Graph Neural Networks: From Zero to Hero

2025 ICARC Tutorial Session

Dr. Dharshana Kasthurirathna
Sri Lanka Institute of Information Technology (SLIIT)

Dr. Mahima Weerasinghe
Sri Lanka Institute of Information Technology (SLIIT)

Dr. Dinuka Sahabandu
University of Washington (UoW)

Mr. Jeewaka Perera
Sri Lanka Institute of Information Technology (SLIIT)

Mr. Asiri Gawesha
Sri Lanka Institute of Information Technology (SLIIT)

Mr. Sanka Mohottala
Sri Lanka Institute of Information Technology (SLIIT)

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

ICARC

BRAIN LABS

An Exciting Research Area.....

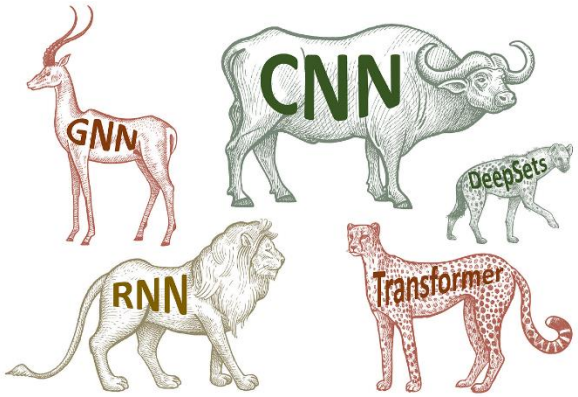


Fig 1: Geometric Deep Learning

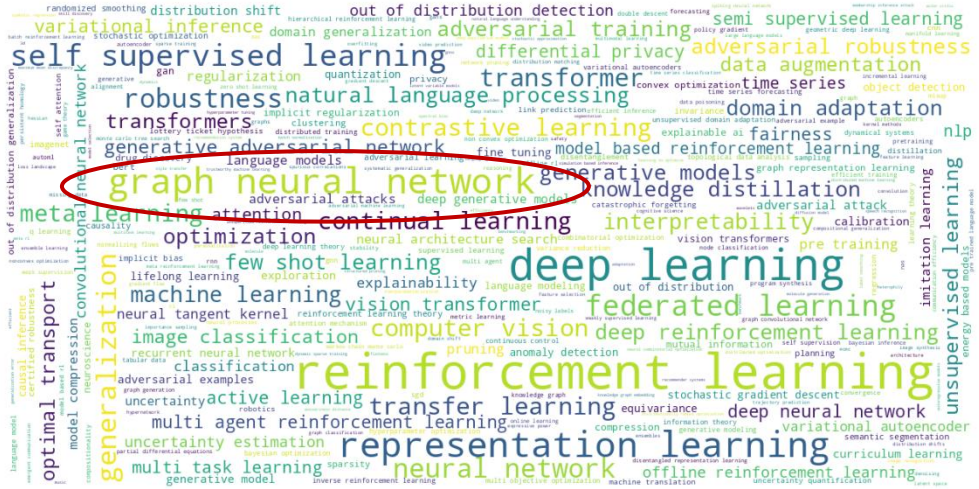


Fig 2: ICLR 2022 key research areas

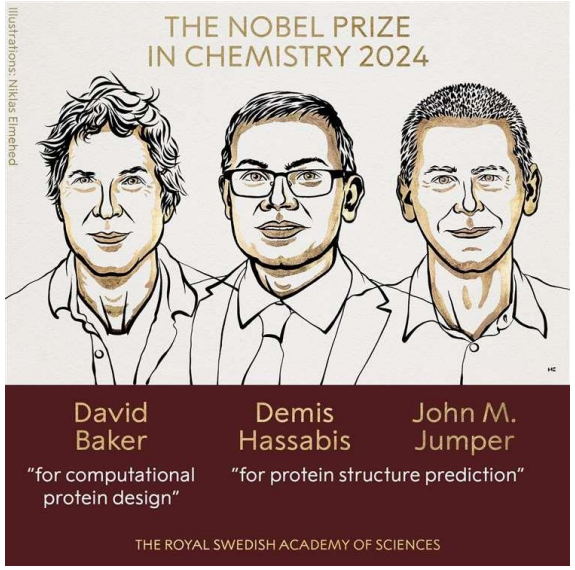


Fig 3: AlphaFold won Chemistry Nobel prize in 2024

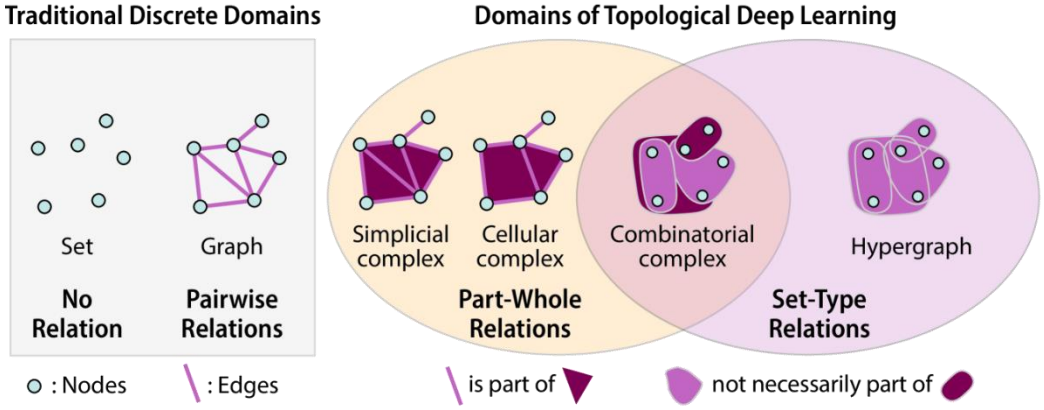
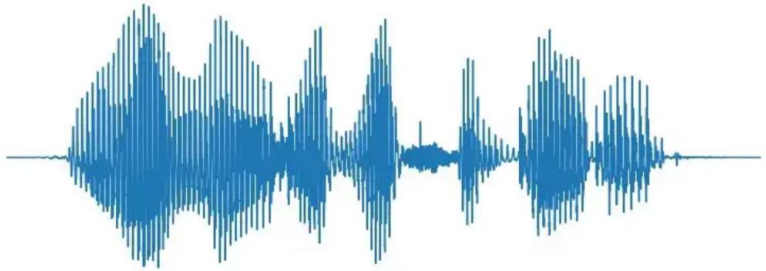


Fig 4: Topological Deep Learning

Fig 5: ICLR 2024 key Research Areas

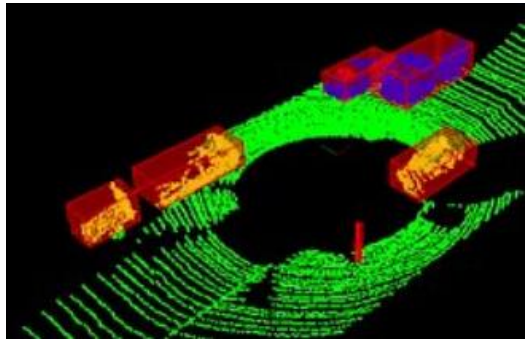
Why GNNs?

- Traditionally , deep learning deals with Euclidian data such as grids, sequences etc.

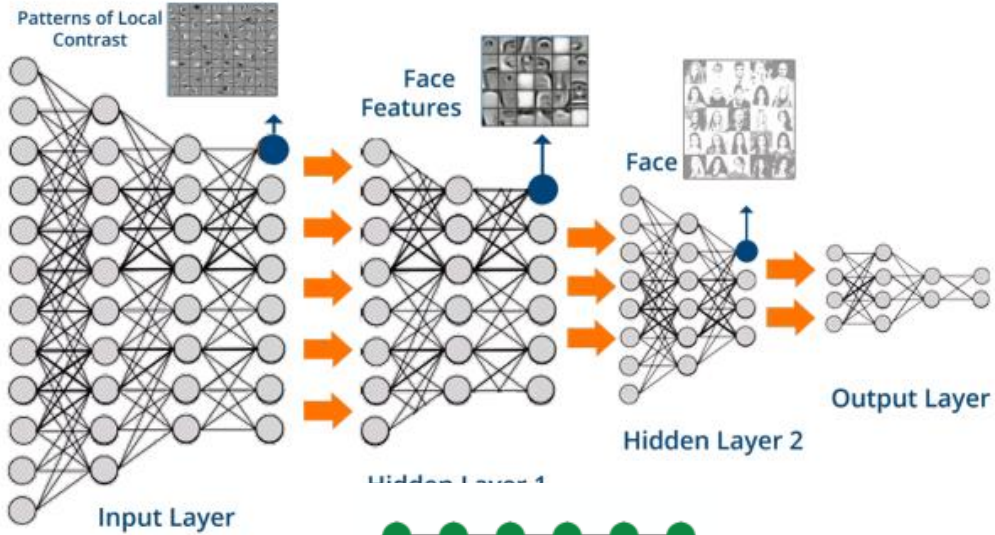


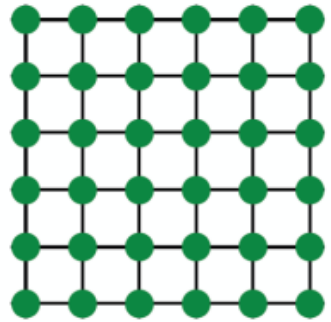
$\mathbb{R} \mapsto \mathbb{R}$: Signal , Speech etc.

$\mathbb{R}^2 \mapsto \mathbb{R}$: Images



$\mathbb{R}^3 \mapsto \mathbb{R}^n$: Point clouds, Volumes

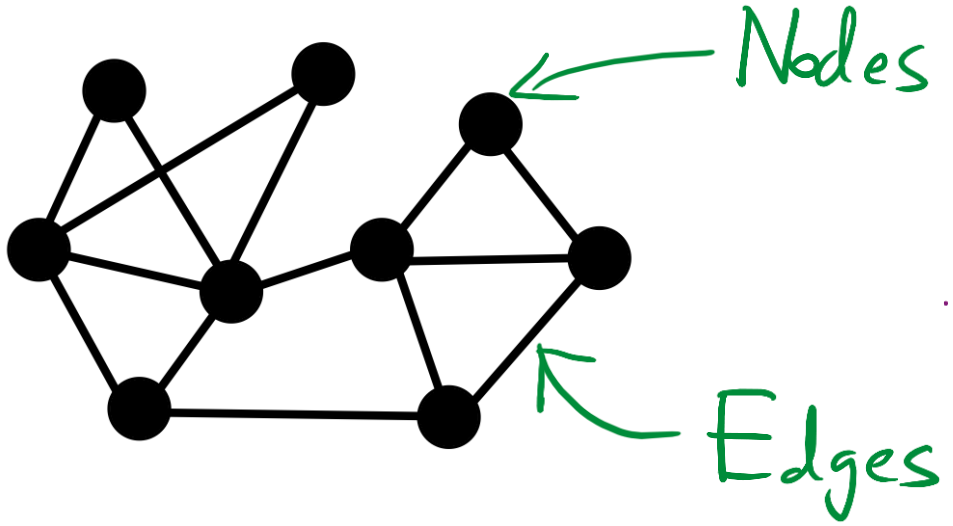




Images

Definitions

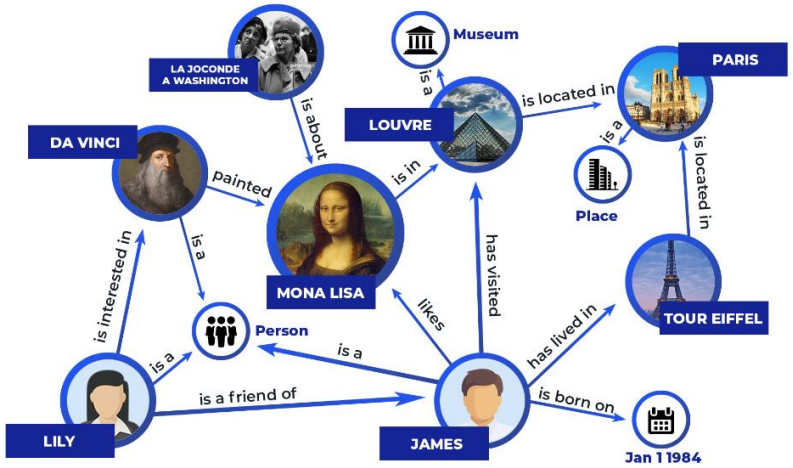
- Graphs are a general language for describing and analyzing entities with relations/interactions.
- Networks are real-world instantiations of graphs with node and/or edge features.



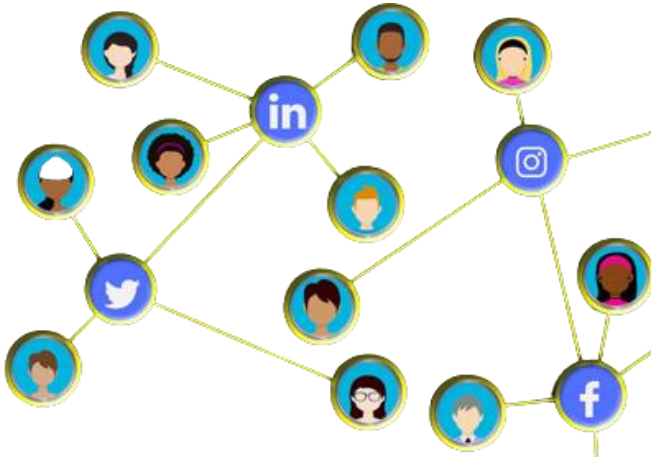
- **Objects:** nodes, vertices — N
- **Interactions:** links, edges — E
- **System:** network, graph — $G(N,E)$

Networks are complex.

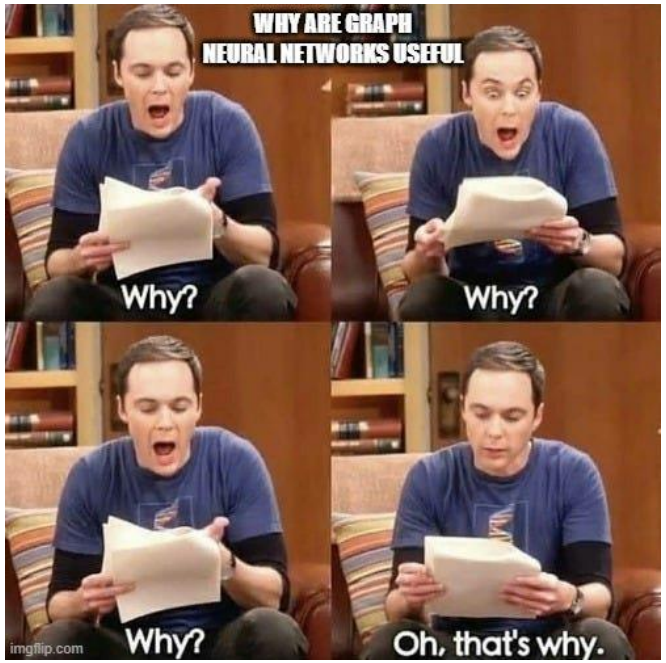
- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)



Knowledge Graph



Social Networks

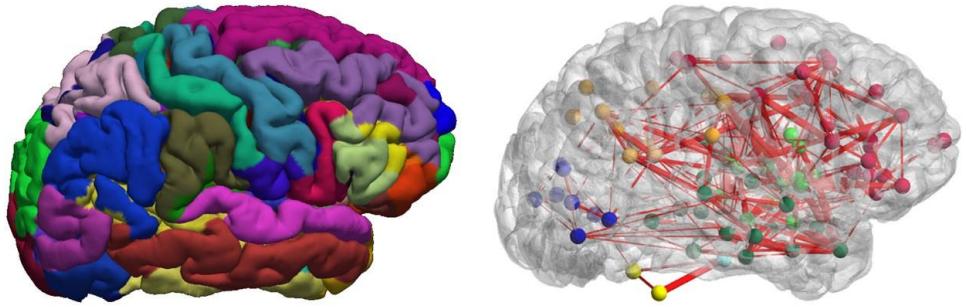


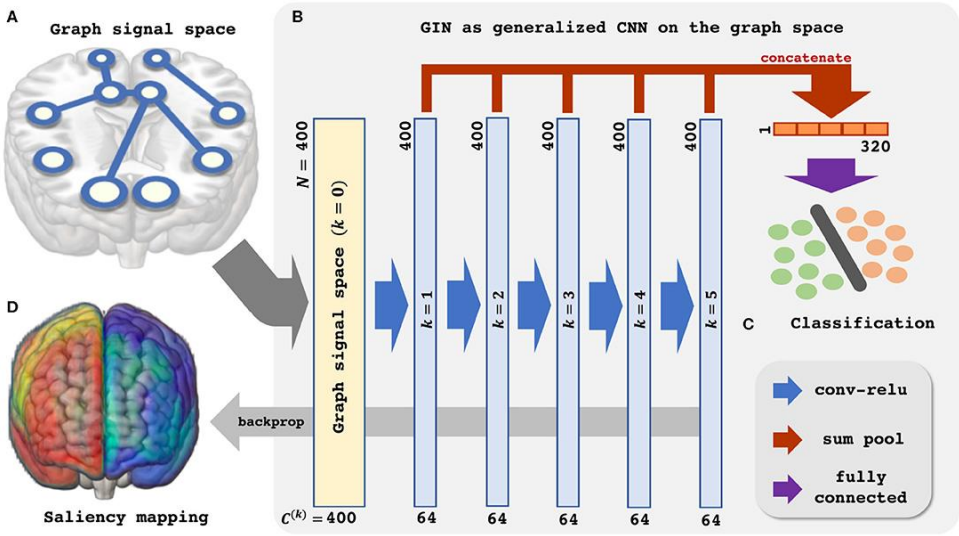
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Graph neural network applications

- Graphs can be used to model brain connectomes taken from fMRI/DTI data

- GNN can be used for fMRI data analysis by modeling the functional connectivity of brain as a graph structure.

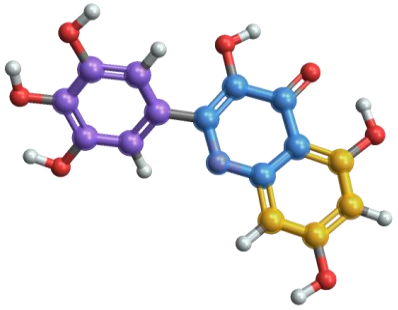


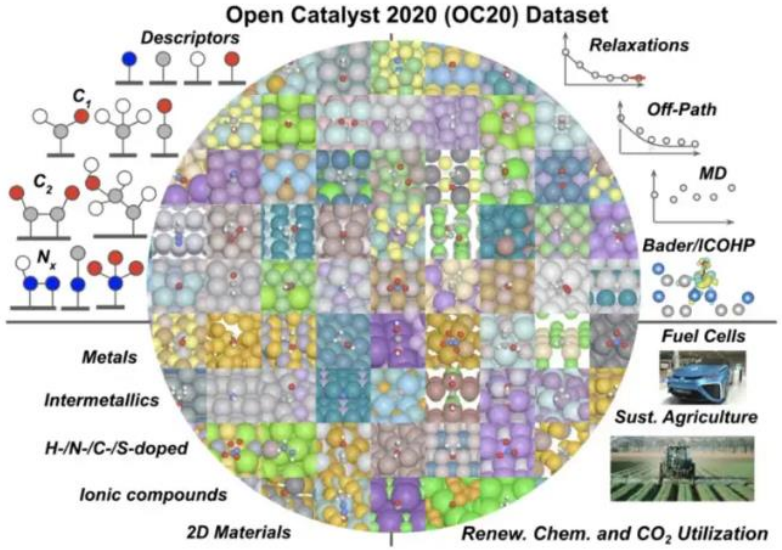


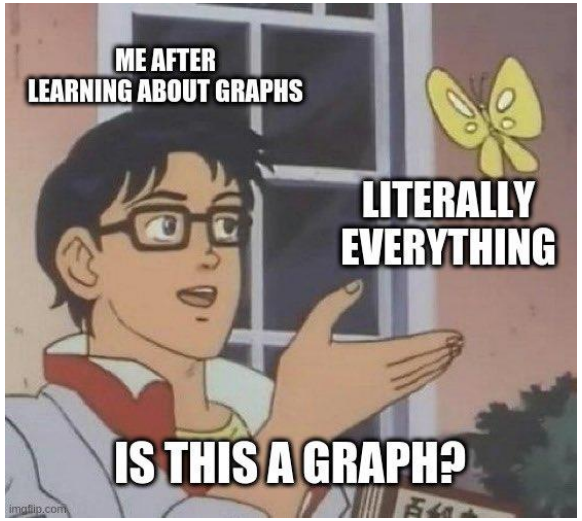
Graphs are also used in Chemistry and Pharmacology

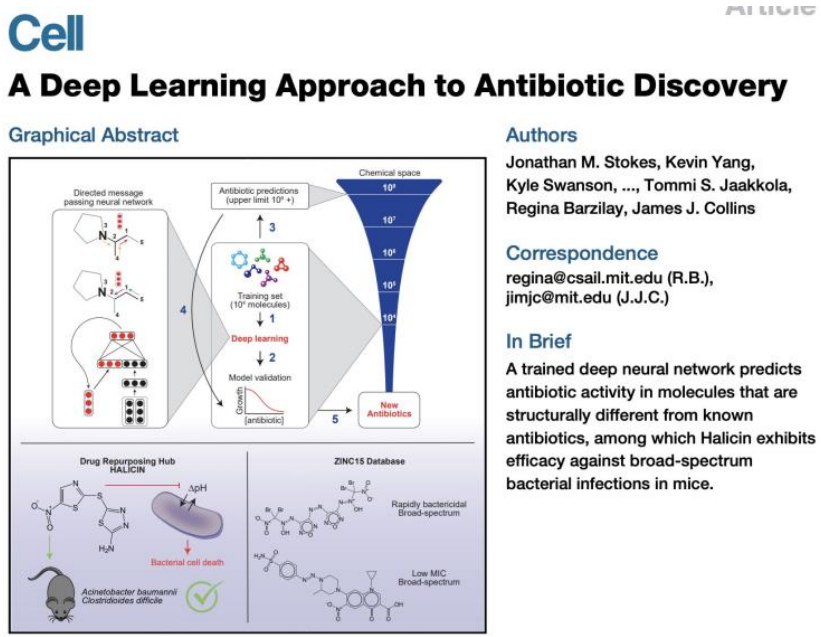
Modeling molecules as graphs data, they can be used with graph neural networks to do

- Molecule Analysis
- Drug discovery
- Drug repurposing
- Catalysts discovery








Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract

Authors

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, ..., Tommi S. Jaakkola, Regina Barzilay, James J. Collins

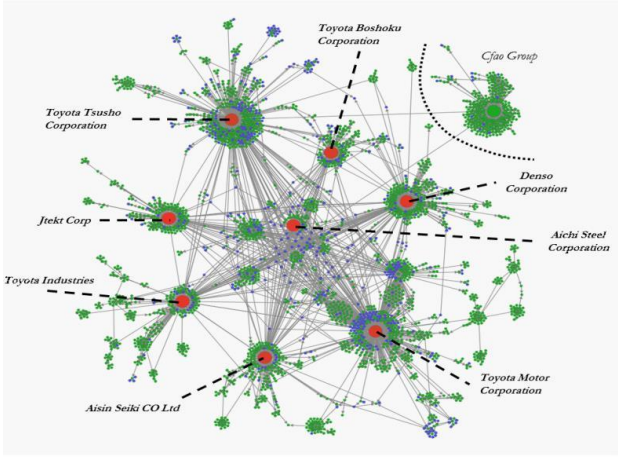
Correspondence

regina@csail.mit.edu (R.B.), jimjc@mit.edu (J.J.C.)

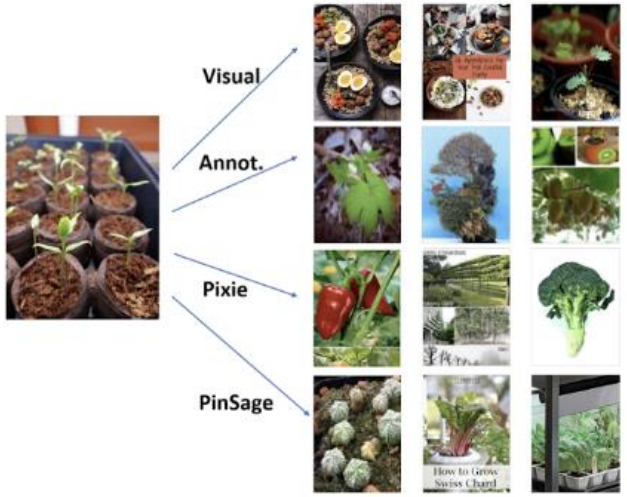
In Brief

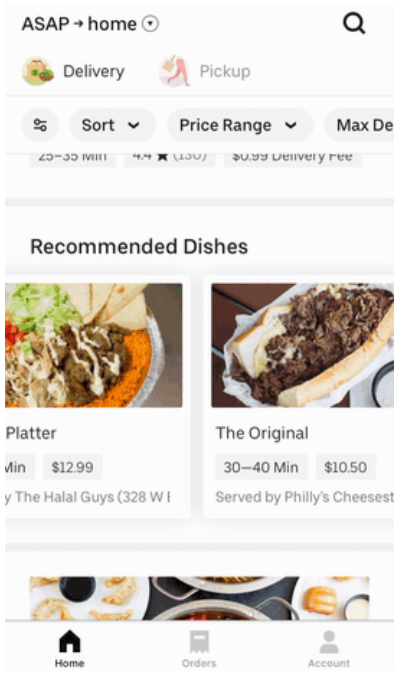
A trained deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which Halicin exhibits efficacy against broad-spectrum bacterial infections in mice.

In financial sector, graphs can be used to model how the markets work as in the case of stock market prediction.

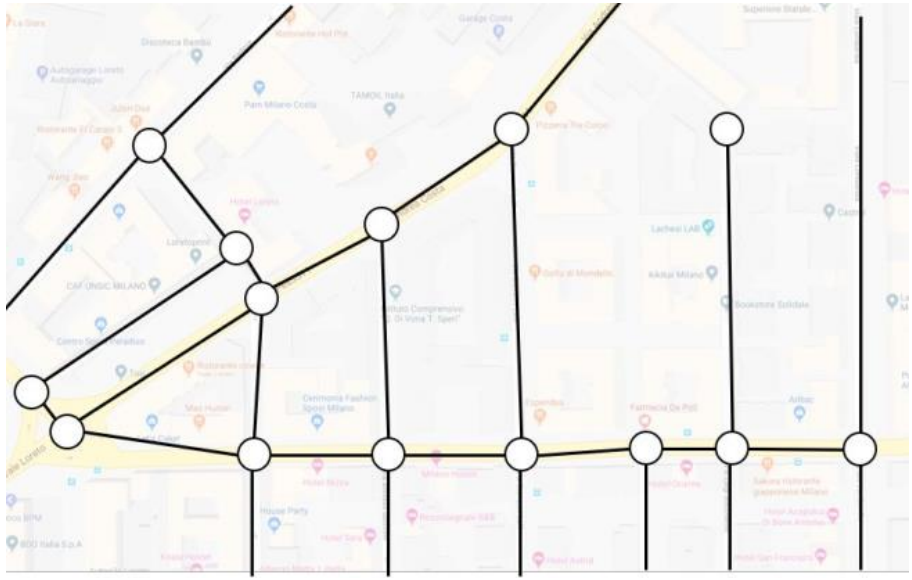


Recommender systems used in Uber Eats and Pinterest uses GNN algorithms.

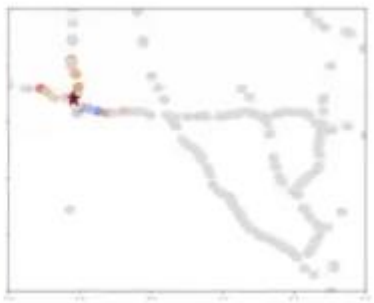


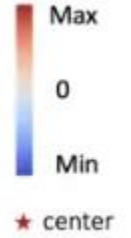


Road systems can also be modeled as graph. In such a system, intersections can be considered as nodes and roads as edges.



GNNs are being used with such data to estimate the time of arrival in Google maps.





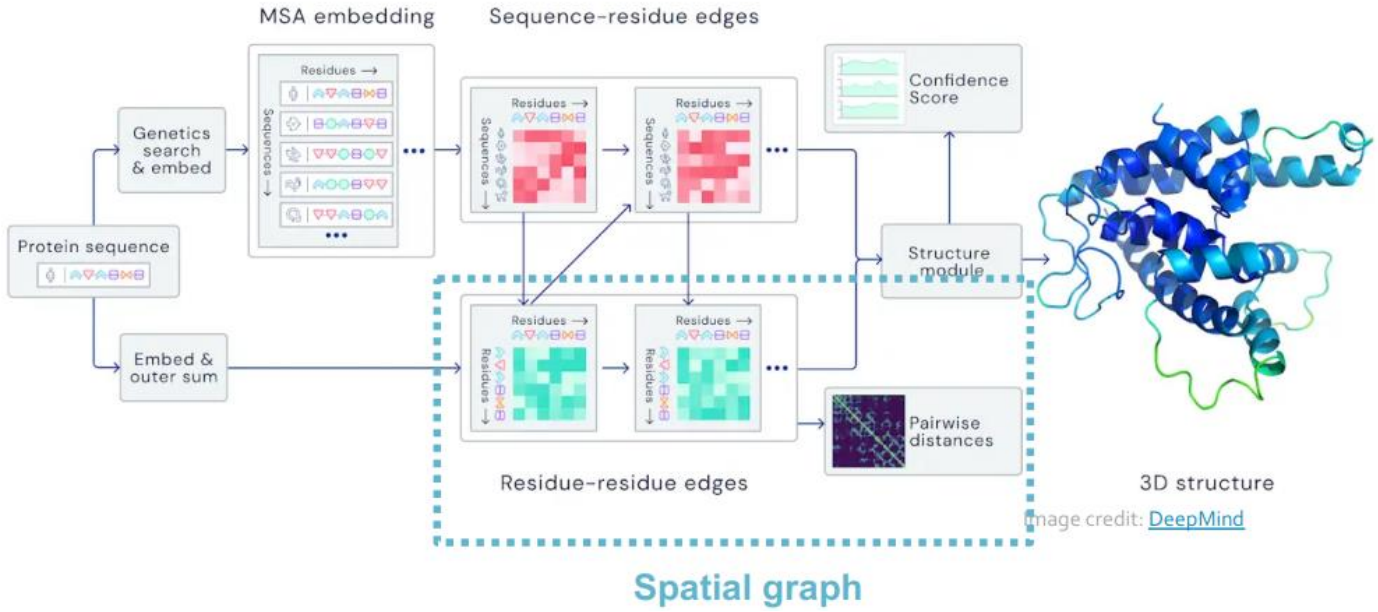
Graph ML Tasks

- **Node classification**: Predict a property of a node
  - **Example:** Categorize online users / items
- **Link prediction**: Predict whether there are missing links between two nodes
  - **Example:** Knowledge graph completion
- **Graph classification**: Categorize different graphs
  - **Example:** Molecule property prediction
- **Clustering**: Detect if nodes form a community
  - **Example:** Social circle detection
- **Other tasks**:
  - **Graph generation**: Drug discovery
  - **Graph evolution**: Physical simulation

Node-level Tasks

Computationally predict a protein's 3D structure based solely on its amino acid sequence

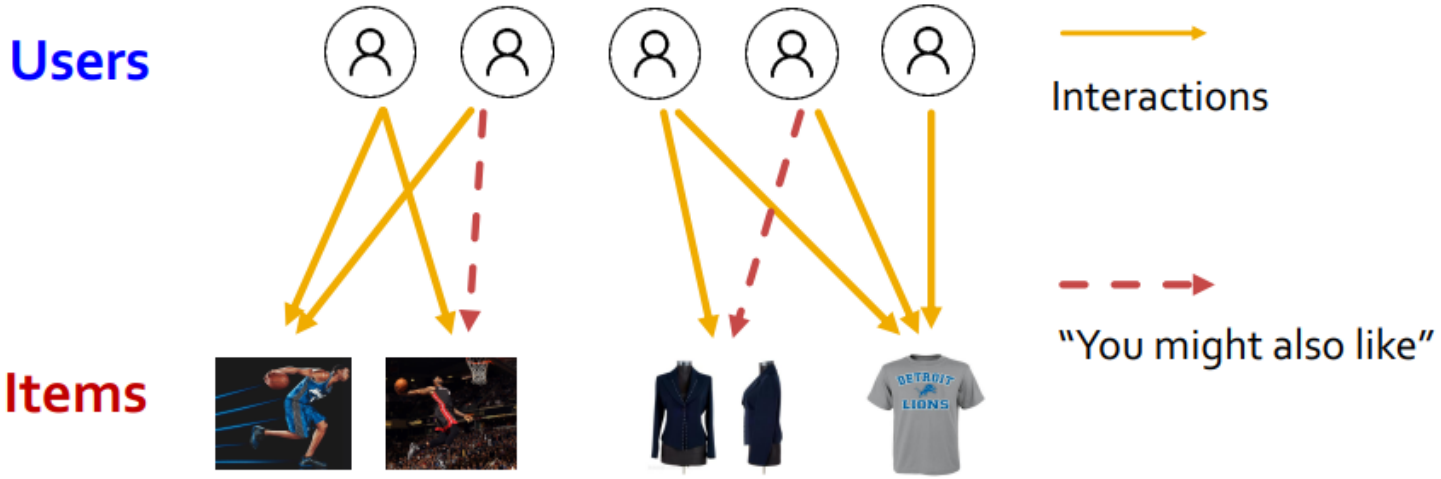
- **Key idea:** "Spatial graph"
  - **Nodes:** Amino acids in a protein sequence
  - **Edges:** Proximity between amino acids (residues)



Edge-level Task

Recommender Systems

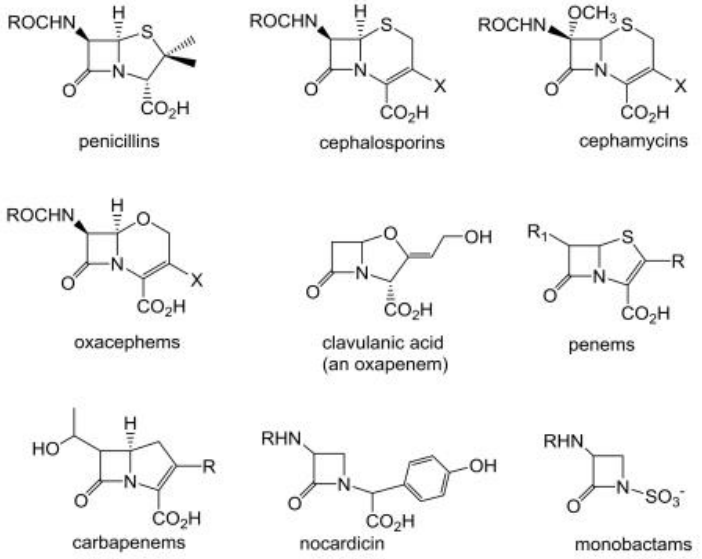
- **Users interacts with items**
  - Watch movies, buy merchandise, listen to music
  - **Nodes:** Users and items
  - **Edges:** User-item interactions
- **Goal: Recommend items users might like**



Graph-level Task

Drug discovery

- **Antibiotics are small molecular graphs**
  - **Nodes:** Atoms
  - **Edges:** Chemical bonds



Konaklieva, Monika I. "Molecular targets of β-lactam-based antimicrobials: beyond the usual suspects." Antibiotics 3.2 (2014): 128-142.

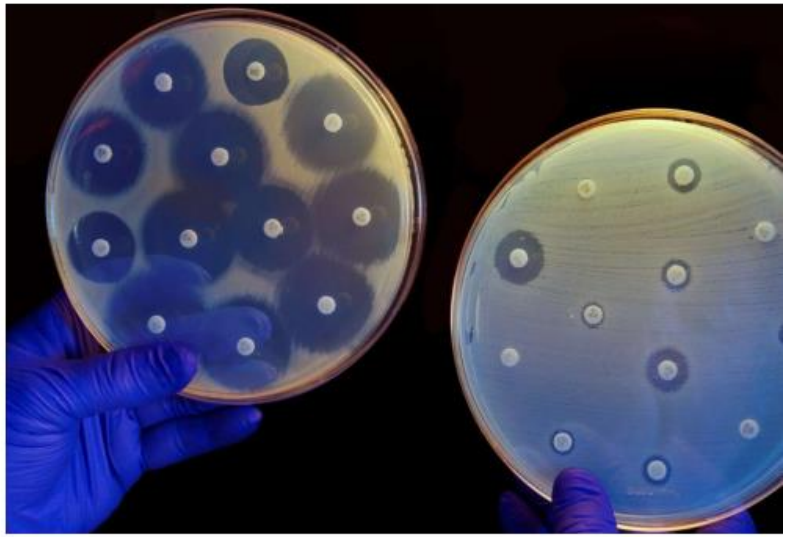


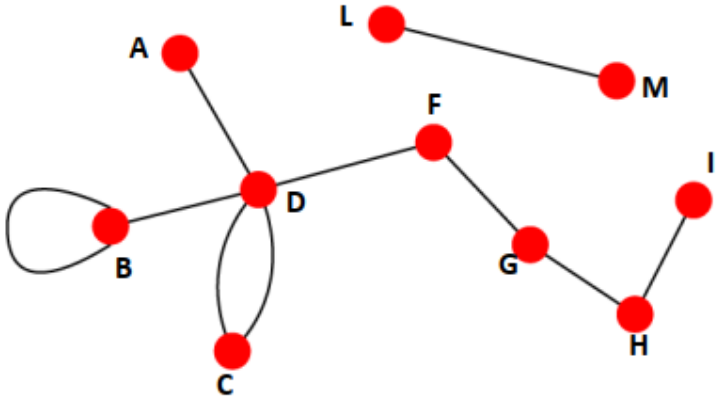
Image credit: CNN

Basic Graph Theory Concepts

Directed vs Undirected Graphs

Undirected

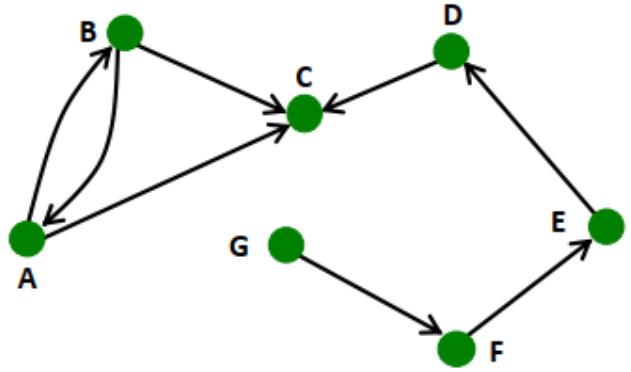
- **Links:** undirected (symmetrical, reciprocal)



- **Examples:**
  - Collaborations
  - Friendship on Facebook

Directed

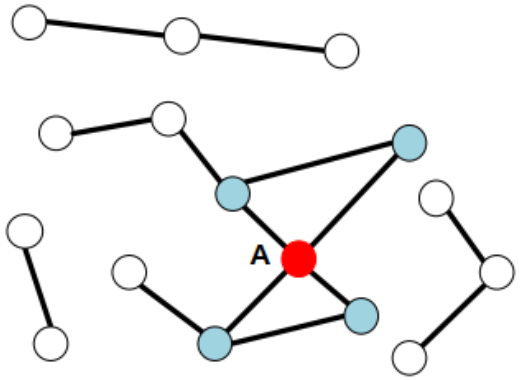
- **Links:** directed (arcs)



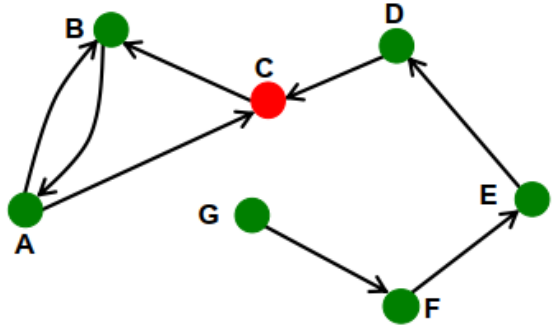
- **Examples:**
  - Phone calls
  - Following on Twitter

Node Degree

Undirected



Directed



Source: Node with $k^{in} = 0$
Sink: Node with $k^{out} = 0$

Node degree, k_i: the number of edges adjacent to node ***i***

$$k_A = 4$$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

In directed networks we define an **in-degree** and **out-degree.**

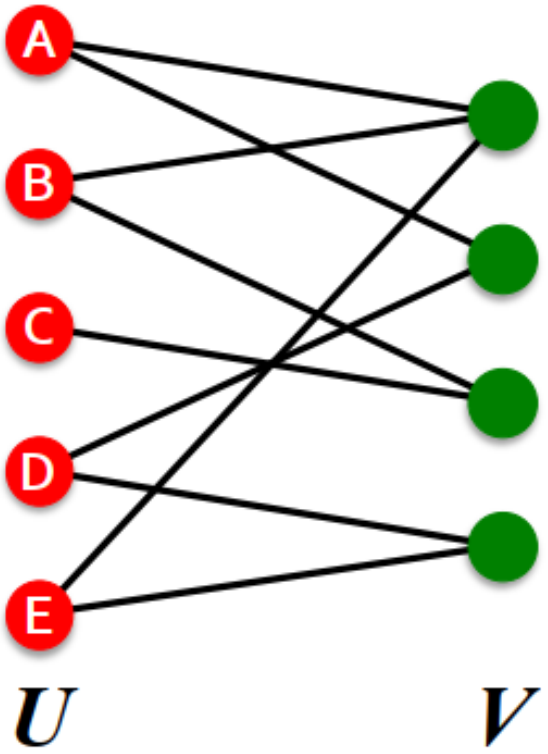
The (total) degree of a node is the sum of in- and out-degrees.

$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

$$\bar{k} = \frac{E}{N} \qquad \overline{k^{in}} = \overline{k^{out}}$$

Bipartite Graph

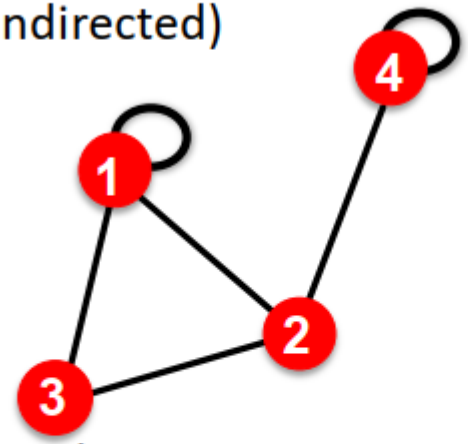
- **Bipartite graph** is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are **independent sets**



- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)

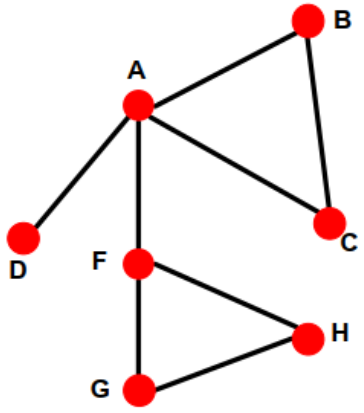
- **Self-edges (self-loops)**

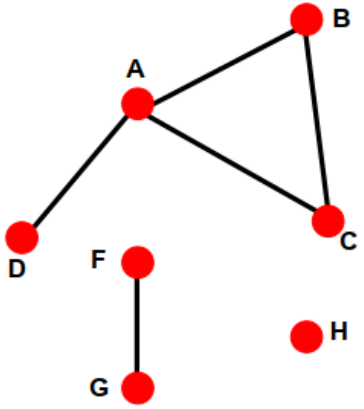
(undirected)



$$A_{ij} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

- **Connected (undirected) graph:**
  - Any two vertices can be joined by a path
- A disconnected graph is made up by two or more connected components

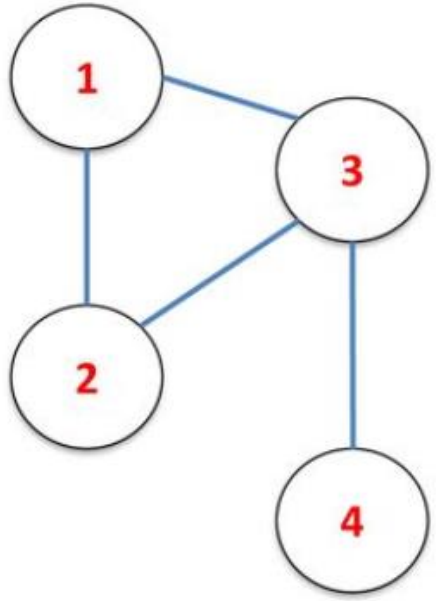




Largest Component:
Giant Component

Isolated node (node H)

Graph Representation : Matrix form



Example graph

Adjacency matrix(A)

	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0

Degree matrix (D)

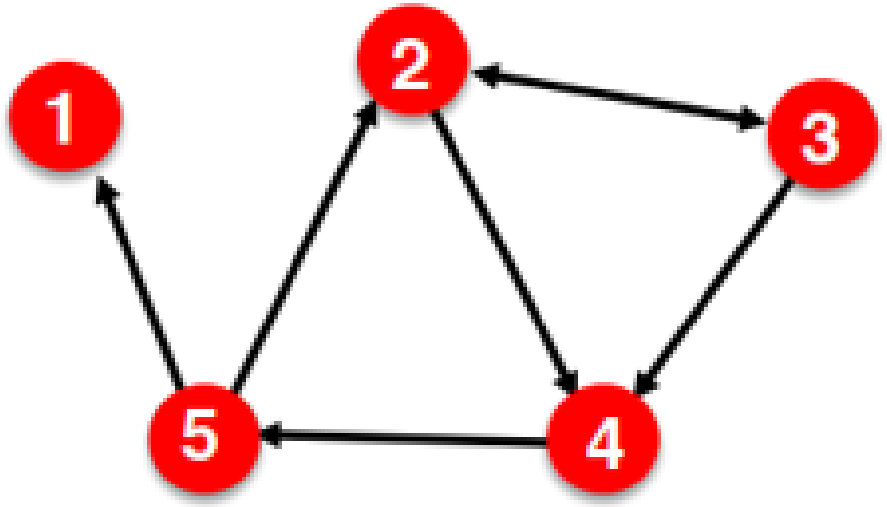
	1	2	3	4
1	2	0	0	0
2	0	2	0	0
3	0	0	2	0
4	0	0	0	1

Laplacian ($L=D-A$)

	1	2	3	4
1	2	-1	-1	0
2	-1	2	-1	0
3	-1	-1	2	-1
4	0	0	-1	1

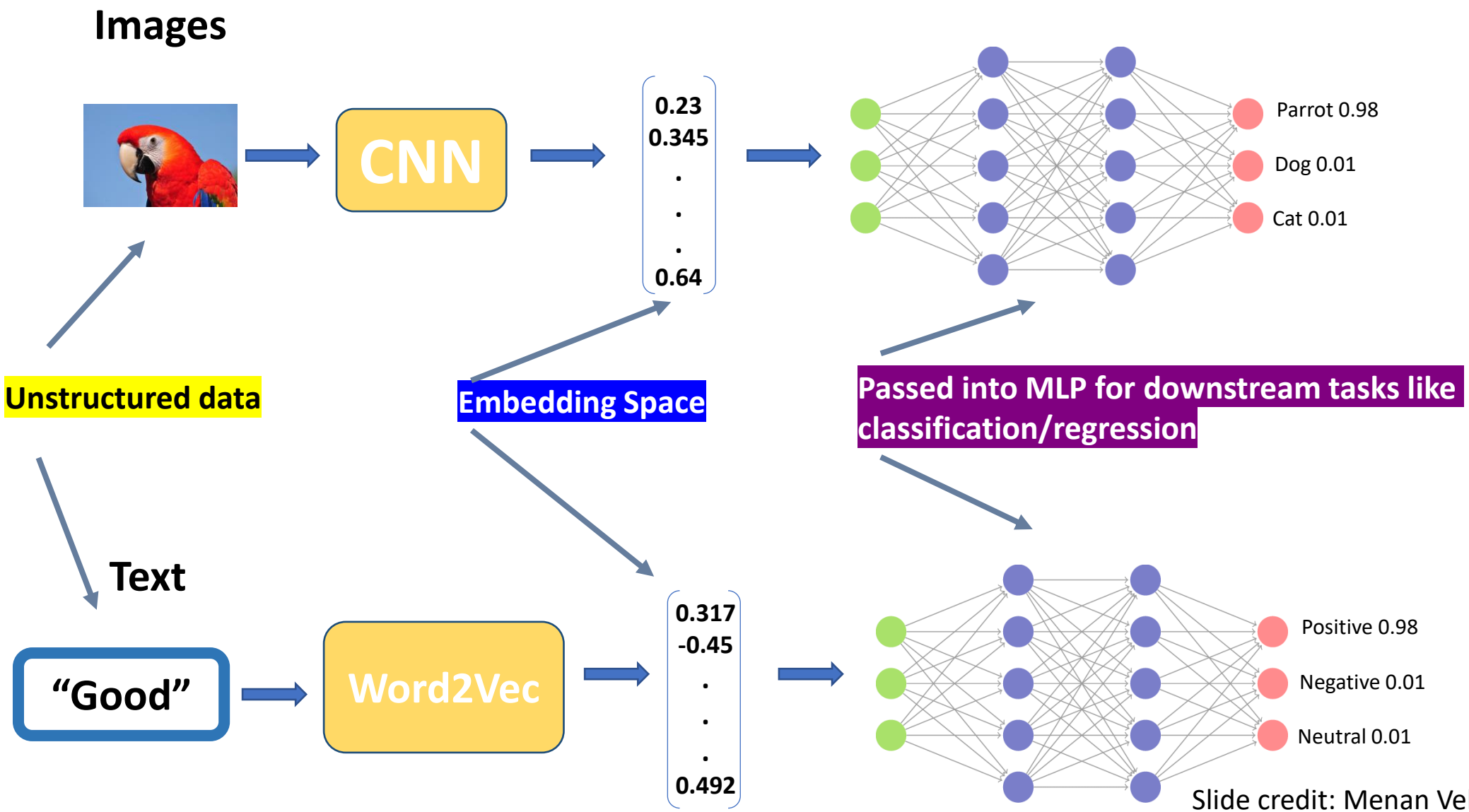
Graph Representation : Other forms

- **Represent graph as a list of edges:**
  - (2, 3)
  - (2, 4)
  - (3, 2)
  - (3, 4)
  - (4, 5)
  - (5, 2)
  - (5, 1)



Representational Learning

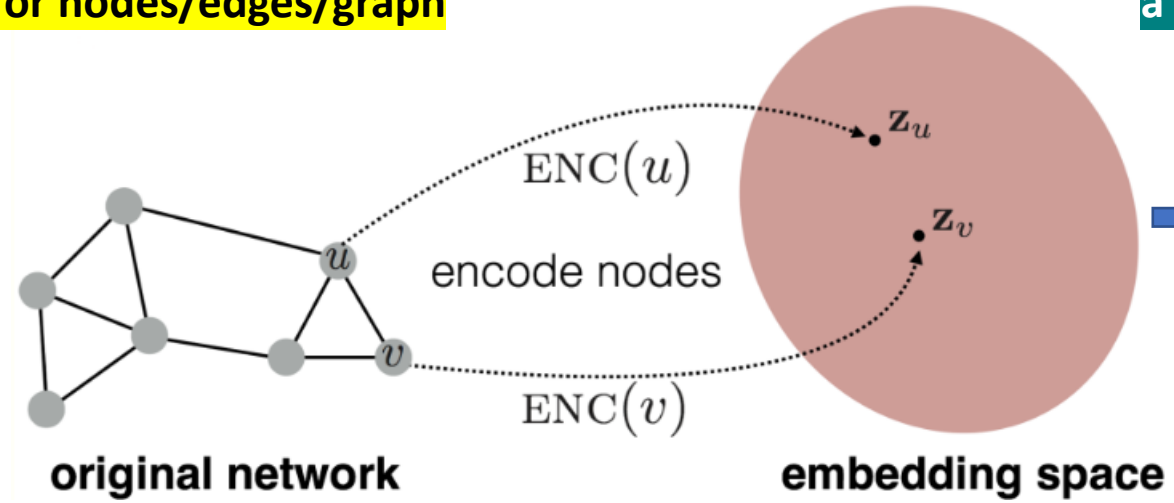
- One common denominator in all the node-level, edge-level and graph-level tasks is finding representations for nodes or edges or the entire graphs.
- This of course isn’t new to machine learning, after all, machine learning is dominated by representational learning.
- Models such as Convolutional Neural Network(CNN), Word2Vec (Skipgram, CBOW models), transformers have been doing this under the hood.



Slide credit: Menan Velayuthan

Same is required for Graph Machine Learning and Graph Neural Networks(GNN) is the state-of-the-art approach to achieve this.

We have a graph and we find a representation for nodes/edges/graph



Take the embedding, in this case a node embedding

$\boldsymbol{z_u}$

Perform down stream task

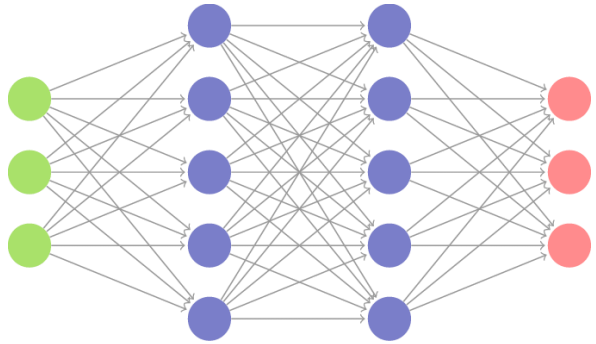


Image source : https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/node-representation-learning

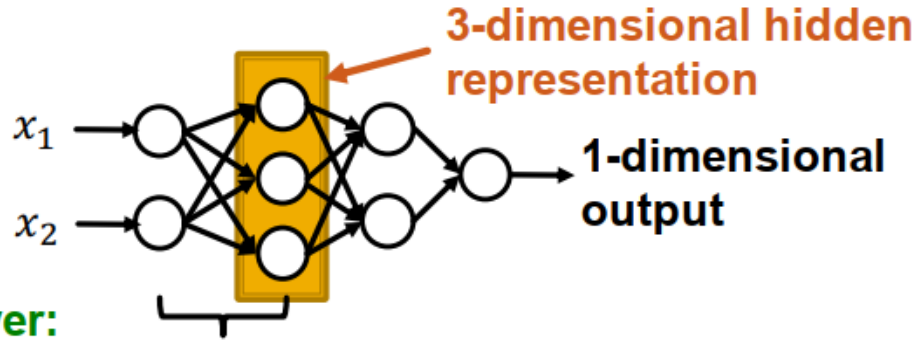
Graph Neural Networks

DL notation recap

- **Each layer of MLP combines linear transformation and non-linearity:**

$$\boldsymbol{x}^{(l+1)} = \sigma(W_l \boldsymbol{x}^{(l)} + b^l)$$

  - where W_l is weight matrix that transforms hidden representation at layer l to layer $l+1$
  - b^l is bias at layer l, and is added to the linear transformation of $\boldsymbol{x}^{(l)}$
  - σ is non-linearity function (e.g., sigmoid)
- Suppose $\boldsymbol{x}$ is 2-dimensional, with entries x_1 and x_2



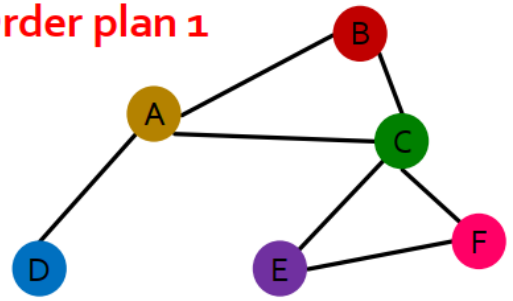
Graph notation recap

- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information
    - When there is no node feature in the graph dataset:
      - Indicator vectors (one-hot encoding of a node)
      - Vector of constant 1: [1, 1, …, 1]

Permutation invariance

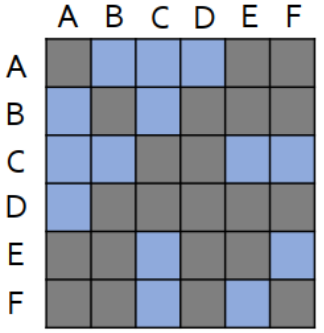
- **Graph does not have a canonical order of the nodes!**

Order plan 1

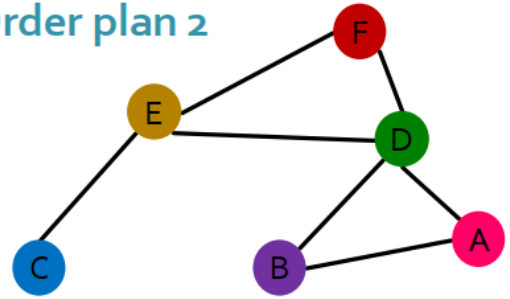


Node features X_1

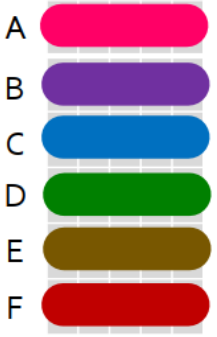
Adjacency matrix A_1



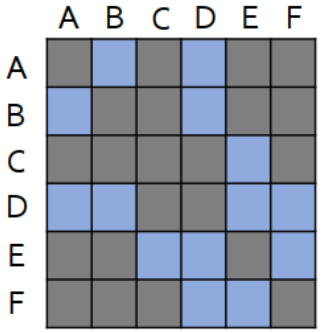
Order plan 2



Node features X_2



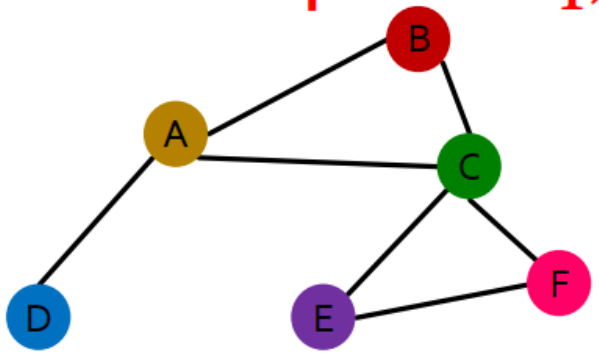
Adjacency matrix A_2



Permutation equivariance

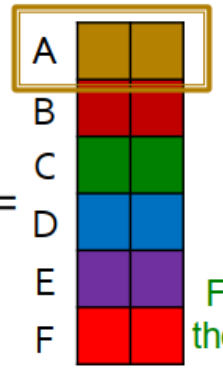
For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{m \times d}$.

Order plan 1: A_1, X_1

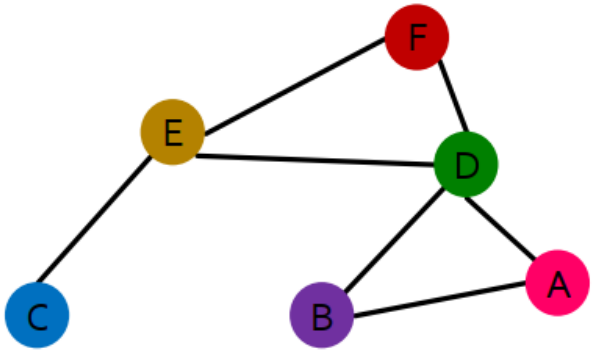


Representation vector of the brown node A

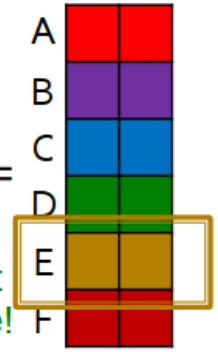
$$f(A_1, X_1) =$$



Order plan 2: A_2, X_2



$$f(A_2, X_2) =$$



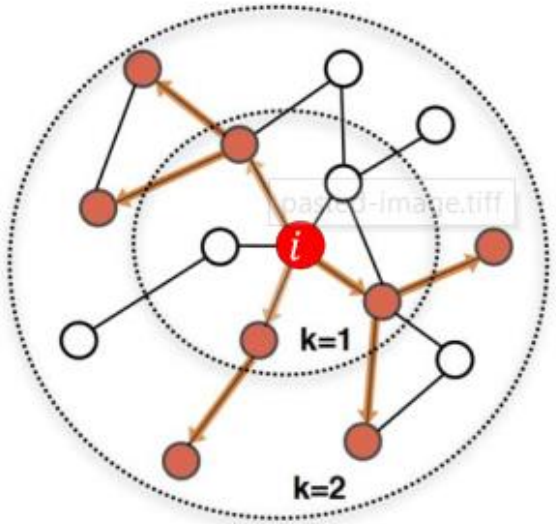
Representation vector of the brown node E

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

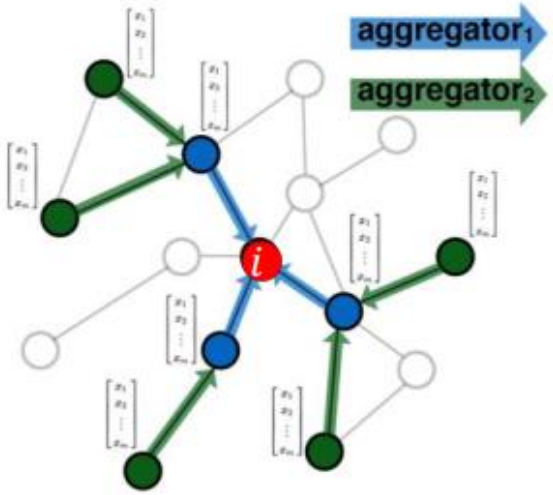
Graph Neural Networks

- GNNs should utilize the bias of graph structures as well as the features present the graph to learn representations.
- Majority of the introduced GNN models follow the 3 building blocks given below,
  - **Message Passing**
  - **Aggregation**
  - **Update**
- Many variants of GNN have been introduced over time, they mainly differ in varying the approach towards one of the 3 mentioned above. Few examples of the variants are Graph Convolution Network (GCN), Graph Sage, Graph Attention Network, Graph Isomorphism Network etc..
- We will address the above-mentioned methodology one by one.
- Here we will take undirected graph with node features only for simplicity. This same method can be extended to tackle edge features or graph level features.

Idea: Node's neighborhood defines a computation graph



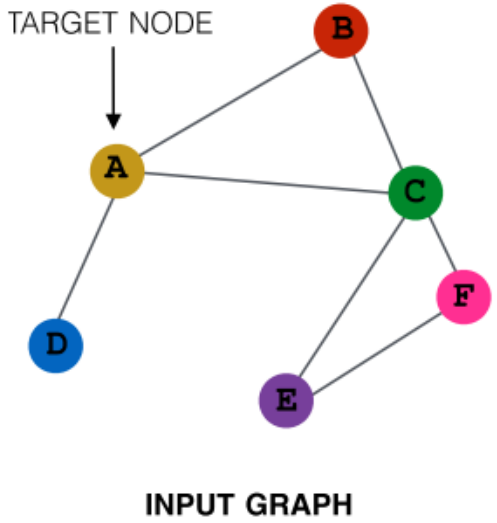
Determine node computation graph



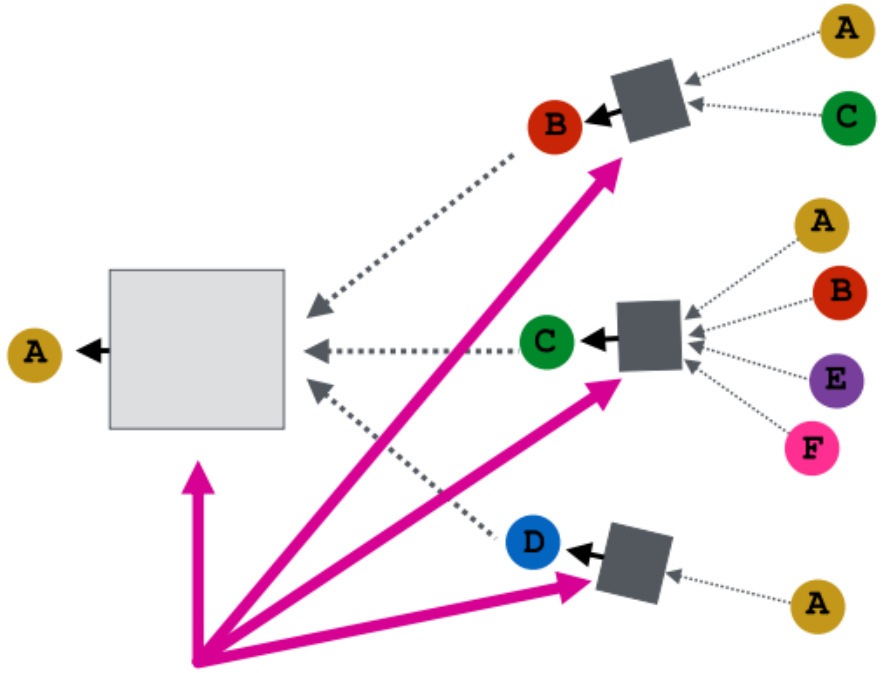
Propagate and transform information

Learn how to propagate information across the graph to compute node features

- **Key idea:** Generate node embeddings based on **local network neighborhoods**
  - **Intuition:** Nodes aggregate information from their neighbors using neural networks



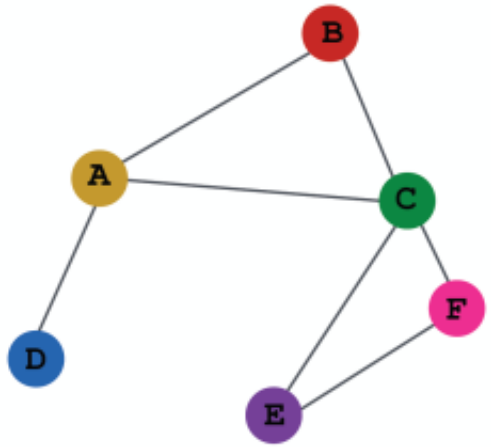
INPUT GRAPH



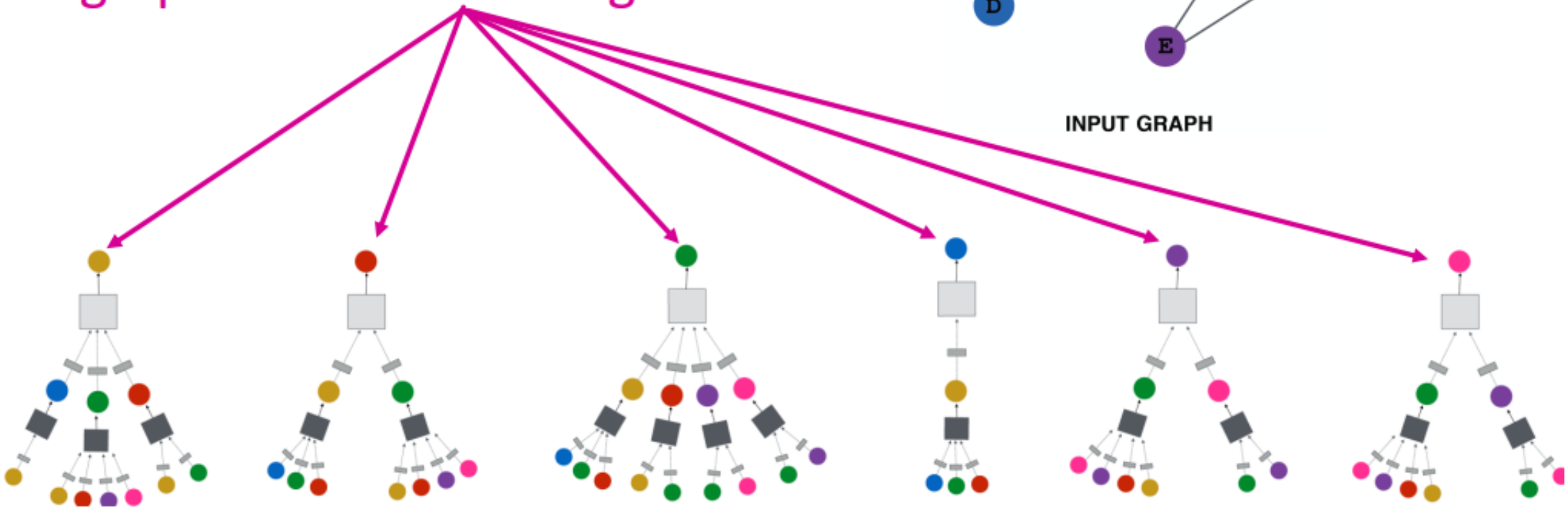
Neural networks

- **Intuition:** Network neighborhood defines a computation graph

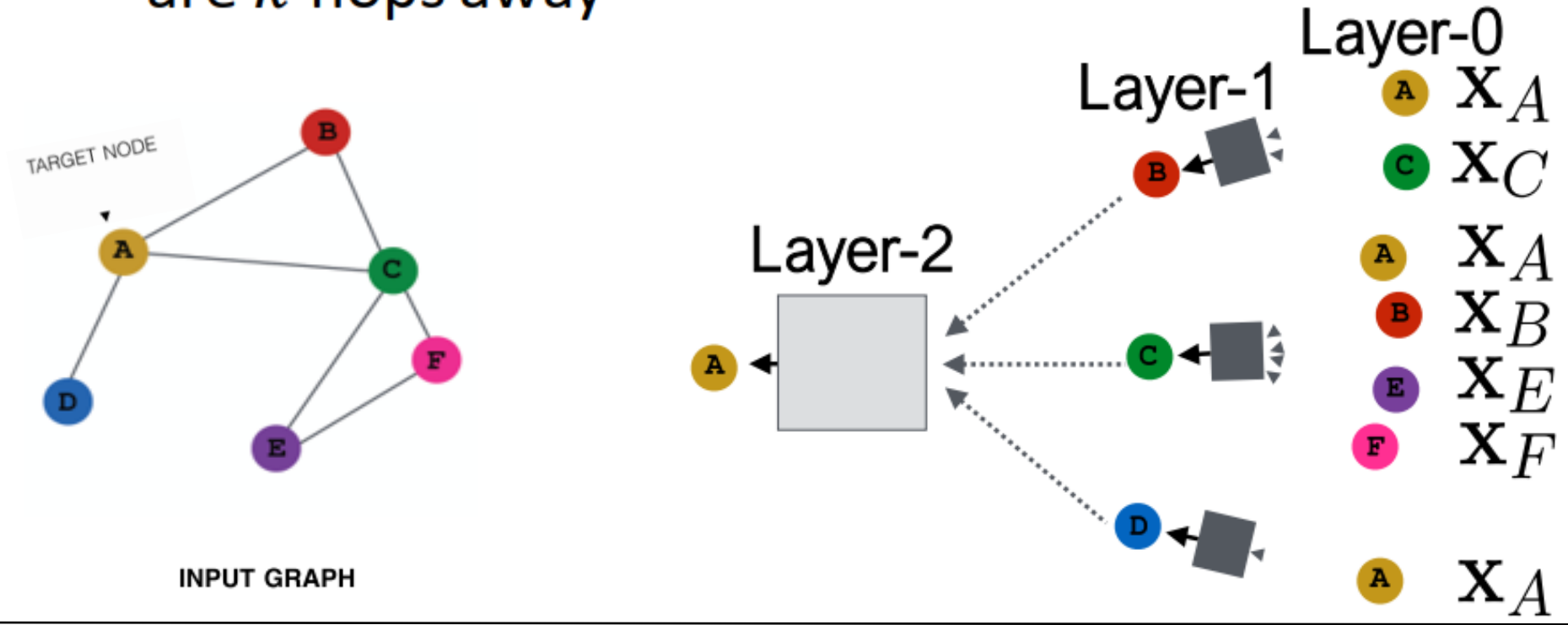
Every node defines a computation graph based on its neighborhood!



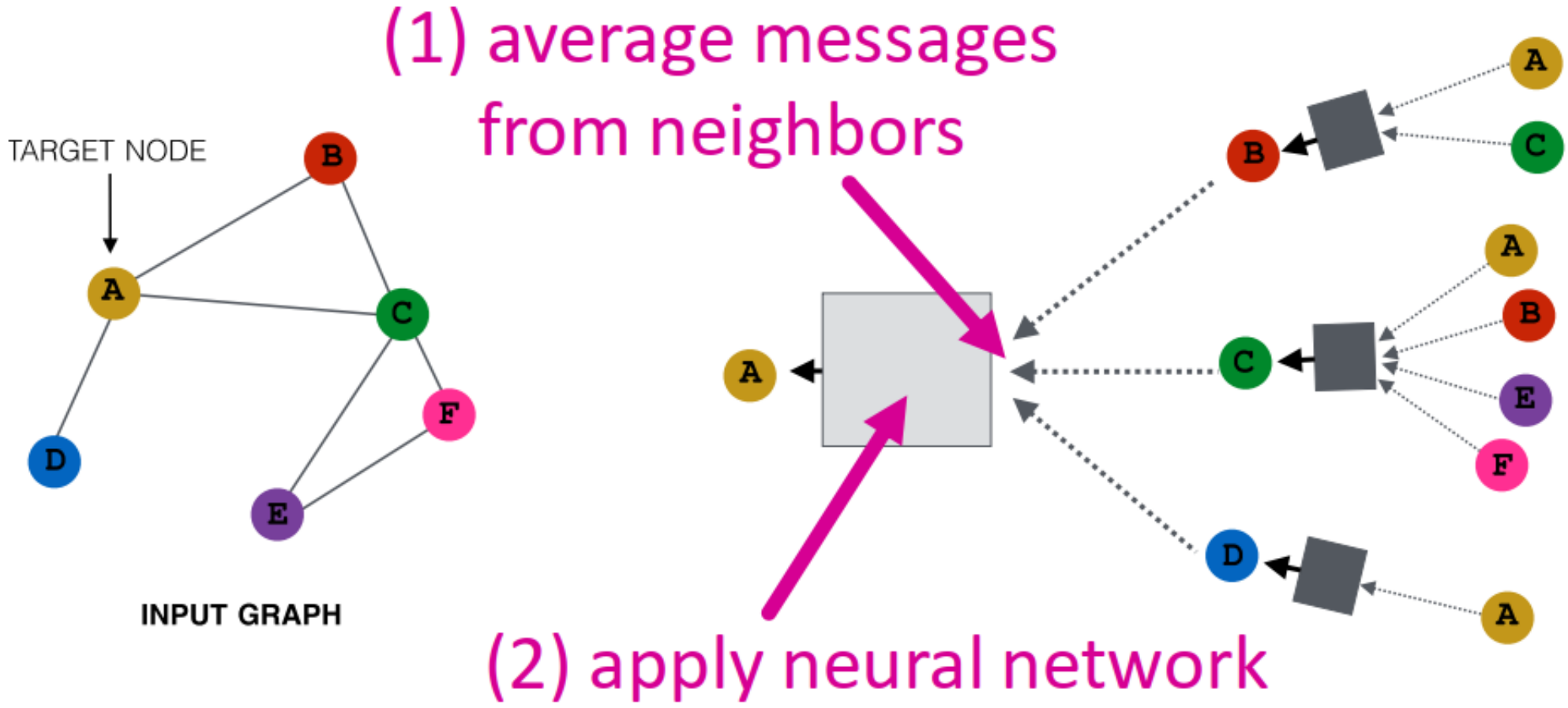
INPUT GRAPH



- 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



- **Basic approach:** Average information from neighbors and apply a neural network



MPGNN (Message Passing GNN)

We begin here with the most basic GNN framework, which is a simplification of the original GNN models proposed by Merkwirth and Lengauer [2005] and Scarselli et al. [2009]

$$\mathrm{h}_u^{(k)} = \sigma\left(\mathrm{W}_{self}^{(k)}\mathrm{h}_u^{(k-1)} + \mathrm{W}_{neigh}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathrm{h}_v^{(k-1)} + \mathrm{b}^{(k)}\right) \tag{7}$$

where,

$$\mathrm{W}_{self}^{(k)}, \mathrm{W}_{neigh}^{(k)} \in \mathbb{R}^{\mathrm{d}^{(k)}\times\mathrm{d}^{(k-1)}}$$
$$\mathrm{b}^{(k)} = \mathrm{b}_{self}^{(k)} + \mathrm{b}_{neigh}^{(k)}$$

In equation 7,

$\mathrm{W}_{self}^{(k)}$: weight matrix of target node's (u) multi-layer perceptron (MLP),

$\mathrm{W}_{neigh}^{(k)}$: weight matrix of MLP used with target node's neighborhood nodes $(\mathcal{N}(u))$,

$\sigma(.)$: activation function.

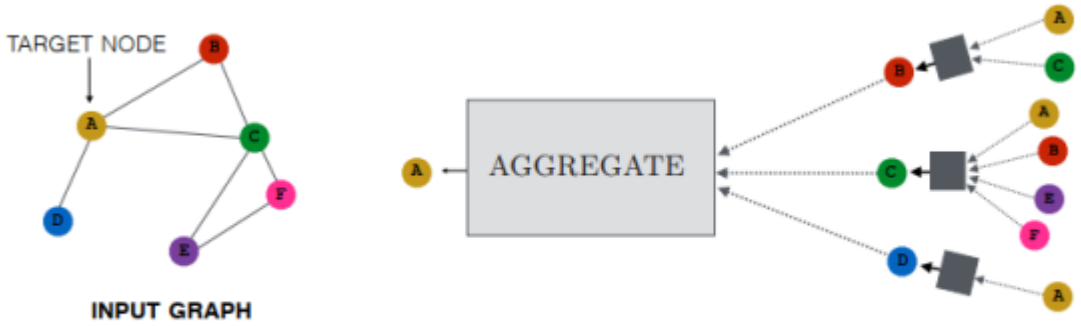


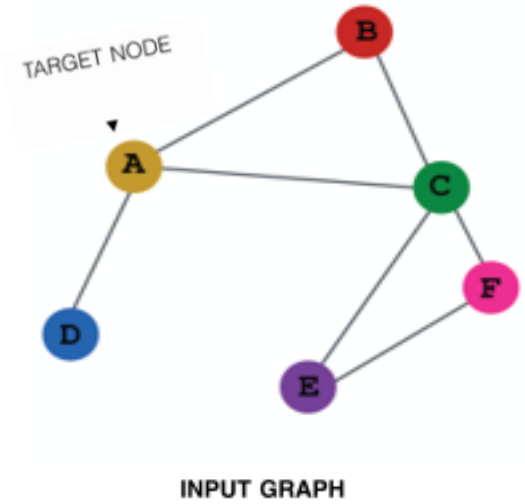
Fig. 1: GNN message passing mechanism

Note : There is no self-loop in Fig.1, but the general MPGNN equation has a self information aggregation

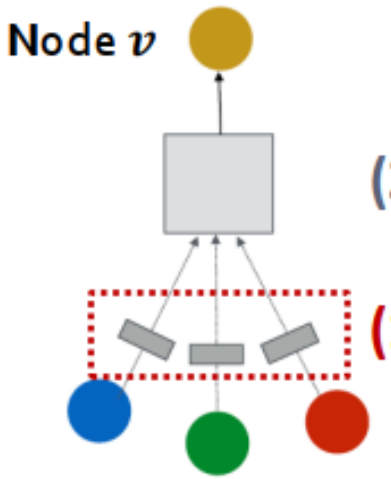
Message Passing

- Message Passing has been the key workforce of Graph Neural Networks, although there are work ongoing beyond Message Passing, it has proven to be effective.
- In English there is a famous saying, ***"A person is known by the company he keeps"***, message passing works similar to this. In a more technical term this called **Homophily,** which means the nodes connected to each other tend to have similar features and/or belong to similar classes.
- For example, in a node classification task, our Message Passing layer will take a source node, identify its neighbours, transform the neighbours features and finally pass it to the source node. This is process is parallelly done to all the nodes in the graph. Hence at the end of this step all the nodes are updated.
- Now lets look at what is discussed using an example.

- # (1) Message computation
  - **Message function**: $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
    - **Intuition:** Each node will create a message, which will be sent to other nodes later
    - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$
      - Multiply node features with weight matrix $\mathbf{W}^{(l)}$



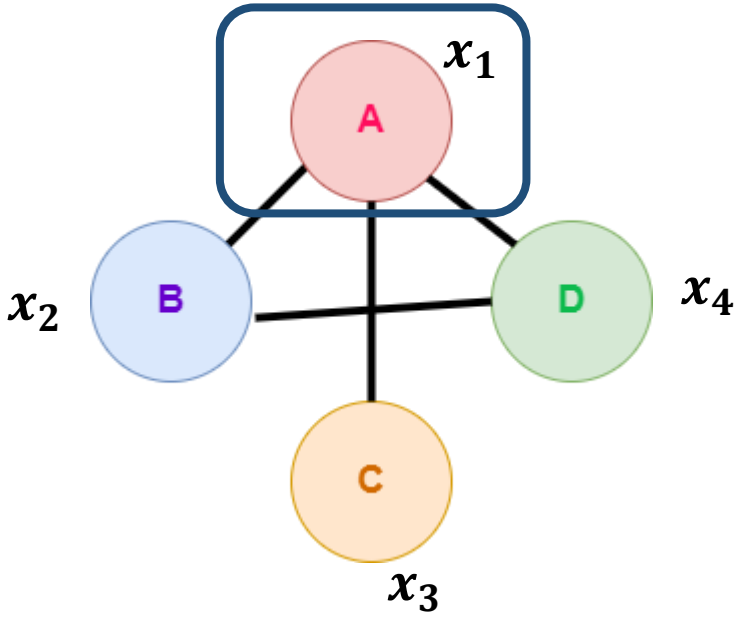
INPUT GRAPH



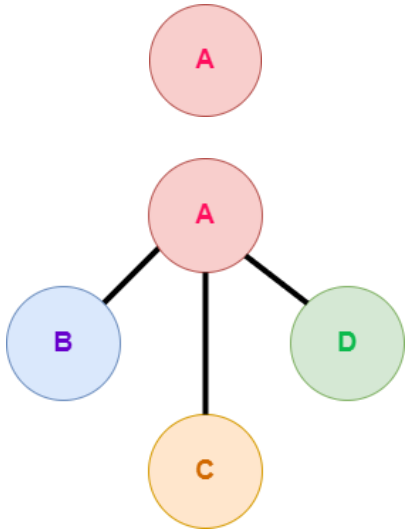
(2) Aggregation

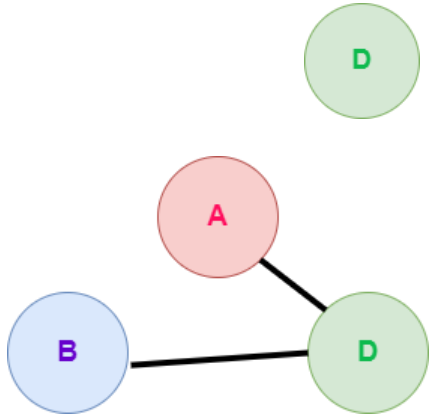
(1) Message

Message Passing Example

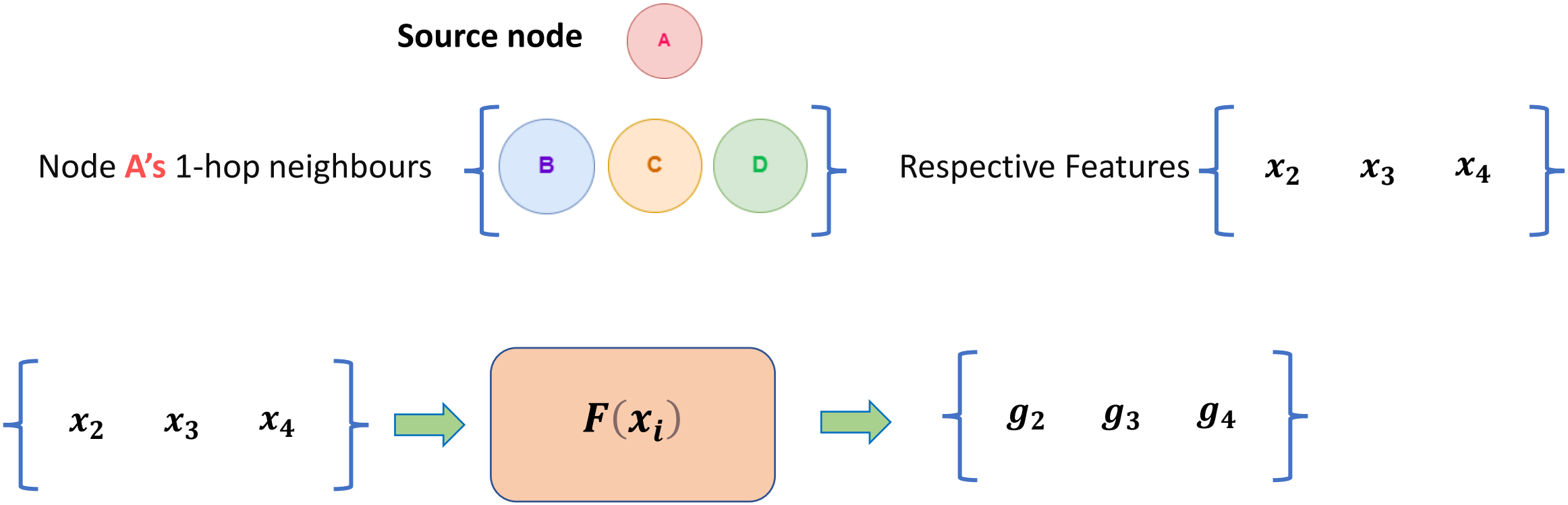


For example, lets identify the **1-hop neighbours** of the graph





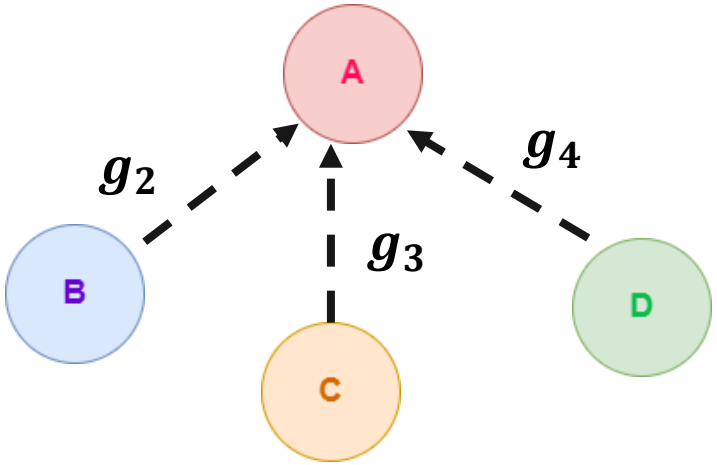
Now let's take Node **A** as the source node and perform message passing



Here $\boldsymbol{F}$ can be MLP or RNN or simple affine function meaning simple linear tranformation will also work.
Hence,

$$F(x_i) = Wx_i + b \ (here\ we\ perform\ linear\ transformation\ on\ x_i\ )$$

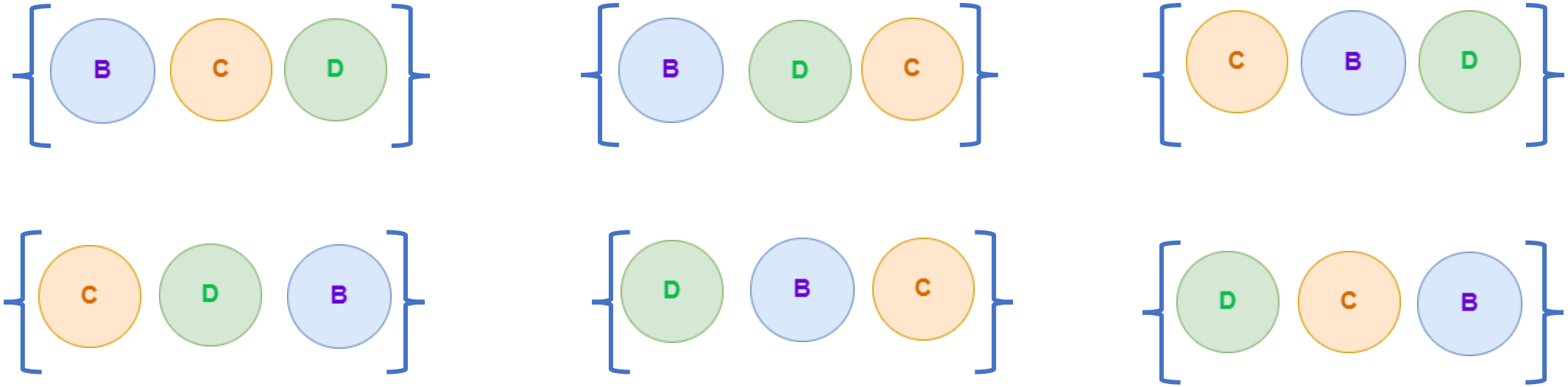
Finally, the transformed messages (g_i) gets passed to the source node



Aggregate

- Now that the messages from the 1-hop neighbour nodes have been passed into the source node.
- We have to now find a way to combine these messages together (AGGREGATE).
- But here is the catch, our aggregate function should be "**Permutation Invariant**"**.** This a key bias we will introduce to the GNNs. The justification is, that the nieghbour nodes should be taken in any order and shouldn't affect the AGGREGATION.

Meaning, independent of the order we take the neighbours of the source node, the AGGREGATE function should produce the same results (Permutation Invariant)



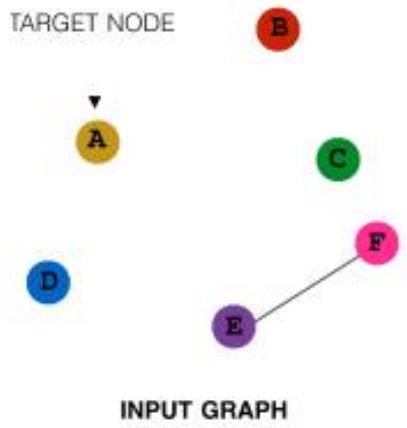
Nothing tells the first order of nodes is better the other combination of node order, this is the reason for Permutation Invariant Aggregation function selection.

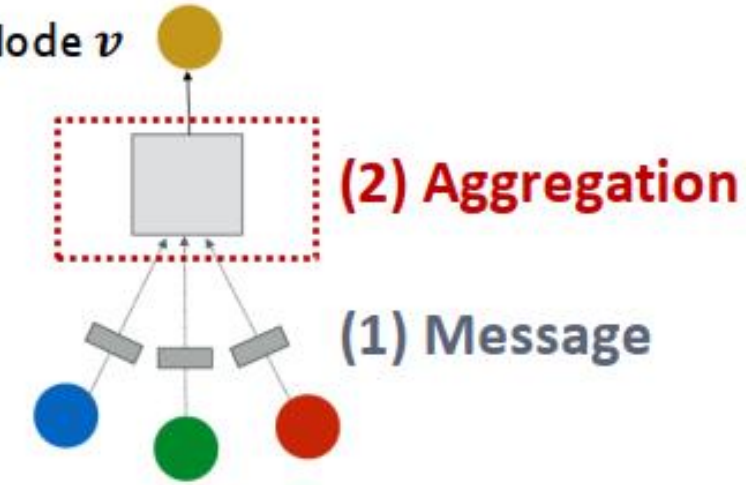
(2) Aggregation

- **Intuition:** Each node will aggregate the messages from node v's neighbors

$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$

- **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
  - $\mathbf{h}_v^{(l)} = \mathrm{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$





Some Famous Permutation Invariant Aggregation Functions

1. Summation ->1+2+3=3+2+1=2+1+3=6
2. Mean -> (1+2+3)/3=(3+2+1)/3=(2+1+3)=6/3=2
3. Min -> min(1,2,3)=min(3,2,1)=min(2,1,3) = 1
4. Max -> max(1,2,3)=max(3,2,1)=max(2,1,3) = 3

Hence let's define permutation invariant function AGG

$$m_i = AGG(g_j; j \in N_i)$$

m_i - Combined message of neighbours of I

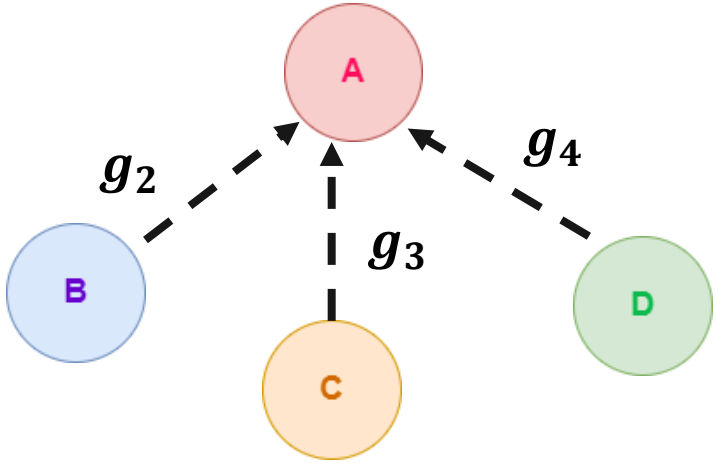
AGG – Can be any permutation invariant function (sum,mean,min ,max etc)

g_j - Transformed features of the neighbour nodes

N_i - Neigbourhood nodes of node i

To Recap

We perform message passing by transforming the features first.



Then we aggregate the messages by using a permutation invariant function

$$m_i = AGG(g_j; j \in N_i)$$

Update

- Finally, we need update the feature of the source node.
- Here we need to utilize the incoming aggregated message as well for the update.

Step 1

- We will first transform the feature of the source node using MLP or an affine function.

$$G(x_i)$$

Step 2

- Now we need to aggregate the transformed feature with m_i .
- To combine m_i and $G(x_i)$ normally in literature we use,
  1. Summation or Averaging
  2. Concatenation.

To elaborate

We can combine $\boldsymbol{m_i}$ and $\boldsymbol{G(x_i)}$ in the following way

$$(\boldsymbol{G(x_i)} + \boldsymbol{m_i})\ \boldsymbol{OR}\ (\boldsymbol{G(x_i)} \,||\, \boldsymbol{m_i}),$$

Here "||" stand for concatenation

NOTE:

- Few items to note, if summation is your choice of combining $\boldsymbol{m_i}$ and $\boldsymbol{G(x_i)}$ then dimensions of $\boldsymbol{m_i}$ and $\boldsymbol{G(x_i)}$ should agree.

Step 3

- Now that we have combined $\boldsymbol{m_i}$ and $\boldsymbol{G(x_i)}$ we will add the usual Neural Networks.
- By that, we will pass the combination of $\boldsymbol{m_i}$ and $\boldsymbol{G(x_i)}$ through the non-linear activation function.

$$\boldsymbol{h_i} = \boldsymbol{\sigma(G(x_i)}, \boldsymbol{m_i})$$

Where,

$\boldsymbol{\sigma}$ = Non-linearity (eg:sigmoid, ReLu, Tanh etc..)

- **Issue:** Information from node v itself **could get lost**
  - Computation of $\mathbf{h}_v^{(l)}$ does not directly depend on $\mathbf{h}_v^{(l-1)}$
- **Solution:** Include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$
  - **(1) Message: compute message from node v itself**
    - Usually, a **different message computation** will be performed

$$\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$$

$$\mathbf{m}_v^{(l)} = \mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)}$$

  - **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node v itself**
    - Via **concatenation** or **summation**

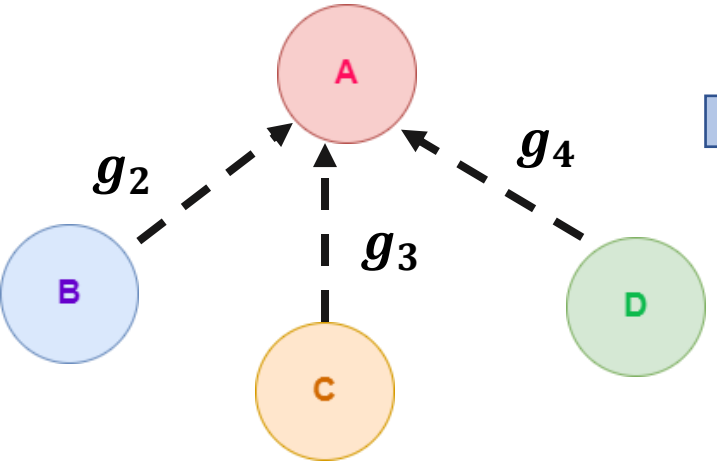
Then aggregate from node itself

$$\mathbf{h}_v^{(l)} = \text{CONCAT}\left(\text{AGG}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right), \mathbf{m}_v^{(l)}\right)$$

First aggregate from neighbors

To sum up the operations in a GNN layer

Message Passing | **Aggregate** | **Update**



$$m_i = AGG(h_j; j \in N_i)$$

$$h_i = \sigma(G(x_i), m_i)$$

Well, it's done, one layer of GNN has been formulated!!

Note:

All these steps are performed on all the nodes in the graph simultaneously.