ETH zürich

BIOMEDICAL INFORMATICS

Intro to Graph Neural Networks

Prof. Dr. Gunnar Rätsch, Dr. Rita Kuznetsova
Institute for Machine Learning, Department of Computer Science

Outline for Today

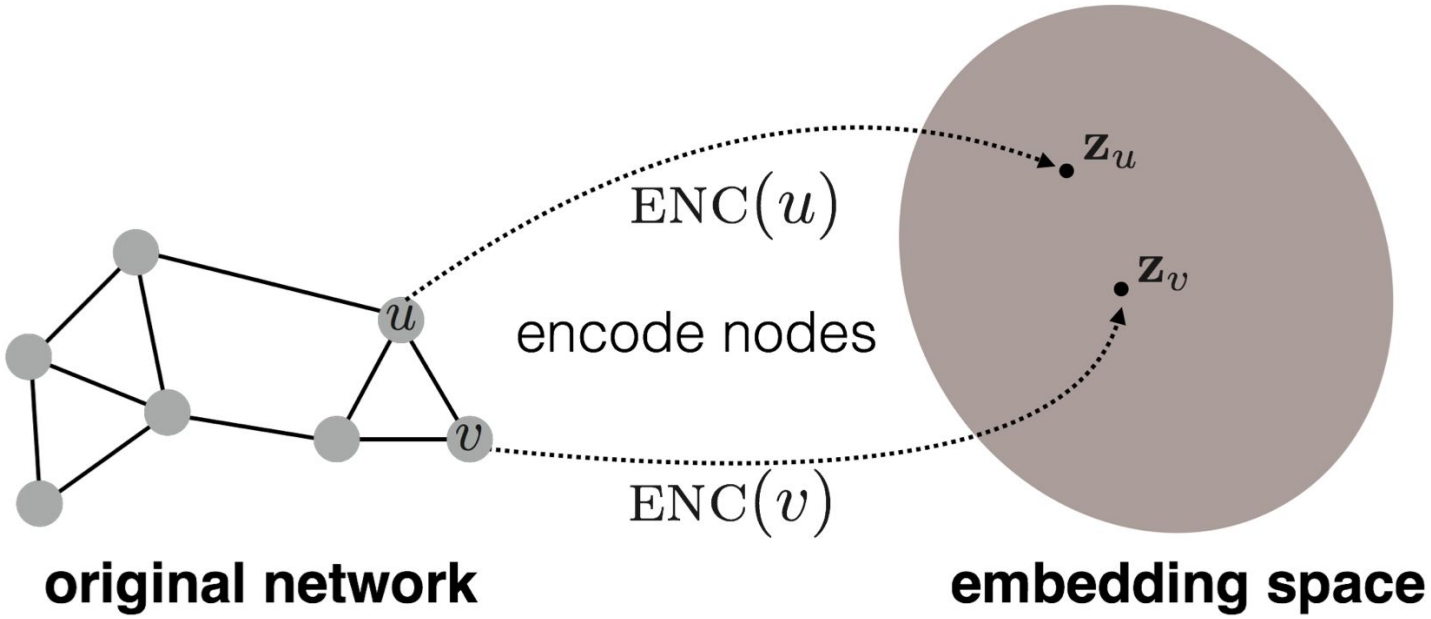
1. Recap: Node Embeddings
2. Goals & Motivation
3. Deep Learning for Graphs
    a. Why MLP does not work
    b. Permutation invariance/Equivariance
    c. Convolutional Neural Network Ideas
4. Graph Neural Networks (GNNs)
    a. GNN Layer
    b. Zoo of GNN Architectures
    c. Simple Setup
    d. GNN vs node2vec
    e. GNN vs Other DL Models
    f. Problems & Limitations
5. Summary & Take-Home Messages

- If you decide to drop the course, please unregister!
- Please, select the paper topic and project teammates - the deadline is tomorrow!
  - Project work / presentation is mandatory for passing a course.

Recap: Node Embeddings

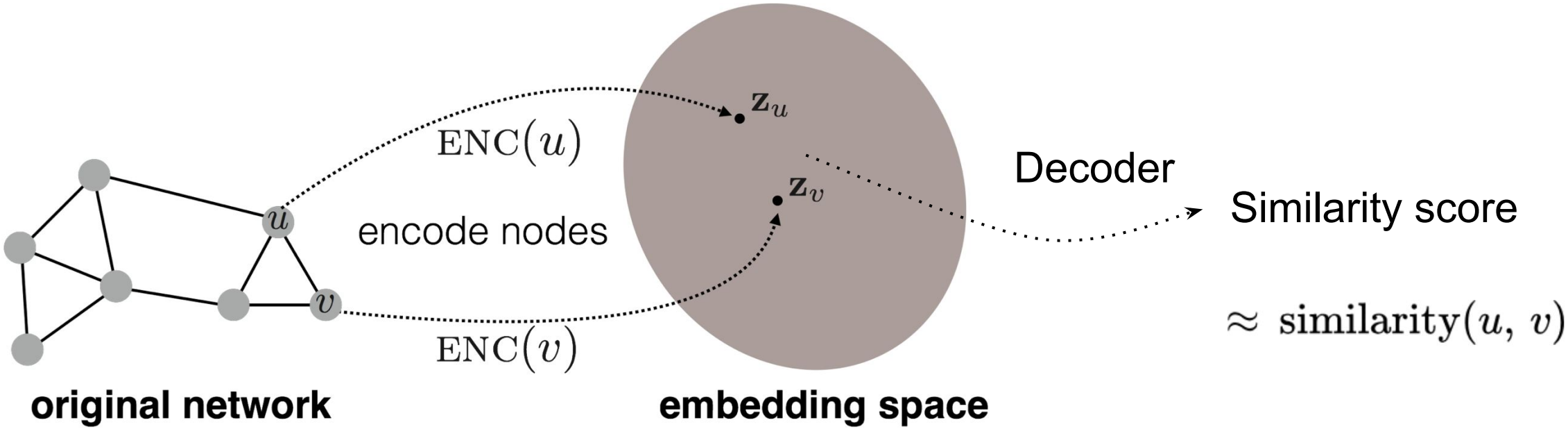
Recap: Node Embeddings

The goal is to encode nodes as low-dimensional vectors, so that similarity in the latent space corresponds to relationships in the graph.



Credit: [1,2]

Recap: Node Embeddings



Credit: [1,2]

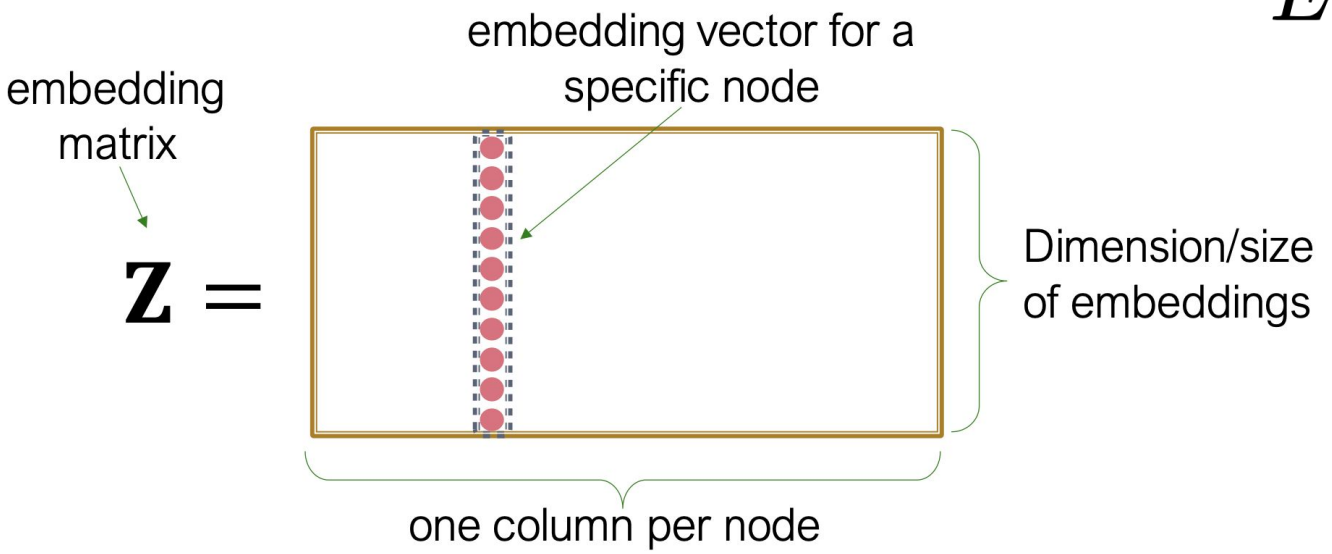
Recap: An Encoder-Decoder Perspective

Encoder maps nodes $v \in V$ to vector embeddings:

$$ENC(v) = z_v \in \mathbb{R}^d$$

Simplest encoding approach: Shallow Encoding

$$ENC(v) = z_v = \underbrace{Z}_{\in \mathbb{R}^{d \times |V|}} \cdot \underbrace{v}_{\in \mathbb{I}^{|\mathbb{V}|}}$$



Recap: An Encoder-Decoder Perspective

Decoder (or pairwise decoder) maps from embeddings to the similarity score:

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+$$

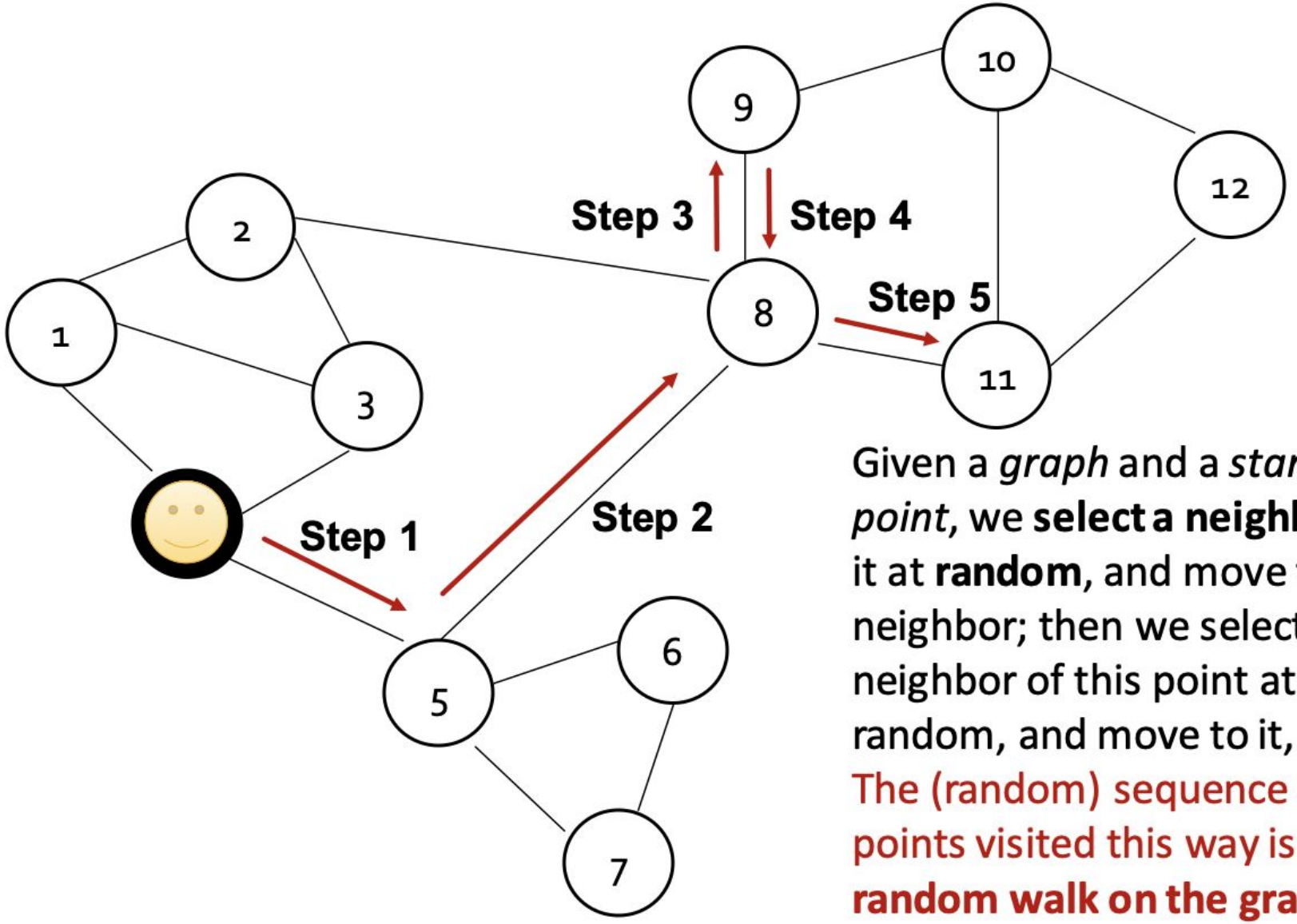
Applying the **decoder** to a pair of embeddings:

$$DEC(ENC(u), ENC(v)) = DEC(z_u, z_v) \approx \text{similarity}(u, v)$$

Similarity function is a graph-based similarity measure between nodes.

Credit: [1,2]

Random Walk Approaches



Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

Credit: [1,2,3,4]

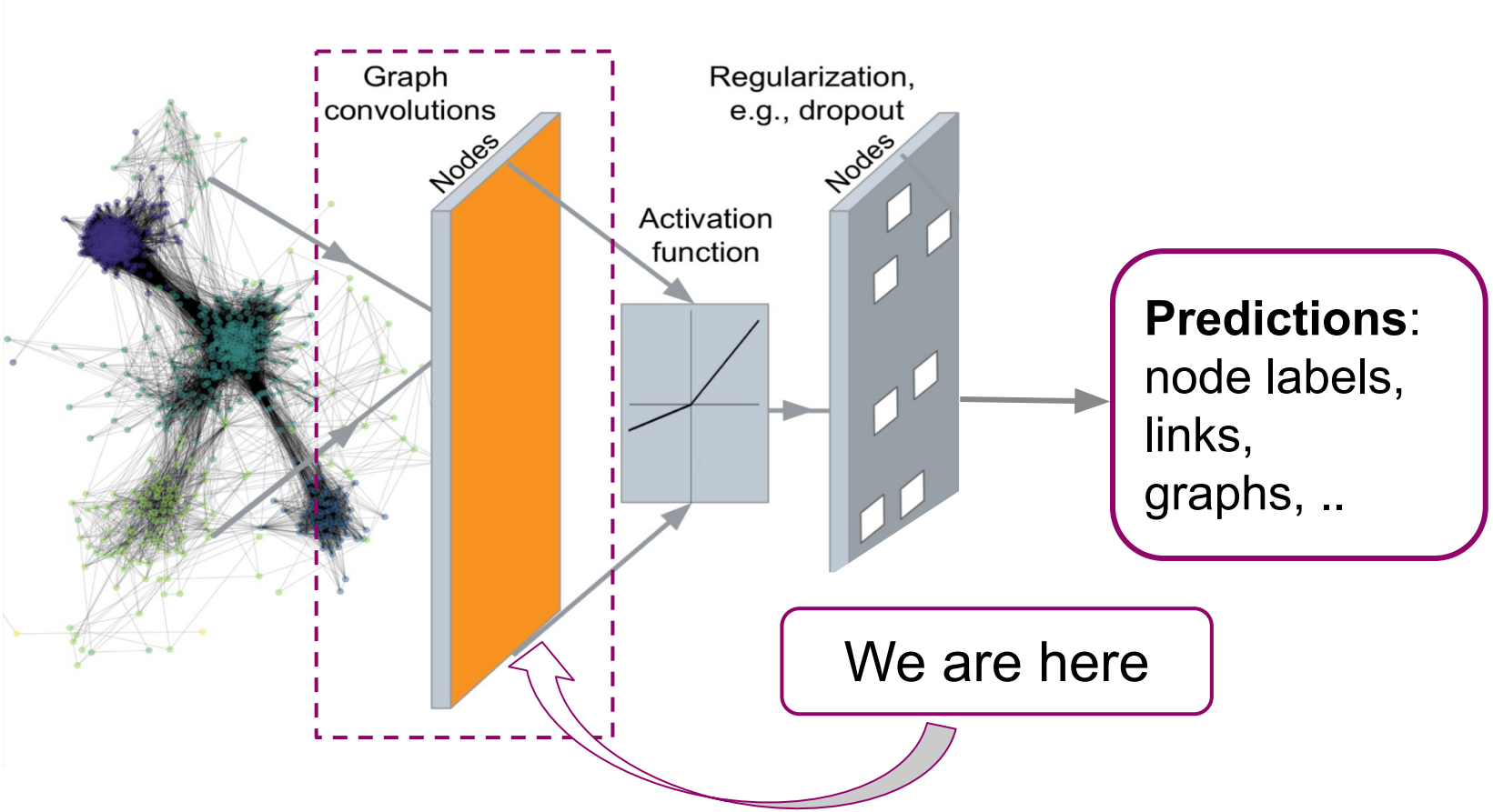
Recap: Limitations on node2vec-like Approaches

1. Can not obtain embeddings for unseen nodes;
2. Can not capture well the structural similarity;
3. Can not utilize directly node, edge and graph features;
4. No sharing of parameters between nodes;
5. Sensitive to the walk length, p/q choice.

Goals & Motivation

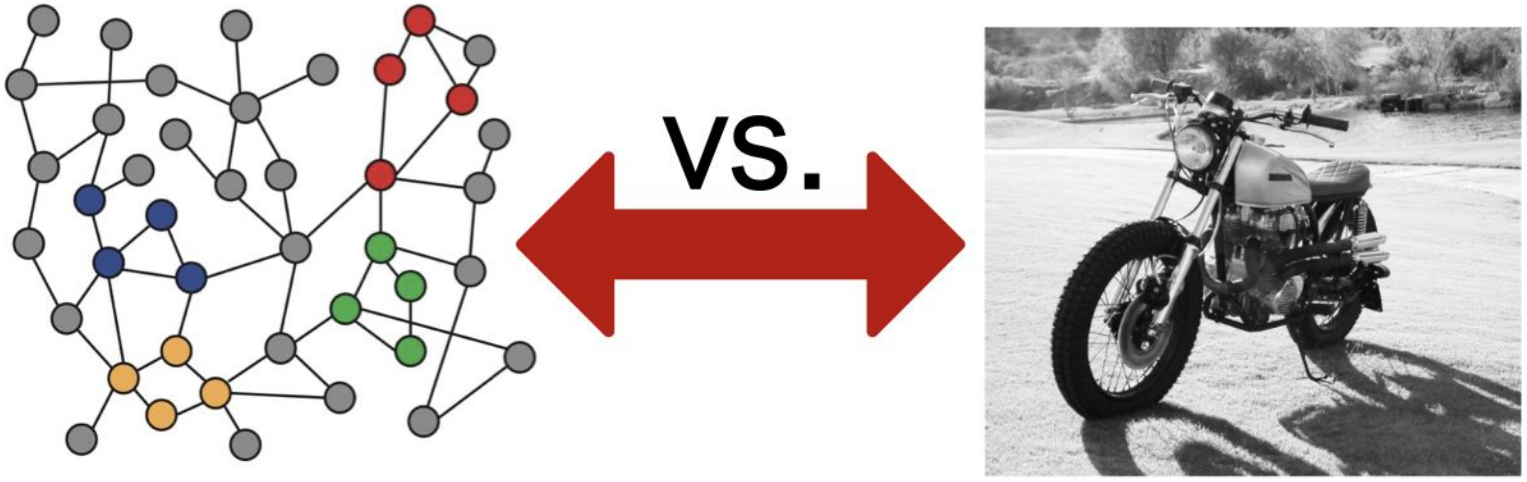
Goal for Today: Deep Encoders

How to generate representations of nodes that actually depend on the structure of the graph, as well as any feature information we might have.



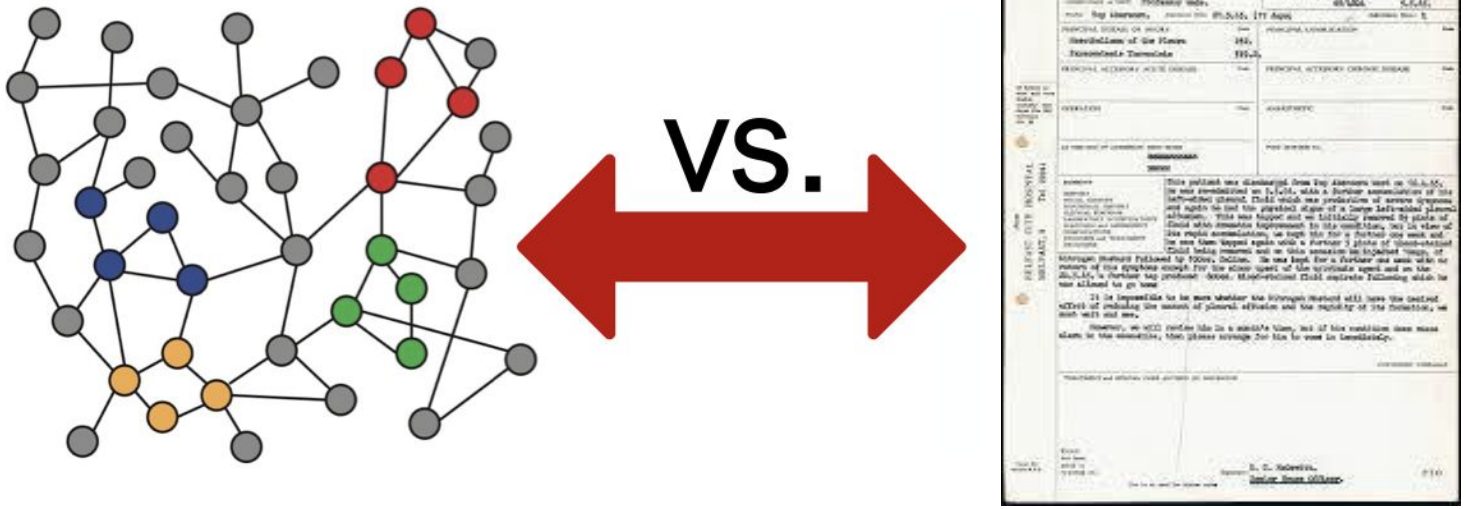
Credit: [1]

Why it is hard to process Graphs/Network structure



Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)

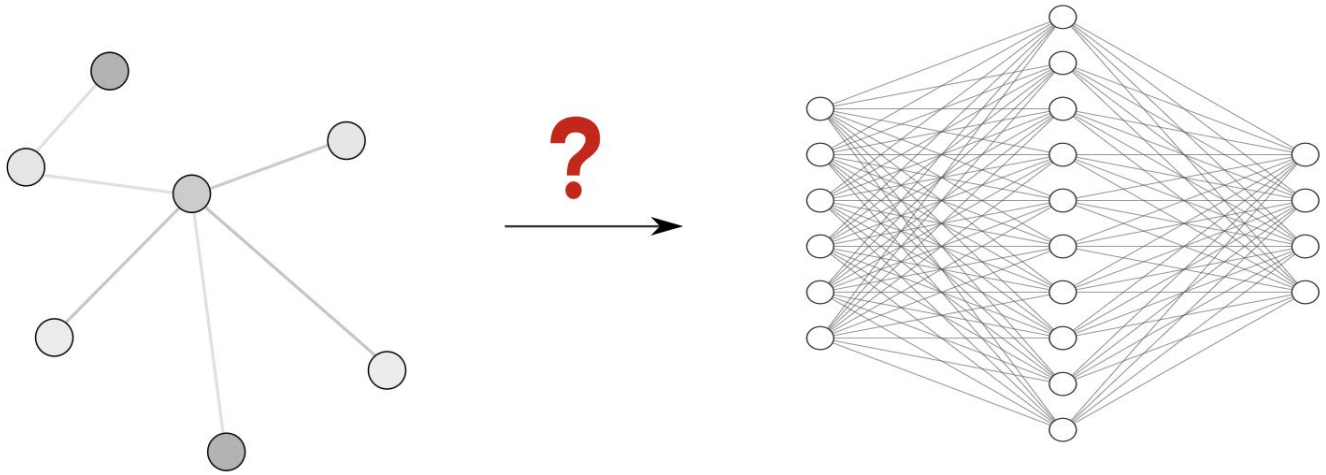
Why it is hard to process Graphs/Network structure



No fixed node ordering or reference point

Credit: [1]

Deep Learning for Graphs



1. Ideas behind the Graph Neural Networks.
2. What is neighborhood and how to define the computation graph.
3. Message aggregation strategy from every node to the resulting embedding.
4. Various GNN architectures with examples.
5. Simple training setup.
6. Connections with the other DL models.

Deep Learning for Graphs

Notations

Given graph $G = (V, E)$:

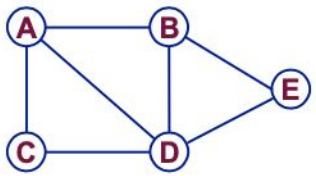
- V – vertices (nodes),
- E – edges,
- A – adjacency matrix (assume binary),
- $X \in \mathbb{R}^{|V| \times d}$ – matrix of the node features (d – feature dimensionality),
- $v \in V$ – the node, $\mathcal{N}(v)$ – neighbours of node v.

Node's Features:

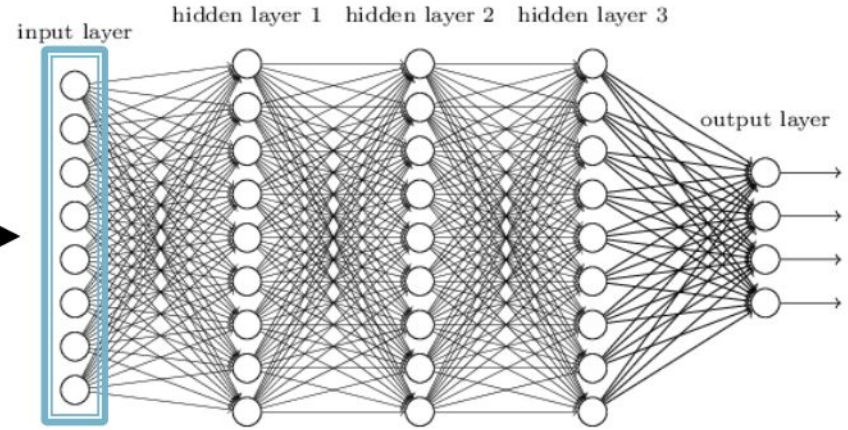
- Social network:
  - **Node**: individual person
  - **Node features**: Age, gender, occupation, location, number of followers, number of posts, etc.
- Citation network:
  - **Node**: academic paper
  - **Node features**: number of authors, publication year, bag-of-words of abstracts, etc.

A Naive Approach

Join adjacency matrix and node features and feed them into a Neural Network.



	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



?

Not applicable to graphs of different sizes → we can not use it!

Permutation Invariance & Equivariance

Graph does not have a canonical order of the nodes →
We need to make sure that permuting the nodes and edges, does not change the outputs.

We recover the following rules a GNN must satisfy:

For any graph function $g : \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times d} \to \mathbb{R}^d$, g is permutation invariant if

$$g(A, X) = g\left(PAP^\top, PX\right)$$

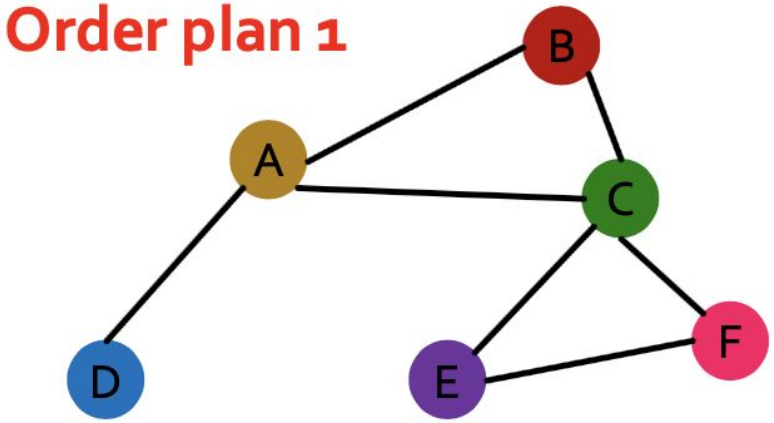
for any permutation matrix P.

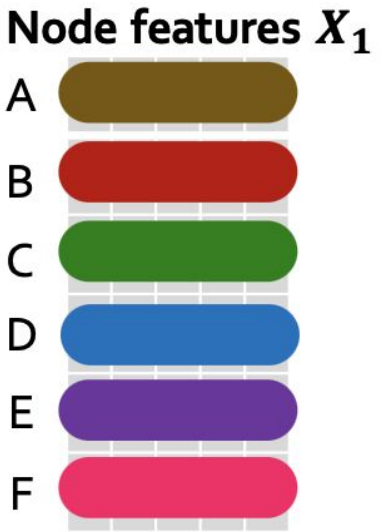
Permute the input (nodes) → output stays the same (map graph to the vector).

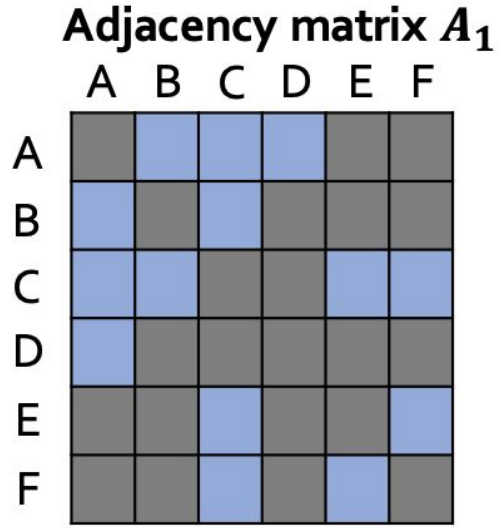
Invariance is often desired when the goal is to produce a global output (e.g., whole-graph classification) since the order of nodes shouldn't matter.

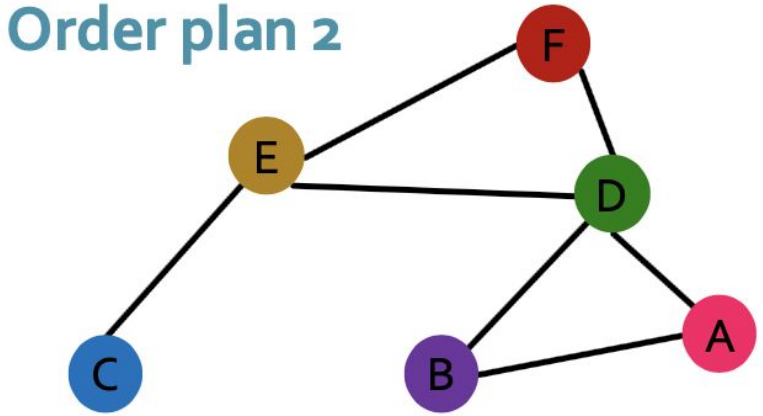
Permutation Invariance

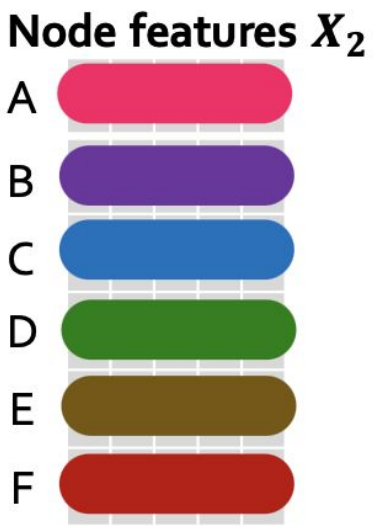
Graph does not have a canonical order of the nodes!

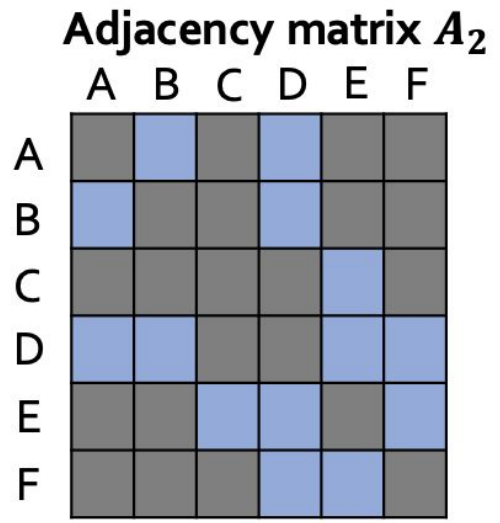






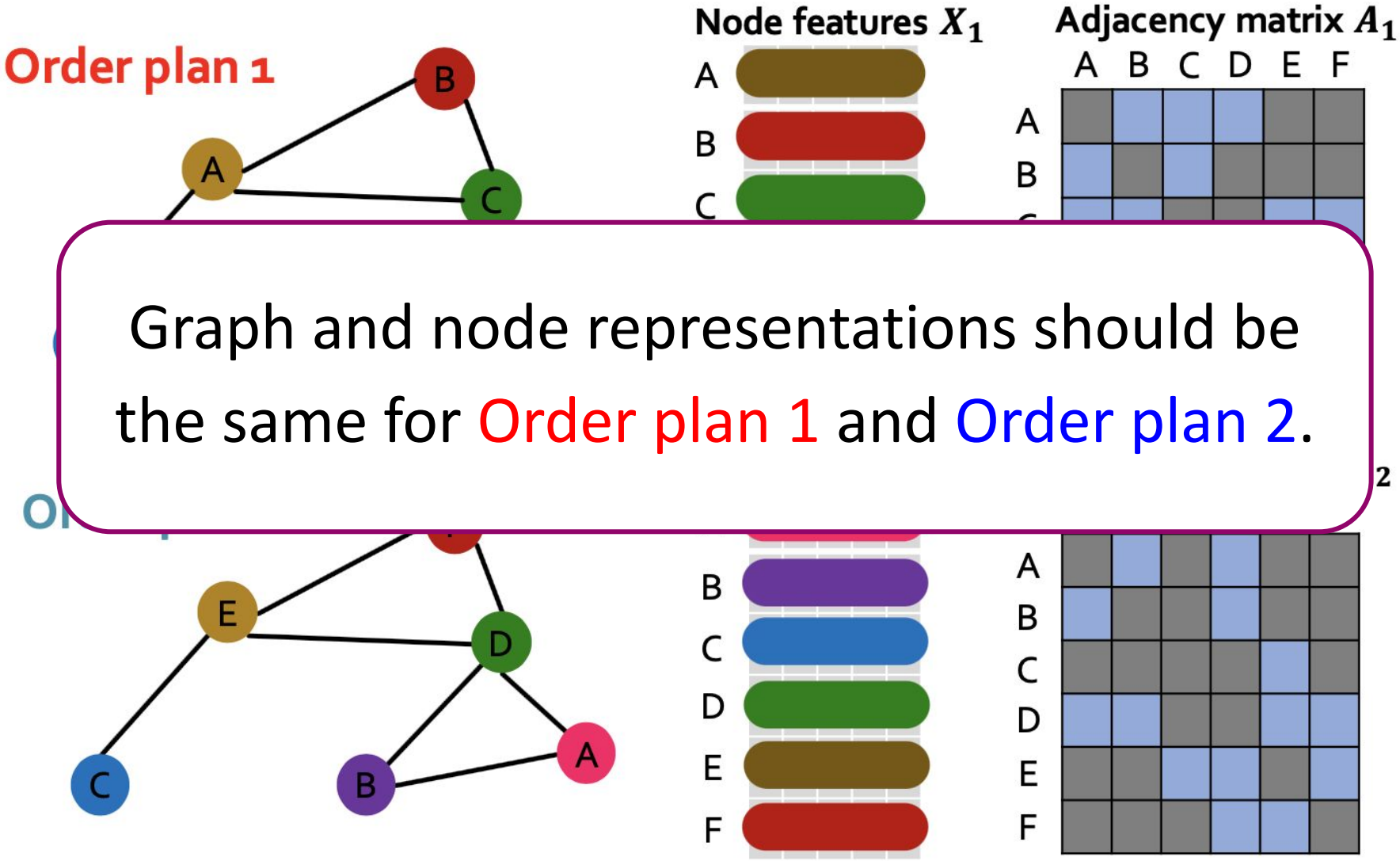






Permutation Invariance

Graph does not have a canonical order of the nodes!

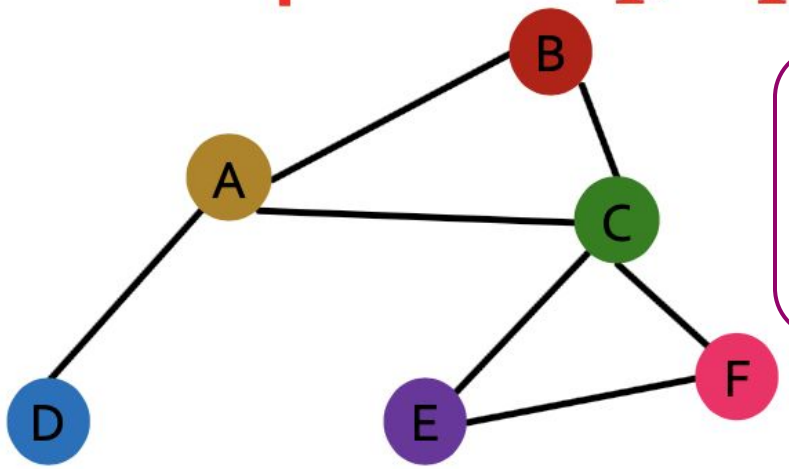


Permutation Invariance

Consider we learn a function g, that maps a graph $G = (A, X)$ to a vector, then

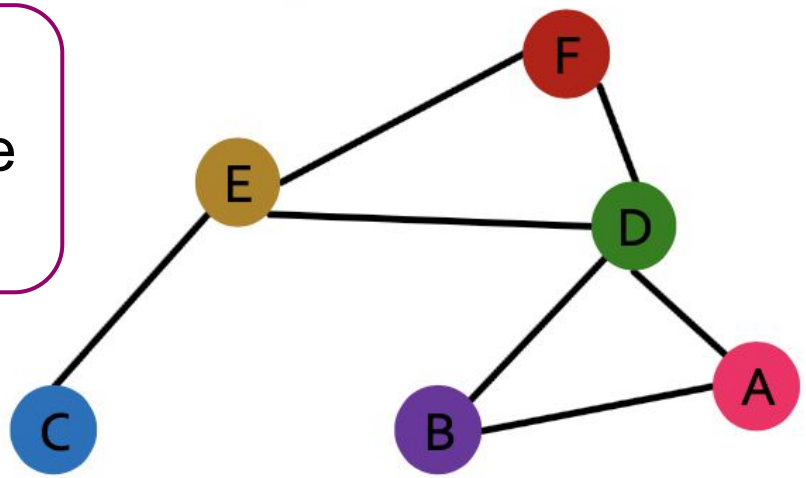
$$g(A_1, X_1) = g(A_2, X_2)$$

Order plan 1: A_1, X_1



For two order plans output of g should be the same!

Order plan 2: A_2, X_2



Permutation Invariance & Equivariance

Graph does not have a canonical order of the nodes →
We need to make sure that permuting the nodes and edges, does not change the outputs.

We recover the following rules a GNN must satisfy:

For any node function $f : \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times d} \to \mathbb{R}^{|V| \times d}$, f is permutation equivariant if

$$Pf(A, X) = f\left(PAP^{\top}, PX\right)$$

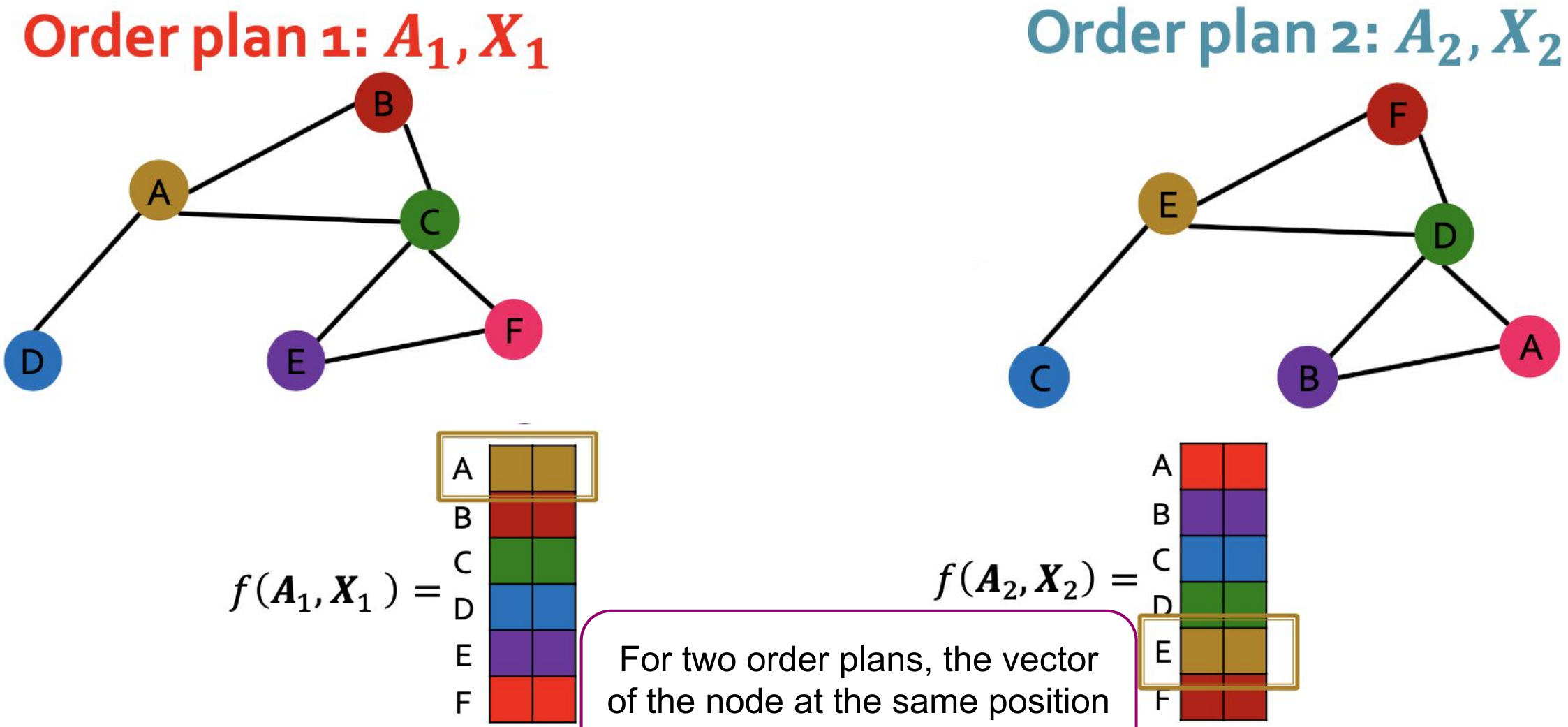
for any permutation matrix P.

Permute the input (nodes) → output also permutes accordingly (map graph to the matrix).

Equivariance is relevant when node-level outputs are important, as it ensures consistent representation across different node orderings.

Permutation Equivariance

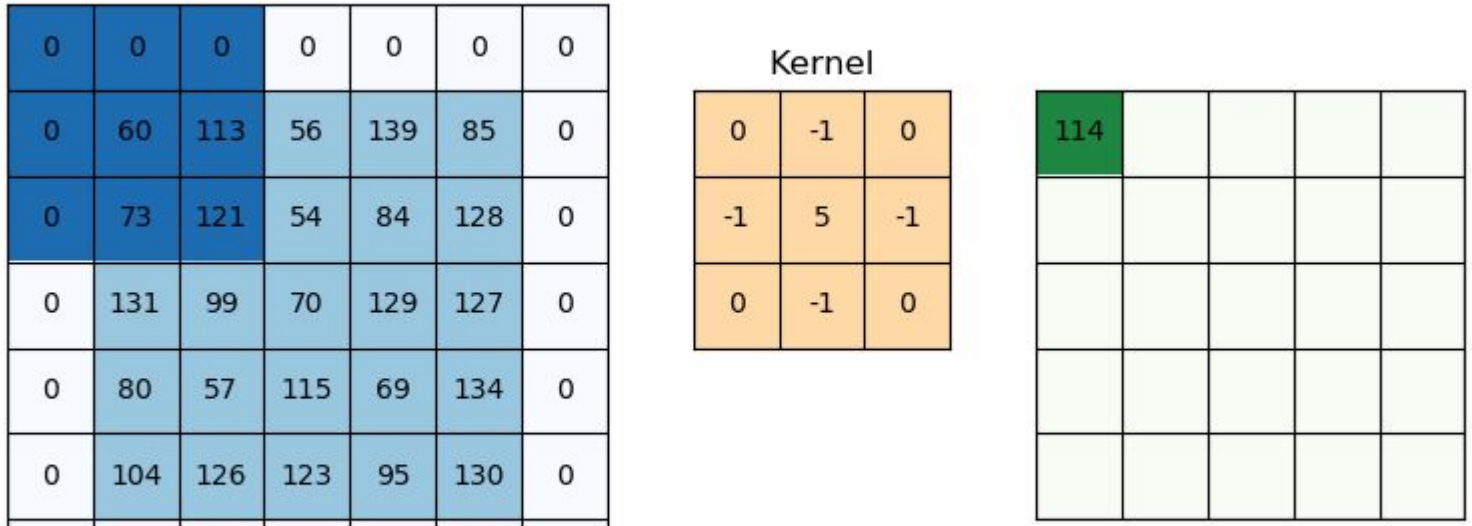
Consider we learn a function f, that maps node of a graph G to a matrix, then



For two order plans, the vector of the node at the same position in the graph is the same!

Credit: [1]

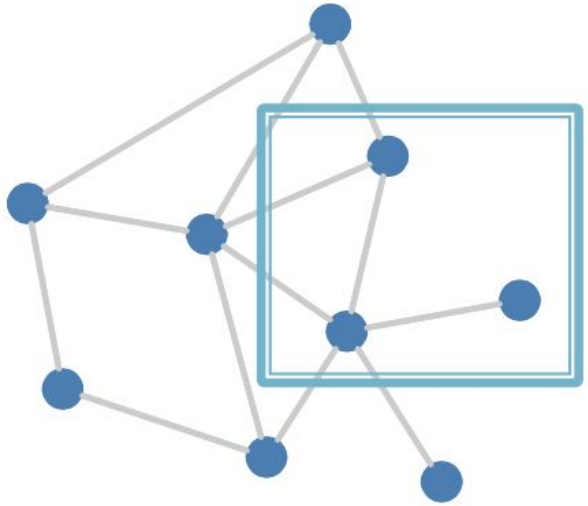
Idea: Convolutional Neural Network

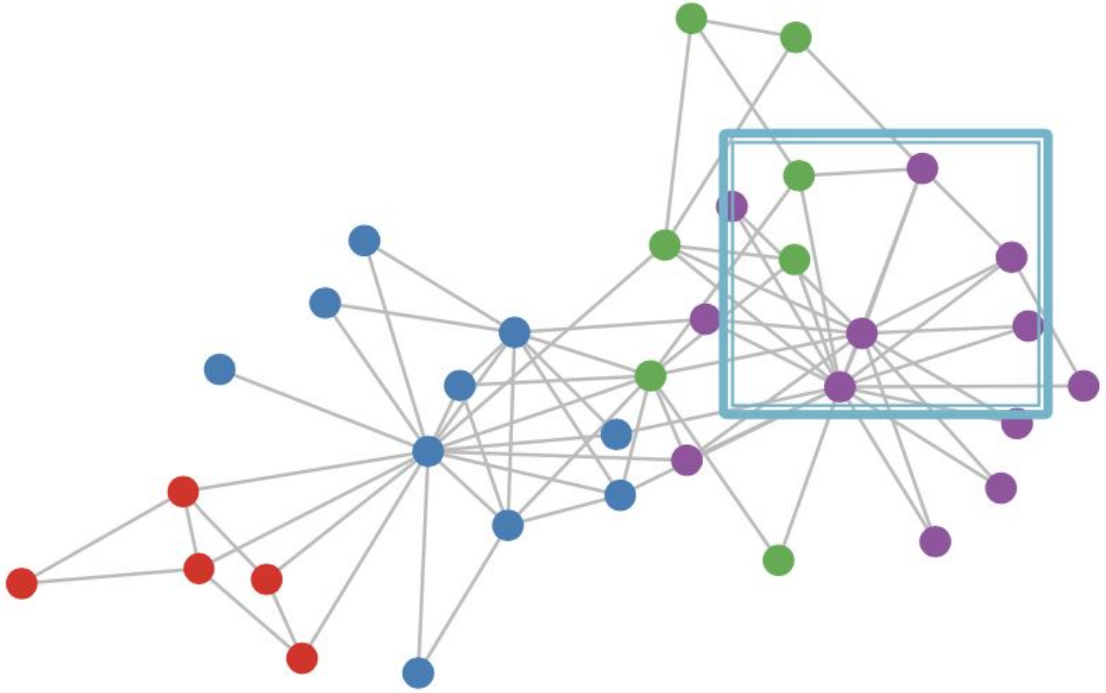


The goal is to generalize convolutions beyond simple lattices and leverage node features.

Credit: [1, 8]

Recap: why it is difficult

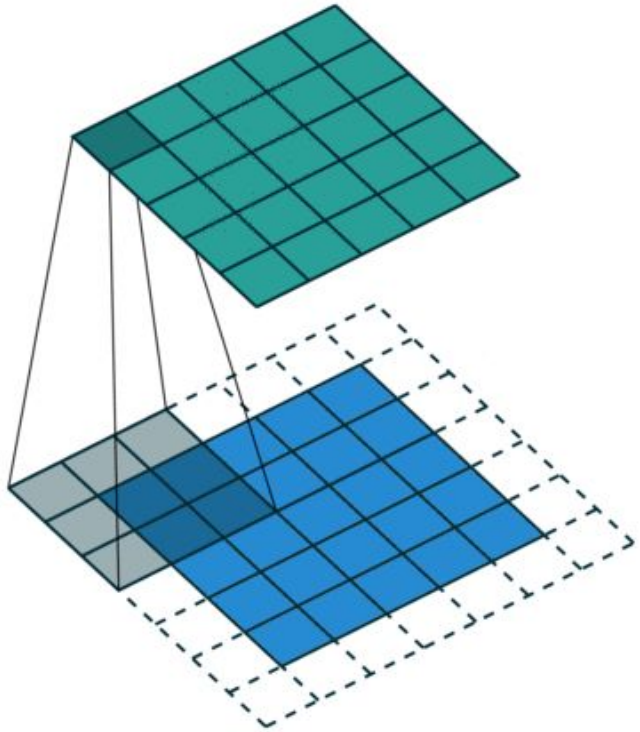




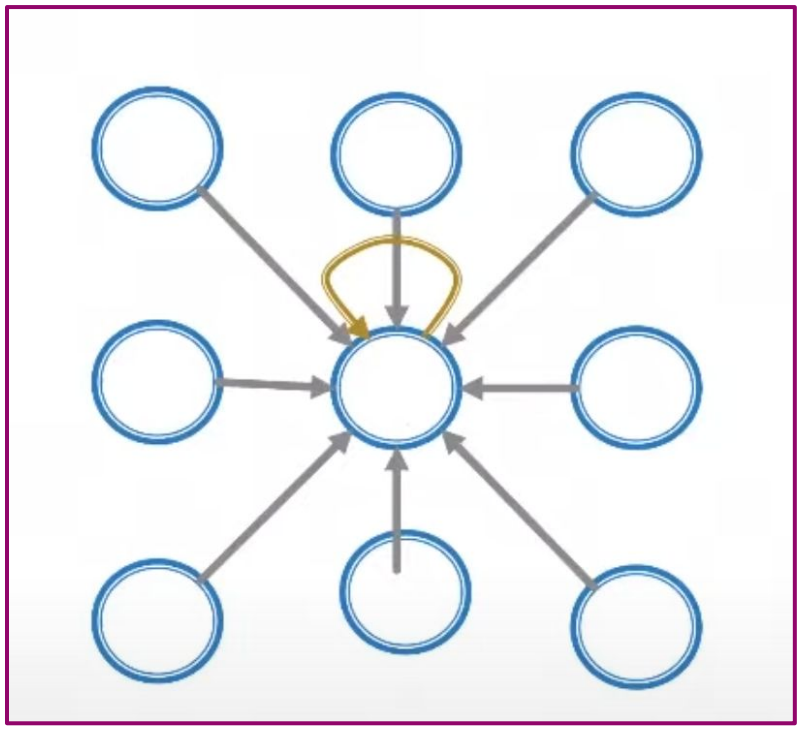
Hard to define what is the window and how it slides.

Credit: [1]

From Images to Graphs



Image

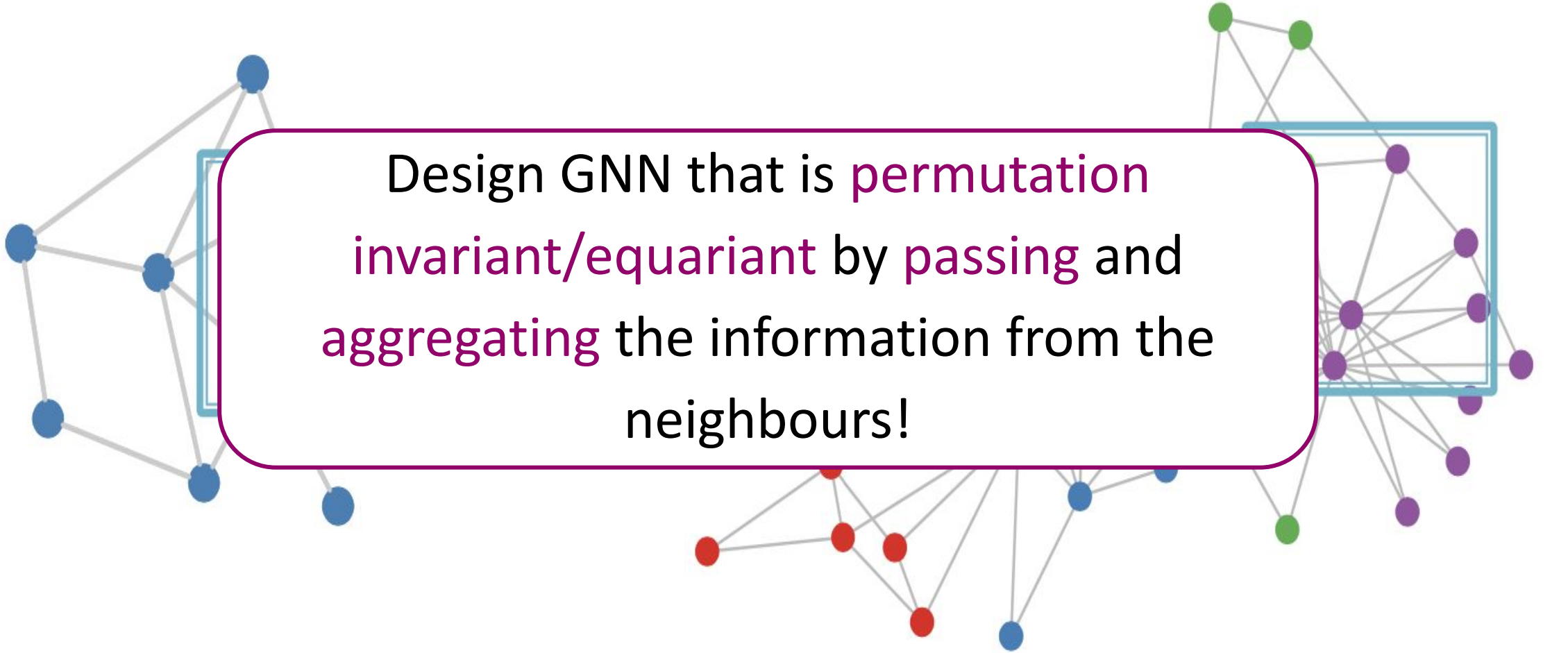


Graph

Idea: transform and aggregate information from the neighbours.

Credit: [1, 8]

Goal

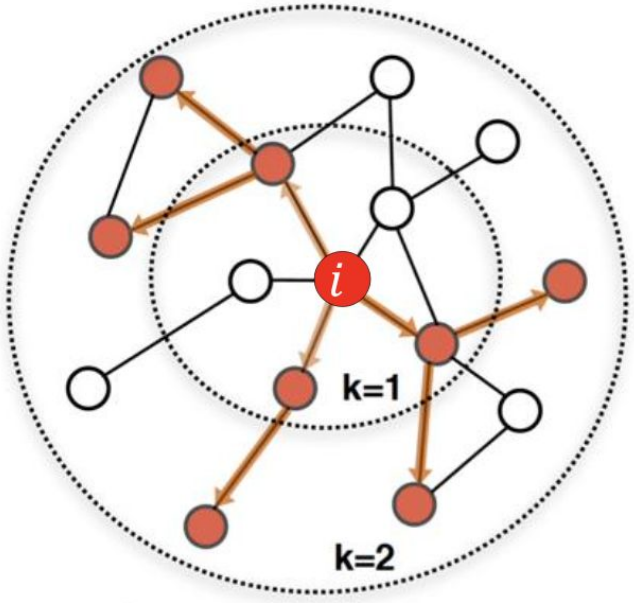


Design GNN that is permutation invariant/equariant by passing and aggregating the information from the neighbours!

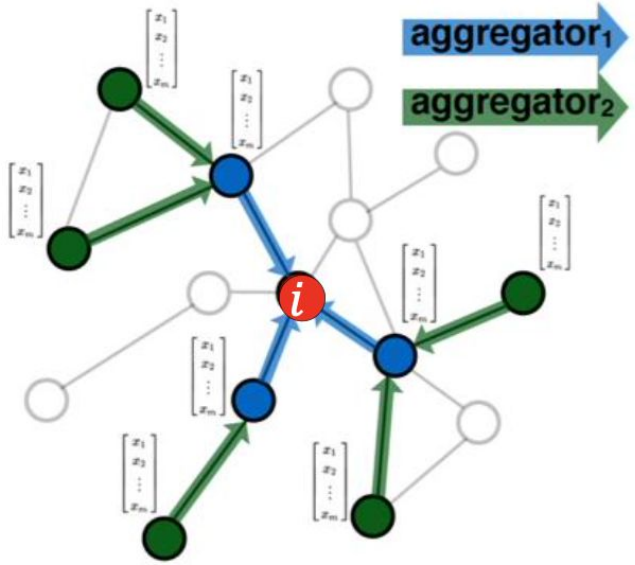
Graph Neural Networks

Graph Neural Networks

Idea: node's neighbourhood defines a computation graph.



Determine node computation graph

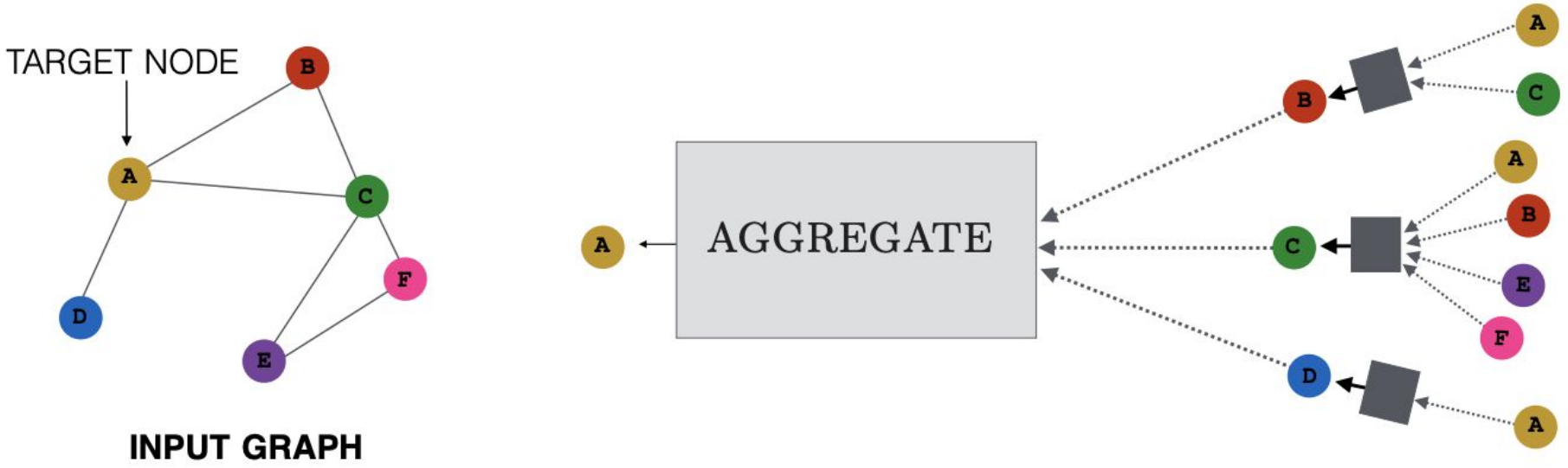


Propagate & transform the information

Credit: [1]

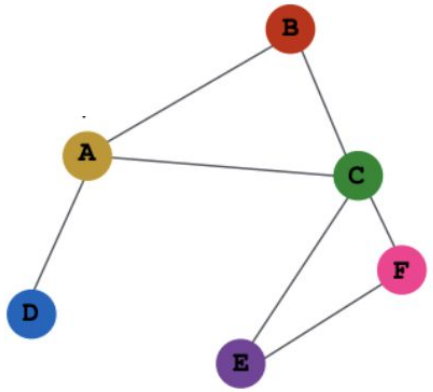
Neural Message Passing

Key idea: generate node embeddings based on local network neighborhood.



Credit: [1]

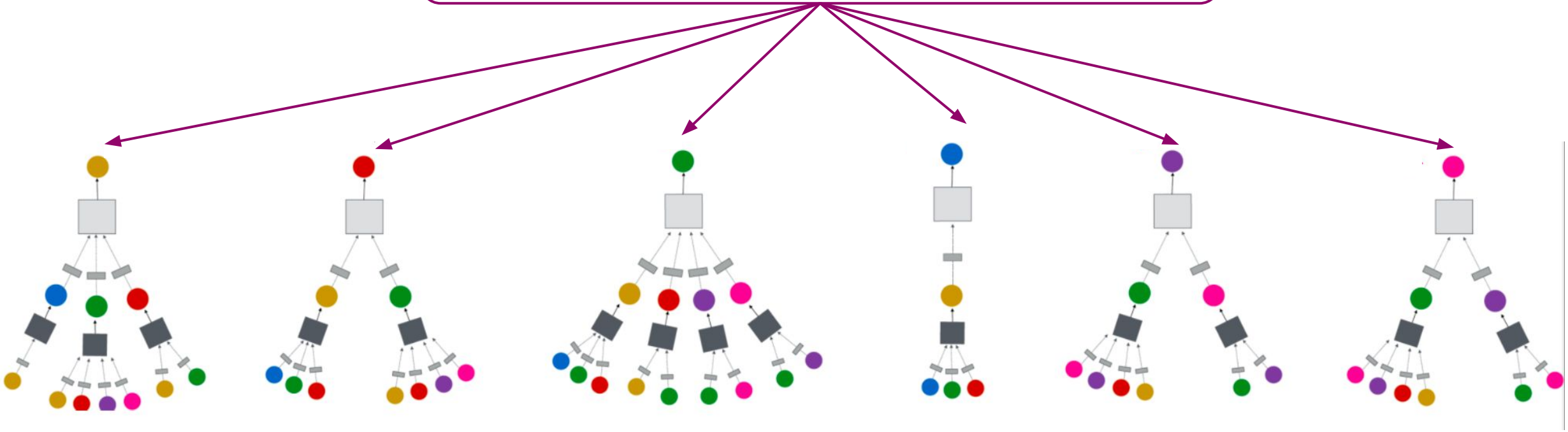
Neural Message Passing



INPUT GRAPH

Network neighbourhood defines a computational graph.

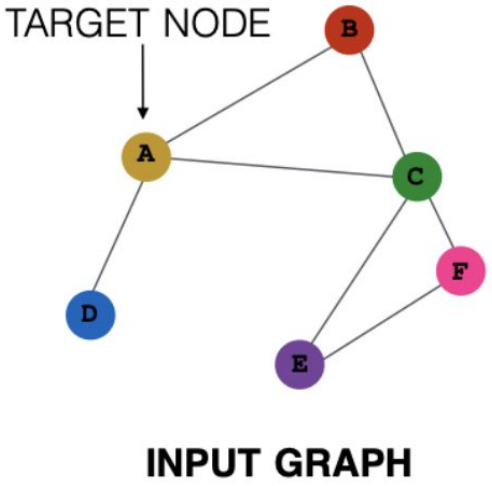
Every node defines a computational graph.

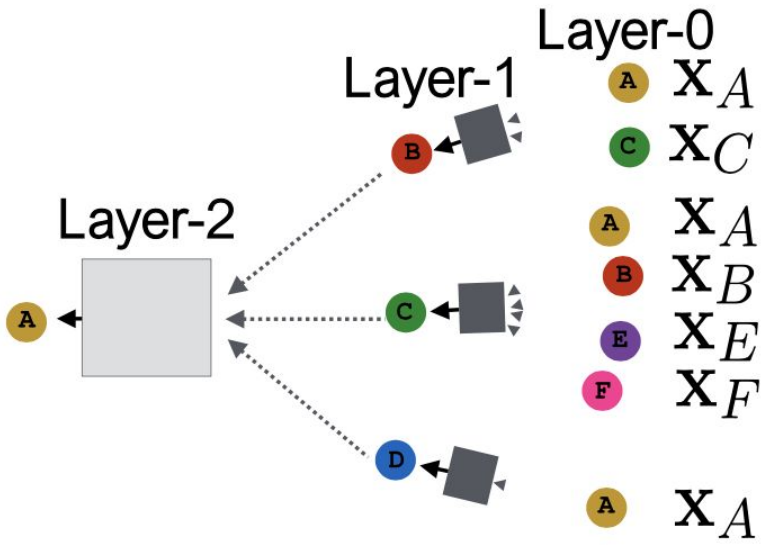


Deep Encoders: many layers

Model can be of arbitrary depth:

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





Credit: [1]

Neural Message Passing: Aggregate & Update

1. **AGGREGATE** function takes as input the set of embeddings of the nodes in v's graph neighborhood $\mathcal{N}(v)$ and generates the message based on this aggregated information.

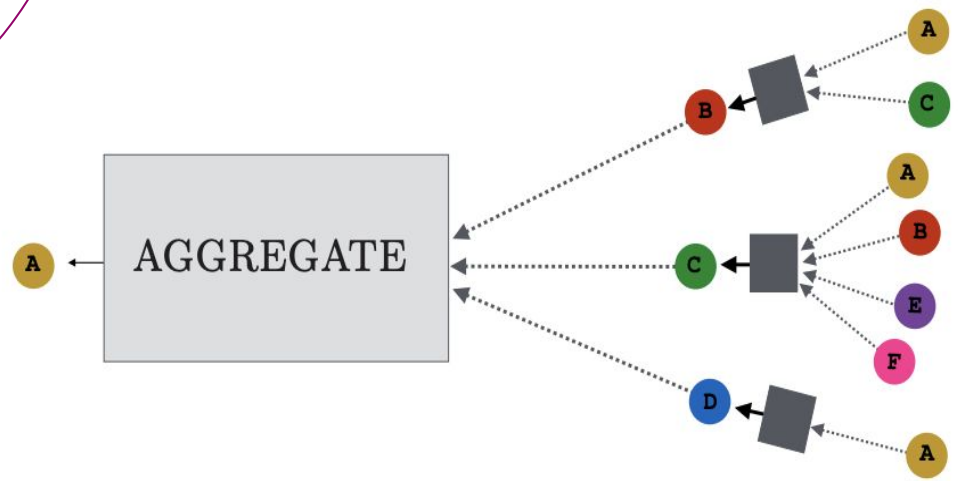
$$h_v^0 = x_v, \forall v \in V$$

Node's v initial embeddings.

For each step (layer) $k = 1, ..., K$:

$$m_{\mathcal{N}(v)}^k = \text{AGGREGATE}\left(\{h_u^k, \forall u \in \mathcal{N}(v)\}\right)$$

Message, aggregated from v's graph neighborhood $N(v)$



Neural Message Passing: Aggregate & Update

1. **AGGREGATE** function takes as input the set of embeddings of the nodes in v's graph neighborhood $\mathcal{N}(v)$ and generates the message based on this aggregated information.

$$h_v^0 = x_v, \forall v \in V$$

Node's v initial embeddings.

For each step (layer) $k = 1, ..., K$:

$$m_{\mathcal{N}(v)}^{k-1} = \text{AGGREGATE}\left(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right)$$

Note that since the **AGGREGATE** function takes a set as input, it's design define is the GNNs permutation equivariant/invariant.

Neural Message Passing: Aggregate & Update

1. **AGGREGATE** function takes as input the set of embeddings of the nodes in v's graph neighborhood $\mathcal{N}(v)$ and generates the message based on this aggregated information.

$$h_v^0 = x_v, \forall v \in V$$

For each step (layer) $k = 1, ..., K$:

$$m_{\mathcal{N}(v)}^{k-1} = \text{AGGREGATE}\left(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right)$$

2. **UPDATE** function combines this message with previous node's v embedding to create the new embedding ($k = 1, ..., K$):

$$h_v^k = \text{UPDATE}\left(h_v^{k-1}, m_{\mathcal{N}(v)}^{k-1}\right)$$

Neural Message Passing

For each step (layer) $k = 1, ..., K$:

$$m_{\mathcal{N}(v)}^{k-1} = \text{AGGREGATE}\left(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right)$$

$$h_v^k = \text{UPDATE}\left(h_v^{k-1}, m_{\mathcal{N}(v)}^{k-1}\right)$$

AGGREGATE and UPDATE are arbitrary differentiable functions (i.e., neural networks).

$$z_v = h_v^K$$

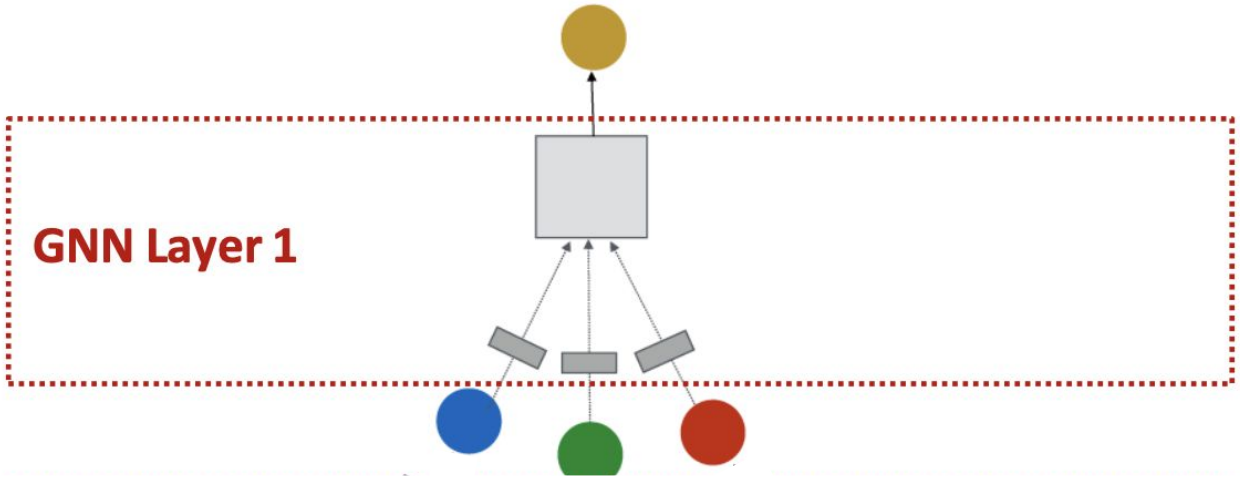
Final node embedding.

We can feed these embeddings into any loss function and run SGD to train the weight parameters.

Neural Message Passing: Summary

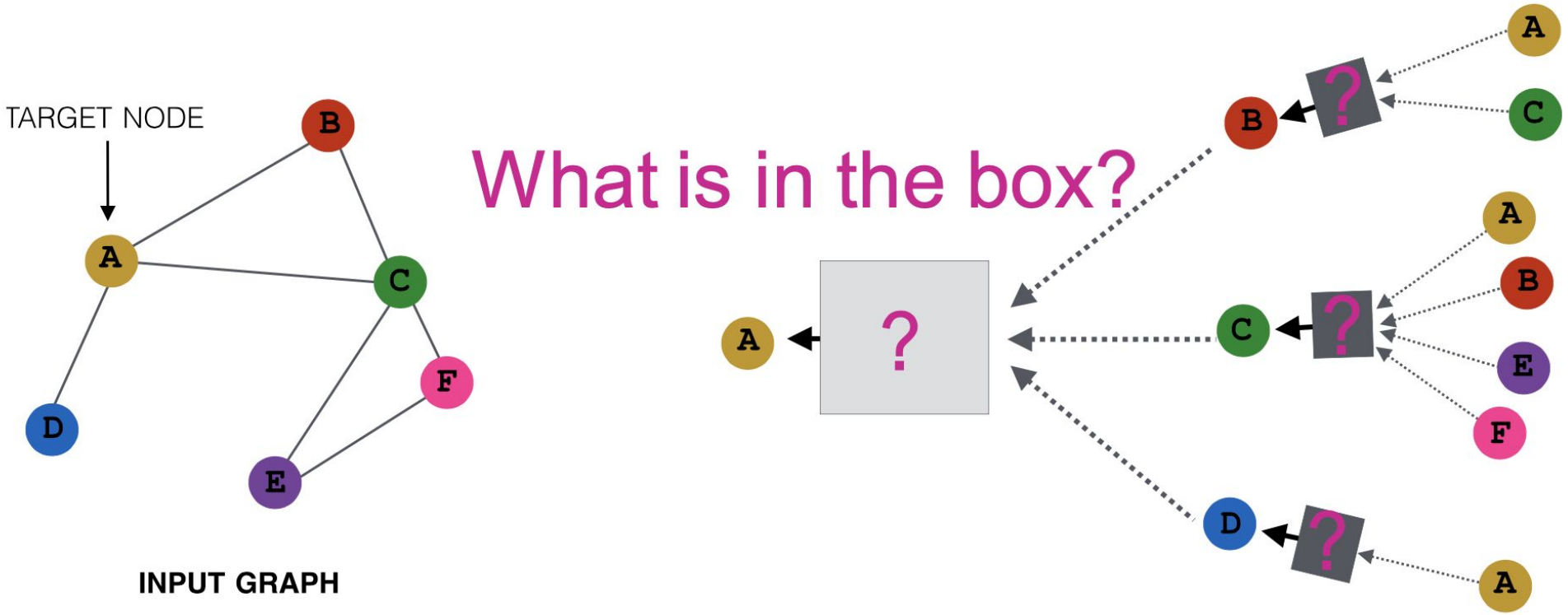
Two-step process:

1. Aggregate messages from the neighbours;
2. Update the node features.



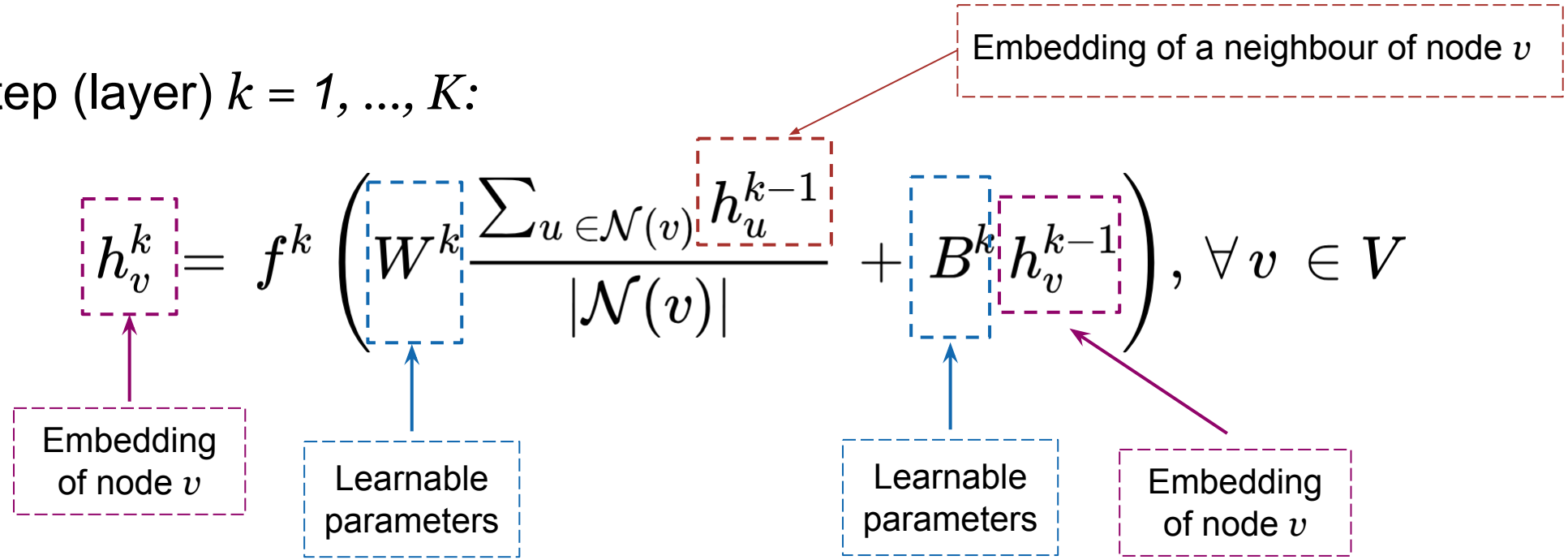
Neural Message Passing

Neighbourhood aggregation: key distinctions are in how different approaches aggregate information across the layers.



Graph Convolutional Networks (GCN)

For each step (layer) $k = 1, ..., K$:



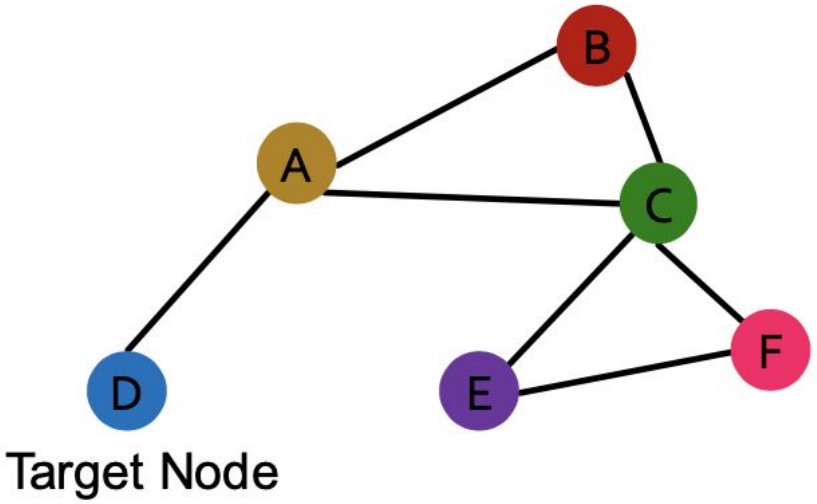
W^k – weight matrix of neighbourhood aggregation;

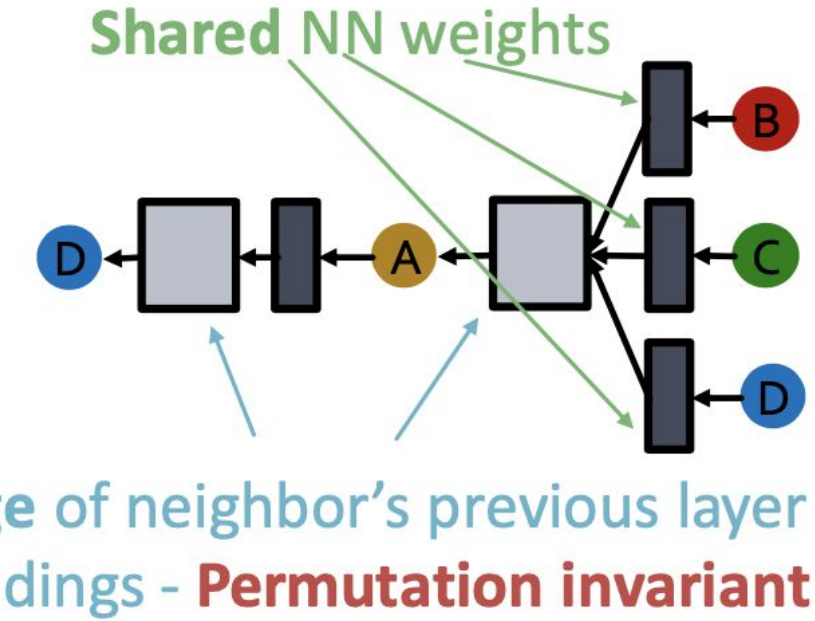
B^k – weight matrix for transforming vector itself.

Are shared across all nodes within the same layer.

Notes on Invariance and Equivariance Properties

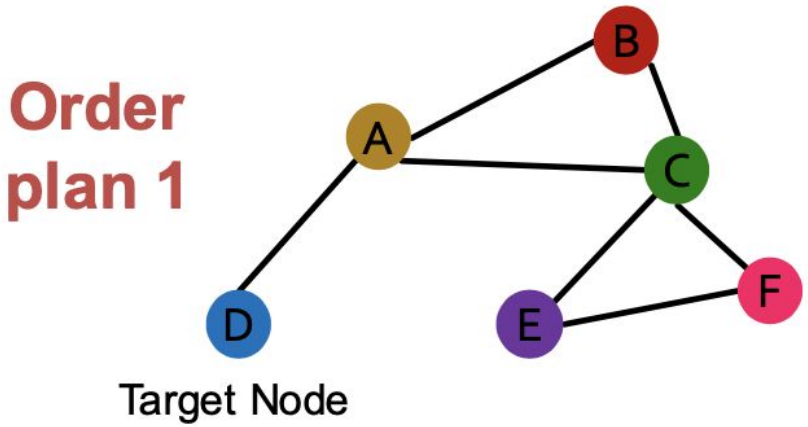
Given a node, GCN that computes its embedding is permutation invariant.

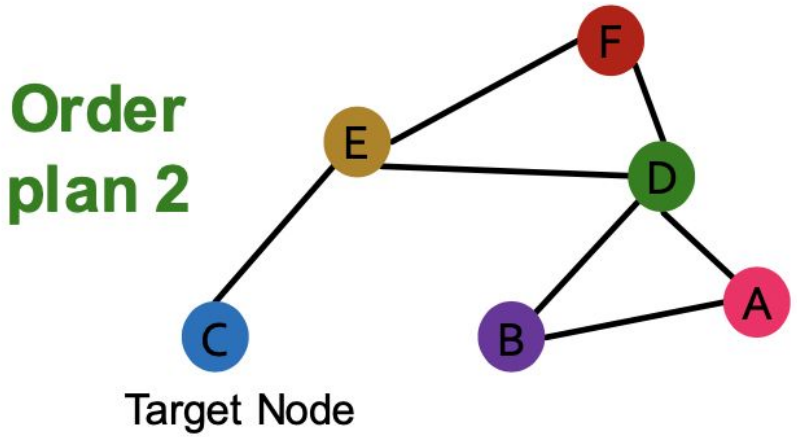


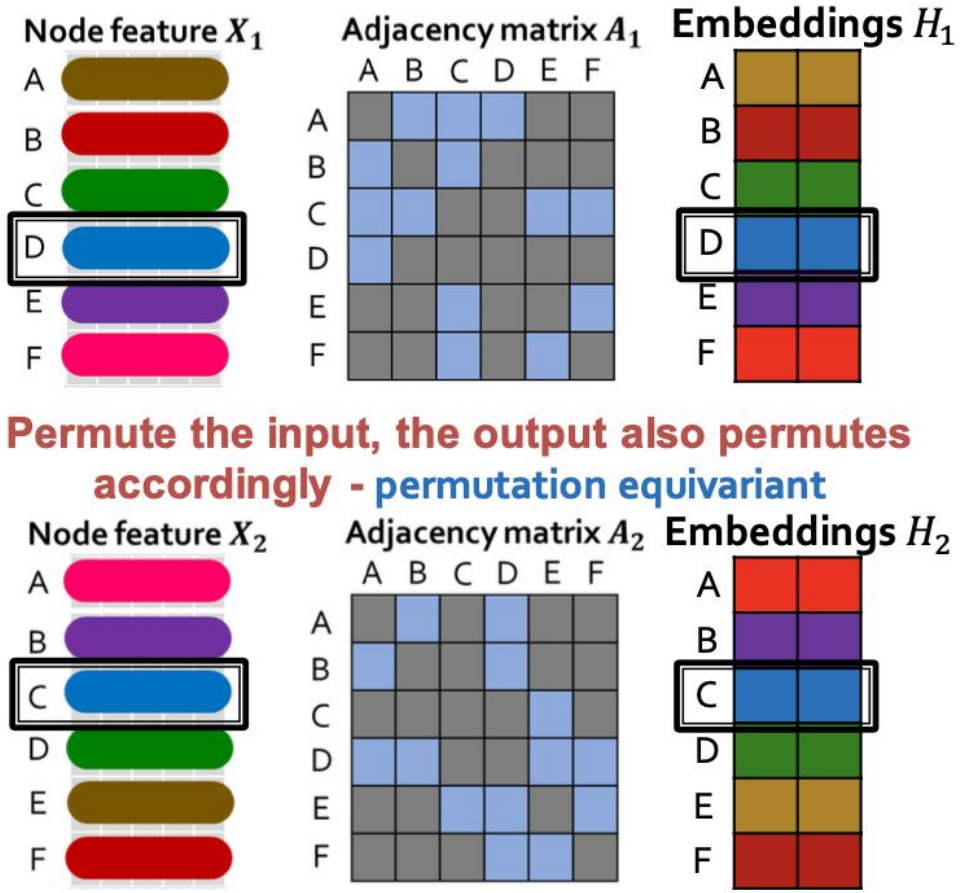


Notes on Invariance and Equivariance Properties

Considering all nodes in a graph, GCN computation is permutation equivariant.







Credit: [1]

Notes on Invariance and Equivariance Properties

Considering all nodes in a graph, GCN computation is permutation equariant.

1. The rows of **input node features** and **output embeddings** are **aligned** (by design).

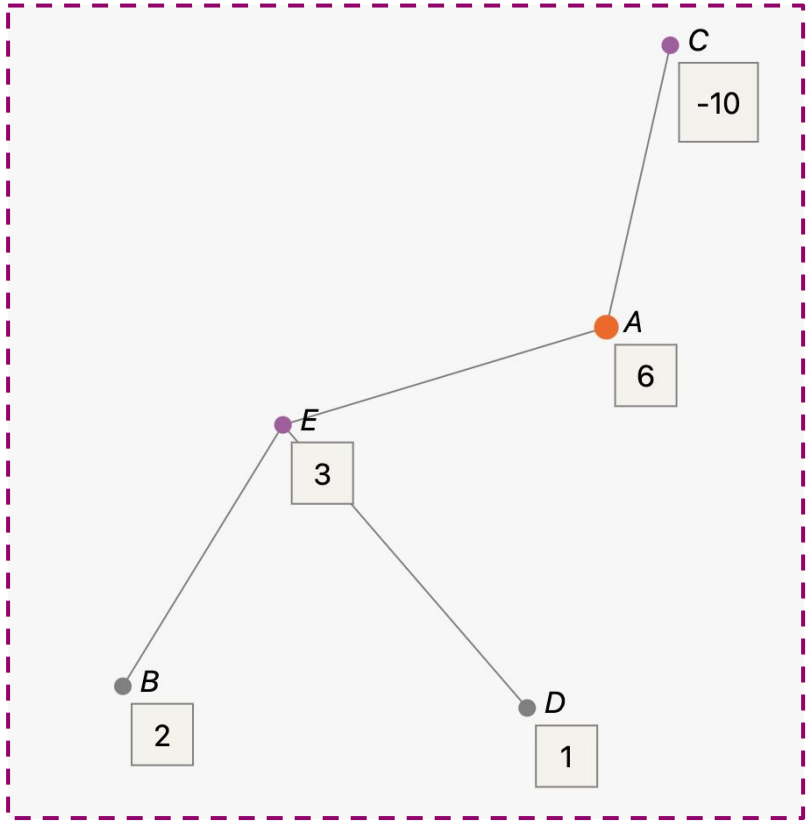
&

2. We know computing the embedding of a **given node** with GCN is **invariant** (see prev. slide).

3. So, after permutation, the **location** of **a given node** in the **input node feature** matrix is changed, but the **the output embedding of a given node stays the same** (because of perm. inv. of the embedding computation).

GCN: example

Layer 1



$$W^1 = 1$$
$$B^1 = 1$$

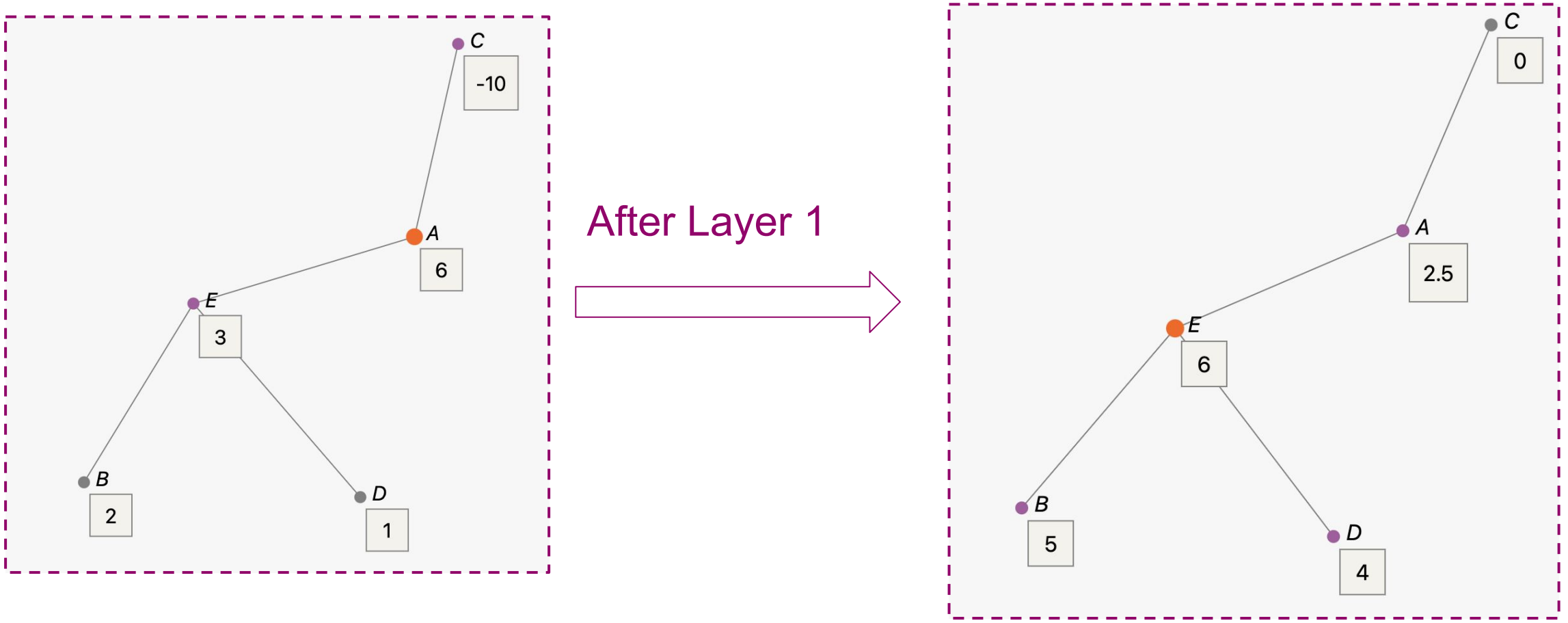
Equation for A

$$\begin{aligned} h_A^{(1)} &= f\left(W^{(1)} \times \frac{h_C^{(0)} + h_E^{(0)}}{2} + B^{(1)} \times h_A^{(0)}\right) \\ &= f\left(1 \times \frac{-10+3}{2} + 1 \times 6\right) \\ &= f\left(-3.5 + 6\right) \\ &= f\left(2.5\right) \\ &= \mathrm{ReLU}\left(2.5\right) = 2.5. \end{aligned}$$

Equation for E

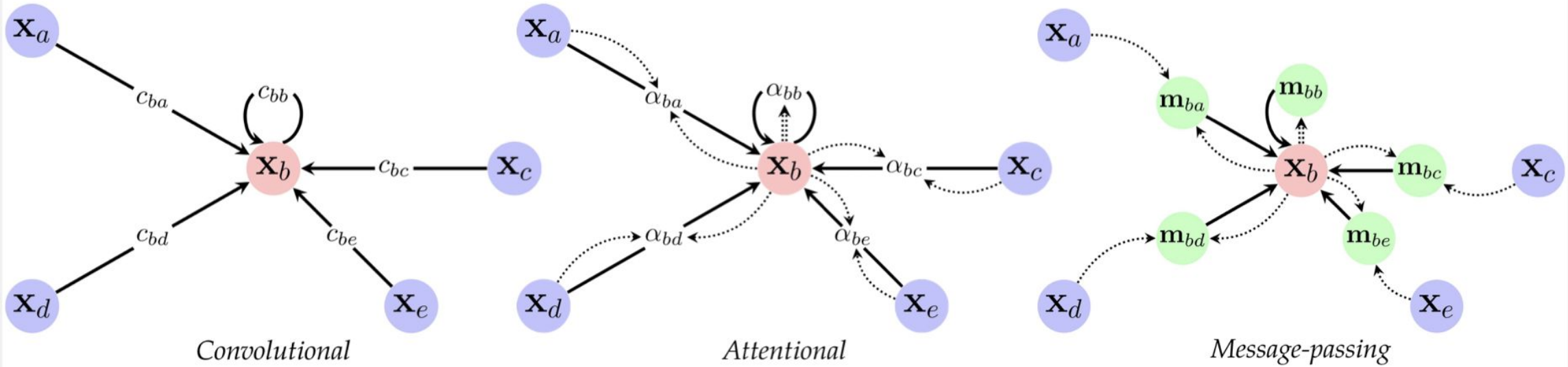
$$\begin{aligned} h_E^{(1)} &= f\left(W^{(1)} \times \frac{h_A^{(0)} + h_B^{(0)} + h_D^{(0)}}{3} + B^{(1)} \times h_E^{(0)}\right) \\ &= f\left(1 \times \frac{6+2+1}{3} + 1 \times 3\right) \\ &= f\left(3 + 3\right) \\ &= f\left(6\right) \\ &= \mathrm{ReLU}\left(6\right) = 6. \end{aligned}$$

GCN: example



Credit: [6]

Zoo of GNN Architectures



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

e.g. GraphSAGE, GCN, SGC

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

e.g. MoNet, GAT, GATv2

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

e.g. IN, MPNN, GraphNet

Credit: [10]

Graph Sample and Aggregate

Graph Sample and Aggregate (GraphSAGE)

$$h_v^k = f^k\left(W^k\left[\mathrm{AGG}_{u \in \mathcal{N}(v)}\left(\{h_u^{k-1}\}\right), h_v^{k-1}\right]\right), \forall v \in V$$

Learnable parameters

Neighborhood definition:

uniformly sample a fixed-size set of neighbors, instead of using full neighborhood set.

Graph Sample and Aggregate

Graph Sample and Aggregate (GraphSAGE)

$$h_v^k = f^k\left(W^k\left[\mathrm{AGG}_{u\in\mathcal{N}(v)}\left(\{h_u^{k-1}\}\right), h_v^{k-1}\right]\right), \forall v \in V$$

Learnable parameters

AGG choice:

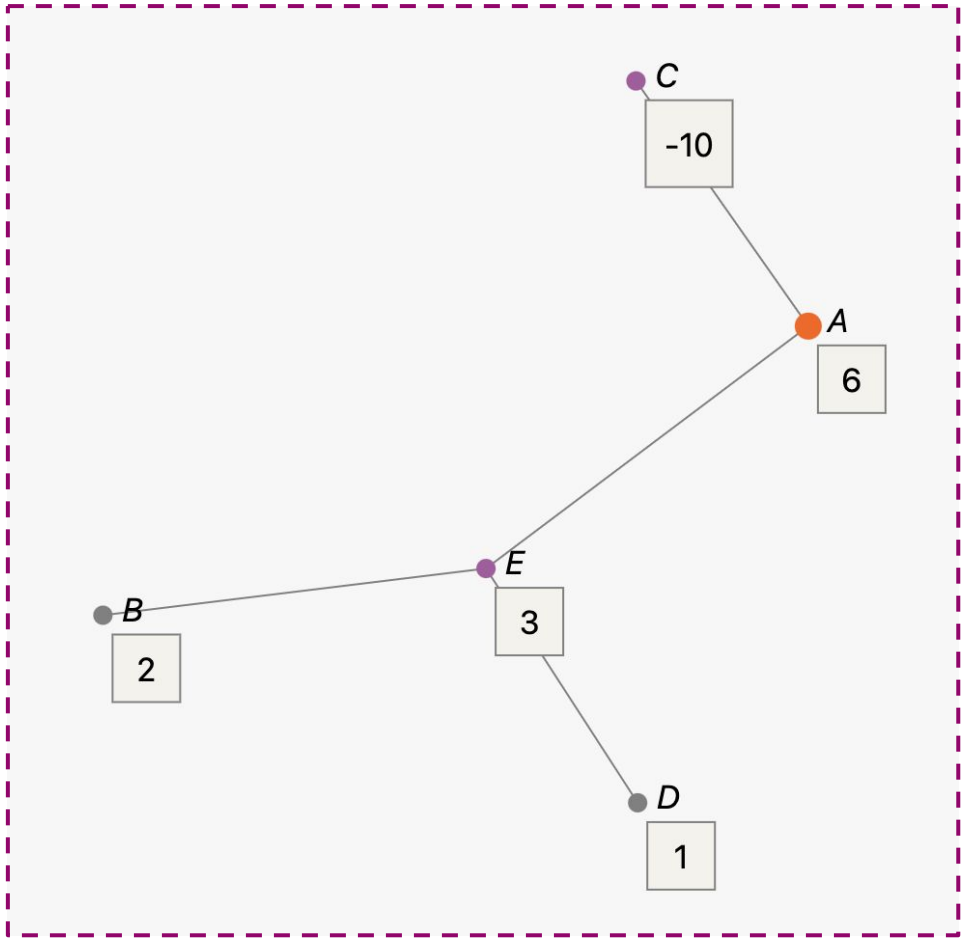
- Mean (similar to the GCN);
- Pool: transform neighbor vectors and apply Mean($\cdot$) or Max($\cdot$):

$$\max\left(\left\{\mathrm{MLP}(\, h_u^{k-1}), \forall u \in \mathcal{N}(v)\right\}\right)$$

- LSTM (after ordering the sequence of neighbours).

GraphSage: example

Layer 1



$$W^1 = 1$$
$$B^1 = 1$$

Equation for A

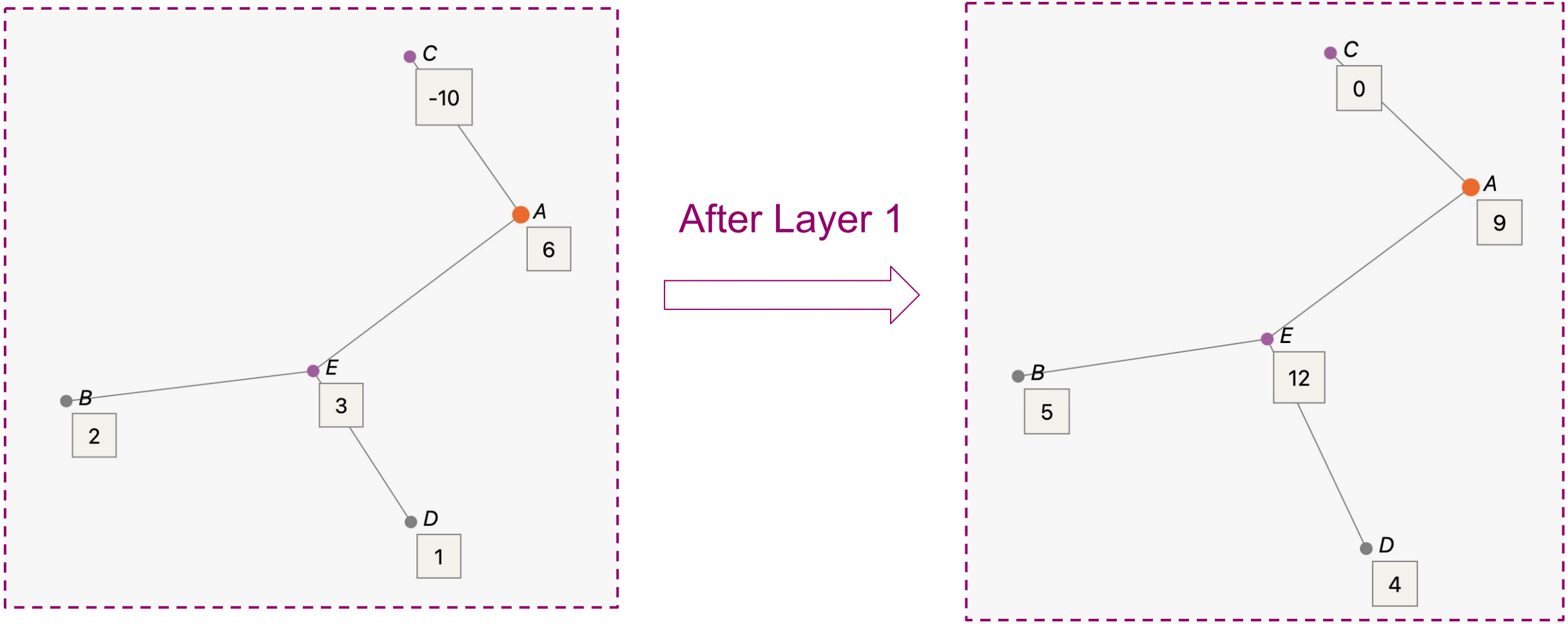
$$\begin{aligned} h_A^{(1)} &= f\left(W^{(1)} \times \mathrm{RNN}\left[h_C^{(0)}, h_E^{(0)}\right] + B^{(1)} \times h_A^{(0)}\right) \\ &= f\left(1 \times 3 + 1 \times 6\right) \\ &= f\left(3 + 6\right) \\ &= f\left(9\right) \\ &= \mathrm{ReLU}\left(9\right) = 9. \end{aligned}$$

Considering the sequence of neighbours as *[C,E]*, the RNN computes:

$$\begin{aligned} s_0 &= 0. \\ s_1 &= \mathrm{ReLU}\left(U_s s_0 + U_h h_C^{(0)}\right) = \mathrm{ReLU}\left(1 \times 0 + 1 \times -10\right) = \mathrm{ReLU}\left(-10\right) = 0. \\ s_2 &= \mathrm{ReLU}\left(U_s s_1 + U_h h_E^{(0)}\right) = \mathrm{ReLU}\left(1 \times 0 + 1 \times 3\right) = \mathrm{ReLU}\left(3\right) = 3. \end{aligned}$$

The returned aggregated value 3 is just the last hidden state of RNN.

GraphSage: example



Credit: [6]

Graph Attention Network

Graph Attention Network (GAT): not all node's neighbors are equally important.

$$h_v^k = f^k\left(W^k\left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu} h_u^{k-1} + \alpha_{vv} h_v^{k-1}\right]\right), \forall v \in V$$

Attention Weights

In GCN / GraphSAGE:

- The importance of node's u message to node v: $\alpha_{vu} = \frac{1}{N(v)}$
- These weights are defined explicitly based on the structural properties of the graph → all neighbours are equally important.

Graph Attention Network

Graph Attention Network (GAT): not all node's neighbors are equally important.

$$h_v^k = f^k\left(W^k\left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu} h_u^{k-1} + \alpha_{vv} h_v^{k-1}\right]\right), \forall v \in V$$

Attention Weights

- Attention focuses on the important parts of data.
- Which part of the data is more important depends on the context and is learned through the training → attention weights should be learned.

Credit: [6, 12]

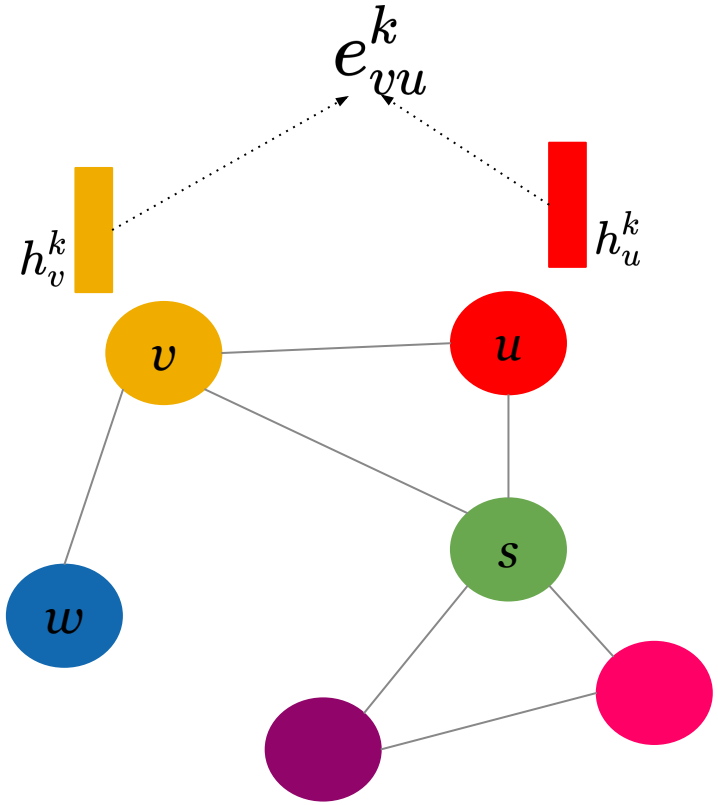
Graph Attention Network

Attention mechanism:

$$e_{vu}^k = a^\top [W h_v^k, W h_u^k]$$

Learnable parameters

Importance of u's message to node v



Any standard attention model from the DL literature can be used.

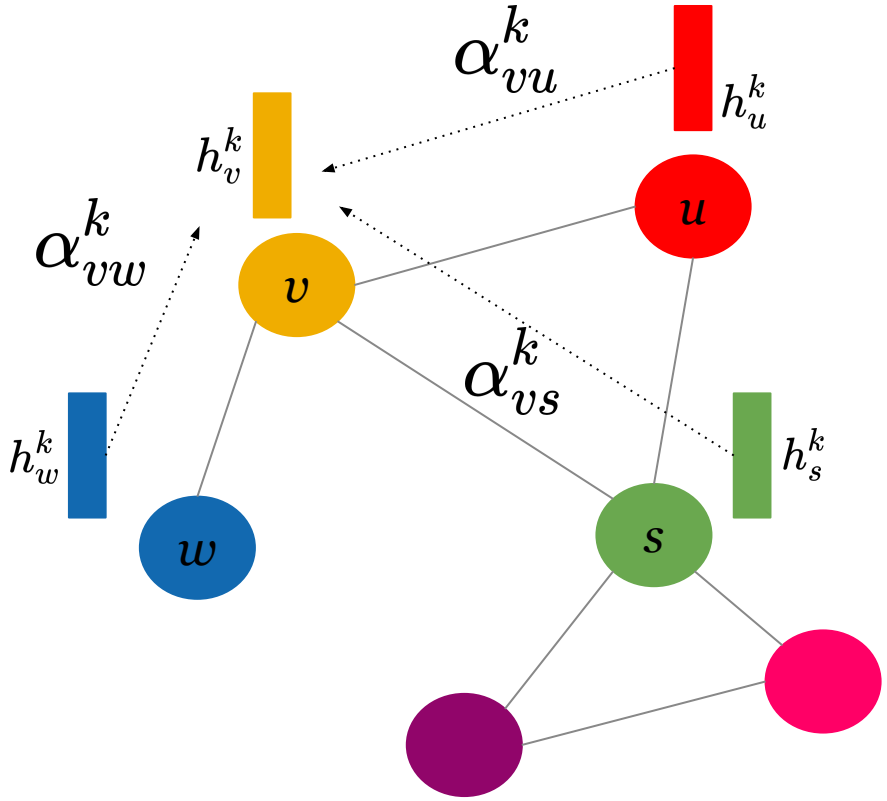
Graph Attention Network

Attention Mechanism:

$$e_{vu}^k = a^\top [W h_v^k, W h_u^k]$$

Normalization

$$\alpha_{vu}^k = \frac{\exp\left(e_{vu}^k\right)}{\sum_{v' \in \mathcal{N}(v)} \exp\left(e_{vv'}^k\right)}$$



Weighted sum

$$h_v^k = f^k\left(W^k\left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu} h_u^{k-1} + \alpha_{vv} h_v^{k-1}\right]\right), \forall v \in V$$

Graph Attention Network

Attention Mechanism – Multihead Attention:

create multiple attention scores (each replica with a different set of parameters);

for 3 heads:

$$h_v^k[1] = f^k\left(W^k\left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^1 h_u^{k-1} + \alpha_{vv}^1 h_v^{k-1}\right]\right)$$

$$h_v^k[2] = f^k\left(W^k\left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^2 h_u^{k-1} + \alpha_{vv}^2 h_v^{k-1}\right]\right)$$

$$h_v^k[3] = f^k\left(W^k\left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^3 h_u^{k-1} + \alpha_{vv}^3 h_v^{k-1}\right]\right)$$

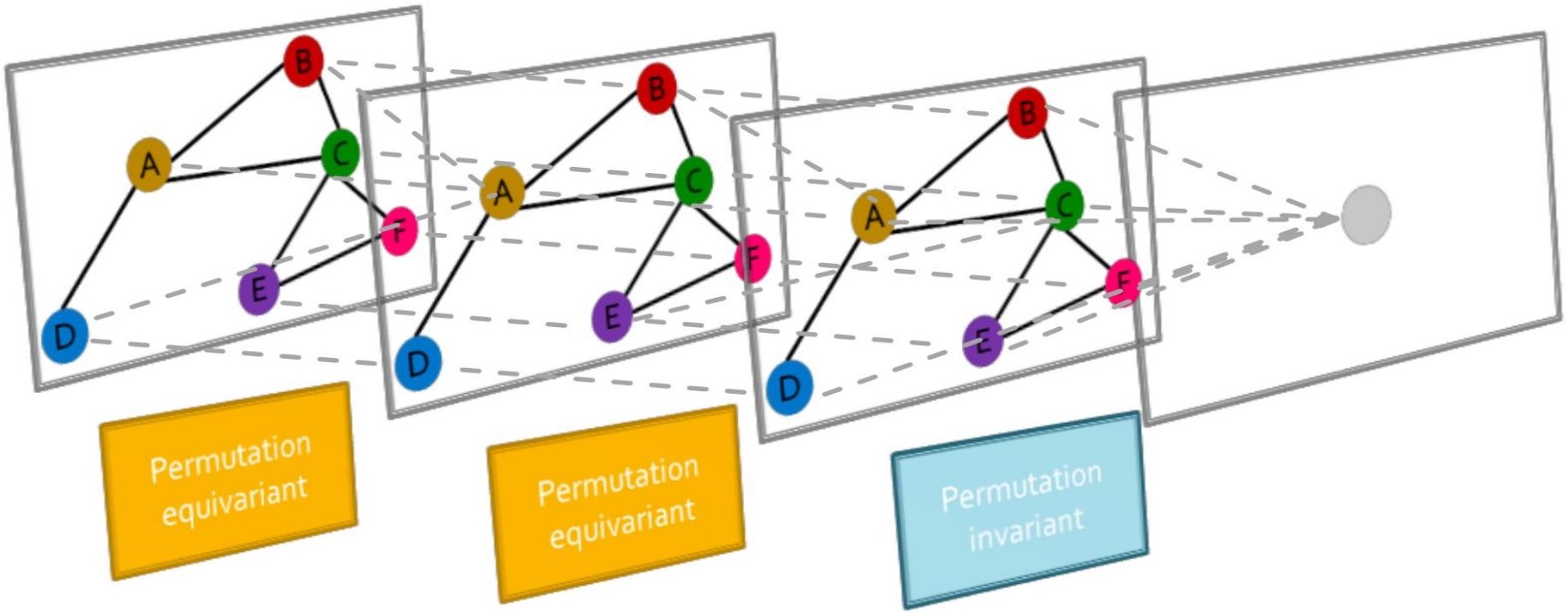
Outputs are aggregated to the one vector by concatenation or summation.

Summary of GNN Architectures

`MessagePassing`	Base class for creating message passing layers of the form
`SimpleConv`	A simple message passing operator that performs (non-trainable) propagation
`GCNConv`	The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper
`ChebConv`	The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper
`SAGEConv`	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper
`CuGraphSAGEConv`	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper.
`GraphConv`	The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper
`GravNetConv`	The GravNet operator from the "Learning Representations of Irregular Particle-detector Geometry with Distance-weighted Graph Networks" paper, where the graph is dynamically constructed using nearest neighbors.
`GatedGraphConv`	The gated graph convolution operator from the "Gated Graph Sequence Neural Networks" paper
`ResGatedGraphConv`	The residual gated graph convolutional operator from the "Residual Gated Graph ConvNets" paper
`GATConv`	The graph attentional operator from the "Graph Attention Networks" paper
`CuGraphGATConv`	The graph attentional operator from the "Graph Attention Networks" paper.
`FusedGATConv`	The fused graph attention operator from the "Understanding GNN Computational Graph: A Coordinated Computation, IO, and Memory Perspective" paper.
`GATv2Conv`	The GATv2 operator from the "How Attentive are Graph Attention Networks?" paper, which fixes the static attention problem of the standard `GATConv` layer.
`TransformerConv`	The graph transformer operator from the "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification" paper

Docs for PyTorch Geometric: Convolutional Layers

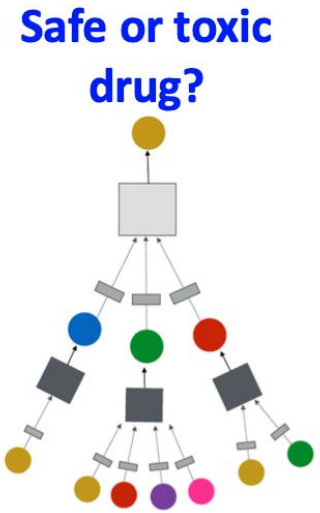
GNN consist of multiple permutation equivariant / invariant layers

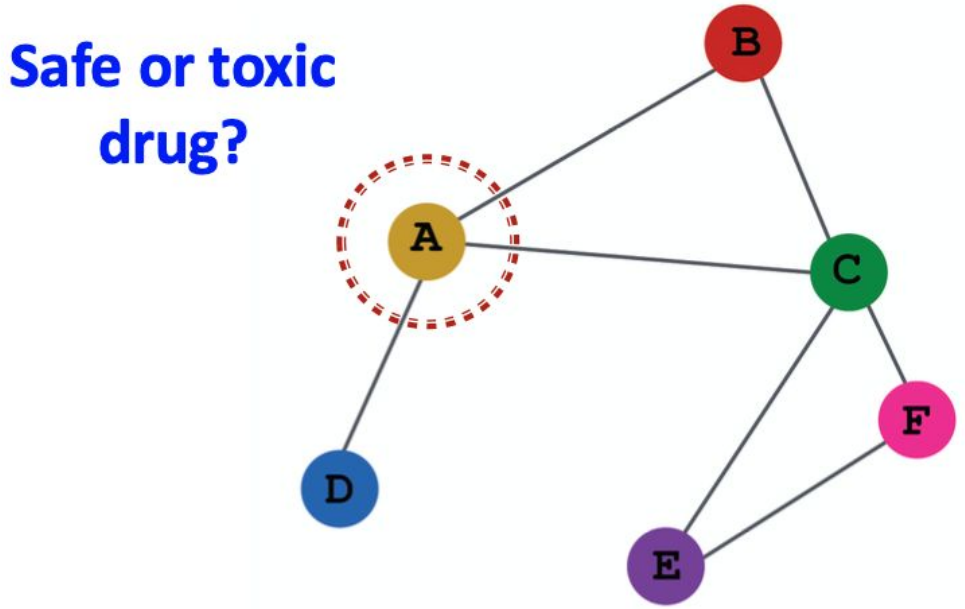


Credit: [1]

How to train a GNN

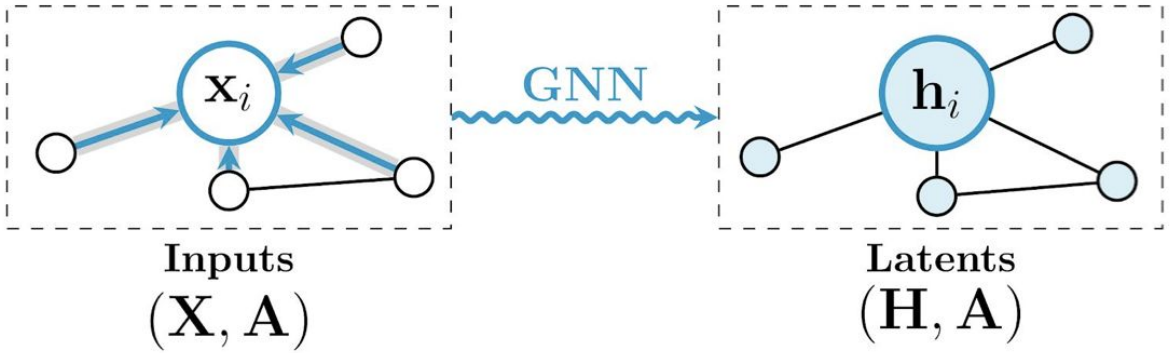
Directly train the model for a **supervised task** (e.g., node / graph classification).





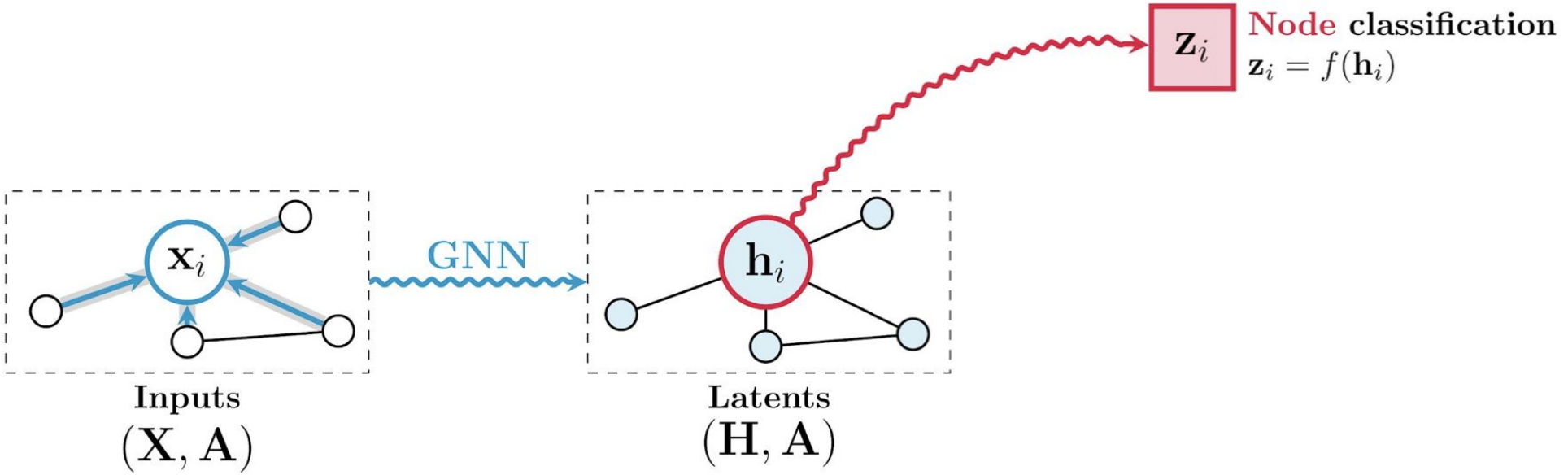
Credit: [1]

Setup



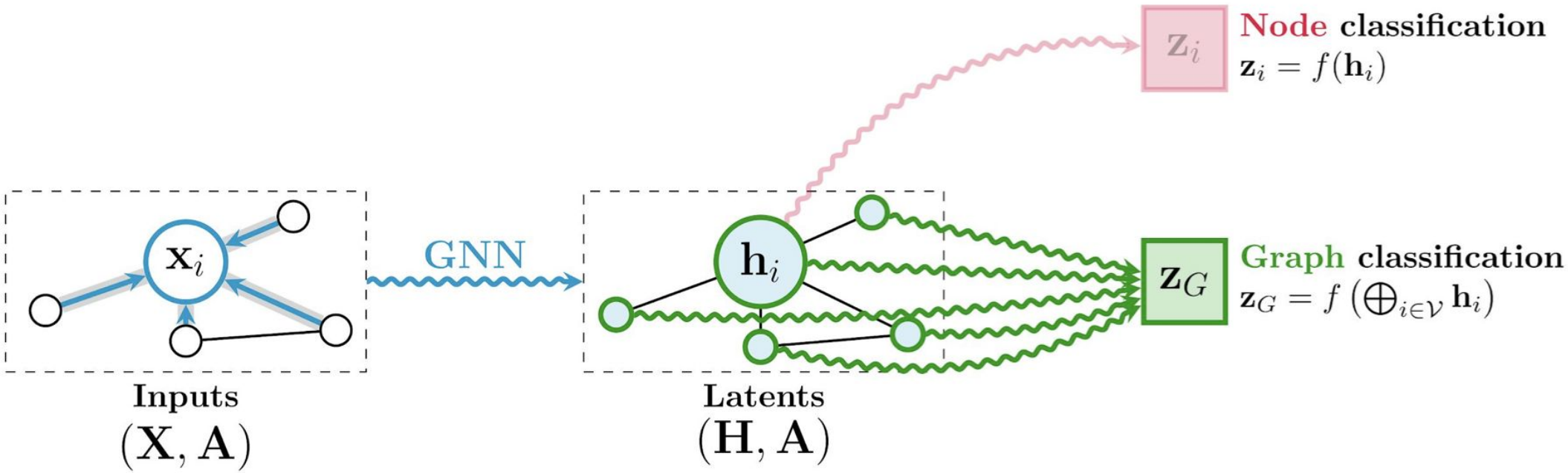
Credit: [5]

Setup



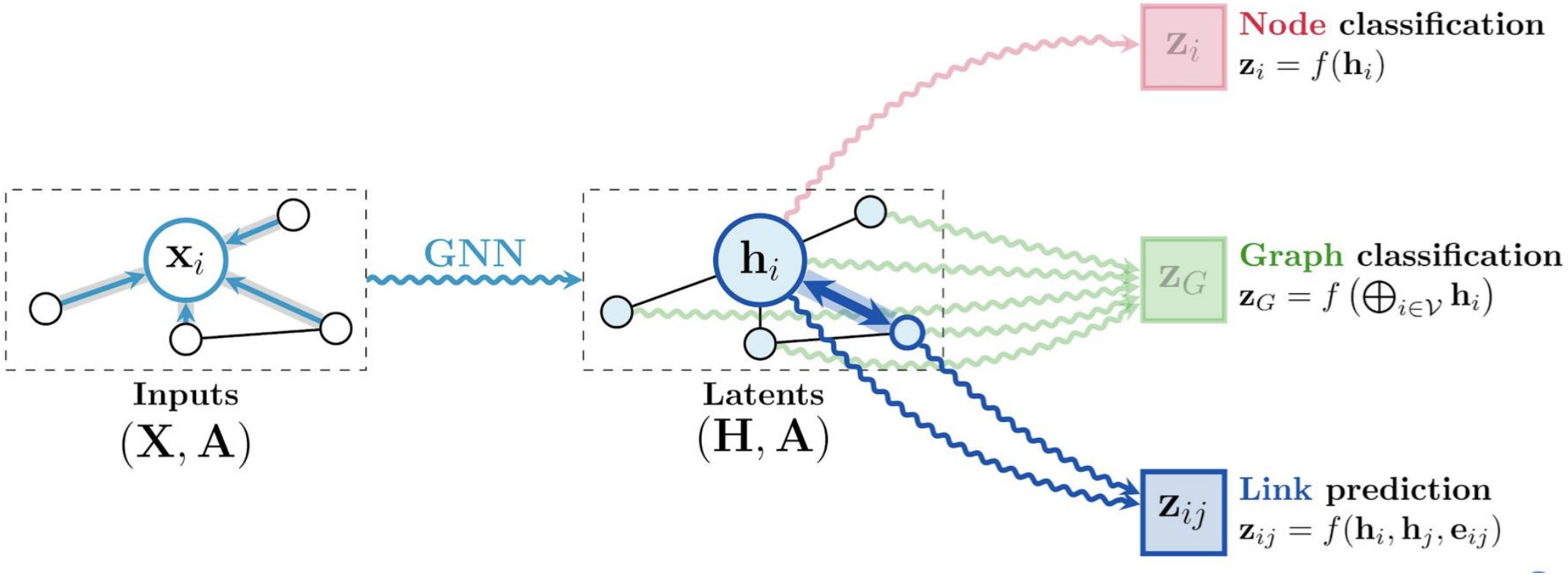
Credit: [5]

Setup



Credit: [5]

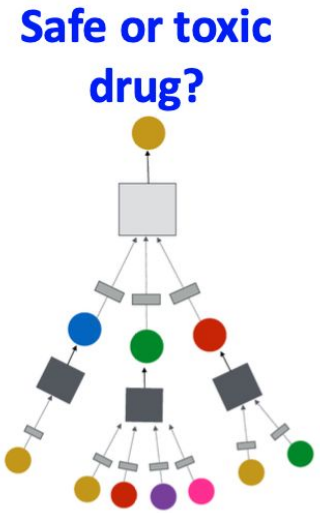
Setup

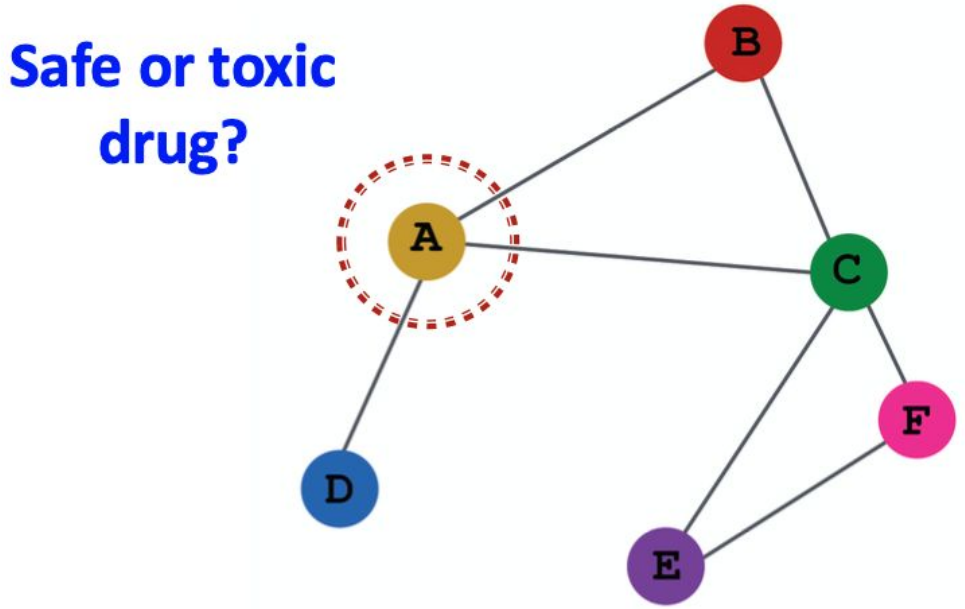


Credit: [5]

How to train a GNN

Directly train the model for a **supervised task** (e.g., node / graph classification).





Unsupervised setting:

Triplet Loss, Contrastive Loss, etc.

Credit: [1]

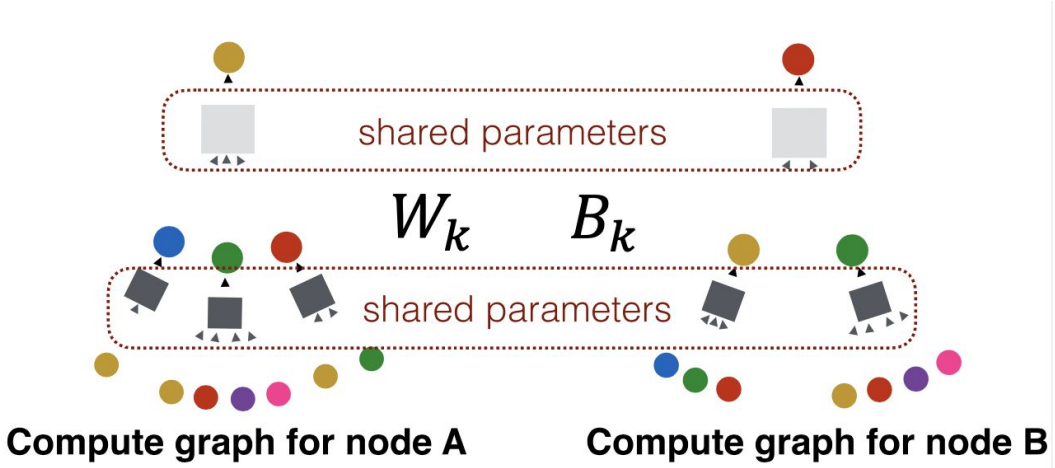
GNN vs node2vec

1. $O(|V|)$ parameters (node2vec)

   The number of parameters is sublinear in $|V|$ (GNN)

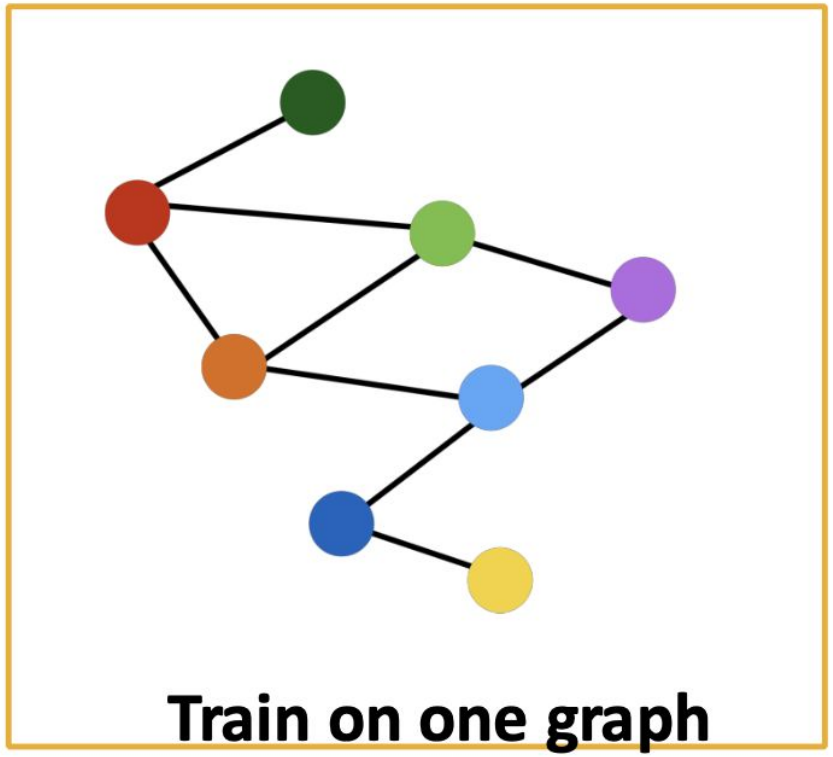
2. No sharing of parameters between nodes (node2vec)

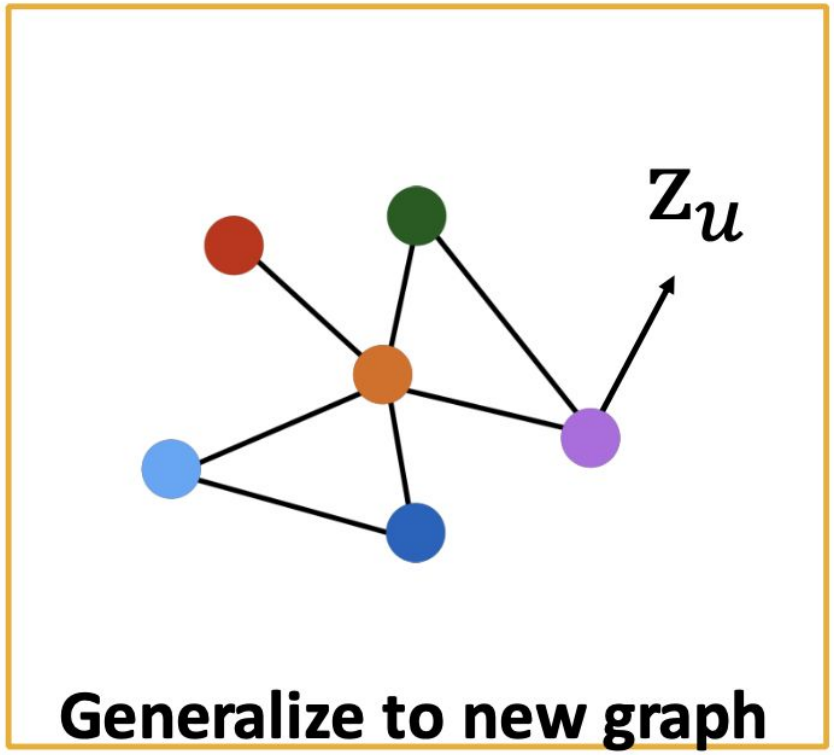
   Shared parameters between nodes (GNN)



Can generalize to unseen nodes!

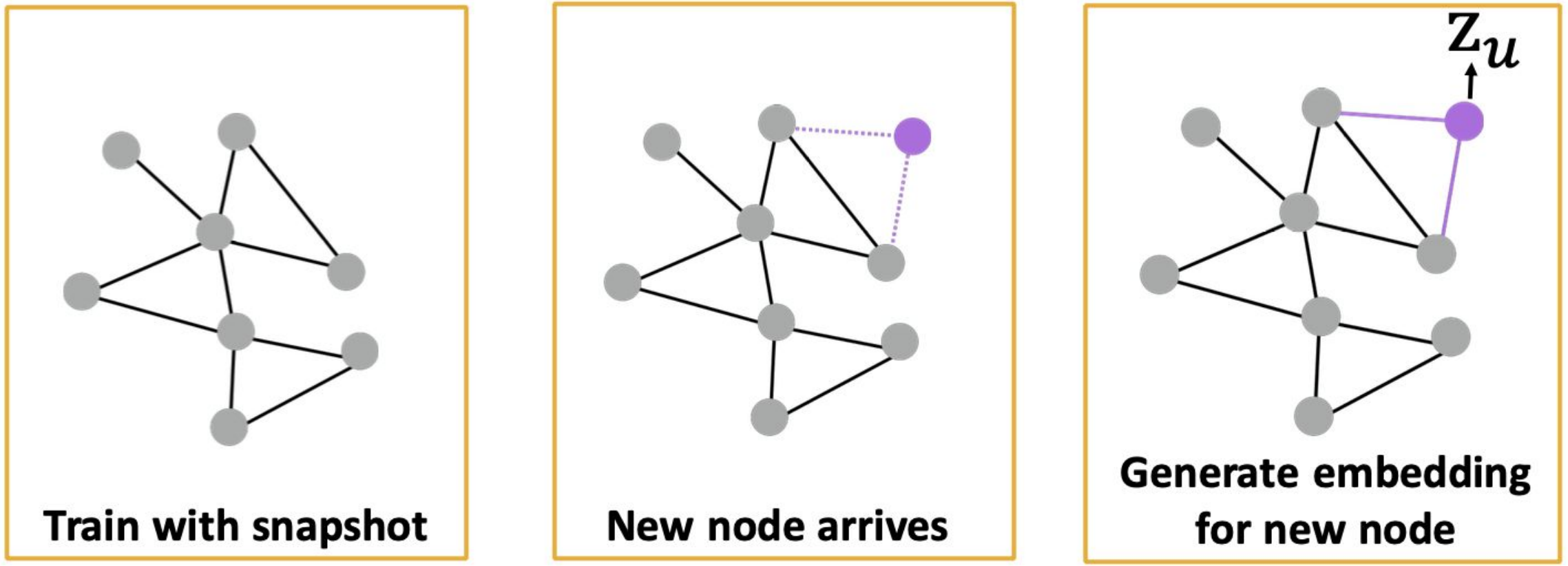
New Graphs





E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B.

New Nodes



Relevant for Evolving Networks: Social Networks, Collaboration Networks, etc.

GNN vs node2vec

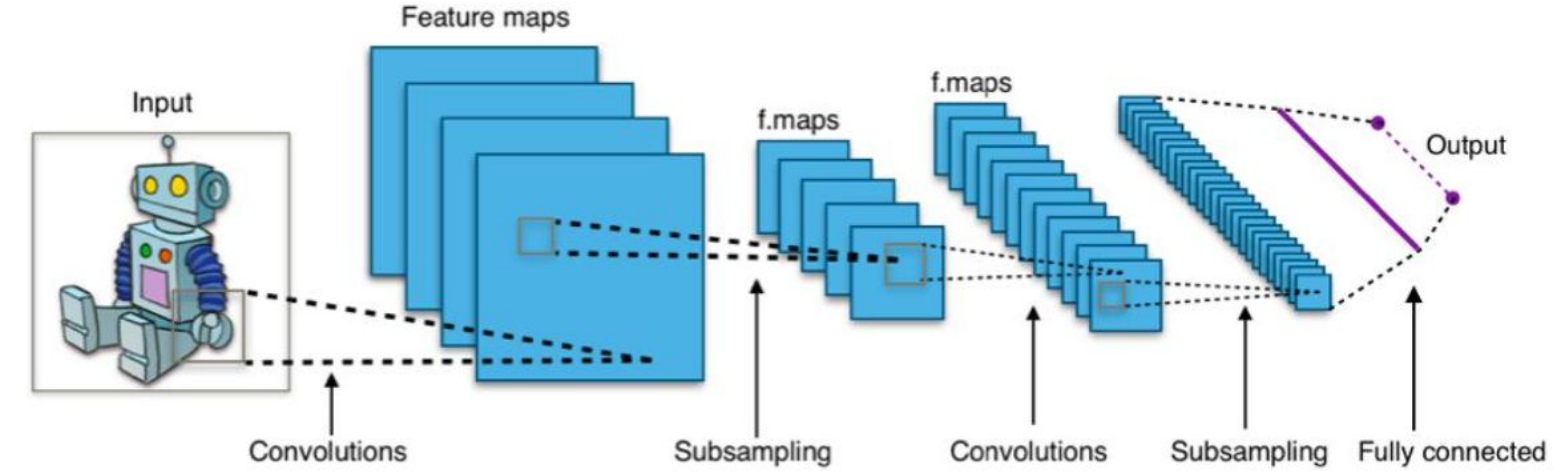
1. Can not utilize directly node, edge and graph features (node2vec)

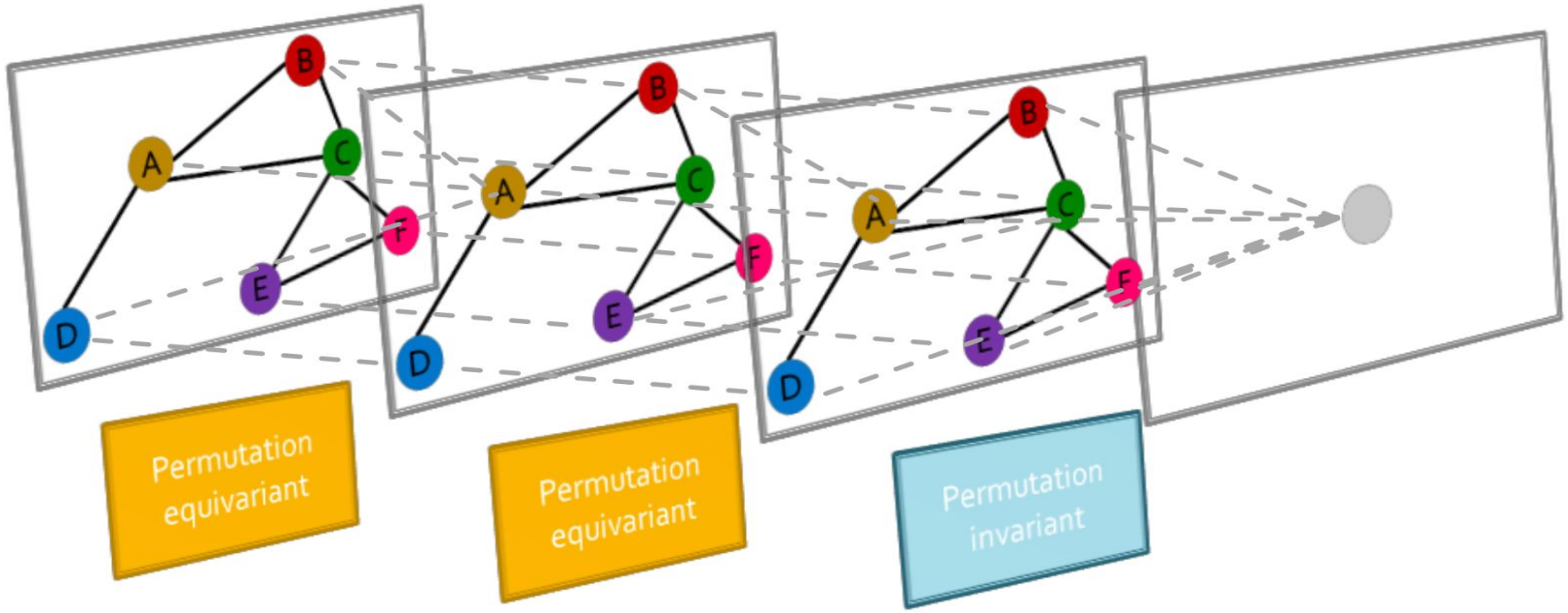
   Use the node / edge / graph features as an input (GNN)

2. Sensitive to the walk length, p/q choice (node2vec)

   Use the input graph structure as is (GNN)

GNN vs CNN





Credit: [1]

GNN vs CNN

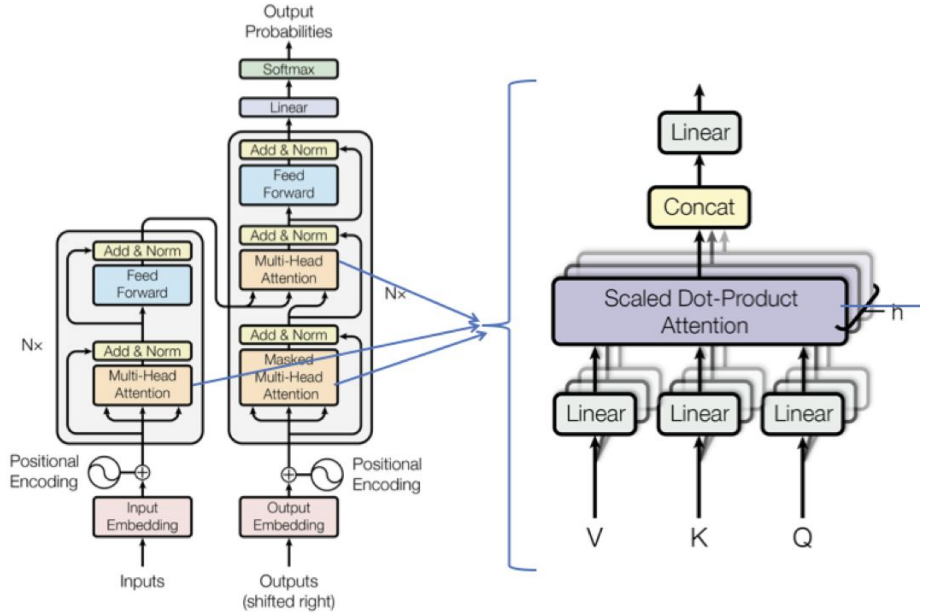
CNN can be seen as a **special GNN** with fixed neighbor size and ordering:

- The size of the filter is pre-defined for a CNN.
- The advantage of GNN is it processes arbitrary graphs with different degrees for each node.
- CNN is not permutation equivariant in terms of convolution - it relies on the local spatial context.
  - Switching the order of the pixels leads to different outputs.
- Some operations in CNN are permutation invariant (average / max pooling).

GNN vs Transformer

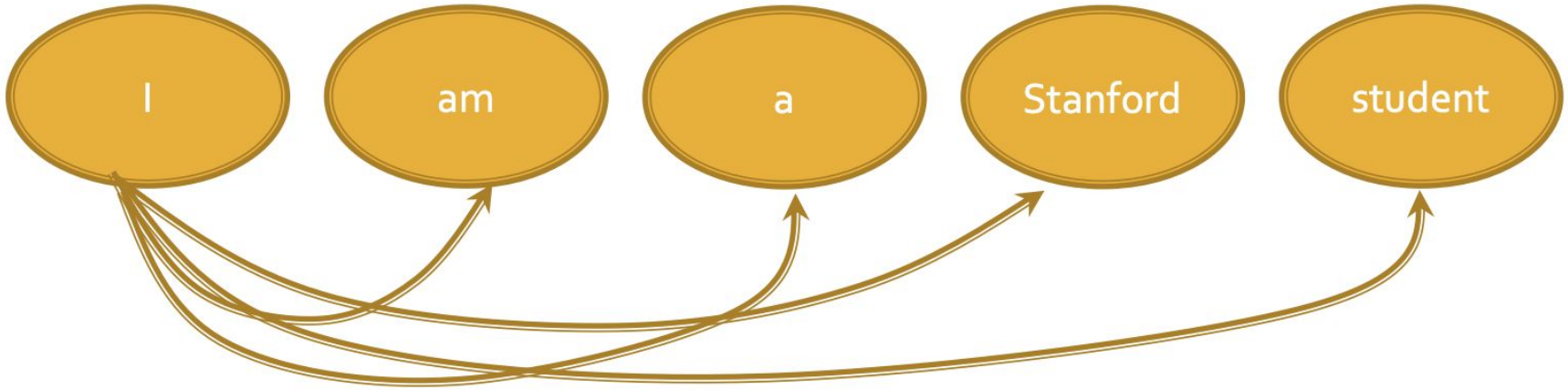
Transformers **are** Graph Neural Networks!

- Fully-connected graph
- Attentional flavour



Key component: self-attention mechanism

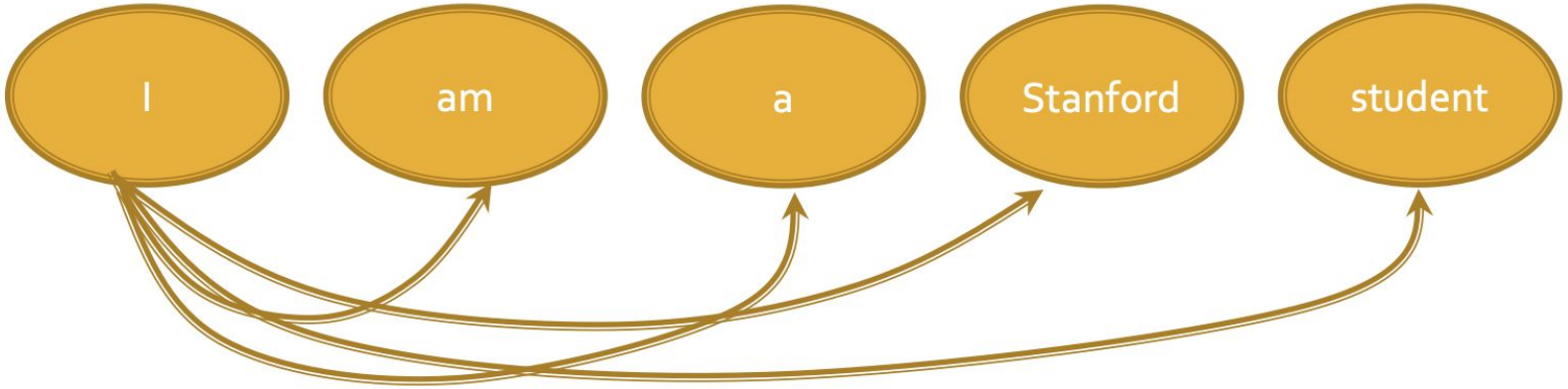
Every token/word attends to all the other tokens/words via matrix calculation.

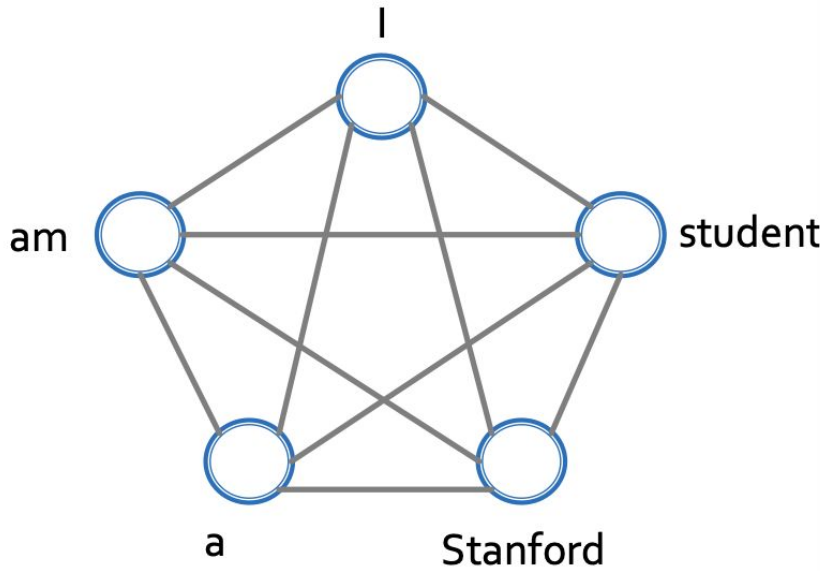


Credit: [1, 13]

GNN vs Transformer

Transformer layer can be seen as a **GNN** that runs on a **fully-connected** "word" graph.





The sequential structural information is injected through the positional embeddings → dropping them leading to fully-connected GAT.

Without positional embeddings → Transformer layer is permutational invariant/equariant.

Credit: [1, 13]

Limitations

1. Over-smoothing: after several iterations of GNN message passing, the representations for all the nodes in the graph can become very similar to one another.
2. It is not straightforward, how to apply GNN to dynamics graphs.
3. For large graphs, capturing structural similarity could still be a problem.
4. Scalability, training instability, adversarial attacks - but most models face the same issues.

Take-Home Messages

1. We discussed ideas behind GNN: connection with CNN; permutation invariance and equivariance properties.
2. Deep Learning on Graphs:
    a. Multiple layers of embedding transformation;
    b. At every layer, use the embedding at previous layer as the input;
    c. Aggregation of neighbors and self-embeddings.
3. GNN Layer:
    a. Aggregation and Update;
    b. Various GNN architectures - GCN, GraphSAGE, GAT and examples.
4. Connections with the other DL models.

Credit: [1]

BIOMEDICAL INFORMATICS

Slides & Image Credits

1. CS224W: Machine Learning with Graphs
2. Graph Representation Learning Book
3. DeepWalk: Online Learning of Social Representations
4. node2vec: Scalable Feature Learning for Networks
5. Everything is Connected: Graph Neural Networks
6. Understanding Convolutions on Graphs
7. Zero-Padding in Convolutional Neural Networks
8. Conv gifs
9. Semi-Supervised Classification with Graph Convolutional Networks
10. Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges
11. Inductive Representation Learning on Large Graphs
12. Graph Attention Networks
13. Attention Is All You Need