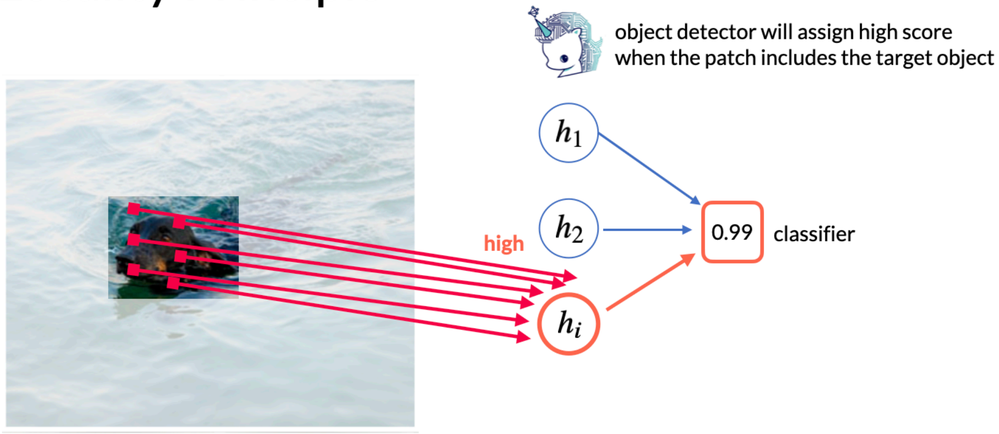
Previously, we learned…

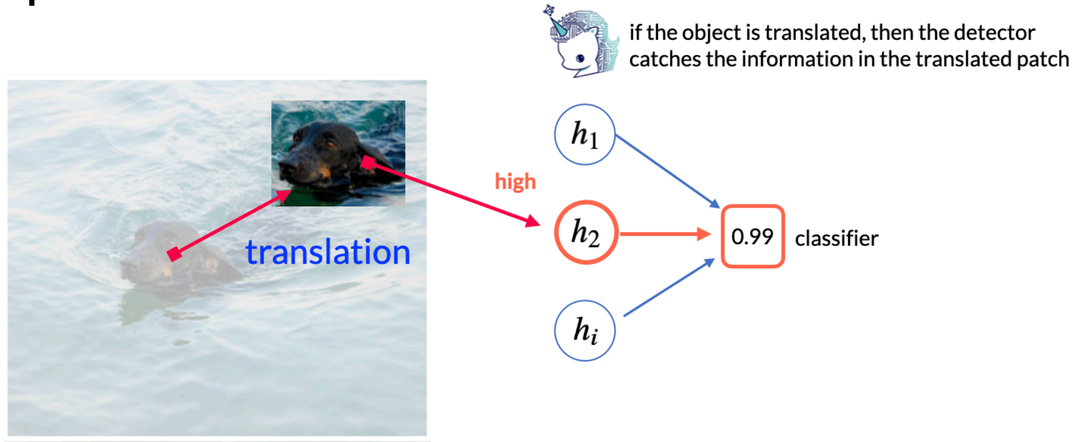
- Two design principle of convolutional networks for visual domain
  - Locality Principle
  - Spatial Invariance

note that these principles can be violated through MLP layer

Locality Principle



Spatial Invariance



Graph Neural Network

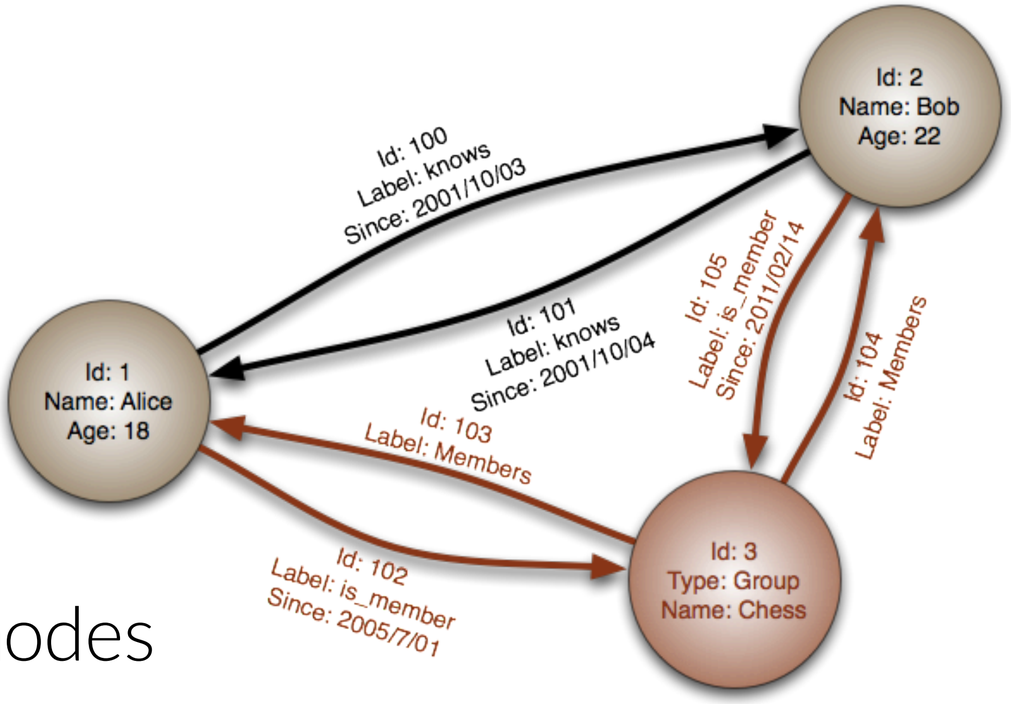
Principles of Deep Learning (AI502/IE408/IE511)

Sungbin Lim (UNIST AIGS & IE)

Contact: ai502deeplearning@gmail.com

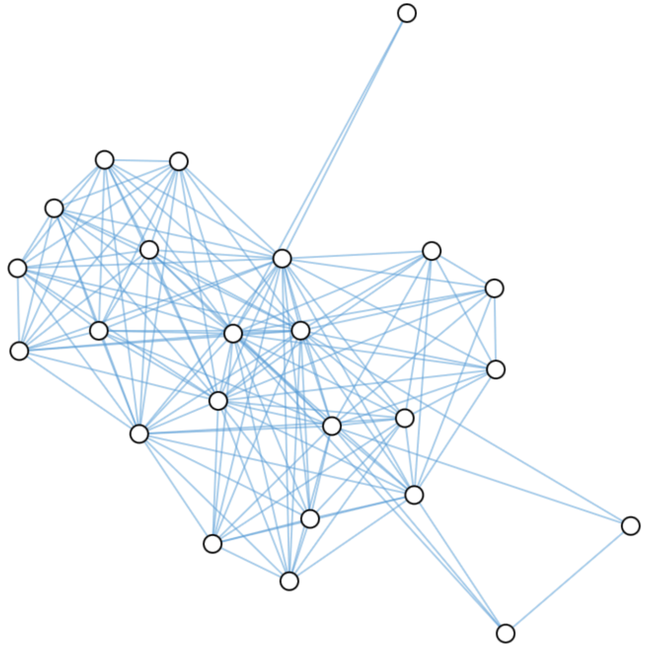
What is Graph Data?

- Graph data is based on graph theory
  - set of objects (node or edge)
  - nodes represent entities or instances ex. people, accounts, item
  - edges connect nodes to other nodes
- Properties are information associated to nodes
- Applications
  - social networks, recommendation system, scientific simulations



Example: social networks

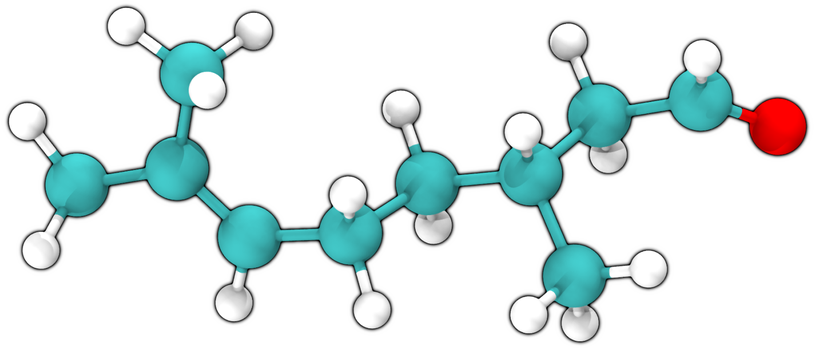
Social Networks



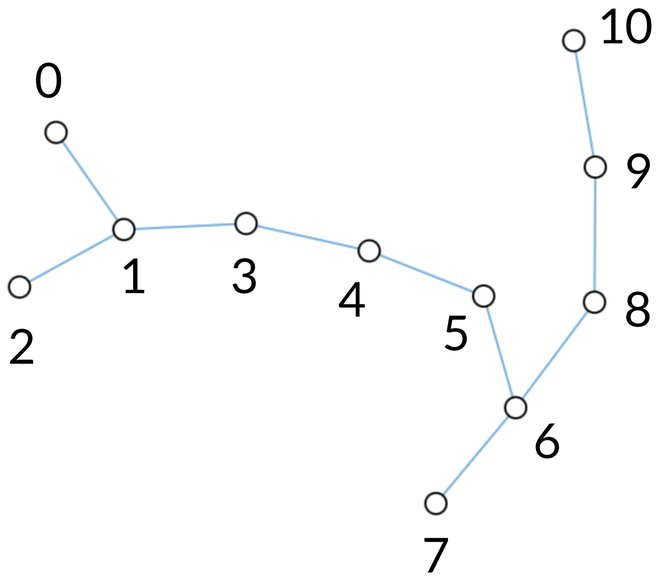
Graph representation

Adjacency matrix

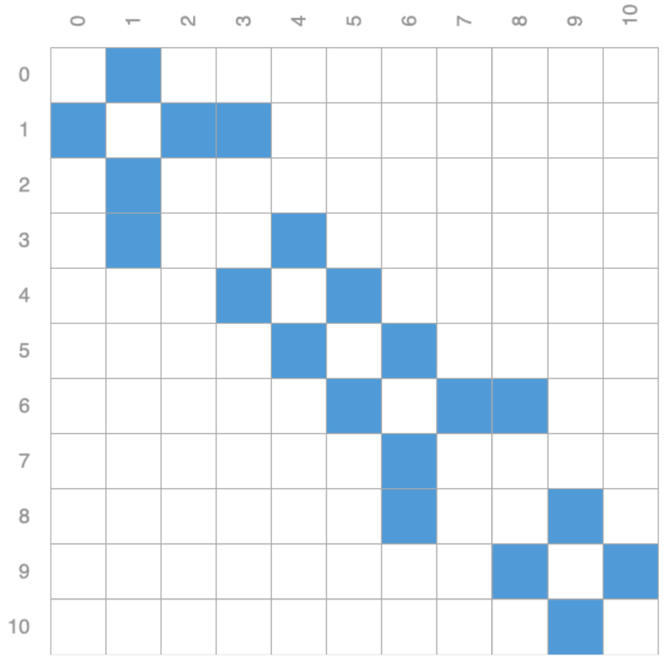
Example: molecules



Citronellal molecule

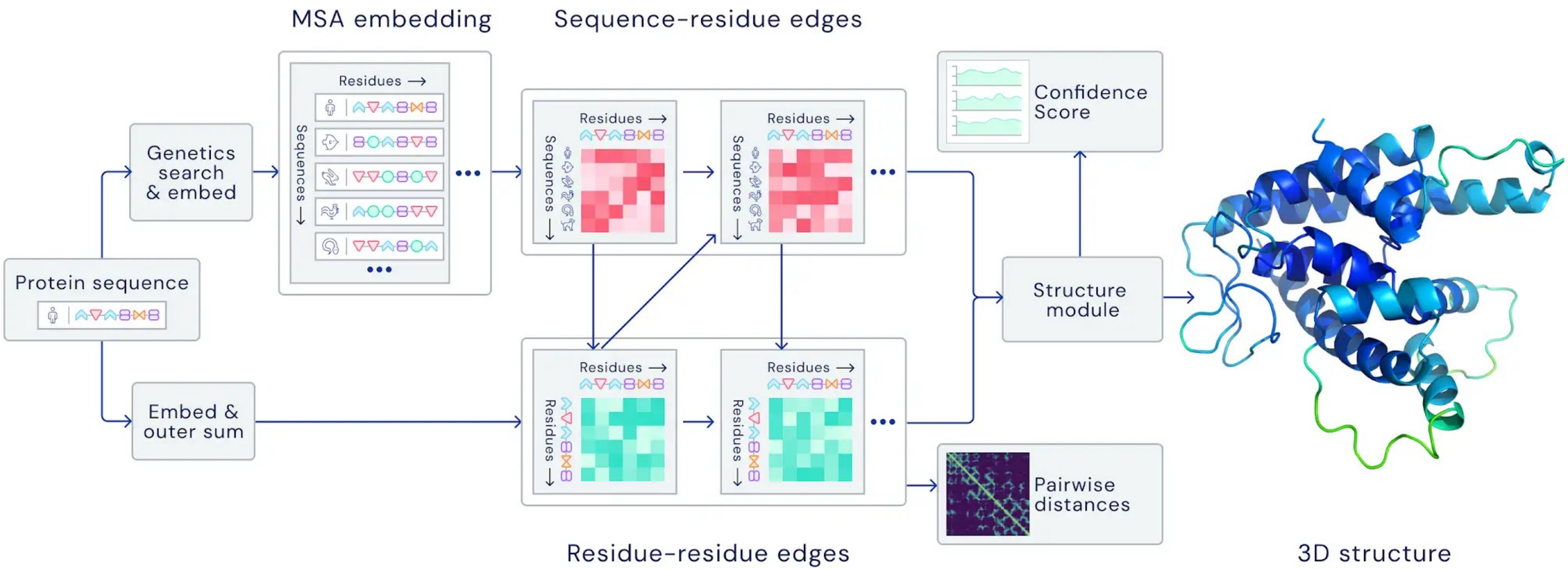


Graph representation



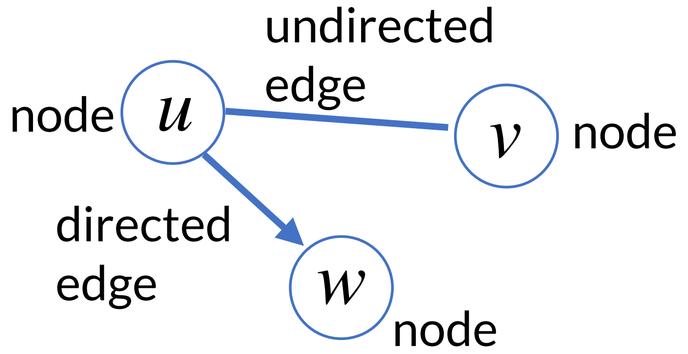
Adjacency matrix

Example: AlphaFold



High Accuracy Protein Structure Prediction Using Deep Learning, Jumper et al., 2020

Graph Theory in a Nutshell



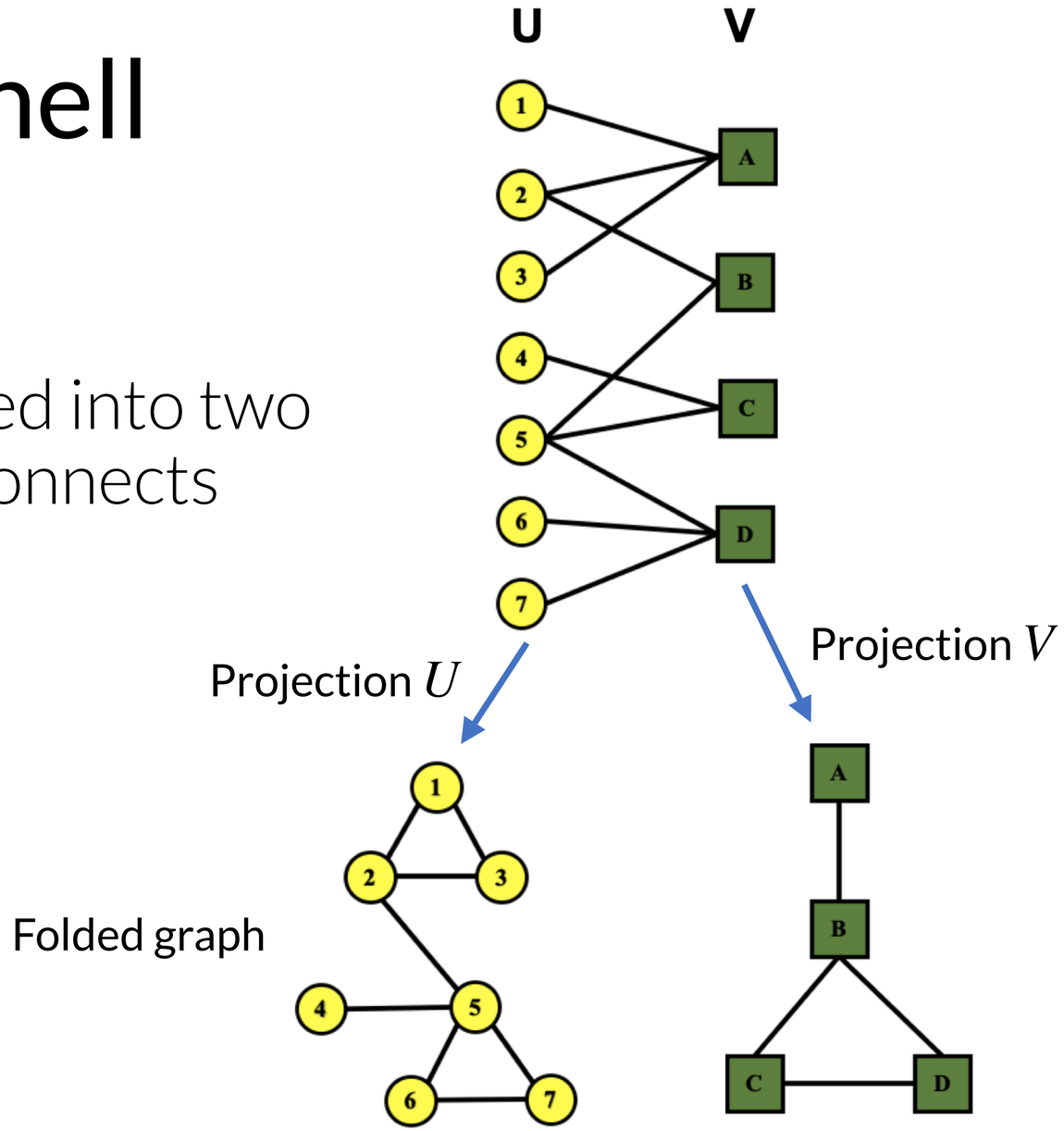
- A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a structure to represent entities and their relations
  - $\mathcal{V}$ the set of nodes (or vertices)
  - $\mathcal{E}$ the set of edges (or arcs)

$$\mathcal{V} = \{u, v, w\}$$
$$\mathcal{E} = \{(u, w), (u, v), (v, u)\}$$

- $(u, v) \in \mathcal{E}$ an edge connecting a pair of nodes indicates there is a relation
  - undirected: capturing symmetric relations (friendship in facebook)
  - directed: capturing asymmetric relations (following in instagram)
- Edges of graph can be weighted
  - (ex) distance, connectivity strength

Graph Theory in a Nutshell

- Bipartite graph
  - a graph whose nodes can be divided into two disjoint sets such that every link connects a node in U to one in V
  - U and V are independent sets
- Examples
  - authors - papers
  - actors - movies
  - recipes - ingredients

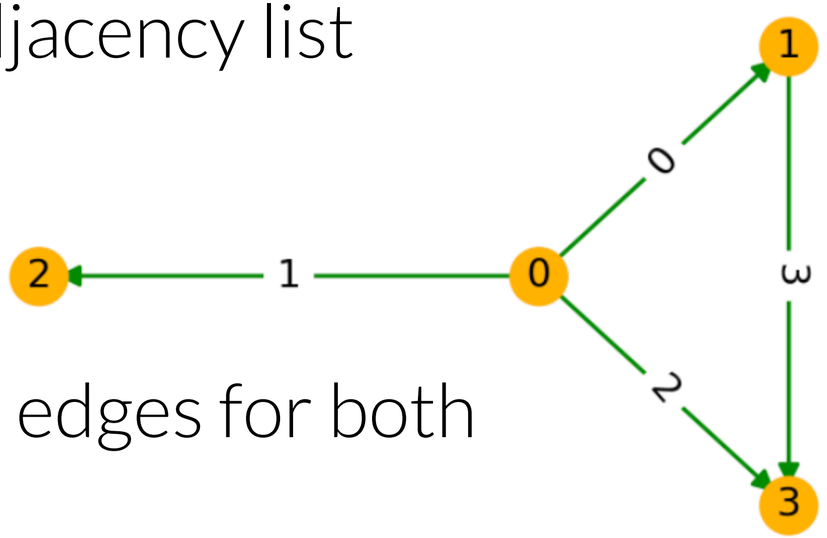


How to represent graph connectivity?

- Adjacency matrix representation has some drawbacks
  - space-inefficient
  - permutation invariant
- We expect the number of edges to be much lower than the number of entries for an adjacency matrix
- Hence we use **adjacency lists** to represent connectivity
  - describe the connectivity of edge $e_k \in \mathcal{E}$ between two nodes as a tuple in the k-th entry of an adjacency list

Graph in DGL

- DGL uses 1D integer tensor of node IDs to represent node-tensors
  - it uses a tuple of node-tensors `(u, v)` where `(u[i], v[i])` decides an edge from `u[i]` to `v[i]` → adjacency list
-
```
u = torch.tensor([0, 0, 0, 1])
v = torch.tensor([1, 2, 3, 3])
g = dgl.graph([u, v])
```
- For an undirected graph, one needs to create edges for both directions.
  - use `dgl.to_bidirected()` to convert a graph into undirected one



Node and Edge Features

- The nodes and edges can have user-defined features for storing specific properties of the nodes and edges
  - in DGL DEEP GRAPH LIBRARY, these features can be accessed via **`ndata`** and **`edata`**
  - **`g.ndata['feature_name'], g.edata['feature_name']`**
- We use the following notation: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

$$\{\mathbf{x}_v \in \mathbb{R}^d : v \in \mathcal{V}\}$$

node feature vector

$$\mathbf{X} = [\mathbf{x}_v \in \mathbb{R}^d : v \in \mathcal{V}] \in \mathbb{R}^{d \times |\mathcal{V}|}$$

matrix of node features

$$\{\mathbf{w}_{uv} \in \mathbb{R}^d : (u, v) \in \mathcal{E}\}$$

edge feature vector

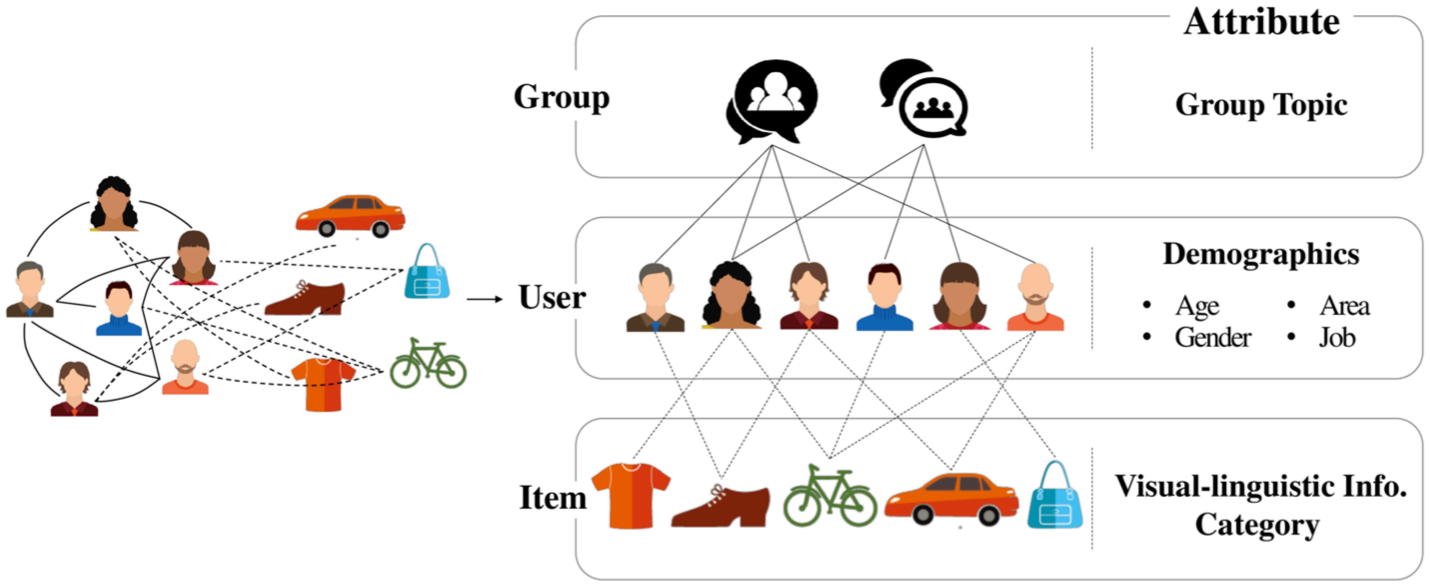
$$\mathbf{W} = [\mathbf{w}_{uv} \in \mathbb{R}^d : (u, v) \in \mathcal{E}] \in \mathbb{R}^{d \times |\mathcal{E}|}$$

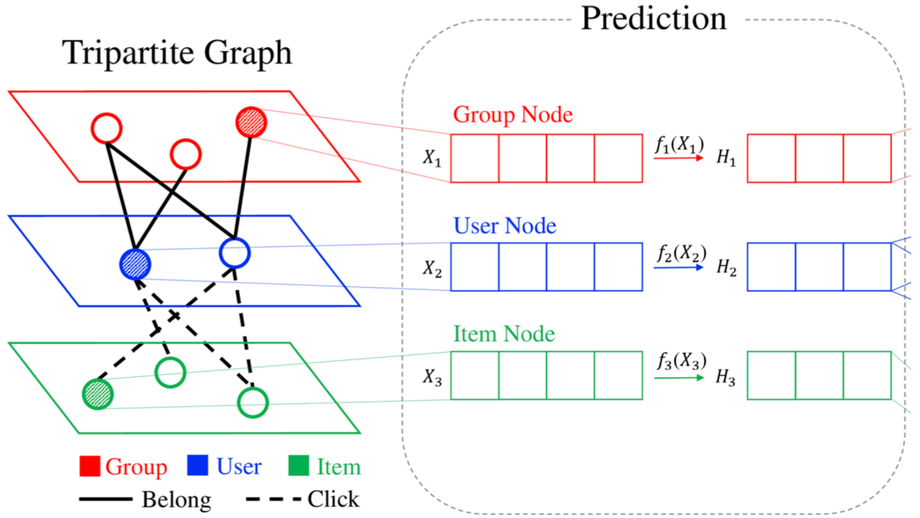
matrix of edge features

Heterogeneous Graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T})$

relation type node type

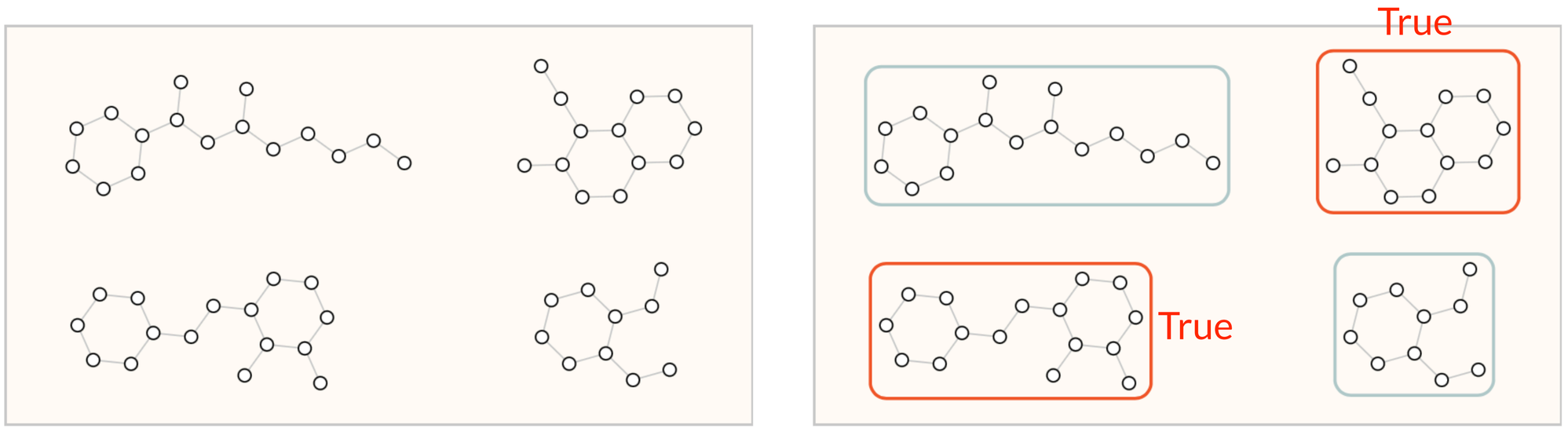
- Some graphs can have nodes and edges of different types
  - e.g. user & group, human & object, action & property
- Different node/edge types have different ID space and feature storage





Tasks in Graph Structured Data

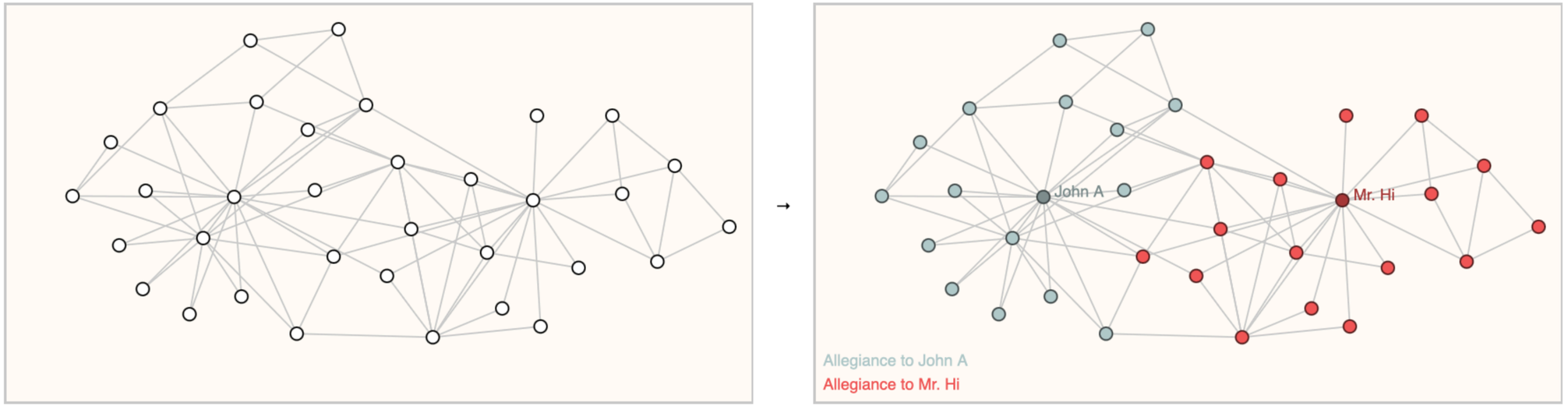
- Graph-level task



Question: Does the graph contain two rings?

Tasks in Graph Structured Data

- Node-level task

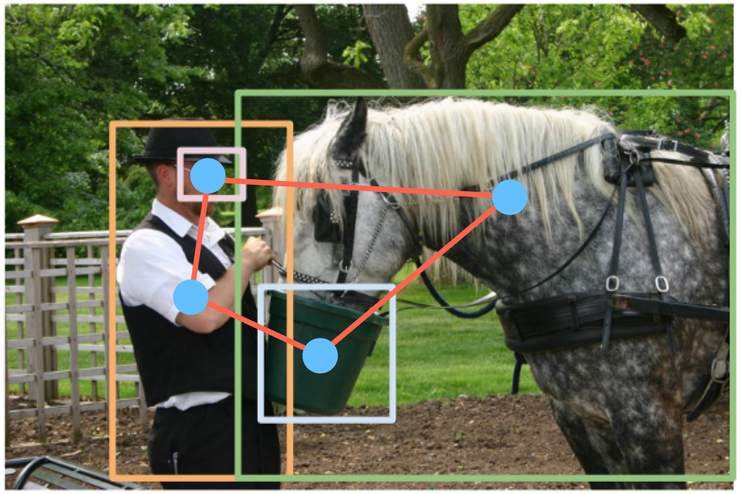


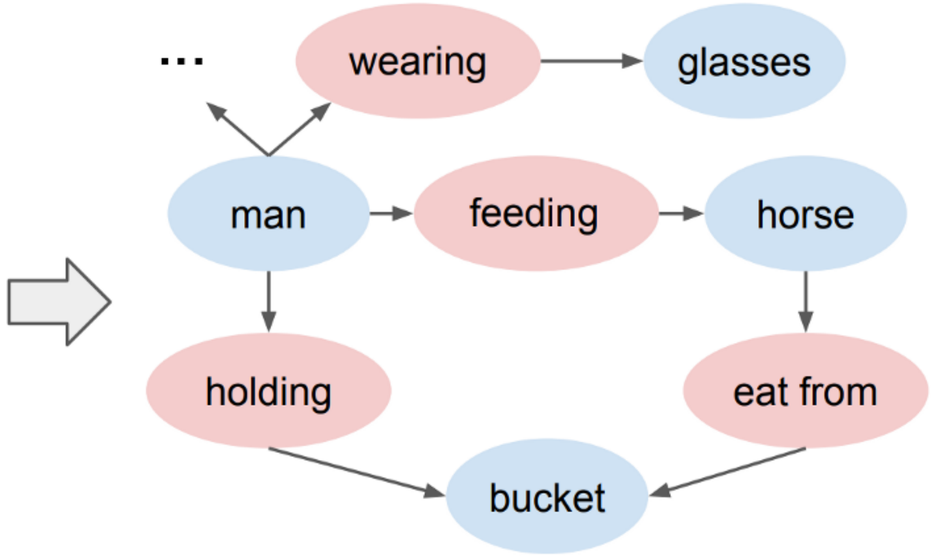
Input Output

Question: What is the label of each node?

Tasks in Graph Structured Data

- Edge-level task





Input

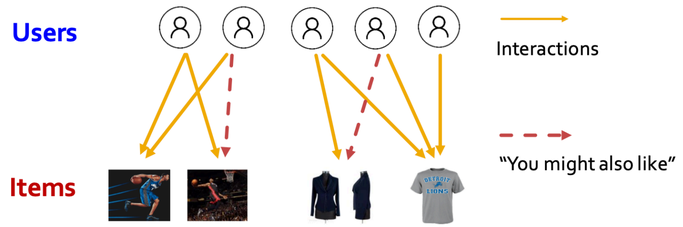
Output

Question: What are the relationships between entities?

Applications of each task

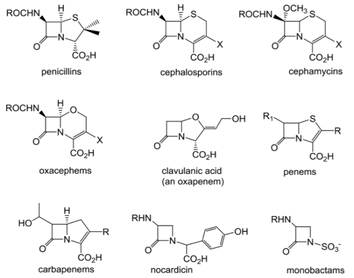
- There are many applications using graph-structured data
- We need to choose proper graph representation

- **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**



Recommender Systems
(Edge-Level Task)

- **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.

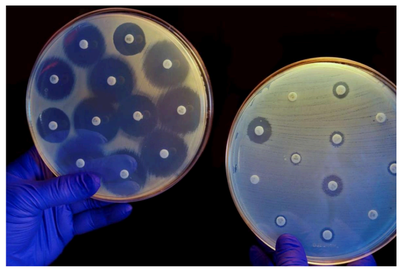
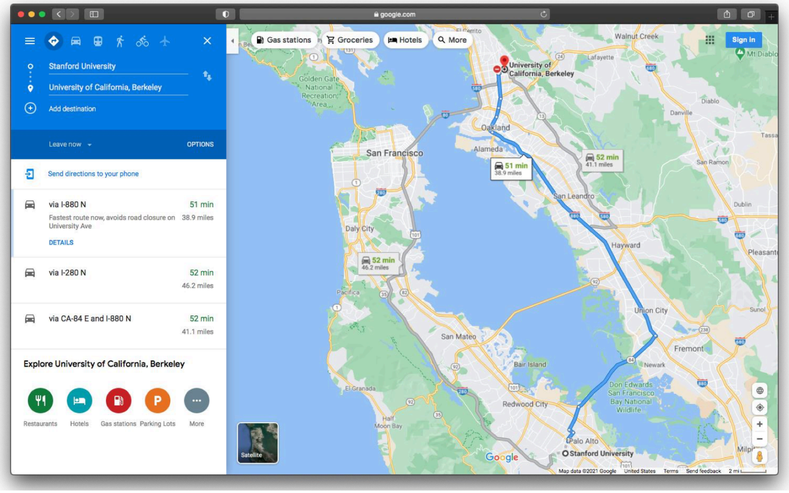


Image credit: CNN

Drug Discovery
(Graph-Level Task)



Traffic Prediction
(Subgraph-Level Task)

Inductiva Biases in Graph

- Design principles in graph data
  - Relationships between Entities
  - Permutation **Invariance**

$$\phi(\pi \mathbf{x}) = \phi(\mathbf{x})$$

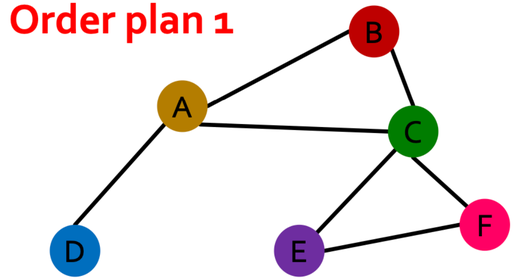
permutation operation

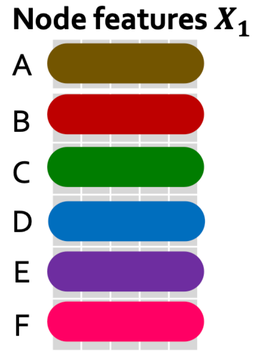
  - Permutation **Equivariance**

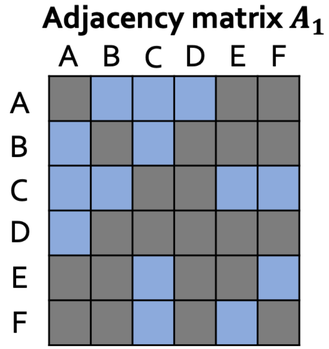
$$\phi(\pi \mathbf{x}) = \pi \phi(\mathbf{x})$$

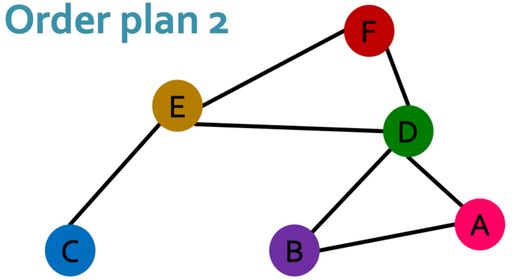
Graph does not have a canonical order of the nodes

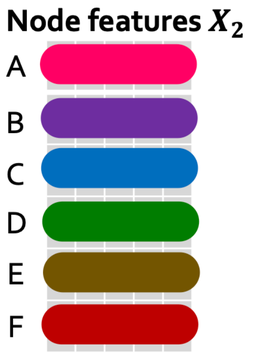
$$\phi(\mathcal{G}_1) = \phi(\mathcal{G}_2) \text{ if } \mathcal{G}_1 = \pi \mathcal{G}_2$$

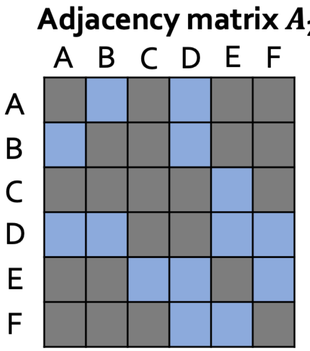




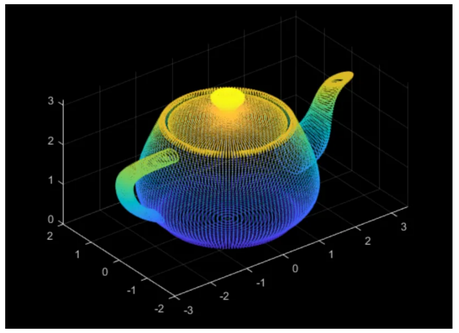




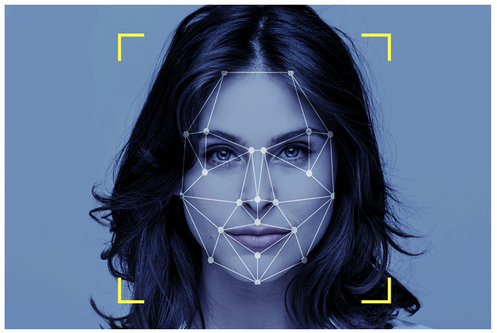




Inductiva Biases in Graph

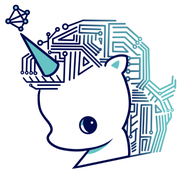


Point Cloud



Mesh Data

- Design principles in graph data
  - Relationships between Entities
  - Permutation **Invariance** $\phi(\pi\mathbf{x}) = \phi(\mathbf{x})$
  - Permutation **Equivariance** $\phi(\pi\mathbf{x}) = \pi\phi(\mathbf{x})$
- Inductive biases in graph are applied to designing transformation on sets
  - the order of operation on nodes or edges should not matter
  - the operation should work on a variable number of inputs

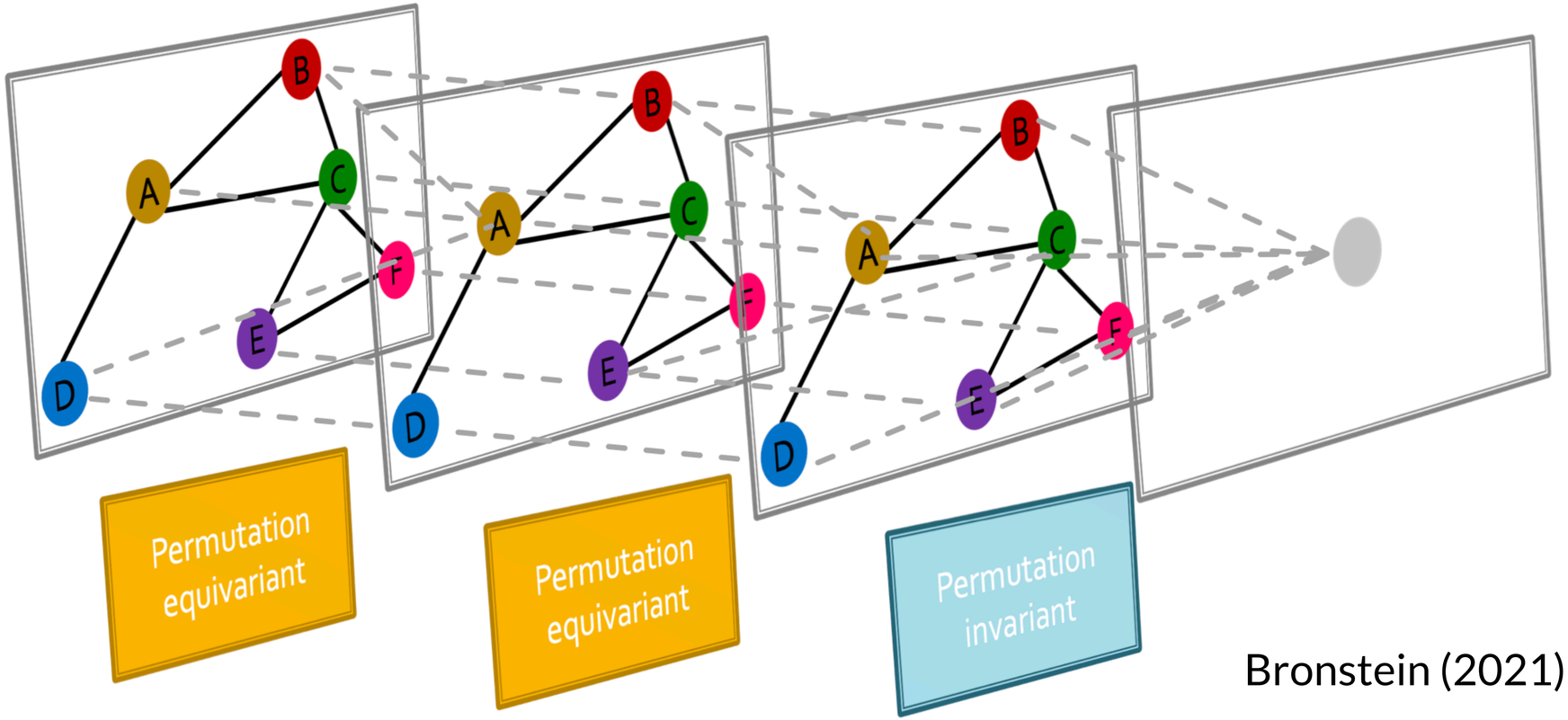


Note that each graph data has a different number of nodes

Design Principles for GNN

Remember that we applied the similar principles in CNN regarding ***translation***

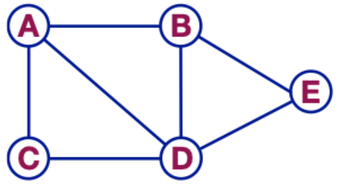
- GNNs consist of multiple permutation equivariant & invariant functions!



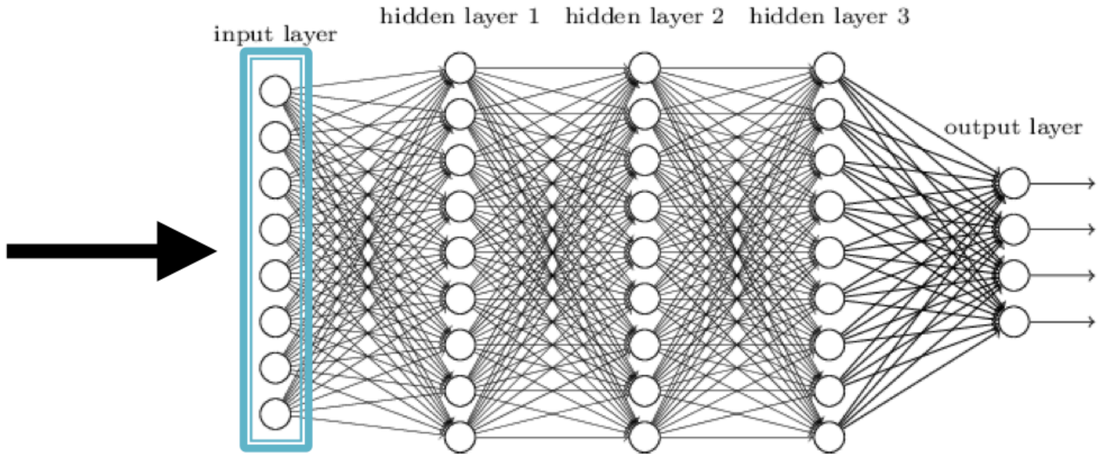
Bronstein (2021)

Can we use MLP?

- **No**
  - switching the order of the input leads to different outputs
  - not applicable to graphs of different sizes
  - we need $\mathcal{O}(|\mathcal{V}|)$ parameters



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



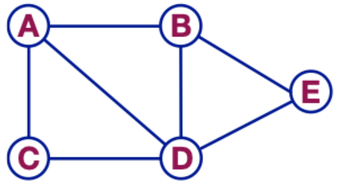
What if we use *pooling* after the input layer?

Average Pooling

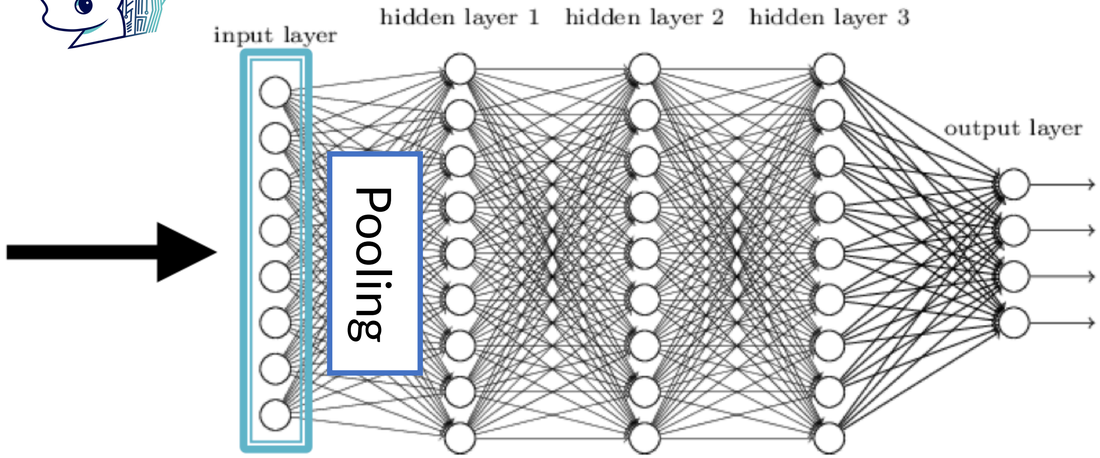
Max Pooling

- **No?**
  - ~~switching the order of the input leads to different outputs~~
  - ~~not applicable to graphs of different sizes~~
  - ~~we need $\mathcal{O}(|\mathcal{V}|)$ parameters~~

However, we loose feature information too much!



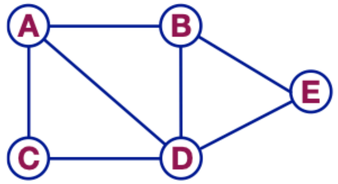
	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



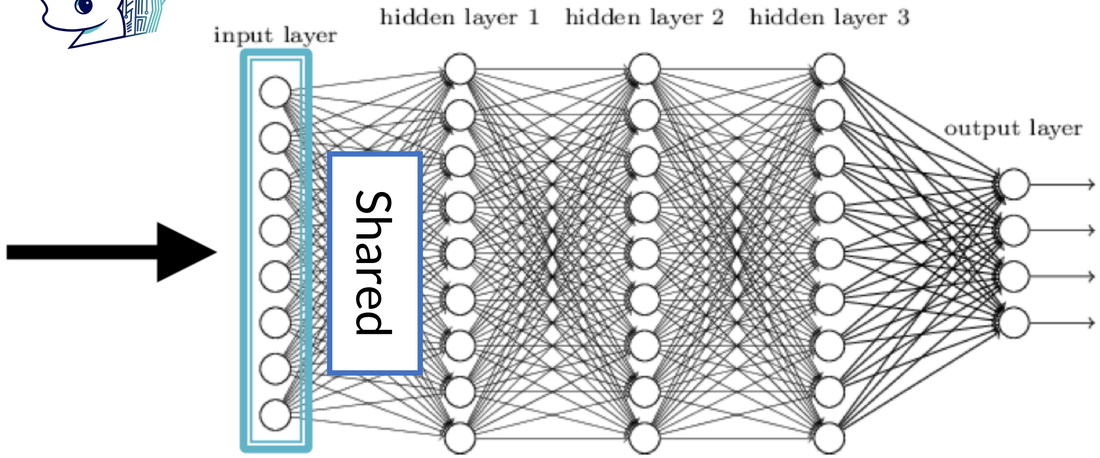
What if we share *parameters* across variables?

- **No?**
  - ~~switching the order of the input leads to different outputs~~
  - ~~not applicable to graphs of different sizes~~
  - ~~we need $\mathcal{O}(|\mathcal{V}|)$ parameters~~

This is same as ***summation*** pooling!

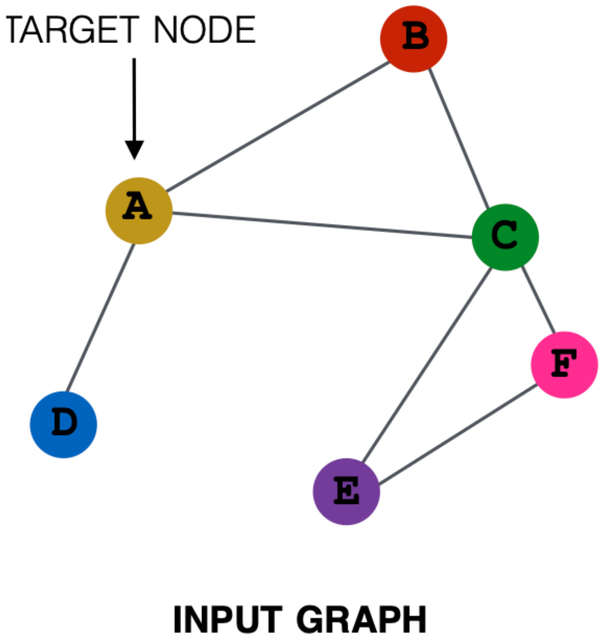


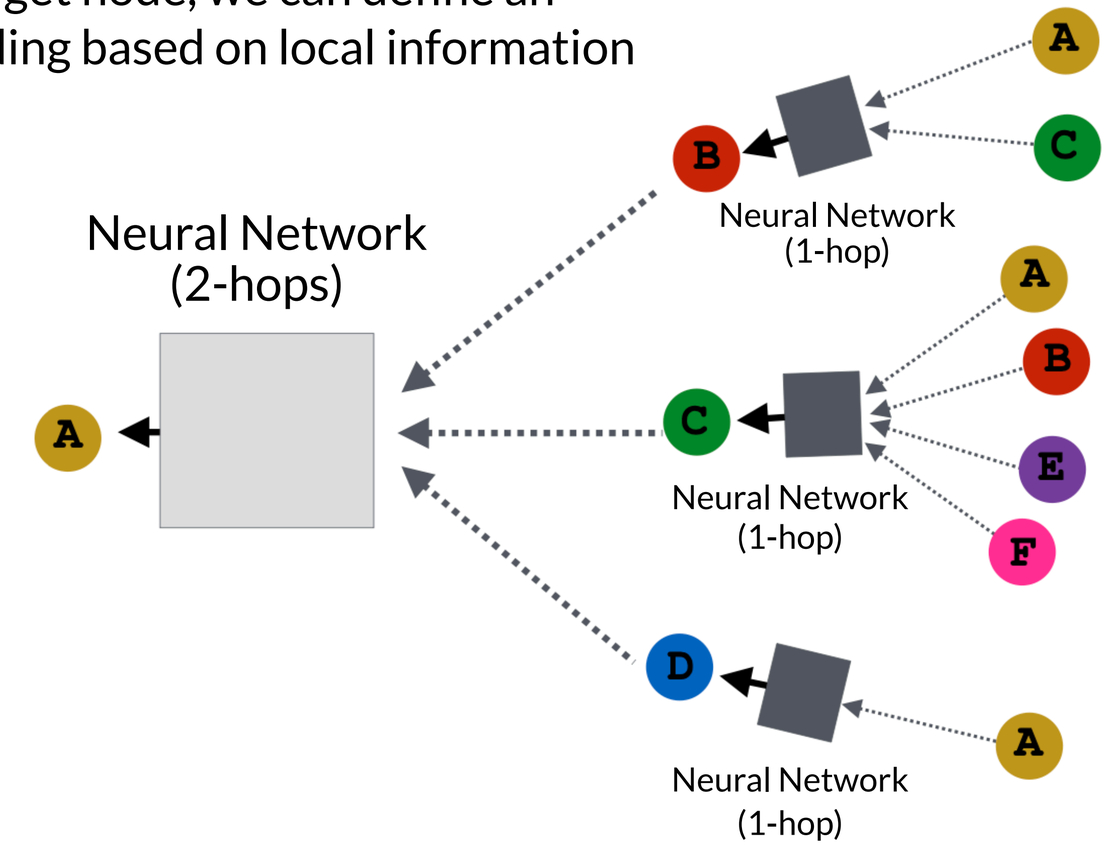
	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



Idea: aggregate features from neighbors

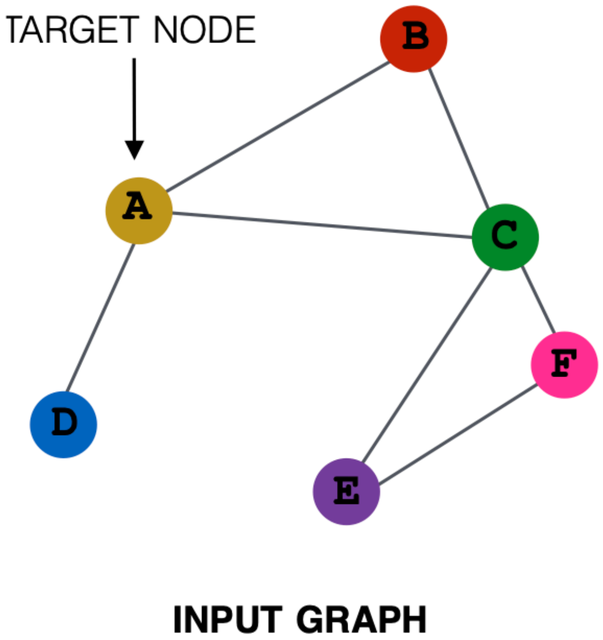
for a target node, we can define an embedding based on local information

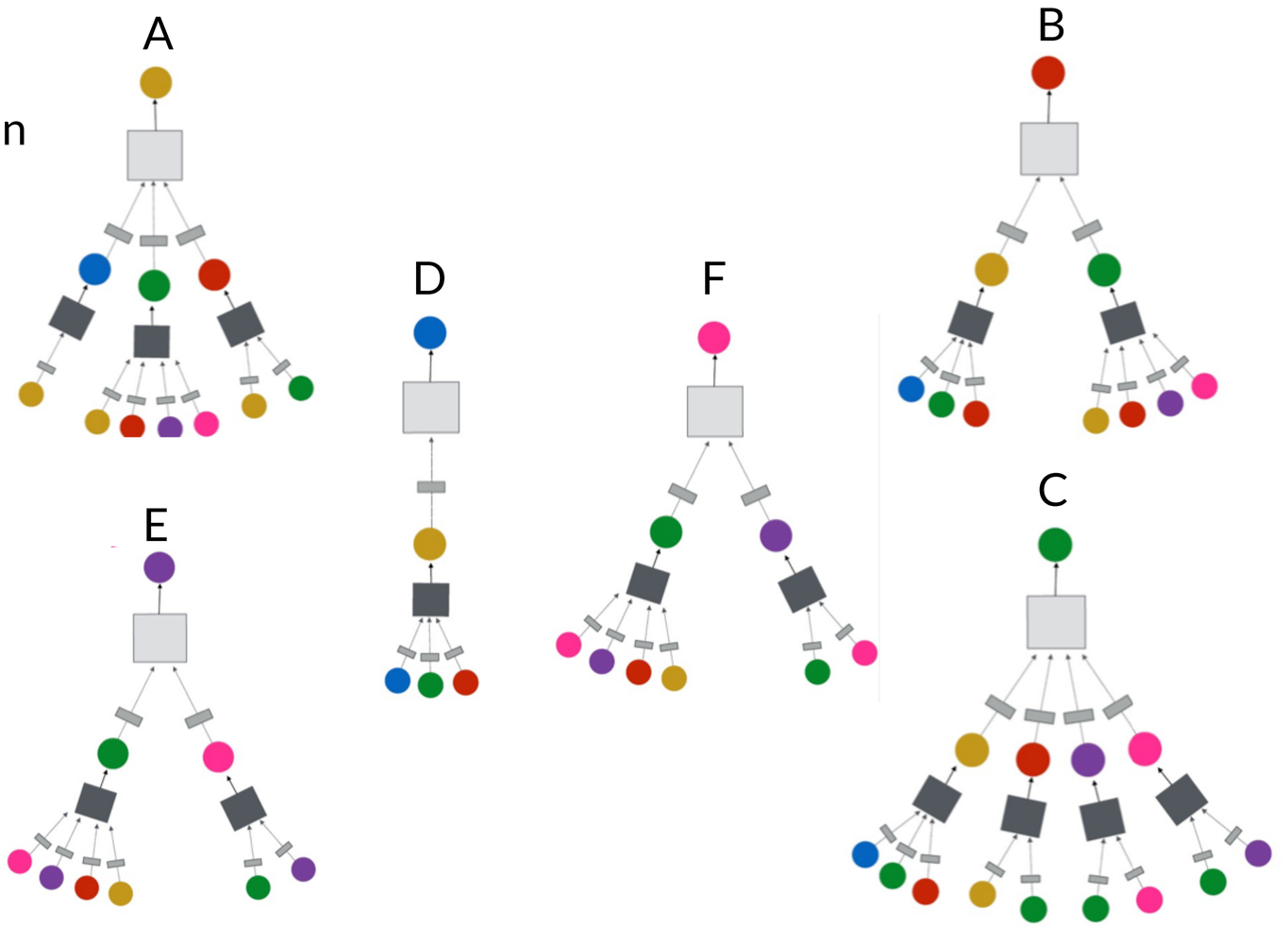




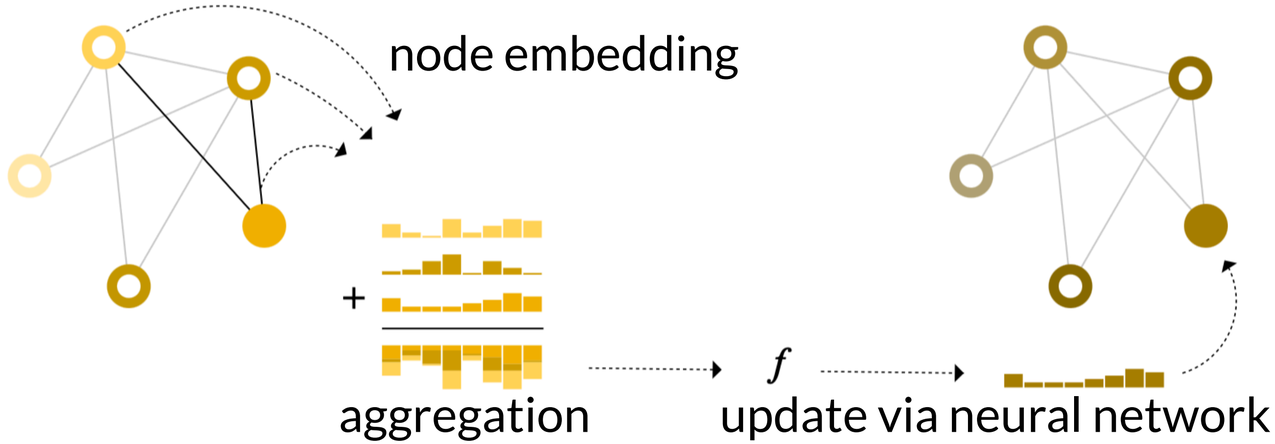
Idea: aggregate features from neighbors

Every node defines an information flow based on its neighborhood





Message Passing

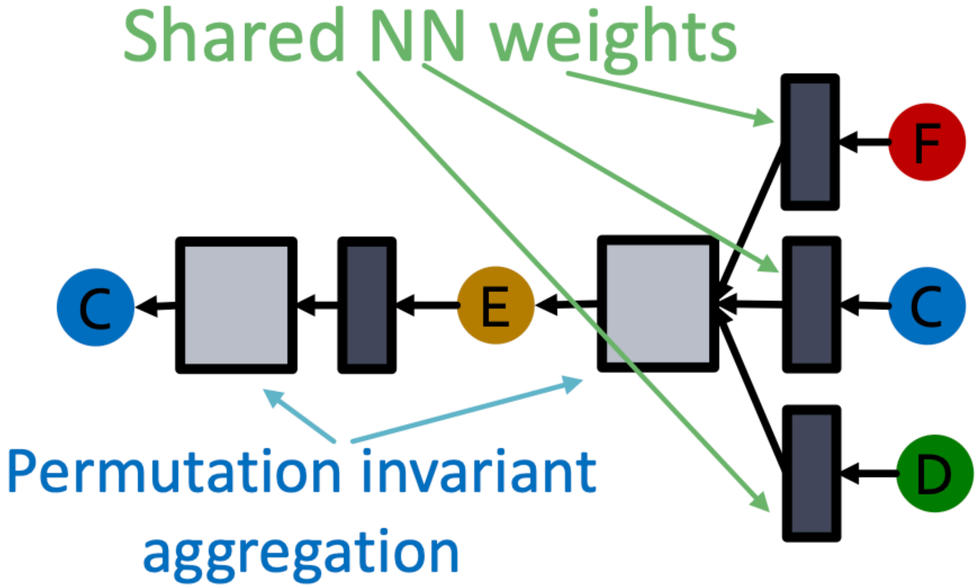


- **Yes!**
  - aggregate feature information from ***adjacent edges***
  - predict graph context and label using aggregated information
- Message Passing in three steps
  - for each node in the graph, gather all the neighboring ***node embeddings*** (message)
  - aggregate all messages via an ***aggregating*** function
  - aggregated messages are ***combined*** through a ***learnable neural network***

Aggregate and Combine are key operations in message passing GNN

Dive into Message Passing

- Graph Convolutional Networks (GCN)



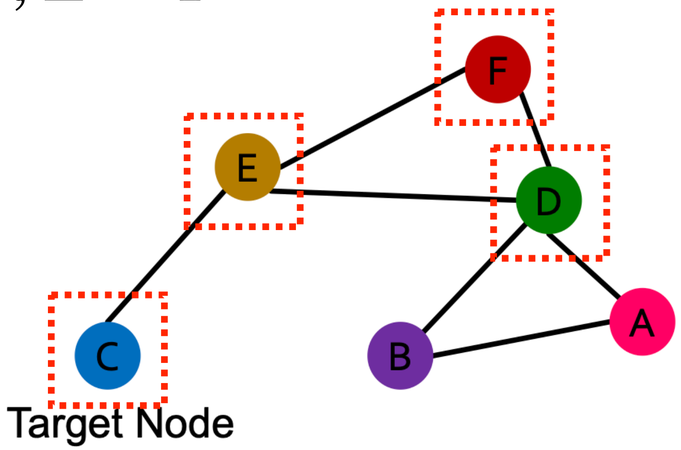
$$\mathbf{h}_v^{(0)} = \mathbf{x}_v$$

$$\mathbf{h}_v^{(k+1)} = \mathrm{ReLU}\left(\mathbf{W}_k \underbrace{\sum_{(u,v)\in\mathcal{E}} \frac{\mathbf{h}_u^{(k)}}{\#\text{ of neighbors}}}_{\text{mean aggregation}} + \mathbf{B}_k \mathbf{h}_v^{(k)}\right), \quad k = 0, \ldots, L-1$$

$$\mathbf{z}_v = \mathbf{h}_v^{(L)}$$

Shared learnable parameters

Note that same aggregation parameters are *shared for all* nodes



Dive into Message Passing

- Graph Convolutional Networks (GCN)

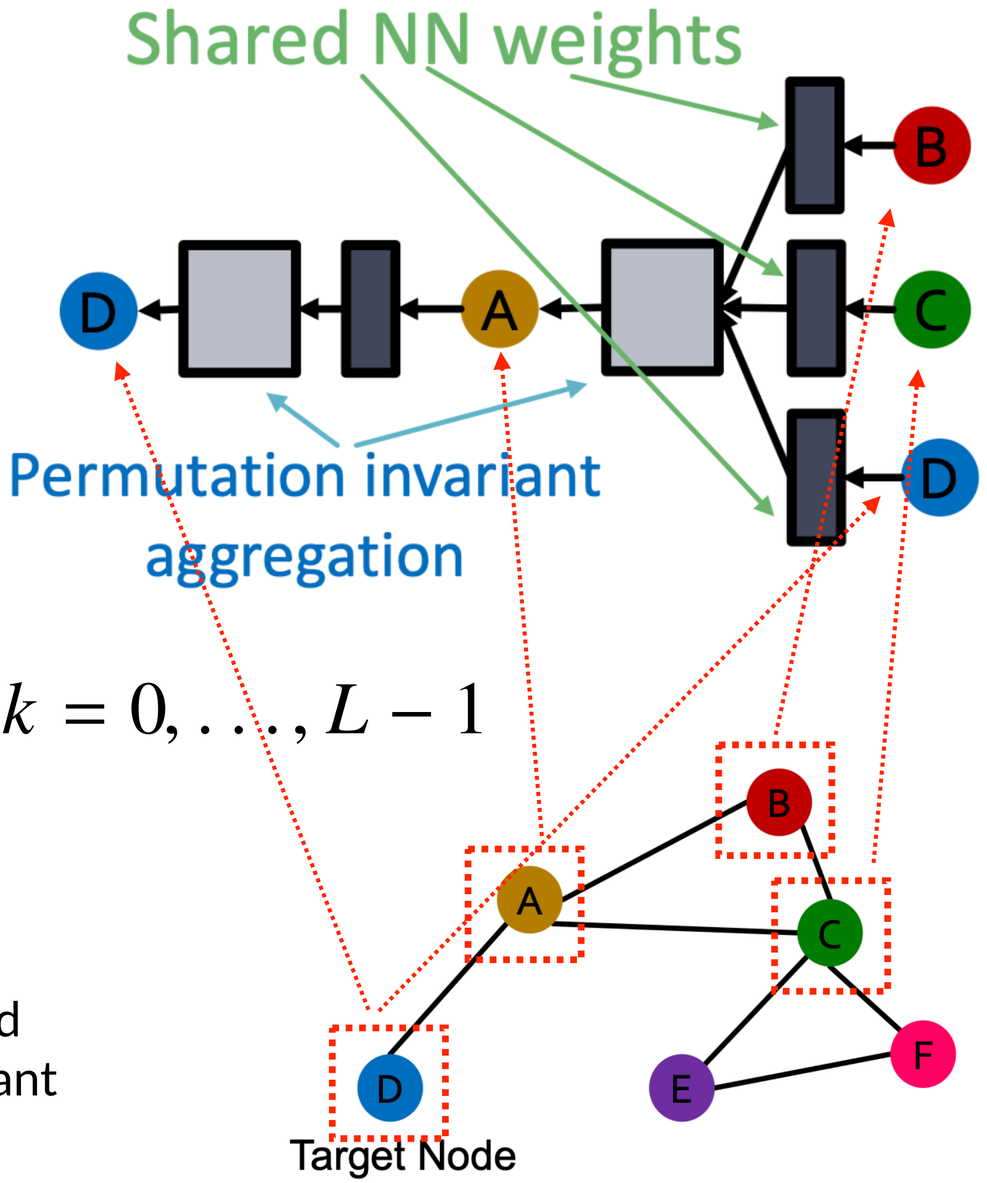
$$\mathbf{h}_v^{(0)} = \mathbf{x}_v$$

$$\mathbf{h}_v^{(k+1)} = \mathrm{ReLU}\left(\mathbf{W}_k \underbrace{\sum_{(u,v)\in\mathcal{E}} \frac{\mathbf{h}_u^{(k)}}{\#\text{ of neighbors}}}_{\text{mean aggregation}} + \mathbf{B}_k \mathbf{h}_v^{(k)}\right), \quad k = 0, \ldots, L-1$$

$$\mathbf{z}_v = \mathbf{h}_v^{(L)}$$

Shared learnable parameters

Hence the message passing and neighborhood aggregation in GCNs is permutation equivariant



Why is it called Graph Convolution? 🤔

- Degree matrix $D_{ii} = \sum_j A_{ij}$ — adjacency matrix

$$\sum_{(u,v)\in\mathcal{E}} \frac{\mathbf{h}_u^{(k)}}{\#\text{ of neighbors}} \rightarrow H^{k+1} = D^{-1}AH^{(k)}$$

- Unnormalized Graph Laplacian $L := D - A$
  - normalized Graph Laplacian $D^{-1/2}LD^{-1/2} := I - D^{-1/2}AD^{-1/2}$
- Graph (inverse) Fourier Transform $\mathcal{F}\{f\} = \hat{f} = U^\top f, \quad \mathcal{F}^{-1}\{g\} = Ug$
- Spectral Graph Convolution

$$k * f := \mathcal{F}^{-1}\{\hat{k} \odot \hat{f}\} = U[U^\top k] \odot [U^\top f] = U\,\mathrm{diag}(\hat{k})U^\top f$$

$$= P_\theta(L, n)f$$

message passing via
n-hop local neighbors

eigenvector

eigenvalue

$P_\theta(\Lambda, n)$
n-th polynomial

Better *message passing*?

- We have multiple design choices in GNN

$$\mathbf{h}_v^{(k+1)} = \text{COMBINE}\left(\underset{(u,v)\in\mathcal{E}}{\text{AGGREGATE}}(\mathbf{h}_u^{(k)}), \mathbf{h}_v^{(k)}\right) \quad k = 0, \ldots, L-1$$

- `AGGREGATE`
  - summation, averaging, **max**, min, …
- `COMBINE`
  - summation, **concatenation**, …

+ L_2-normalization + batch sampling

this is called **GraphSAGE**

Better *message passing*?

- We have multiple design choices in GNN

$$\mathbf{h}_v^{(k+1)} = \text{COMBINE}\left(\underset{(u,v)\in\mathcal{E}}{\text{AGGREGATE}}(\mathbf{h}_u^{(k)}), \mathbf{h}_v^{(k)}\right) \quad k = 0, \ldots, L-1$$

- `AGGREGATE`
  - summation, averaging, max, min, …
- `COMBINE`
  - summation, concatenation, ***learnable neural network***, …

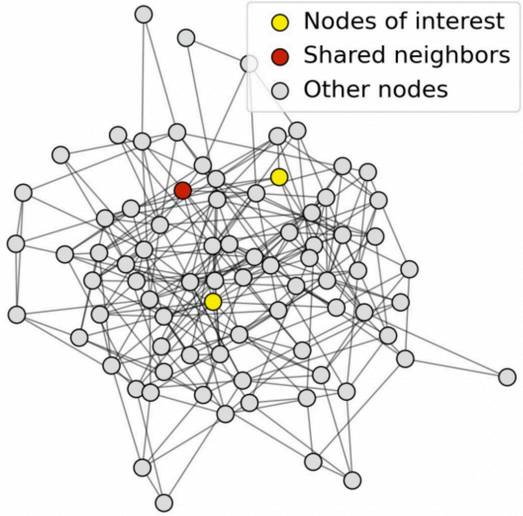
We will return to this topic after we learn ***attention*** mechanism!

Oversmoothing Problem

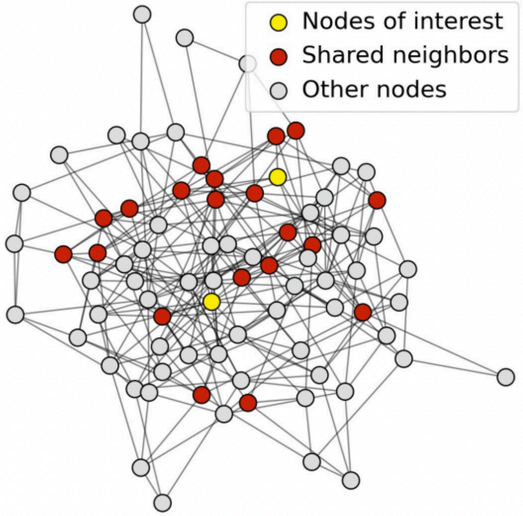
Be cautious when adding GNN layers

- Node embeddings converge to the same value as we stack many layers
  - the shared neighbors quickly grows when we increase the layers

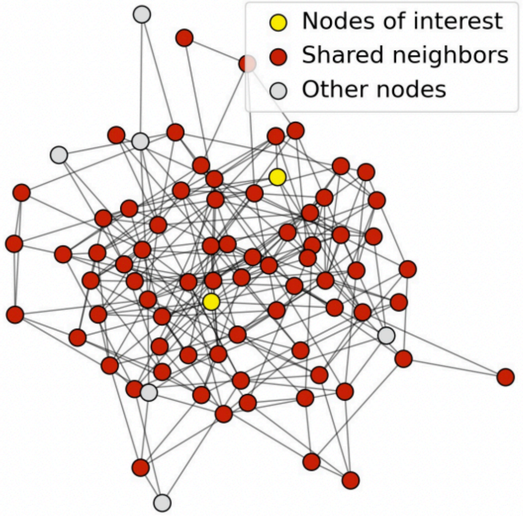
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



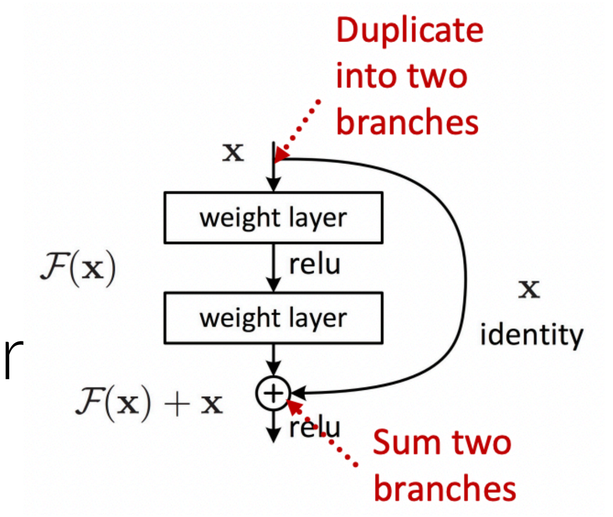
3-hop neighbor overlap
Almost all the nodes!

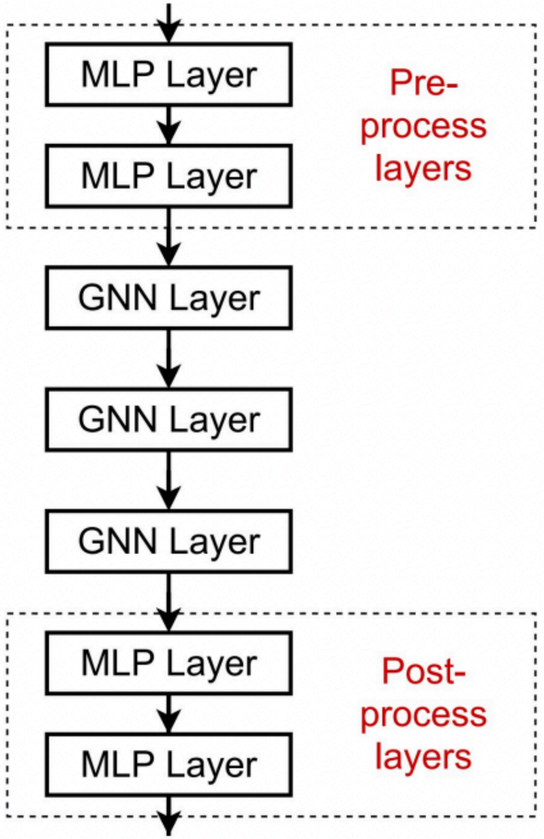


How to solve?

A GNN does not necessarily *only* contain GNN layers

- Make a shallow GNN layer more expressive
  - add MLP layers that do not pass messages
    - pre-processing layers
    - post-processing layers
  - add skip connections
    - idea: residual network
    - increase the impact of earlier layers on the final layer





Q&A