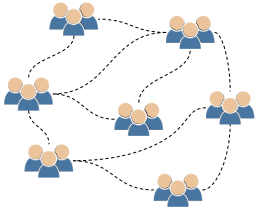
Introduction to Machine Learning on Graphs

APM_5DS30_TP, Machine Learning with Graphs

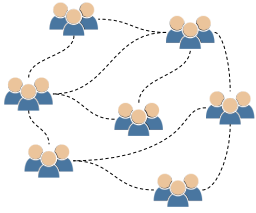
Jhony H. Giraldo (Enseignant-Chercheur).
jhony.giraldo@telecom-paris.fr
October 3, 2025

Motivation
Data on Networks

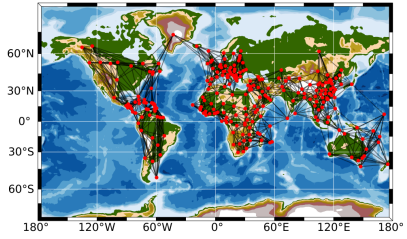
Logical networks
- Social networks
- The web



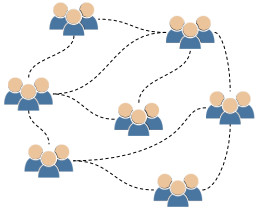
Logical networks
- Social networks
- The web



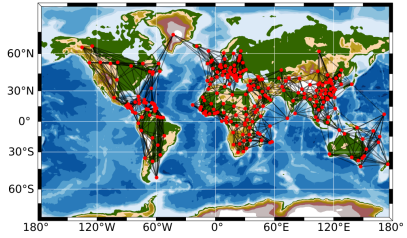
Physical networks
- Sensor
- Internet



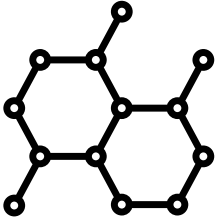
Logical networks
- Social networks
- The web



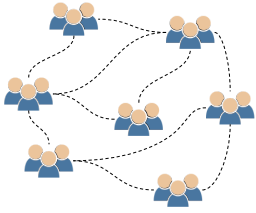
Physical networks
- Sensor
- Internet



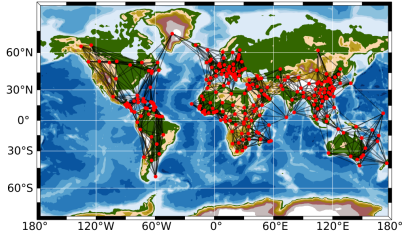
Biology and chemistry
- Molecules
- Proteins



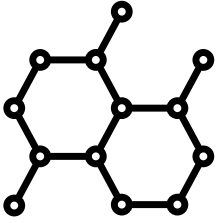
Logical networks
- Social networks
- The web



Physical networks
- Sensor
- Internet

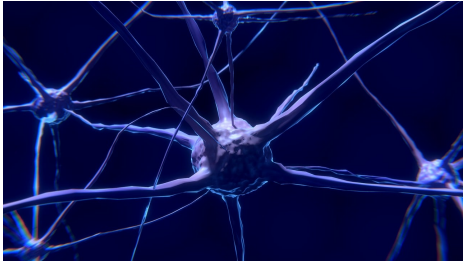


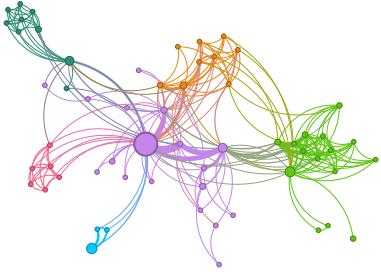
Biology and chemistry
- Molecules
- Proteins



- We need to take into account the structure behind the data!

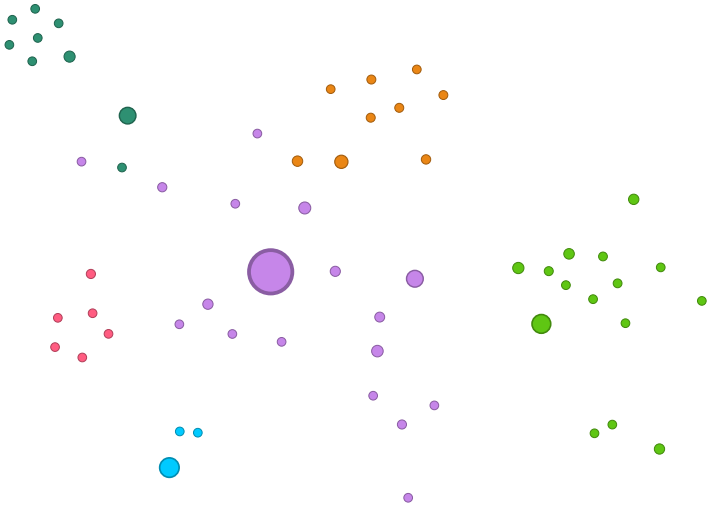
"Deep" Learning + Structure





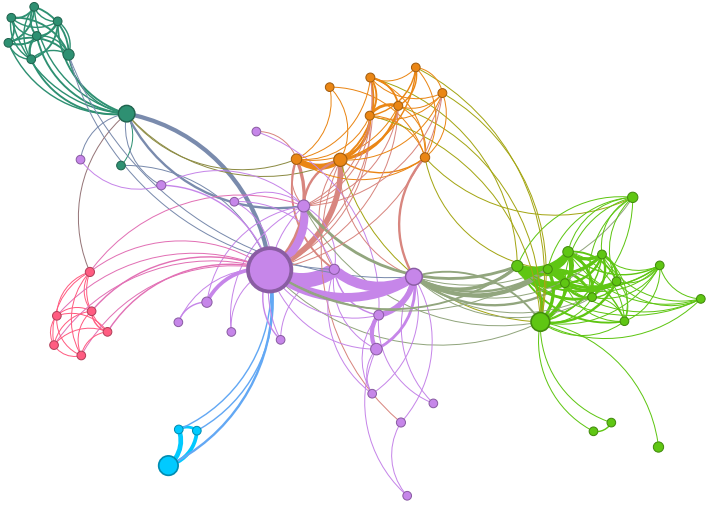
What is a Graph?

Beyond Isolated Data Points



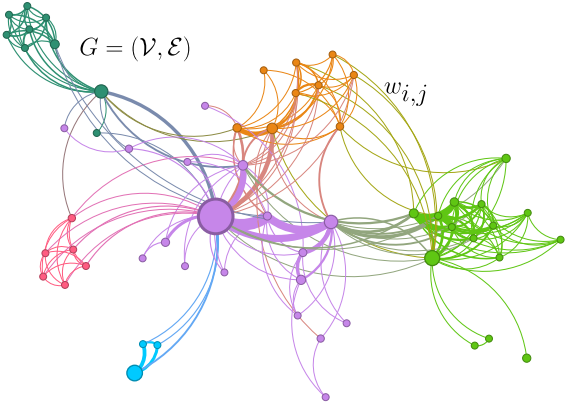
What is a Graph?

Beyond Isolated Data Points



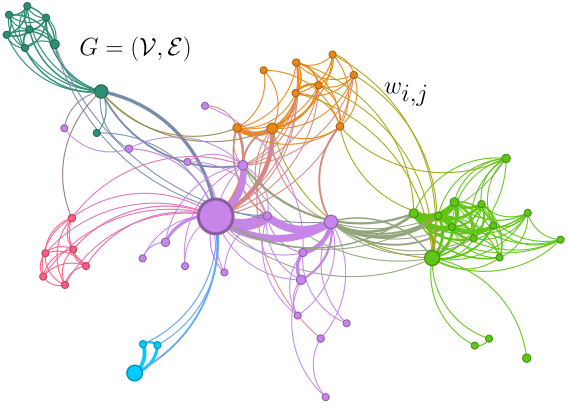
What is a Graph?

- In this course, we will explore machine learning models that extract high-level features from graphs, a process known as graph representation learning.



What is a Graph?

- In this course, we will explore machine learning models that extract high-level features from graphs, a process known as graph representation learning.
- Prerequisites: fundamental knowledge of **linear algebra** and the basics of **deep learning**.



Objectives

- The two objectives of the lectures in machine learning with graphs are:

Applications

Develop the ability to use graph machine learning models in practical applications

Fundamentals

Understand the fundamental properties of the most important graph machine learning models

Objectives

- The two objectives of the lectures in machine learning with graphs are:

Applications

Develop the ability to use graph machine learning models in practical applications

Fundamentals

Understand the fundamental properties of the most important graph machine learning models

- At the end of the course, the students will:
  - Understand fundamental concepts of graph neural networks (GNNs).
  - Understand the challenges to scale-up GNNs, spatiotemporal analysis with graphs, graph generation, and recommender systems with graphs.
  - Be able to use GNNs in some basic applications like semi-supervised learning, molecular property prediction, time-series forecasting, and graph generation.

Content (03 October - 12 December)

1. CM: Introduction to Machine Learning on Graphs (October 03)
2. CM: Introduction to GNNs (October 10)
3. TP: GNNs (October 17)
4. CM: Scaling up GNNs (October 24)
5. CM: Spatiotemporal Analysis with GNNs (November 07)
6. TP: Spatiotemporal Analysis with GNNs (November 14)
7. CM: Recommender Systems using Graphs (November 21)
8. CM: Graph Generation (November 28)
9. TP: Graph Generation (December 05)
10. Exam (December 12)[1]

[1] I might need some people to volunteer to do the exam in some time slots outside the regular slot since it's an oral exam. Maybe the morning.

Course Format and Grading Scheme

- We'll have lectures (CM) of three hours divided into two blocks of one and a half hours each. Lectures 4 and 7 have a guided practical session in the second block, so please bring your laptops for those
- I'll grade two of your TPs picked at random (40% of the final grade).
- I'll occasionally do non-graded quizzes and exercises during the lectures (this could be useful in case you're in trouble at the end)
- The final exam will give 60% of the final grade
- For the final exam, I'll publish a list of recent papers in machine learning on graphs in the best venues. You will pick one of these papers, present it, do some exercises, and write a report. More details will follow soon
- The retake exam is written. If you work, there's no reason to fail

Resources

- Course machine learning with graphs at Stanford (https://web.stanford.edu/class/cs224w/)
- Course graph neural networks at Upenn (https://gnn.seas.upenn.edu/)
- Book: Graph Representation Learning by William L. Hamilton (https://www.cs.mcgill.ca/~wlh/grl_book/)

Expectations and Policies

- Submit the TPs on time. Please be sure about this (double check), the penalty for being late follows an exponential function
- Limit the use of cell phones during lectures for non-academic purposes
- Participation in the lectures is important
- Academic integrity: plagiarism, cheating, and misuse of LLMs will be penalized
- Examples of misuse of LLMs:
  - Asking an LLM to do your whole TP without understanding what's going on
  - Hallucinations in your report or TPs will be heavily penalized; you are responsible for what you write

Disclaimer and Internship Positions (M2)

- I occasionally use images from the course Machine Learning with Graphs at Stanford (`https://web.stanford.edu/class/cs224w/`)
- I occasionally use images from my papers

Disclaimer and Internship Positions (M2)

- I occasionally use images from the course Machine Learning with Graphs at Stanford (`https://web.stanford.edu/class/cs224w/`)
- I occasionally use images from my papers
- If you're interested in doing your M2 internship on geometric deep learning, please contact me. PhD positions might be available for strong candidates, depending on funding availability. Topics of interest are:
  - Spatiotemporal analysis with machine learning on graphs (forecasting and imputation)
  - Topological deep learning models
  - Graph/hypergraphs generation
  - Vision graph neural networks

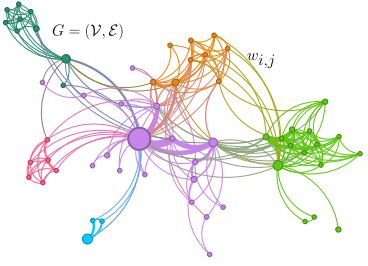
Outline

Definition

A **graph** is a mathematical structure that allows modeling pairwise relations or interactions between entities.

Denoting $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

- set of **nodes** $\mathcal{V}$
- set of **edges** $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$

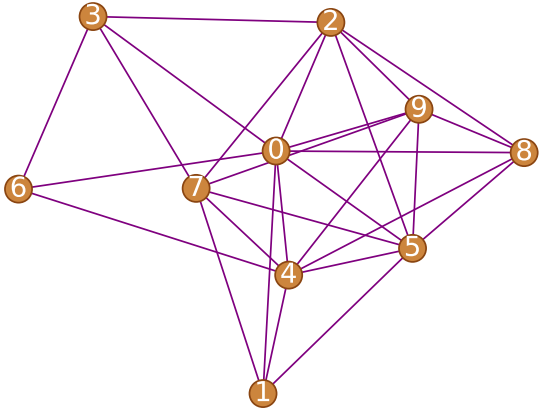


$(u, v) \in \mathcal{E}$ indicates the existence of an edge between nodes u and v.

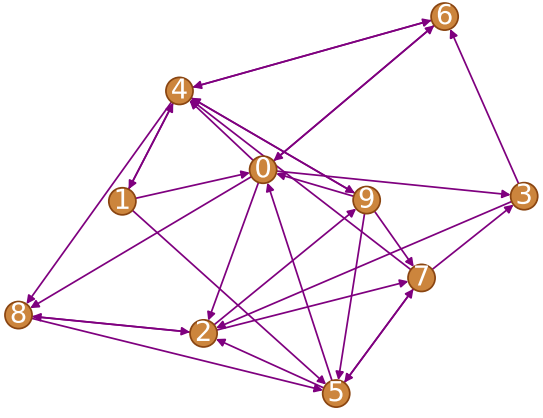
Basic Definitions

Example of a Graph

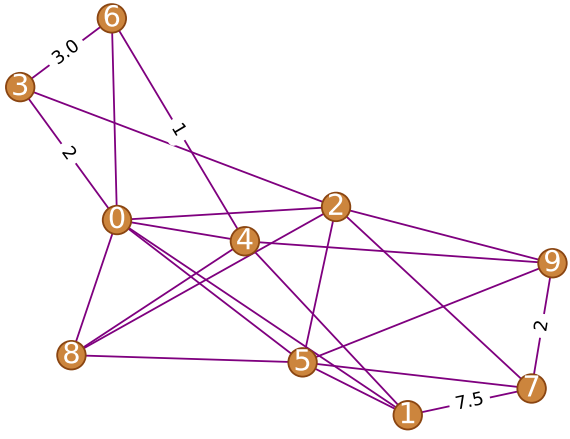
- $\mathcal{V} = \{0, 1, \cdots, 9\}$
- $N = 10$
- $\mathcal{E} = \{(0, 1), (0, 2), \cdots, (9, 5), (9, 8)\}$



Directed graphs: edges are directed (not symmetrical)



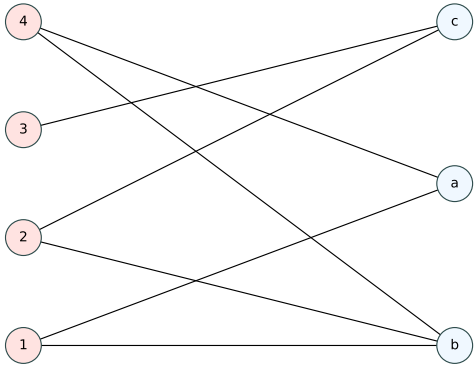
Weighted Graphs: edge attribute in the form of $W = w_e, \forall e \in \mathcal{E}$



Vertices can be decomposed into two disjoint and independent sets

$$\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$$

Example: $\mathcal{U} = \{1, 2, 3, 4\}, \quad \mathcal{V} = \{a, b, c\}$



Machine Learning on Graphs

- Yes! Nowadays, there are many successful applications of machine learning on graphs in various fields

- Yes! Nowadays, there are many successful applications of machine learning on graphs in various fields
  - Physics, particles simulation [Sanchez-Gonzalez et al.(2020)].
  - Robotics, learning decentralized controllers in collaborative autonomy [Tolstaya et al.(2020)].
  - Biology, AlphaFold [Jumper et al.(2021)].
  - Climate, GraphCast [Lam et al.(2022)], GenCast [Price et al.(2023)].

- Yes! Nowadays, there are many successful applications of machine learning on graphs in various fields
  - Physics, particles simulation [Sanchez-Gonzalez et al.(2020)].
  - Robotics, learning decentralized controllers in collaborative autonomy [Tolstaya et al.(2020)].
  - Biology, AlphaFold [Jumper et al.(2021)].
  - Climate, GraphCast [Lam et al.(2022)], GenCast [Price et al.(2023)].
- For you as students, there are not so many people who know machine learning. There are even fewer people who know machine learning on graphs.

Complex domains have a rich relational structure, which can be represented as a relational graph

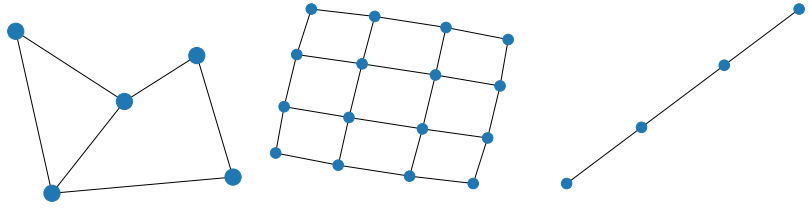
By explicitly modeling relationships, we achieve better performance (prediction, recommendation etc.)

Complex domains have a rich relational structure, which can be represented as a relational graph

By explicitly modeling relationships, we achieve better performance (prediction, recommendation etc.)

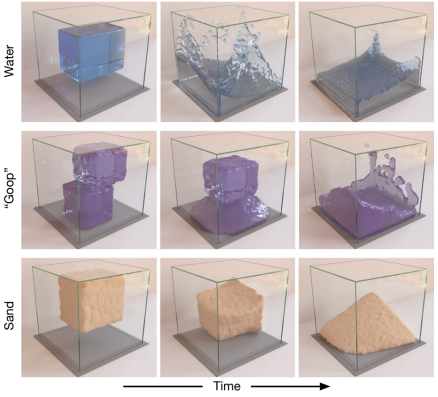
Modern machine learning toolboxes are designed for **sequences** (like audio or text) or **regular grids** (like images)!

- Arbitrary size and complex topological structure (*i.e.*, no spatial locality - left/right/up/down)



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

- Learning to simulate complex physics with GNNs. [Video] [Video]
- Each particle is simulated as a node in the graph

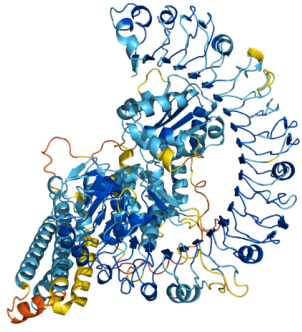


Simulation of complex physics with GNNs. **Source**: [Sanchez-Gonzalez et al.(2020)]

- Learning decentralized controllers in collaborative autonomy. [Video]
- Each drone is simulated as a node in the graph [Tolstaya et al.(2020)]

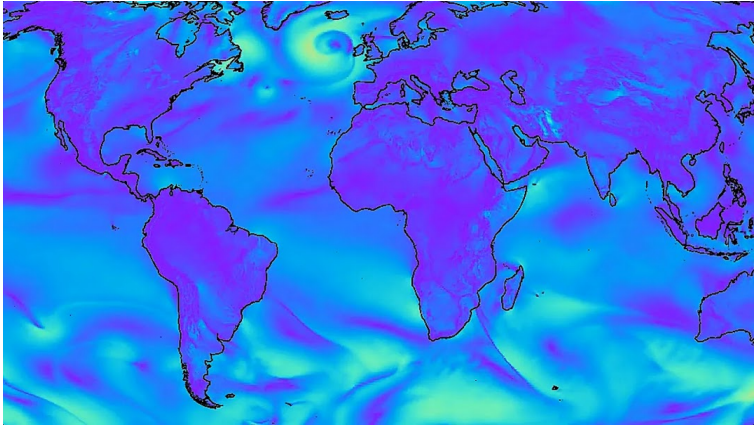
Distributed controllers for large networks of mobile robots.

- Prediction of the three-dimensional structure that a protein will adopt based solely on its amino acid sequence [Video]



Protein folding prediction. **Source**: DeepMind

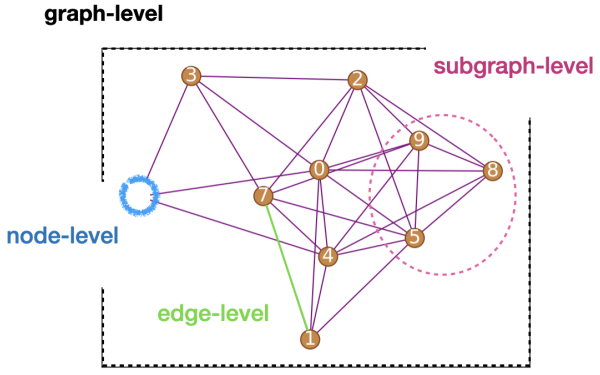
- Weather forecasting using graph neural networks [Video]



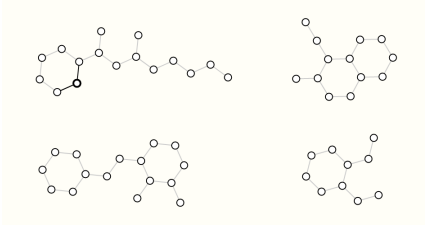
Weather forecasting. **Source**: DeepMind

Graph Machine Learning Tasks

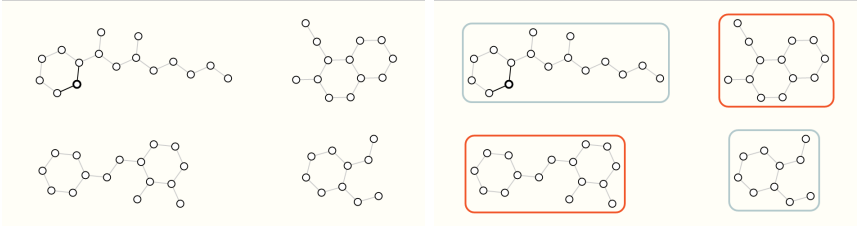
- Graph-level: Predict a property for a whole graph
- Node-level: Predict a property for each node in the graph
- Edge-level: Predict the property/presence of edges in a graph.



Goal: Predict a label for the whole graph
Example: Does the graph contain 2 rings?



Goal: Predict a label for the whole graph
Example: Does the graph contain 2 rings?



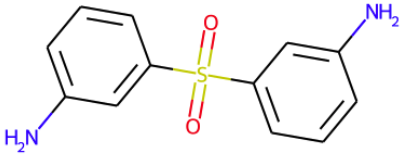
Taken from distill.pub

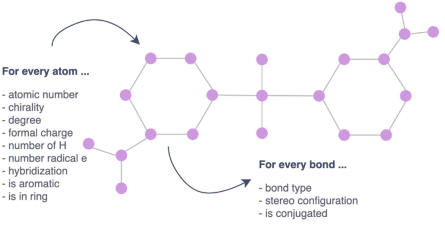
Graph Machine Learning Tasks

Example: Predicting the Toxicity of a Molecule

Input: A molecule

Output: A binary label - Yes if the molecule is toxic, 0 otherwise.





DeepTox: Toxicity Prediction using Deep Learning

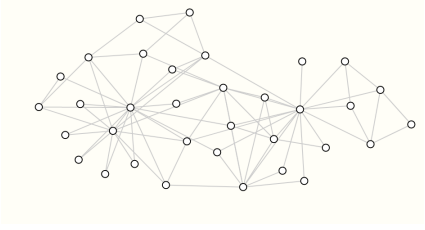
Andreas Mayr[1,2†], Günter Klambauer[1†], Thomas Unterthiner[1,2†] and Sepp Hochreiter[1*]

[1] Institute of Bioinformatics, Johannes Kepler University Linz, Linz, Austria
[2] RISC Software GmbH, Johannes Kepler University Linz, Hagenberg, Austria

Goal: Predict a label for each node of the graph

Example: Karate club

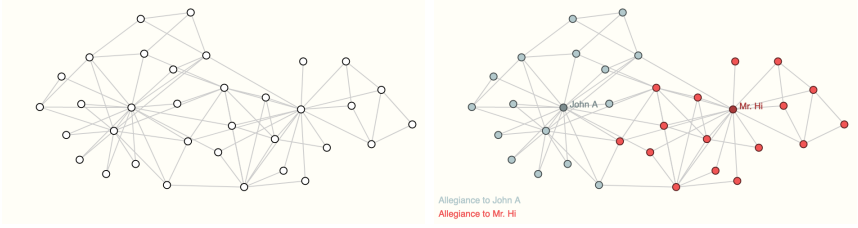
- Nodes represent karate practitioners
- Edges are interactions outside of karate
- Node label: Who are you loyal to?



Goal: Predict a label for each node of the graph

Example: Karate club

- Nodes represent karate practitioners
- Edges are interactions outside of karate
- Node label: Who are you loyal to?



Taken from distill.pub

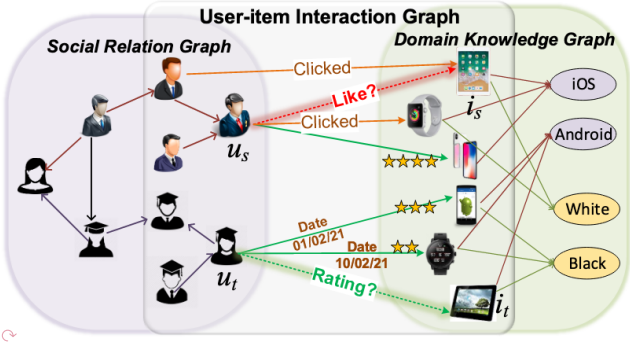
Graph Machine Learning Tasks

Edge-level: Edge Prediction

Goal: Predict the existence of an edge given a pair of nodes

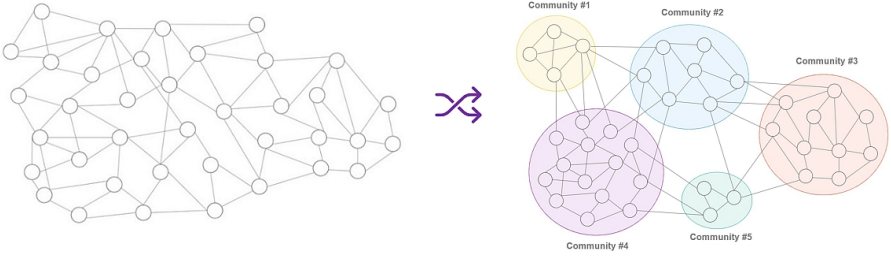
Example: Recommender systems

- Nodes are users and items (*e.g.* products, music, movie)
- Edges are user-item interactions (*e.g.* clicks, like, ratings)
- Task: Recommend relevant items to users

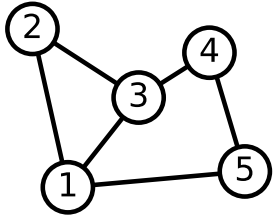


Goal: Find subgroups (or communities) of *densely connected* nodes

- Identification of criminal user groups
- Influences of political ideologies
- Customer segmentation



Choice of representation



- **Objects**: Nodes, vertices $\mathcal{V}$
- **Interactions**: Links, edges $\mathcal{E}$
- **System:** Network, graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

Graphs: A Universal Language

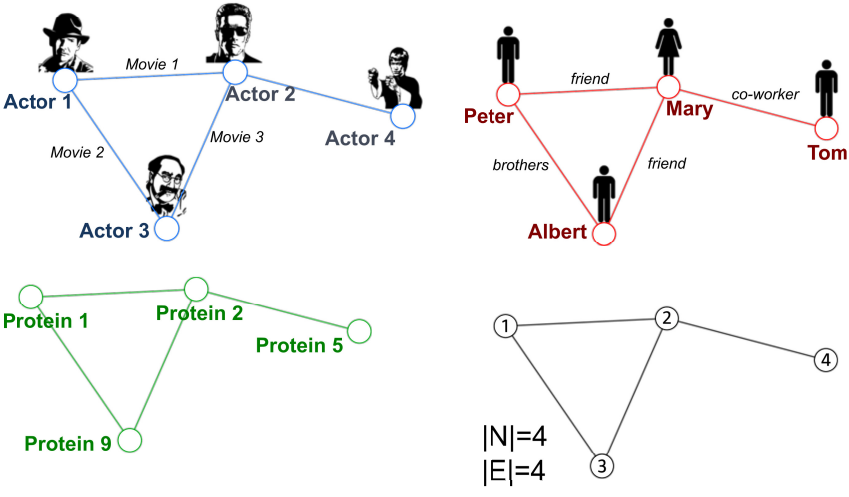


Image from Jure Leskovec course at Standford.

Definition of the domain

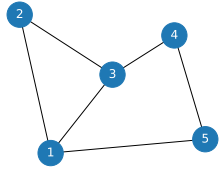
How to build a graph?

- What are nodes?
- What are edges?

> The way we define our domain strongly impacts our ability to use networks and machine learning successfully

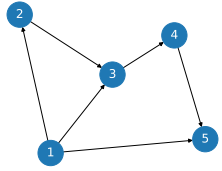
- For natural graphs, the choice of representation is unique
- The way we create links (define closeness) is crucial

Node Degrees



Undirected graphs

Node degree k_i: Number of edges adjacent to i
Example: $k_4 = 2$
Average degree $\bar{k} = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$



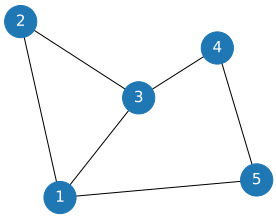
Directed graphs

In degree k_i^{in}: Number of incoming edges
Out degree k_i^{out}: Number of outcoming edges
Degree $k_i^{in} + k_i^{out}$
Average degree $\bar{k} = \frac{E}{N}$

Representation of a Graph: Adjacency Matrix

Adjacency matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Laplacian matrix

$$\mathbf{L} = \mathbf{D} - \mathbf{A},$$

where $\mathbf{D}$ is the diagonal degree matrix.
Diagonal elements of $\mathbf{L}$ capture the importance of nodes (degree), other entries capture the connectivity of the graph.

Graph Spectrum

The eigenvalues and associated eigenvectors of the adjacency matrix $\mathbf{A}$ or the laplacian matrix $\mathbf{L}$ define the **graph spectrum**.

The graph spectrum provides rich information about the properties of a graph[a]

[a] see https://cs.uwaterloo.ca/~lapchi/cs860-2025/ASGT.pdf

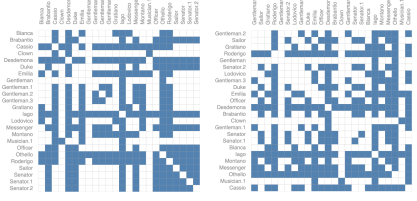
Example - Spectral Characterization of Connected Graphs. Let $\mathcal{G}$ be an undirected graph and let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of its Laplacian matrix $\mathbf{L}$. Then $\mathcal{G}$ is a connected graph if and only if $\lambda_2 > 0$.

Adjacency Matrix Drawbacks

1. Graphs have relatively few connections per node → adjacency matrices are **sparse** → **space inefficient**

					Edges per node (degree)		
Dataset	**Domain**	**graphs**	**nodes**	**edges**	**min**	**mean**	**max**
karate club	Social network	1	34	78		4.5	17
qm9	Small molecules	134k	≤ 9	≤26	1	2	5
Cora	Citation network	1	23,166	91,500	1	7.8	379
Wikipedia links, English	Knowledge graph	1	12M	378M		62.24	1M

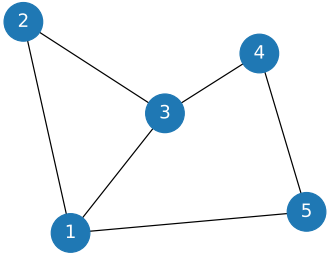
2. Adjacency matrices are not permutation invariant *i.e*, different matrices can produce the same graph



Graph Representation: Edge List and Adjacency List

List of edges:
$(1, 2)$ - (1,3) - (1, 5)
(2, 3) - (3, 4) - (4, 5)

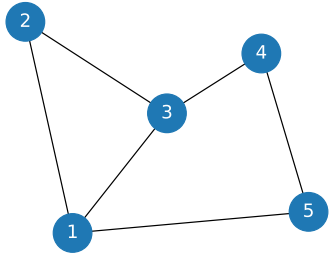
Adjacency list:
1: 2, 3, 5
2: 1, 3
3: 1, 2, 4
4: 3, 5
5: 1, 4



Graph Representation: Edge List and Adjacency List

List of edges:
$(1, 2)$ - (1,3) - (1, 5)
(2, 3) - (3, 4) - (4, 5)

Adjacency list:
1: 2, 3, 5
2: 1, 3
3: 1, 2, 4
4: 3, 5
5: 1, 4

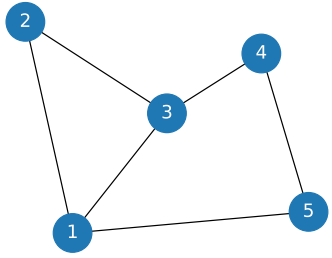


\+ **List of edges**: Easier to work with if the graph is large and sparse.

Graph Representation: Edge List and Adjacency List

List of edges:
$(1, 2)$ - (1,3) - (1, 5)
(2, 3) - (3, 4) - (4, 5)

Adjacency list:
1: 2, 3, 5
2: 1, 3
3: 1, 2, 4
4: 3, 5
5: 1, 4



+ **List of edges**: Easier to work with if the graph is large and sparse.
+ **Adjacency list**: Easy to retrieve all neigbors of a given node.

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Examples

- Weight on edges (frequency of communication)

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Examples

- Weight on edges (frequency of communication)
- Ranking (best friend, second best friend...)

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Examples

- Weight on edges (frequency of communication)
- Ranking (best friend, second best friend...)
- Type of relation (friend, relative, co-worker)

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Examples

- Weight on edges (frequency of communication)
- Ranking (best friend, second best friend...)
- Type of relation (friend, relative, co-worker)
- Sign (friend vs. enemy, trust vs. distrust)

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Examples

- Weight on edges (frequency of communication)
- Ranking (best friend, second best friend...)
- Type of relation (friend, relative, co-worker)
- Sign (friend vs. enemy, trust vs. distrust)
- Properties depending on the structure of the rest of the graph (number of common friends)

Nodes and Edges Attributes

Attributed graph: Nodes and/or edges can have attributes

Examples

- Weight on edges (frequency of communication)
- Ranking (best friend, second best friend...)
- Type of relation (friend, relative, co-worker)
- Sign (friend vs. enemy, trust vs. distrust)
- Properties depending on the structure of the rest of the graph (number of common friends)
- Node features (gender, city)

Wooclap

Answers

- Imagine we have a graph with $N = 25'000,000$ and we use a full adjacency matrix to represent this graph
- If we represent each number of the matrix as a double tensor (in PyTorch for example), what's going to be the total size of the matrix?
- Hint: we use a 64-bit floating point to represent a double tensor in Pytorch
- Answer: $64/8 = 8$ bytes per number, so $8 \times N^2 = 8 \times 25'000,000^2 = 5e + 15$, *i.e.*, 5 Petabytes of memory RAM!
- The fastest computer on earth (in the US) has 4 Petabytes of memory RAM

Answers

- Let's do again the computation for the big graph with a list of edges:
- Let's assume that each node in that big graph has 300 neighbors (you can think of a social network of $25'000,000$ people, where each person has 300 connections on average, is this reasonable to you? Think on your own social networks)
- 8 (bytes) $\times N \times 300 \times 2 = 8 \times 25'000,000 \times 300 \times 2 = 120e+9$, *i.e.*, 120 Gigabytes of memory RAM. The 2 is because we need two numbers to represent each edge

Extracting Node Features

Machine Learning Pipeline

- ML algorithms take vectors as input data
- Need to extract/learn node, edge or graph features

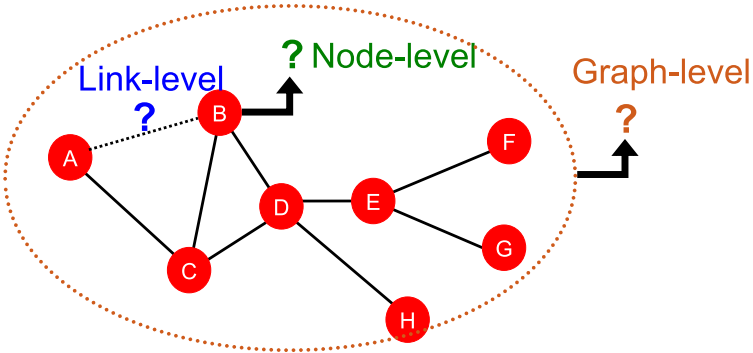
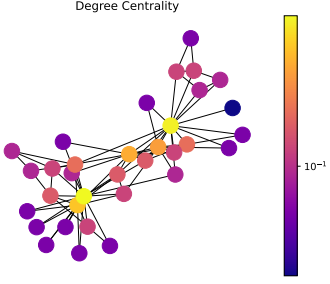


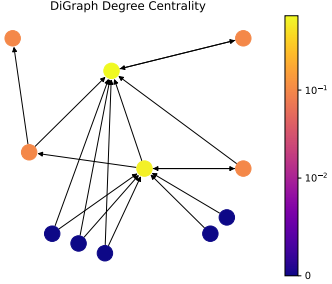
Image from Jure Leskovec course at Standford.

Node-level Features?

Node degree is a good start

Intuition: Nodes with more connections are more influential and more important

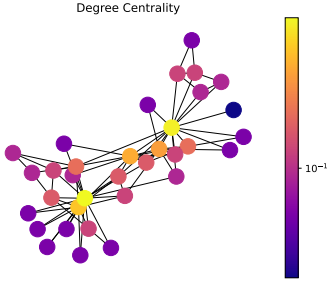


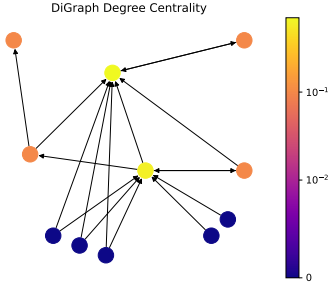


Node-level Features?

Node degree is a good start

Intuition: Nodes with more connections are more influential and more important





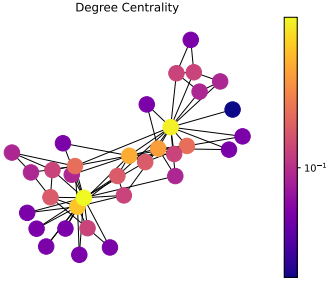
Is it enough?

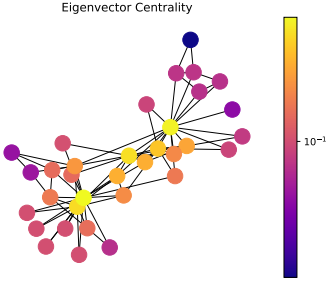
Node Centrality: Eigenvector Centrality

Takes into account two aspects

- How important is a node?
- How important are the neighbors of a node?

Intuition: The importance of a node is proportional to its neighbors importance





Definition of Eigenvector Centrality

- Proposed by E. Landau in the 19th Century
- Proposed to rank professional chess players
- Key ingredients of **PageRank** and HITS

Eigenvector Centrality Definition

- Let λ be the largest eigenvalue of $\mathbf{A}$ with eigenvector $\mathbf{v}$

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

- Scale v so that $\max_i \quad v_i = 1$
- The **eigenvector centrality** of a node i is the entry v_i.

Intuition Behind Eigenvector Centrality

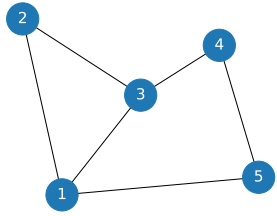
The measure calculates node power based on the power of its neighbors

1. Assume each node is equal, with a power of 1
2. Repeat: Update node power by summing the power of its neighbors until convergence
3. Scale the final vector to bound the max to 1

Eigenvector Centrality: Illustration

new power $\leftarrow$ sum of power of my neighbors

Node	init.	iter1	iter2	iter3
1	1	3	7	18
2	1	2	6	14
3	1	3	7	18
4	1	2	5	12
5	1	2	5	12

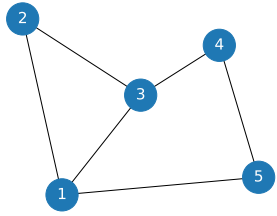


After scaling, we have that $c = [1, 0.78, 1, 0.67, 0.67]$

Eigenvector Centrality: Illustration

new power $\leftarrow$ sum of power of my neighbors

Node	init.	iter1	iter2	iter3
1	1	3	7	18
2	1	2	6	14
3	1	3	7	18
4	1	2	5	12
5	1	2	5	12



After scaling, we have that $c = [1, 0.78, 1, 0.67, 0.67]$

The **update rule** is given by $\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1}$

Connection to largest eigenvalues relies on the **Perron-Frobenius** theorem - Check the great explanation given by Andrew Beveridge

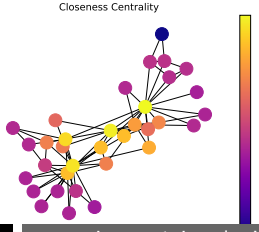
Closeness Centrality

Intuition: A node is important if it's close to all other nodes. Example: In the rail network of France, Paris has a high closeness centrality since you can reach every place quickly.

Definition of closeness centrality

Let d_{ij} be the length of the shortest path between nodes i and j. The closeness centrality C_i is given by

$$C_i = \frac{(N-1)}{\sum_j d_{ij}}$$



Betweeness Centrality

Intuition: A node is important if it lies on many shortest paths between other nodes

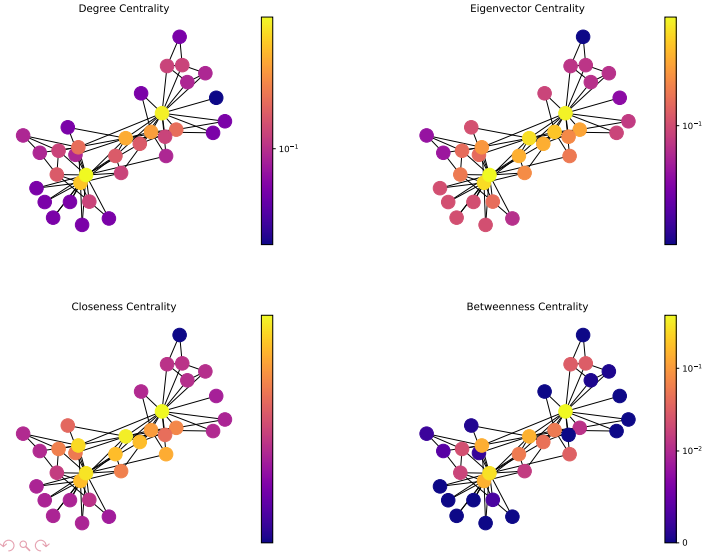
Definition of betweeness centrality

Starting from a node s to a node t and given the node i ($i \neq t \neq s$), $n_{st}^i = 1$ if node i lies on the shortest path between s and t, 0 therwise. The betweenness centrality is defined as

$$B_i = \sum_{s \neq i \neq t} \frac{n_{st}^i}{N_{st}},$$

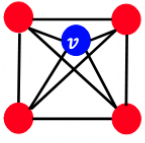
where N_{st} is the number of shortest paths between s and t.

Summing-up Importance Features

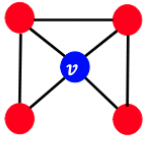


Clustering coefficient measures how connected v's neighboring nodes are

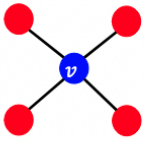
$$e_v = \frac{\#\text{ edges among neighboring nodes}}{\#\text{ of potential edges between pairs of neighboring nodes}}$$



$e_v = 1$



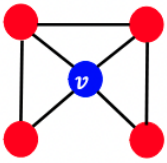
$e_v = 0.5$



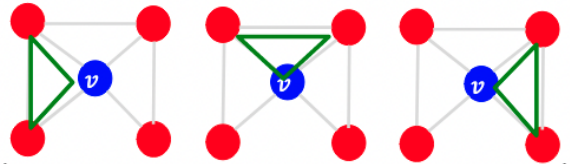
$e_v = 0$

Topological Features: Graphlets

Observation: Clustering coefficient counts the # of triangles in the ego network



$e_v = 0.5$

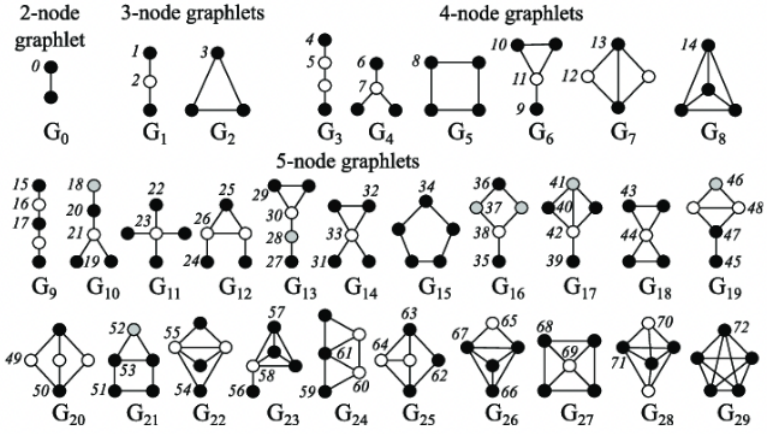


3 triangles (out of 6 node triplets)

We can generalize the above by counting # of pre-specified subgraphs, *i.e.*, **graphlets**

Graphlets: Definition

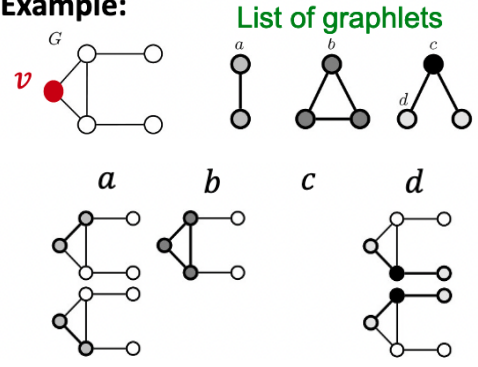
Rooted connected non-isomorphic subgraphs



Graphlet Degree Vector

Graphlet degree vectors (GDV) count the number of graphlets rooted at a given node.

Example:



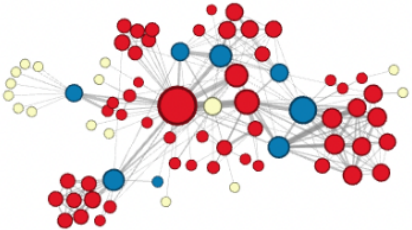
$$GDV_v = [2, 1, 0, 2]$$

Slide from Jure Leskovec - Stanford course

Remark: For c, we induced the whole subgraph, and we already counted the triangle in b

General Remarks

- Considering graphlets on 2 to 5 nodes, we get a vector of 73 coordinates to describe the topology of the node's neighborhood
- Centrality features capture the importance of a node in a graph
- Structural features capture the role of a node in a graph



Traditional Machine Learning for Graphs

Given an input graph

1. Extract structure feature
2. Machine learning algorithm (*e.g.* SVM)
3. Make a prediction

Feature engineering needs to be done **every single time**!

Traditional Machine Learning for Graphs

Given an input graph

1. Extract structure feature
2. Machine learning algorithm (*e.g.* SVM)
3. Make a prediction

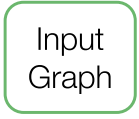
Feature engineering needs to be done **every single time**!

Can we learn the features automatically?

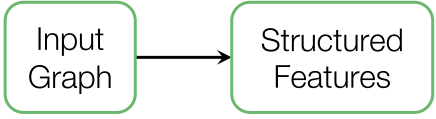
Graph Representation Learning

Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.

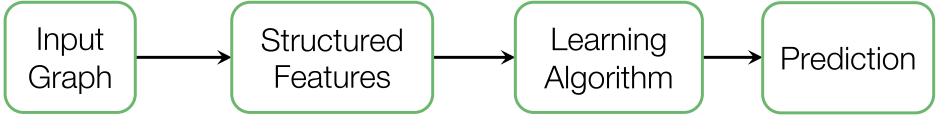
Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.



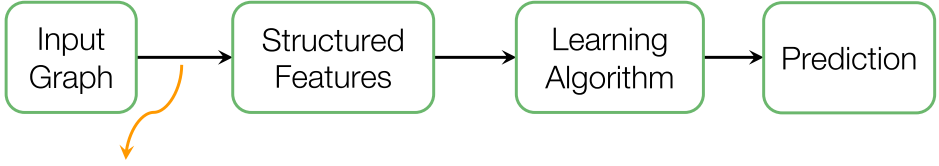
Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.



Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.



Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.



Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.

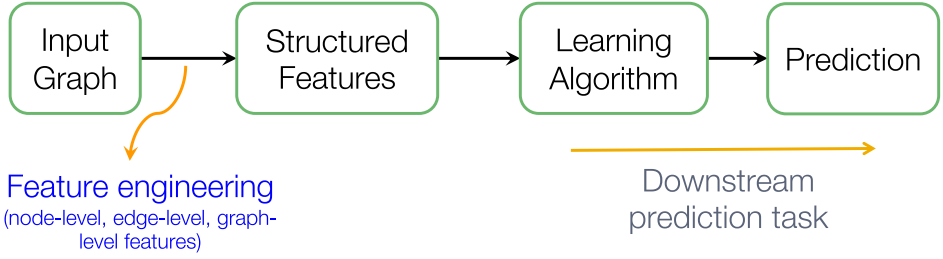


Image from Leskovec's lecture at Stanford

Graph representation learning alleviates the need to do feature engineering every single time.

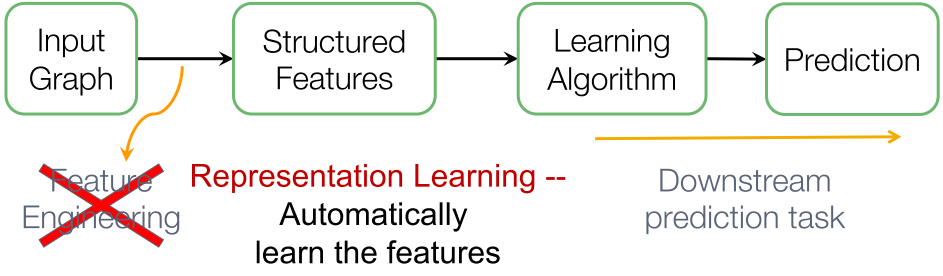
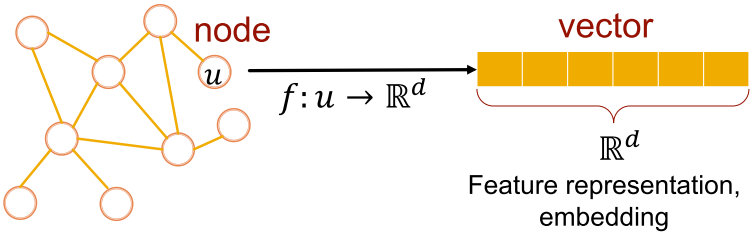


Image from Leskovec's lecture at Stanford

Goal: Efficient task-independent feature learning for machine learning with graphs aka **node embedding**

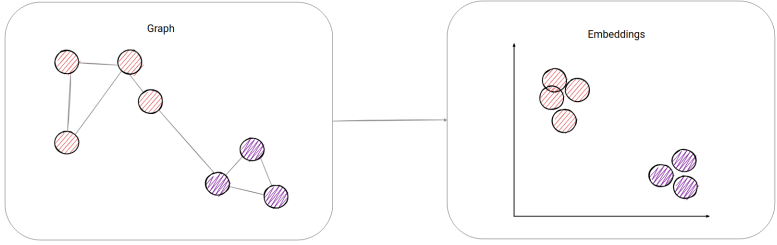


How can we learn $f(\cdot)$?

Node Embedding: Definition

Goal: Map nodes into a d-dimensional space embedding space

- Similarity of embeddings between two nodes should reflect their similarity in the graph
- Encode network information
- Can be used for many downstream task



Example

2D embedding of nodes of the Zachary's Karate Club network:

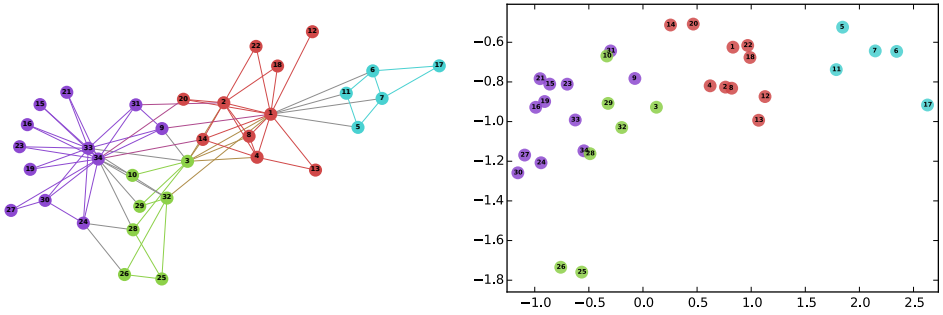


Image from "DeepWalk" (Perozzi *et al.* 2014)

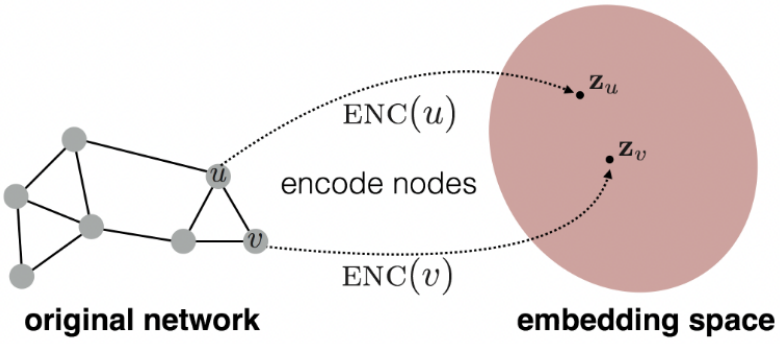
Set-up

- Input: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- *No additional features*
- Output: A matrix of embedding $\mathbf{Z} \in \mathbb{R}^{d \times N}$
- **Goal**: Learn $f : u \to \mathbb{R}^d$, $f(u) = \mathbf{z}_u$ for all nodes

We are not utilizing the node labels = unsupervised learning
Node embeddings are **task independent**

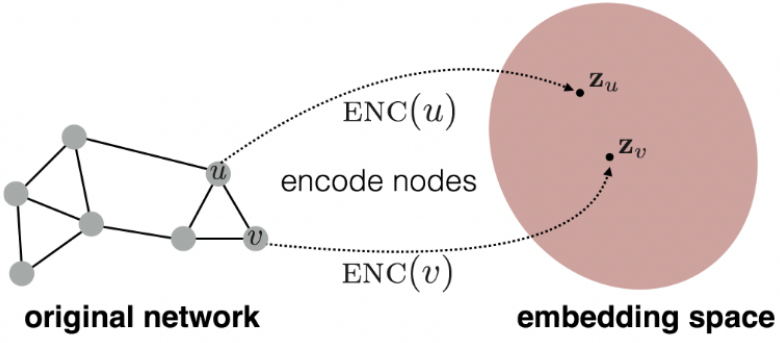
Embedding Nodes

Goal: Encode nodes so that similarity in the embedding space approximates similarity in the graph



Embedding Nodes

Goal: similarity$(u, v) \approx \mathbf{z}_u^\top \mathbf{z}_v$. Dot product = cosine of the angle between the vectors. If they are close together or in the same direction from the origin, the result is higher.



We need to define similarity(u, v)

Learning Node Embeddings

1. **Encoder** maps from nodes to embeddings
2. Need to define a node similarity function in the original space
3. **Decoder** maps from the embedding to the similarity score (we typically use the dot product in the embedding space)
4. Optimize the parameters such that the decoded similarity corresponds as closely as possible to the similarity in the original network

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

Shallow Encoding

Simplest approach: **Encoder** is just an embedding look-up

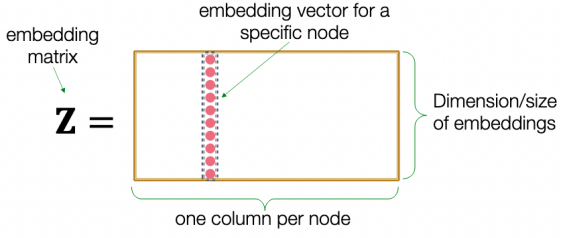
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z}.v$$

$\mathbf{Z} \in \mathbb{R}^{d \times N}$

$v \in \mathbb{I}^N$

Each column is a node embedding
(what we learn)

All zeroes except a one in column indicating node v



Note on Node Embeddings

- This is **unsupervised/self-supervised** way of learning node embeddings.
  - We are not utilizing node labels
  - We are not utilizing node features
  - The goal is to directly estimate a set of coordinates (*i.e.*, the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved.

Note on Node Embeddings

- This is **unsupervised/self-supervised** way of learning node embeddings.
  - We are not utilizing node labels
  - We are not utilizing node features
  - The goal is to directly estimate a set of coordinates (*i.e.*, the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved.
- These embeddings are **task independent**:
  - They are not trained for a specific task but can be used for any task.

Encoder + Decoder Framework

- **Shallow encoder**: Embedding lookup.
- Parameters to optimize: $\mathbf{Z}$ matrix of embeddings. For each node we have d parameters to learn.
- **Decoder**: Based on node similarity in the embedding space
- **Objective**: Maximize $\mathbf{z}_u^\top \mathbf{z}_v$ for pairs (u, v) that are similar.

Encoder + Decoder Framework

- **Shallow encoder**: Embedding lookup.
- Parameters to optimize: $\mathbf{Z}$ matrix of embeddings. For each node we have d parameters to learn.
- **Decoder**: Based on node similarity in the embedding space
- **Objective**: Maximize $\mathbf{z}_u^\top \mathbf{z}_v$ for pairs (u, v) that are similar.

How do we define similarity in the original space?

Encoder + Decoder Framework

- **Shallow encoder**: Embedding lookup.
- Parameters to optimize: $\mathbf{Z}$ matrix of embeddings. For each node we have d parameters to learn.
- **Decoder**: Based on node similarity in the embedding space
- **Objective**: Maximize $\mathbf{z}_u^\top \mathbf{z}_v$ for pairs (u, v) that are similar.

How do we define similarity in the original space?

Examples: Are linked, share neighbors, $\mathbf{z}_v^\top \mathbf{z}_u$ similar structural roles, etc.

Encoder + Decoder Framework

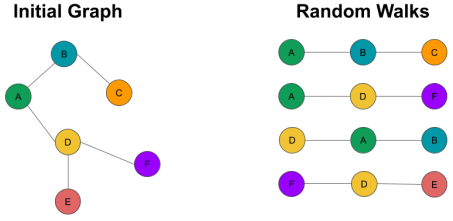
- **Shallow encoder**: Embedding lookup.
- Parameters to optimize: $\mathbf{Z}$ matrix of embeddings. For each node we have d parameters to learn.
- **Decoder**: Based on node similarity in the embedding space
- **Objective**: Maximize $\mathbf{z}_u^\top \mathbf{z}_v$ for pairs (u, v) that are similar.

How do we define similarity in the original space?

Examples: Are linked, share neighbors, $\mathbf{z}_v^\top \mathbf{z}_u$ *similar structural roles, etc.*
Idea: Define similarity of node based on **random walks**

Notations

- Learn a **vector** $\mathbf{z}_u$: the embedding of node u
- **Probability** $P(v|\mathbf{z}_u)$: Probability of how similar a node u is to some node $v \rightarrow$ predicted probability of visiting node v on random walks starting from node u



Given a *starting node*, we *select a neighbor* of it *at random*, and move to this neighbor; then repeat starting from the selected neighbors. **Random sequence of points visited this way = random walk.**

Objective: We want to learn the coordinate of $\mathbf{Z}$ such that

$$\mathbf{z}_u^\top \mathbf{z}_v \approx \text{the probability that } u \text{ and } v \text{ co-occur on a random walk}$$

1. Estimate probability of visiting node v on a random walk starting from u using a random walk strategy $R \rightarrow P_R(v|u)$
2. Optimize embeddings to encode these random walk statistics $\mathbf{z}_u^\top \mathbf{z}_v \propto P_R(v|u)$

Why Random Walks

1. **Expressivity**: Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information

   Idea: If random walk starting from node u visits v with high probability, u and v are similar

2. **Efficiency**: Do not need to consider all node pairs when training - only need to consider pairs that co-occur on random walk

Unsupervised Feature Learning

- Intuition: Find embedding of nodes in d-dimensional space that preserves similarity
- Idea: Learn node embedding such that nearby nodes are close together in the network
- Given a node u, how do we define nearby nodes?
  - $N_R(u)$... neighbourhood of u obtained by some random walk strategy R

Feature Learning as Optimization

- Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Our goal is to learn $f : u \rightarrow \mathbf{z} \in \mathbb{R}^d$ such that

$$\max_f \sum_{u \in \mathcal{V}} \log P(N_R(u)|\mathbf{z}_u),$$

where $N_R(u)$ is the neighborhood of node u by strategy R.

Given node u we want to learn features representations that are predictive of the nodes that appear in its random walk neighborhood.

Random Walk Optimization

1. Run short fixed length random walks starting from each node u in the graph using some random walk strategy
2. For each node u collect $N_R(u)$ the multiset of nodes visited on random walks starting from u.
3. Optimize embeddings $\mathbf{z}_u$ to maximize the likelihood of random walk co-occurences.

$$\max_f \sum_{u \in \mathcal{V}} \log P(N_R(u)|\mathbf{z}_u)$$

Equivalently, we can write this objective as

$$\mathcal{L} = \sum_{u \in \mathcal{V}} \sum_{v \in N_R(u)} -log(P(v|\mathbf{z}_u))$$

Parametrize $P(v|\mathbf{z}_u)$ using softmax

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

Maximize the dot product between embedding of starting node u and node $v \in N_R(u)$.

Parametrize $P(v|\mathbf{z}_u)$ using softmax

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

Maximize the dot product between embedding of starting node u and node $v \in N_R(u)$.

Why softmax? We want to assign as much probability mass to this dot product and as little to other dot product $\rightarrow$ we want node v to be most similar to node u out of all nodes n.

Putting it all together

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings

=

Finding embeddings $\mathbf{z}_u$ that minimizes $\mathcal{L}$

Putting it all together

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings

=

Finding embeddings $\mathbf{z}_u$ that minimizes $\mathcal{L}$

Nested sum over all the nodes gives a complexity of N^2

Negative Sampling

Can we approximate the normalizing term from $\mathcal{L}$?

Idea: Instead of normalizing w.r.t. all nodes, we normalize against k random negative samples n.

$$\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k} \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})),$$

with $n_i \sim P_{\mathbb{V}}$ (random distribution over nodes), and $\sigma(\cdot)$ the sigmoid function.

Probability of sampling a node is proportional to its degree

The objective is a form of **Noise Contrastive Estimation** which approximately maximizes the log probability of softmax.

Random Walks: Summary

1. Run short-fixed length random wak starting from each node on the graph
2. For each node collect $N_R(u)$, the multiset of nodes visited on random walks starting from u.
3. Optimize embeddings using stochastic gradient descent
   - Initialize $\mathbf{z}_i$ at some random value for all i
   - Iterate until convergence: $\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
     - Sample a node i, for all j calculate the derivatives $\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{z}_j}$
     - For all j, update $\mathbf{z}_j \leftarrow \mathbf{z}_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{z}_j}$

Random Walks: Summary

1. Run short-fixed length random wak starting from each node on the graph
2. For each node collect $N_R(u)$, the multiset of nodes visited on random walks starting from u.
3. Optimize embeddings using stochastic gradient descent
   - Initialize $\mathbf{z}_i$ at some random value for all i
   - Iterate until convergence: $\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
     - Sample a node i, for all j calculate the derivatives $\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{z}_j}$
     - For all j, update $\mathbf{z}_j \leftarrow \mathbf{z}_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{z}_j}$

When the random walk are unbiased: DeepWalk (Perozzi *et al.* 2014)

Random Walks Embedding

Goal: Learn similar node embedding for nodes with similar network neighborhoods.

General idea: Frame this goal as a maximum likelihood optimization problem, independent to the downstream task.

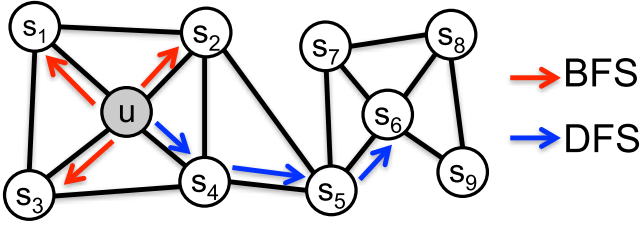
+ Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings.

Can we improve DeepWalk?

Node2Vec: Biased Walks (Grover and Leskovec, 2016)

General idea: Use flexible, biased random walks that can trade-off between **local** and **global** views in the network

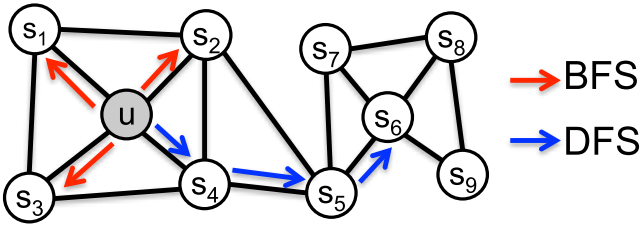
- Global: Depth-first search[2] as a way to explore as much of the network
- Local: Breadth-first search[3] fashion



[2](DFS) exploring depth
[3](BFS) explore locally

Biased Walk: Example

Two classic strategies to define a neighborhood $N_R(u)$ of a given node u.



Let's consider walks of length 3 ($N_R(u)$ of size 3):
$N_{BFS}(u) = \{s_1, s_2, s_3\}$ - Local microscopic view
$N_{DFS}(u) = \{s_4, s_5, s_6\}$ - Global macroscopic view

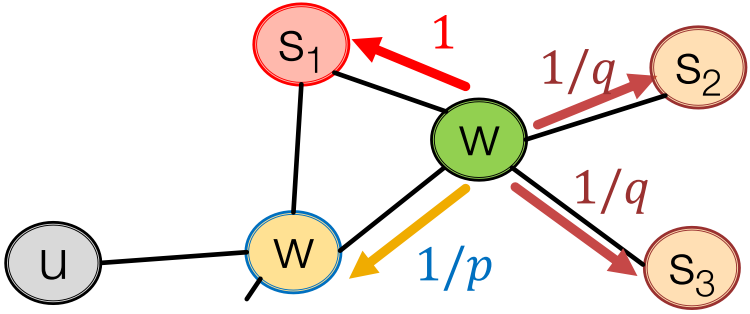
Definition of biased Random Walks

Goal: Generate biased fixed-length random walk R that produce $N_R(u)$ for a given node u.

- Introduce two hyper-parameters
  1. p: Return parameter to return back to the previous node
  2. q: In-out parameter that is the ratio between BFS vs DFS
- Produce 2^{nd} order biased random walk: remembers where it came from

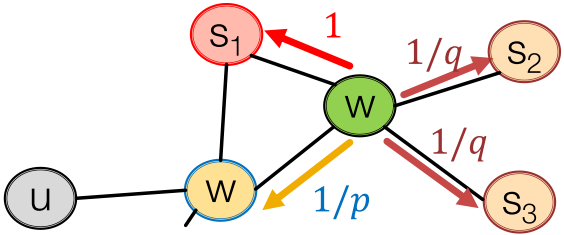
Biased Random Walks

Example: walker came over edge (w, w) and is at w. **Where to go next?**



- p and q models transition probabilities
  - p return parameter
  - q walk-away parameter

Example: walker came over edge (w, w) and is at w. **Where to go next?**



	Target x	Prob.	Dist. (t, x)
W →	t	$1/p$	0
	S_1	1	1
	S_2	$1/q$	2
	S_3	$1/q$	2

- BFS-like: low value of p
- DFS-like: low value of q

$N_R(u)$ is the set of nodes visited by the biased walk

Node2Vec Algorithm

1. Compute random walk probabilities
2. Simulate r random walks of length l starting from each node u
3. Optimize the node2vec negative sampling objective using Stochastic Gradient Descent

Wooclap

Main Takeaways

- Graphs are interesting structures for modeling structured-type data

Main Takeaways

- Graphs are interesting structures for modeling structured-type data
- There are many ways to represent a graph (adjacency matrix, list of edges, adjacency list). You should select one or more according to your application

Main Takeaways

- Graphs are interesting structures for modeling structured-type data
- There are many ways to represent a graph (adjacency matrix, list of edges, adjacency list). You should select one or more according to your application
- In classical graph ML, we perform hand-crafted engineering to later use some ML algorithm

Main Takeaways

- Graphs are interesting structures for modeling structured-type data
- There are many ways to represent a graph (adjacency matrix, list of edges, adjacency list). You should select one or more according to your application
- In classical graph ML, we perform hand-crafted engineering to later use some ML algorithm
- There are many classical node-level features like degree, eigenvector, closeness, and betweenness centrality

Main Takeaways

- Graphs are interesting structures for modeling structured-type data
- There are many ways to represent a graph (adjacency matrix, list of edges, adjacency list). You should select one or more according to your application
- In classical graph ML, we perform hand-crafted engineering to later use some ML algorithm
- There are many classical node-level features like degree, eigenvector, closeness, and betweenness centrality
- In graph representation learning, we try to learn automatically good node embeddings: DeepWalk and Node2Vec

Main Takeaways

- Graphs are interesting structures for modeling structured-type data
- There are many ways to represent a graph (adjacency matrix, list of edges, adjacency list). You should select one or more according to your application
- In classical graph ML, we perform hand-crafted engineering to later use some ML algorithm
- There are many classical node-level features like degree, eigenvector, closeness, and betweenness centrality
- In graph representation learning, we try to learn automatically good node embeddings: DeepWalk and Node2Vec
- The issue with DeepWalk-type models is that we need to launch the optimization problem for every new graph: next week we'll see GNNs that alleviate this issue

Thank you!

Jhony H. Giraldo: jhony.giraldo@telecom-paris.fr
(https://jhonygiraldo.github.io/)

References I

J. Jumper et al.
Highly accurate protein structure prediction with alphafold.
Nature, 596(7873):583–589, 2021.

R. Lam et al.
GraphCast: Learning skillful medium-range global weather forecasting.
arXiv preprint arXiv:2212.12794, 2022.

Ilan Price et al.
GenCast: Diffusion-based ensemble forecasting for medium-range weather.
arXiv preprint arXiv:2312.15796, 2023.

References II

A. Sanchez-Gonzalez et al.
Learning to simulate complex physics with graph networks.
In *International Conference on Machine Learning*, 2020.

E. Tolstaya et al.
Learning decentralized controllers for robot swarms with graph neural networks.
In *Conference on Robot Learning*, 2020.