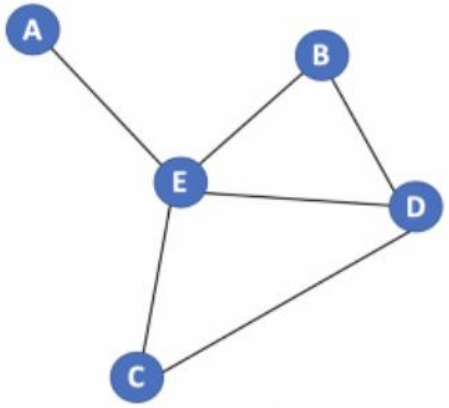
- Structure of real-world graphs can vary greatly - some graphs have many nodes with few connections between them, or vice versa.
- Arbitrary size and complex topological structure (no spatial locality like grids)
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Terminologies

- Graph is a set of vertices (nodes) and edges
  - **G = (V,E)**



- **Adjacency Matrix - NxN**



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

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

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

- **Identity Matrix**

Terminologies

- **Degree Matrix -**

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

New adjacency matrix $\tilde{A}$

2	0	0	0	0
0	3	0	0	0
0	0	3	0	0
0	0	0	4	0
0	0	0	0	5

New degree matrix $\tilde{D}$

1/2	0	0	0	0
0	1/3	0	0	0
0	0	1/3	0	0
0	0	0	1/4	0
0	0	0	0	1/5

$\tilde{D}^{-1}$

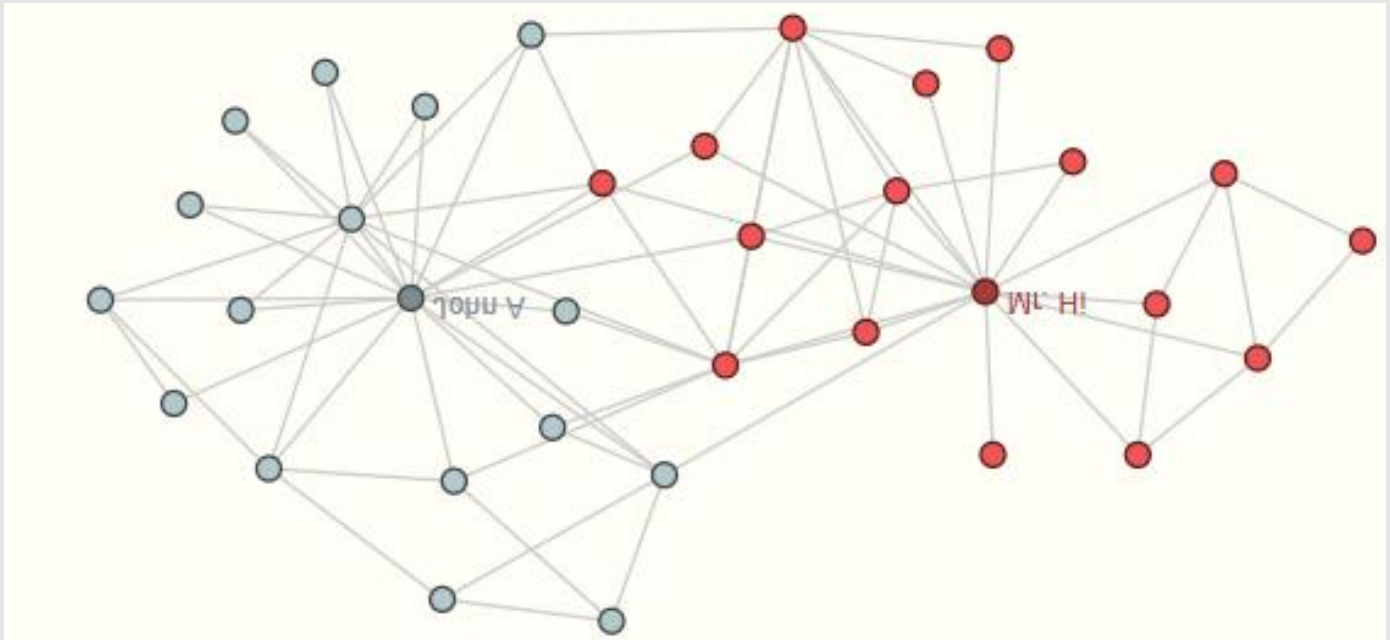
- **Node features** and **embeddings** - Storing some meaningful information about a node. [n_nodes, n_node_features]

A	-1.1	3.2	4.2
B	0.4	5.1	-1.2
C	1.2	1.3	2.1
D	1.4	-1.2	2.5
E	1.4	2.5	4.5

Feature vector X

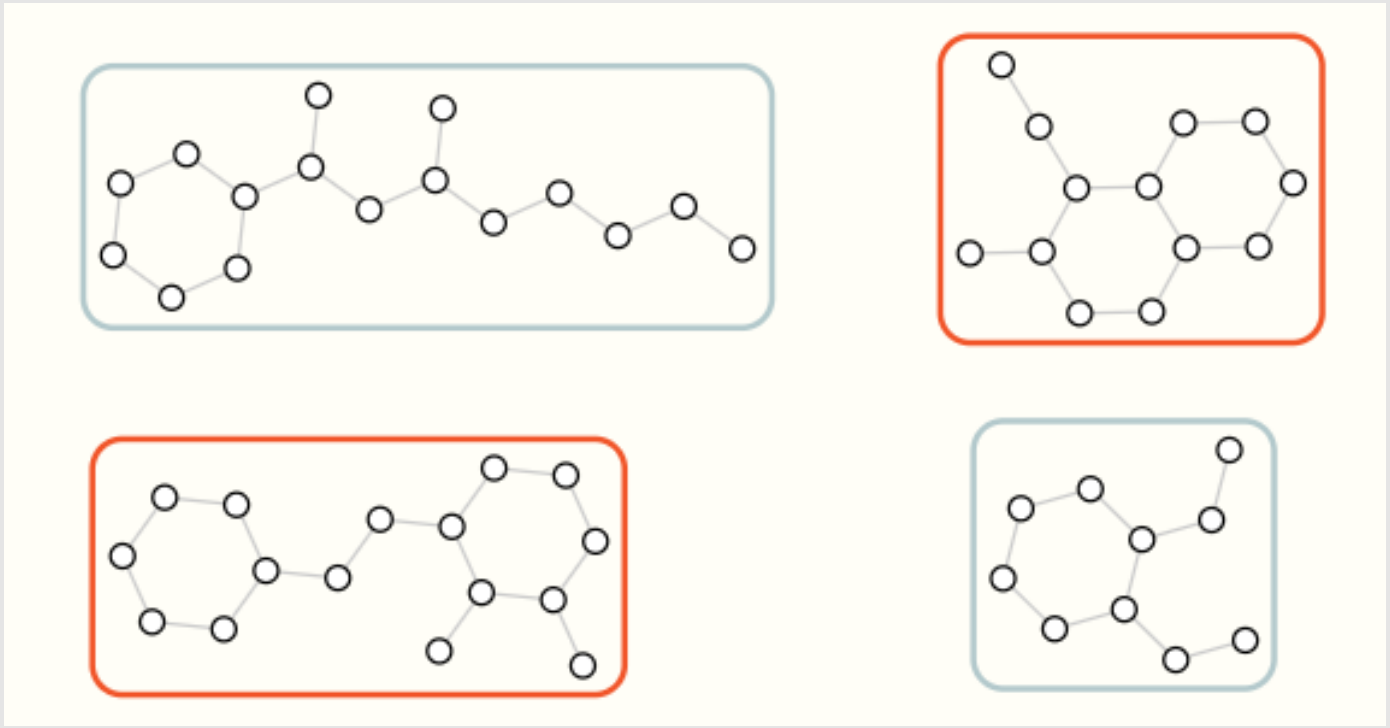
- **Edge features** and **embeddings** - Storing some meaningful information about the connections between the nodes. [n_edges, n_edge_features]

Node level classification



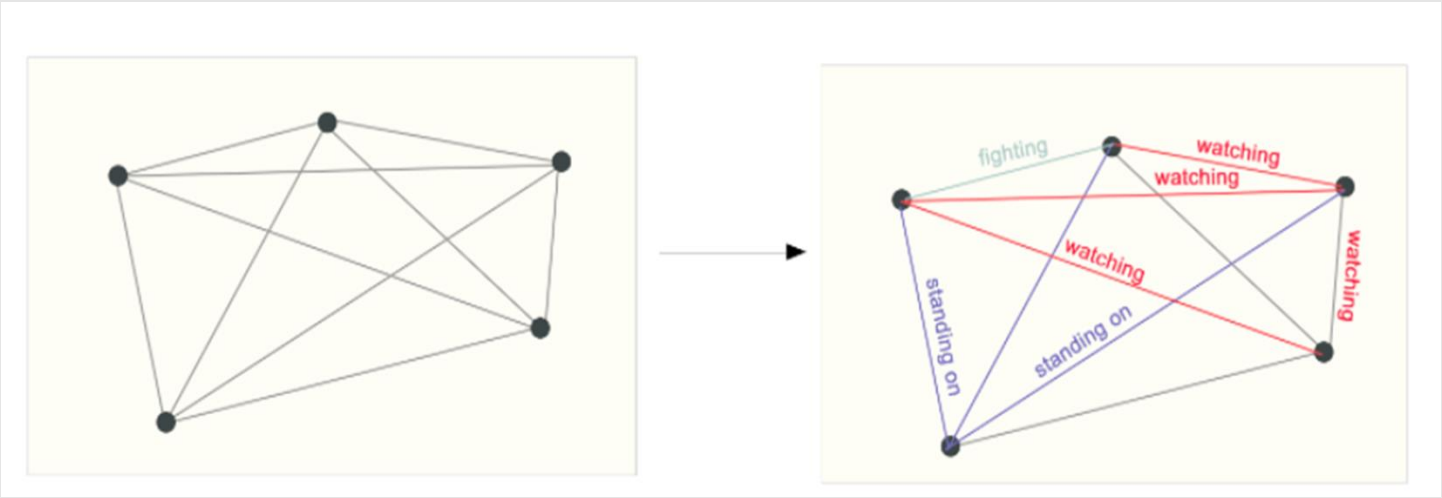
- Predict role of each node within a graph. Categorize online users, items, documents.
- Analogous to *image segmentation* - trying to label the role of each pixel in an image.
- Analogous to predicting parts of speech of each word in a sentence.

Graph level classification



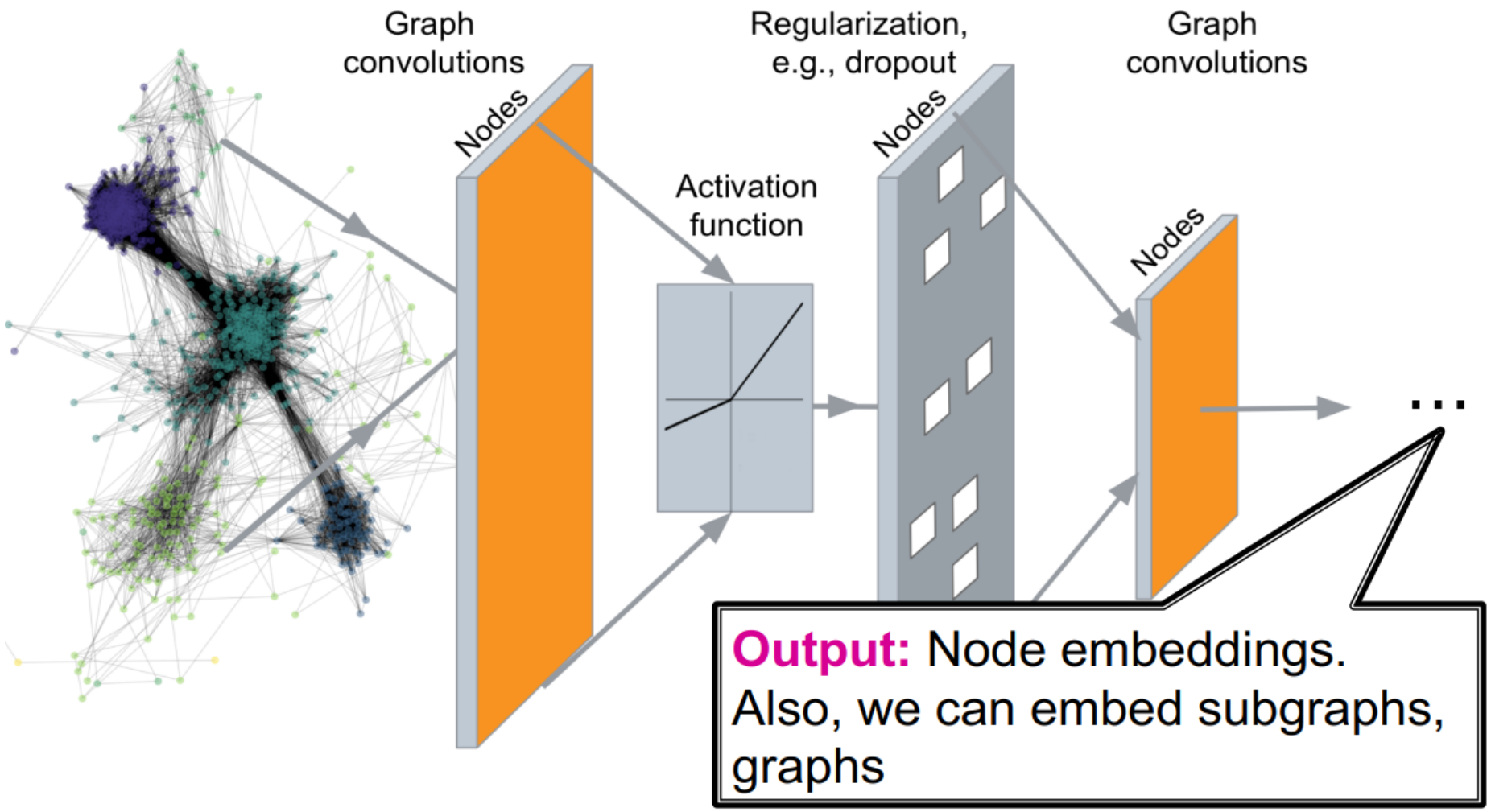
- Predict property of an entire graph.
- Analogous to *image classification* problems - CIFAR or MNIST datasets where we try to predict a label corresponding to a full image.
- Analogous to predicting a sentiment of a full sentence.
- Does the graph contain 2 rings ?
- Graphs representing chemical properties of a compound and answering whether it is an enzyme or not.
- At sub graph level - estimate time from one point to another.

Edge level classification



- Predict labels for each edge.
- Predict whether there are missing links.
- Recommender systems.
- Given a pair of drugs, predict side effects.

GNNs


Graph convolutions
Nodes
Activation function
Regularization, e.g., dropout
Nodes
Graph convolutions
Nodes
...
Output: Node embeddings.
Also, we can embed subgraphs, graphs

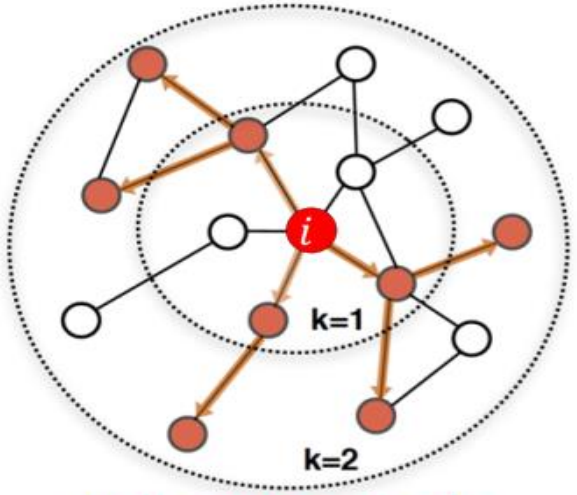
GNNs - Key Idea

- Assume we have a graph $\boldsymbol{G}$:
  V is the vertex set
  $\boldsymbol{A}$ is the adjacency matrix (assume binary)

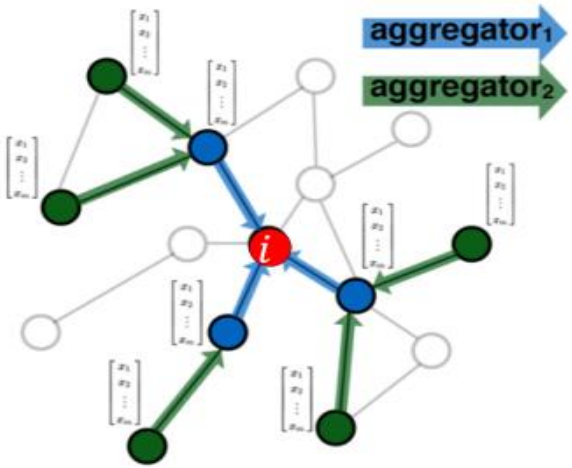
  $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ a matrix of node features. Can be a vector of constant 1 if no features available.

  v: a node in V; $N(v)$: the set of neighbors of v.

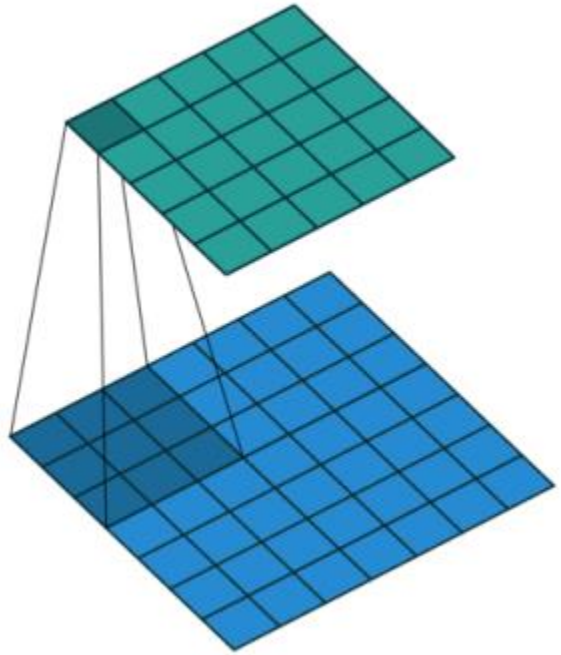
- Node's neighborhood defines a computation graph. Learn how to propagate information across the nodes to learn node embeddings.



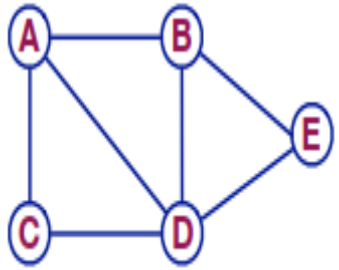
Determine node computation graph



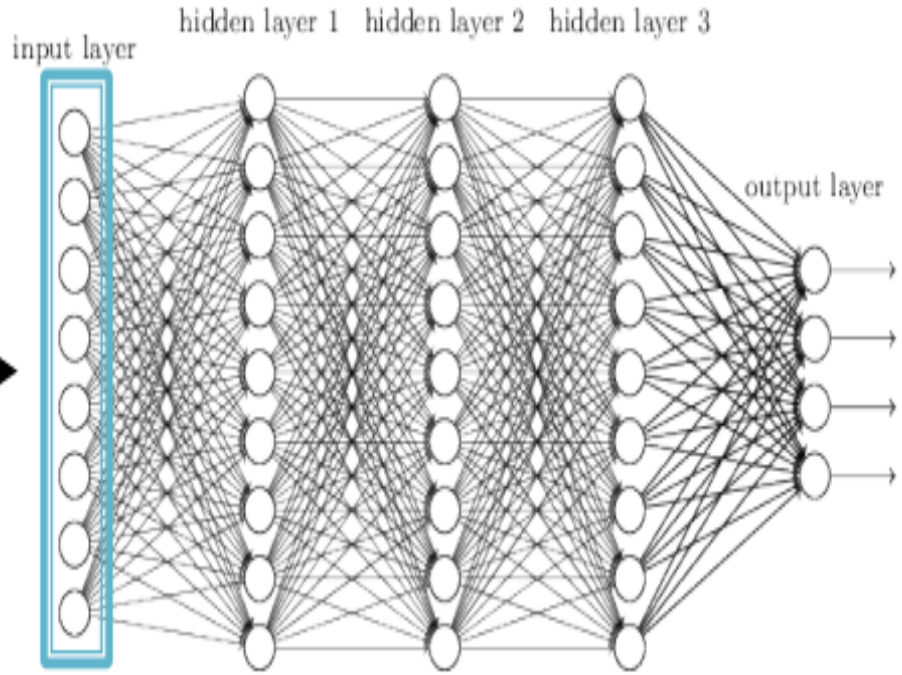
Propagate and transform information



A simple approach



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

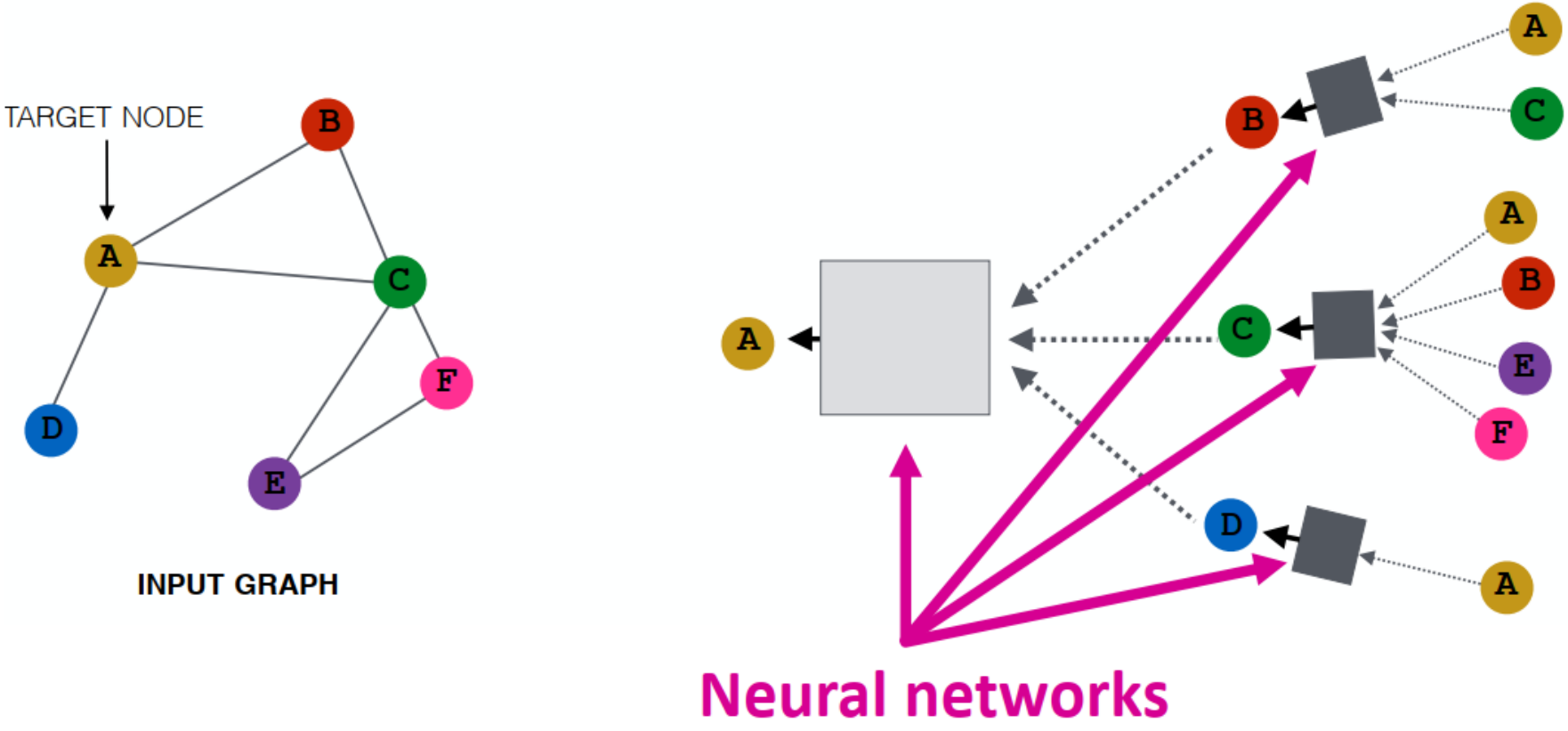


Limitations:

- Greater number of parameters
- No logical interactions
- Not applicable to graphs of different sizes

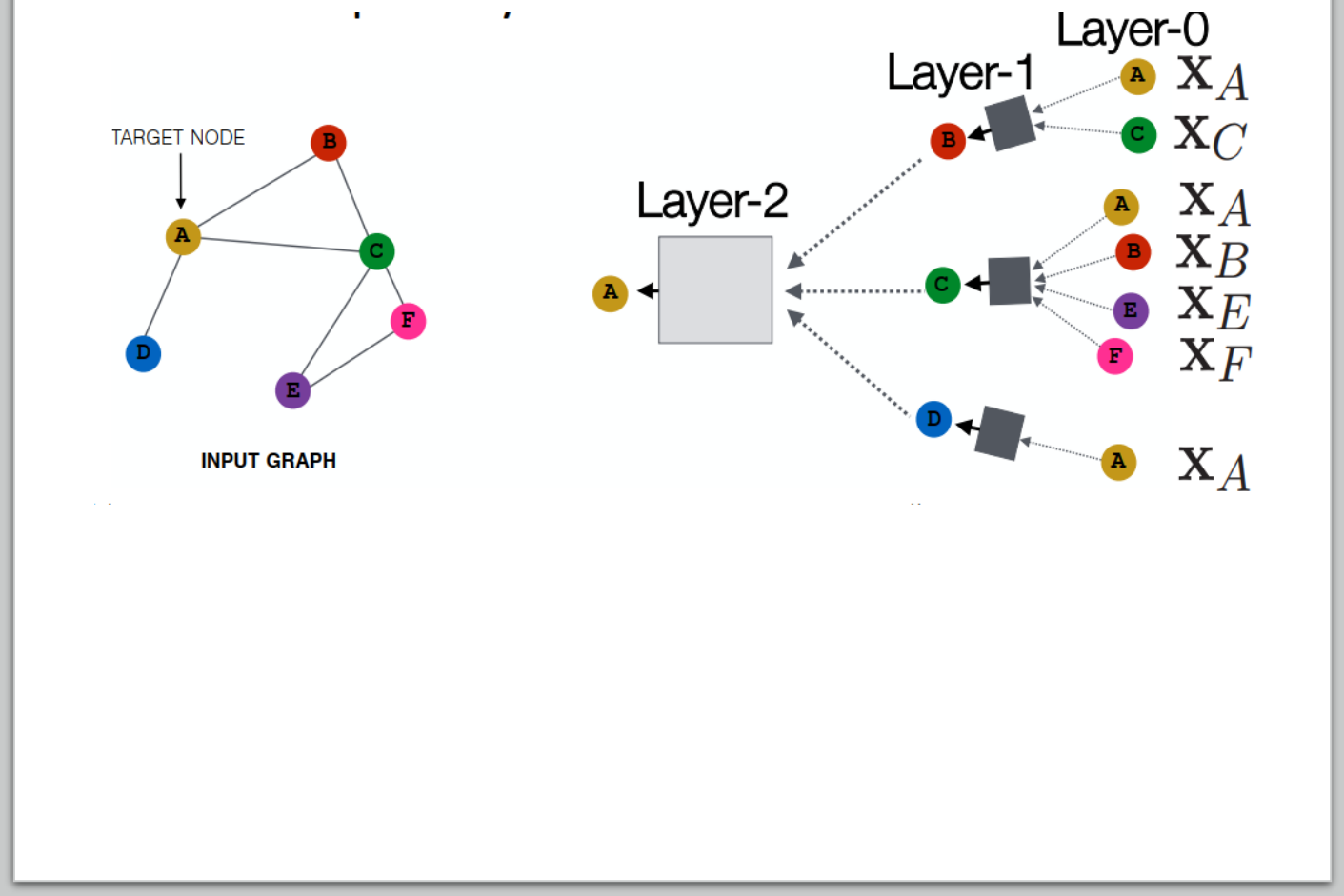
GNNs: Aggregate Neighbors

Generate node embeddings based on local network neighborhoods using neural networks

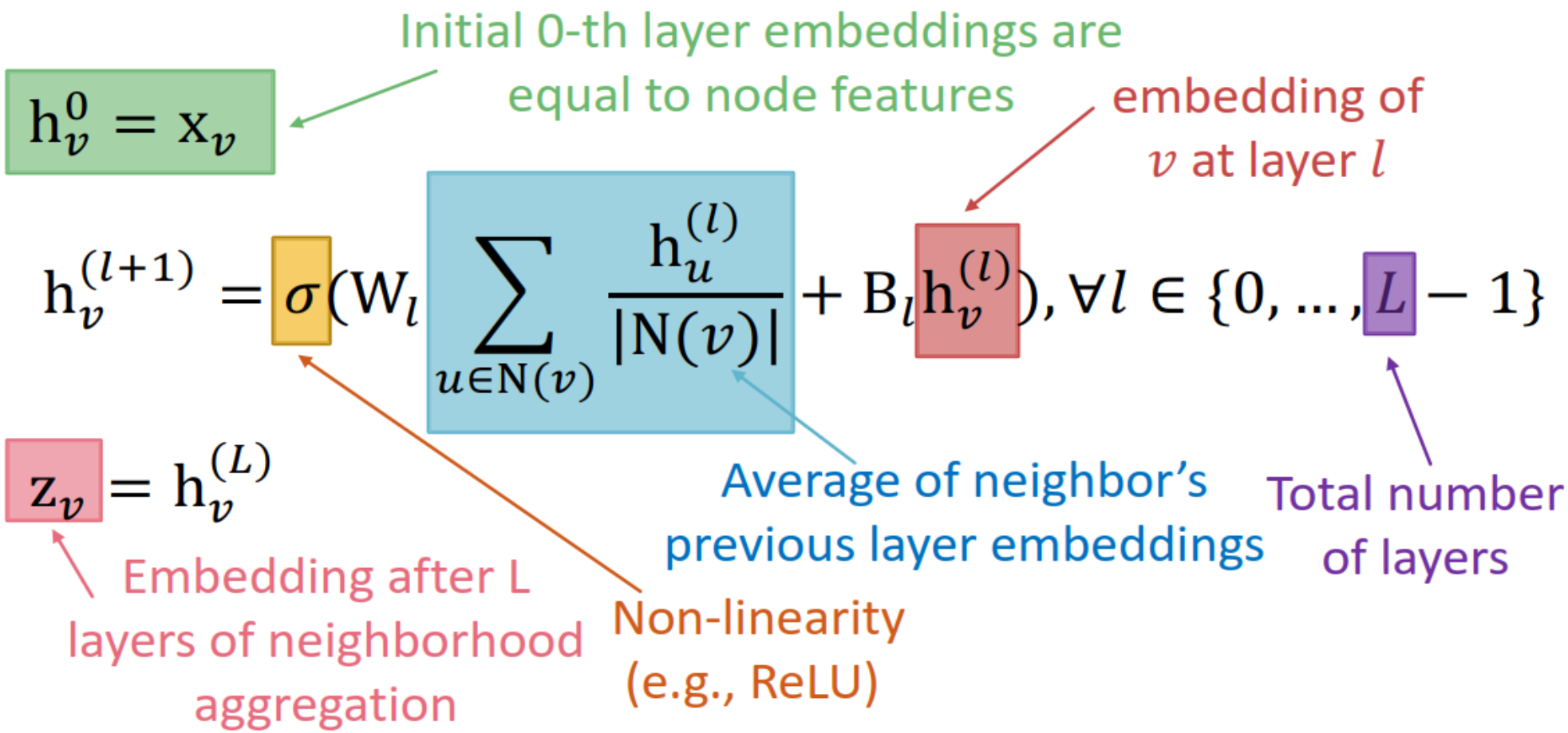


GNNs: Notion of depth

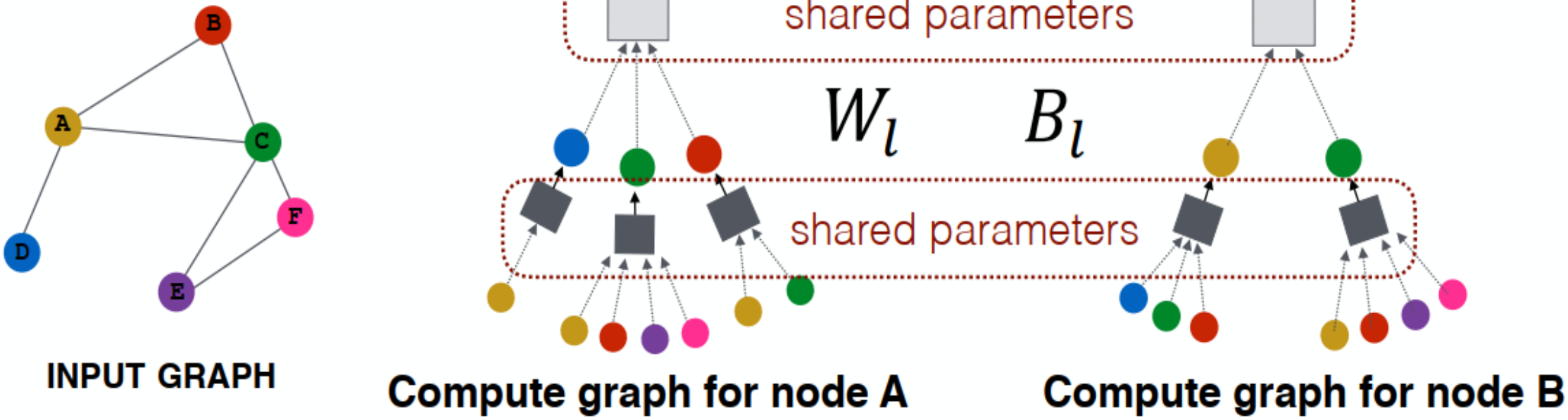
- Nodes have embeddings at each layer
- Layer-0 embedding of node u is its input feature, xu
- Layer-k embedding gets information from nodes that are K hops away



GCN Maths - Basic approach - Average neighborhood messages and apply transformations



Sharing of parameters



Number of parameters is sublinear

Matrix formulation

Let $H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$

Then: $\sum_{u \in N_v} h_u^{(l)} = A_{v,:} H^{(l)}$

Let D be diagonal matrix where

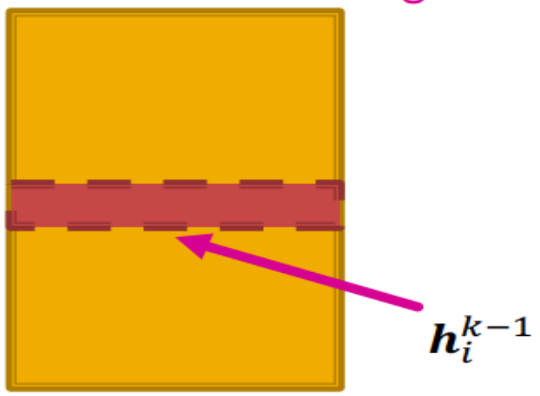
$D_{v,v} = \text{Deg}(v) = |N(v)|$

- The inverse of D: D^{-1} is also diagonal:

  $D_{v,v}^{-1} = 1/|N(v)|$

Therefore,

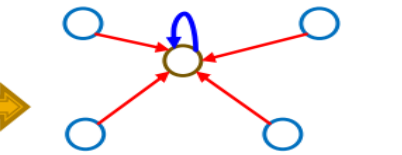
Matrix of hidden embeddings H^{k-1}



$$\sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|} \Rightarrow H^{(l+1)} = D^{-1} A H^{(l)}$$

$$H^{(l+1)} = \sigma(\tilde{A} H^{(l)} W_l^T + H^{(l)} B_l^T)$$

where $\tilde{A} = D^{-1} A$

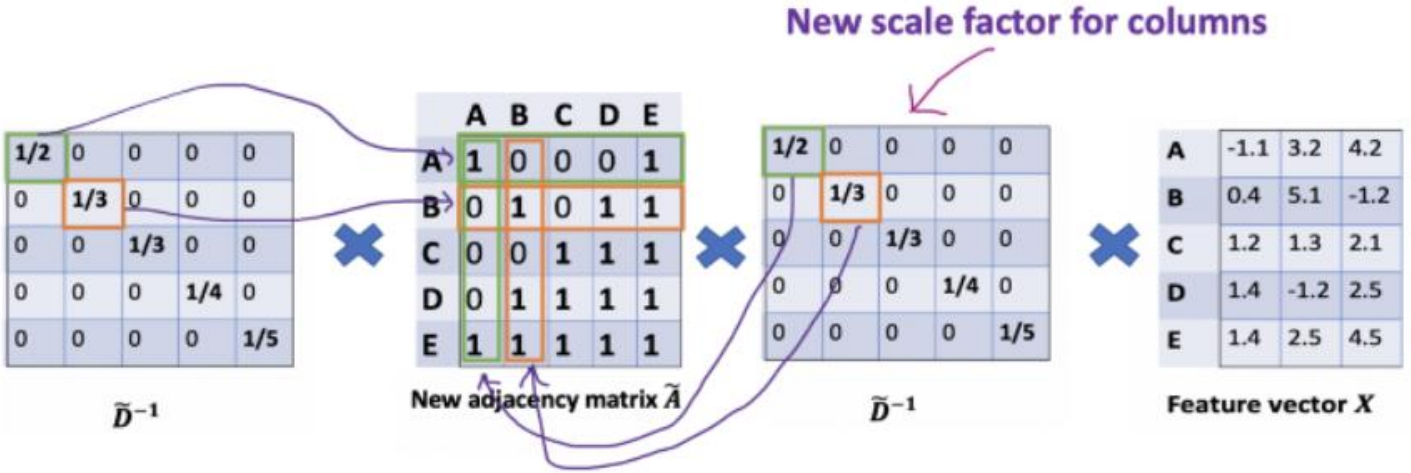


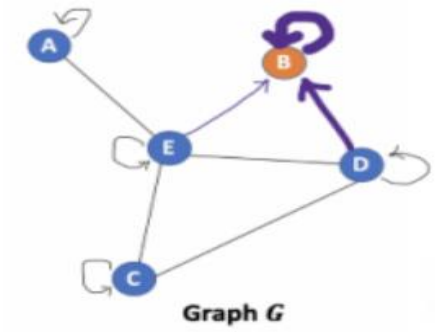
$H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$

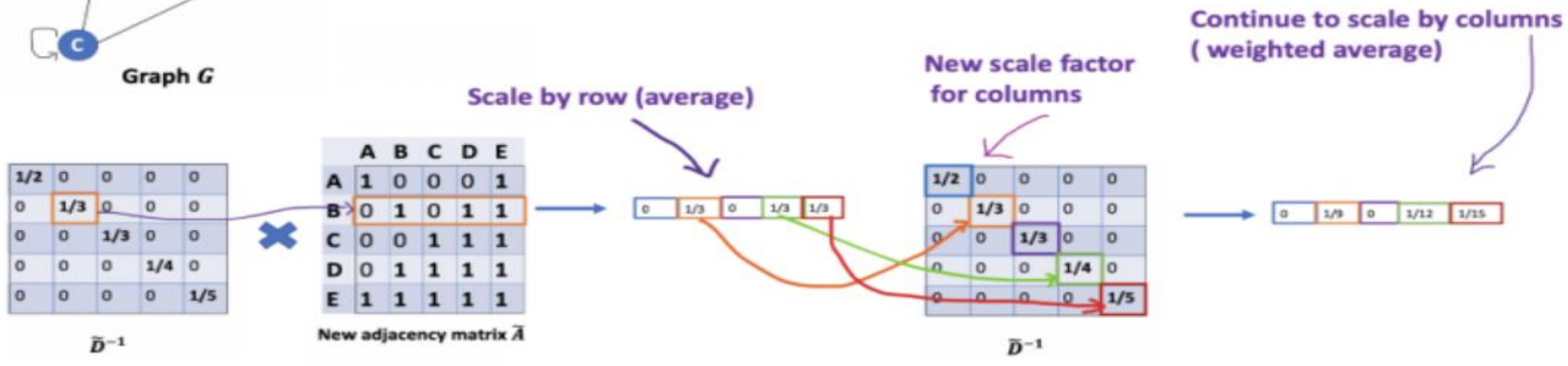
- Red: neighborhood aggregation
- Blue: self transformation

Matrix formulation

$$\tilde{D}^{-1}\tilde{A}\tilde{D}^{-1}X$$

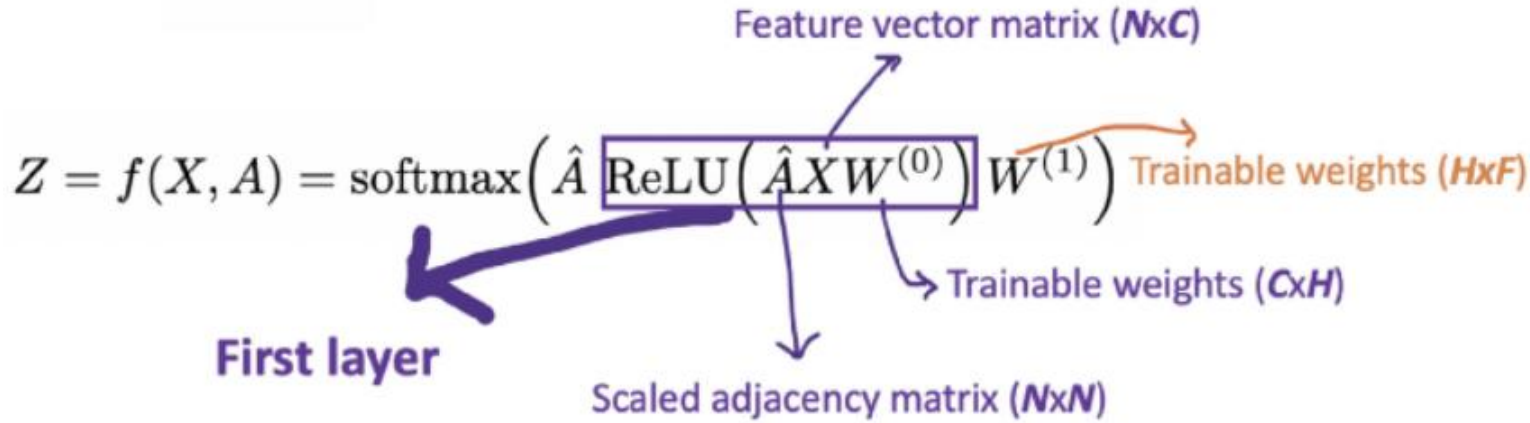




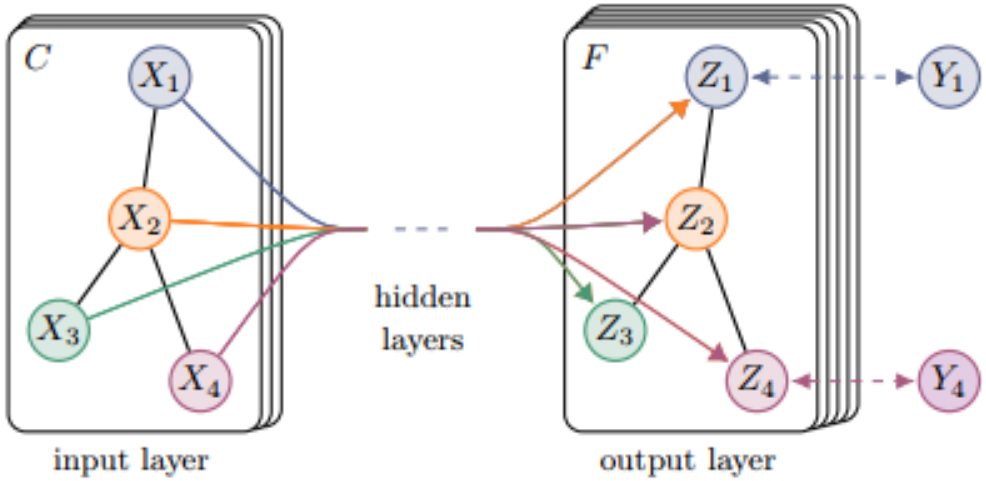


When aggregating feature at node B, we assign the biggest weight for node B itself (degree of 3), and the lowest weight for node E (degree of 5)

2-layer GCN trained on CORA dataset with 5% of labels (Kipf and Welling, 2017)



where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$



(a) Graph Convolutional Network

(b) Hidden layer activations

CORA dataset
Nodes – 2708 Papers

Edges – citations between the papers

Features – 1433 bag of words

Labels - 7

GraphSage Idea (Hamilton et al., 2017)

GNNs

$$h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)})$$

Graph Sage

Concatenate neighbor embedding and self embedding

$$h_v^{(l+1)} = \sigma([W_l \cdot \mathrm{AGG}\left(\left\{h_u^{(l)}, \forall u \in N(v)\right\}\right), B_l h_v^{(l)}])$$

Flexible aggregation function instead of mean

GraphSage Aggregation Variants

- **Mean -**

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$$

Aggregation — Message computation

- **Pool -**

$$\text{AGG} = \text{Mean}(\{\text{MLP}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

Aggregation — Message computation

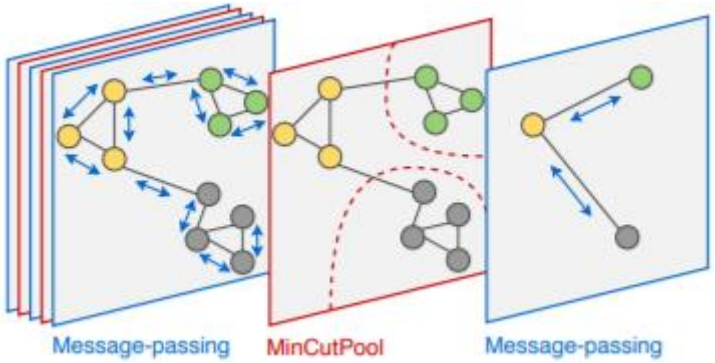
- **LSTM -**

$$\text{AGG} = \text{LSTM}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

Aggregation

Global Pooling Layers

- **GlobalAvgPool -** Pools a graph by average of its node features
  - Node features of shape ([batch], n_nodes, n_node_features)
  - Pooled node features of shape (batch, n_node_features)
- **GlobalMaxPool, GlobalSumPool**
- **Global Attention Pool**
  - Node features of shape ([batch], n_nodes, n_node_features)
  - Pooled node features of shape (batch, channels)
- **DiffPool** (Ying et al., 2018) – hierarchal soft clustering of nodes, **MinCutPool** (Bianchi et al., 2020)

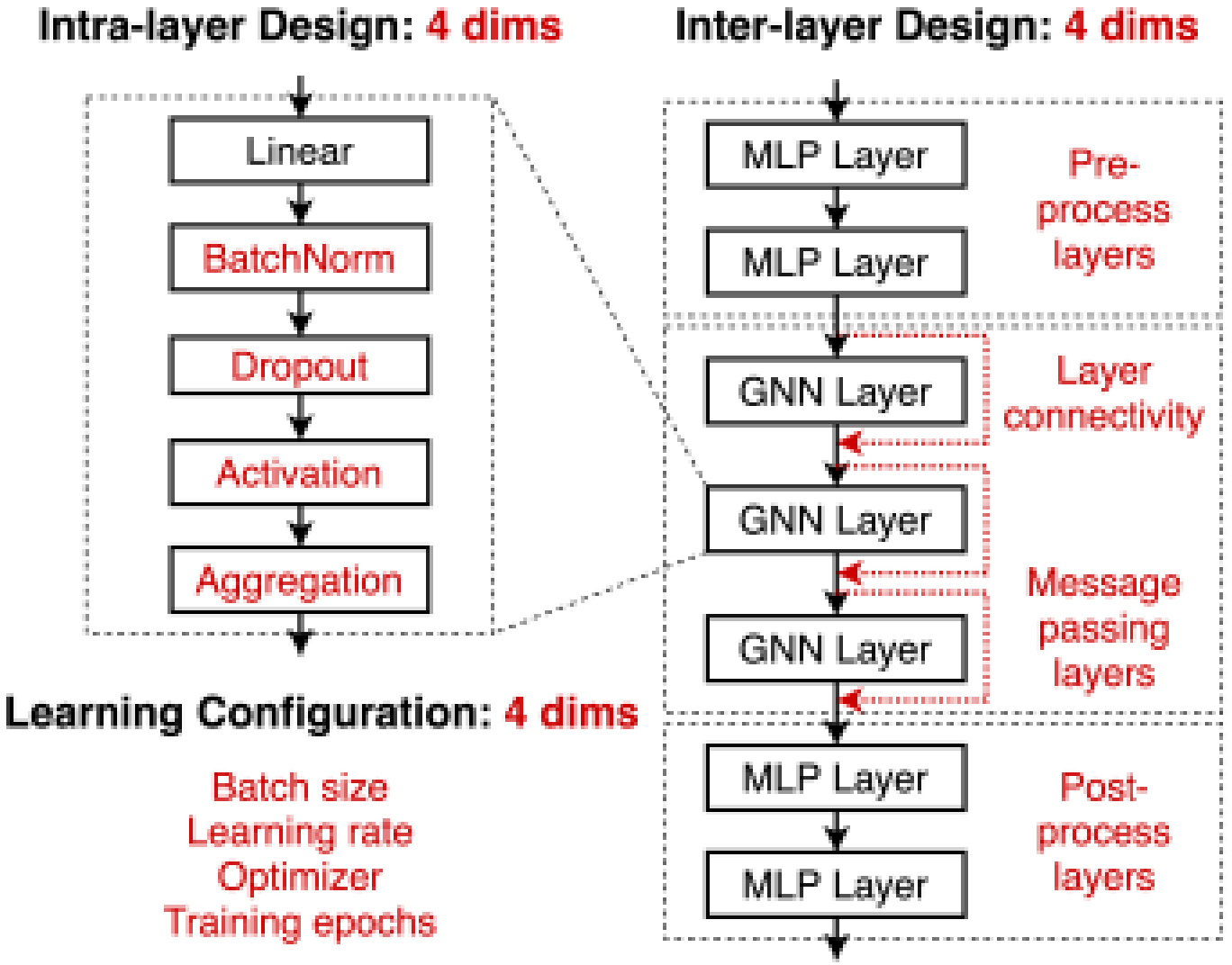


Transductive vs Inductive Learning

- **Transductive:** training / validation / test sets are on the same graph
  - The dataset consists of one graph
  - The entire graph can be observed in all dataset splits, we only split the ground truth labels
  - Only applicable to node / edge prediction tasks

- **Inductive**: training / validation / test sets are on different graphs
  - Dataset consists of multiple graphs
  - Each split can only observe the graph(s) within the split.
  - A successful model should generalize to unseen graphs
  - Applicable to node / edge / graph tasks

Graph Neural Network Design (You et al., 2020)



Key Takeaway

$$H^{[i+1]} = \sigma(W^{[i]} H^{[i]} A^{*})$$

Questions

Next we will cover advanced concepts of Graph Attention and Graph Transformer Networks

Graph Attention Networks (GAT)

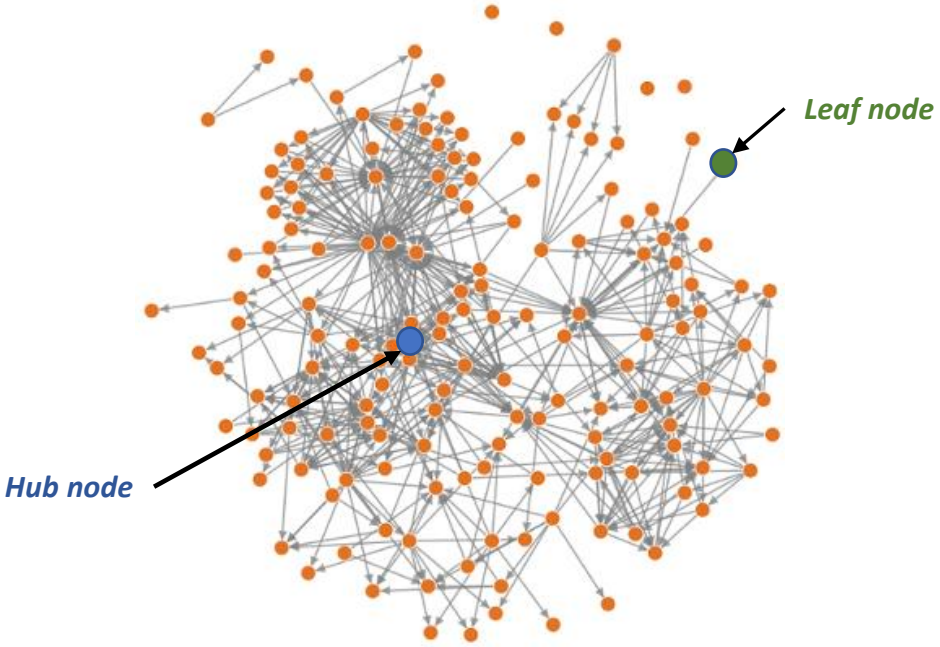
[Velickovic et al. ICLR 2018]

Graph Attention Networks (GAT) *[Velickovic et al. ICLR 2018]*

Input features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$

Let's recall the GCN neighbourhood aggregation:

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} Wx_u + b\right), \quad \forall v \in \mathcal{V}.$$



No restriction on the influence-neighbourhood!*

Graph Attention Networks (GAT) [Velickovic et al. ICLR 2018]

Input features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$

Attention/Importance of node j to node i: $\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) \qquad e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$

Inspiration: Use self-attention similar to that implemented in Transformers!

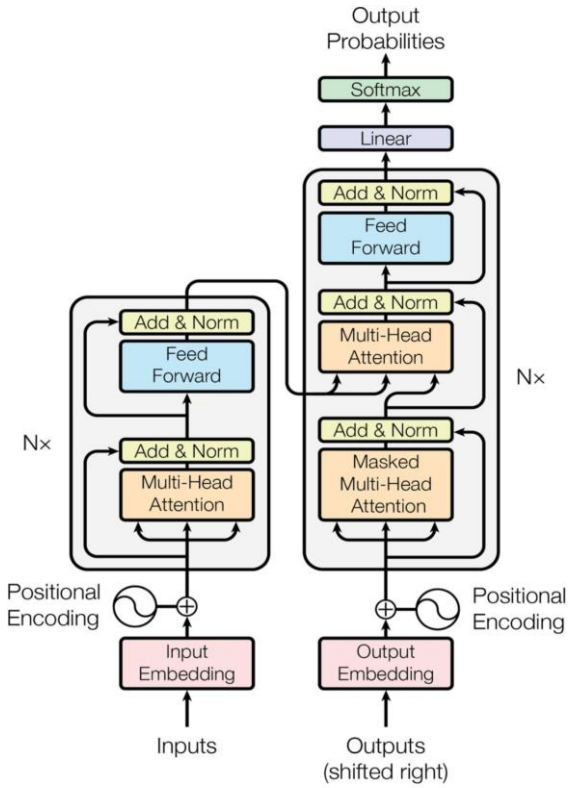
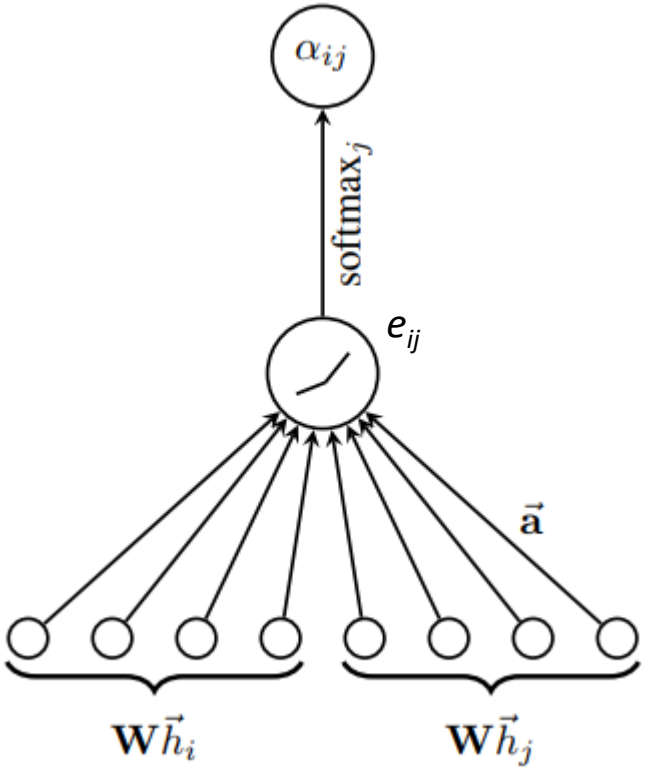


Figure 1: The Transformer - model architecture.

Graph Attention Networks (GAT) [Velickovic et al. ICLR 2018]

Input features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$



Attention-mechanism *a*, parametrized by a weight vector, applying a LeakyReLU activation

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\vec{h}_j\right)$$

Node i feature update

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j) \qquad \mathbf{W} \in \mathbb{R}^{F' \times F}$$

Non-normalized attention coefficients; that indicate the importance of node j's features to node i

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

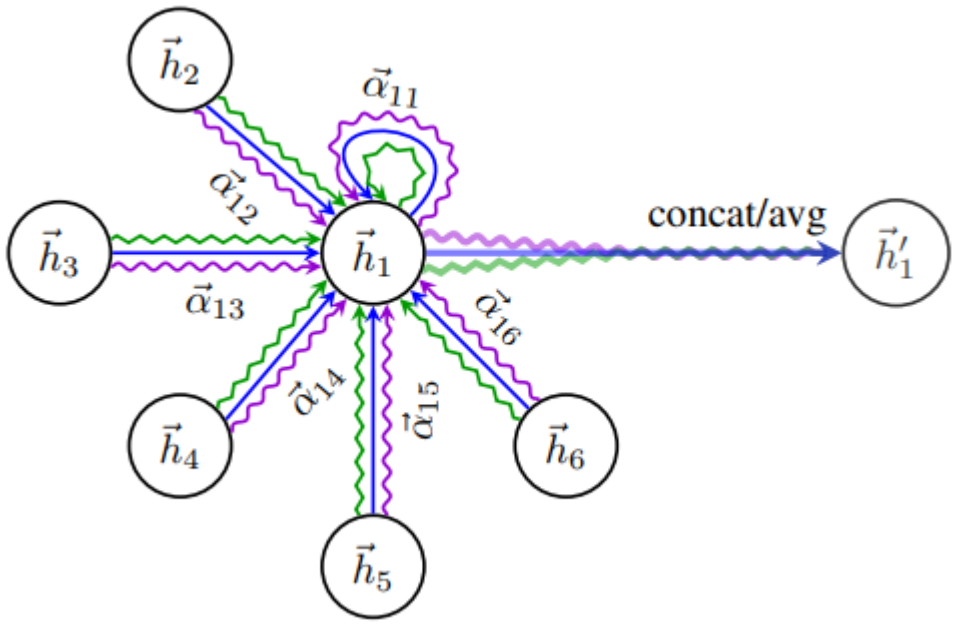
Normalized attention coefficients; to make coefficients easily comparable across different nodes

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

A single-layer feed-forward network as the attention-mechanism

Regularization in GATs *[Velickovic et al. ICLR 2018]*

Input features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$



Multi-head attention for node 1, with K=3 heads

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right)$$

Output feature without multi-head attention.

$$\vec{h}'_i = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

K independent attention-mechanisms employed. Resultant features are then concatenated. Final output **h'** will have **KF'** features for each node.

$$\vec{h}'_i = \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

For final layer, the K output features are averaged, followed by applying non-linearity.

GAT Results [Velickovic et al. ICLR 2018]

Inductive

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

Summary of results in terms of micro-averaged F1 scores on the Protein-protein interaction (PPI) dataset

Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Summary of results in terms of classification accuracies

Graph Transformers

[Dwivedi and Bresson, DLG-AAAI'21]

Graph Transformers *[Dwivedi and Bresson, DLG-AAAI'21]*

Transformers based neural-networks are one of the most successful architectures for representation-learning in NLP overcoming bottlenecks of RNNs caused by sequential-processing.

Can transformers be generalized to be applicable on graphs?

2 key aspects for generalizing transformers to graphs:

- Graph Sparsity
- Positional Encodings

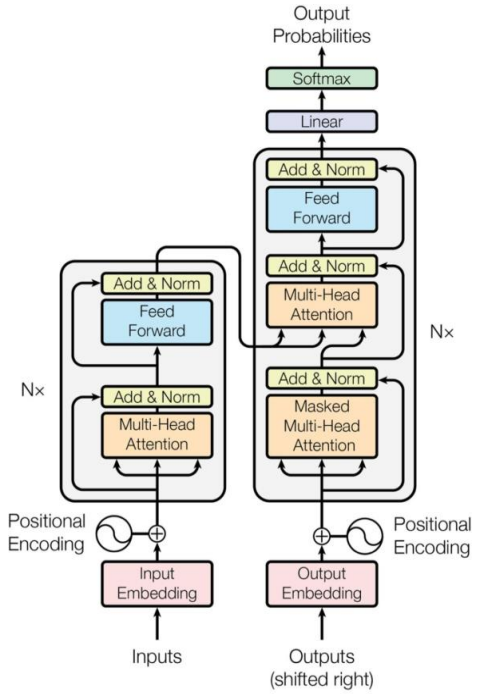


Figure 1: The Transformer - model architecture.

Graph Transformers *[Dwivedi and Bresson, DLG-AAAI'21]*

Graph Sparsity

Full attention vs Attention restricted to local neighbourhood *(image source)*

Graph Transformers *[Dwivedi and Bresson, DLG-AAAI'21]*

Positional Encodings

<Sequence>
This is a transformer

v/s

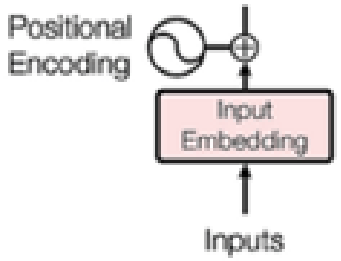
<Multi-set>
{This, is, a, transformer}

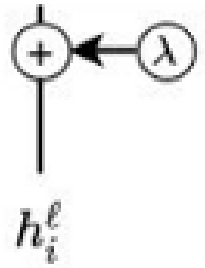
With and without positional info (image source)

If we use only attention on the nodes, the transformer becomes invariant to the ordering of the nodes.

Need some way to consider the positional info of the nodes!

Sinusoidal PE

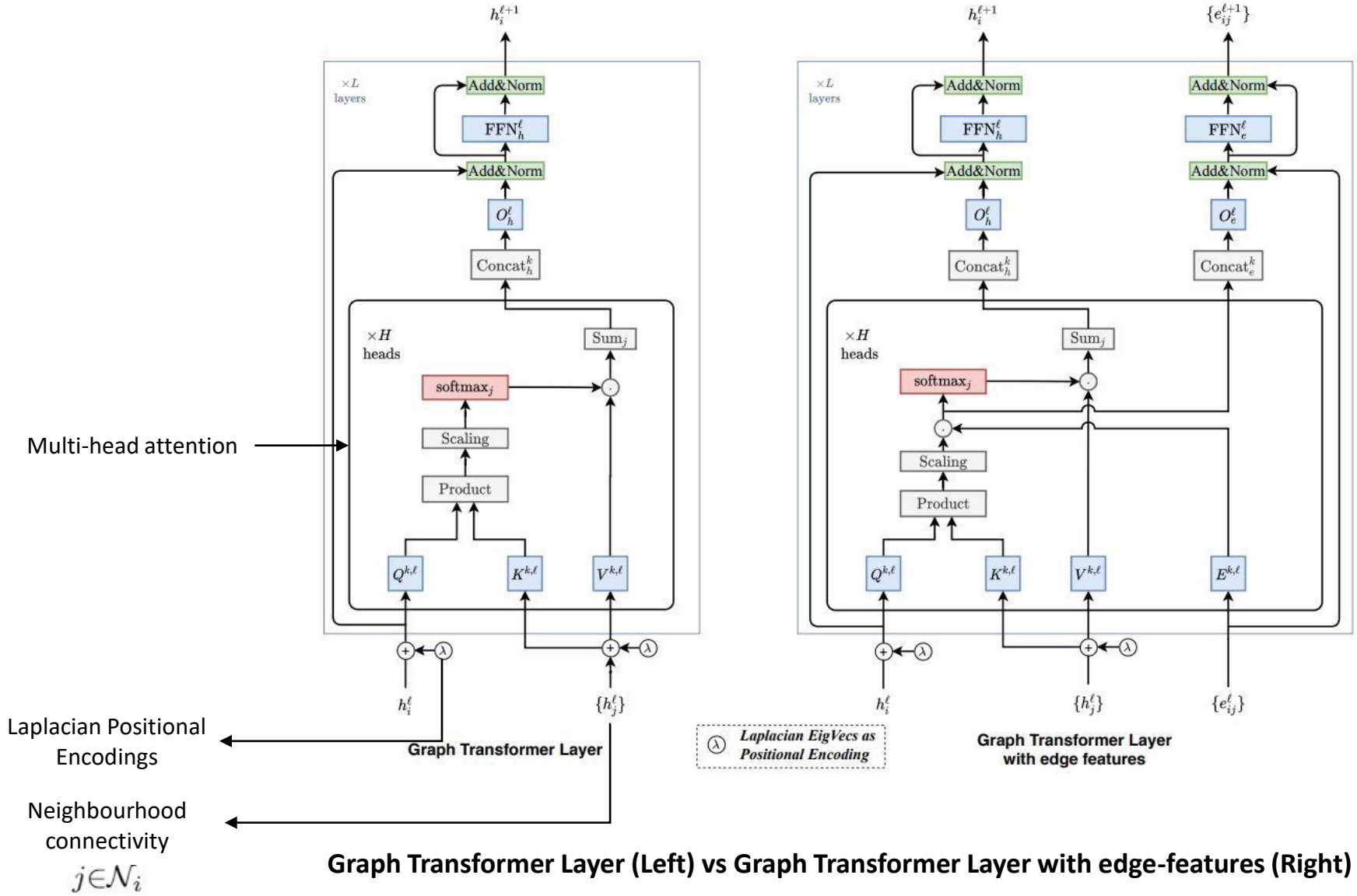




Laplacian Eigen-vectors

Positional Encodings (PEs) in original Transformer v/s proposed Graph Transformer *(image source)*

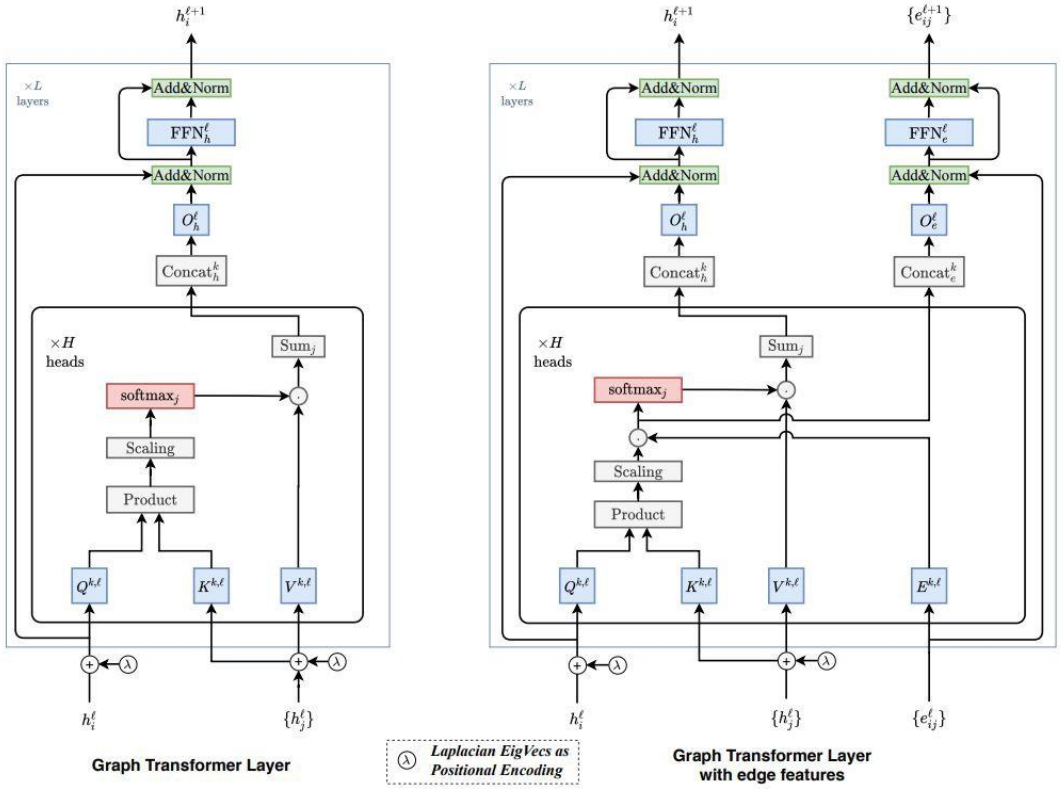
Graph Transformers *[Dwivedi and Bresson, DLG-AAAI'21]*



Graph Transformer Layer (Left) vs Graph Transformer Layer with edge-features (Right)

Graph Transformers *[Dwivedi and Bresson, DLG-AAAI'21]*

Key features:



Attention-mechanism only applied to the neighbourhood of the node

$$\hat{h}_i^{\ell+1} = O_h^\ell \Big\Vert_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^\ell \Big),$$

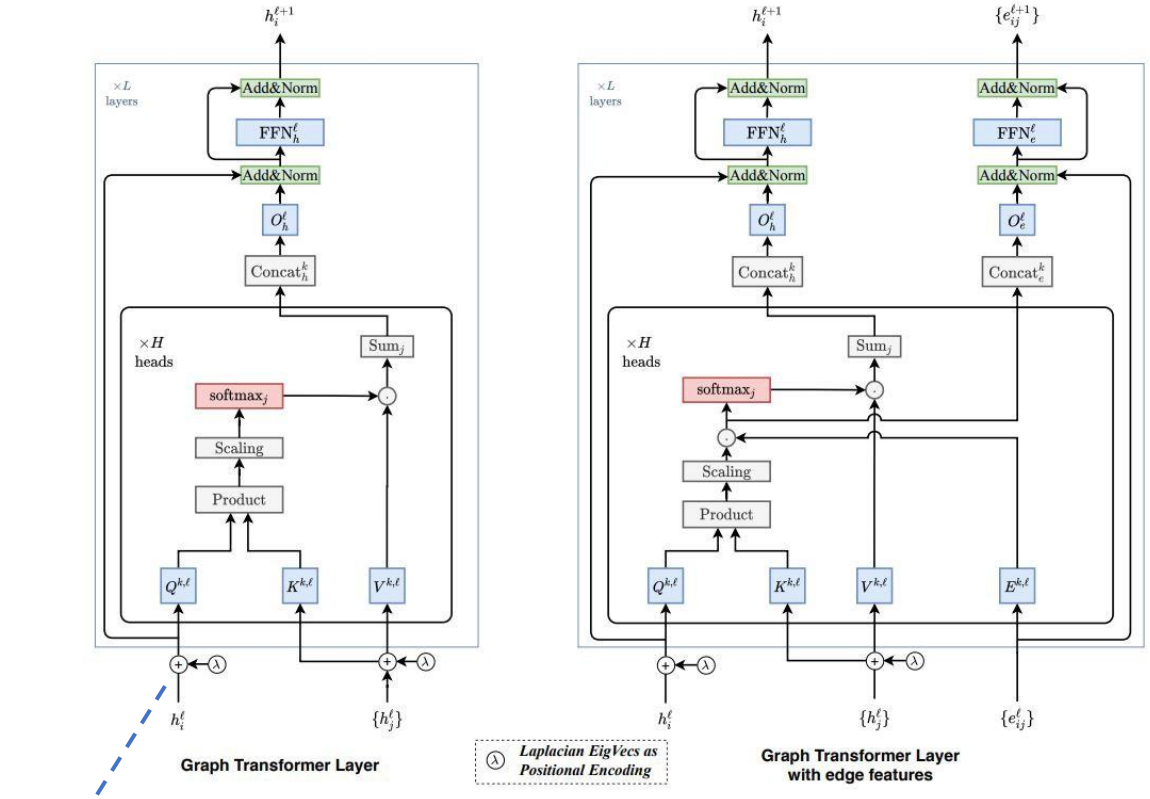
$$\text{where, } w_{ij}^{k,\ell} = \text{softmax}_j \Big(\frac{Q^{k,\ell} h_i^\ell \cdot K^{k,\ell} h_j^\ell}{\sqrt{d_k}} \Big),$$

Here, $Q^{k,\ell}, K^{k,\ell}, V^{k,\ell} \in \mathbb{R}^{d_k \times d}, O_h^\ell \in \mathbb{R}^{d \times d}$

, and no. of heads k = 1 to H

Graph Transformers *[Dwivedi and Bresson, DLG-AAAI'21]*

Key features:



Positional encoding represented by Laplacian Eigen-vectors

Laplacian PE's help encode distance-aware information. (i.e., nearby nodes have similar positional features and farther nodes have dissimilar positional features)

$$\Delta = \mathbf{I} - D^{-1/2} A D^{-1/2} = U^T \Lambda U$$

The k-smallest non-trivial eigenvectors of a node are used as its positional encoding and are denoted by λ_i for node i.

Summary

GCN

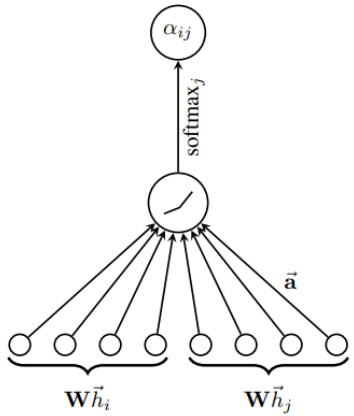
$$h_v^{(l+1)} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)})$$

; where W_l and B_l are the shared-parameters

GAT

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right)$$

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



GraphSage

Concatenate neighbor embedding and self embedding

$$h_v^{(l+1)} = \sigma([W_l \cdot \text{AGG}\left(\left\{h_u^{(l)}, \forall u \in N(v)\right\}\right), B_l h_v^{(l)}])$$

Flexible aggregation function instead of mean

We saw the Mean, Pool and LSTM aggregation-variants

Graph Transformers

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\|_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big),$$

$$\text{where, } w_{ij}^{k,\ell} = \text{softmax}_j\Big(\frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \Big),$$

2 key aspects for generalizing transformers to graphs:

- Graph Sparsity: Attention restricted to local neighbourhood
- Positional Encodings: Using Laplacian eigen-vectors as node positional encodings