IDC410
A course on Image Processing and Machine Learning
(Lecture 22)

Shashikant Dugad,

IISER Mohali

Graphical Neural Network

GNN Lectures adapted from following References

1. **Youtube Lectures given by Petar Velickovic on YouTube:**

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

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

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

2. **Other Youtube Videos**

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

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

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

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

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

3. Notes:

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

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

Graph Neural Network (GNN)

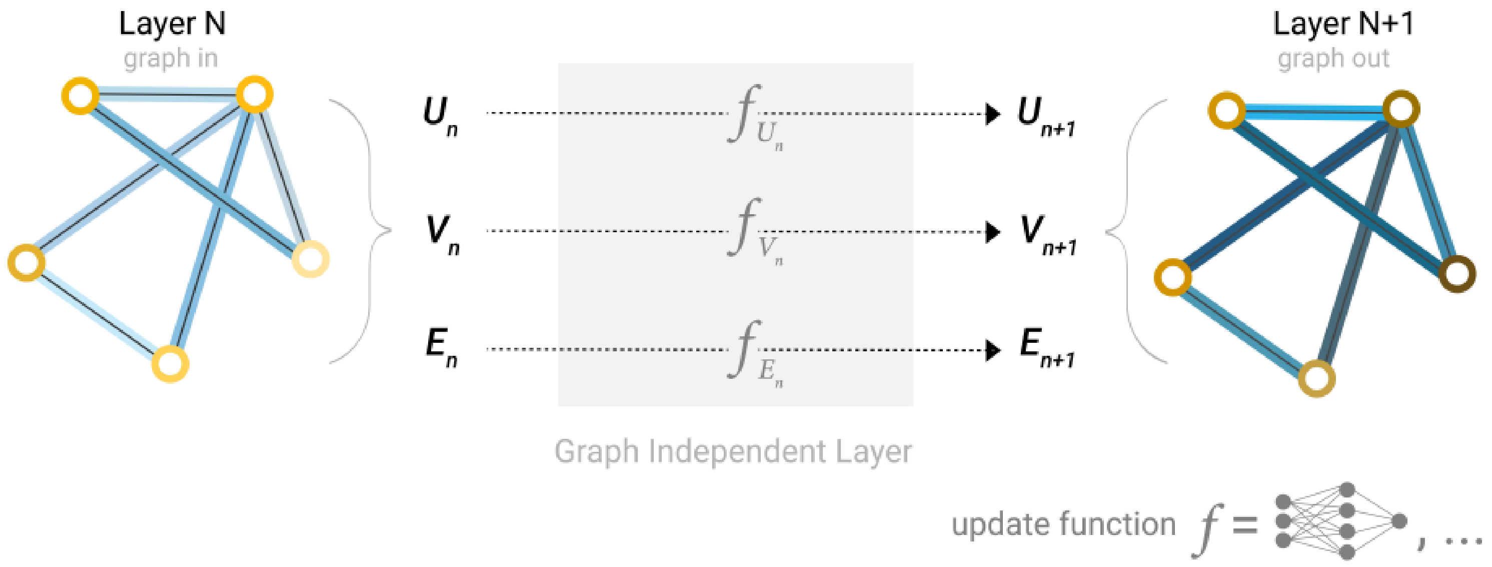
- Describe graph in a matrix format which is permutation invariant. Graph can be uniquely defined with following matrices
  - a) Adjacency Matrix, b) Node Feature matrix and c) Edge Feature Matrix. Degree Matrix can be derived from the Adjacency Matrix
- The ***graph neural networks (GNNs)*** used to solve graph prediction tasks provide formalism for optimizable transformation on all attributes of the graph (nodes, edges, global-context) with a compliances of ***preserving graph symmetries*** (permutation invariances)
- ***Example:*** Build GNNs using the ***message passing neural network*** framework (Gilmer et al.) using the ***Graph Nets architecture*** schematics introduced by Battaglia et al

Simplest GNN Architecture

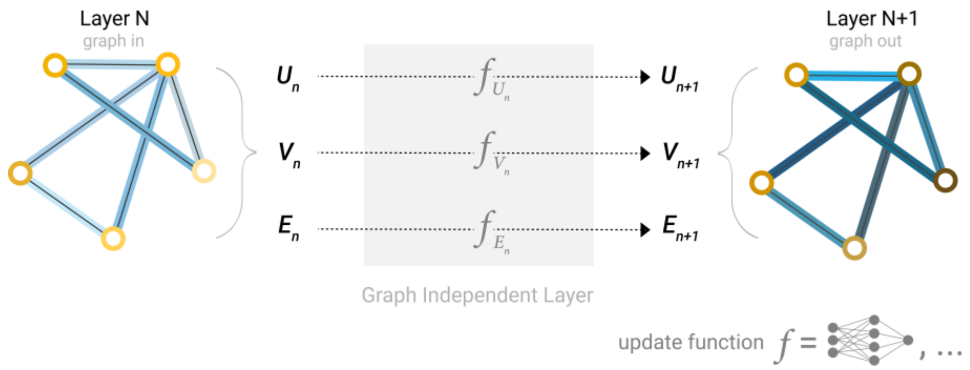
- GNNs adopt a ***graph-in, graph-out*** architecture meaning that these model types accept a graph as an input, with information loaded into its nodes, edges and global-context, and progressively ***transform these embeddings, without changing the connectivity of the input graph***
- In simplest GNN architecture, we learn (update) new embeddings for all the graph attributes (nodes, edges, global) without using the connectivity of the graph by passing it through a multilayer perceptron (MLP) (or differentiable model).
- Output feature vector obtained from this MLP; with the updated embeddings is referred as *learned vector.*
    - Example: *learned node-vector, learned edge-vector and learned graph-vector*

Graph Neural Network (GNN)

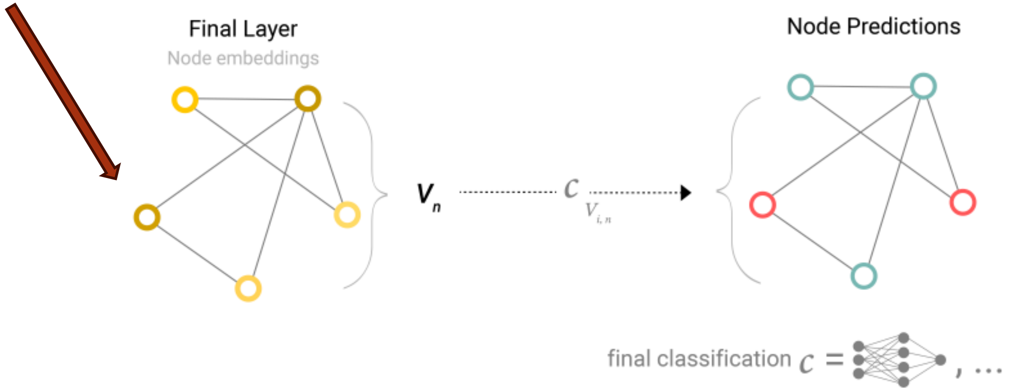
- Note: This operation does not change the connectivity of the input graph, we can describe the output graph with new embeddings with the same adjacency list and the same number of ***learned feature vectors*** (node, edge, graph) as the input graph.



GNN: Node Level Binary Predictions



Update Function f



Classification C

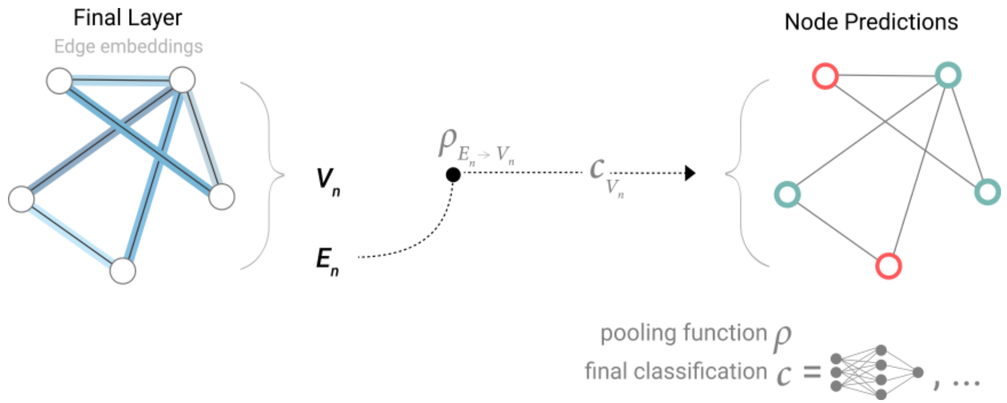
GNN: Node Level Binary Predictions by Pooling

- Binary predictions on nodes with the graph already containing the node level information can be obtained updating nodes and feeding final updated embedding of each node to a ***linear classifier*** on as explained before

- However, if we have ***only edge level*** information, but ***no information at node level***, but still want to make predictions on the nodes then we need to have an approach based on the ***edge feature data*** used for nodes prediction. We can do this by ***aggregating (pooling) edge level final learned-vectors*** that are connected to a node with following steps:
  - For each edge-item to be pooled, ***gather*** each of the final edge level embeddings and concatenate them into a matrix.
  - The gathered edge-embeddings are ***aggregated (pooled)***, usually via a ***sum*** (or average, mean etc.) operation and then aggregated edge-embeddings are passed through a ***linear classifier*** to make the node-level prediction using edge-level feature data

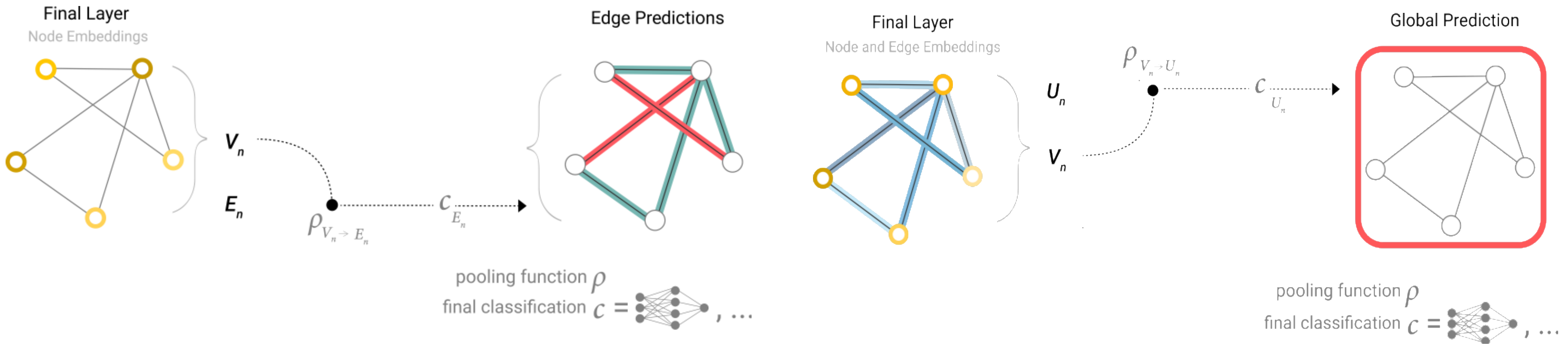
GNN: Node Level Binary Predictions by Pooling

- We represent the ***pooling function*** as **ρ**, and denote that we are gathering (or pooling) information from edges to nodes as $pE_n \rightarrow V_n$.
  - Note: Connectivity information is used only for pooling information, therefore output graph dimensions are still the same is input graph
- Schematics of a model, making node level binary predictions using edge-level features (in absence of node level data) is as shown below



Graph Neural Network (GNN)

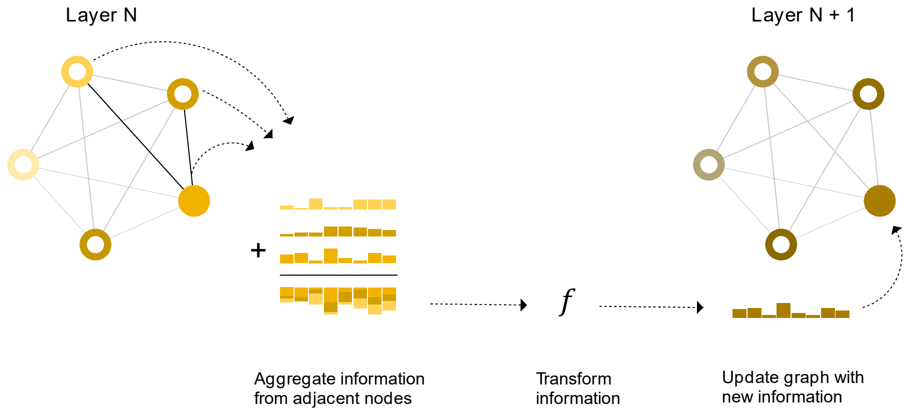
- **Similar procedure can be used for Edge prediction in absence of Edge features and and Global (Graph) prediction in absence of Global features**



GNN: Predictions with Simplest Message Passing

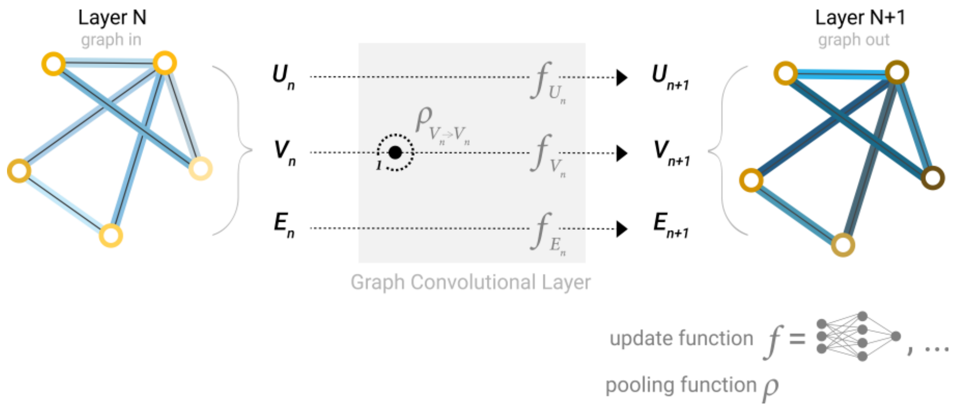
- More accurate predictions can be made by using pooling within the GNN layer by making learned embeddings aware of the graph connectivity
- This can be done using message passing, where neighbouring nodes or edges exchange information and influence each other's updated embeddings.
  - For each node in the graph, *gather* all the neighbouring node embeddings (or messages), which is the function *g* described earlier.
  - *Aggregate* all messages via an aggregate function (like sum).
  - All *pooled messages* are passed through an *update function*, usually a *learned neural network.*
  - Message passing can occur between either nodes or edges.

GNN: Predictions with Message Passing

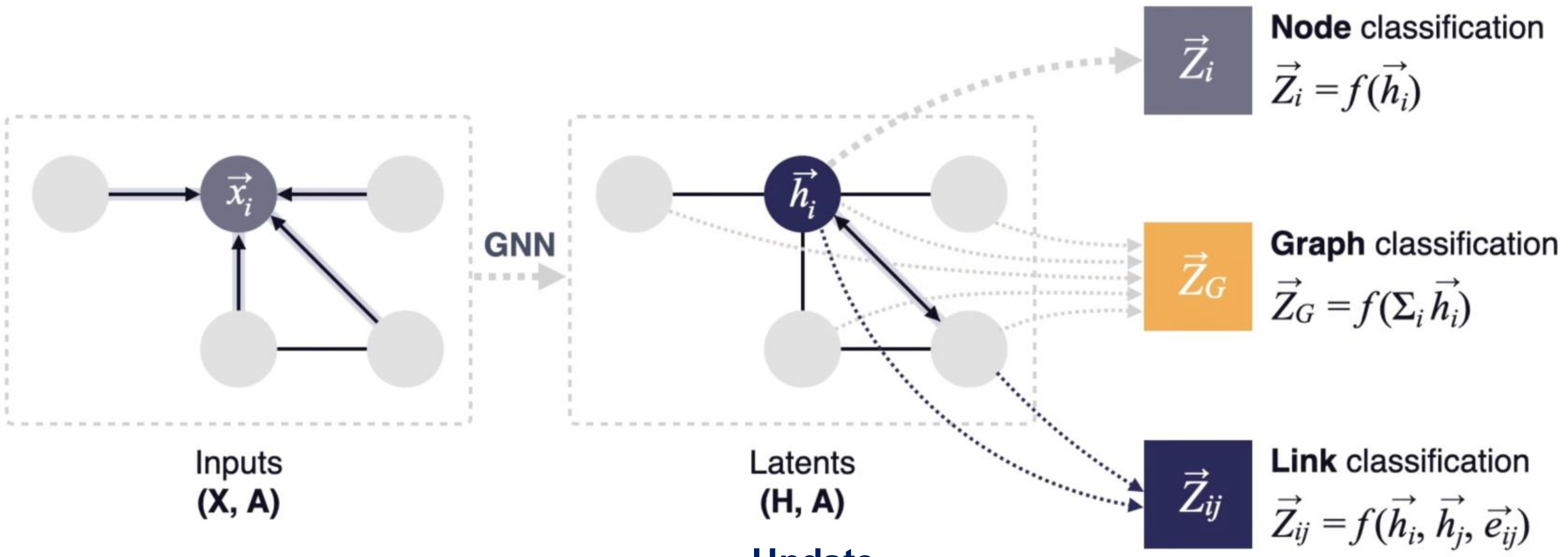

Layer N
Layer N + 1
+
f
Aggregate information from adjacent nodes
Transform information
Update graph with new information

The process of message passing is similar to standard convolution in image processing. Both are operations to aggregate and process the information of an element's neighbours and then update the element's value.

In graphs, the element is a node, and in images, the element is a pixel. However, the number of neighbouring nodes in a graph can be variable, unlike in an image where each pixel has a fixed number of neighbouring elements for a given size of kernel

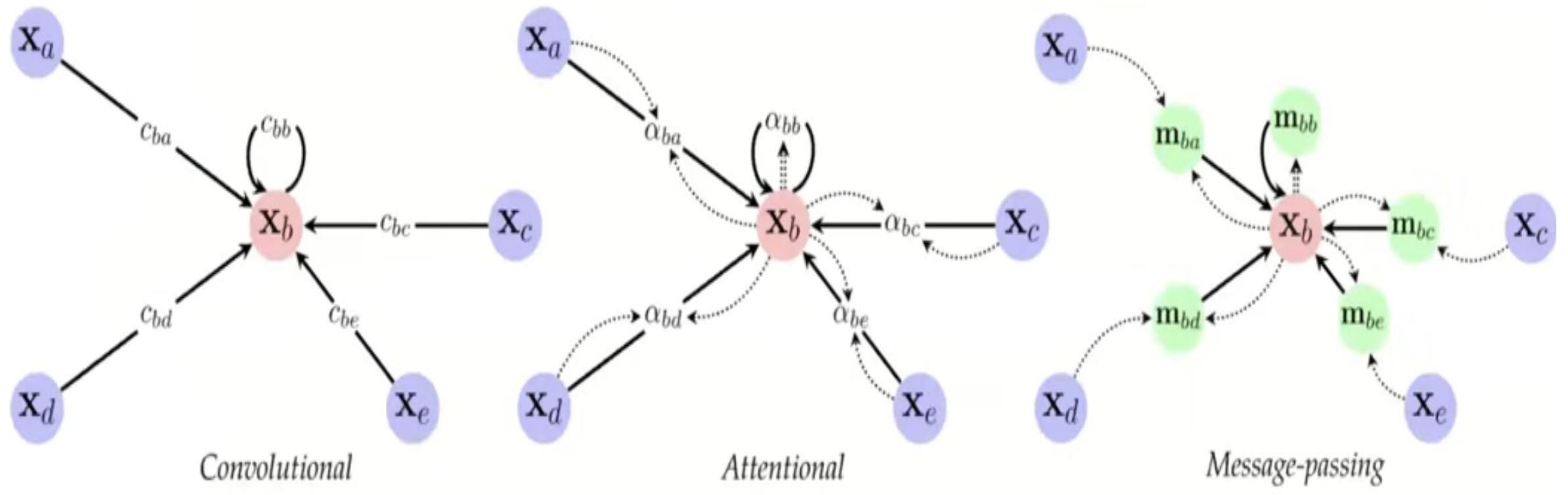


Graph Neural Network (GNN)



Update

Embedding Algorithms



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

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

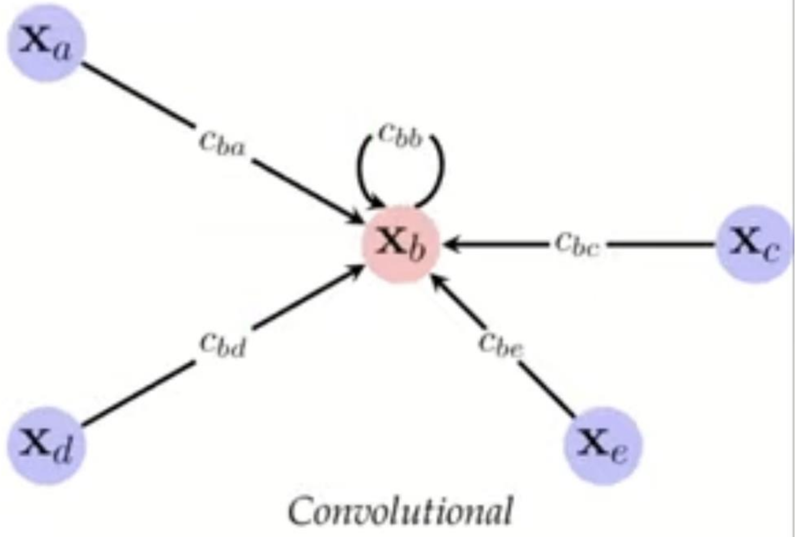
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} \psi(\mathbf{x}_i,\mathbf{x}_j)\right)$$

Convolutional GNN → GCN

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

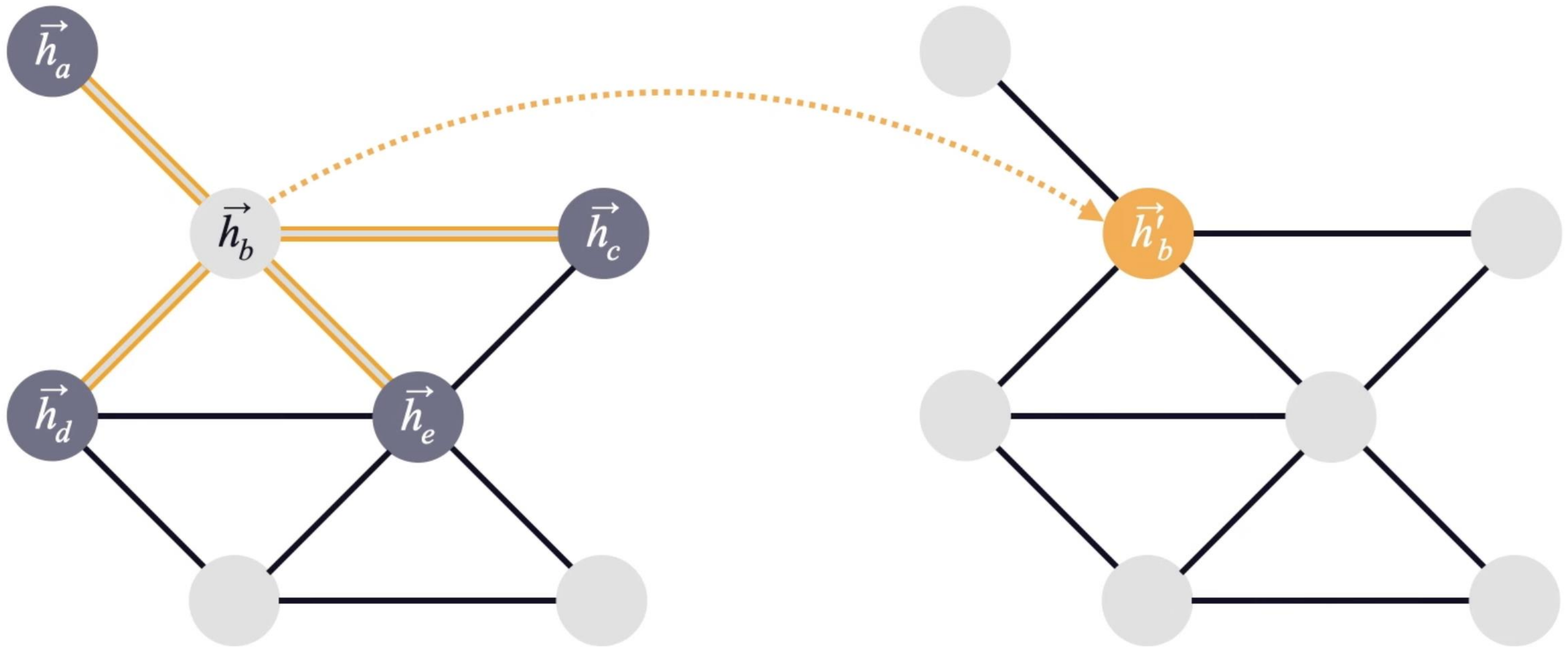
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

- Usually, the weights depend directly on **A**.
    - ChebyNet (Defferrard *et al.*, NeurIPS'16)
    - GCN (Kipf & Welling, ICLR'17)
    - SGC (Wu *et al.*, ICML'19)
- Useful for **homophilous** graphs and **scaling up**
    - When edges encode *label similarity*



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

Convolutional GNN → GCN



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

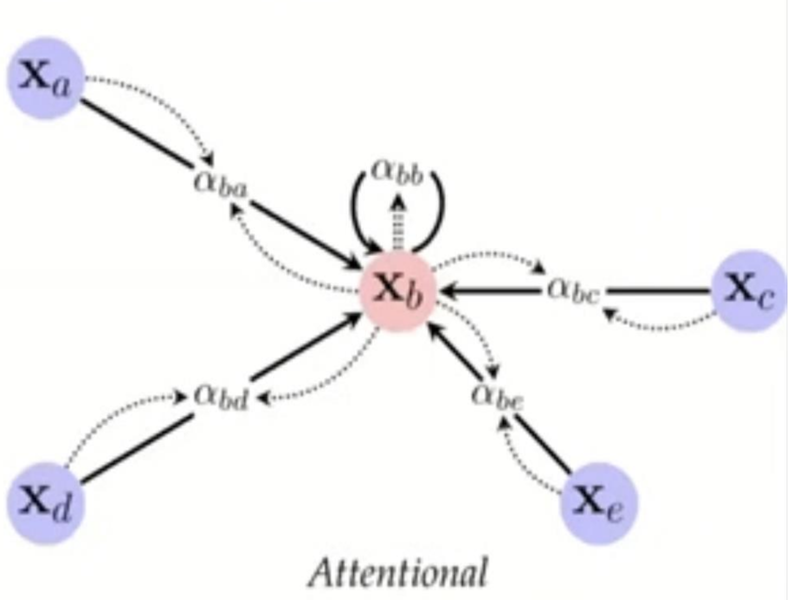
Attentional GNN → GAT

- Features of neighbours aggregated with **implicit** weights (via *attention*)

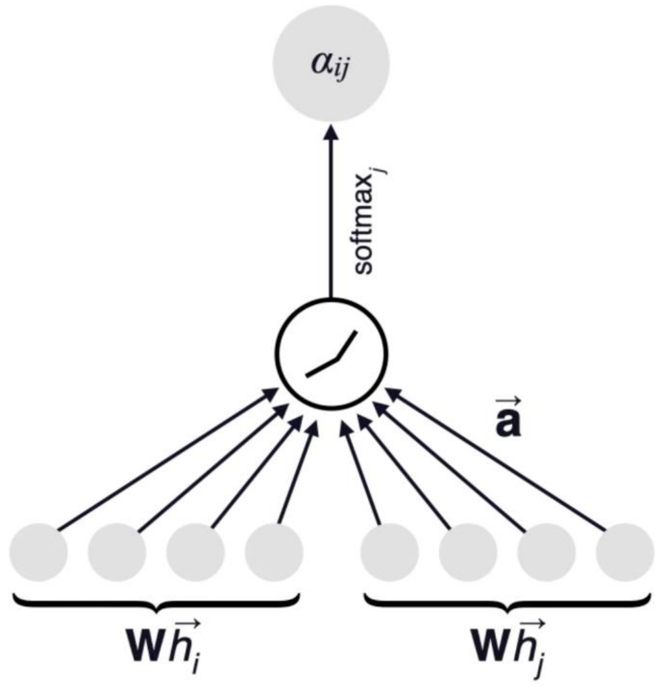
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

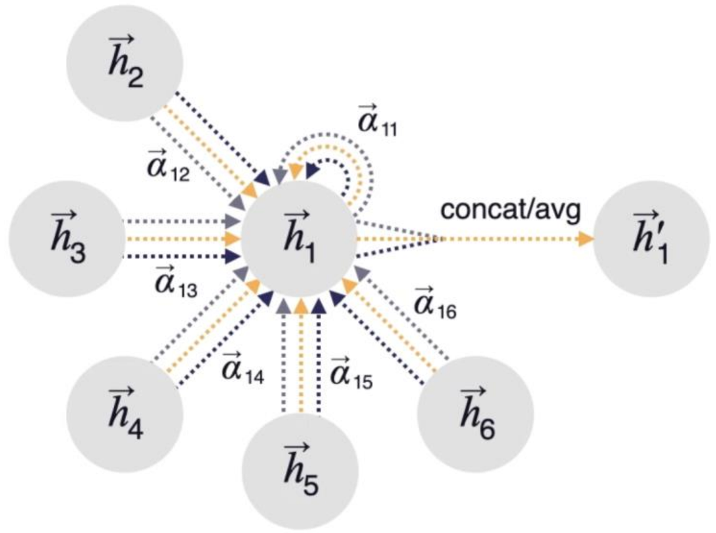
- Attention weight computed as $\alpha_{ij} = a(x_i, x_j)$
  - MoNet (Monti *et al.*, CVPR'17)
  - GAT (Veličković *et al.*, ICLR'18)
  - GaAN (Zhang *et al.*, UAI'18)

- Useful as "middle ground" w.r.t. **capacity** and **scale**
  - Edges need not encode homophily
  - But still compute *scalar* value in each edge



A single GAT step, visualised



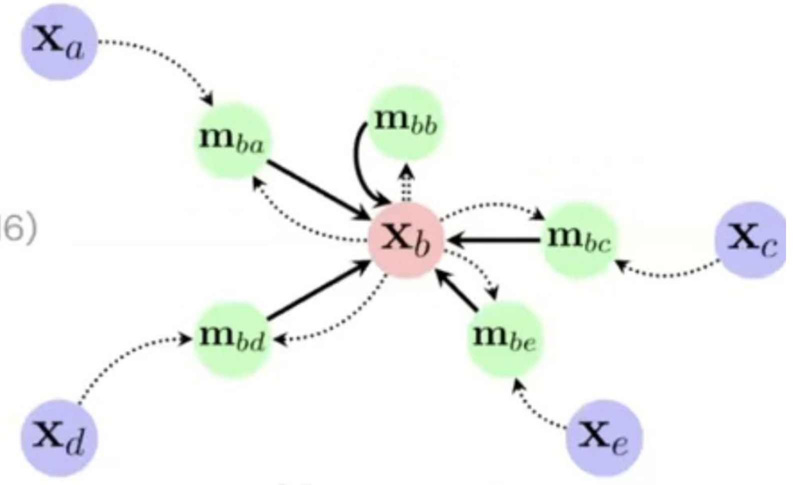


Message Passing GNN

- Compute **arbitrary vectors** (*"messages"*) to be sent across edges

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

- Messages computed as $\mathbf{m}_{ij} = \psi(\mathbf{x}_i, \mathbf{x}_j)$
  - Interaction Networks (Battaglia *et al.*, NeurIPS'16)
  - MPNN (Gilmer *et al.*, ICML'17)
  - GraphNets (Battaglia *et al.*, 2018)

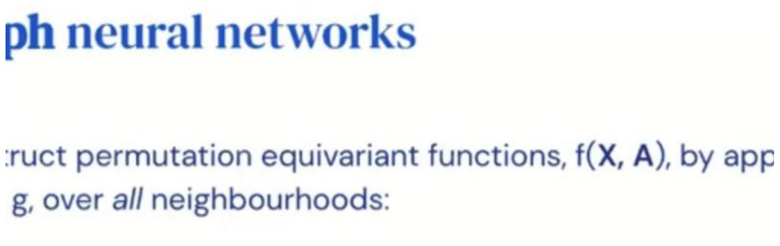


Message-passing

- Most **generic** GNN layer
  - May have *scalability* or *learnability* issues
  - Ideal for *computational chemistry, reasoning* and *simulation*

Convolution Function Construct for Graphs

- Permutation equivariant function *f(X,A)* can be constructed by applying ***local function g*** over all neighborhoods as shown:



ph neural networks

:ruct permutation equivariant functions, f(**X**, **A**), by app
g, over *all* neighbourhoods:

- To ensure equivariance of *f(X,A)*, the function *g* should not depend on the order nodes in X_{Ni}
  - Hence *g* should be ***permutation invariant***

Super Adjacency Matrix

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