Machine Learning with Graphs

Simona Juvină

National University of Science and Technology POLITEHNICA Bucharest, Romania
BIOSINF Master Program

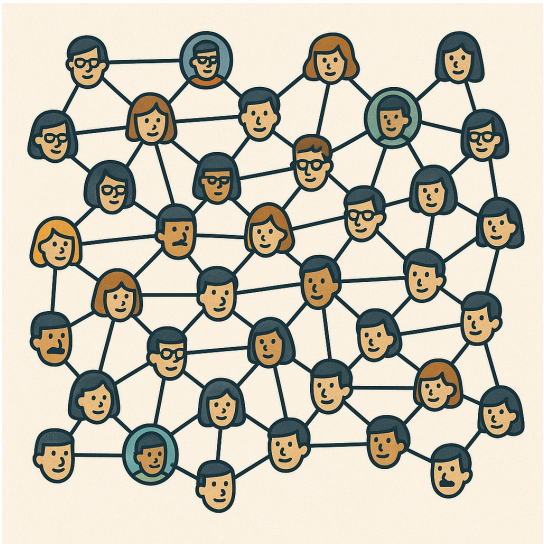
May 2025

Why Graphs?

- Graphs are a universal language for understanding relationships and interactions.
- What data can we model as graphs?

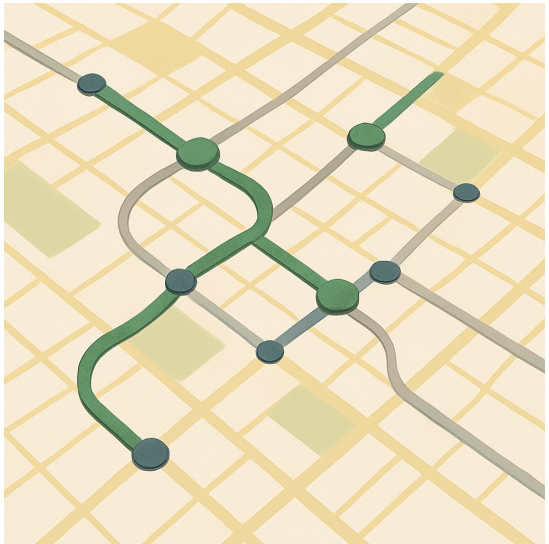
Why Graphs?

- Graphs are a universal language for understanding relationships and interactions.
- What data can we model as graphs?



Social Networks

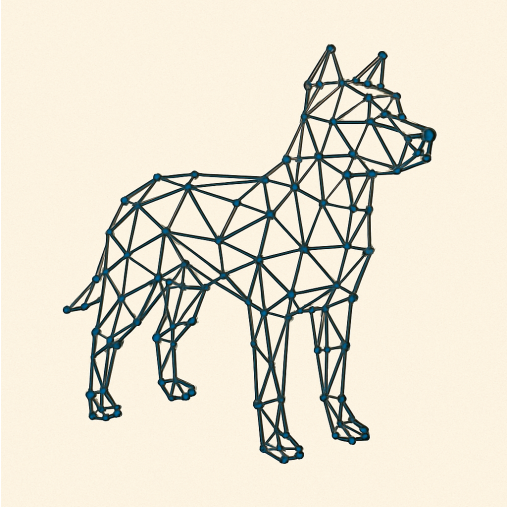
Computer Networks



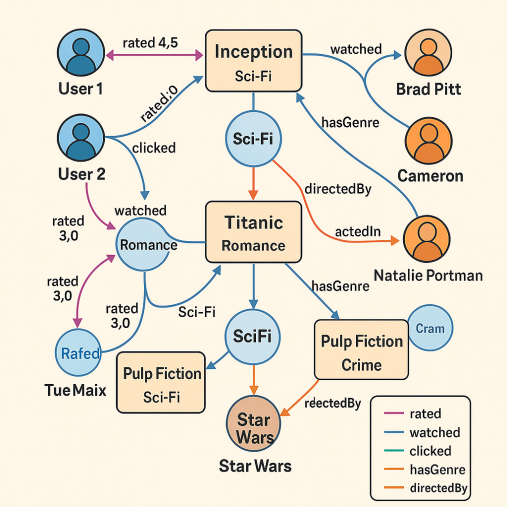
Road Networks

Why Graphs?

Citation Networks

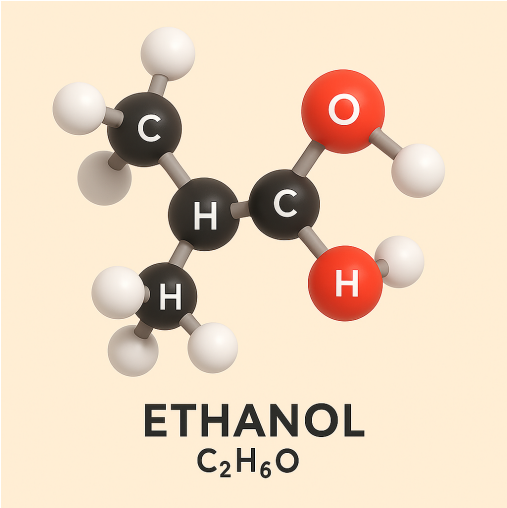


3D Meshes

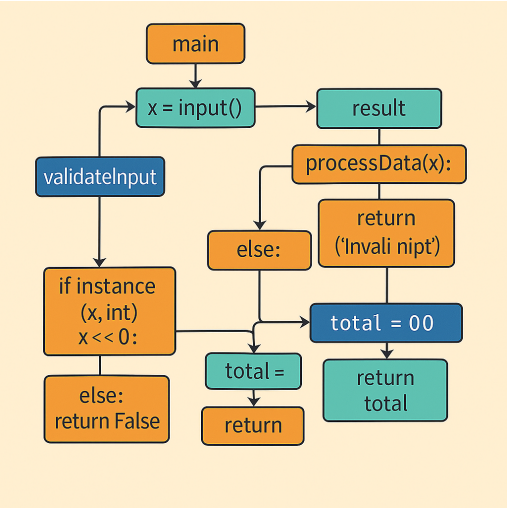


Recommender Graphs

Brain Connectomes



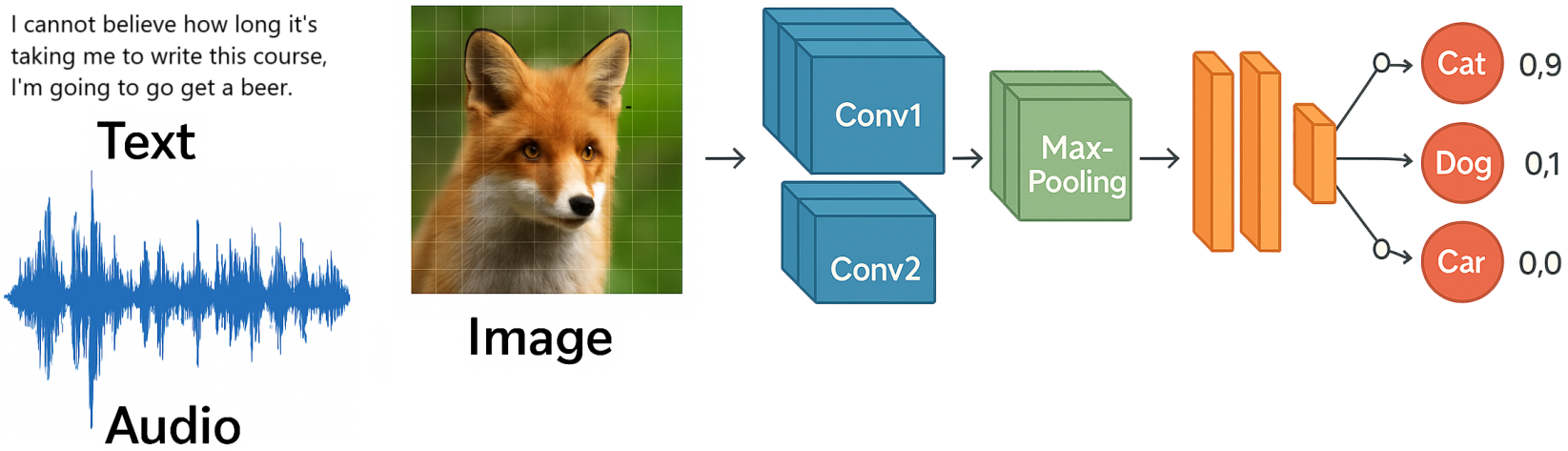
Molecular Graphs



Program Code Graphs

Why Graphs? Limitations of Traditional Deep Learning

- **Modern deep learning models** — such as **CNNs**, **RNNs**, and **Transformers** — are designed to operate on data organized as **grids** (e.g., images) or **sequences** (e.g., text, time series).
- These formats are:
  - Well understood
  - Supported by **highly optimized architectures**
  - Ideal for domains like computer vision, NLP, and time-series prediction



But There's a Gap...

Why Traditional Models Fall Short

- Many real-world problems involve **complex relationships** between entities:
  - Social connections
  - Biological interactions
  - Recommender systems
  - Scientific structures (e.g., molecules)
- These dependencies **don't fit neatly into a grid or sequence.**

But There's a Gap...

Why Traditional Models Fall Short

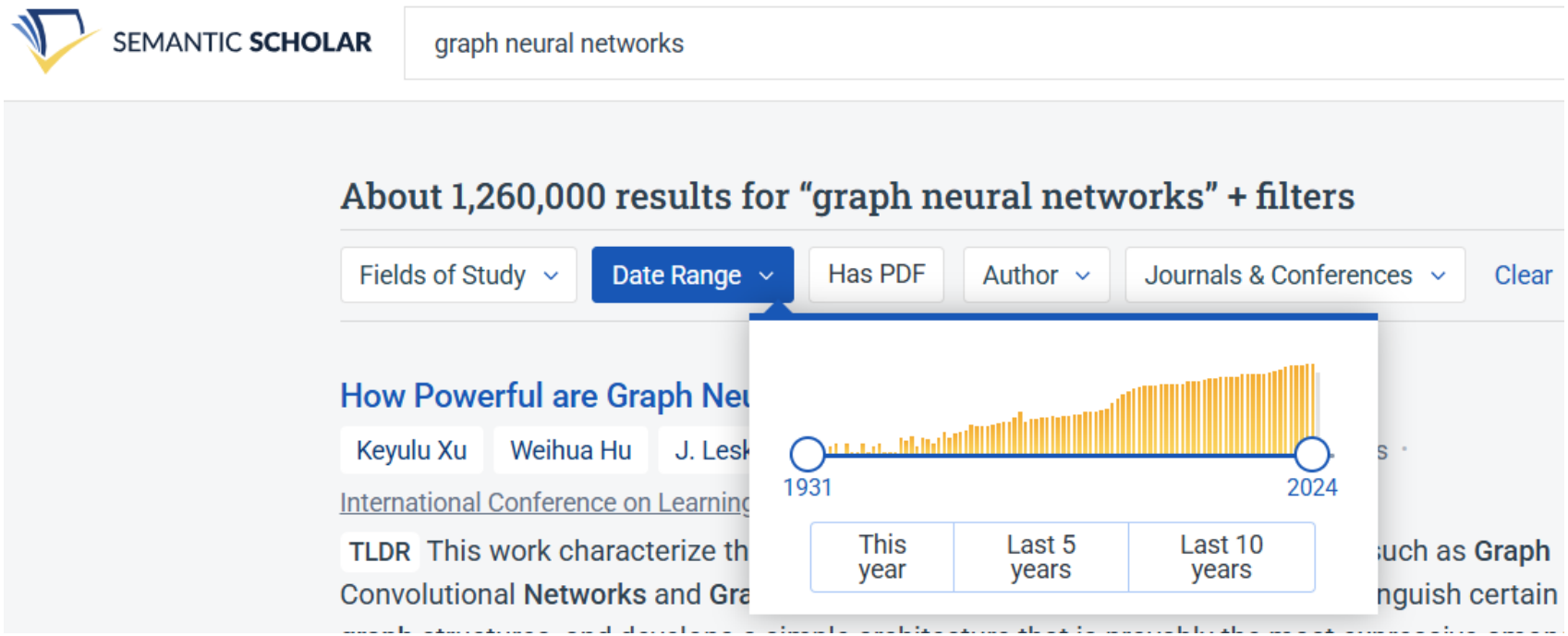
- Many real-world problems involve **complex relationships** between entities:
  - Social connections
  - Biological interactions
  - Recommender systems
  - Scientific structures (e.g., molecules)
- These dependencies **don't fit neatly into a grid or sequence.**

Graphs Fill This Gap

Graphs enable flexible, relational representations
*where structure is **irregular**, **sparse**, and **non-Euclidean** —*
and traditional deep learning models struggle.

The Rise of Graph Neural Networks

- Research on Graph Neural Networks (GNNs) has grown dramatically over the past decade.
- The figure below shows the increasing number of papers published each year with "Graph Neural Networks" in their title.



Why GNNs Are More Than Just a Trend

- GNNs combine the structure of graphs with the predictive power of deep learning.
- They can learn sophisticated node and edge representations (embeddings).
- Real-world systems actively use them for tasks requiring relational reasoning.
- Their key strength: capturing both node attributes and relationships between entities.

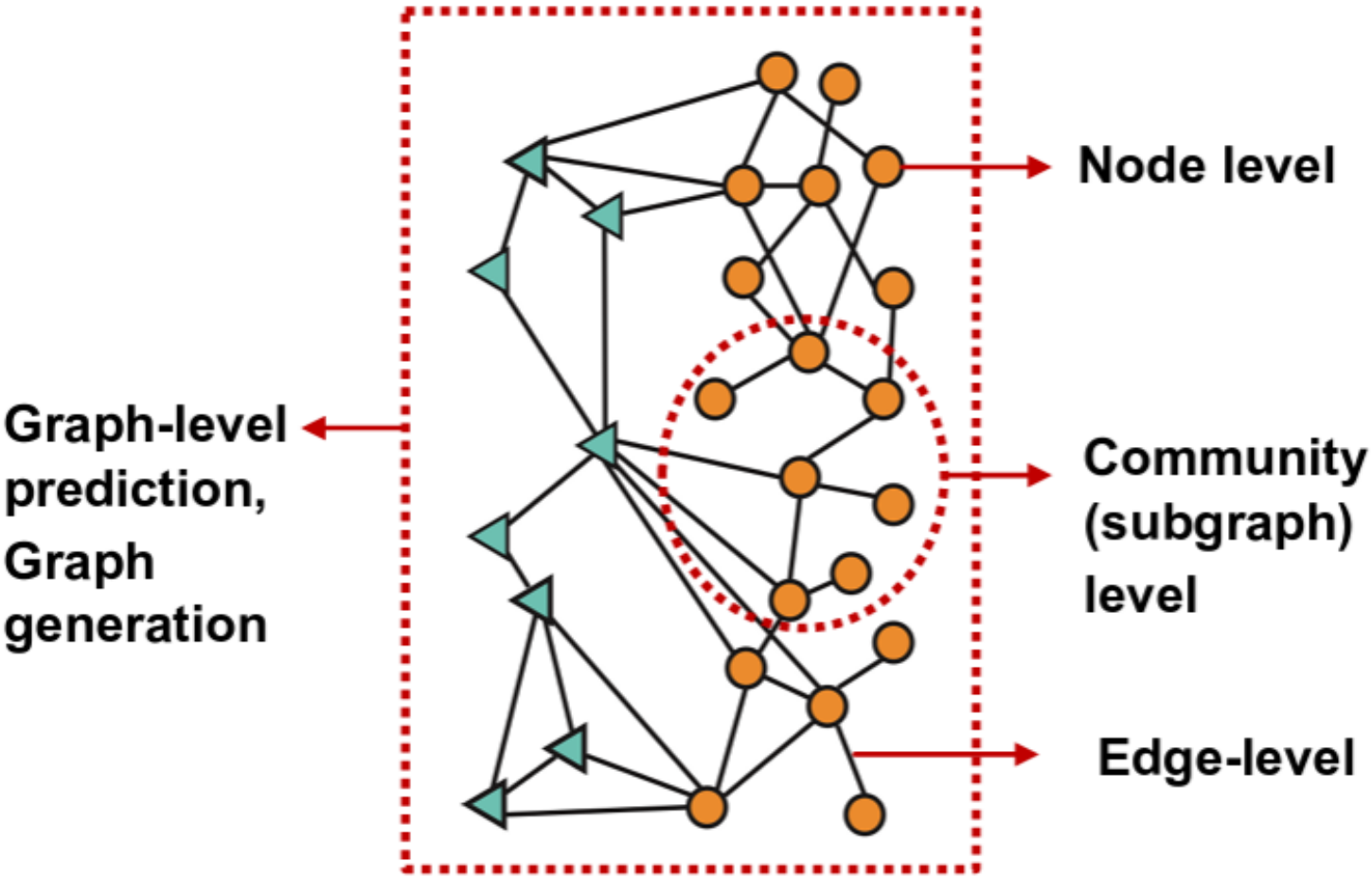
Why GNNs Are More Than Just a Trend

- GNNs combine the structure of graphs with the predictive power of deep learning.
- They can learn sophisticated node and edge representations (embeddings).
- Real-world systems actively use them for tasks requiring relational reasoning.
- Their key strength: capturing both node attributes and relationships between entities.

Is My Problem a Graph Problem?

If your data has natural **connections or interdependencies** between entities — users, products, merchants, documents — it may benefit from a graph-based approach.

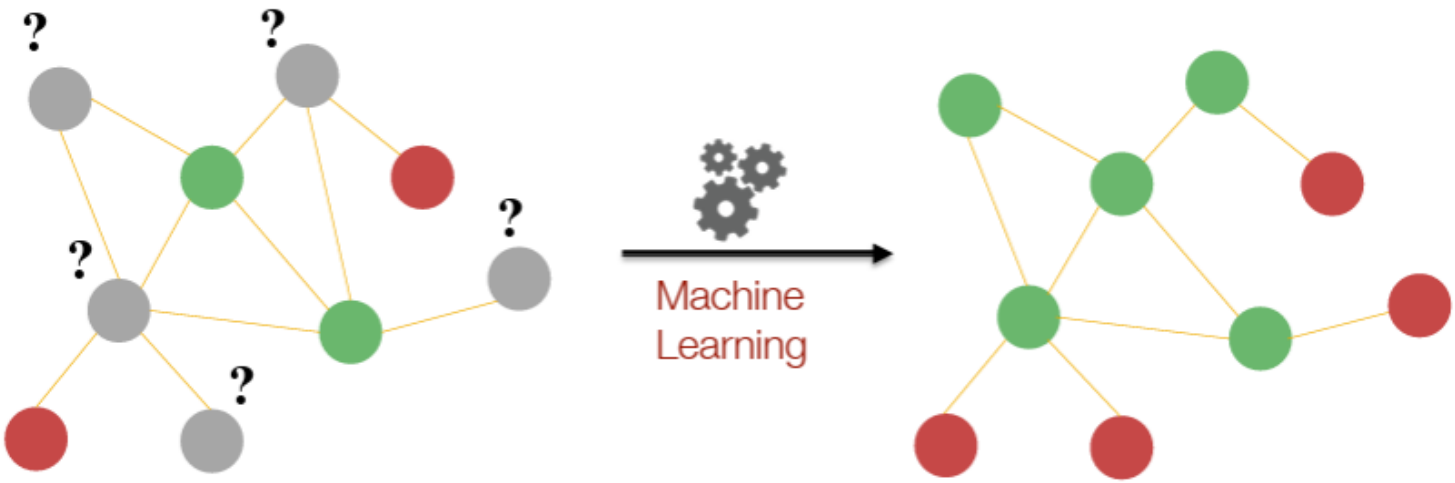
Types of Tasks



Node-Level Tasks

Goal

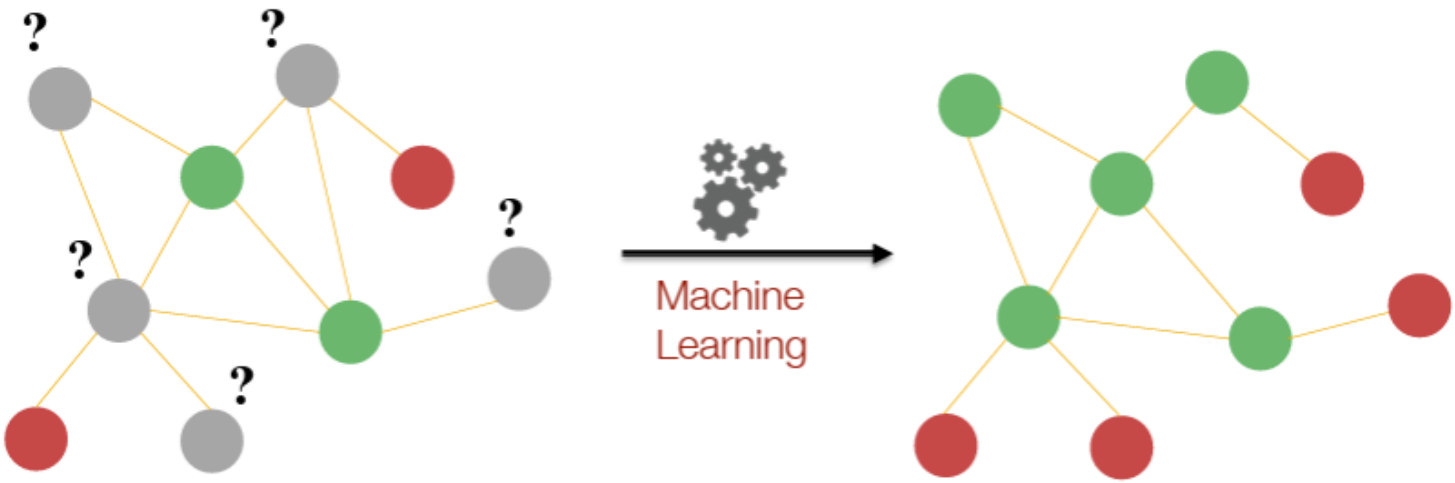
Predict a label, category, or attribute for each individual node within a graph.



Node-Level Tasks

Goal

Predict a label, category, or attribute for each individual node within a graph.



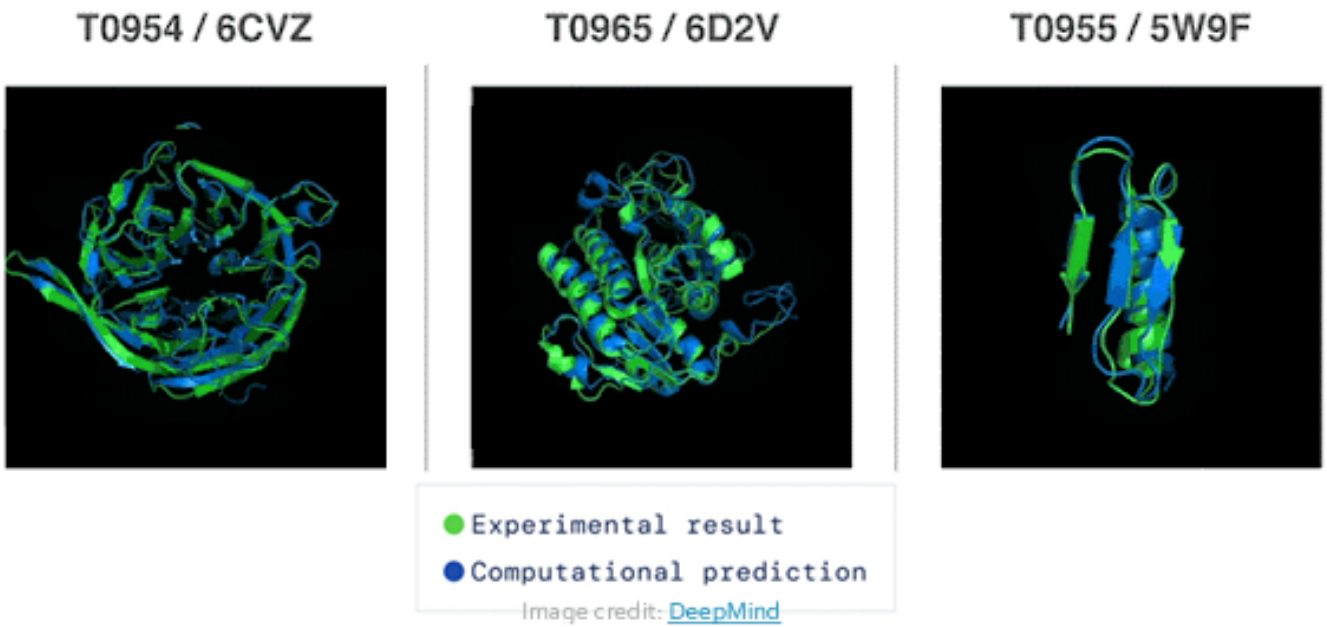
Common tasks include:

- **Node classification** (e.g., user type, document topic)
- **Node regression** (e.g., ranking score, age prediction)

Example: Node-Level Regression for Protein Folding

Breakthrough: Predicting Protein Structures with GNNs

- Recent advances in deep learning, notably by DeepMind's **AlphaFold**, have revolutionized the protein folding problem.
- Proteins can be modeled as **graphs**, where:
  - **Nodes:** Amino acids (residues) in the protein sequence
  - **Edges:** Spatial proximity or learned interaction between residues
  - **Task:** For each node, **predict its 3D coordinates** (x, y, z)



Example: Node-Level Regression for Protein Folding

Why Graph Neural Networks?

- Proteins are naturally irregular, flexible, and structured — ideal for graphs
- Their folded structure depends on long-range, non-local interactions — not just sequence order.
- GNNs enable message passing between residues to learn spatial dependencies
- Enables accurate modeling of physical constraints and folding behavior

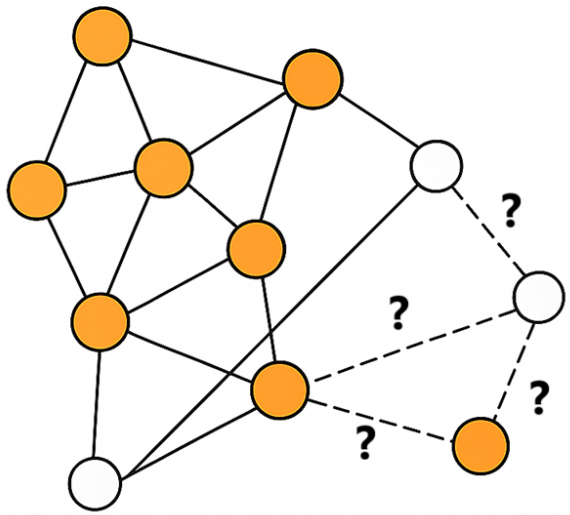
Why It Matters

AlphaFold achieved near-experimental accuracy in predicting protein 3D structures, solving a 50-year grand challenge in biology — enabled in part by graph-based learning.

Edge-Level Tasks

Goal

- Predict whether an **edge exists** between two nodes — even if it is currently missing or unknown.
- At test time: Given a set of node pairs with **no existing edge**, assign a score to each pair and **rank** them.
- The top-K ranked pairs are predicted as new or likely edges.
- It is essentially a **binary classification / scoring task** over node pairs.



Example: Edge-Level Prediction for Recommender Systems

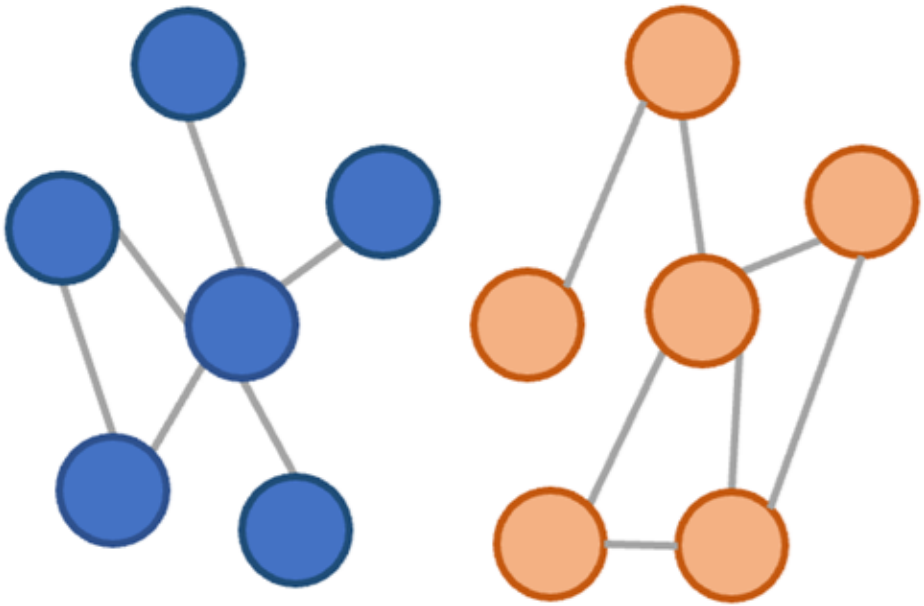
Problem Setting

- Model user-item interactions as a bipartite graph:
  - **Nodes:** Users and items (e.g., movies, books, products)
  - **Edges:** Observed interactions (e.g., rating, click, purchase)
- Goal: **Predict new edges** between users and items they haven't interacted with yet.
- This is a classic **link prediction** task on a bipartite graph.

Graph-Level Tasks

What Are Graph-Level Tasks?

- The goal is to predict a label, property, or output for an entire graph — not individual nodes or edges.
- The model learns a **global representation (embedding)** of the entire graph structure and features.

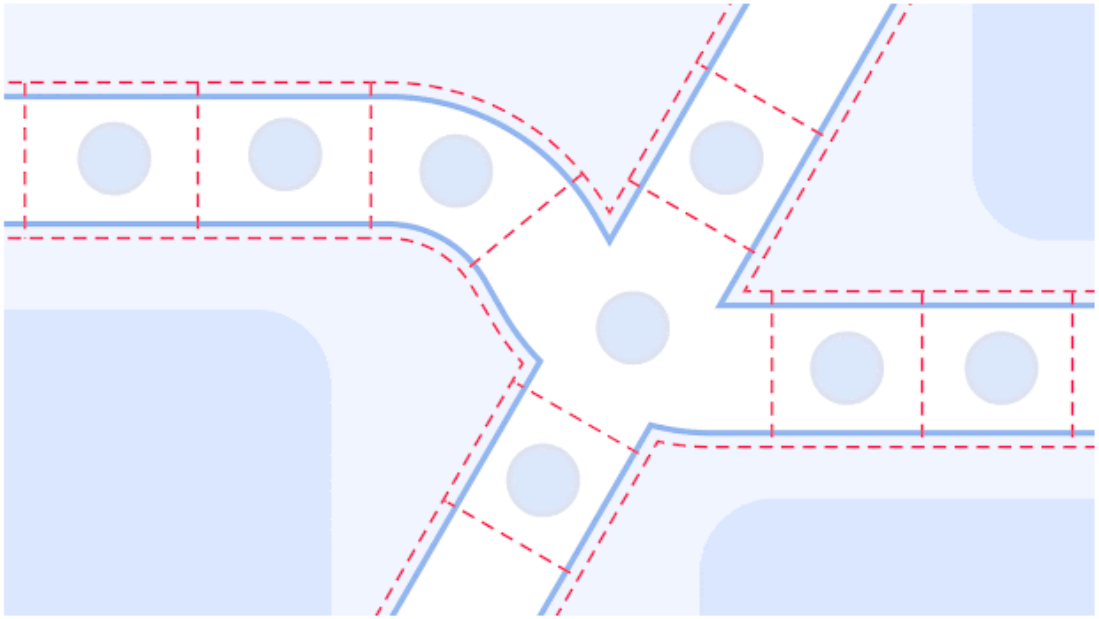


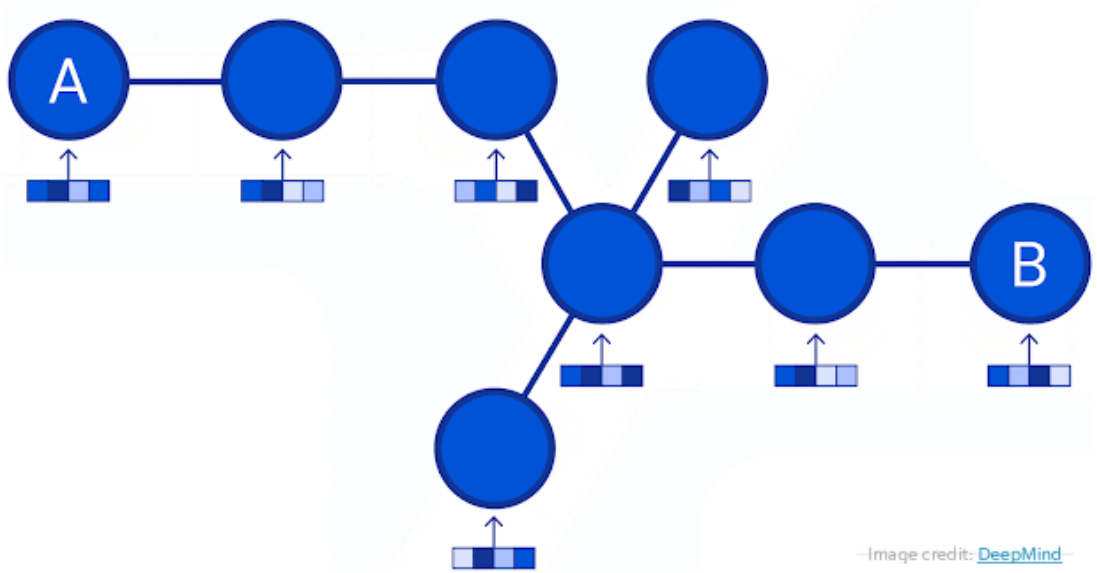
Example: ETA Predictions in Google Maps

Problem Setting

- Road Network as a Graph
  - **Nodes:** Represent road segments or intersections.
  - **Edges:** Connectivity between road segments / intersections (roads).
- Task: Predict the **Time of Arrival (ETA)** at the destination given the starting point.
- Use **real-time traffic data** (traffic congestion, accidents) to adjust the travel time along each road segment.

Example: ETA Predictions in Google Maps





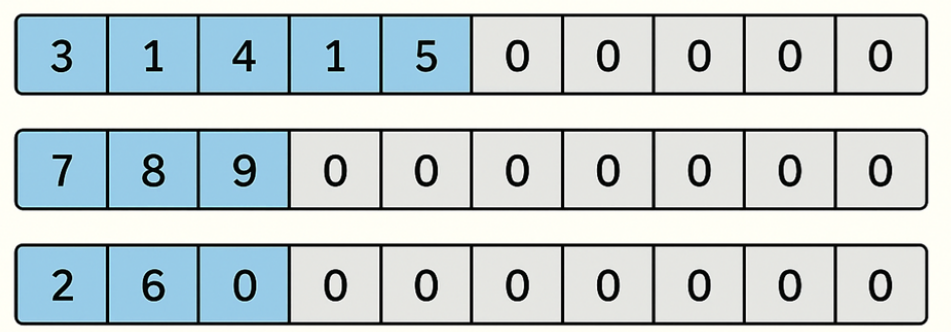
Why is Graph Deep Learning Challenging?

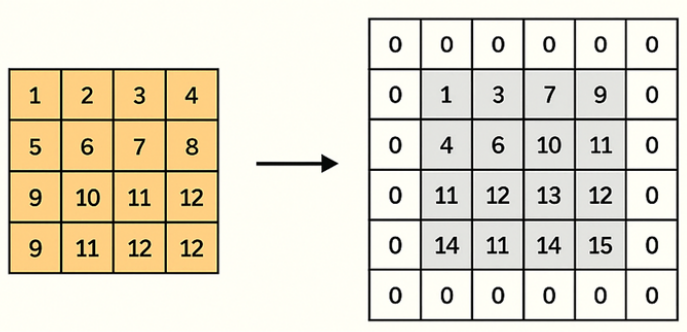
Irregular structure

Networks are inherently complex

Irregular Structure of Graphs

- Unlike images or sequences, graphs have a **highly irregular structure**.
- Traditional models like CNNs or RNNs rely on fixed input shapes (e.g., images with consistent height and width, or padded sequences).





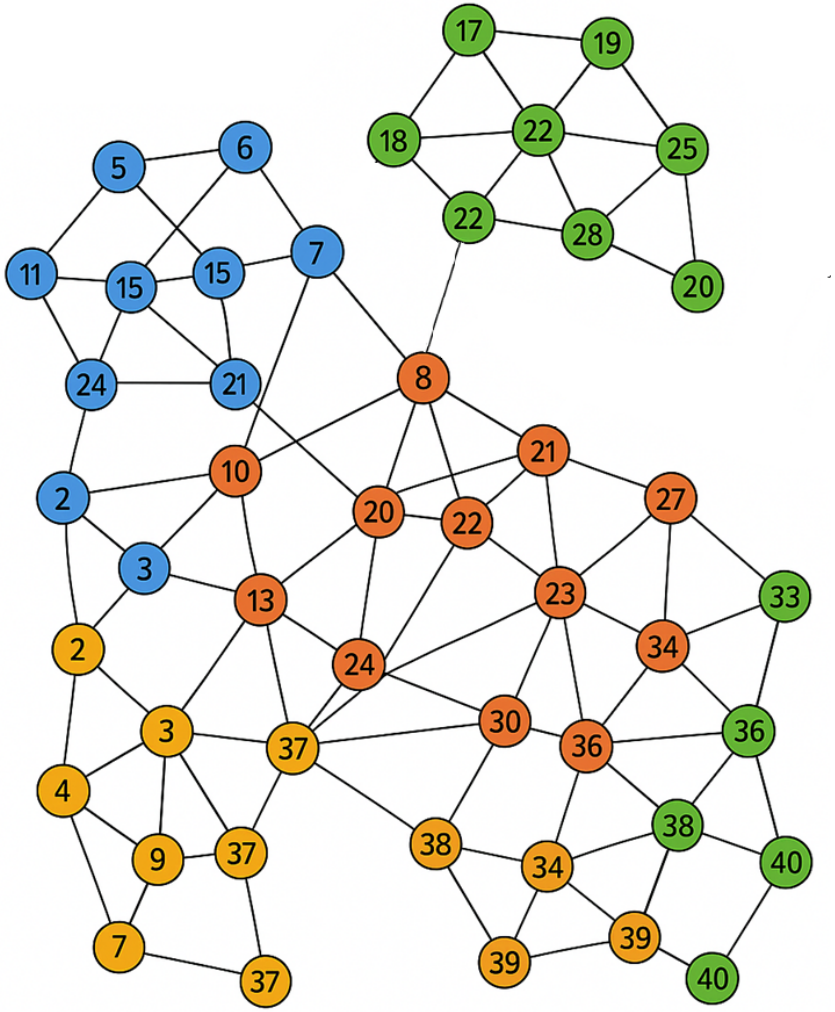
Why is Graph Deep Learning Challenging?

Irregular structure

- Graphs, however, can:
  - Vary widely in the number of **nodes**.
  - Have **nodes with varying degrees of connectivity**.

Key Implications

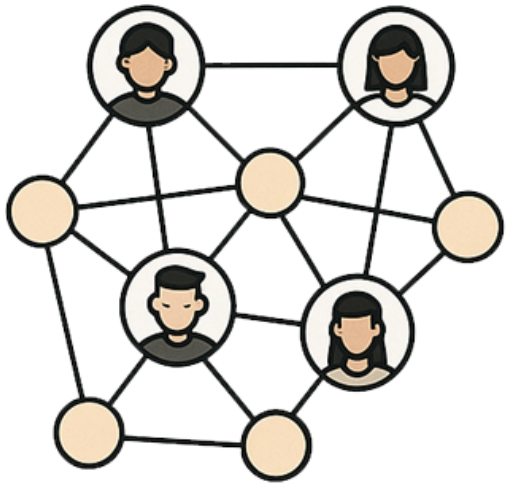
- This irregularity makes it difficult to apply standard neural operations (e.g., **convolution** or **pooling**), which assume consistent input sizes.
- Developing operations that can effectively work with these varying structures is a fundamental challenge in graph deep learning.



Why is Graph Deep Learning Challenging?

Interdependence of Data Points

- Traditional machine learning models typically assume that instances are **independent** of each other.
- However, in graph data, this assumption doesn't hold:
  - Each **node** (data point) is inherently **connected** to others.



Why This Matters

- The interconnected nature of graph data requires models that can handle these **dependencies**.
- Traditional models are not designed to account for these relationships.
- We need models that can learn from the *interactions* and *dependencies* between nodes.

Why is Graph Deep Learning Challenging?

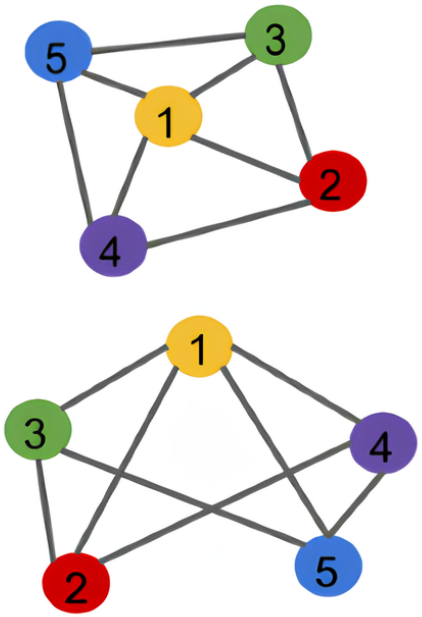
Permutation Invariance

Traditional data: fixed order

- **Images**, **text**: the order of data is meaningful
- **CNNs** and **RNNs** process data in a fixed order.

Graph data: **NO** fixed order

- In **graphs**, nodes (entities) have no inherent order.
- **Permutation invariance**: edges connect nodes, but the order of the nodes doesn't change the graph.



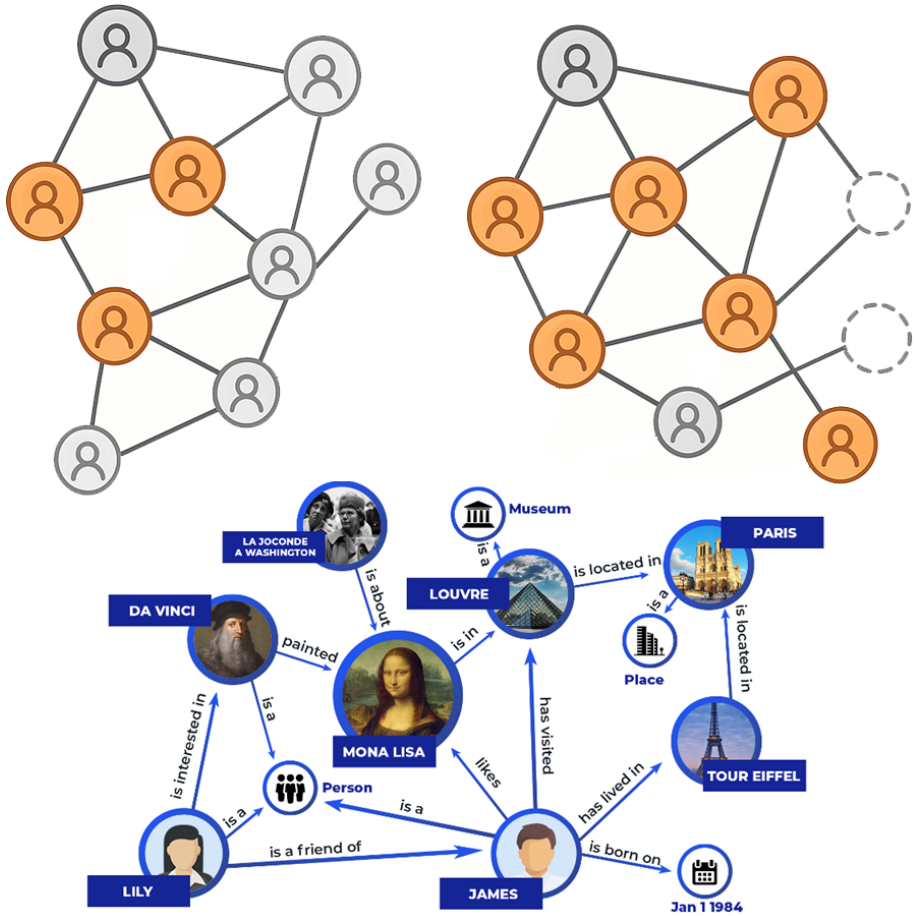
Challenges for Graph Models

- The model should give the same output regardless of the order in which nodes and edges are presented.
- Shuffling node IDs does not change the graph structure, posing challenges for traditional models that rely on fixed input structures.

Why is Graph Deep Learning Challenging?

Dynamic and Multimodal Graphs

- **Dynamic** graphs:
  - Real-world graphs are evolving over time: nodes and edges can be added, removed, or updated.
- **Multimodal** graphs:
  - Contain different types of nodes and edges
  - Different node and edge types can have various associated features, such as:
    - **User nodes** with features like age, location, interests
    - **Transaction edges** with attributes like purchase amount, time, and product type.



Basics of Deep Learning

Supervised Learning: predict an output $\boldsymbol{y}$ from an input $\boldsymbol{x}$.

- The input x can be vectors, sequences, matrices, or **graphs**.
- The task is framed as an optimization problem:

$$\min_{\theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$$

- θ: a set of **parameters** we optimize
  - could contain one or more scalar, vector, matrices, etc.
- $\mathcal{L}$: **Loss Function**. Examples:
  - **L2 loss:**

$$\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x})) := \|y - f(x)\|^2$$

  - **Cross Entropy:**

$$\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x})) := -\sum_{i=1}^{C} y_i \log f(x)_i$$

    where $y \in \{0, 1\}^C$ is a categorical vecor (one-hot encoding).
  - Other loss function: L1 loss, max margin (hinge loss), etc.

Basics of Deep Learning

- **f** can be a simple linear layer, an MLP, CNN, or other neural networks (e.g., a GNN later).
  - A multi-layer perceptron (MLP) is a neural network composed of stacked layers:

$$f_i(\boldsymbol{x}) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

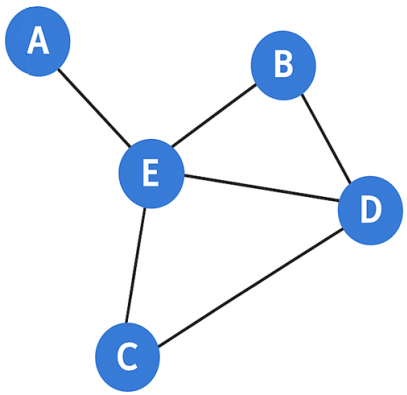
    - $\boldsymbol{W}$ and $\boldsymbol{b}$ are learnable parameters (weights and biases).
    - σ is the activation function (e.g., ReLU, Sigmoid).
- **Optimization**: We solve the optimization problem using gradients:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$$

  - η is the learning rate.
  - $\boldsymbol{\nabla_\theta \mathcal{L}} \rightarrow$ directional derivative in the direction of largest increase
  - Use stochastic gradient descent (SGD) to optimize $\mathcal{L}$ for weights θ over many iterations.
  - Other higher order optimisers: RMSprop, Adam, Adagrad, etc.

Setup

- Assume we have a graph $G = (V, E)$, where:
  - $V \rightarrow$ the set of graph nodes $V = \{v_1, ..., v_N\}$, $N =$ num. nodes
  - $E \rightarrow$ the edge set, each edge defined as (v_i, v_j)
- $X = [x_1, ..., x_N] \rightarrow$ node feature set, $X \in \mathbb{R}^{NxD}$, $D =$ dimension of x_i
- $A \in \mathbb{R}^{NxN} \rightarrow$ adjacency matrix:
  - binary: $1's$ and $0's$
  - weighted: each edge has a strength $e_{ij} \in R$:
    - $e_{ij} > 0$, if $(v_i, v_j) \in E$
    - $e_{ij} = 0$, if $(v_i, v_j) \notin E$
- $D_{ii} = \sum_j A_{ij}$ degree matrix



Graph G

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

Adjacency matrix A

	A	B	C	D	E
A	1	0	0	0	0
B	0	2	0	0	0
C	0	0	2	0	0
D	0	0	0	3	0
E	0	0	0	0	4

Degree matrix D

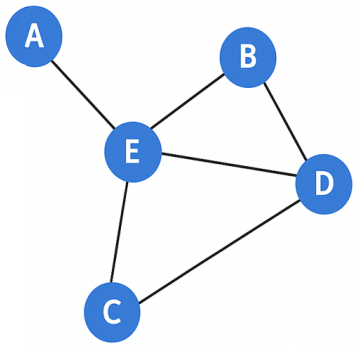
A	-1.1	3.2	4.3
B	0.2	6.1	-1.3
C	1.2	1.3	1.4
D	-1.4	4.3	3.1
E	1.3	2.8	4.2

Feature vector X

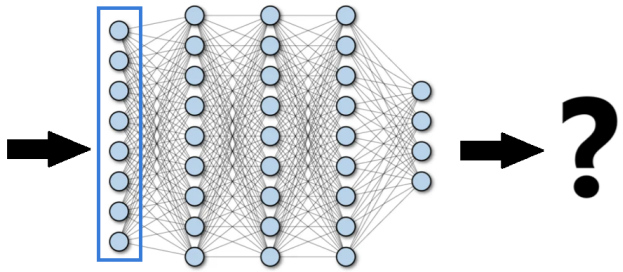
How Can We Apply Neural Networks to Graphs?

Naive Approach for Graph Representation

- Join the *adjacency matrix* and *node features*
- Feed them into a deep neural network.



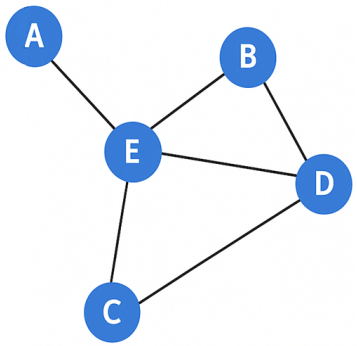
	A	B	C	D	E	Feat		
A	0	0	0	0	**1**	-1.1	3.2	4.3
B	0	0	0	**1**	**1**	0.2	6.1	-1.3
C	0	0	0	**1**	**1**	1.2	1.3	1.4
D	0	**1**	**1**	0	0	-1.4	4.3	3.1
E	**1**	**1**	**1**	**1**	0	1.3	2.8	4.2



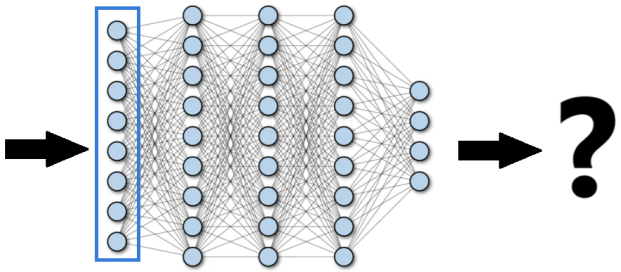
How Can We Apply Neural Networks to Graphs?

Naive Approach for Graph Representation

- Join the *adjacency matrix* and *node features*
- Feed them into a deep neural network.



	A	B	C	D	E	Feat		
A	0	0	0	0	1	-1.1	3.2	4.3
B	0	0	0	1	1	0.2	6.1	-1.3
C	0	0	0	1	1	1.2	1.3	1.4
D	0	1	1	0	0	-1.4	4.3	3.1
E	1	1	1	1	0	1.3	2.8	4.2

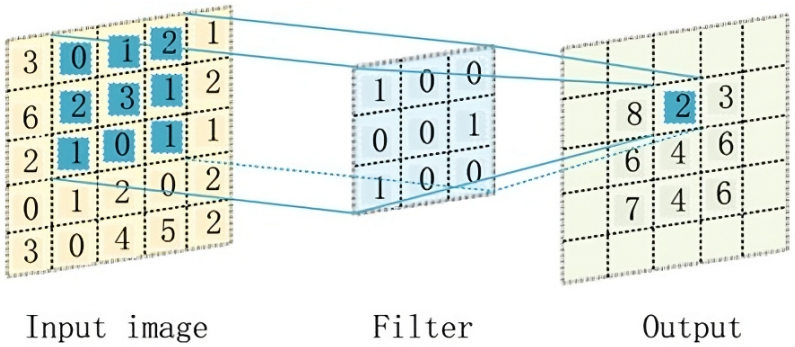


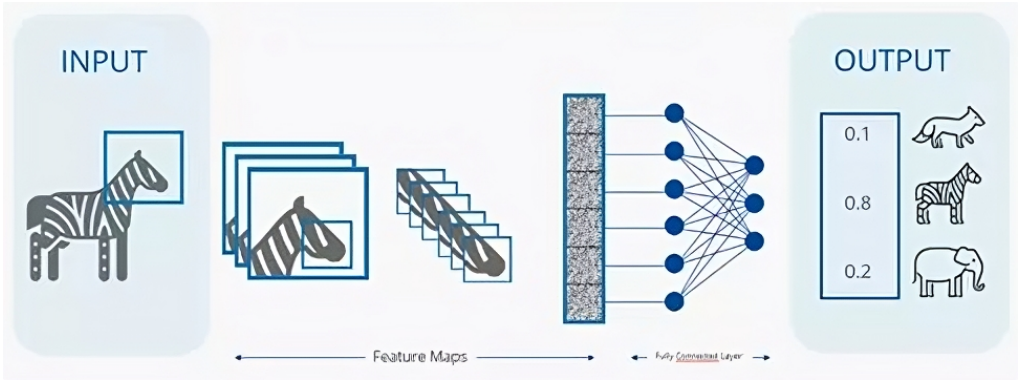
Problems with This Approach

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

How Can We Apply Neural Networks to Graphs?

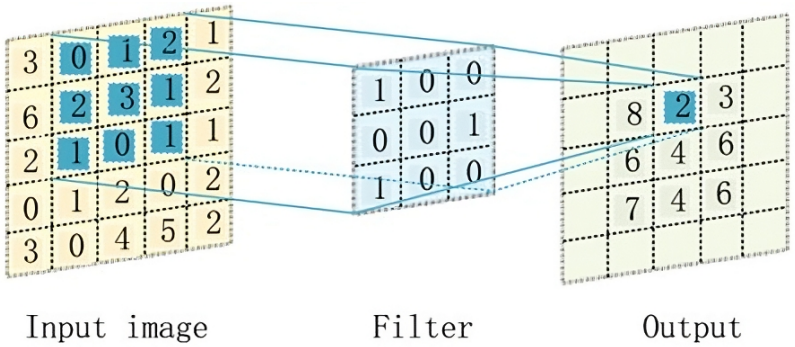
Idea: Convolutional Networks

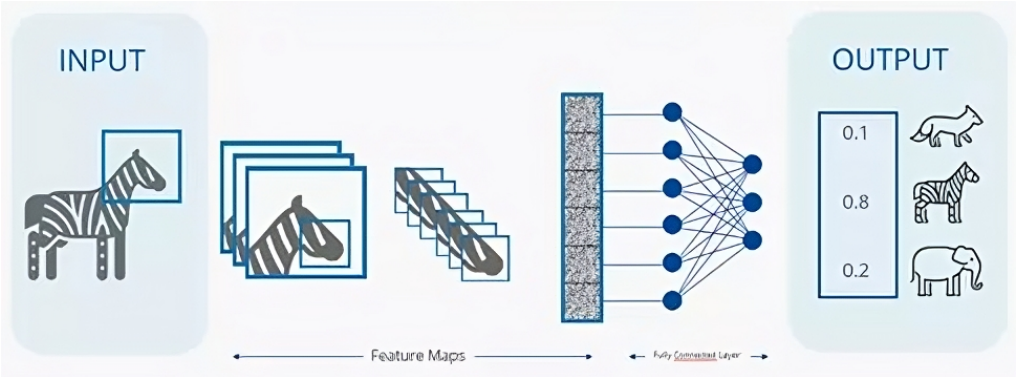




How Can We Apply Neural Networks to Graphs?

Idea: Convolutional Networks





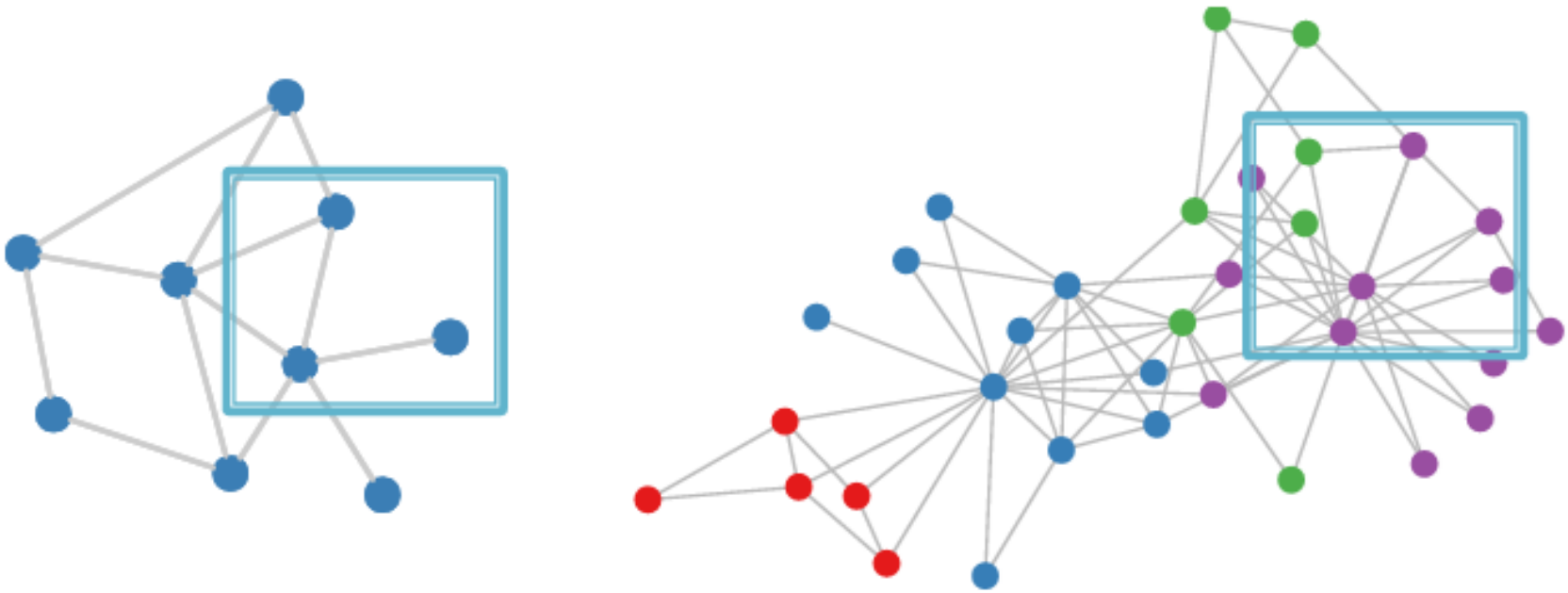
Goals

- Generalize convolutions beyond simple lattices
- Leverage node features/attributes.

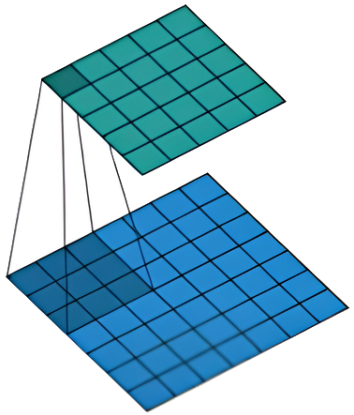
From Images to Graphs

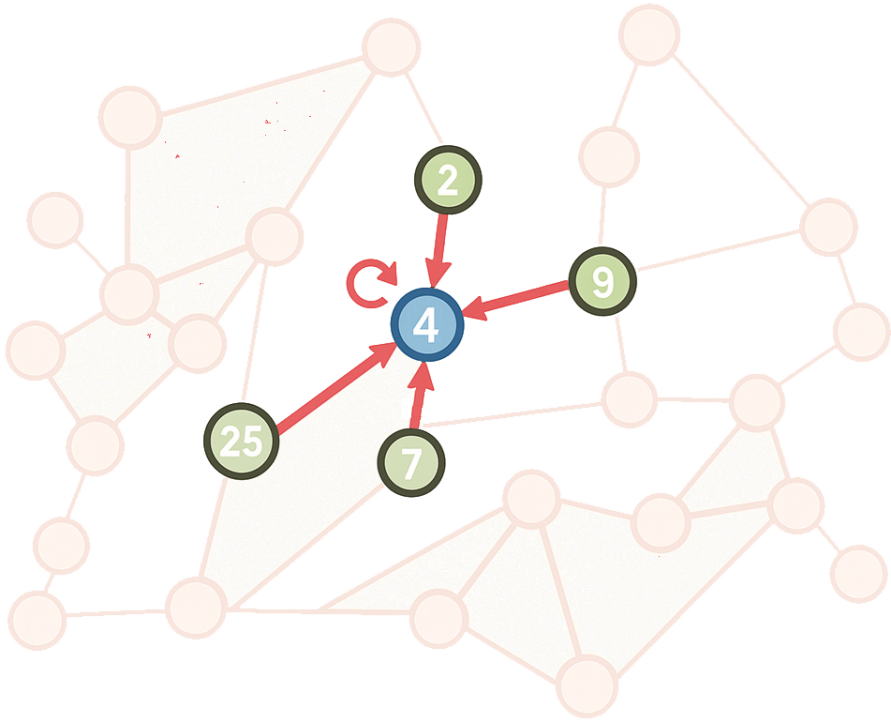
What if Our Data Looks Like This?

- *Graph-structured data* lacks a fixed notion of locality.
- There's no *sliding window* over graphs.
- Graphs are **permutation-invariant**.



From Images to Graphs





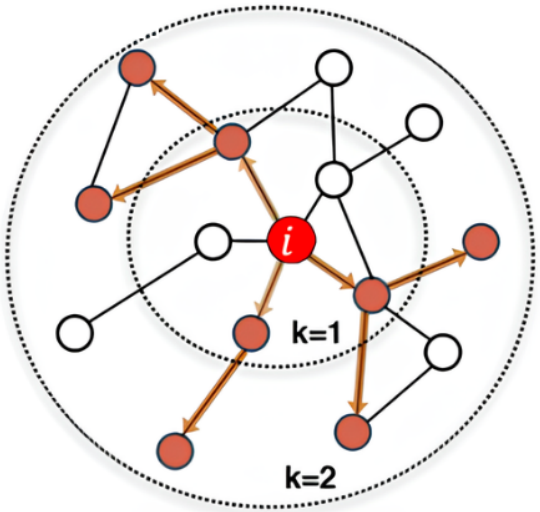
Idea

- **Transform** neighbor information and **combine** it:
  - Transform *messages* h_i from neighbors: $W_i h_i$
  - Sum them up: $\sum_i W_i h_i$

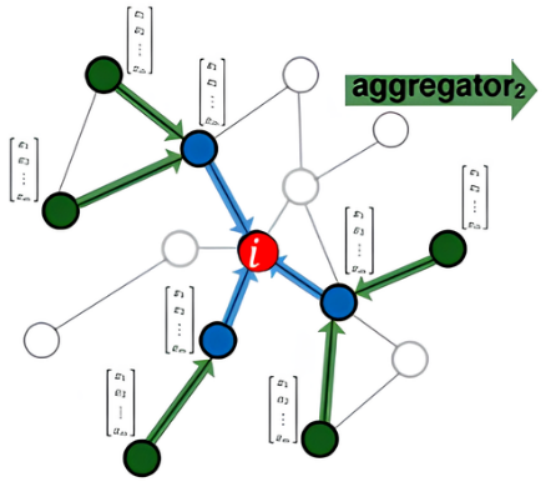
Graph Neural Networks

Key Ideas

- The neighborhood of a node defines a **computation graph**
- Learn how to **propagate** information across the graph to compute node features



Determine node computation graph

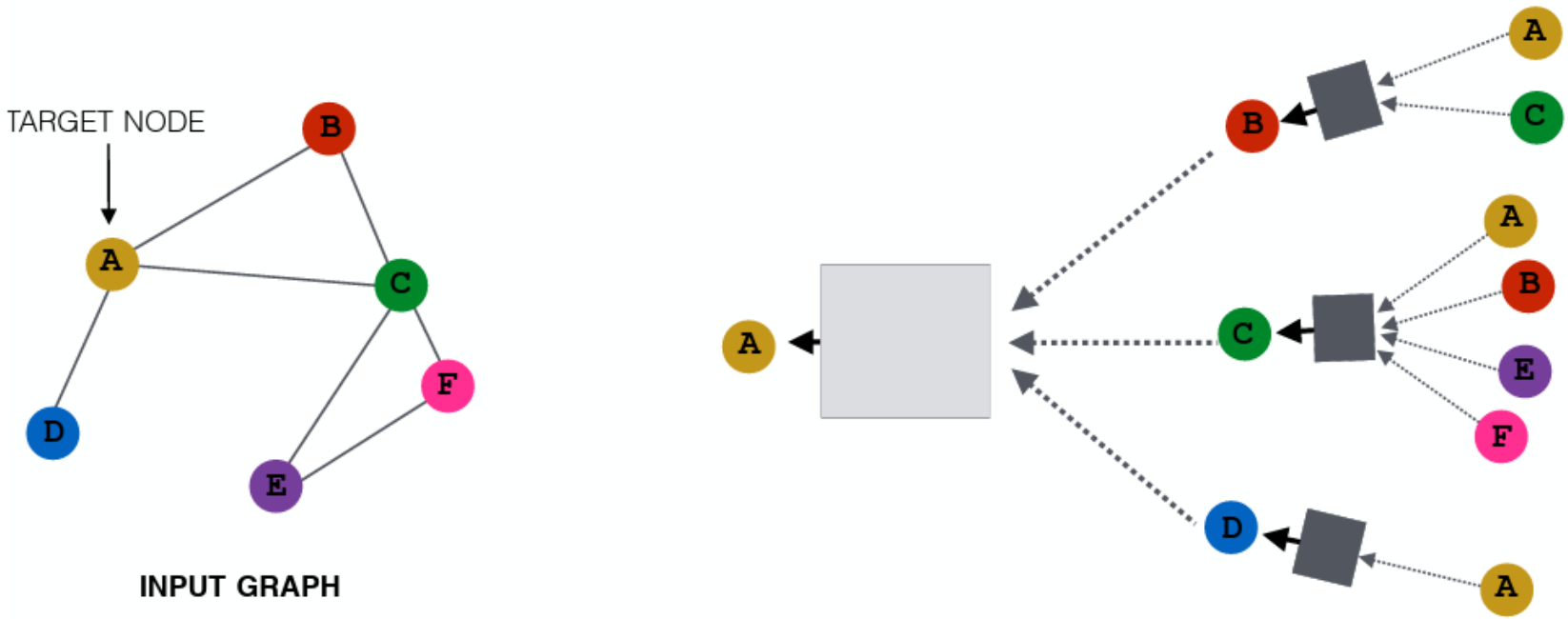


Propagate and transform information

Idea: Aggregating Information from Neighbors

Key Idea

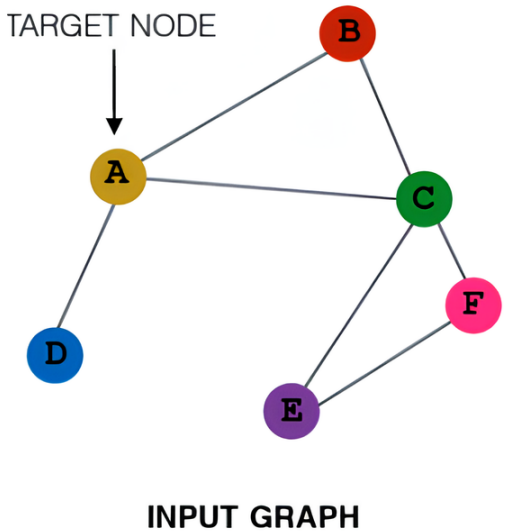
- Generate node embeddings based on **local network neighborhoods**

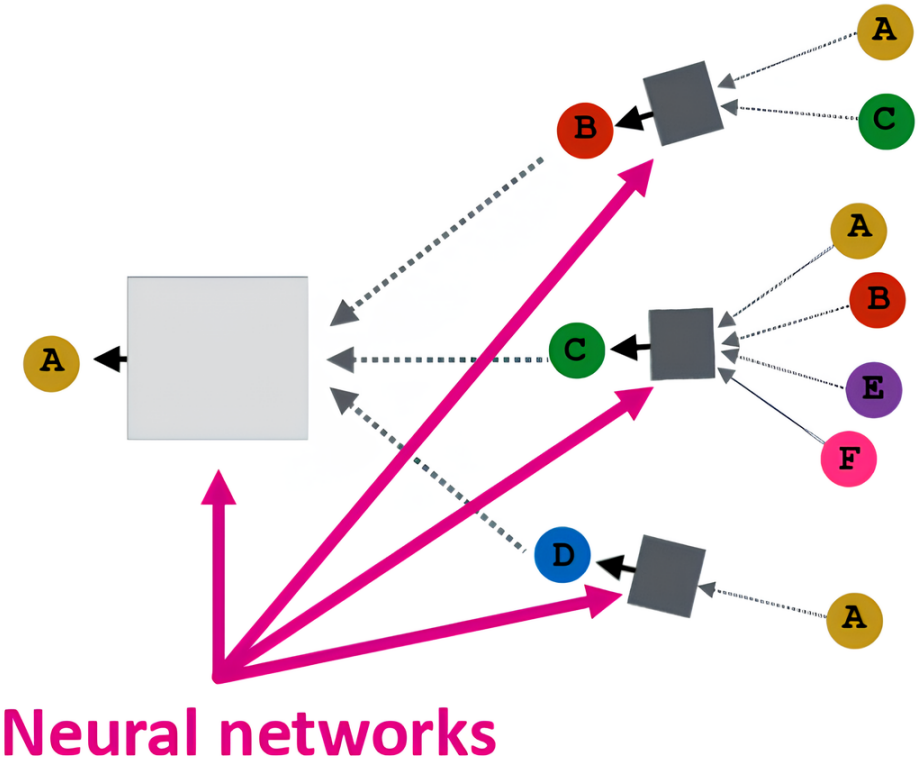


Idea: Aggregating Information from Neighbors

Key Idea

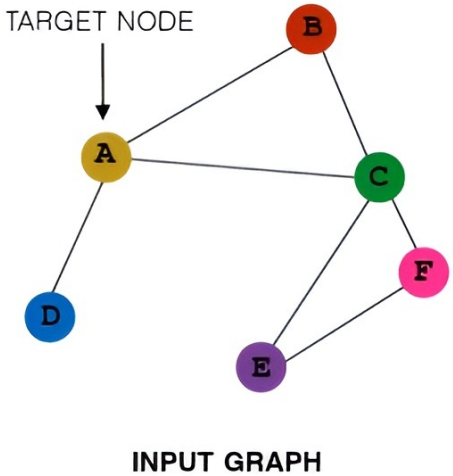
- Generate node embeddings based on **local network neighborhoods**

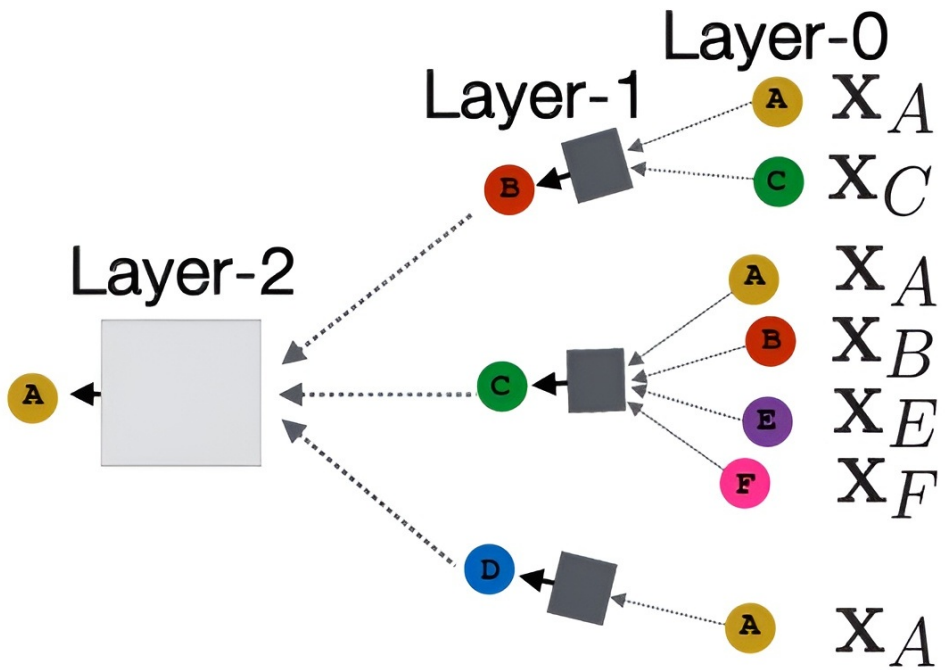




Network Depth

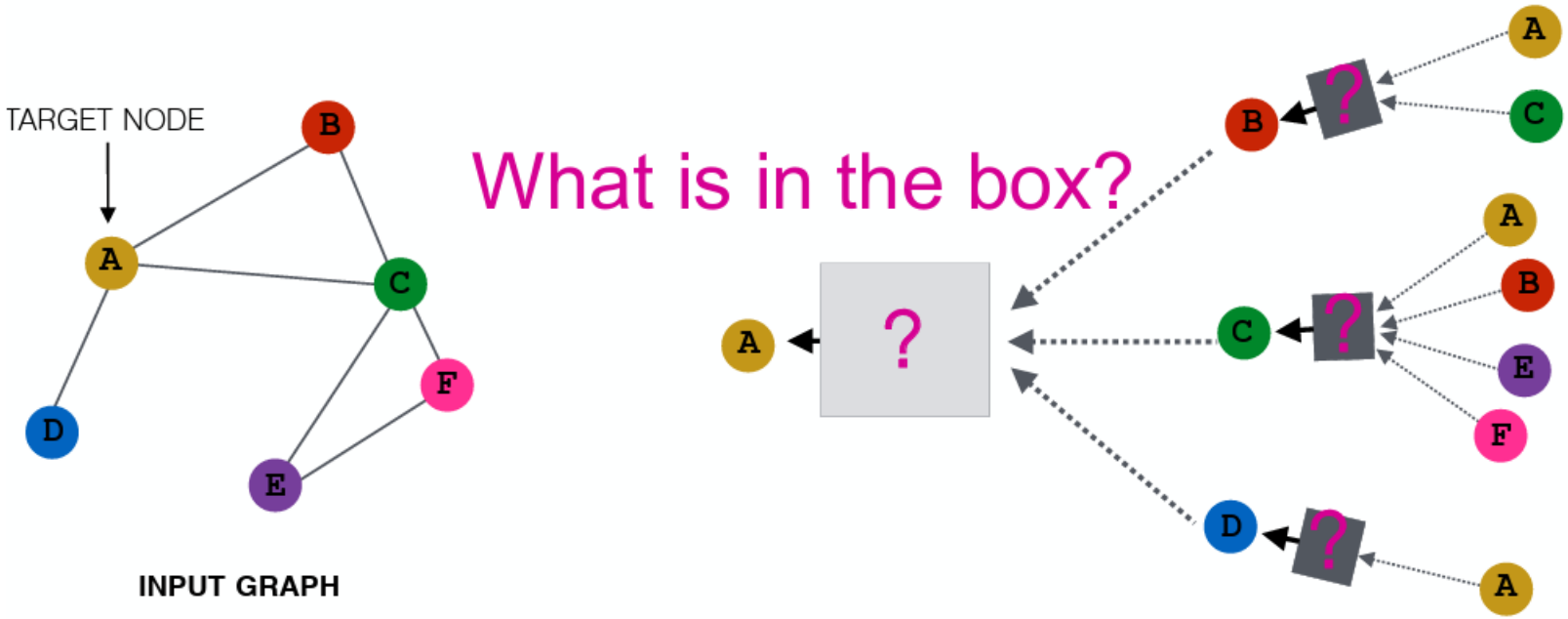
- The network can have an arbitrary depth.
- At each layer, nodes have **embeddings** (features).
- **Layer 0**: embedding of node v is its input feature x_v.
- **Layer-k**: embeddings h_v contain information from nodes that are K hops away in the graph.





Neighborhood Aggregation

Key distinction between GNNs: How different approaches aggregate information across layers.



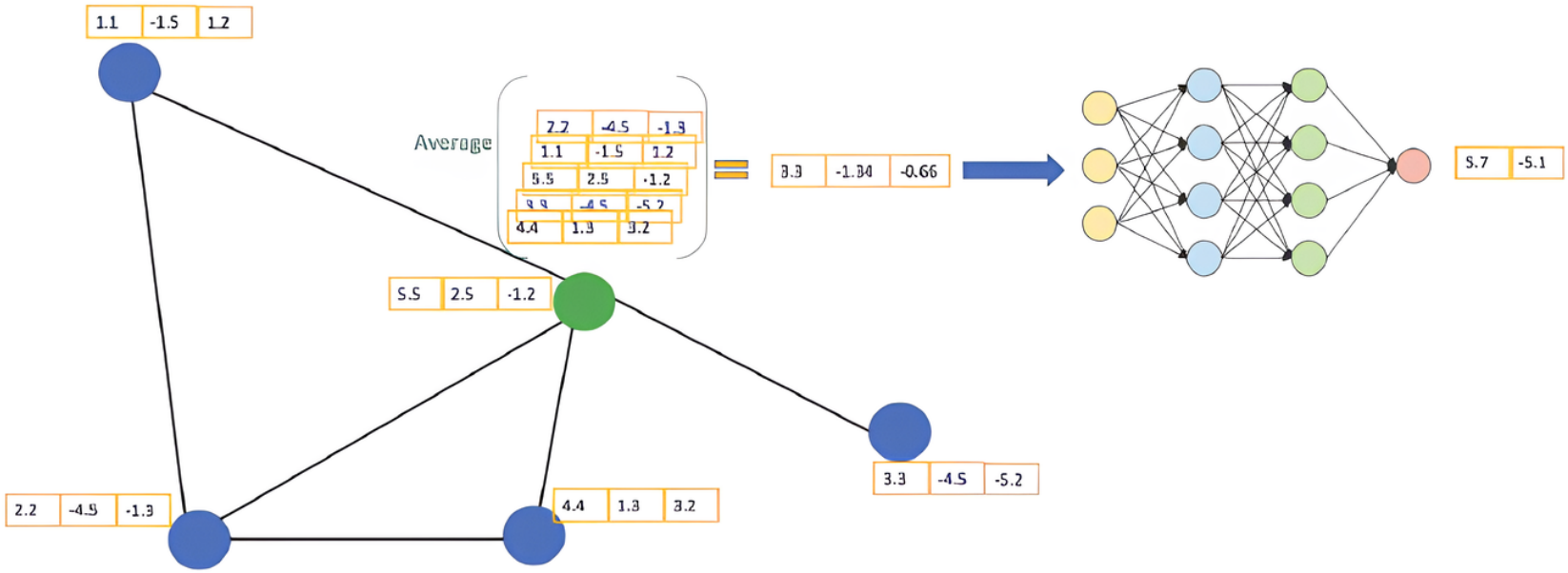
Neighborhood Aggregation

Basic approach?

Neighborhood Aggregation

Basic approach?

Average information from neighbors and apply a neural network.



Graph Deep Encoder: The Math

Basic approach?

Average information from neighbors and apply a **neural network**.

$h_v^{(0)} = x_v$ ← Initial 0-th layer embeddings are equal to node features

Graph Deep Encoder: The Math

Basic approach?

Average information from neighbors and apply a **neural network**.

$h_v^{(0)} = x_v$ ← Initial 0-th layer embeddings are equal to node features

$$h_v^{(k+1)} = \sigma\left(W_k^{(0)} h_v^{(k)} + W_k^{(1)} \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}\right), \forall k \in \{0, \ldots, K-1\}$$

Graph Deep Encoder: The Math

Basic approach?

Average information from neighbors and apply a **neural network**.

$h_v^{(0)} = x_v$ ← Initial 0-th layer embeddings are equal to node features

Embedding of v at layer k

$$h_v^{(k+1)} = \sigma\left(W_k^{(0)} h_v^{(k)} + W_k^{(1)} \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}\right), \forall k \in \{0, \ldots, K-1\}$$

Graph Deep Encoder: The Math

Basic approach?

Average information from neighbors and apply a **neural network**.

$h_v^{(0)} = x_v$ ← Initial 0-th layer embeddings are equal to node features

Embedding of v at layer k

$$h_v^{(k+1)} = \sigma\left(W_k^{(0)} h_v^{(k)} + W_k^{(1)} \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}\right), \forall k \in \{0, \ldots, K-1\}$$

Average of neighbor's previous layer embeddings

Graph Deep Encoder: The Math

Basic approach?

Average information from neighbors and apply a **neural network**.

$h_v^{(0)} = x_v$ ← Initial 0-th layer embeddings are equal to node features

Non-linearity (e.g., ReLU) → σ

Embedding of v at layer k → $h_v^{(k)}$

$$h_v^{(k+1)} = \sigma\left(W_k^{(0)} h_v^{(k)} + W_k^{(1)} \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}\right), \forall k \in \{0, \ldots, K-1\}$$

Average of neighbor's previous layer embeddings → $\sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}$

Graph Deep Encoder: The Math

Basic approach?

Average information from neighbors and apply a **neural network**.

$$h_v^{(0)} = x_v$$

Initial 0-th layer embeddings are equal to node features

Non-linearity (e.g., ReLU)

Embedding of v at layer k

$$h_v^{(k+1)} = \sigma\left(W_k^{(0)} h_v^{(k)} + W_k^{(1)} \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}\right), \forall k \in \{0, \ldots, K-1\}$$

Average of neighbor's previous layer embeddings

Number of layers

Graph Deep Encoder: The Math

Basic approach?
Average information from neighbors and apply a **neural network**.

$h_v^{(0)} = x_v$ ← Initial 0-th layer embeddings are equal to node features

Non-linearity (e.g., ReLU)

Embedding of v at layer k

$$h_v^{(k+1)} = \sigma\left(W_k^{(0)} h_v^{(k)} + W_k^{(1)} \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|}\right), \forall k \in \{0, \ldots, K-1\}$$

Average of neighbor's previous layer embeddings

Number of layers

$z_v = h_v^{(K)}$ ← Embedding after K layers of neighborhood aggregation

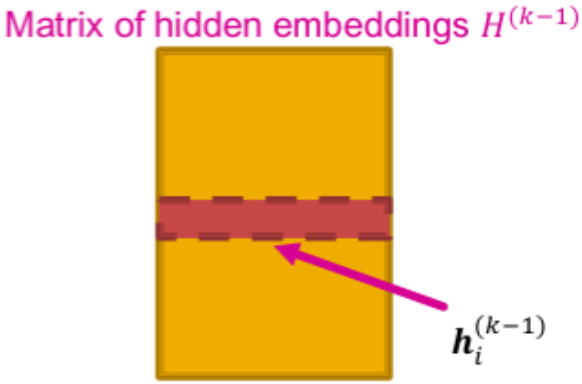
Efficient Aggregation Using Matrix Operations

Many aggregations can be performed efficiently by (sparse) matrix operations

Let $H^{(k)} = [h_1^{(k)} \quad h_2^{(k)} \quad \dots \quad h_{|V|}^{(k)}]^T$

Then the node feature aggregation can be written as:

$$\sum_{v \in N(u)} h_v^{(k)} = A_{:,v} H^{(k)}$$



Let D be the diagonal degree matrix:

$$D_{v,v} = \text{Deg}(v) = |N(v)|$$

Note: The inverse of D is also diagonal.

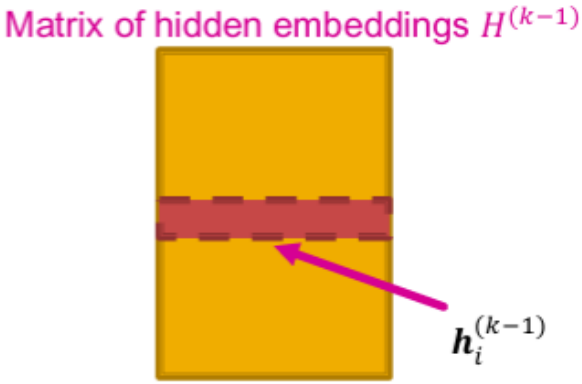
Efficient Aggregation Using Matrix Operations

Many aggregations can be performed efficiently by (sparse) matrix operations

Let $H^{(k)} = [h_1^{(k)} \quad h_2^{(k)} \quad \dots \quad h_{|V|}^{(k)}]^T$

Then the node feature aggregation can be written as:

$$\sum_{v \in N(u)} h_v^{(k)} = A_{:,v} H^{(k)}$$



Let D be the diagonal degree matrix:

$$D_{v,v} = \text{Deg}(v) = |N(v)|$$

Note: The inverse of D is also diagonal.

Therefore:

$$\sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} \Longrightarrow \boldsymbol{H^{(k+1)} = D^{-1} A H^{(k)}}$$

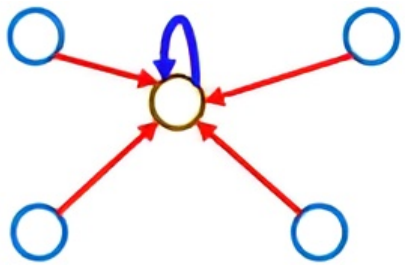
GNN Update Rule: Matrix Form

Update Equation

$$H^{(k+1)} = \sigma\left(\tilde{A}H^{(k)}W_k^{(1)^T} + H^{(k)}W_k^{(0)^T}\right)$$

where

- $\tilde{A} = D^{-1}A$: normalized adjacency matrix
- $H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^\top$: node features at layer k



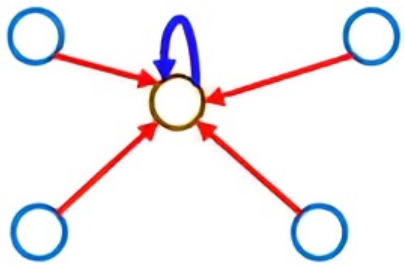
GNN Update Rule: Matrix Form

Update Equation

$$H^{(k+1)} = \sigma\left(\tilde{A}H^{(k)}W_k^{(1)^T} + H^{(k)}W_k^{(0)^T}\right)$$

where

- $\tilde{A} = D^{-1}A$: normalized adjacency matrix
- $H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^\top$: node features at layer k



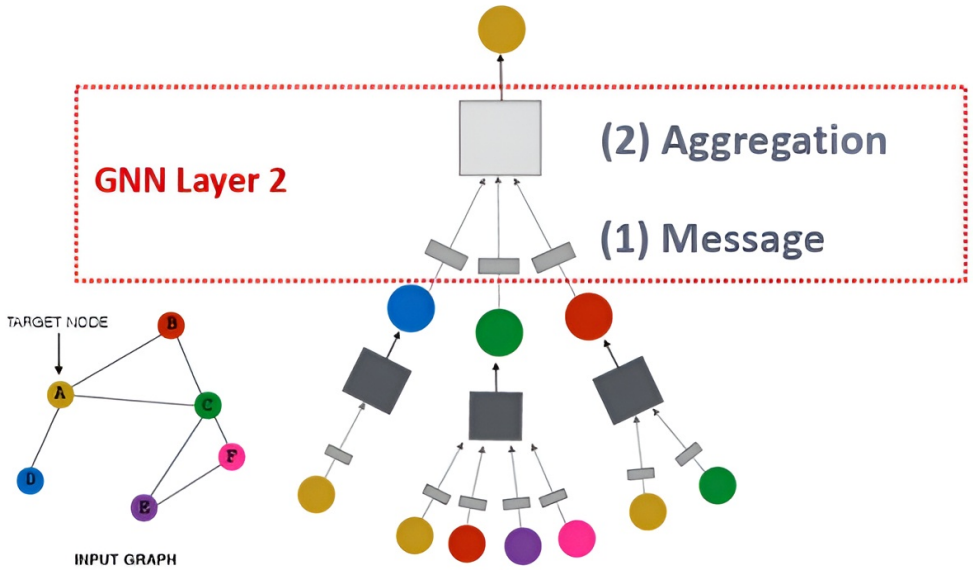
Efficiency Note

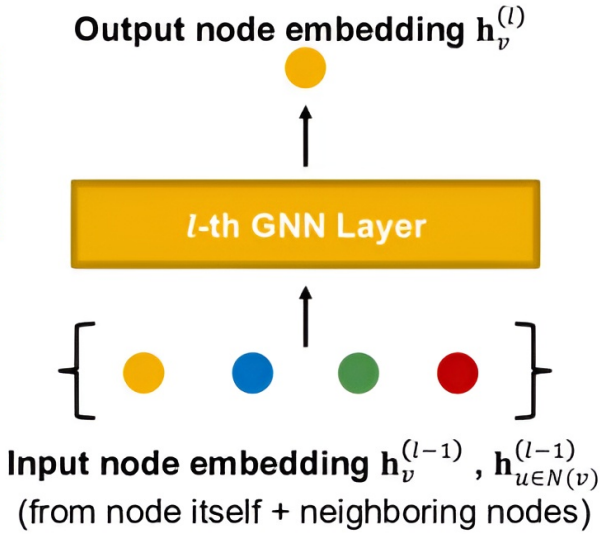
A is sparse $\Rightarrow$ enables efficient sparse matrix multiplication

A Single GNN Layer

Key Idea

- Compress a set of vectors into a single vector
- **GNN Layer = Message + Aggregation**





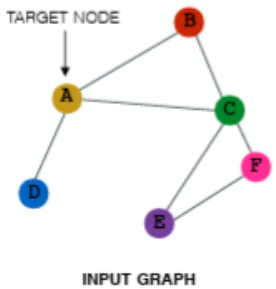
A Single GNN Layer: Message Computation

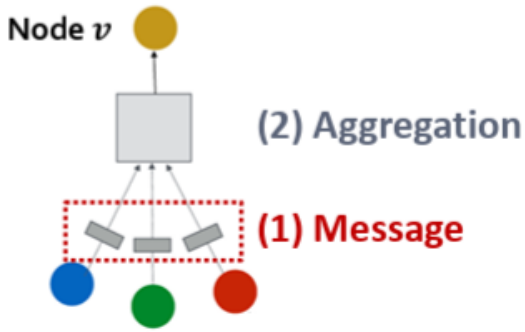
Message Computation

$$\mathbf{m}_u^{(k)} = \text{MSG}^{(k)}(\mathbf{h}_u^{(k-1)})$$

- **Intuition:** Each node creates a message to be sent to its neighbors.
- **Example:** Linear transformation

$$\mathbf{m}_u^{(k)} = \boldsymbol{W}_1^{(k)} \mathbf{h}_u^{(k-1)}$$





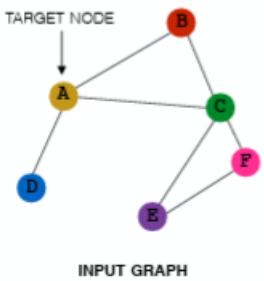
A Single GNN Layer: Aggregation

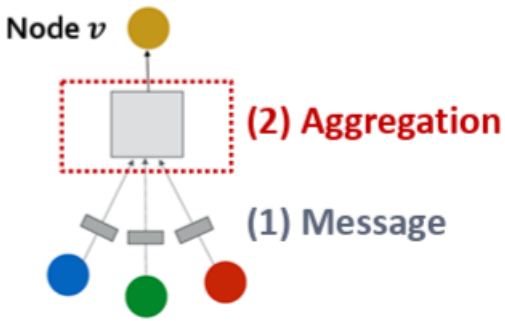
Aggregation

$$\mathbf{h}_v^{(k)} = \text{AGG}^{(k)}\left(\left\{\mathbf{m}_u^{(k)} \mid u \in \mathcal{N}(v)\right\}\right)$$

- **Intuition:** Node v aggregates messages from its neighbors $u \in \mathcal{N}(v)$
- **Examples of AGG:** `sum()`, `mean()`, or `max()`
- **Example:**

$$\mathbf{h}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \mathbf{m}_u^{(k)}$$





A Single GNN Layer: Aggregation Issue and Solution

Issue: Node's Own Information May Be Lost

- Aggregation only over neighbors: $\mathbf{h}_v^{(k)} = \text{AGG}^{(k)}\left(\left\{\mathbf{m}_u^{(k)} \mid u \in \mathcal{N}(v)\right\}\right)$
- **Problem:** $\mathbf{h}_v^{(k)}$ does not directly depend on node v's own features.

A Single GNN Layer: Aggregation Issue and Solution

Issue: Node's Own Information May Be Lost

- Aggregation only over neighbors: $\mathbf{h}_v^{(k)} = \text{AGG}^{(k)}\left(\left\{\mathbf{m}_u^{(k)} \mid u \in \mathcal{N}(v)\right\}\right)$
- **Problem:** $\mathbf{h}_v^{(k)}$ does not directly depend on node v's own features.

Solution: Include Self-Messages

- **(1) Message from node v itself:** $\mathbf{m}_v^{(k)} = \boldsymbol{W}_{\mathbf{0}}^{(k)}\mathbf{h}_v^{(k-1)}$
  (separate from $\boldsymbol{W}_{\mathbf{1}}^{(k)}$ used for neighbors)
- **(2) Aggregation:**
  - First aggregate from neighbors:

$$\mathbf{h}_{\mathcal{N}(v)}^{(k)} = \text{AGG}^{(k)}\left(\left\{\mathbf{m}_u^{(k)} \mid u \in \mathcal{N}(v)\right\}\right)$$

  - Then combine with self-message:

$$\mathbf{h}_v^{(k)} = \text{CONCAT}\left(\mathbf{h}_{\mathcal{N}(v)}^{(k)}, \mathbf{m}_v^{(k)}\right)$$

Classical GNN Layers: GCN

Graph Convolutional Network (GCN) Update Rule

$$\mathbf{h}_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_u d_v}} \boldsymbol{W}^{(k)} \mathbf{h}_u^{(k-1)}\right)$$

- **Message:**

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

  Each neighbor (and the node itself) sends a message using shared weight matrix $\boldsymbol{W}^{(k)}$.

- **Aggregation:**
  - Normalize messages by node degrees d_u, d_v
  - Sum messages from neighbors **and the node itself**
  - Apply non-linearity σ (e.g., ReLU)
- **Note:** The input graph is assumed to include self-loops.

This formulation is from Kipf & Welling (2017).

Classical GNN Layers: GraphSAGE

GraphSAGE Update Rule

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

Two-Stage Aggregation

- **Stage 1: Neighborhood Aggregation**

$$\mathbf{h}_{\mathcal{N}(v)}^{(k)} = \text{AGG}^{(k)}\left(\left\{\mathbf{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\right\}\right)$$

- **Stage 2: Combine with Node Itself**

$$\mathbf{h}_v^{(k)} = \sigma\left(\boldsymbol{W}^{(k)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_{\mathcal{N}(v)}^{(k)}\right)\right)$$

This formulation is from Hamilton et al., (2017).
It formulation supports inductive learning on unseen graphs.

Classical GNN Layers: GraphSAGE Neighbor Aggregation

- **Mean**: Take an average of neighbor features

$$\text{AGG}^{(k)} = \underbrace{\sum_{u \in \mathcal{N}(v)} \frac{\mathbf{h}_u^{(k-1)}}{|\mathcal{N}(v)|}}_{\text{Aggregation} \quad \text{Message Computation (Identity)}}$$

- **Pool**: Apply transformation before aggregation (e.g., MLP)

$$\text{AGG}^{(k)} = \underbrace{\text{Mean}\left(\left\{\text{MLP}^{(k)}(\mathbf{h}_u^{(k-1)}) \mid u \in \mathcal{N}(v)\right\}\right)}_{\text{Aggregation} \quad \text{Message Computation}}$$

- **LSTM**: Order-sensitive aggregation over neighbors

$$\text{AGG}^{(k)} = \underbrace{\text{LSTM}^{(k)}\left(\left[\mathbf{h}_u^{(k-1)} \mid u \in \pi(\mathcal{N}(v))\right]\right)}_{\text{Aggregation (learned over sequence)}}$$

Questions?