Lecture - 1 Notes

1. Fundamental Paradigm Shift

Standard machine learning (computer vision, NLP) often relies on the i.i.d. assumption (independent and identically distributed data). Graph ML rejects this. It models the dependencies between data points explicitly.

The central hypothesis of Graph ML is Representation Learning. Instead of manual feature engineering (like calculating node centrality by hand); we learn a mapping function f that encodes topological structure into a low-dimensional vector space.

$$f : u \rightarrow R^d$$

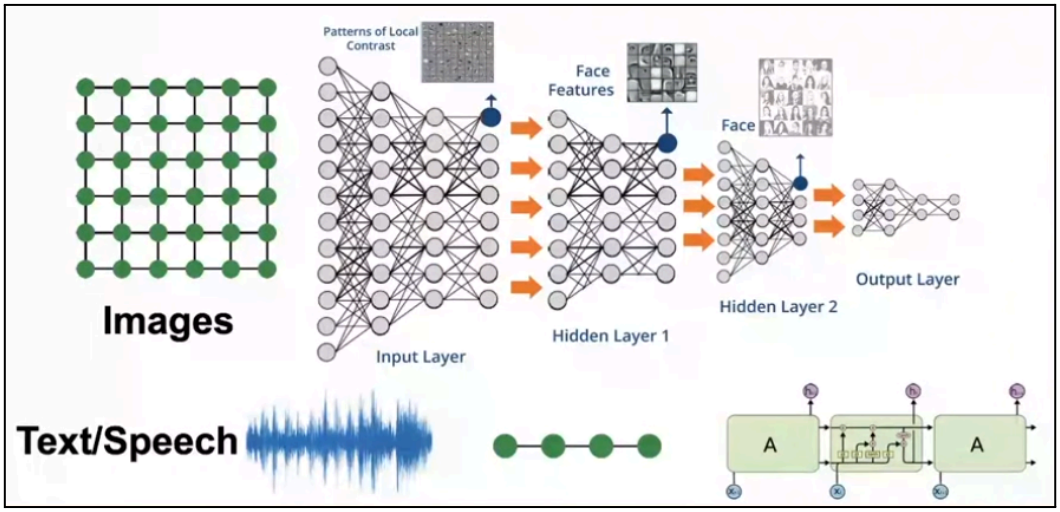
Optimization Goal: Map nodes u and v to embedding vectors z_u and z_v such that their geometric relationship in the embedding space approximates their structural relationship in the original graph.

$$Similarity\ (u, v) \approx z_u^{\top} z_v$$

Generalization of Standard Deep Learning

Traditional neural architectures are actually special cases of Graph Neural Networks (GNNs):

- CNNs: Operate on Grid Graphs. Nodes are pixels and edges are fixed spatial adjacencies (up, down, left, right). The structure is regular and static.
- RNNs/Transformers: Operate on Linear/Fully Connected Graphs. Text is a sequence (linear graph) or a fully connected graph with attention weights as edge attributes



2. Task Taxonomy & Applications

To solve a graph problem, you must first classify it into one of four levels. The choice of level dictates the loss function and architecture.

1: Node-Level Tasks

Goal: Classify or regress a property of a single node u.

- Mechanism: The model aggregates information from the node's *neighborhood* (receptive field).
- Case Study: AlphaFold.
    - *Problem:* Predict 3D coordinates of atoms in a protein.
    - *Graph:* Nodes = Amino acids; Edges = Proximity (sequence-wise or spatial).
    - *Task:* Node regression (predicting coordinates).

2: Edge-Level Tasks

Goal: Link Prediction. Given nodes u and v, predict probability $P(A_{uv} = 1)$

- Mechanism: Often formulated as Matrix Completion. The model learns to fill in missing entries of the adjacency matrix.
- Case Study: Recommender Systems
  - *Graph:* Bipartite (Users U Items V) : these are the two types of nodes
  - *Task:* Predict edges between U and V
  - *Key Constraint:* Scalability. Standard GNNs struggle with massive graphs (Pinterest has $3B$ nodes/edges); requires efficient sampling
- Case Study: Polypharmacy
  - *Problem:* Drug-drug interaction side effects.
  - *Graph:* Bio-medical knowledge graph.
  - *Task:* Predict labeled edges ($Drug_A$, $Drug_B$, $type = Hepatotoxicity$)

3: Subgraph-Level Tasks

Goal: Clustering or Community Detection.

- Mechanism: Partitioning the graph G into subgraphs G_1, G_2, ... based on edge density.
- Case Study: Traffic Prediction (DeepMind).
  - *Graph:* Road network
  - *Task:* Predict "Time of Arrival" by analyzing flow through a specific path (subgraph).

4: Graph-Level Tasks

Goal: Classify or generate an entire graph G.

- Mechanism: Pooling node embeddings into a global graph embedding z_G
- Case Study: Drug Discovery
  - *Graph:* Molecule (Nodes = Atoms, Edges = Bonds).
  - *Task:* Binary classification (Toxic / Non-toxic).
  - *Generative Task:* Generate valid molecular graphs that optimize a property (e.g., solubility)

3. Graph Theory Formalisms & Representation

This section covers the mathematical structures used to represent graphs computationally.

A graph is defined as $G = (N, E)$ where:

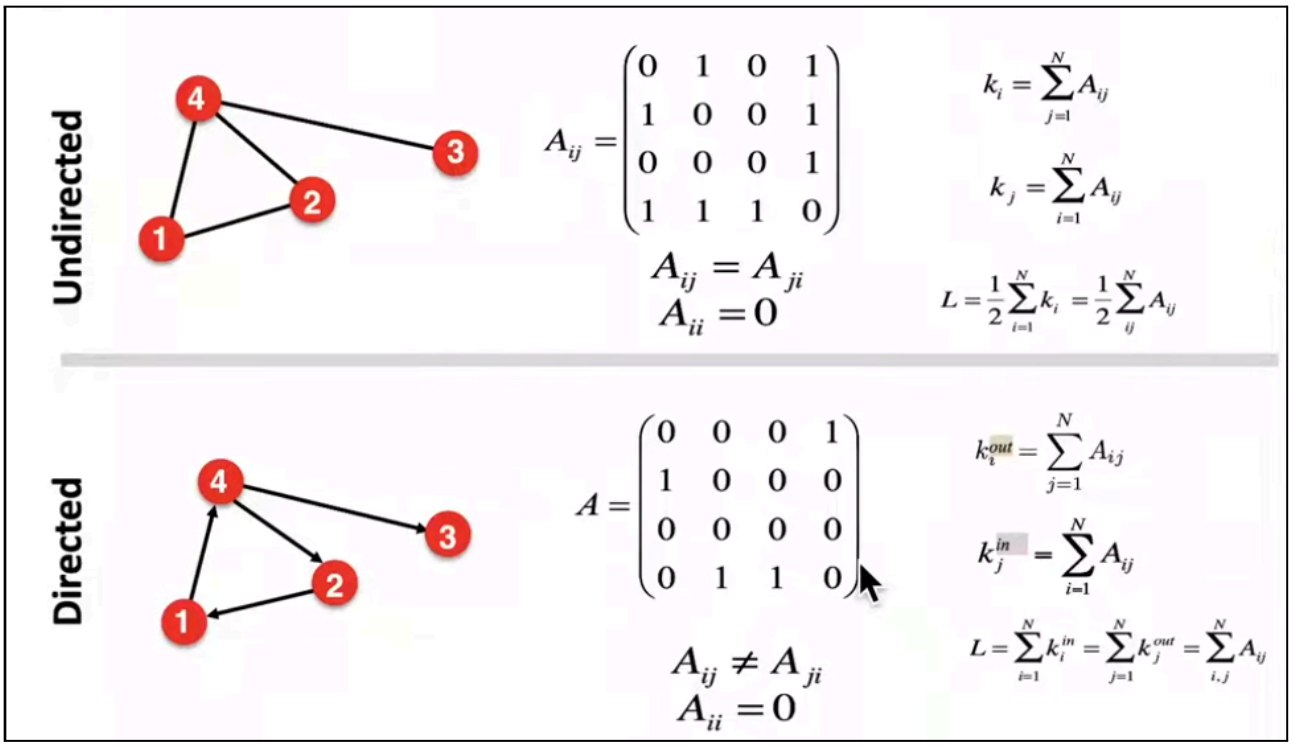
- N: Set of Nodes (Vertices)
- $|N|$ = Number of nodes.
- E: Set of Edges (Links)
- $|E|$ = Number of edges.

Adjacency Matrix (A)

The standard algebraic representation is a square matrix $A \in R^{|N| * |N|}$

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Undirected Graphs: A is Symmetric $A_{ij} = A_{ji}$
- Directed Graphs: A is generally Asymmetric. Directed edges imply: $(i,j) \in E$ *does not imply* $(j,i) \in E$



Node Degrees & Handshaking Lemma

The degree k_i is the number of edges adjacent to node i.

Undirected Case:

- $$k_i = \sum_{j=1}^{|N|} A_{ij}$$
- Average Degree: $\bar{k} = \frac{2|E|}{|N|}$
- *Derivation:* The sum of degrees is $2|E|$ because every edge is counted twice (once for each endpoint).

Directed Case:

We distinguish between incoming and outgoing connections.

- In-Degree (k_{in}): $\sum_j A_{ji}$ (Sum of column i).
- Out-Degree (k_{out}): $\sum_j A_{ij}$ (Sum of row i).
- Average Degree: $\bar{k} = \frac{|E|}{|N|}$
- *Note:* The factor of 2 vanishes because an edge has a specific source and destination.

Bipartite Graphs

A graph where nodes can be decomposed into two disjoint sets U and V such that every edge connects a node in U to one in V. No edges exist within U or within V.

Folded Networks (Projected Graphs):

You can reduce a bipartite graph to a single-mode graph.

- *Example:* Author-Paper graph.
- *Projection:* Author-Author graph. Two authors are connected if they share a neighbor (Paper) in the bipartite graph.

- *Math:* If B is the interaction matrix between sets U and V, the projection for U is often derived from $BB^{\top}$

Connectivity & Topology

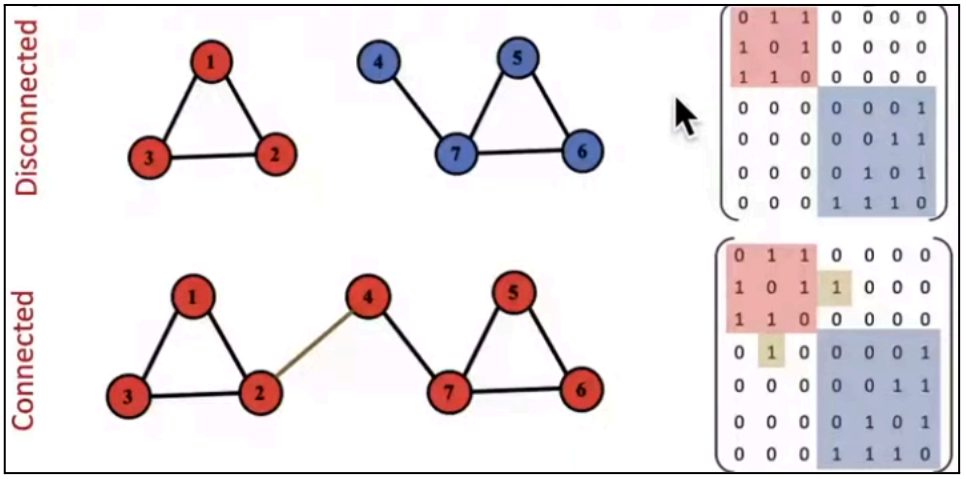
Connectivity describes how information flows through a network. It is defined by the existence of paths (sequences of edges) between nodes.

Undirected Connectivity

- Connected Graph: For any pair of nodes U and V there exists a path connecting them.
- Disconnected Graph: The graph is fragmented into isolated subgraphs called Connected Components.
- NOTE: If you reorder the node IDs of a disconnected graph by their component membership, the Adjacency Matrix A becomes Block Diagonal

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$$

  - Non-zero entries are clustered in square blocks (A_1, A_2) along the diagonal.
  - All entries outside these blocks are strictly 0 (no edges between components)



Directed Connectivity

In directed graphs, edge direction constrains the flow, leading to two definitions:

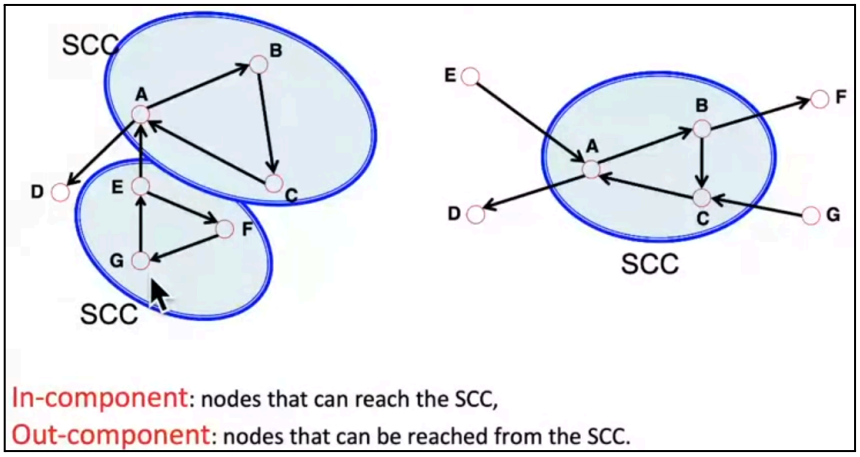
1. Strong Connectivity:
   A pair of nodes (U, V) is strongly connected if there is a directed path $U \to V$ AND a directed path $V \to U$.
   Strongly Connected Component (SCC): A subgraph where *every* node can reach *every other* node within that subgraph.
2. Weak Connectivity:
   A pair (U, V) is weakly connected if a path exists between them when ignoring edge directions (treating the graph as undirected).

Example:

Real-world directed networks (like the World Wide Web) generally exhibit a distinct component structure:

- Giant SCC: The core of the internet (major hubs) where navigation is easy.
- In-Component: Pages that link *into* the SCC but cannot be reached *from* it (e.g., new pages).

- Out-Component: Pages reached *from* the SCC but link nowhere (e.g., "sink" nodes or dead ends)



4. System Design: Handling Sparsity

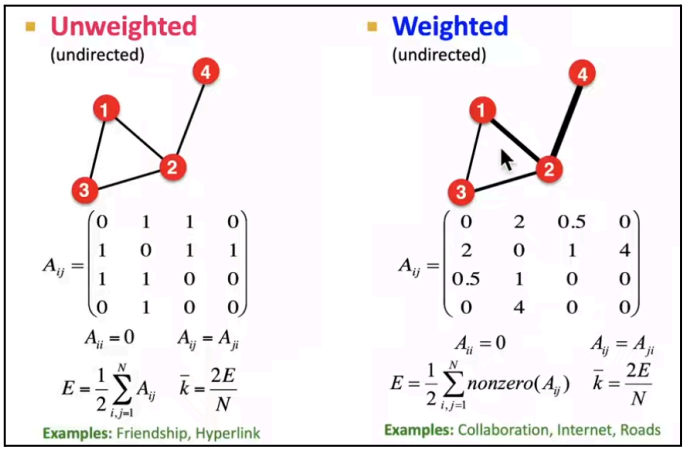
The Sparsity Problem:
In real-world networks (Social, Web, Biology), the graph is extremely sparse.

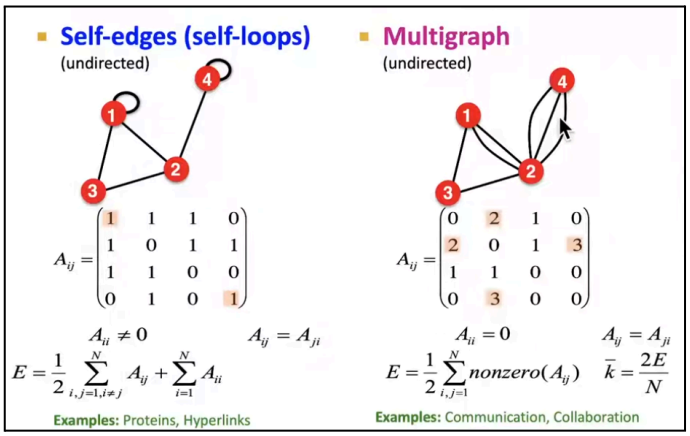
$$\text{Density} = \frac{|E|}{|N|(|N|-1)} \approx 0$$

- *Example:* A social network with $N = 10^9$ might have avg degree $\overline{k} = 100$
- Total edges $E \approx 50 \times 10^9$.
- Matrix Size $N^2 = 10^{18}$.
- Most entries in A are zero.

Representation Choices:

1. Edge List:
    - *Structure:* List of tuples [(u, v), (w, x), ...].
    - *Use:* Simple storage, often inputs for deep learning frameworks.
2. Adjacency List:
    - *Structure:* Dict { Node_ID : [List_of_Neighbors] }
    - *Space:* $O(|E|)$
    - *Use:* Standard for traversal algorithms (BFS/DFS) and most GNN frameworks (PyTorch Geometric uses edge_index tensors in Coordinate Format, effectively a list).
3. Adjacency Matrix:
    - *Space:* $O(N^2)$
    - *Use:* Only for dense graphs or spectral theory derivation. Do not implement this for large graphs.

- 

- 

Note: When analyzing algorithm complexity in Graph ML, always assume the Adjacency List representation. Therefore, operations usually scale with $O(|E|)$ rather than $O(N^2)$