EE7207: Lecture 11

Graph Neural Networks

Lecturer: Zhang Handuo
zhanghanduo.github.io/
handuo.zhang@shanda.com

About me

Zhang Handuo

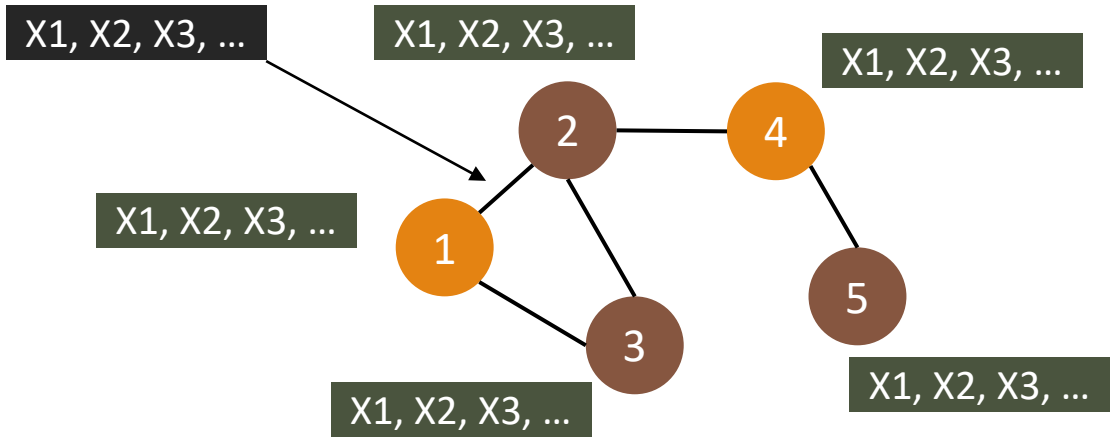
AI Scientist
Shanda Group

zhanghanduo.github.io/
handuo.zhang@shanda.com

Outline

- 1 What is GNN and why?
- 2 Graph Representation
- 3 Machine Learning Types in GNN
- 4 Graph Neural Networks: Basics
- 5 GNN variants: GCN, GAT, GraphSage
- 6 GNN Layers in Practice

1. What is a Graph

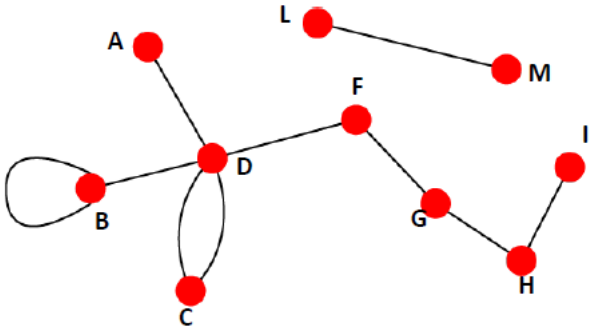


- Entities: nodes, vertices $\mathbf{V} = \{v_1, \ldots, v_N\}$
- Interactions: links, edges $\mathbf{E} = \{e_1, \ldots, e_M\}$
- Graph: network $\mathrm{G} = (\mathbf{V}, \mathbf{E})$

Directed vs. Undirected Graphs

Undirected

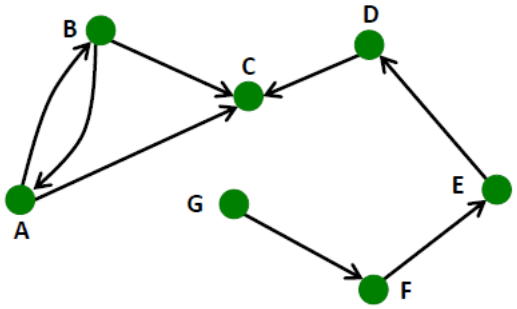
- **Links:** undirected (symmetrical, reciprocal)



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

Directed

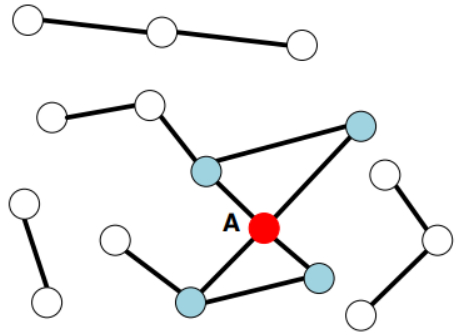
- **Links:** directed



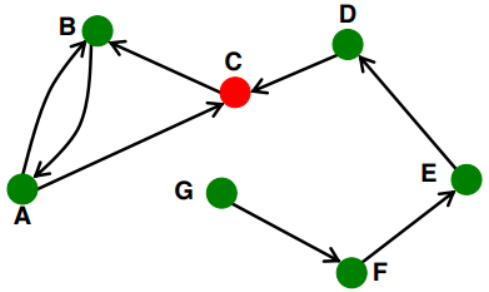
- **Examples:**
  - Phone calls
  - Following on Twitter

Directed vs. Undirected Graphs

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.**

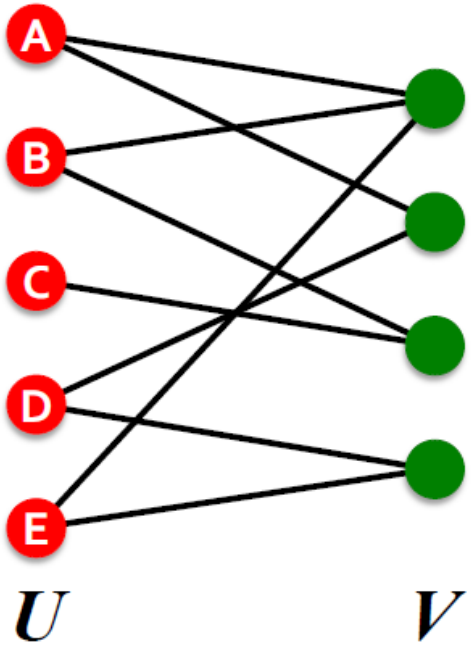
The (total) degree of a node is the sum of in- and out-degrees.

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

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

Bipartite Graph

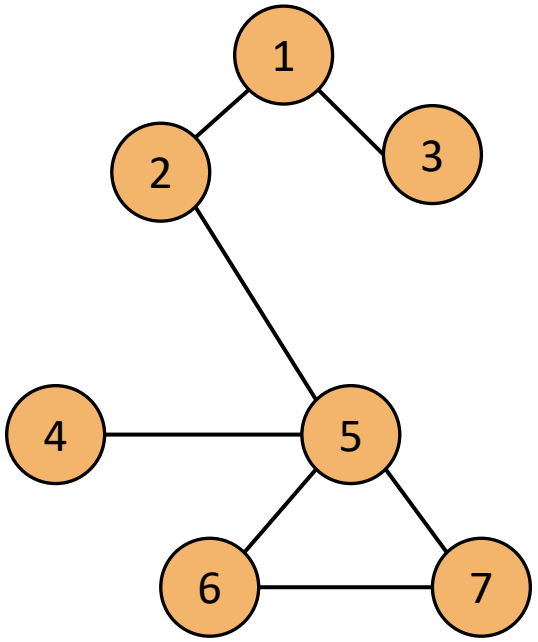
- **Bipartite graph** is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are **independent sets**



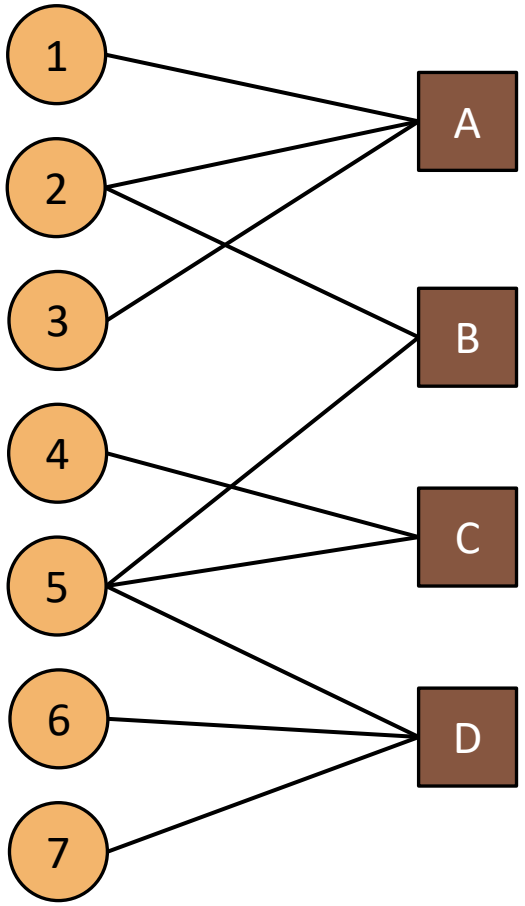
- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)
- **"Folded" networks:**
  - Author collaboration networks
  - Movie co-rating networks

Projected Bipartite Graph

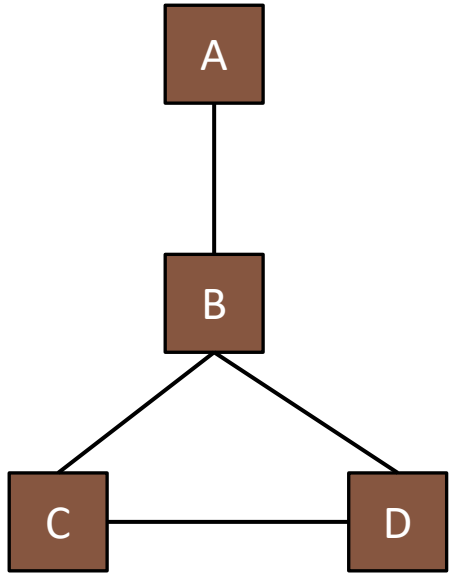
Projection U



U **V**

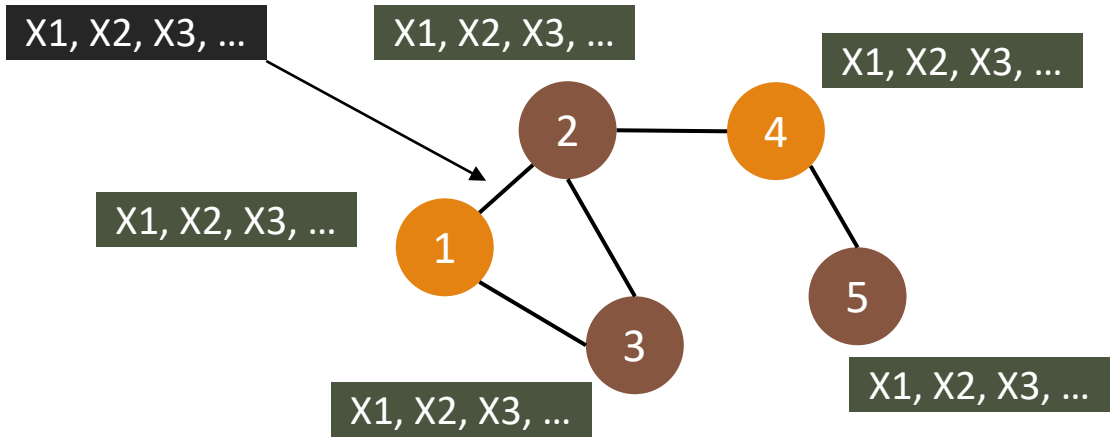


Projection V



Adjacency Matrix

Standard Network architectures don't work on such data



- Entities: nodes, vertices V
- Interactions: links, edges E
- System: network, graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$

Representing Graphs

$A_{ij} = 1$ if there is a link from node i to node j

$A_{ij} = 0$ otherwise

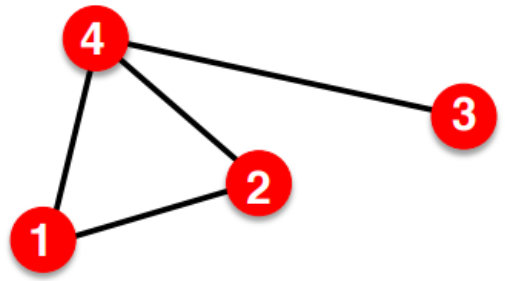
Adjacency Matrix $\mathrm{V} \times \mathrm{V}$

	V1	V2	…
V1	0	1	…
V2	1	0	…
V3	1	1	…
…	…	…	…

Note that for an undirected graph the adjacent matrix is symmetric

Adjacency Matrix

Undirected



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

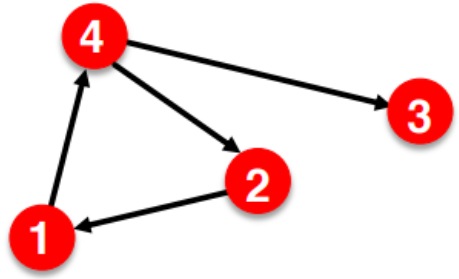
$$A_{ij} = A_{ji}$$
$$A_{ii} = 0$$

$$k_i = \sum_{j=1}^{N} A_{ij}$$

$$k_j = \sum_{i=1}^{N} A_{ij}$$

$$L = \frac{1}{2}\sum_{i=1}^{N} k_i = \frac{1}{2}\sum_{ij} A_{ij}$$

Directed



$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

$$A_{ij} \neq A_{ji}$$
$$A_{ii} = 0$$

$$k_i^{out} = \sum_{j=1}^{N} A_{ij}$$

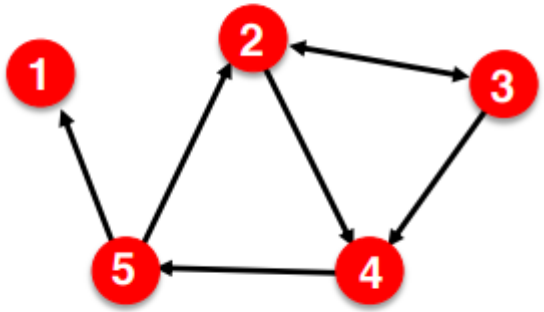
$$k_j^{in} = \sum_{i=1}^{N} A_{ij}$$

$$L = \sum_{i=1}^{N} k_i^{in} = \sum_{j=1}^{N} k_j^{out} = \sum_{i,j}^{N} A_{ij}$$

In real world applications, the graph is sparse -- adjacency matrix is filled with zeros

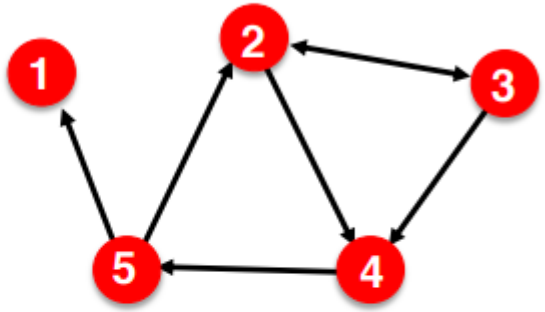
Edge list

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



Adjacency list

- **Adjacency list:**
  - Easier to work with if network is
    - Large
    - Sparse
  - Allows us to quickly retrieve all neighbors of a given node
    - 1:
    - 2: 3, 4
    - 3: 2, 4
    - 4: 5
    - 5: 1, 2



Comparisons of the 3 representations

Aspect	Adjacency Matrix	Edge List	Adjacency List
Best for	Dense graphs	Sparse graphs, iterating over edges	Most situations, especially sparse graphs
Memory Usage	High ($O(V^2)$)	Low ($O(E)$)	Moderate ($O(V+E)$)
Edge Existence Check	Very efficient ($O(1)$)	Inefficient ($O(E)$)	Moderate ($O(V)$ in worst case)
Add/Remove Edge	Efficient ($O(1)$)	Varies ($O(1)$ to $O(E)$ depending on implementation)	Efficient ($O(1)$ if implemented as a linked list)
Add/Remove Vertex	Inefficient (requires matrix resizing)	Efficient ($O(1)$ for adding)	Efficient for adding ($O(1)$), varies for removing
Iterate over Neighbors	Efficient ($O(V)$)	Inefficient ($O(E)$ in worst case)	Efficient ($O(\text{degree of vertex})$)
Space Efficiency	Inefficient for sparse graphs	Efficient for sparse graphs	Efficient for sparse graphs
Operations Suited	Matrix operations, checking adjacency	Algorithms iterating over edges (e.g., Kruskal's)	Adjacency queries, adding/removing edges

New Drug Discovery in 2020

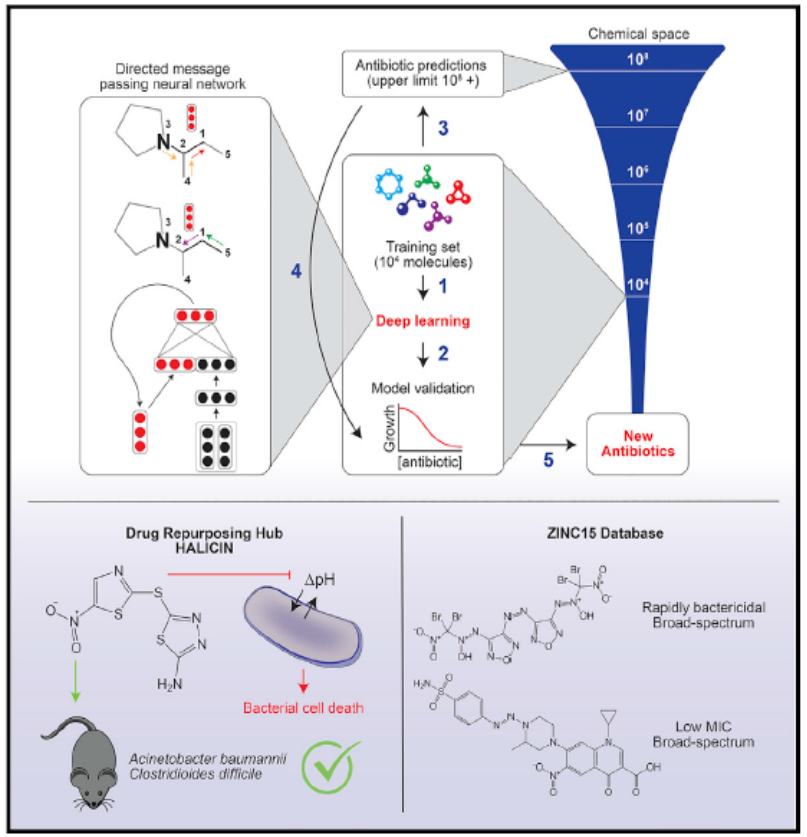
3 Machine Learning Types in GNN

Article

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.

Authors

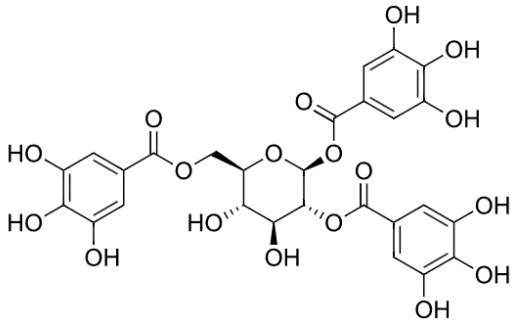
- Jonathan M. Stokes, Kevin Yang,
- Kyle Swanson, ..., Tommi S. Jaakkola,
- Regina Barzilay, James J. Collins

In Brief

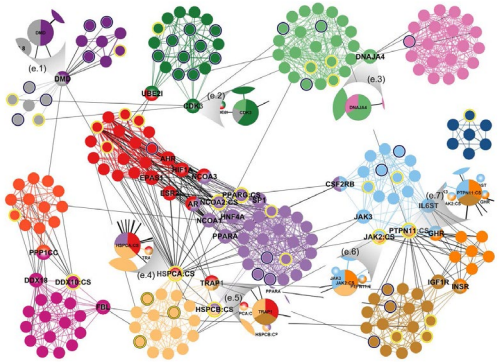
A deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which *Halicin* exhibits efficacy against broad-spectrum bacterial infections in mice.

Graph Data is everywhere

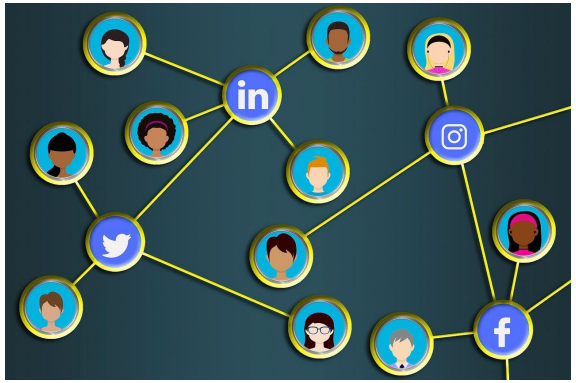
3 Machine Learning Types in GNN



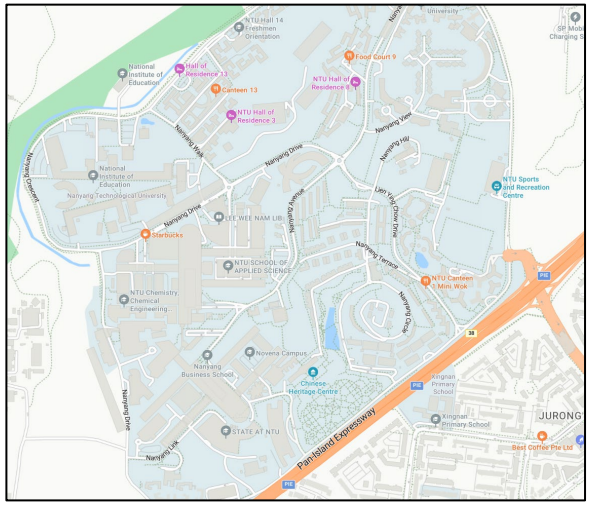
Molecules



Medicine / Pharmacy

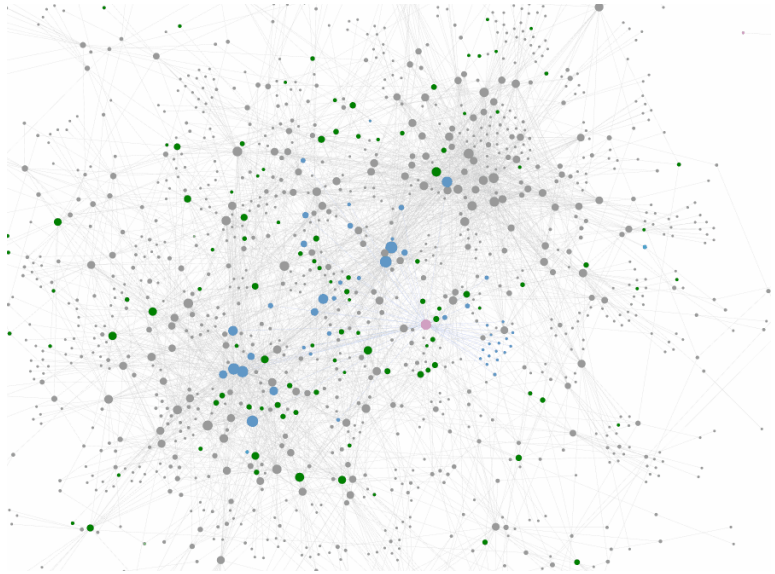


Social Networks



Map of NTU Campus

3D Games / Meshes



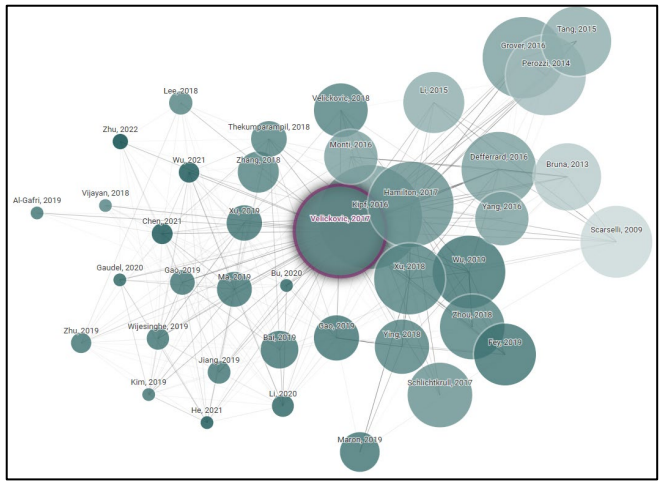
Knowledge Graph

Most real-world data is not as structured and regular as grid data

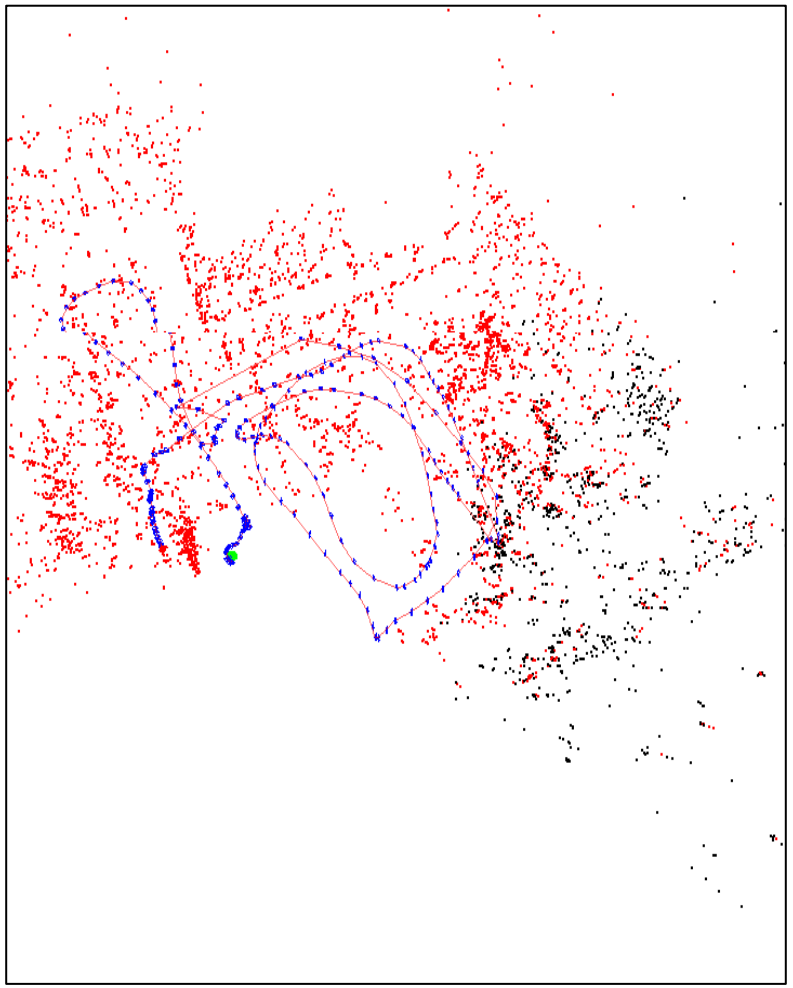
Graph Data is everywhere

3 Machine Learning Types in GNN

Crime board pinned with Evidence



Citation Network

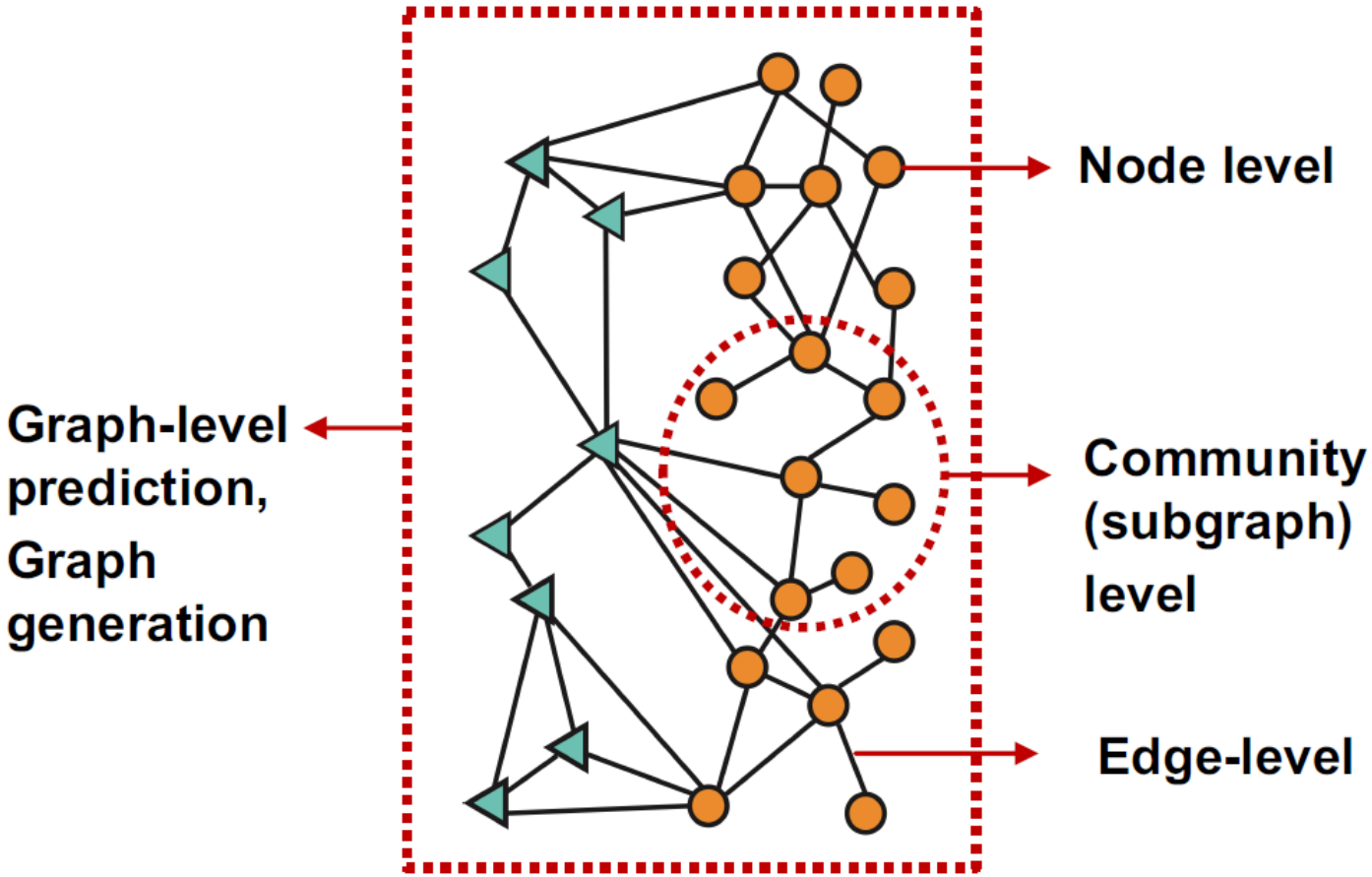


Robot Landmark Map

Code Graph

Most real-world data is not as structured and regular as grid data

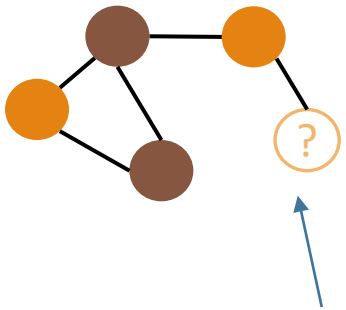
ML Problems with Graph Data



ML Problems with Graph Data

3 Machine Learning Types in GNN

Node-level predictions

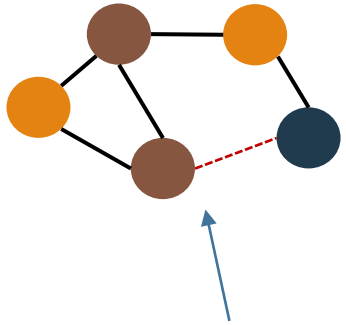


Does this person smoke?
(Unlabeled node)

Predict a property of a node

Categorize online users / items

Edge-level predictions
Link predictions

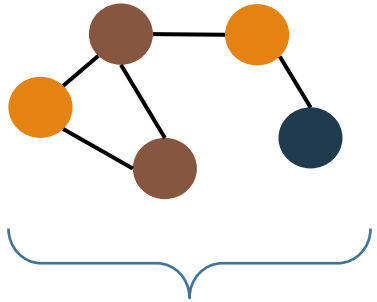


Facebook this person you might know?
(Friend recommendation)

Predict whether there are missing links between two nodes

Knowledge graph completion

Graph-level predictions



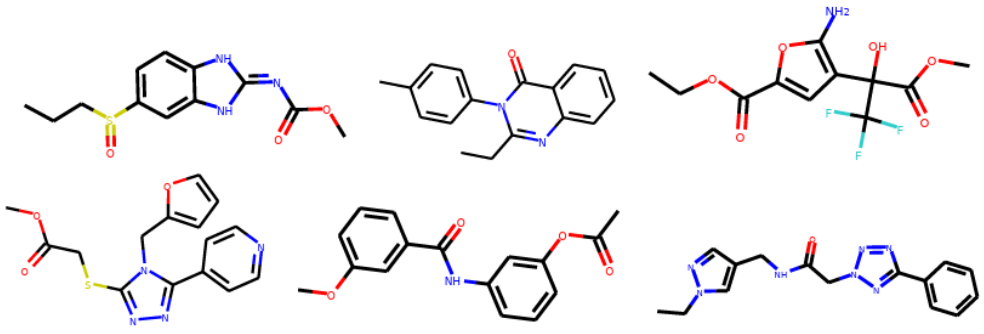
Is this molecule a suitable drug?

Categorize different graphs

Molecule property prediction

ML Problems with Graph Data

- Node Regression: ZINC molecule graph dataset [1]

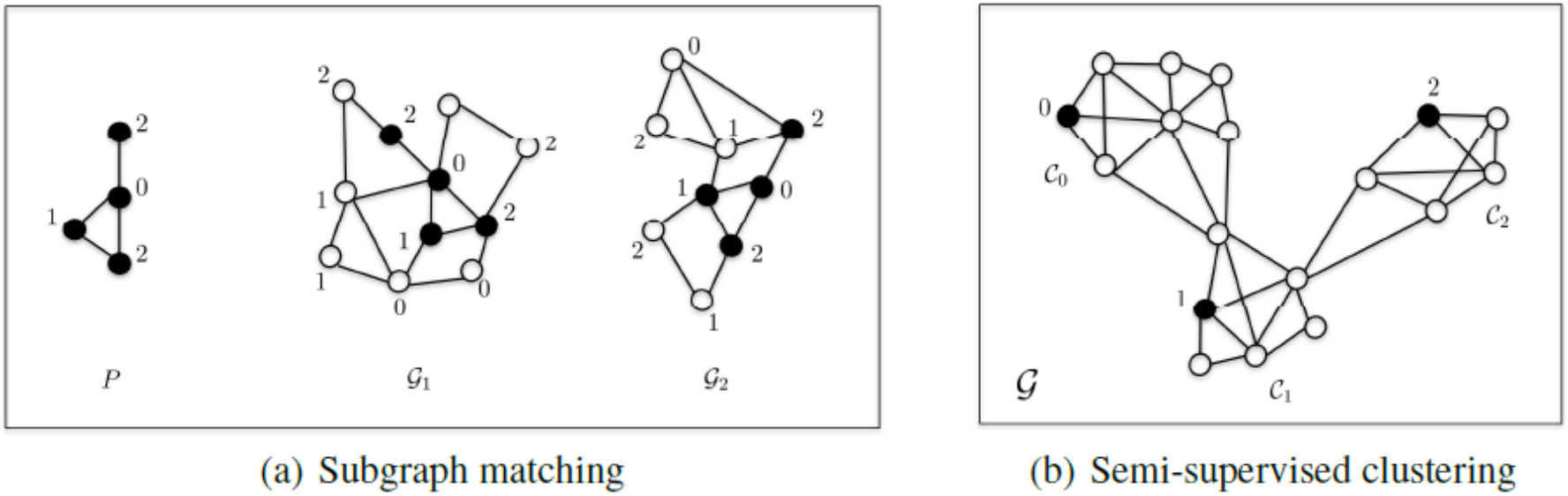


Model	L	#Param	AQSOL[2] TestMAE±s.d.	AQSOL[2] TrainMAE±s.d.	Epochs	Epoch/Total
MLP	4	114525	1.744±0.016	1.413±0.042	85.75	0.61s/0.02hr
vanilla GCN	4	108442	1.483±0.014	0.791±0.034	110.25	1.14s/0.04hr
	16	511443	1.458±0.011	0.567±0.027	121.50	2.83s/0.10hr
GraphSage	4	109620	1.431±0.010	0.666±0.027	106.00	1.51s/0.05hr
	16	509078	1.402±0.013	0.402±0.013	110.50	3.20s/0.10hr
GCN	4	108442	1.372±0.020	0.593±0.030	135.00	1.28s/0.05hr
	16	511443	1.333±0.013	0.382±0.018	137.25	3.31s/0.13hr
MoNet	4	109332	1.395±0.027	0.557±0.022	125.50	1.68s/0.06hr
	16	507750	1.501±0.056	0.444±0.024	110.00	3.62s/0.11hr
GAT	4	108289	1.441±0.023	0.678±0.021	104.50	1.92s/0.06hr
	16	540673	1.403±0.008	0.386±0.014	111.75	4.44s/0.14hr
GatedGCN	4	108325	1.352±0.034	0.576±0.056	142.75	2.28s/0.09hr
	16	507039	1.355±0.016	0.465±0.038	99.25	5.52s/0.16hr
GatedGCN-E	4	108535	1.295±0.016	0.544±0.033	116.25	2.29s/0.08hr
	16	507273	1.308±0.013	0.367±0.012	110.25	5.61s/0.18hr
GatedGCN-E-PE	16	507663	**0.996±0.008**	0.372±0.016	105.25	5.70s/0.30hr

[1] Sterling, Teague, and John J. Irwin. "ZINC 15–ligand discovery for everyone." *Journal of chemical information and modeling* 55.11 (2015): 2324-2337.
[2] Dwivedi, Vijay Prakash, et al. "Benchmarking graph neural networks." *Journal of Machine Learning Research* 24.43 (2023): 1-48.

ML Problems with Graph Data

- Node Classification: Stochastic Block Model dataset [1]



(a) Subgraph matching

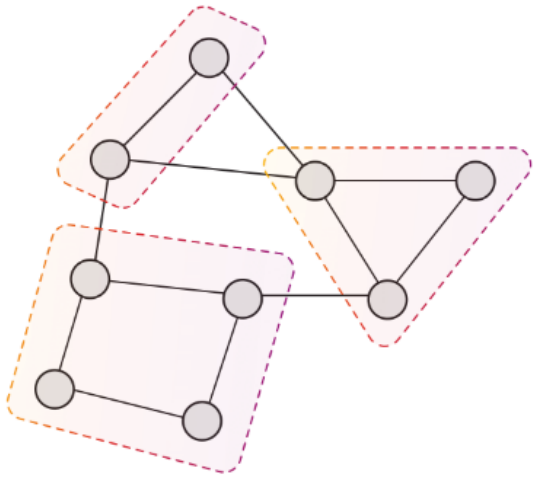
(b) Semi-supervised clustering

[1] Bresson, Xavier, and Thomas Laurent. "Residual gated graph convnets." *arXiv preprint arXiv:1711.07553* (2017).

ML Problems with Graph Data

3 Machine Learning Types in GNN

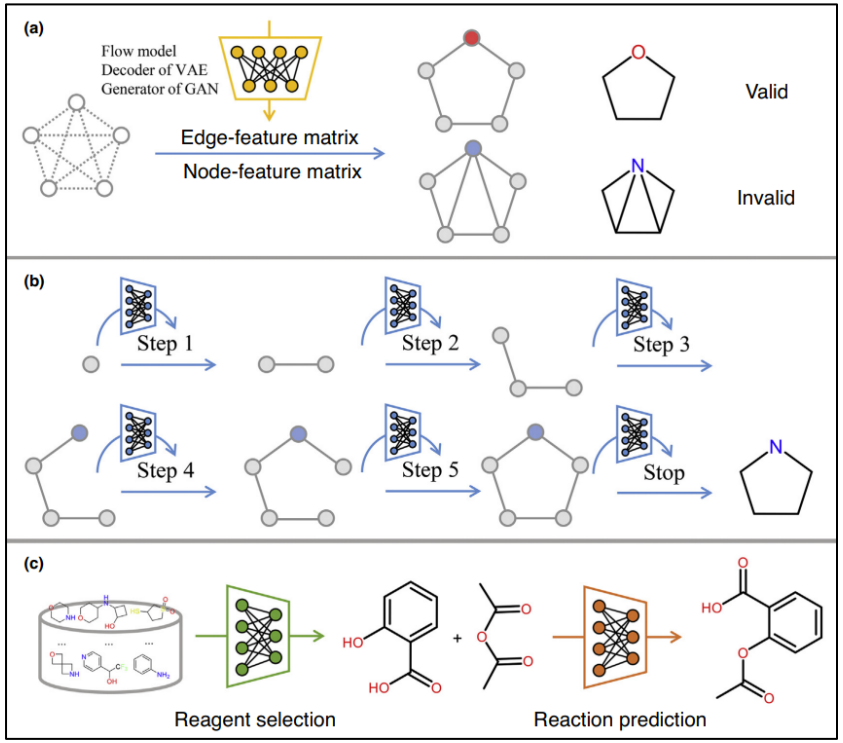
Clustering



Detect if nodes form a community

Social circle detection

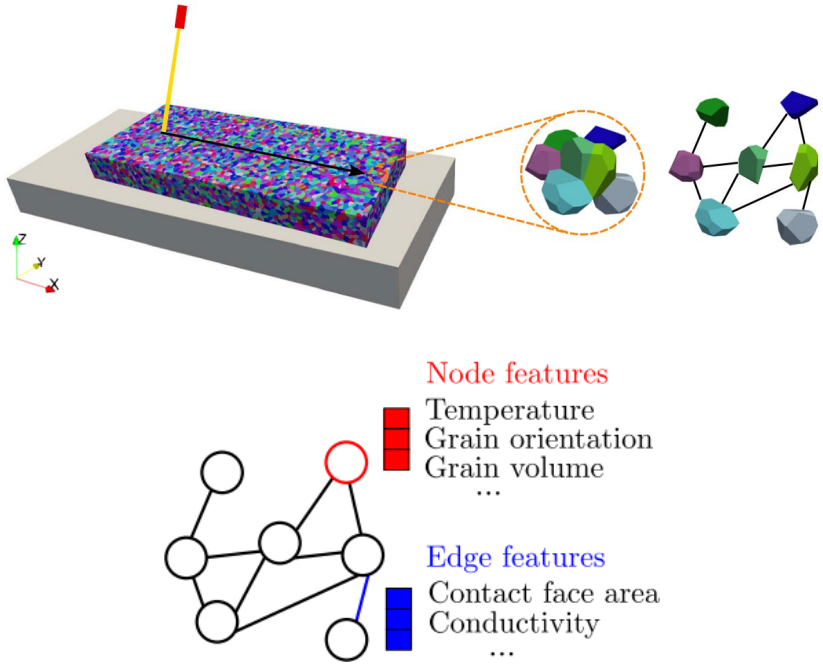
Graph Generation



Generate novel, realistic graphs with desired properties

Drug discovery [1]

Graph Evolution



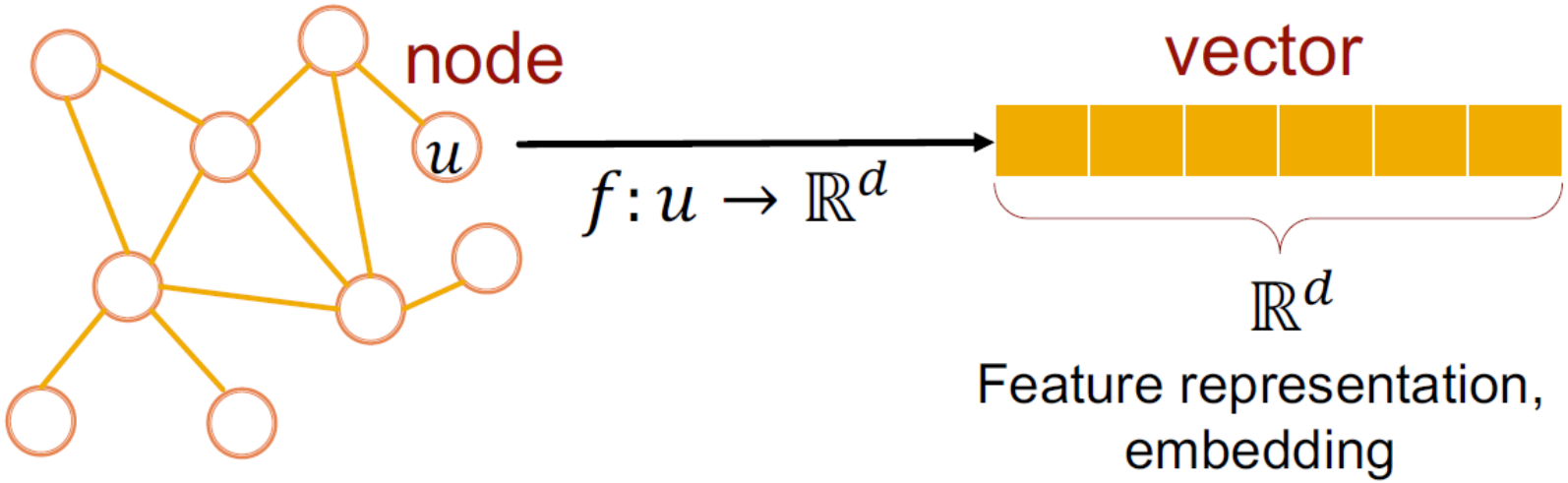
Generate simulations of complex physics over time

Physical simulation [2]

[1] Xiong, Jiacheng, et al. "Graph neural networks for automated de novo drug design." *Drug Discovery Today* 26.6 (2021): 1382-1393.
[2] Sanchez-Gonzalez, Alvaro, et al. "Learning to simulate complex physics with graph networks." *International conference on machine learning*. PMLR, 2020.

Graph Representation Learning

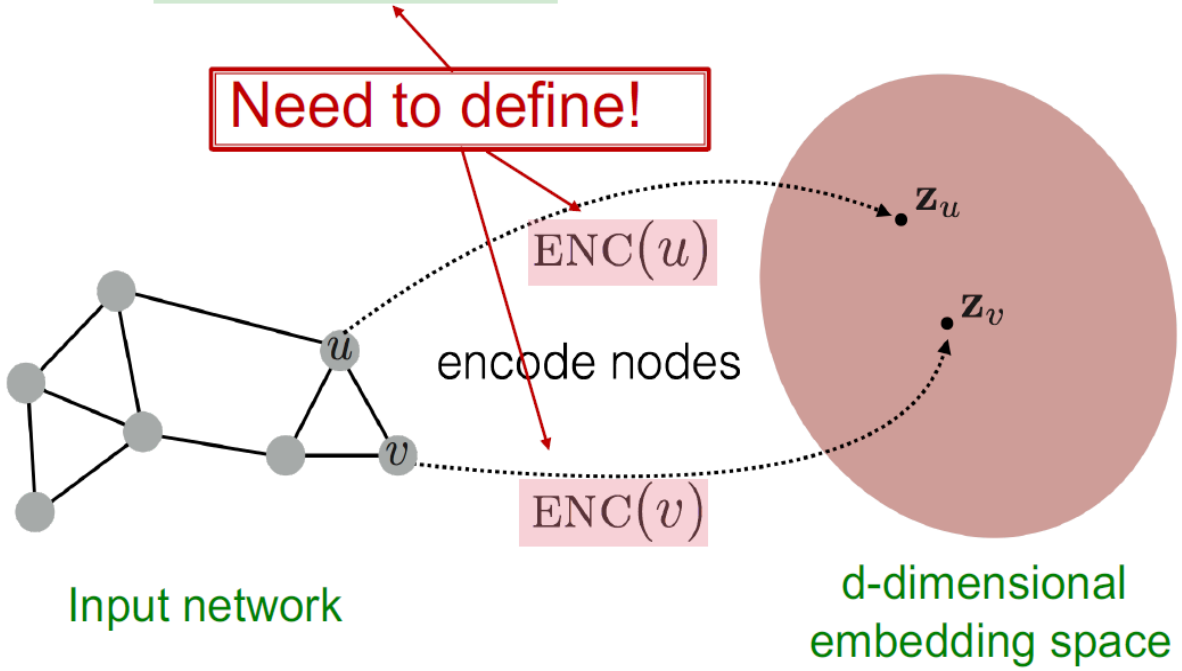
Map nodes to d-dimensional embeddings such that similar nodes in the network are embedded close together



How to learn the mapping function f?

Graph Representation Learning

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$



- **Encoder:** Maps each node to a low-dimensional vector

$$\text{ENC}(v) = \mathbf{z}_v$$

$\mathbf{z}_v$: d-dimensional embedding

v: node in the input graph

- **Similarity function:** Specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

Decoder

$\text{similarity}(u, v)$: Similarity of u and v in the original network

$\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$: dot product between node embeddings

Shallow encoding: encoder is just an embedding-lookup

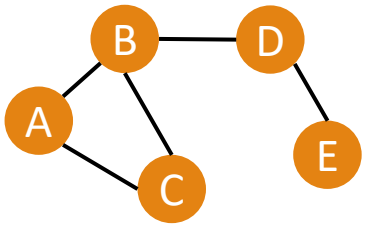
Graph Representation Learning

- **Limitations of shallow embedding methods:**
  - $O(|V|)$ parameters are needed
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - Inherently "transductive"
    - Cannot generate embeddings for nodes that are not seen during training

- Transduction learning
  - In contrast to 2 other learning types: inductive learning and deductive learning
  - Transduction predict specific output for specific examples within the same data distribution as the training data.

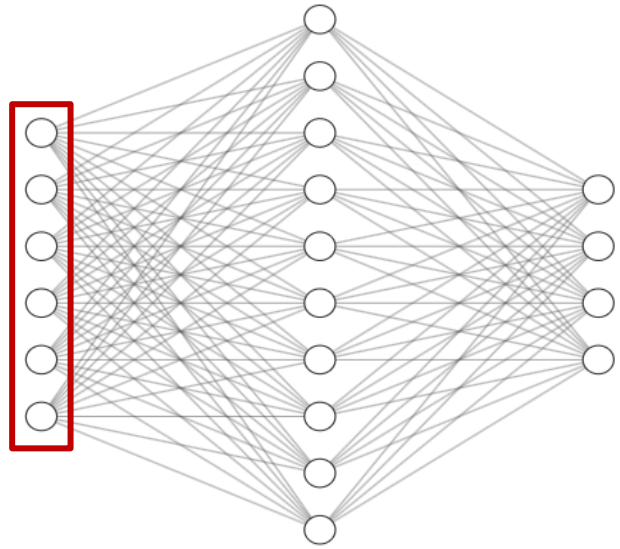
Deep Graph Encoders

- **We will discuss deep neural networks on graph: Graph Neural Networks (GNNs)**
  - $\mathrm{ENC}(v) =$ Multiple layers of perceptrons based on graph structure

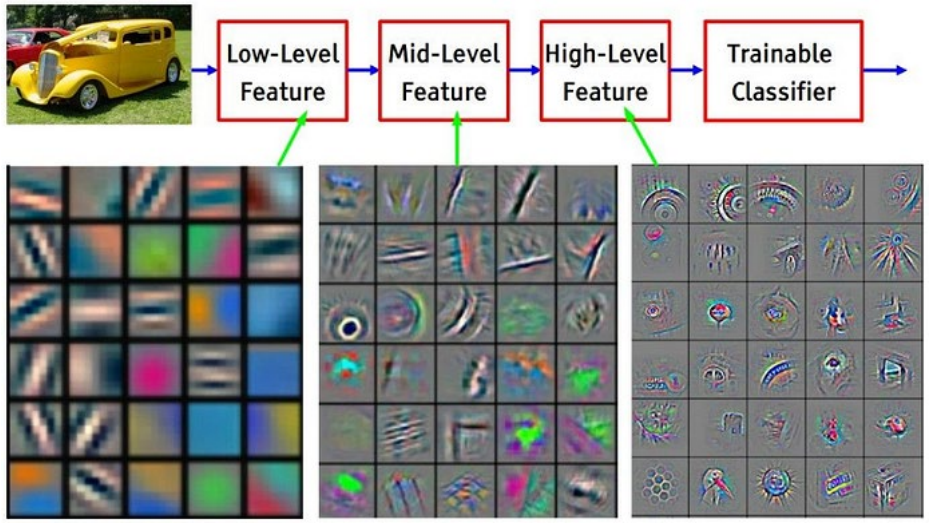


$\mathrm{G} = (\mathrm{V}, \mathrm{E})$

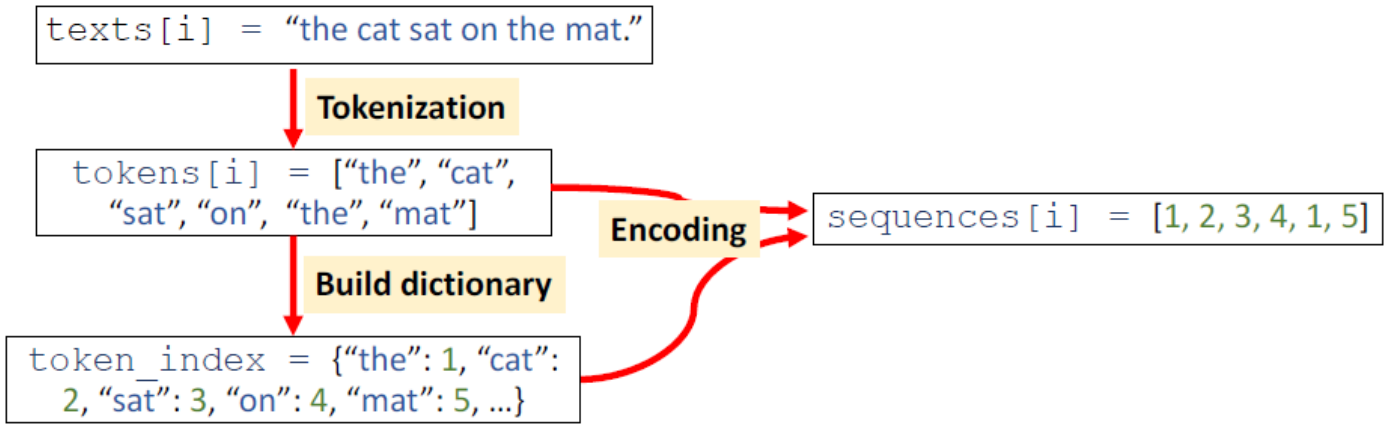
	A	B	C	D	E	Feat	
A	0	1	1	0	0	1	0
B	1	0	1	1	0	0	0
C	1	1	0	0	0	0	1
D	0	1	0	0	1	1	1
E	0	0	0	1	0	1	0



Neural Networks we have learned

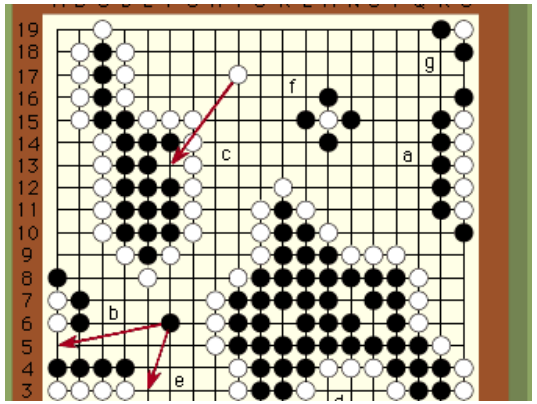


Feature map of image data: convolution



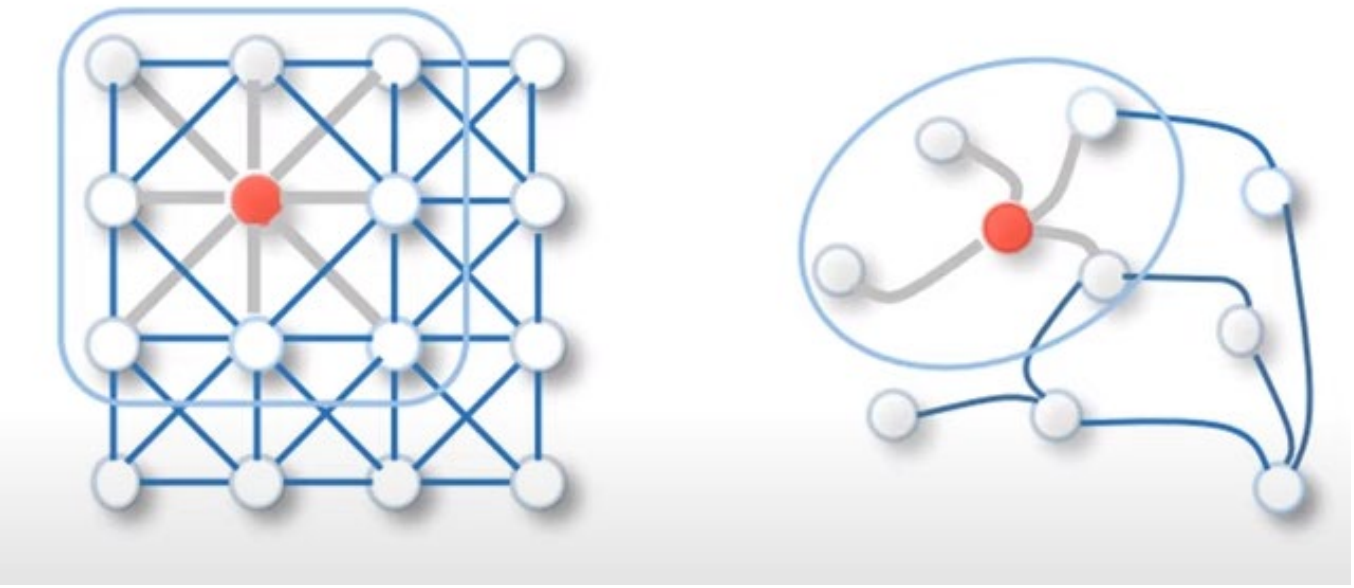
Natural language processing: tokenization and encoding

Speech data: spectrum



Grid based games: Go

Differences with image and graph data



- Arbitrary size
- Complex topological structure
- No fixed node ordering
- Often dynamic and have multimodal features

- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant

Setup (Notation)

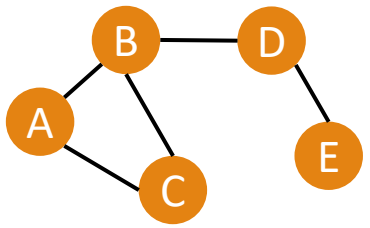
- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{|V| \times m}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: 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]

We often notate each neighbor of node v as $u \in N(v)$

This is called feature augmentation

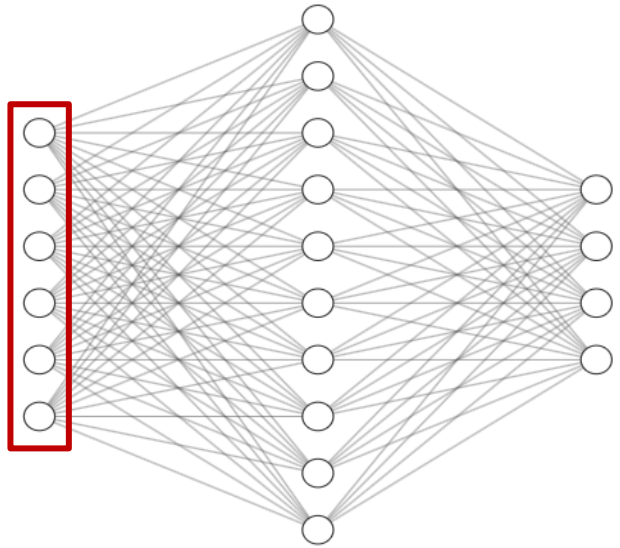
Naïve MLP Approach

- Concatenate adjacency matrix and features, then feed into a deep neural network



$G = (V, E)$

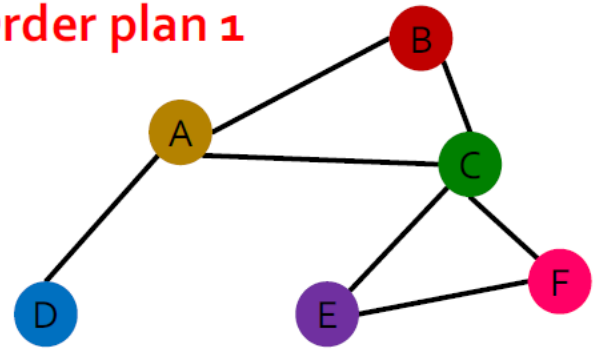
	A	B	C	D	E	Feat	
A	0	1	1	0	0	1	0
B	1	0	1	1	0	0	0
C	1	1	0	0	0	0	1
D	0	1	0	0	1	1	1
E	0	0	0	1	0	1	0



- Issues
  - Parameters $O(|V|)$
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

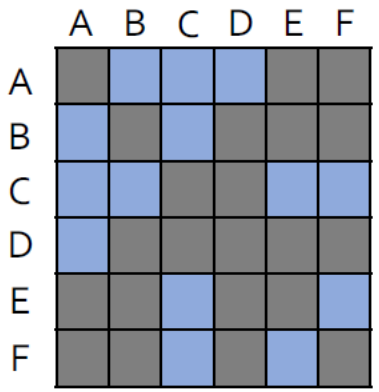
Permutation Invariance

Order plan 1

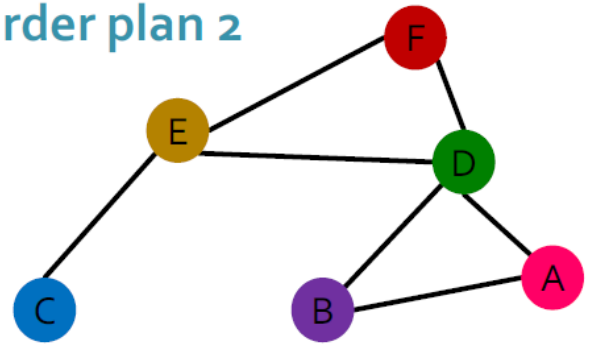


Node features X_1

Adjacency matrix A_1

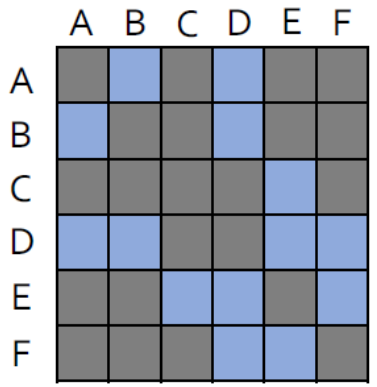


Order plan 2



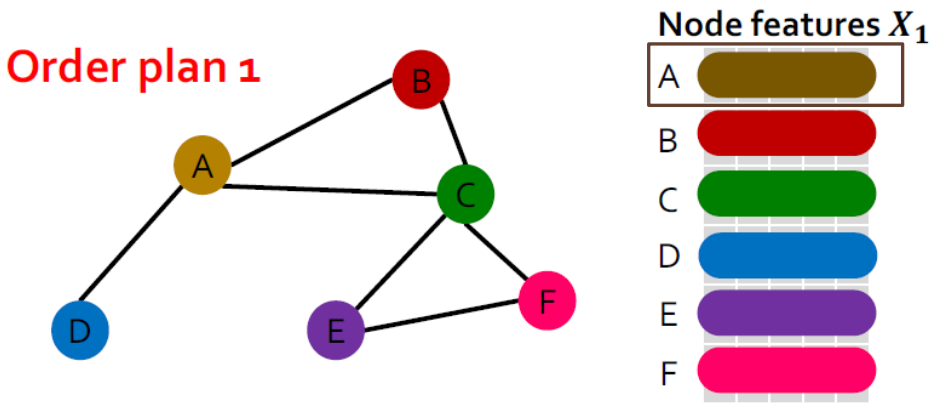
Node features X_2

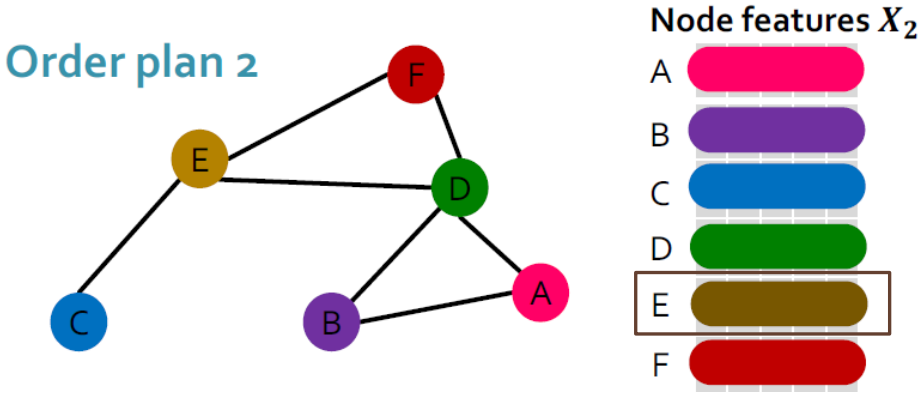
Adjacency matrix A_2



- Graph representations should be the same for Order plan 1 and Order plan 2
- Suppose a function f maps a graph to a d-dim embedding, then $f(A_1, X_1) = f(A_2, X_2)$
- **Definition:** For any graph function f, it is **permutation-invariant** if $f(A, X) = f(PAP^T, PX)$ for any permutation P.
- Permute the input, the output stays the same.

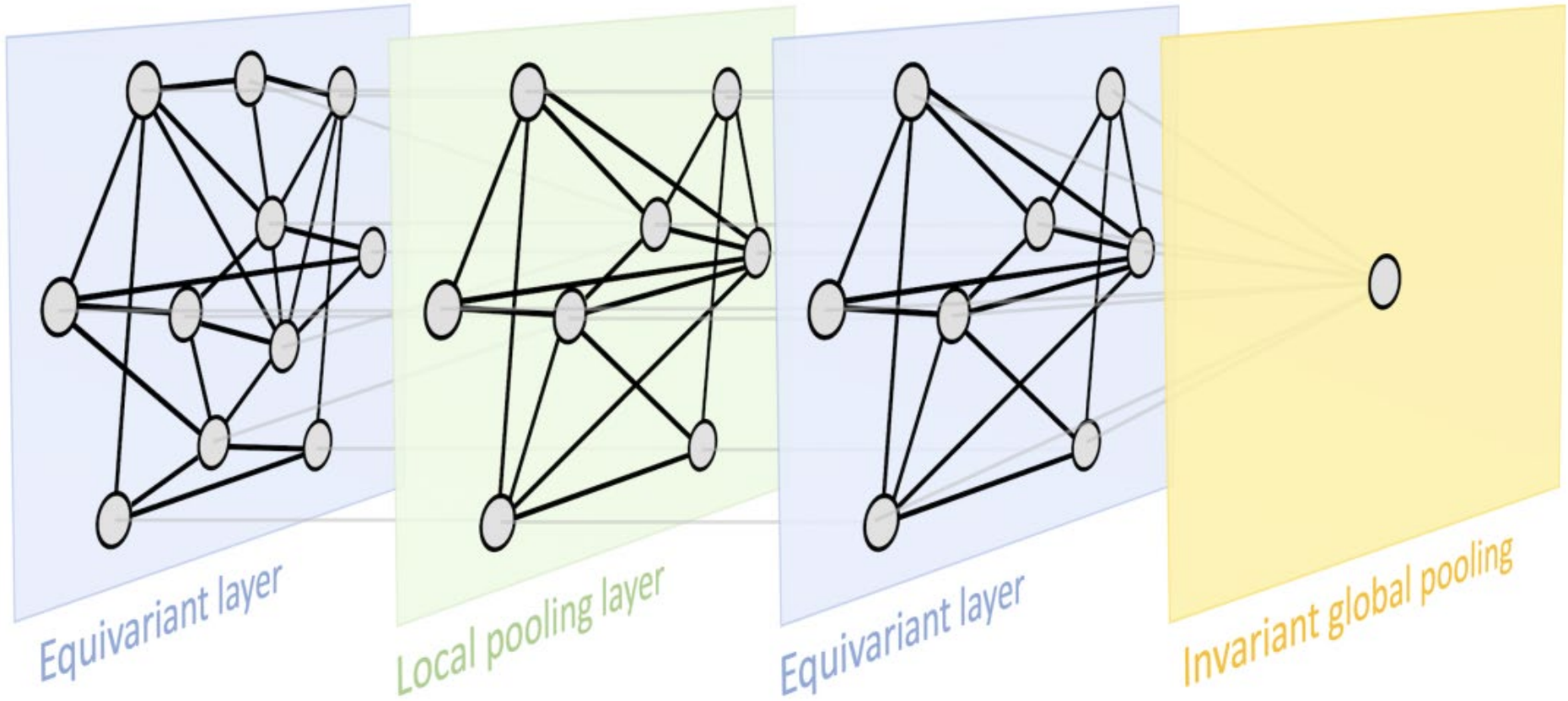
Permutation Equivariance





- Node representations: Learn a function f that maps nodes of $G = (\boldsymbol{A}, \boldsymbol{X})$ to a matrix $\mathbb{R}^{|V| \times d}$
- For two order plans, the vector of node at the same position in the graph is the same.
- If the output vector of a node at the same position in the graph remains unchanged for any order, we say f is permutation equivariant.
- **Definition:** For any node function f, it is **permutation-equivariant** if $\mathrm{P}f(A, X) = f(PAP^T, PX)$ for any permutation P.
- Permute the input, output also permutes accordingly.

Graph Neural Network Overview



Geometric Deep Learning blueprint, exemplified on a graph. A typical Graph Neural Network architecture may contain permutation equivariant layers (computing node-wise features), local pooling (graph coarsening), and a permutation-invariant global pooling layer (readout layer) [1].

[1] Bronstein, Michael M., et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges." *arXiv preprint arXiv:2104.13478* (2021).

What is a GNN/DGN

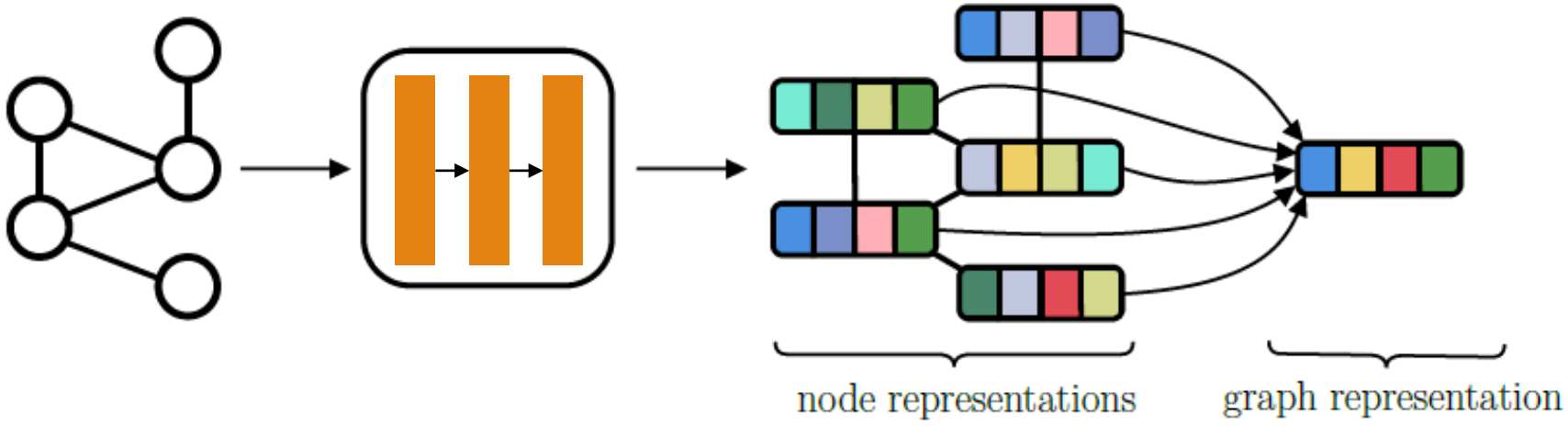


Figure 2: The bigger picture that all graph learning methods share. A "Deep Graph Network" takes an input graph and produces node representations $\mathbf{h}_v \ \forall v \in \mathcal{V}_g$. Such representations can be aggregated to form a single graph representation $\mathbf{h}_g$.

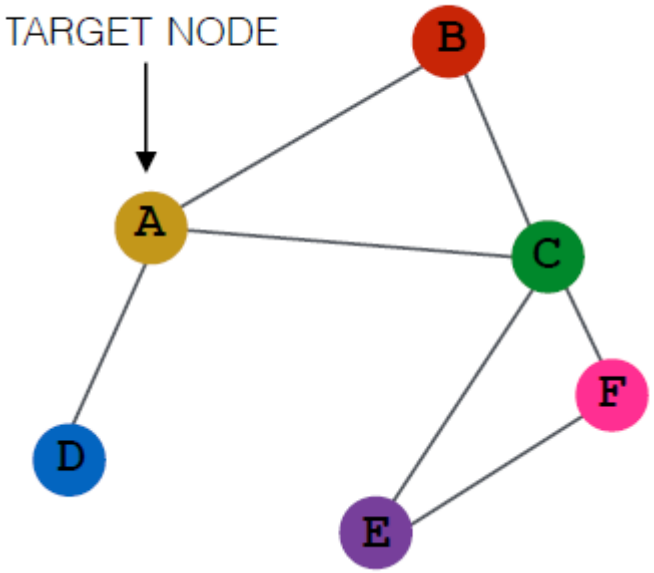
By passing and aggregating information from neighbours, we could design permutation invariant / equivariant networks!

Via Message Passing Layers

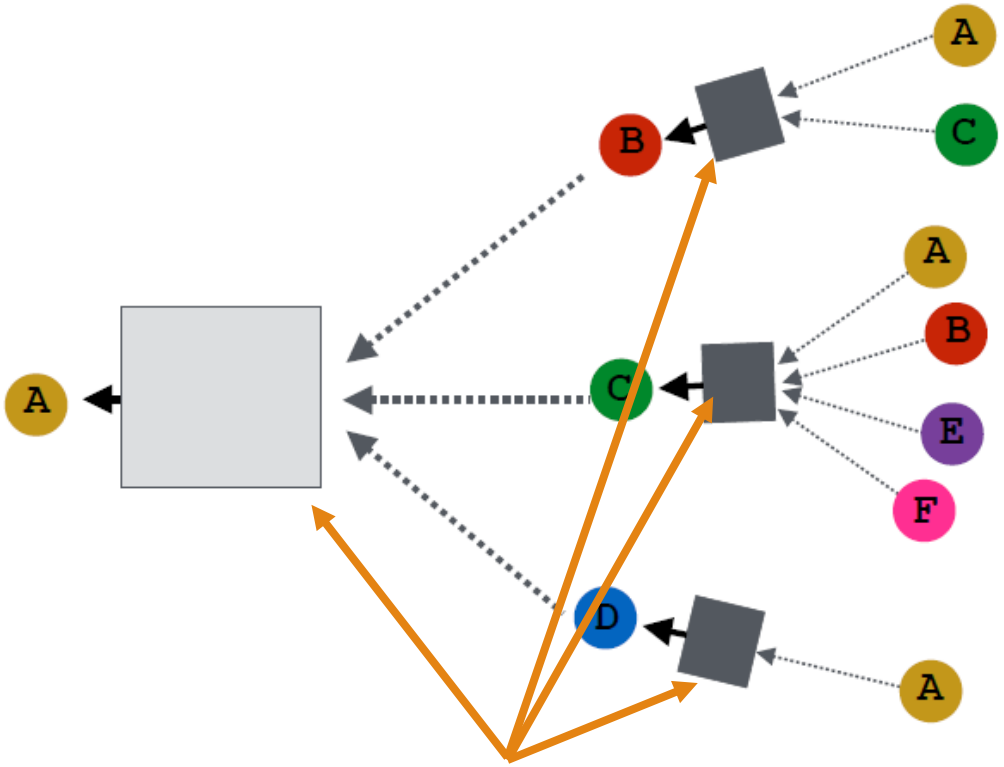
[1] Bacciu, Davide, et al. "A gentle introduction to deep learning for graphs." *Neural Networks* 129 (2020): 203-221.

Idea: Aggregate neighbours

- Key Idea: Generate node embeddings based on local network neighbourhoods

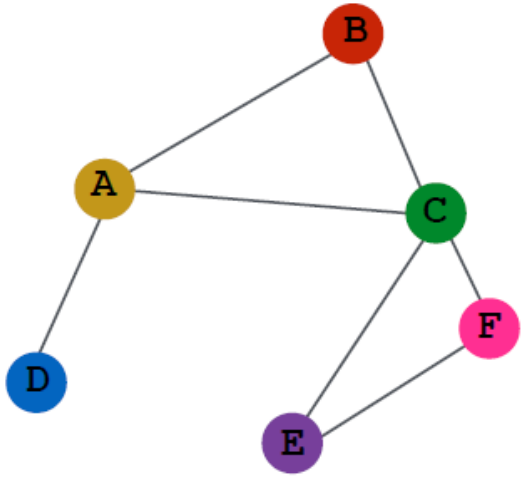


Input Graph

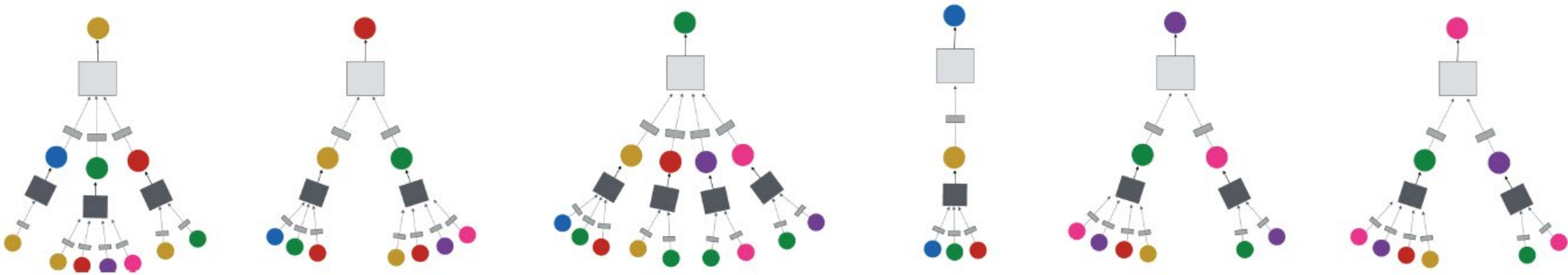


- Via neural networks

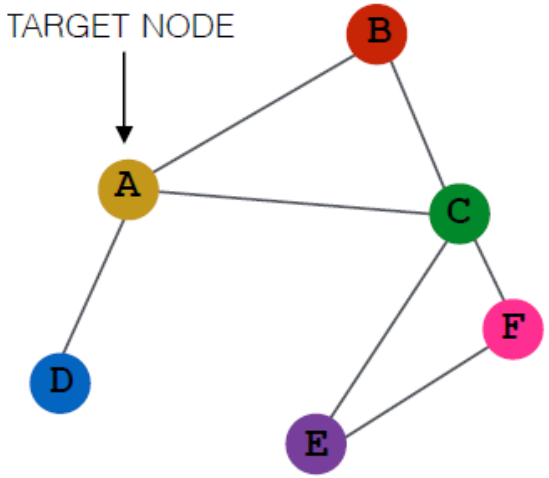
Idea: Aggregate neighbours from 1 layer



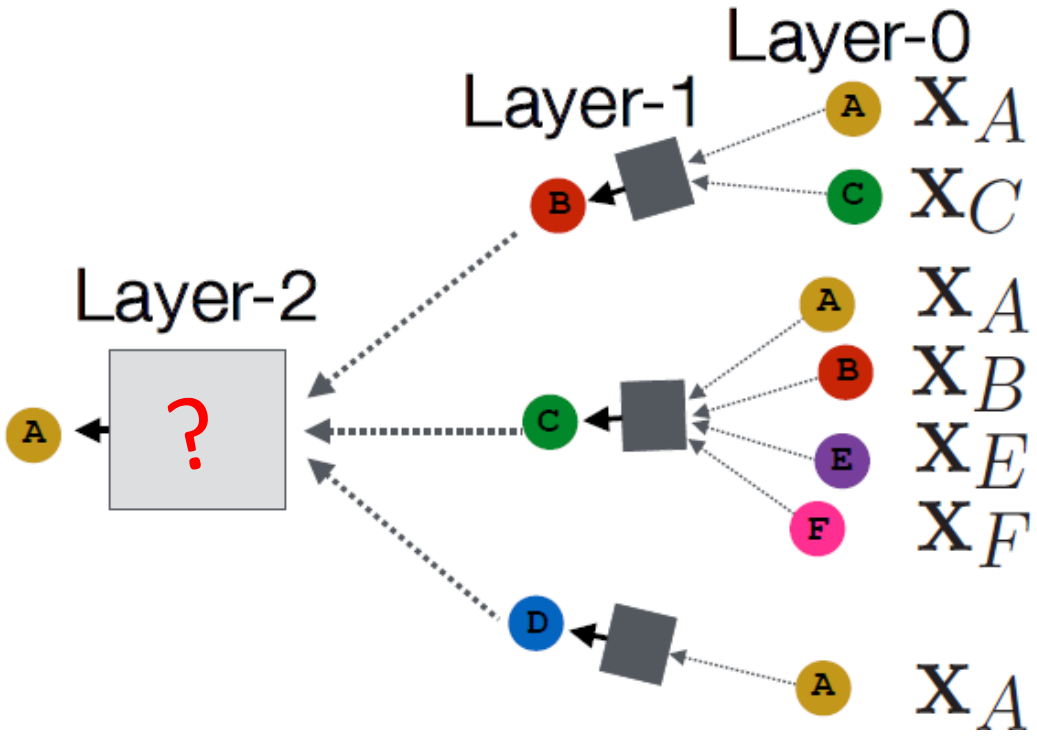
- Every node defines a computation graph based on its neighbourhood



Idea: Aggregate neighbours across layers

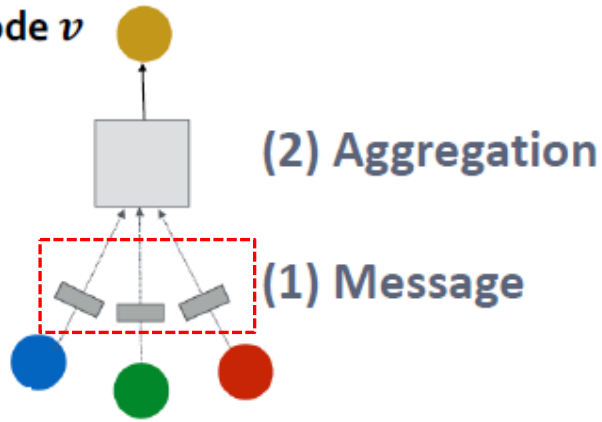


- Model can be of arbitrary depth
  - Layer-0 embedding of node v is the input feature x_v
  - Layer-k embedding gets information from nodes k hops away



A Single GNN Layer

- Construct a GNN Layer
  - 1) Message passing
  - 2) Aggregation



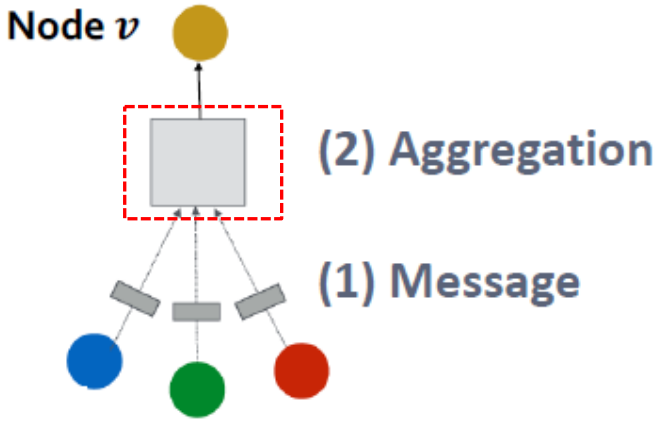
- **(1) Message computation**
  - **Message function**: $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
    - **Intuition:** Each node will create a message, which will be sent to other nodes later
    - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$
      - Multiply node features with weight matrix $\mathbf{W}^{(l)}$
- To avoid information loss from node v, also include $\mathbf{h}_v^{(l-1)}$
  - **(1) Message: compute message from node v itself**
    - Usually, a **different message computation** will be performed

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

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

A Single GNN Layer

- Construct a GNN Layer
  - 1) Message passing
  - 2) Aggregation
  - 3) Update
  - 4) Readout (Graph-level)



- **(2) Aggregation**
  - **Intuition:** Node v will aggregate the messages from its neighbors u:

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

  - **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
    - $\mathbf{h}_v^{(l)} = \mathrm{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$
- **(3) Update**
  - After aggregating from neighbors, we aggregate the message from node v itself, with an "update" operation.

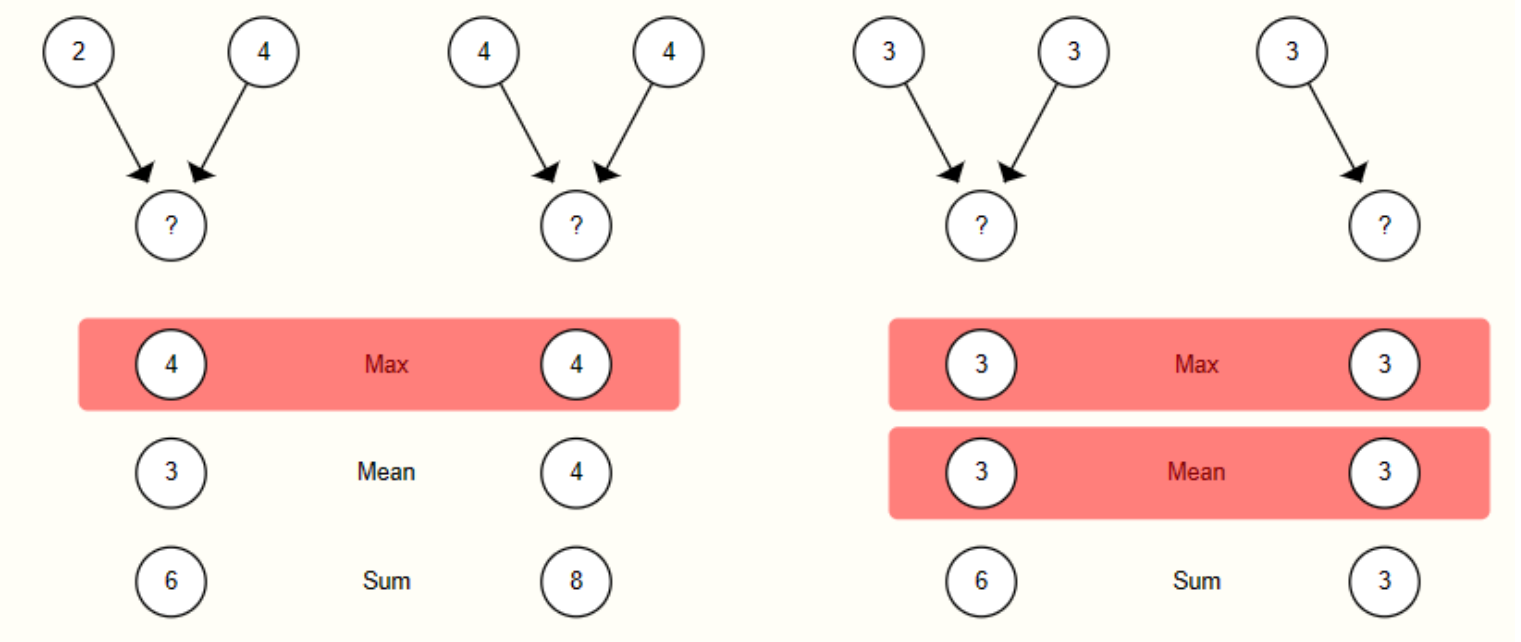
Then aggregate from node itself

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

Update to new state of node v

First aggregate from neighbors

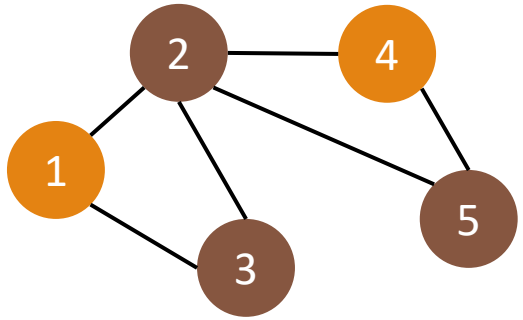
Aggregation Operations



There is no operation that is uniformly the best choice.

Matrix Multiplication Process

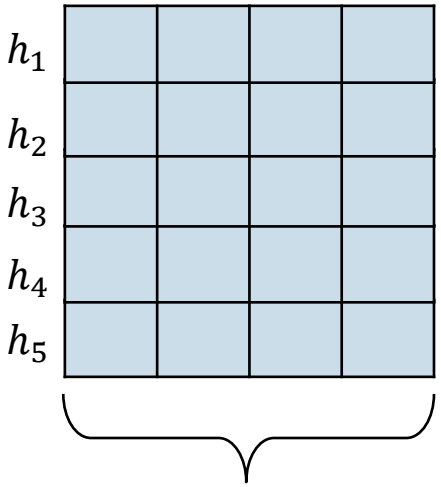
0.4	1	0	1.5



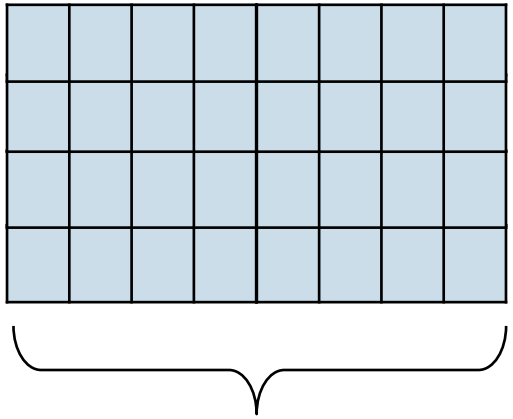
$$\boldsymbol{h'}_v = \sigma\left(\sum_{u \in N(v)} \mathbf{W} * \boldsymbol{h}_u\right)$$

Adjacency Matrix [5, 5]

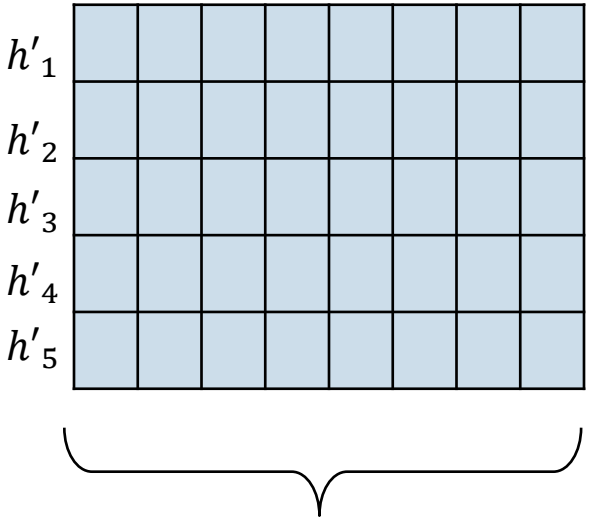
0	1	1	0	0
1	0	1	1	1
1	1	0	0	0
0	1	0	0	1
0	1	0	1	0



Features per node [5, 4]



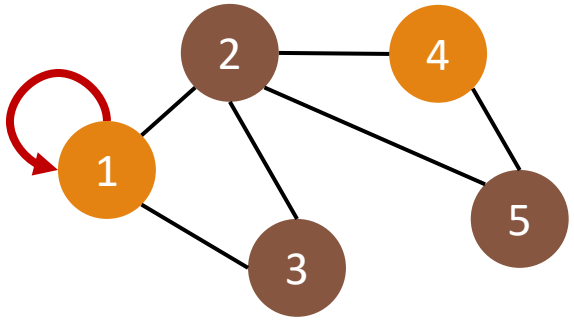
Learnable weight matrix[4, 8]



Embedding per node [5, 8]

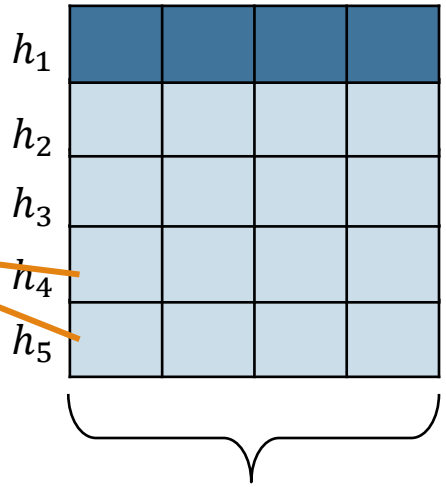
Matrix Multiplication Process

$$\boldsymbol{h'}_v = \sigma\left(\sum_{u \in N(v)} \mathbf{W} * \boldsymbol{h}_u\right)$$

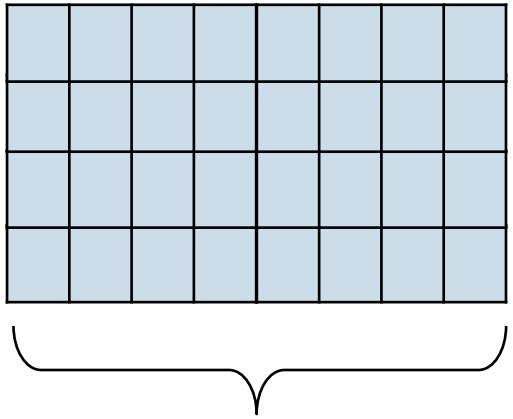


Adjacency Matrix [5, 5]

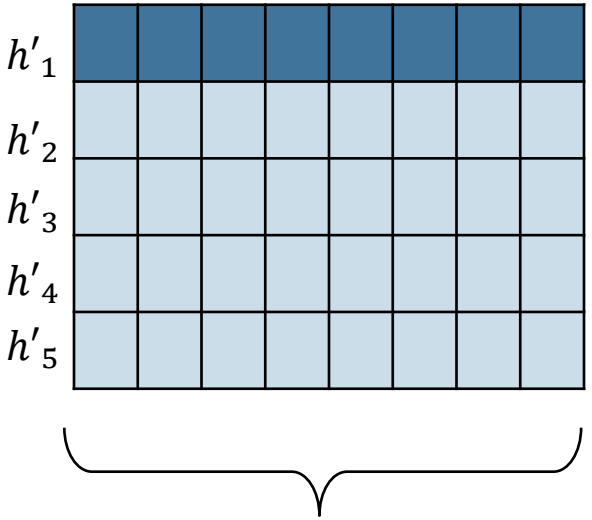
1	1	1	0	0
1	1	1	1	1
1	1	1	0	0
0	1	0	1	1
0	1	0	1	1



Features per node [5, 4]



Learnable weight matrix[4, 8]



Embedding per node [5, 8]

A Single GNN Layer

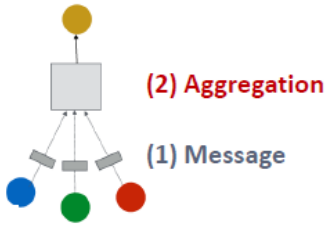
- **Putting things together:**
  - **(1) Message**: each node computes a message

$$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$

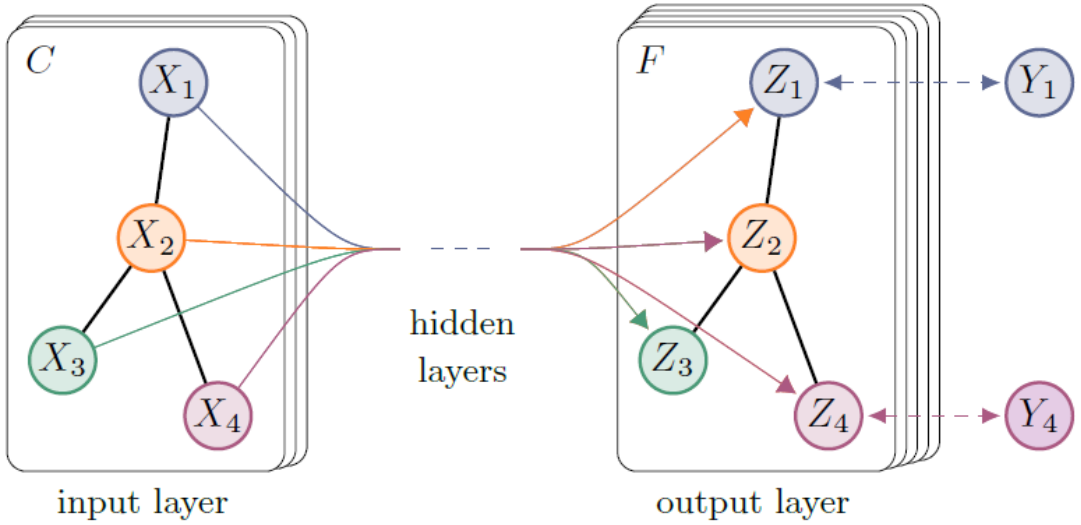
  - **(2) Aggregation**: aggregate messages from neighbors

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

  - **Nonlinearity (activation):** Adds expressiveness
    - Often written as $\sigma(\cdot)$. Examples: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, ...
    - Can be added to **message or aggregation**



5.1 Graph Convolution Networks



$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- Multi-layer Graph Convolutional Network with C input channels and F feature maps in the output layer (3-layer)
  - The graph structure is shared over layers
  - Labels denoted by Y_i
- Input-to-hidden weight matrix $W^{(0)} \in \mathbb{R}^{C \times H}$ for a hidden layer with H feature maps
- Hidden-to-output weight matrix $W^{(1)} \in \mathbb{R}^{H \times F}$.

[1] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." ICLR 2017 (arXiv 2016).

5.1 Graph Convolution Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$

Normalized by node degree

(This is a simplified version. Not the optimal form in the GCN paper[1])

- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

In GCN the input graph is assumed to have self-edges that are included in the summation.

[1] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." ICLR 2017 (arXiv 2016).

5.1 Graph Convolution Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

$$= \sigma\left(\widehat{\mathbf{D}}^{-1}\widehat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{l}\right)$$

where $\widehat{\mathbf{A}} = \mathbf{A} + \boldsymbol{I}$, $\boldsymbol{I}$ is the identity matrix indicating the self-loop of node v,$\widehat{\mathbf{D}}$ is the diagonal node degree matrix

$$\mathbf{h}_v^{(l)} = \sigma\left(\widehat{\mathbf{D}}^{-\frac{1}{2}}\widehat{\mathbf{A}}\widehat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{W}^{l}\right)$$

- To counteract the scale discrepancy and ensure that each neighbor's contribution is considered fairly, normalization is applied. Normalizing $\mathbf{A}$ such that all rows sum to one gets rid of the issue[1].
- Normalization in GCNs is akin to saying, "Let's not let the quantity of connections overshadow the quality of information each neighbor brings."
- In practice, we use a symmetric normalization $\mathbf{D}^{-\frac{1}{2}}\widehat{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}$ to the adjacency matrix. This form is derived from the symmetric Laplacian in spectral graph theory. We will skip the derivation of the formulas here.
- For details, please refer to [2] (the original classic paper)

[1] https://tkipf.github.io/graph-convolutional-networks/

[2] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." ICLR 2017 (arXiv 2016).

5.1 Graph Convolution Networks

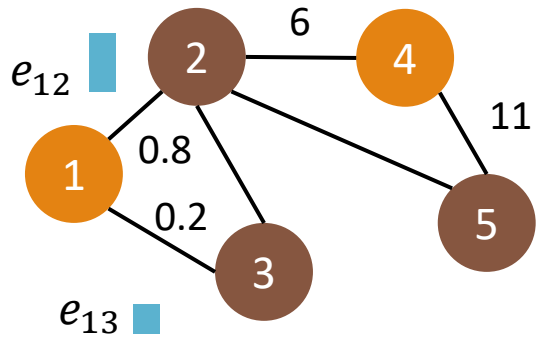
- Key drawbacks of GCN:
  - 1. During the aggregation process, all neighbors are equally important to node v, so the edge weights are fixed, lacking flexibility.
  - 2. Scalability is poor because it involves graph convolution fusion over the entire graph. Gradient updates are performed on the entire graph, which becomes too slow and unsuitable when the graph is relatively large.
  - 3. As the number of layers increases, the results tend to become overly smooth, with the feature representations of individual nodes becoming highly similar.

5.2 Graph Attention Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

- In GCN, $\alpha_{vu} = \frac{1}{|N(v)|}$ is weighting factor for every neighboring node u's message to node v (importance)
- Not all nodes' neighbors are equally important in GAT idea
  - Attention is inspired by the popular NLP attention mechanism -> learning message importance
  - α_{vu} is called Attention Weights in GAT
  - The attention α_{vu} focuses on the important parts of the input data and ignores the rest.
  - Which part of the data is more important depends on the context and is learned during training.
- Attention mechanism a computes attention coefficients e_{vu} across pairs of nodes u, v
  - Take it as a weight for each edge of the neighbors that tells us how much attention we should pay to that specific node.
  - $e_{vu} = a(\mathbf{W}\boldsymbol{h}_v, \mathbf{W}\boldsymbol{h}_u)$
- In fact, NLP on text data and GNN on graph data is similar. We can see the sentence as a graph, the words as nodes, and the word vectors as node feature vectors.

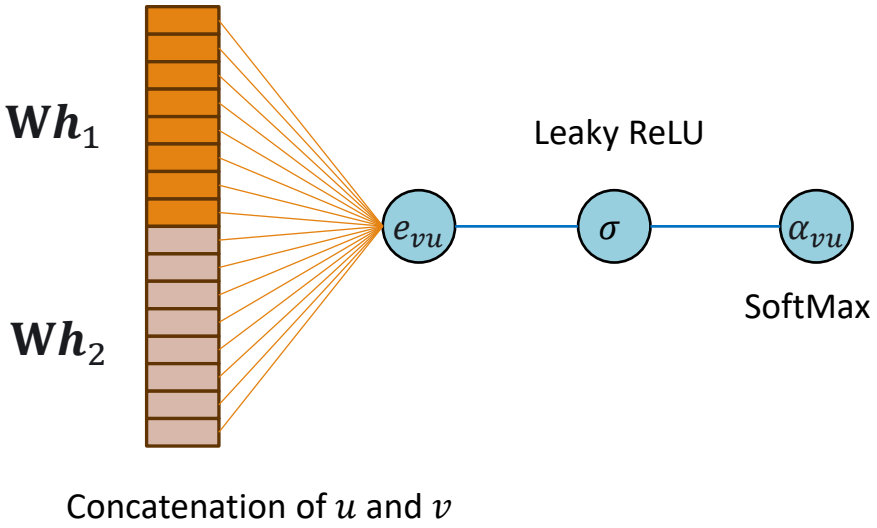
5.2 Attention Weights



$$e_{vu} = a(\mathbf{W}\boldsymbol{h}_v, \mathbf{W}\boldsymbol{h}_u)$$

$$\alpha_{vu} = \text{softmax}_v(e_{vu}) = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})} = \frac{\exp\big(a(\mathbf{W}\boldsymbol{h}_v, \mathbf{W}\boldsymbol{h}_u)\big)}{\sum_{k \in N(v)} \exp\big(a(\mathbf{W}\boldsymbol{h}_v, \mathbf{W}\boldsymbol{h}_k)\big)}$$

- We can the attention method “mask attention” because we mask out all the other nodes that are not 1-neighbor to the node v during the calculation of α.

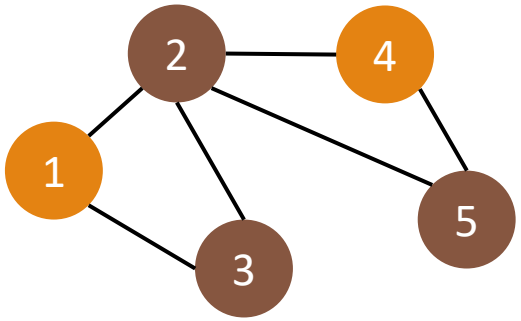


$$\alpha_{vu} = \frac{\exp\big(\text{LeakyReLU}(\vec{a}^T[\mathbf{W}\boldsymbol{h}_v || \mathbf{W}\boldsymbol{h}_u])\big)}{\sum_{k \in N(v)} \exp\big(\text{LeakyReLU}(\vec{a}^T[\mathbf{W}\boldsymbol{h}_v || \mathbf{W}\boldsymbol{h}_k])\big)}$$

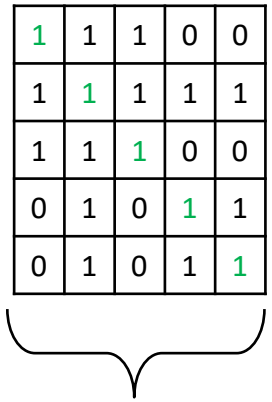
$$\vec{a} \in \mathbb{R}^{2F'}$$

Velickovic, Petar, et al. "Graph attention networks." *stat* 1050.20 (2017): 10-48550.

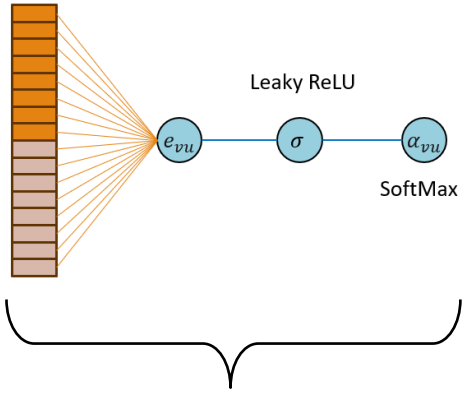
5.2 Graph Attention Networks



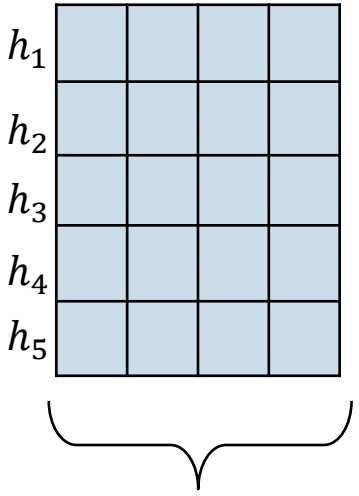
$$\boldsymbol{h'}_v = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W} * \boldsymbol{h}_u\right)$$



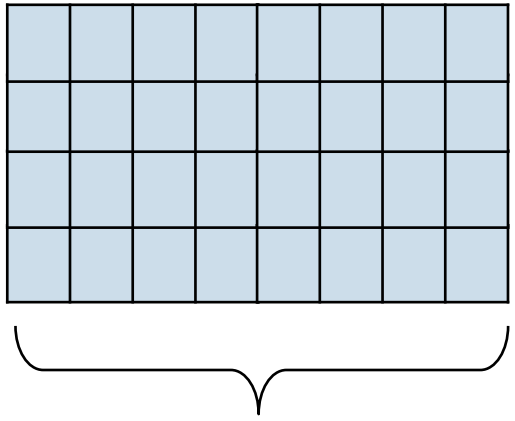
Adjacency matrix [5, 5]



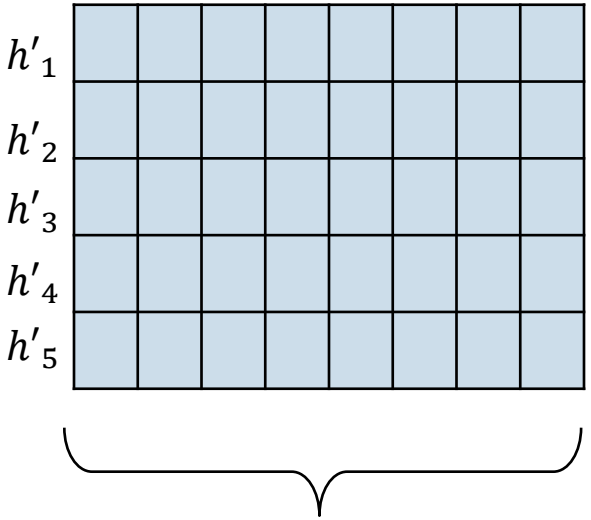
Masked attention



Features per node [5, 4]

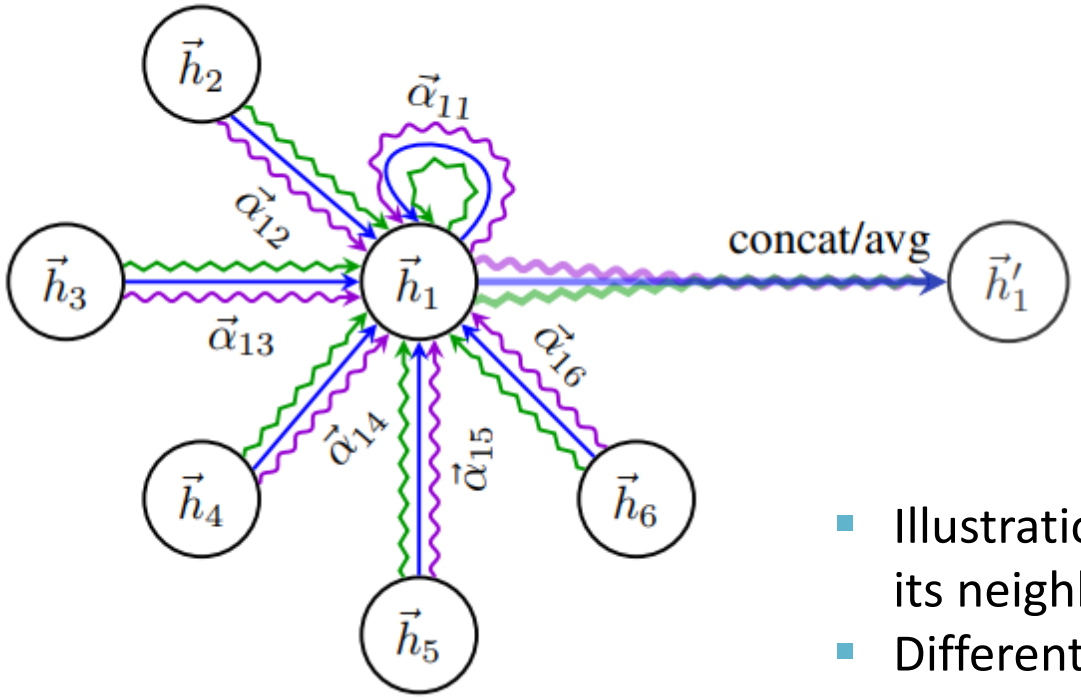


Learnable weight matrix [4, 8]



Embedding per node [5, 8]

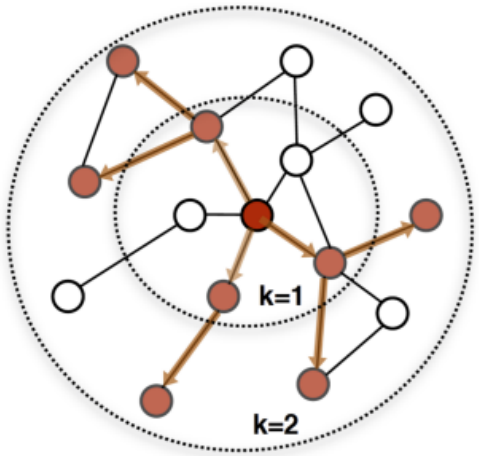
5.2 Graph Attention Networks



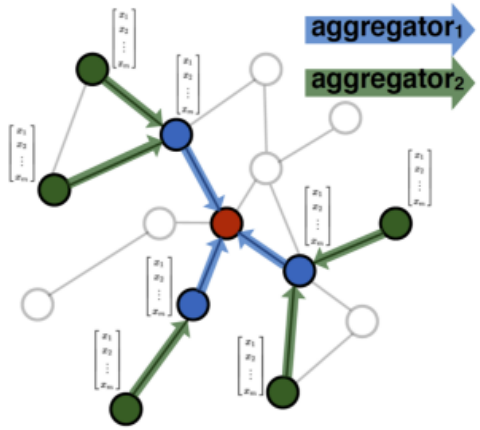
- Illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood.
- Different arrow styles and colors denote independent attention computations.
- The aggregated features from each head are concatenated or averaged to obtain the final $\vec{h}'_1$

5.3 GraphSAGE

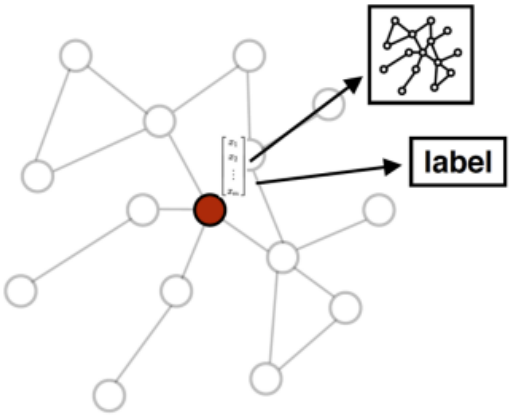
- Short for Graph Sample and AggreGatE [1]
  - Scalability: By sampling and aggregating information from a node's local neighborhood, GraphSAGE can generate embeddings without processing the entire graph. Suitable for large-scale graphs.
  - Inductiveness: GraphSAGE's learned function can generate embeddings for new nodes without retraining, as long as it has access to the node's attributes and its neighborhood structure. Suitable for dynamic graphs where new nodes and edges are continuously added.
  - Four Steps
    - Node Sampling
    - Feature Aggregation
    - Update
    - Normalization



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

[1] Hamilton et al. Inductive Representation Learning on Large Graphs, NIPS 2017

5.3 GraphSAGE

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \text{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$$

- Concatenate the self-node embeddings with the aggregated neighbor embeddings to increase expressive power on top of GCN.
- Then in most scenarios, we apply ℓ_2 normalization to $\mathbf{h}_v^{(l)}$ at every layer (the Euclidean length of the embedding vector $\left\|\mathbf{h}_v^{(l)}\right\|_2$ will always be equal to 1), which improves great performance.
- Inductive vs. Transductive Learning
  - Transductive learning models are trained on a specific graph and are designed to make predictions for nodes or edges that are part of the same graph seen during training. These models do not generalize to unseen nodes or graphs.
  - Inductive learning models are trained in a way that allows them to generalize to unseen nodes or entirely new graphs. They learn a function that can be applied to nodes not seen during training, based on node features and local graph structure.

[1] Hamilton et al. Inductive Representation Learning on Large Graphs, NIPS 2017

5.3 GraphSAGE Aggregation

- Mean: Take a weighted average of neighbors

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

- Pool: Transform neighbor vectors and apply symmetric vector function Mean(·) or Max(·)

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

- LSTM: Apply LSTM to the reshuffled neighbors recursively

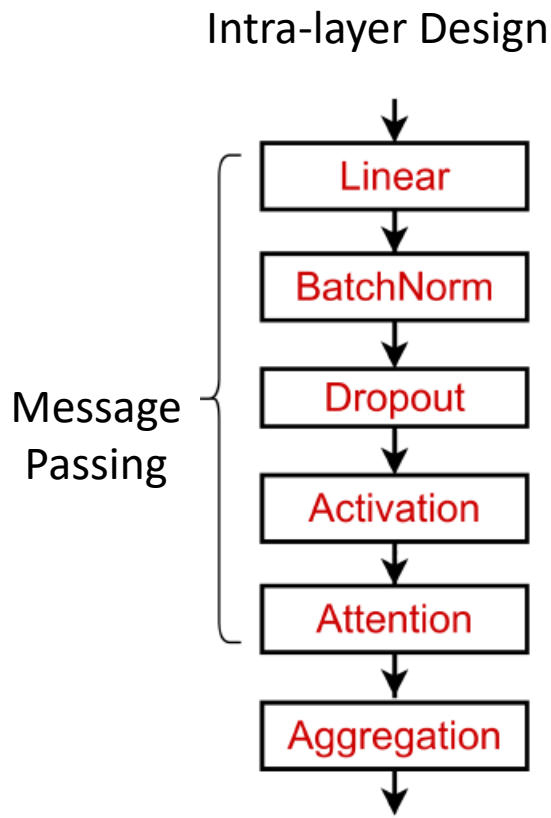
$$\text{AGG} = \text{LSTM}\left(\left[\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))\right]\right)$$

  - When we apply sequence model to the messages from the neighbors, we have to bear in mind that it is not order invariant
  - During training, we keep permuting the orderings so that we teach the sequence model to ignore the ordering of the received messages

[1] Hamilton et al. Inductive Representation Learning on Large Graphs, NIPS 2017

6.1 GNN Layer Design

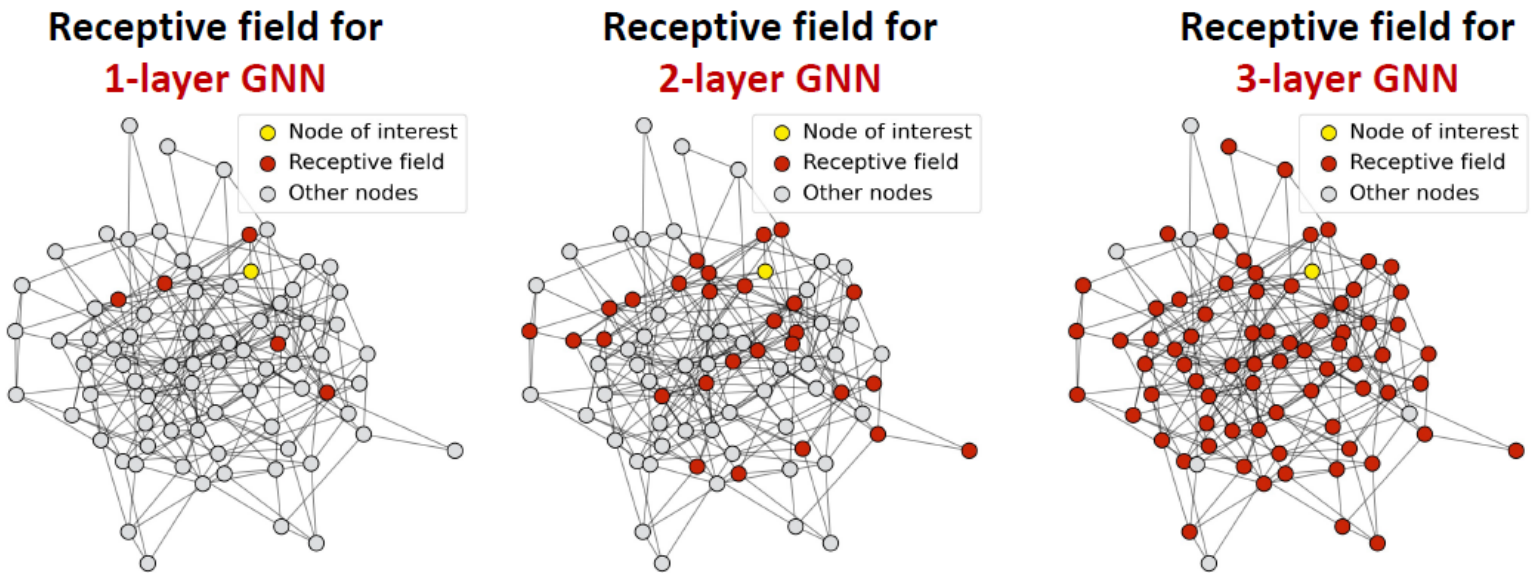
- In practice, we introduce modern deep learning modules to improve the performance[1].
- Batch Normalization[2]
  - Re-center the node embeddings into zero mean
  - Re-scale the variance into unit variance
  - Stabilize neural network training process
- Dropout
  - Prevent overfitting
- Attention or Gating
  - Control the importance of a message
- And more …



[1] J. You, R. Ying, J. Leskovec. Design Space for Graph Neural Networks, NeurIPS 2020
[2] S. Loffe, C.Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

6.1 GNN Layer Design

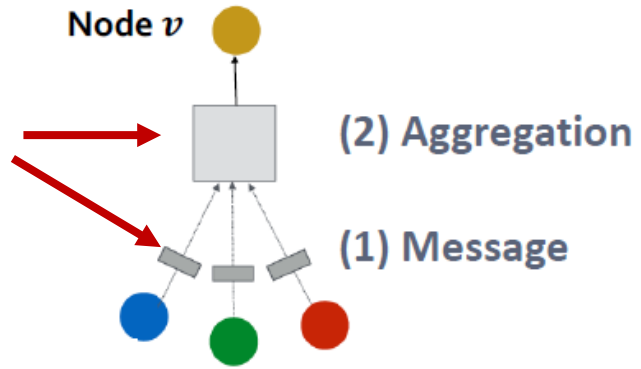
- GNN suffers from over-smoothing problem when stacking many GNN layers
  - All the node embeddings converge to the same value
- Receptive field: the set of nodes that determine the embedding of a node of interest
  - In a K-layer GNN, each node has a receptive field of K-hop neighborhood
  - The shared neighbors quickly grows when layers increase



6.1 GNN Layer Design

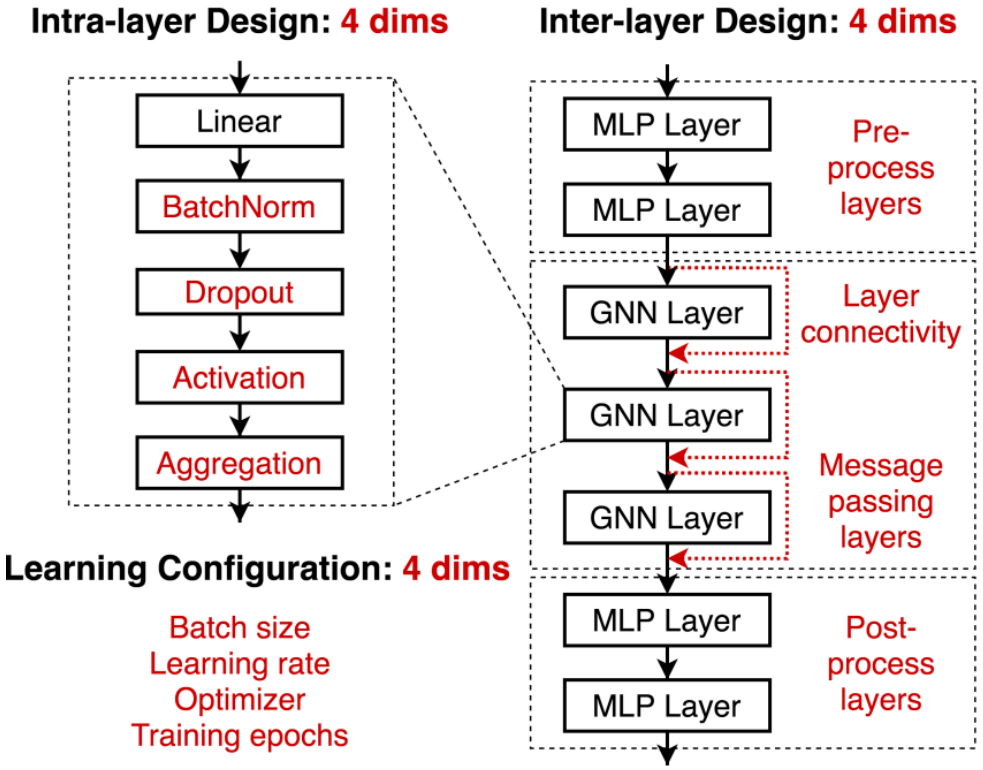
- We have to make a shallow but a powerful GNN ☹
- Solution: Increase the expressive power within each GNN layer
  - We can make message passing / aggregation process a deep neural network!

If needed, each box can contain a 3-layer MLP



6.1 GNN Layer Design

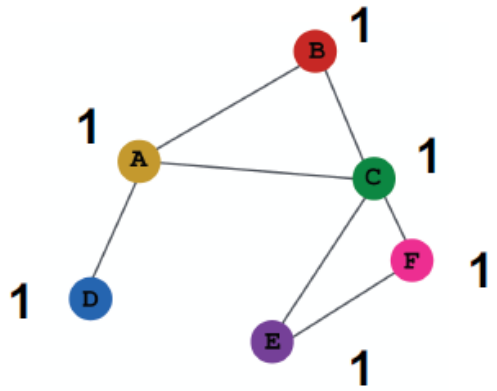
- We have to make a shallow but a powerful GNN ☹
- Solution 2: Add layers that do not pass messages
  - Not all layers in a GNN have to be GNN layers
  - We can have MLP layers (applied to each node) before and after GNN layers

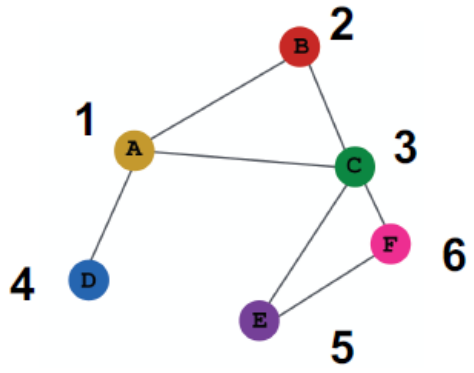


- Pre-processing layers
  - Important when encoding node features is necessary
  - E.g., when node represent images / text
- Post-processing layers:
  - Important when reasoning / transformation over node embeddings are needed
  - E.g., graph classification, knowledge graphs
- Add skip connections
  - Like residual network, adding **shortcuts** in GNN can increase the impact of earlier layers

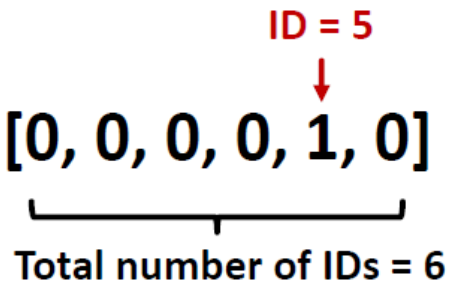
6.2 Graph Feature Manipulation

- Sometimes the input graph does not have node features
  - Quite common scenario when we only have adjacency matrix
- Solution 1: Assign constant values to nodes
- Solution 2: Assign unique IDs to nodes (**one-hot vectors**)



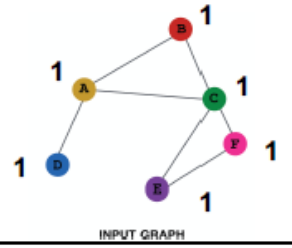
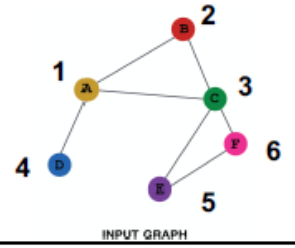


One-hot vector for node ID=5



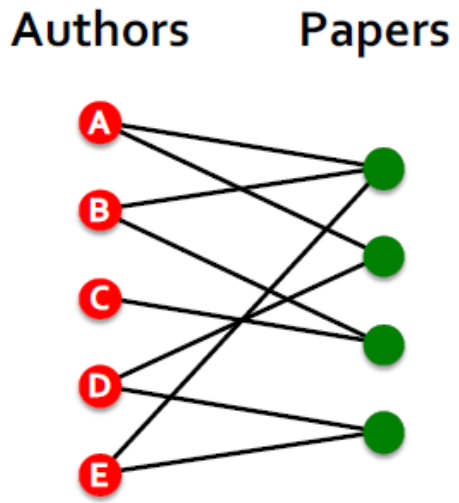
6.2 Graph Feature Manipulation

- Feature augmentation approach: **constant** vs.. **one-hot vectors**

	Constant node feature	One-hot node feature
		
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**Low**. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	**Low**. Only 1 dimensional feature	**High**. High dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

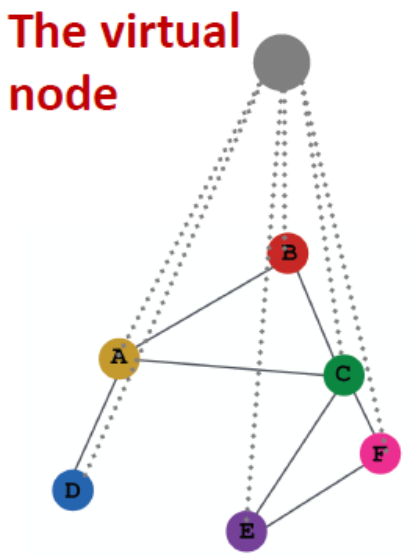
6.3 Graph Structure Manipulation

- Augment sparse graphs
- Solution 1: Add virtual edges
  - **Common approach:** Connect 2-hop neighbors via virtual edges
  - **Intuition:** Instead of using adj. matrix A for GNN computation, use $A + A^2$
  - **Use cases:** Bipartite graphs
    - Author-to-papers (they authored)
    - 2-hop virtual edges make an author-author collaboration graph



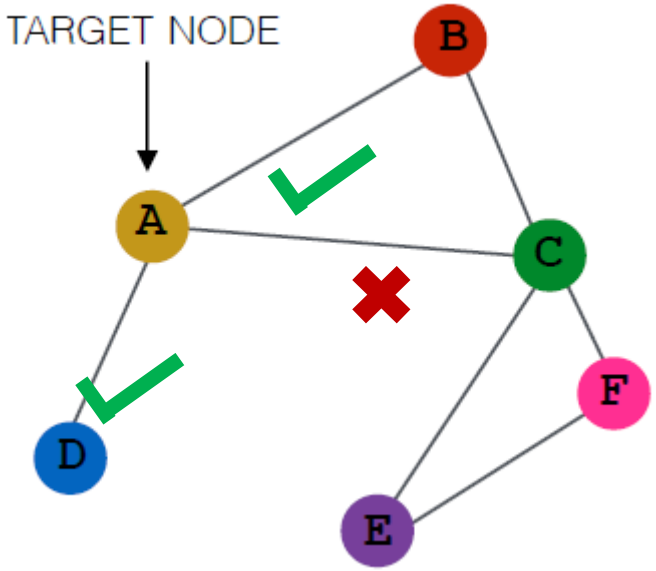
6.3 Graph Structure Manipulation

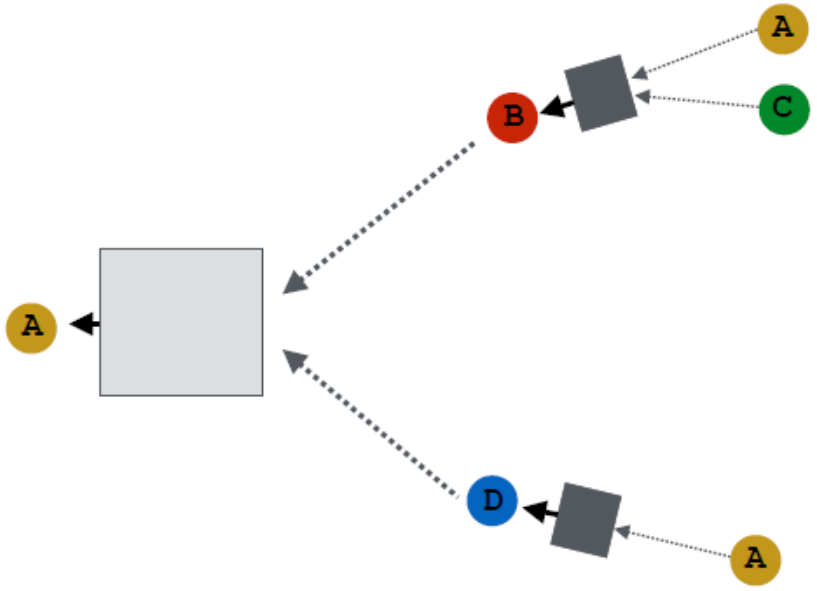
- Augment sparse graphs
- Solution 2: Add virtual nodes
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of 2**
      - **Node A – Virtual node – Node B**
  - Greatly improves message passing efficiency in sparse graphs



6.3 Graph Structure Manipulation

- For dense / large graphs, high-degree nodes
- Solution: not all the neighbors are used for message passing[1]
  - Randomly determine a node's neighborhood for message passing
  - For example, we can randomly choose 2 neighbors to pass messages
    - Only nodes *B* and *D* will pass message to *A*

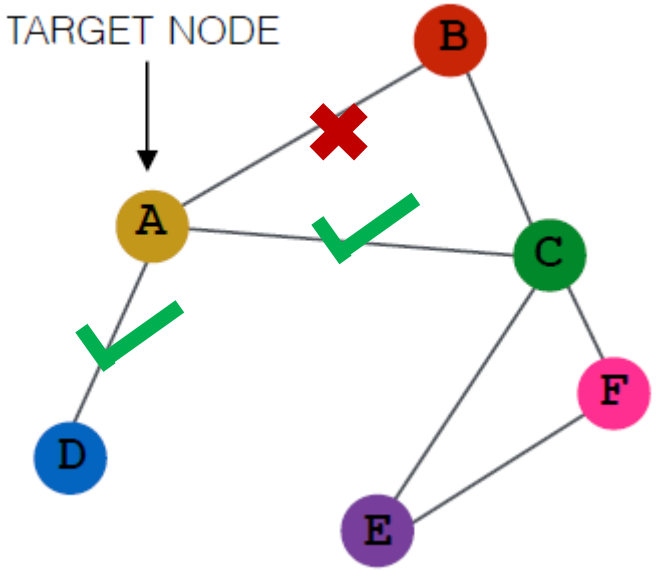


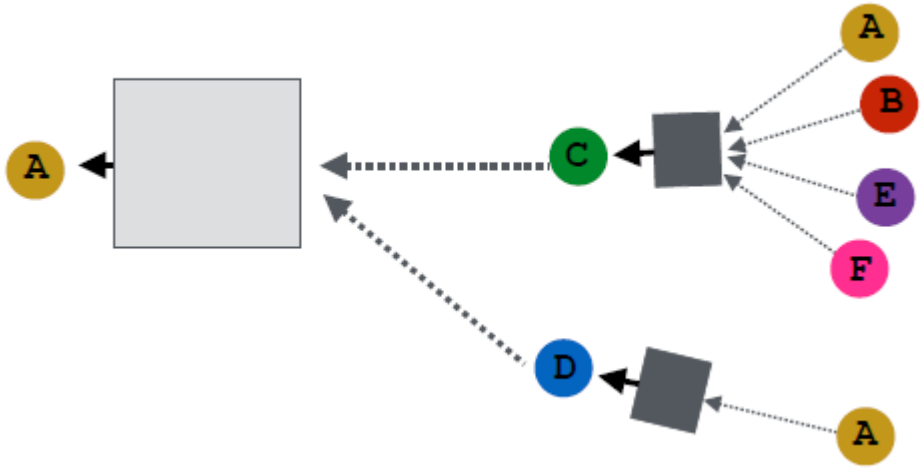


[1] Ying et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems, KDD 2018

6.3 Graph Structure Manipulation

- For dense / large graphs, high-degree nodes
- Solution: not all the neighbors are used for message passing
  - Randomly determine a node's neighborhood for message passing
  - Next time when computing embeddings, we sample different neighbors
  - It greatly reduce computational cost while still maintaining accuracy





References

- **Deep Learning on Graphs:** www.cse.msu.edu/~mayao4/dlg_book/dlg_book.pdf
- **Graph Representation Learning:** https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf
- GitHub curation: thunlp/GNNPapers: Must-read papers on graph neural networks (GNN) (github.com)

GNN Libraries

Library Name	Comment	Supported Framework
PyTorch Geometric (PyG)[1]	De-facto standard framework for GNNs in PyTorch ecosystem. PyG provides automatic batching and established community, with the ease of integration of common benchmark datasets.	Pytorch
Deep Graph Library (DGL)[2]	Focused on high-performance and scalable development, with a useful higher-level abstraction, allowing for auto-batching.	Pytorch, TF, MxNet
TF-GNN[3]	Designed to scale to large graphs, also support distributed training.	TF2
Spektral[4]	TF2 framework but without batching support.	TF2

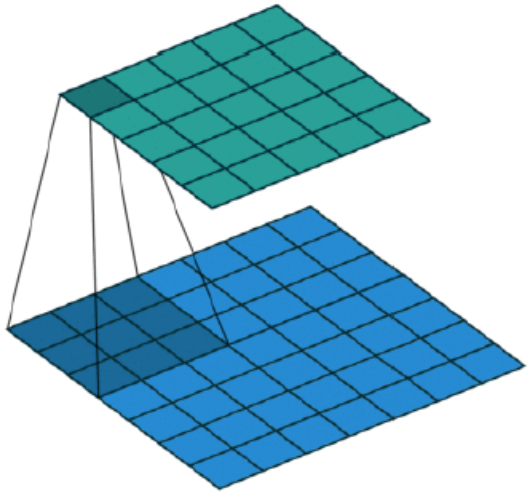
1: https://pytorch-geometric.readthedocs.io/
2: https://www.dgl.ai/
3: https://github.com/tensorflow/gnn
4: https://graphneural.network/

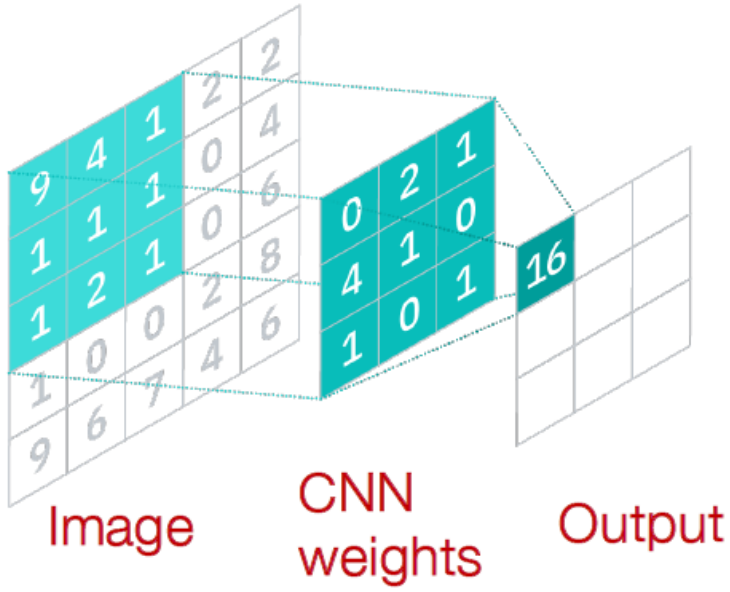
Thank you!

Convolutional Neural Network

Supplementary Material

Convolutional neural network (CNN) layer with 3x3 filter:



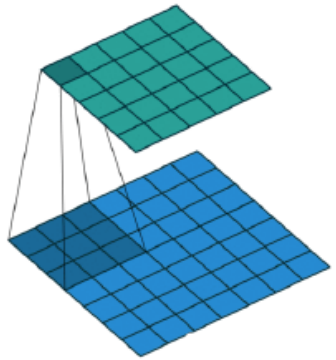


CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v) \cup \{v\}} \mathrm{W}_l^u \mathrm{h}_u^{(l)}), \quad \forall l \in \{0, \ldots, L-1\}$

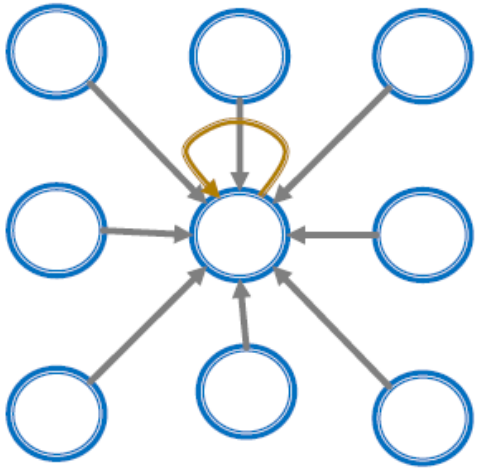
$N(v)$ represents the 8 neighbor pixels of v.

GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:



Image



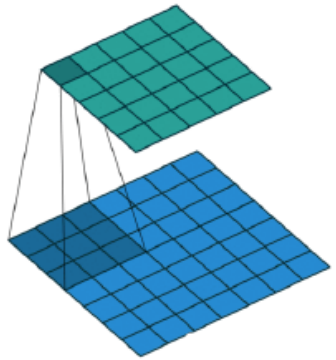
Graph

- GNN formulation: $h_v^{(l+1)} = \sigma(\mathbf{W_l} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \forall l \in \{0, \dots, L-1\}$
- CNN formulation: (previous slide) $h_v^{(l+1)} = \sigma(\sum_{u \in N(v) \cup \{v\}} W_l^u h_u^{(l)}), \forall l \in \{0, \dots, L-1\}$

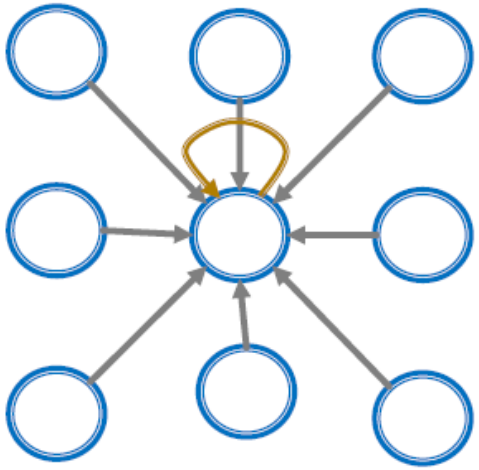
  if we rewrite: $h_v^{(l+1)} = \sigma(\sum_{u \in N(v)} \mathbf{W_l^u} h_u^{(l)} + B_l h_v^{(l)}), \forall l \in \{0, \dots, L-1\}$

GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:



Image



Graph

GNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

Key difference: We can learn different W_l^u for different "neighbor" u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using **relative position** to the center pixel: {(-1,-1). (-1,0), (-1, 1), …, (1, 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
  - We implement resizing and padding to make all input samples unified size
  - The advantage of GNN is it processes arbitrary graphs with different degrees for each node
- CNN is not permutation invariant / equivariant
  - Switching the order of pixels leads to different outputs