# Lecture 1: Introduction

## 1.1 Why Graphs

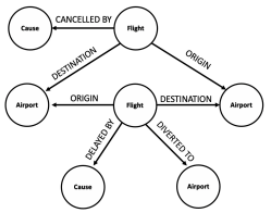
### Perquisite

- Machine Learning
- Algorithms and graph theory
- Probability and statistics

### Graph Machine Learning Tools

1. Pytorch Geometric (PYG)
2. DeepSNAP: Library that assists deep learning on graphs.
3. GraphGym: Platform for designing graph neural Networks

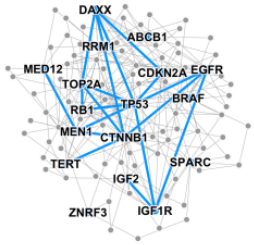
### Many Types of Graph




**Event Graphs**

Image credit: SalientNetworks

**Computer Networks**




**Disease Pathways**

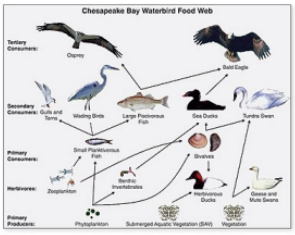



Image credit: Wikipedia

**Food Webs**

Image credit: Pinterest

**Particle Networks**

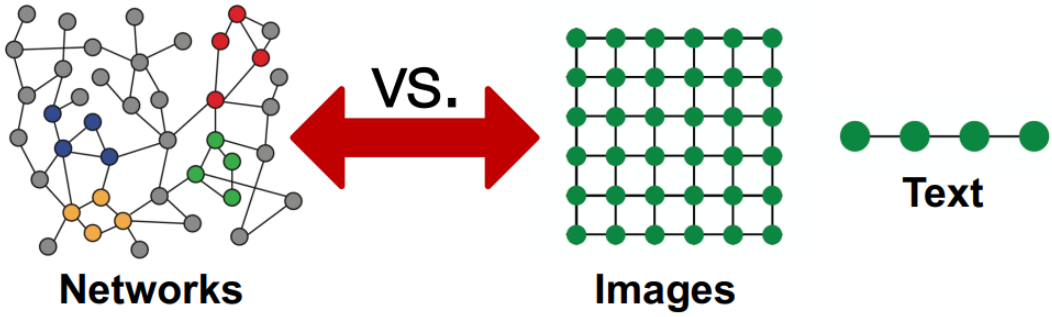
Image credit: visitlondon.com

**Underground Networks**

### Why is it Hard?

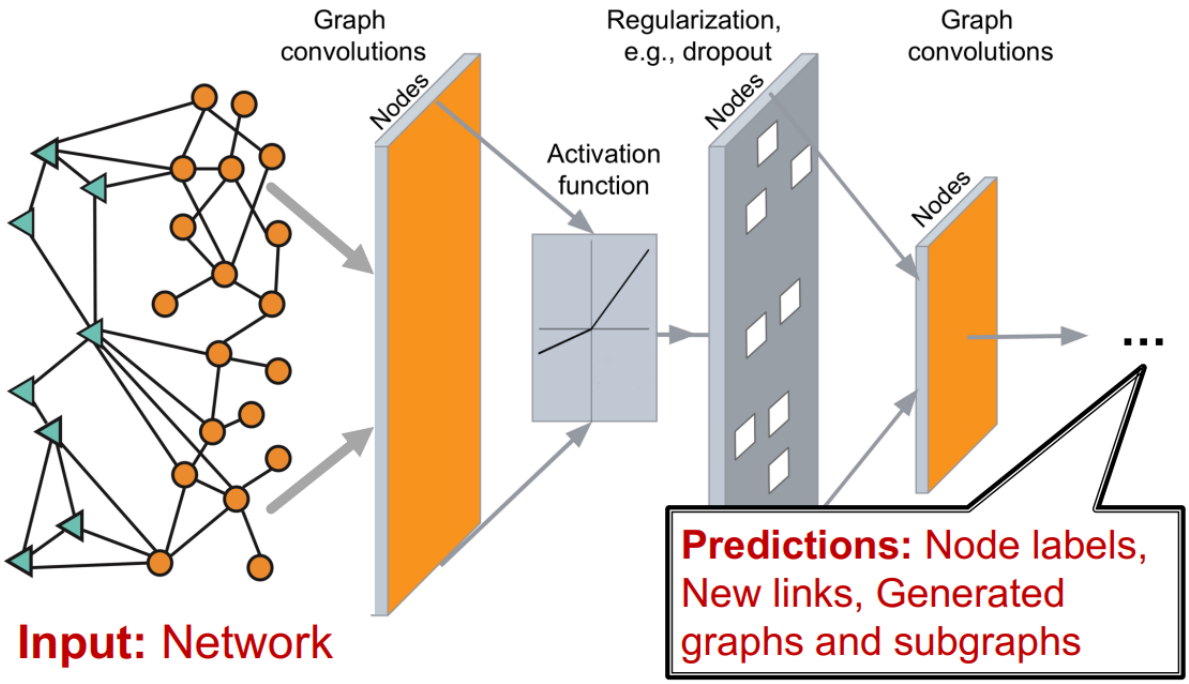
**Networks are complex.**

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



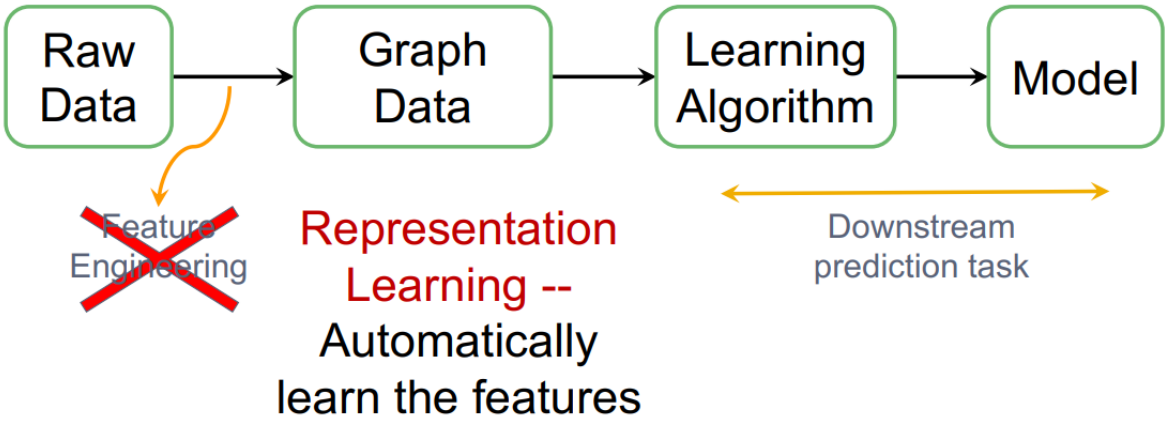

- No fixed node ordering or reference point
- Often dynamic and have multimodal features

## Deep Learning in Graphs



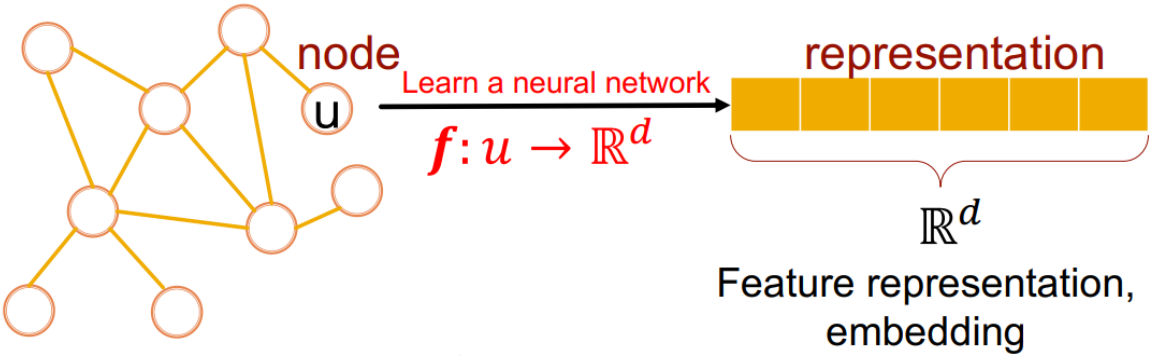

## Supervised Machine Learning Lifecycle

In traditional ML pipeline, we need to apply feature engineering on the raw data, such as manually extracting features. Now we can use representation learning methods to automatically learn this features, and apply them directly to downstream prediction tasks.



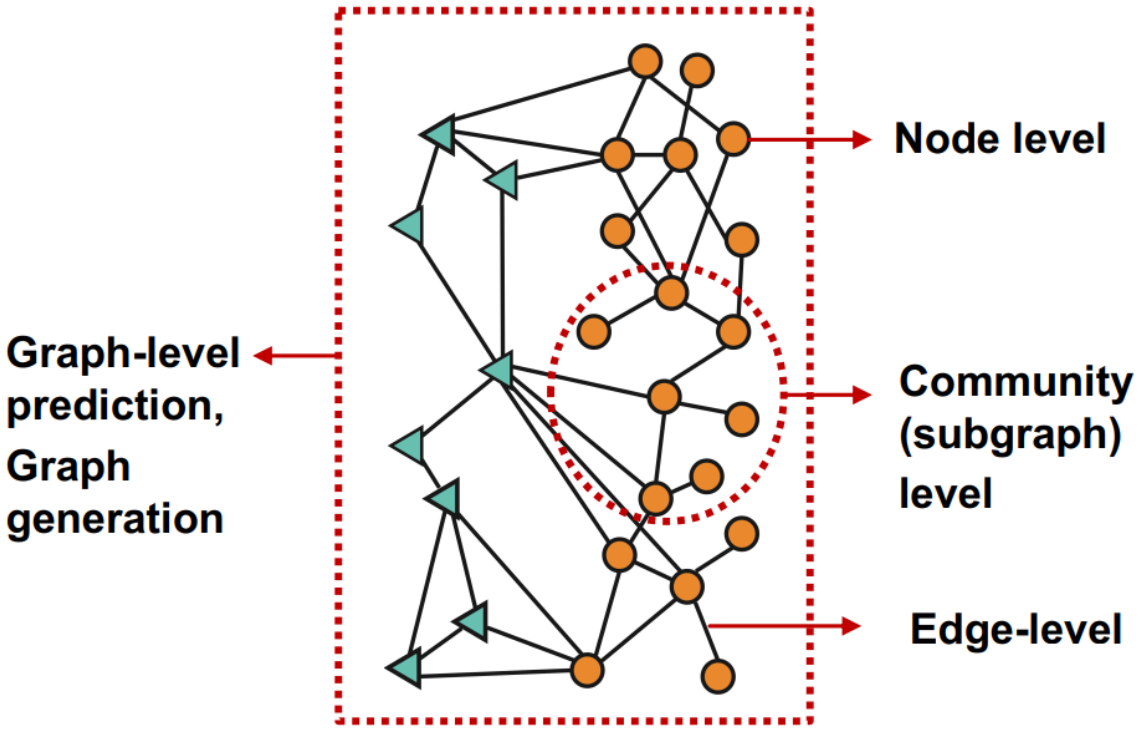

### Representation Learning

Representation Learning aims to map nodes to a d- dimensional **embeddings** such that **similar nodes in the network are embedded close together.**




## 1.2 Applications of Graph ML

Generally, there are 4 main different types of Graph ML tasks:



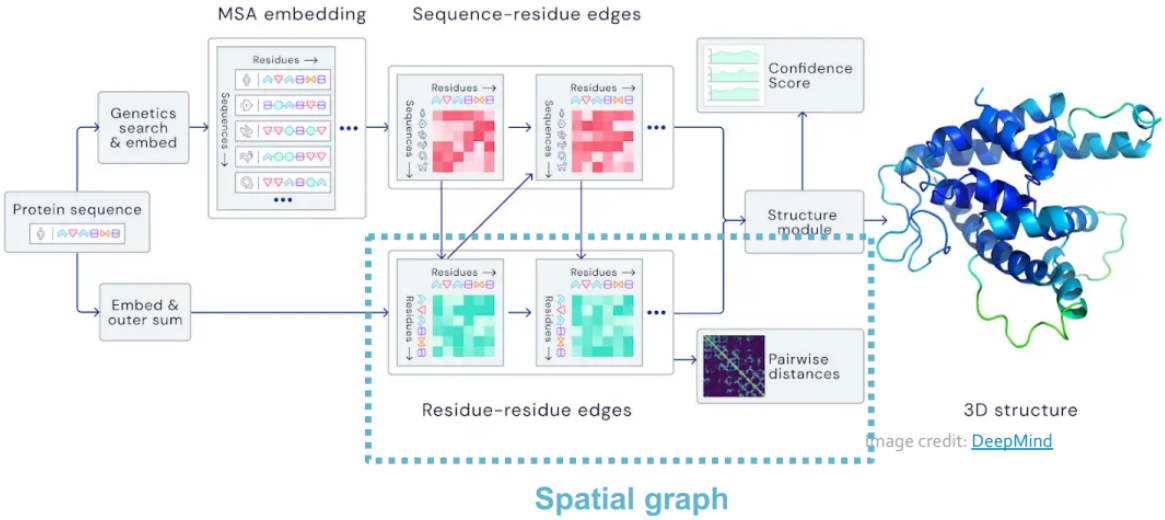

1. Node level

2. Edge level
3. Community/subgraph level
4. Graph level, including graph-level prediction and graph generation

## Example of Node-level ML Tasks

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

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

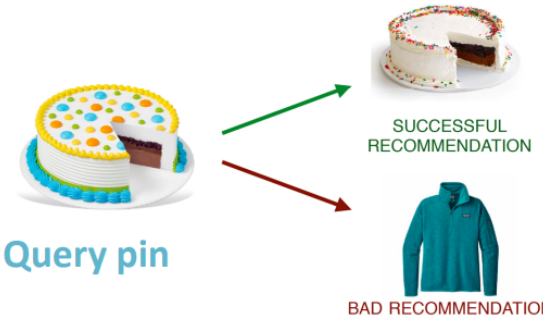



## Example of Edge-level ML Tasks

Recommender Systems: **PinSage**

The task is to recommend items users might like

**Task: Recommend related pins to users**




**Task:** Learn node embeddings $z_i$ such that

$$d(z_{cake1}, z_{cake2}) < d(z_{cake1}, z_{sweater})$$

**Predict whether two nodes in a graph are related**

## Example of Edge-level ML Tasks

Google Map predicts the length and time-consuming of a certain distance: model the road section into a graph, and establish a prediction model on each subgraph.

- **Nodes:** Road segments
- **Edges:** Connectivity between road segments

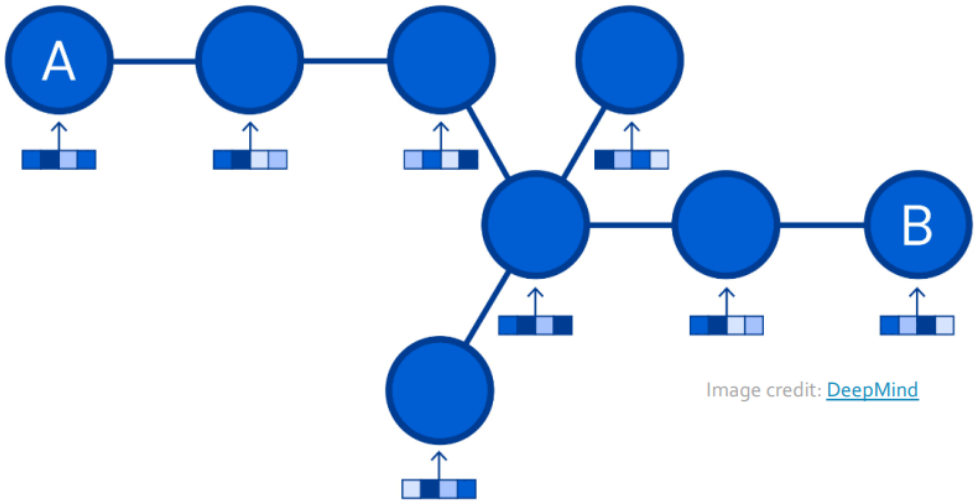
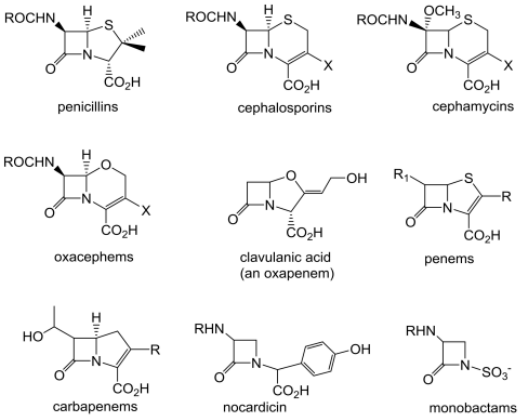



Image credit: DeepMind

### Example of Graph-level ML Tasks

- **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.

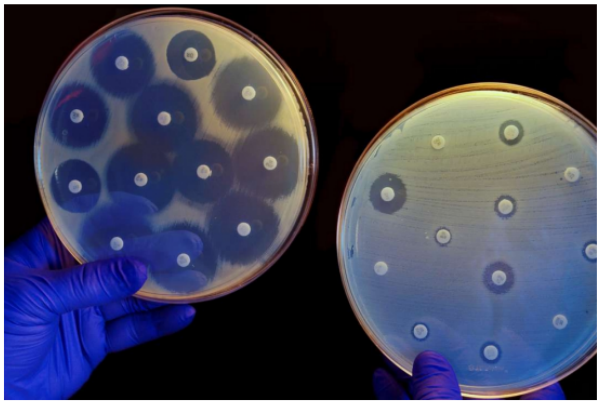
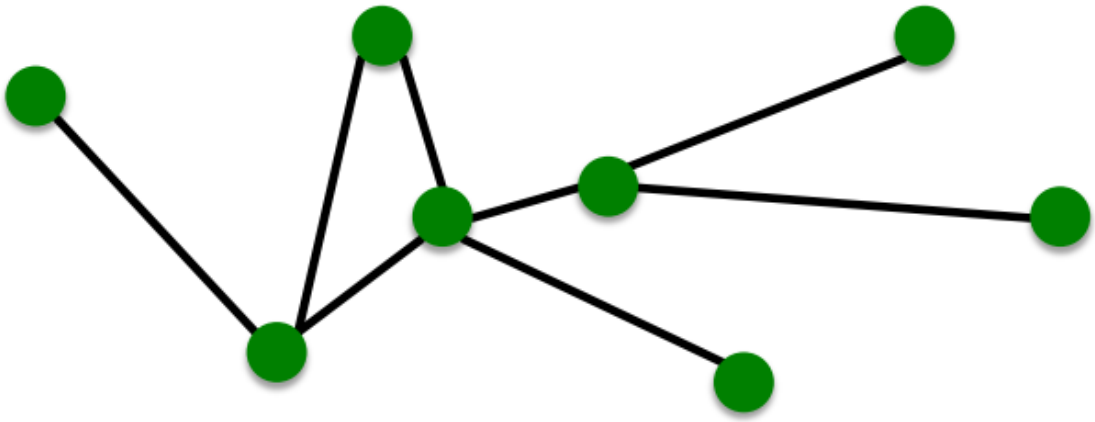


Image credit: CNN

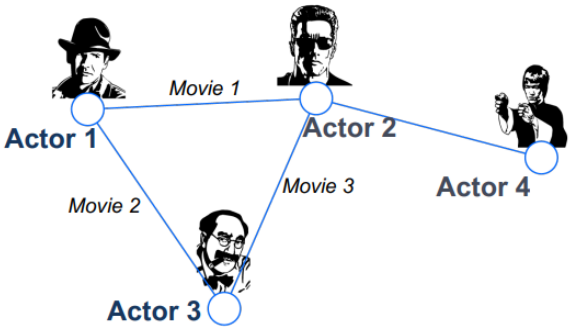
## 1.3 Choice of Graph Representation

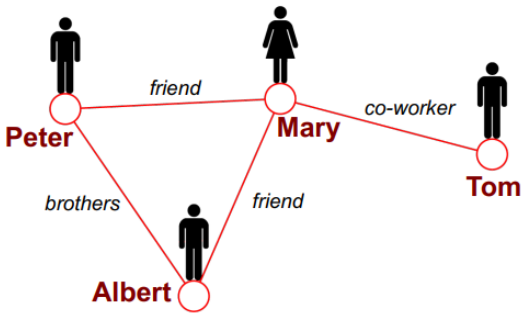
### Component of a Network

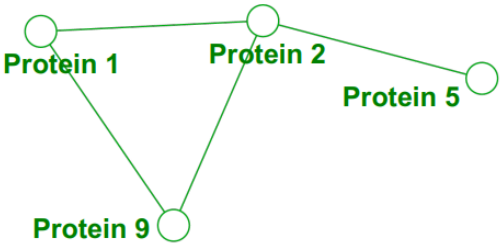
- **Objects:** nodes, vertices **N**
- **Interactions:** links, edges **E**
- **System:** network, graph **G(N,E)**

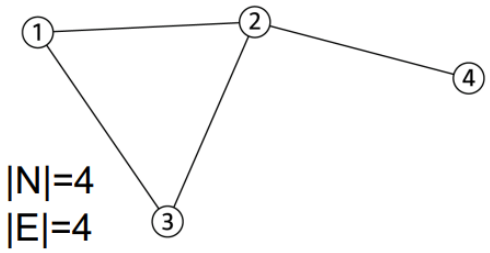


Graphs is a common language for solving relational problems, a unified mathematical representation in various situations. By abstracting the problem into a graph, all problems can be solved with the same machine learning algorithm




## Choosing a Proper Representation

- If you connect individuals that work with each other, you will explore a **professional network**;
- If you connect those that have a sexual relationship, you will be exploring **sexual networks**;
- If you connect scientific papers that cite each other, you will be studying the **citation network**.

## How do you define a graph

- How to build a graph:

  What are nodes?

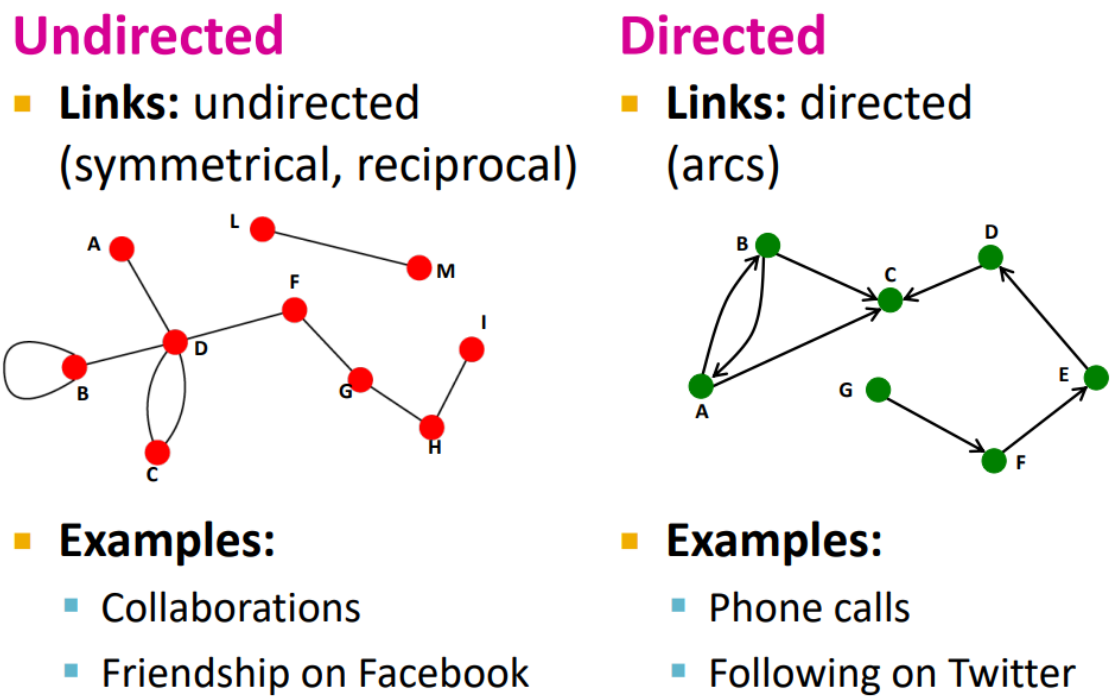
  What are edges?

- Choice the proper representation of a given domain determines our ability to use networks
  1. In some cases there is a unique, unambiguous representation;
  2. In other cases, the representation is by no means unique;

3. The way you assign links will determine the nature of the question you can study.

## Directed vs. Undirected Graphs

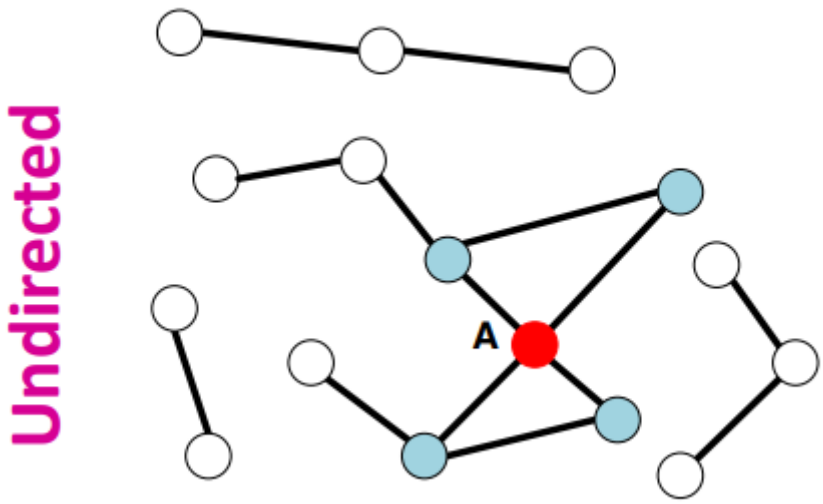
Some examples of designing choices we care faced with when creating graphs



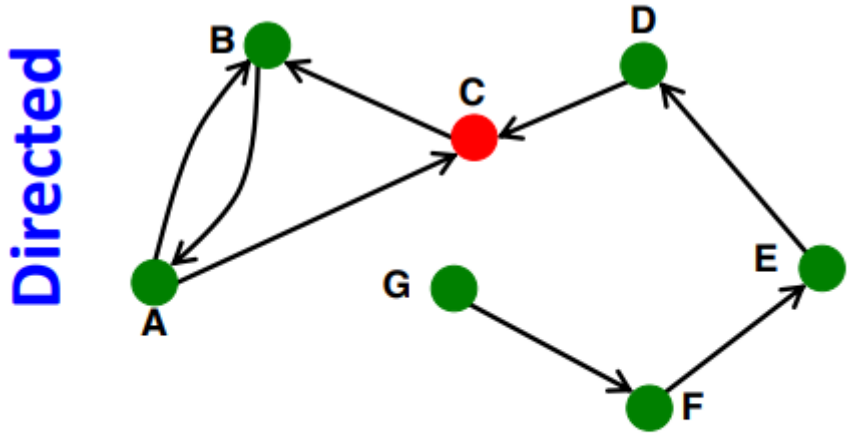

**Node Degree, $k_i$:** the number of edges adjacent to node **i**

**Avg. degree:**

$$\overline{k} = < k > = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$$




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.




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

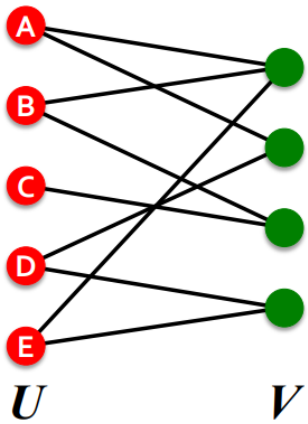
**Sink:** Node with $k^{out} = 0$

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

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

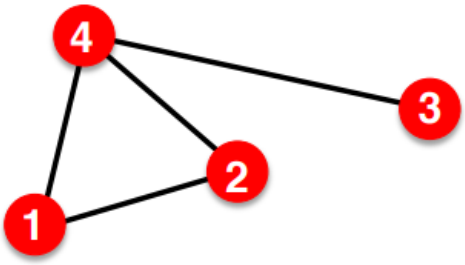
## Bipartite Graph

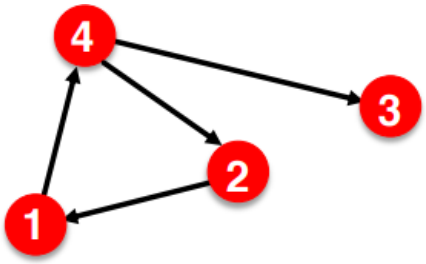
- **Bipartite graph** is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are **independent sets**




- **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

## Adjacency Matrix



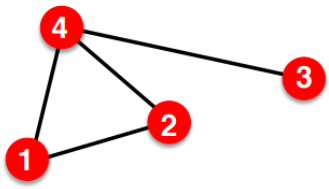




$A_{ij} = 1$ if there is a link from node $i$ to node $j$

$A_{ij} = 0$ otherwise

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

**Undirected**




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

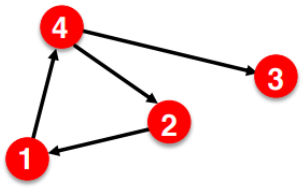
$$A_{ij} = A_{ji}$$
$$A_{ii} = 0$$

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

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

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

**Directed**




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

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

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

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

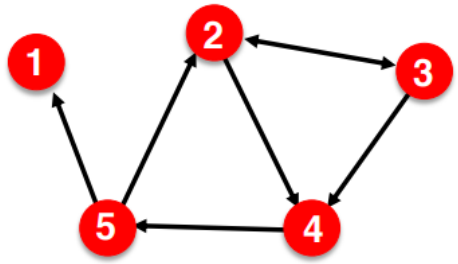
$$L = \sum_{i=1}^{N} k_i^{in} = \sum_{j=1}^{N} k_j^{out} = \sum_{i,j}^{N} A_{ij}$$

Normally, Adjacency matrix are Sparse.

## Representing Graphs

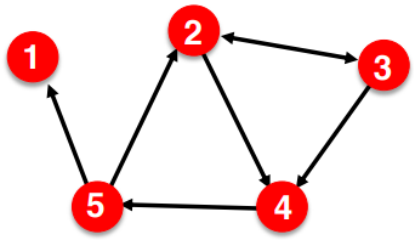
- Edge List: This method is often used in deep learning frameworks because the graph can be directly represented as a two-dimensional matrix. The problem with this representation method is that it is difficult to operate and analyze the graph, even if it is just to calculate the degree of the node in the graph, it will be difficult.

  - (2, 3)
  - (2, 4)
  - (3, 2)
  - (3, 4)
  - (4, 5)
  - (5, 2)
  - (5, 1)




- **Adjacency list:** A much better representation for a graph analysis and manipulation is the notion of 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




## Node and Edge Attributes

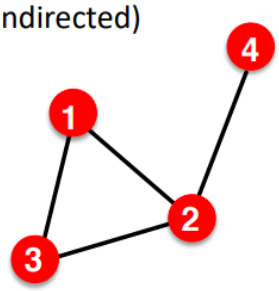
**Possible options:** Not only topological nodes and edges, but also their attributes.

- Weight (e.g. frequency of communication)
- Ranking (best friend, second best friend...) § Type (friend, relative, co-worker)
- Sign: Friend vs. Foe, Trust vs. Distrust
- Properties depending on the structure of the rest of the graph: Number of common friends

## Many Types of Graphs

- **Unweighted**

(undirected)




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

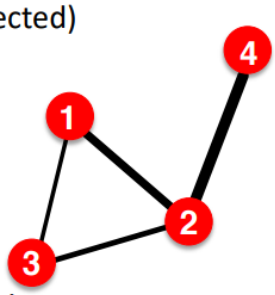
$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$E = \frac{1}{2}\sum_{i,j=1}^{N} A_{ij} \qquad \bar{k} = \frac{2E}{N}$$

**Examples:** Friendship, Hyperlink

- **Weighted**

(undirected)




$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

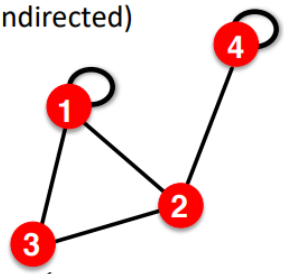
$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$E = \frac{1}{2}\sum_{i,j=1}^{N} nonzero(A_{ij}) \qquad \bar{k} = \frac{2E}{N}$$

**Examples:** Collaboration, Internet, Roads

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

(undirected)




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

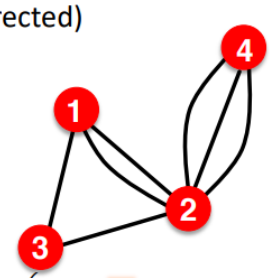
$$A_{ii} \neq 0 \qquad A_{ij} = A_{ji}$$

$$E = \frac{1}{2}\sum_{i,j=1,i\neq j}^{N} A_{ij} + \sum_{i=1}^{N} A_{ii}$$

**Examples:** Proteins, Hyperlinks

- **Multigraph**

(undirected)




$$A_{ij} = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

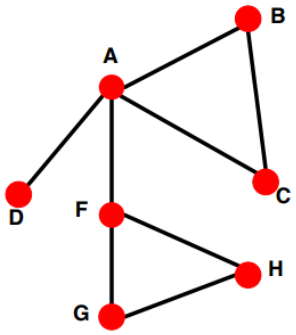
$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

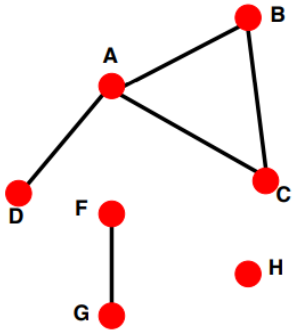
$$E = \frac{1}{2}\sum_{i,j=1}^{N} nonzero(A_{ij}) \qquad \bar{k} = \frac{2E}{N}$$

**Examples:** Communication, Collaboration

## Connectivity of Undirected Graphs

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



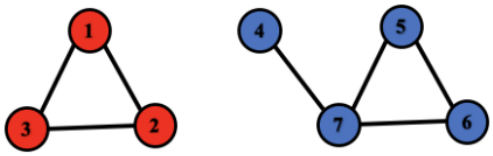




Largest Component:
**Giant Component**

**Isolated node** (node H)

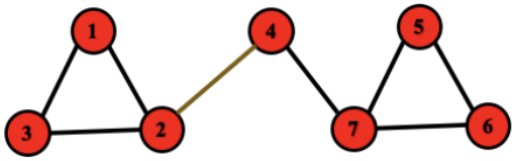
The adjacency matrix of a network with several components can be written in a block- diagonal form, so that nonzero elements are confined to squares, with all other elements being zero:

Disconnected




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

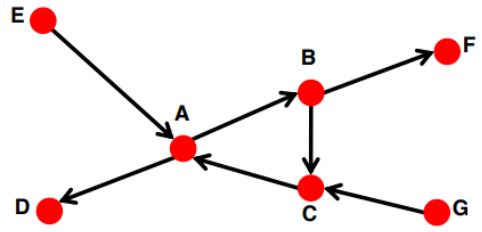
Connected




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

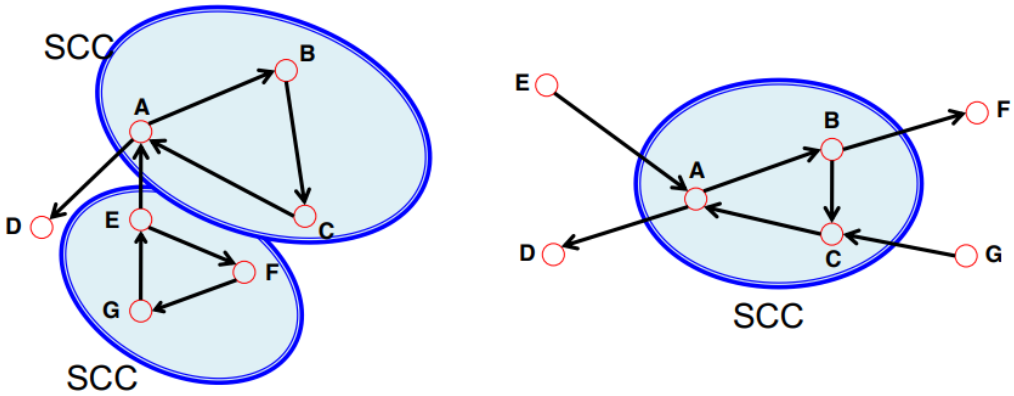
**Connectivity of Directed Graphs**

- **Strongly connected directed graph**
  - has a path from each node to every other node and vice versa (e.g., A-B path and B-A path)
- **Weakly connected directed graph**
  - is connected if we disregard the edge directions




Graph on the left is connected but not strongly connected (e.g., there is no way to get from F to G by following the edge directions).

- **Strongly connected components (SCCs)** can be identified, but not every node is part of a nontrivial strongly connected component.




In-component: nodes that can reach the SCC,
Out-component: nodes that can be reached from the SCC.

## Summary

- Machine Learning with Graphs

  Applications and use cases

- Different Types of Tasks:
  - Node level
  - Edge level
  - Graph level
- Choice of a graph representation
  - Directed, undirected, bipartite, weighted, adjacency matrix