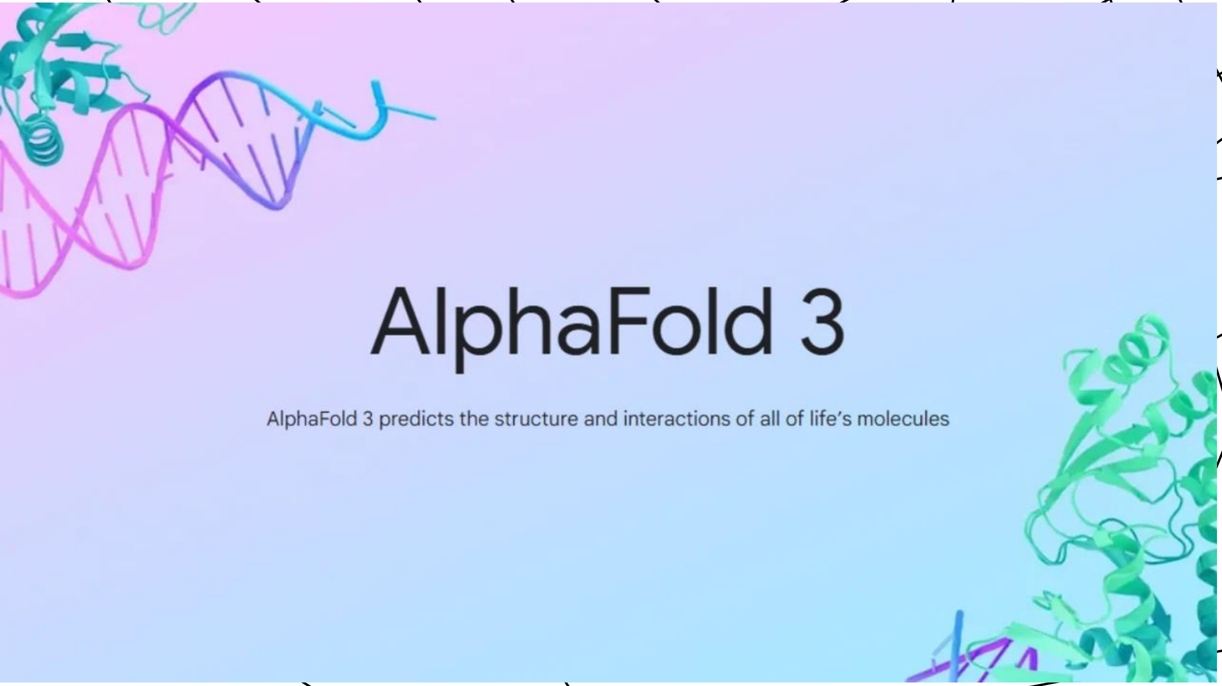
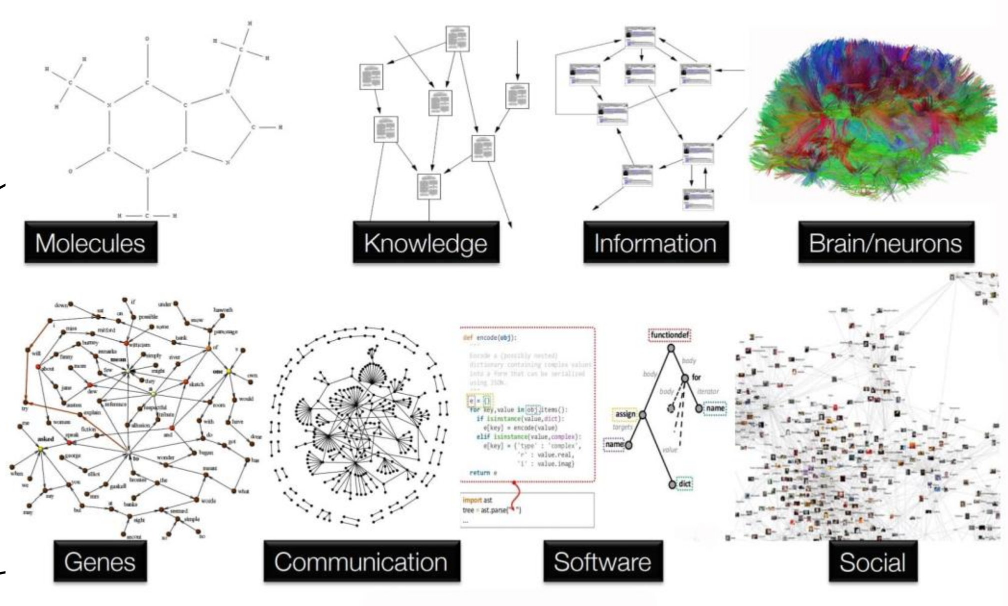
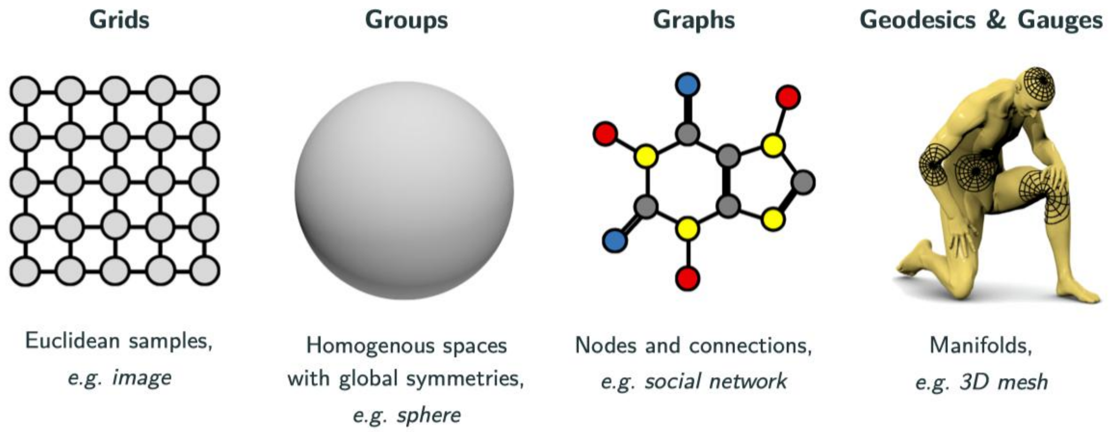




INTRODUCTION TO GRAPH REPRESENTATION LEARNING

Andreas Makris

LAI Reading Group



Images from NVIDIA, DeepMind,
Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges

Resources

Presentation based on:

- Introduction to Graph Representation Learning
  https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf
- Deep Graph-Based Learning Course
  https://github.com/basiralab/DGL/tree/main

For people that like videos:

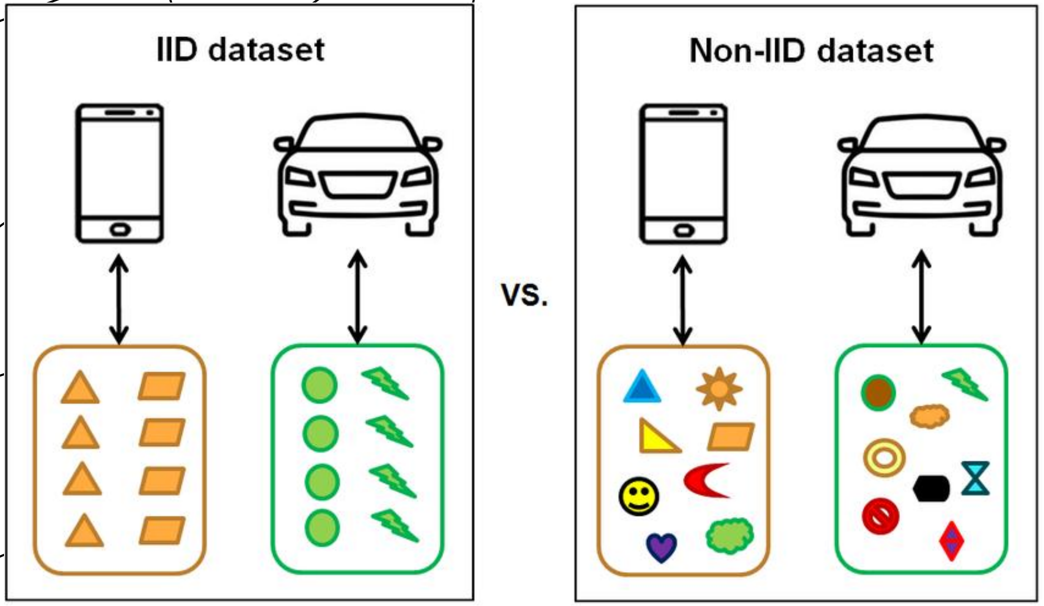
Stanford CS224W: Machine Learning with Graphs

https://www.youtube.com/playlist?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn

Deep Graph Learning

https://www.youtube.com/playlist?list=PLug43ldmRSo14Y_vt7S6vanPGh-JpHR7T

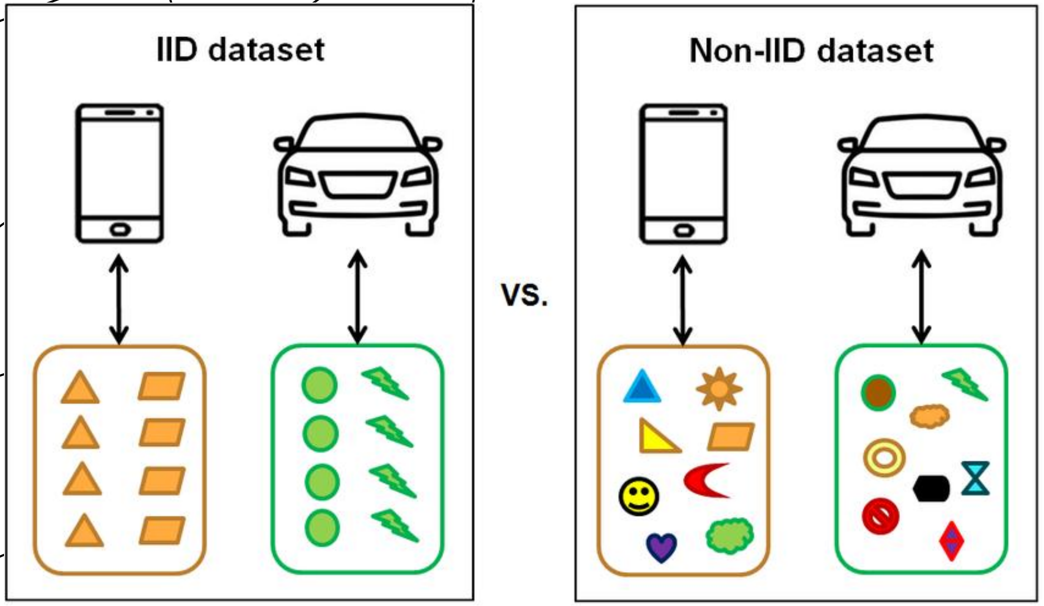
The Usual ML/DL Assumption



Our data is independent and identically distributed!

Image from Entropy to Mitigate Non-IID Data Problem on Federated Learning for the Edge Intelligence Environment

The Usual ML/DL Assumption

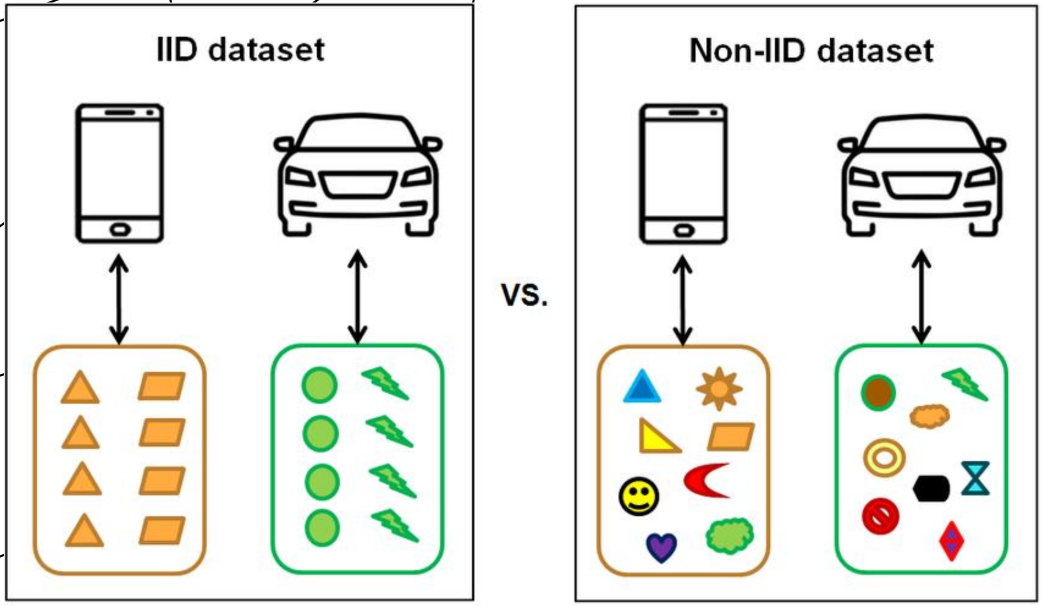


Our data is independent and identically distributed!

What if it isn’t?

Image from Entropy to Mitigate Non-IID Data Problem on Federated Learning for the Edge Intelligence Environment

The Usual ML/DL Assumption



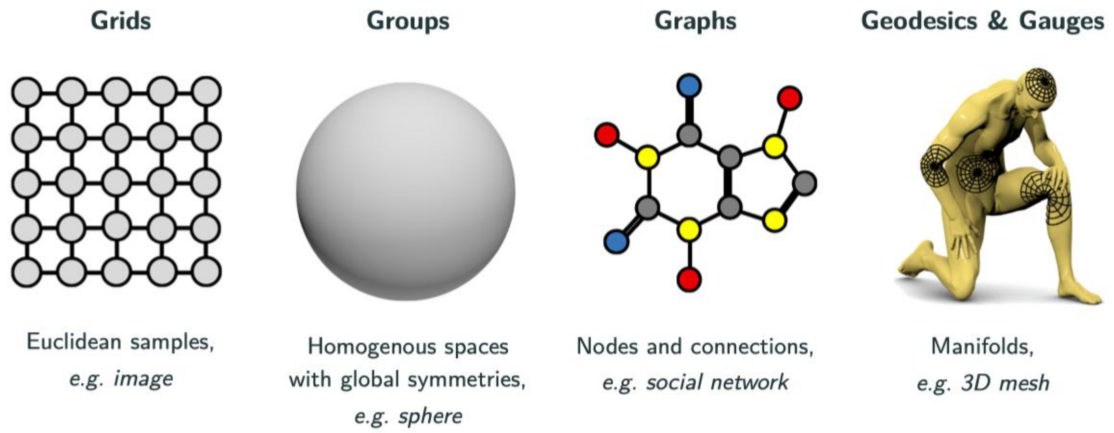
Our data is independent and identically distributed!

What if it isn't?

How can we use that to enhance our models?

Image from Entropy to Mitigate Non-IID Data Problem on Federated Learning for the Edge Intelligence Environment

Geometric Deep Learning

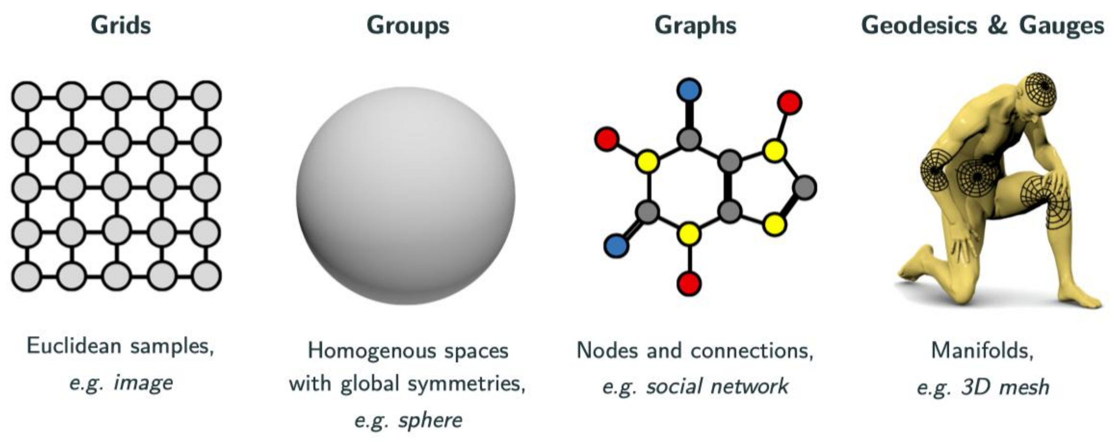


The structure of the data has additional information.

Inductive bias refers to the assumptions a learning algorithm makes to generalize beyond its training data.

Image from Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges

Geometric Deep Learning



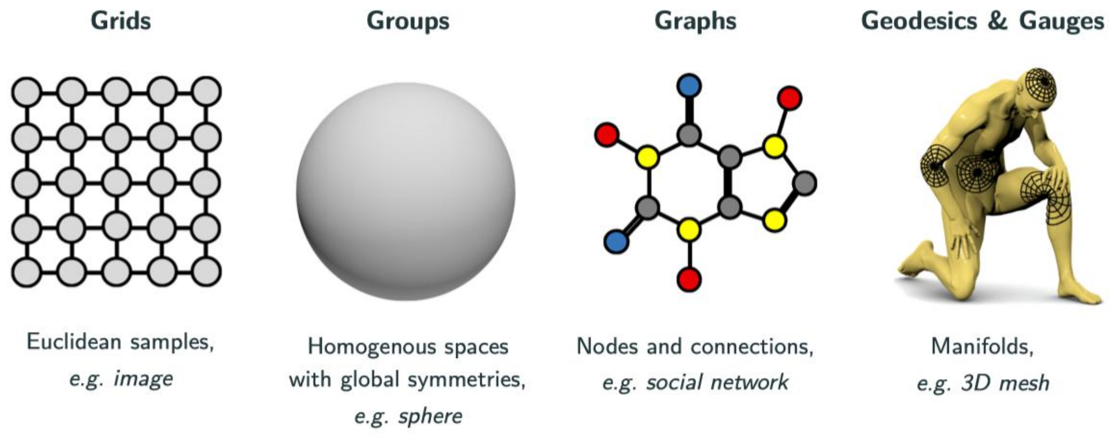
The structure of the data has additional information.

Geometric Deep Learning utilizes the structure of the data to improve our models.

Inductive bias refers to the assumptions a learning algorithm makes to generalize beyond its training data.

Image from Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges

Geometric Deep Learning



The structure of the data has additional information.

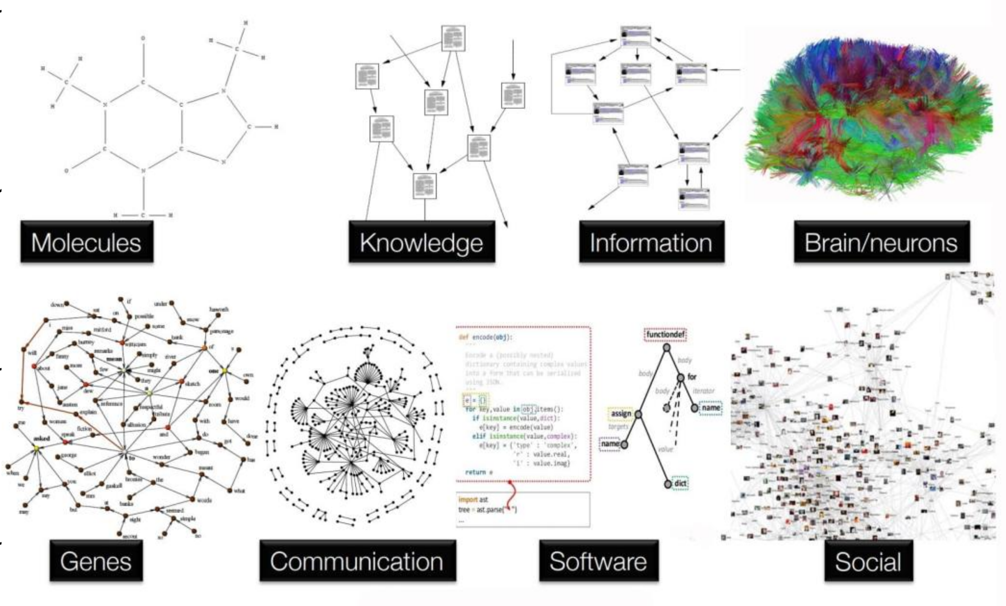
Geometric Deep Learning utilizes the structure of the data to improve our models.

We say we use **inductive bias** in our models.

Inductive bias refers to the assumptions a learning algorithm makes to generalize beyond its training data.

Image from Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges

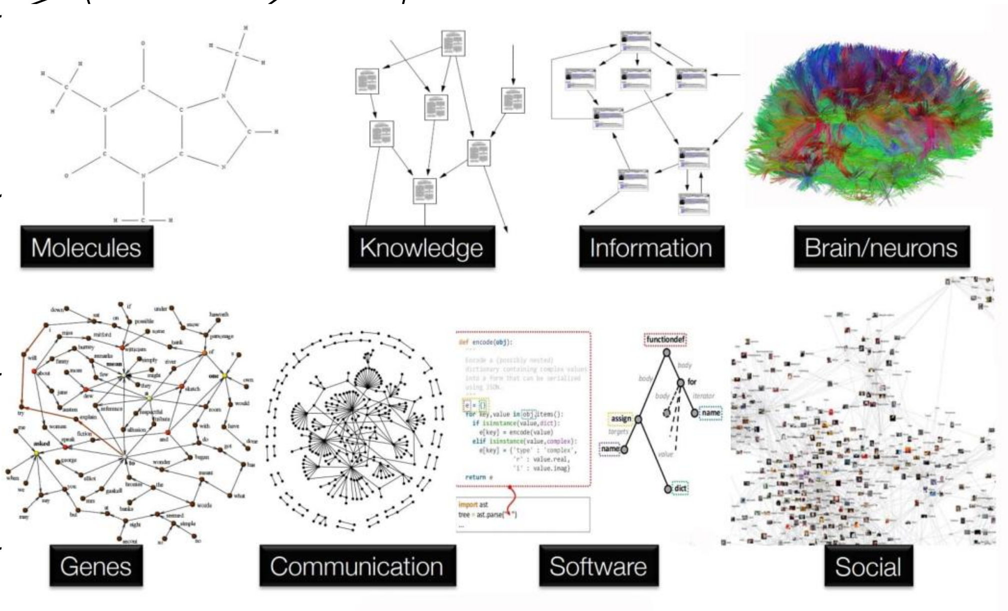
Graph Neural Networks



The most common and popular structure that we utilize in Deep Learning are graphs.

Image from https://blogs.nvidia.com/blog/what-are-graph-neural-networks/

Graph Neural Networks

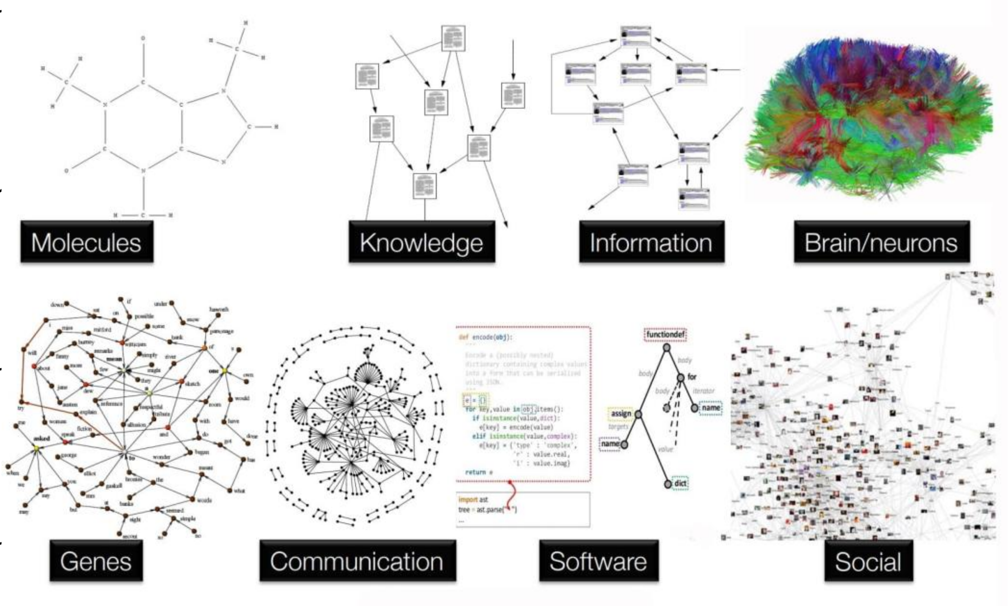


The most common and popular structure that we utilize in Deep Learning are graphs.

Neural Network architectures/techniques that consider the graph-structure of the data.

Image from https://blogs.nvidia.com/blog/what-are-graph-neural-networks/

Graph Neural Networks



The most common and popular structure that we utilize in Deep Learning are graphs.

Neural Network architectures/techniques that consider the graph-structure of the data.

We will mainly focus on GNNs this term!

Image from https://blogs.nvidia.com/blog/what-are-graph-neural-networks/

Background for GNNs

Graph Theory

Deep Learning

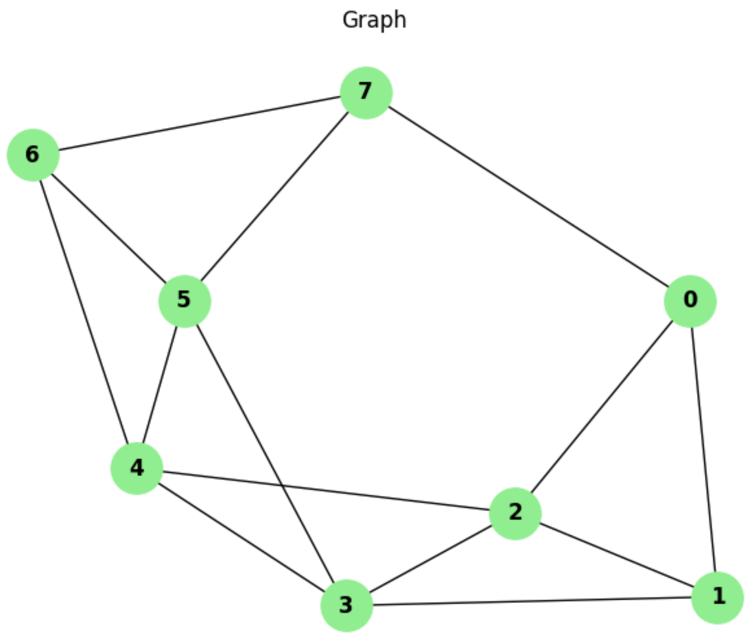
Background for GNNs

Graph Theory

Deep Learning

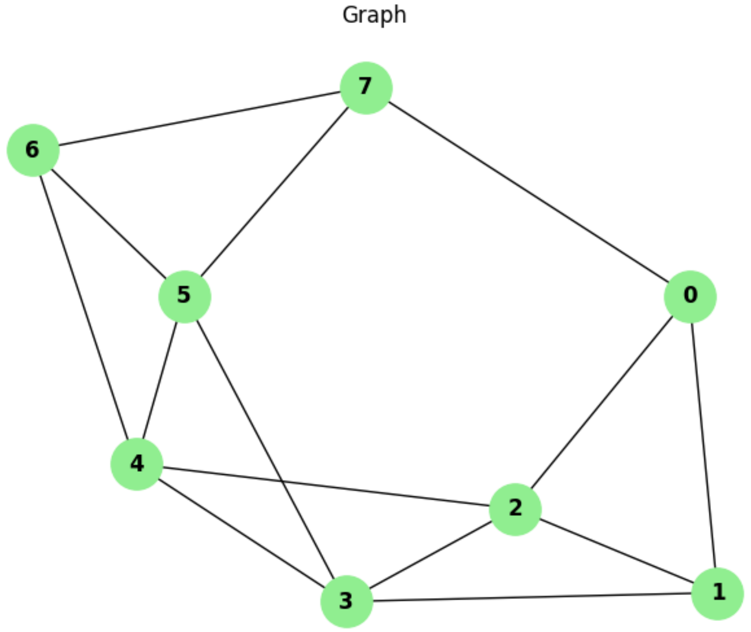
We assume some ML/DL background.
This and next week we aim to gain some experience on Graph Theory and traditional ML for Graphs

What is a Graph?



“A graph or a network is a collection of objects along with a set of interactions between pairs of these objects.” (GRL Book)

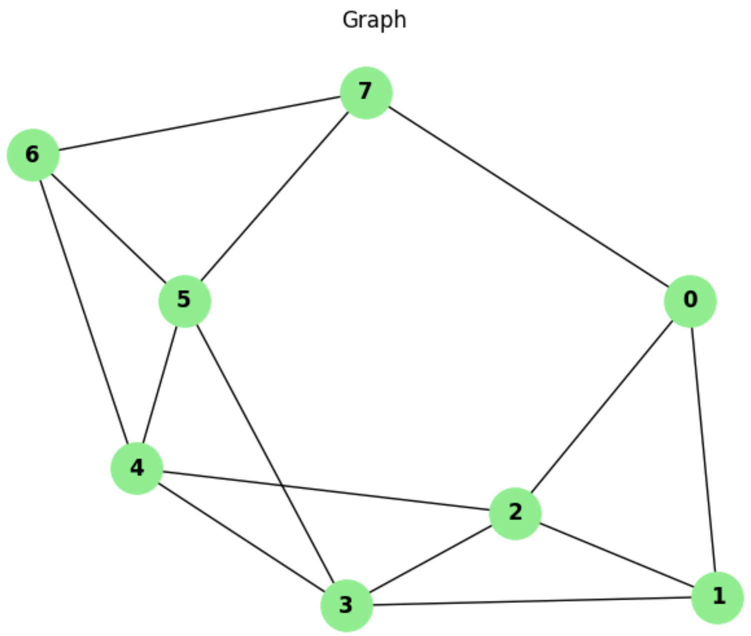
What is a Graph?



“A graph or a network is a collection of objects along with a set of interactions between pairs of these objects.” (GRL Book)

“Formally, a graph **G = (V, E)** is defined by a set of nodes **V** and a set of edges **E** between these nodes.” (GRL Book)

What is a Graph?



"A graph or a network is a collection of objects along with a set of interactions between pairs of these objects." (GRL Book)

"Formally, a graph **G = (V, E)** is defined by a set of nodes **V** and a set of edges **E** between these nodes." (GRL Book)

For example, **V={0, 1, 2, 3, 4, 5, 6, 7}** and
E={(0, 1), (0, 2), (0, 7), (1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7)}

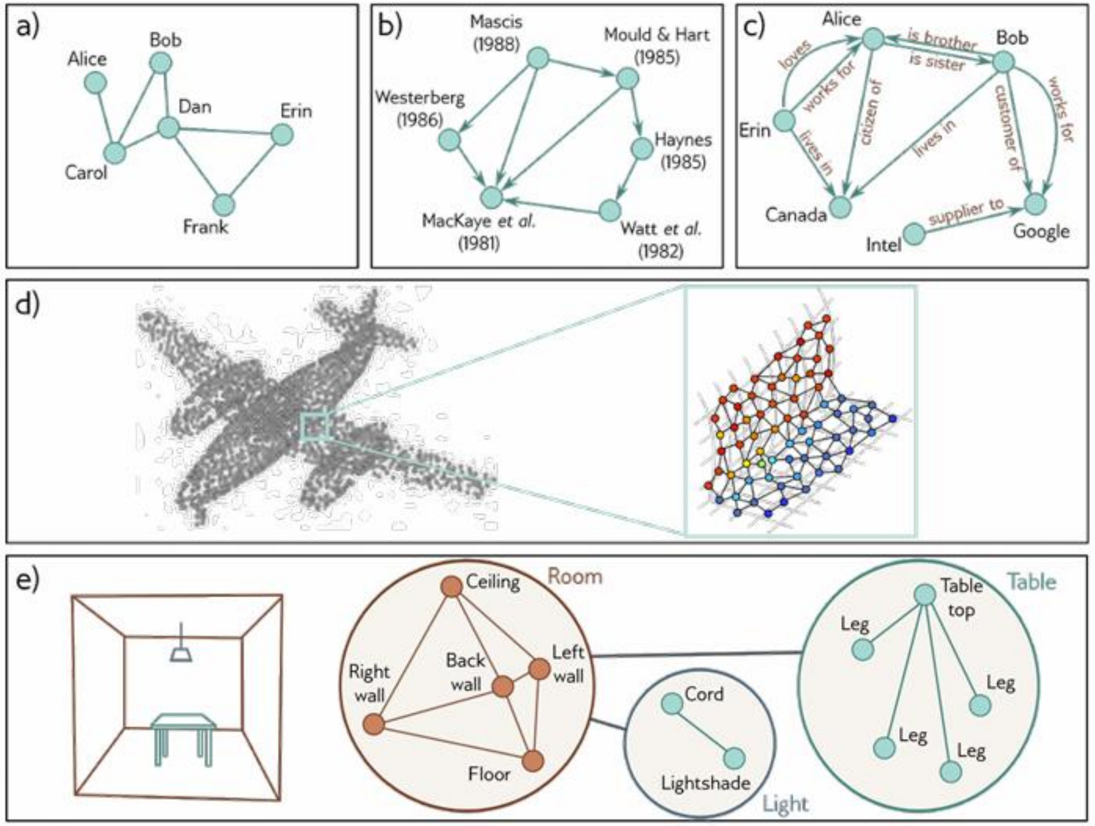


Figure 2 Types of graph. a) A social network is an undirected graph; the connections between people are symmetric. b) A citation network is a directed graph; one publication cites another, so the relationship is asymmetric. c) A knowledge graph is a directed heterogeneous multigraph. The nodes are heterogeneous in that they represent different object types (people, places, companies) and there may be multiple edges representing different relations between each node. d) A point set can be converted to a graph by forming edges between nearby points. Each node has an associated position in 3D space and this is termed a geometric graph (adapted from Hu et al. 2021). e) The scene on the left can be represented by a hierarchical graph. The topology of the room, table, and light are all represented by graphs and these graphs form nodes in a larger graph representing object adjacency (adapted from Fernández-Madrigal & González 2002).

Figure from Basira Lab

Why do we care about Graphs?

Graphs are everywhere!

- Social Network Graph (nodes: individuals; edges: relationships or friendships)
- Computer Network Graph (nodes: computers, routers, or servers; edges: data connections or cables)
- Transportation Network Graph (nodes: cities or intersections; edges: roads, rail lines, or flight paths)
- Web Graph (nodes: webpages; edges: hyperlinks between pages)
- Molecule Structure Graph (nodes: atoms; edges: chemical bonds)
- Academic Papers Graph (nodes: authors; edges: whether they co-authored a paper)

Graph Learning Tasks

Node Level

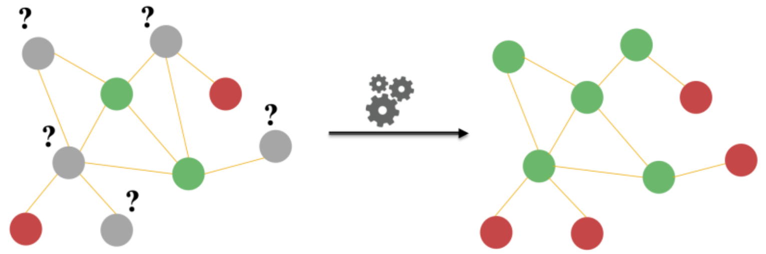
Edge Level

Graph Level

Graph Learning Tasks

Node Level

- Node classification/regression
  e.g. price for each node
- Node clustering
- Representation learning (node embeddings)



Edge Level

Graph Level

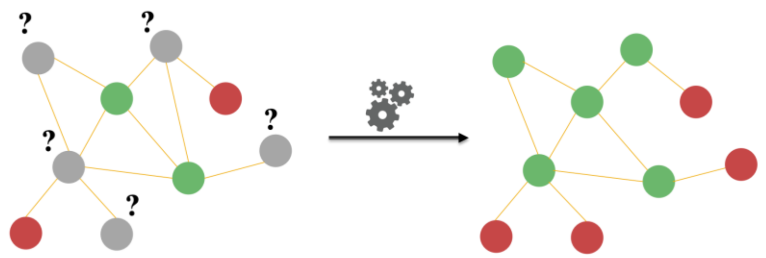
cs224w

A Survey on Knowledge Graph Embeddings for Link Prediction

Graph Learning Tasks

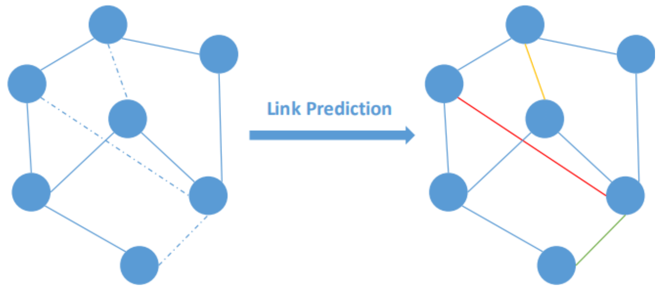
Node Level

- Node classification/regression e.g. price for each node
- Node clustering
- Representation learning (node embeddings)



Edge Level

- Edge classification/regression e.g. edges are transactions, fraud or not
- Link prediction e.g. recommend movies



Graph Level

cs224w

A Survey on Knowledge Graph Embeddings for Link Prediction

Graph Learning Tasks

Node Level

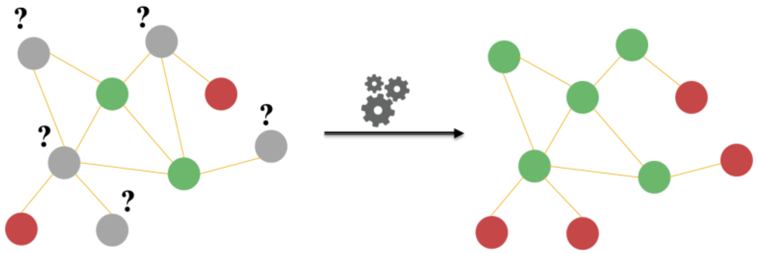
- Node classification/regression e.g. price for each node
- Node clustering
- Representation learning (node embeddings)

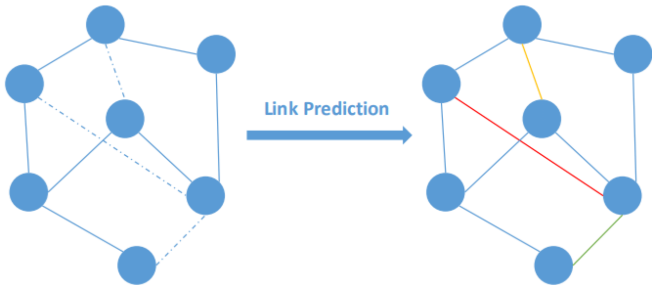
Edge Level

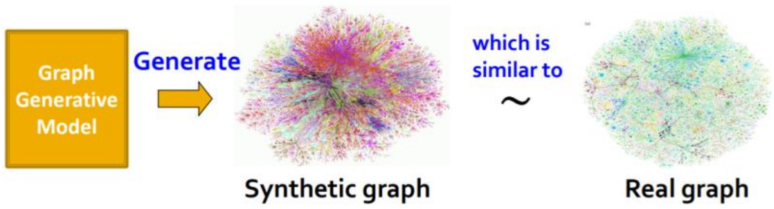
- Edge classification/regression e.g. edges are transactions, fraud or not
- Link prediction e.g. recommend movies

Graph Level

- Graph classification/regression e.g. is the molecule toxic
- Graph clustering
- Graph generation e.g. create new molecules







cs224w

A Survey on Knowledge Graph Embeddings for Link Prediction

How to use a Graph for ML?

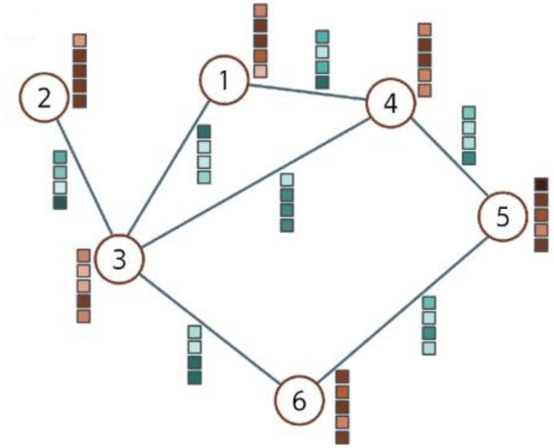
Bashira Lab

How to use a Graph for ML?

- Adjacency matrix **A** (N x N)
- Node data **X** (N x D)
- Edge data **E** (N x L)
- Graph data (embedding for each graph)

N: number of nodes
D: node embedding dimension
L: edge embedding dimension

```python
adj_matrix = np.array([
    #0  1  2  3  4  5  6  7
    [0, 1, 1, 0, 0, 0, 0, 1],  # Node 0 connected to 1, 2, 7
    [1, 0, 1, 1, 0, 0, 0, 0],  # Node 1 connected to 0, 2, 3
    [1, 1, 0, 1, 1, 0, 0, 0],  # Node 2 connected to 0, 1, 3, 4
    [0, 1, 1, 0, 1, 1, 0, 0],  # Node 3 connected to 1, 2, 4, 5
    [0, 0, 1, 1, 0, 1, 1, 0],  # Node 4 connected to 2, 3, 5, 6
    [0, 0, 0, 1, 1, 0, 1, 1],  # Node 5 connected to 3, 4, 6, 7
    [0, 0, 0, 0, 1, 1, 0, 1],  # Node 6 connected to 4, 5, 7
    [1, 0, 0, 0, 0, 1, 1, 0]   # Node 7 connected to 0, 5, 6
])
```



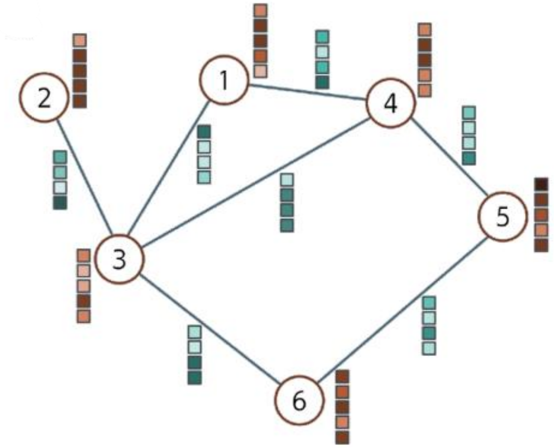
How to use a Graph for ML?

- Adjacency matrix **A** (N x N)
- Node data **X** (N x D)
- Edge data **E** (N x L)

N: number of nodes
D: node embedding dimension
L: edge embedding dimension

```
adj_matrix = np.array([
    #0  1  2  3  4  5  6  7
    [0, 1, 1, 0, 0, 0, 0, 1],  # Node 0 connected to 1, 2, 7
    [1, 0, 1, 1, 0, 0, 0, 0],  # Node 1 connected to 0, 2, 3
    [1, 1, 0, 1, 1, 0, 0, 0],  # Node 2 connected to 0, 1, 3, 4
    [0, 1, 1, 0, 1, 1, 0, 0],  # Node 3 connected to 1, 2, 4, 5
    [0, 0, 1, 1, 0, 1, 1, 0],  # Node 4 connected to 2, 3, 5, 6
    [0, 0, 0, 1, 1, 0, 1, 1],  # Node 5 connected to 3, 4, 6, 7
    [0, 0, 0, 0, 1, 1, 0, 1],  # Node 6 connected to 4, 5, 7
    [1, 0, 0, 0, 0, 1, 1, 0]   # Node 7 connected to 0, 5, 6
])
```

How do we get node/edge data?

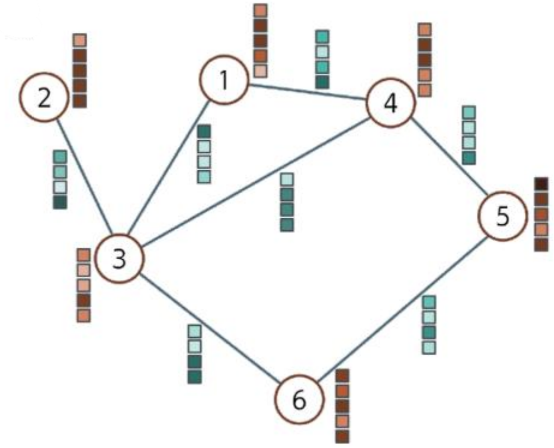


How to use a Graph for ML?

- Adjacency matrix **A** (N x N)
- Node data **X** (N x D)
- Edge data **E** (N x L)

N: number of nodes
D: node embedding dimension
L: edge embedding dimension

```
adj_matrix = np.array([
    #0  1  2  3  4  5  6  7
    [0, 1, 1, 0, 0, 0, 0, 1],  # Node 0 connected to 1, 2, 7
    [1, 0, 1, 1, 0, 0, 0, 0],  # Node 1 connected to 0, 2, 3
    [1, 1, 0, 1, 1, 0, 0, 0],  # Node 2 connected to 0, 1, 3, 4
    [0, 1, 1, 0, 1, 1, 0, 0],  # Node 3 connected to 1, 2, 4, 5
    [0, 0, 1, 1, 0, 1, 1, 0],  # Node 4 connected to 2, 3, 5, 6
    [0, 0, 0, 1, 1, 0, 1, 1],  # Node 5 connected to 3, 4, 6, 7
    [0, 0, 0, 0, 1, 1, 0, 1],  # Node 6 connected to 4, 5, 7
    [1, 0, 0, 0, 0, 1, 1, 0]   # Node 7 connected to 0, 5, 6
])
```



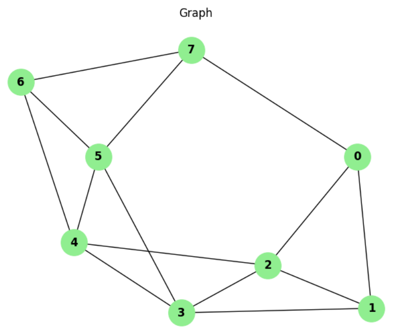
How do we get node/edge data?

Before answering that, let's look at some types of graphs

Types of Graphs

Simple Graphs

- At most 1 edge between each pair of nodes.
- A node can't have an edge with itself.
- Edges are undirected



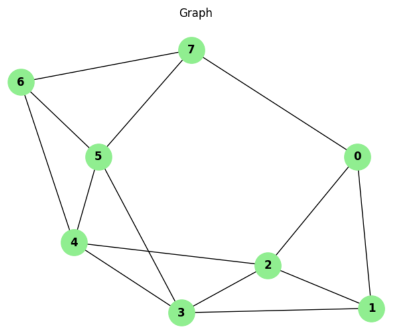
https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/

Wikipedia

Types of Graphs

Simple Graphs

- At most 1 edge between each pair of nodes.
- A node can’t have an edge with itself.
- Edges are undirected

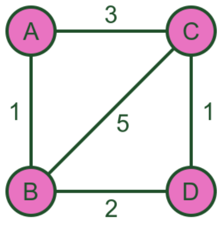


https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/

Wikipedia

Weighted Graphs

- Weighted graphs have a weight associated with each edge. For example, distance to travel between cities.
- We usually use the weight matrix rather than the adjacency matrix.



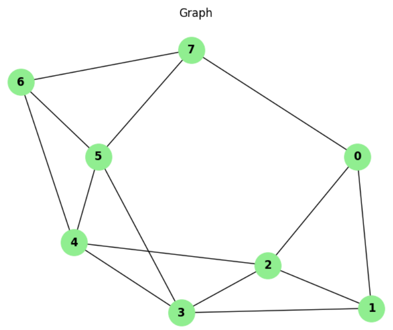
Weighted graph

	A	B	C	D
A	0	1	3	0
B	1	0	5	2
C	3	5	0	1
D	0	2	1	0

Types of Graphs

Simple Graphs

- At most 1 edge between each pair of nodes.
- A node can't have an edge with itself.
- Edges are undirected

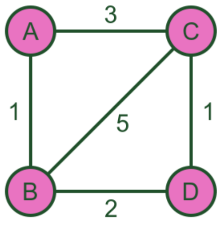


https://graphicmaths.com/computer-science/graph-theory/adjacency-matrices/

Wikipedia

Weighted Graphs

- Weighted graphs have a weight associated with each edge. For example, distance to travel between cities.
- We usually use the weight matrix rather than the adjacency matrix.

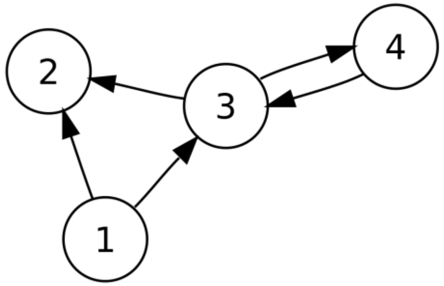


Weighted graph

	A	B	C	D
A	0	1	3	0
B	1	0	5	2
C	3	5	0	1
D	0	2	1	0

Directed Graphs

- Edges have direction.



Multi-Relational Graphs

Graphs that have different types of edges. Then we get an adjacency tensor of shape N x # of types of edges x N. Example: relationship network, edges can be Facebook, Instagram, Twitter

HMSG: Heterogeneous Graph Neural Network based on Metapath Subgraph Learning

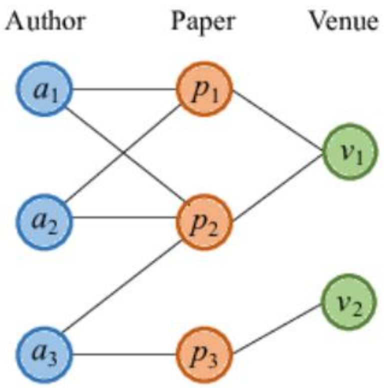
Learning embeddings for multiplex networks using triplet loss

Multi-Relational Graphs

Graphs that have different types of edges. Then we get an adjacency tensor of shape
N x # of types of edges x N. Example: relationship network, edges can be Facebook, Instagram, Twitter

Heterogeneous Graphs

- We also have nodes of different types!
- Example: users and movies for recommender systems.



[HMSG: Heterogeneous Graph Neural Network based on Metapath Subgraph Learning]

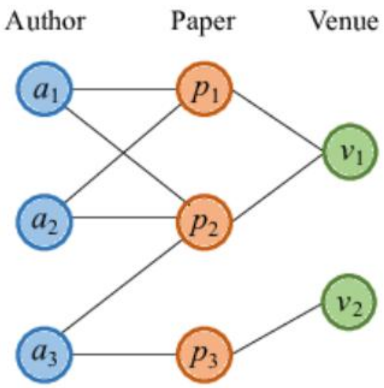
Learning embeddings for multiplex networks using triplet loss

Multi-Relational Graphs

Graphs that have different types of edges. Then we get an adjacency tensor of shape N x # of types of edges x N. Example: relationship network, edges can be Facebook, Instagram, Twitter

Heterogeneous Graphs

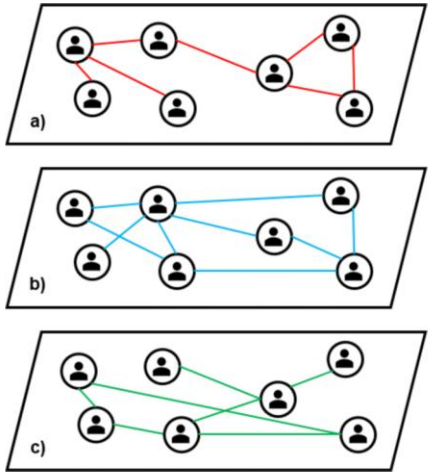
- We also have nodes of different types!
- Example: users and movies for recommender systems.



[HMSG: Heterogeneous Graph Neural Network based on Metapath Subgraph Learning]

Multiplex Graphs

- The graph can be decomposed in a set of different layers.
- Each layer corresponds to a different type of edge.
- Example: Transportation network



Learning embeddings for multiplex networks using triplet loss

ML for Graphs

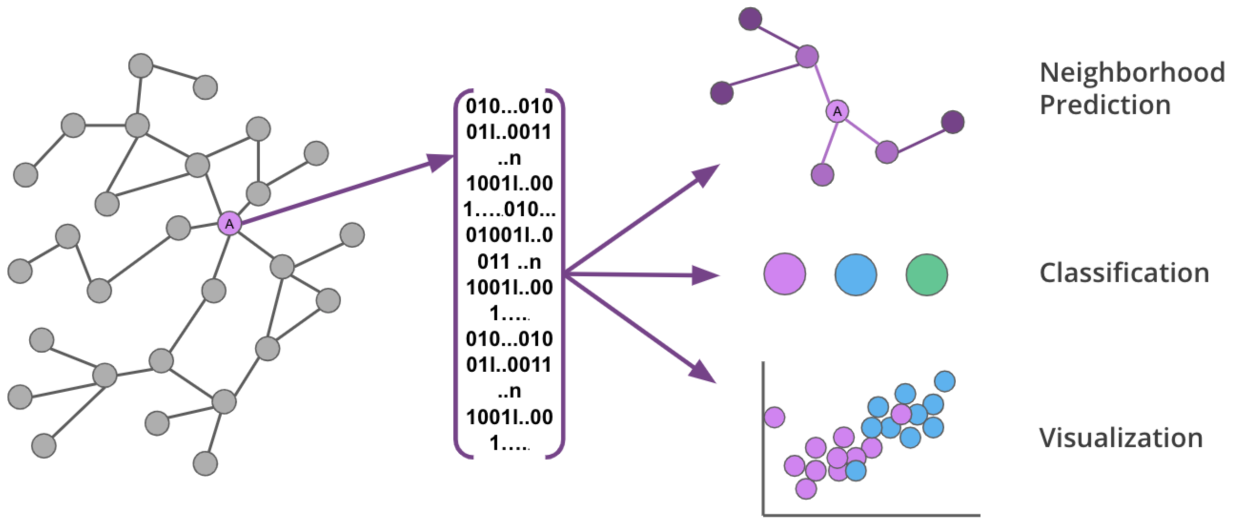
“ The challenge in these graph-level tasks, however, is how to define useful features that take into account the relational structure within each datapoint” – GRL Book

https://neo4j.com/blog/machine-learning/announcing-graph-native-machine-learning-in-neo4j/

ML for Graphs

“ The challenge in these graph-level tasks, however, is how to define useful features that take into account the relational structure within each datapoint” – GRL Book

1. Extract statistics for each node/edge/graph.
2. Use them as input to a standard ML model!



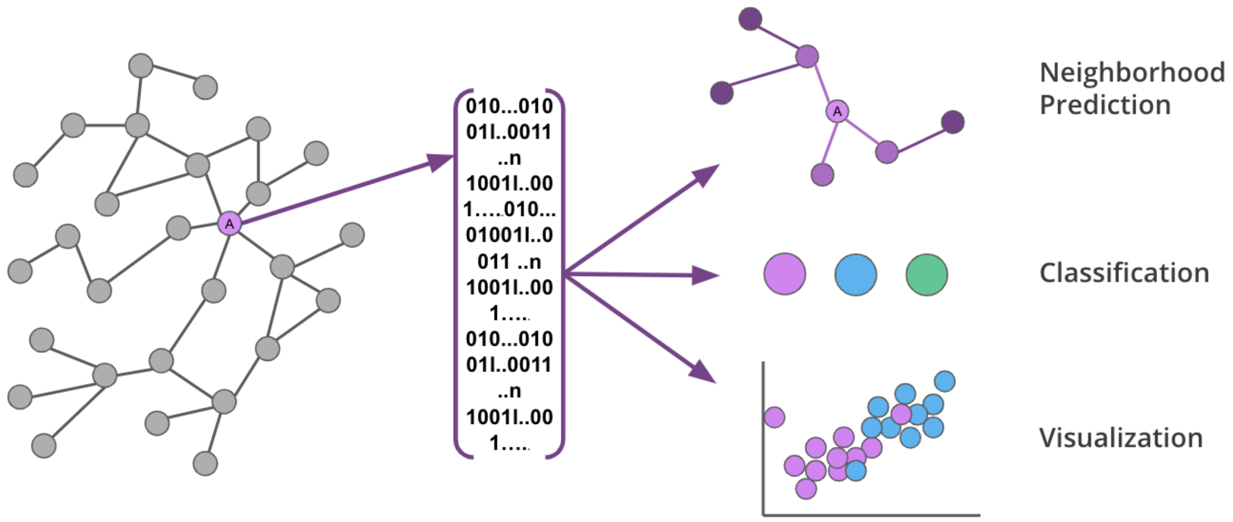
https://neo4j.com/blog/machine-learning/announcing-graph-native-machine-learning-in-neo4j/

ML for Graphs

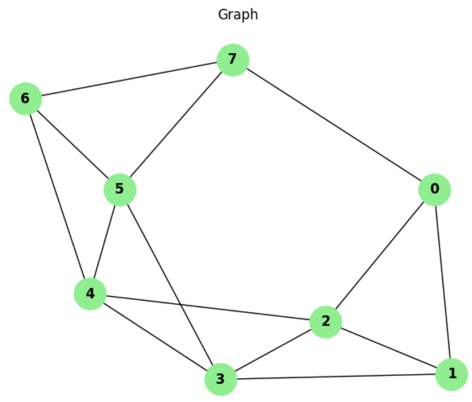
“ The challenge in these graph-level tasks, however, is how to define useful features that take into account the relational structure within each datapoint” – GRL Book

1. Extract statistics for each node/edge/graph.
2. Use them as input to a standard ML model!

What type of statistics can we extract



https://neo4j.com/blog/machine-learning/announcing-graph-native-machine-learning-in-neo4j/



The Power of **A**

A^k is the matrix of the **number of paths of length k** between each pair of nodes!

Adjacency Matrix

0	1	1	0	0	0	0	1
1	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0
0	1	1	0	1	1	0	0
0	0	1	1	0	1	1	0
0	0	0	1	1	0	1	1
0	0	0	0	1	1	0	1
1	0	0	0	0	1	1	0

A**2

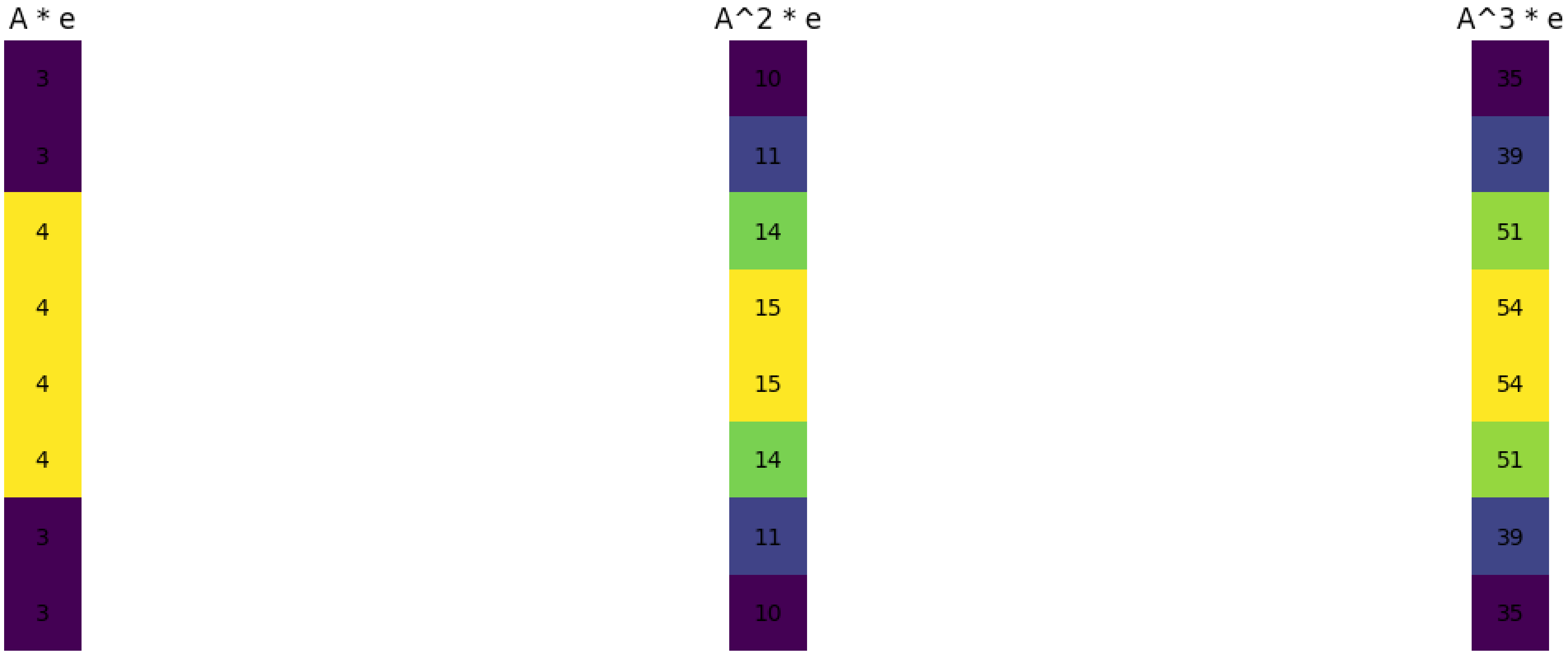
3	1	1	2	1	1	1	0
1	3	2	1	2	1	0	1
1	2	4	2	1	2	1	1
2	1	2	4	2	1	2	1
1	2	1	2	4	2	1	2
1	1	2	1	2	4	2	1
1	0	1	2	1	2	3	1
0	1	1	1	2	1	1	3

A**3

2	6	7	4	5	4	2	5
6	4	7	8	4	4	4	2
7	7	6	9	9	5	4	4
4	8	9	6	9	9	4	5
5	4	9	9	6	9	8	4
4	4	5	9	9	6	7	7
2	4	4	4	8	7	4	6
5	2	4	5	4	7	6	2

The Power of **A**

Each entry is the total number of k-step walks starting from the corresponding vertex in the graph



e is the unit vector

Node-level Statistics

- Node degree: number of edges a node is connected to. $$d_u = \sum_{v \in V} \mathbf{A}[u, v].$$

https://networkx.org/documentation/stable/reference/algorithms/centrality.html

Node-level Statistics

- Node degree: number of edges a node is connected to. $d_u = \sum_{v \in V} \mathbf{A}[u, v].$
- Node centrality:
  - Considers how important a node's neighbours are.
  - There are many different centrality measures (e.g. eigenvector centrality, betweenness centrality).

https://networkx.org/documentation/stable/reference/algorithms/centrality.html

Node-level Statistics

- Node degree: number of edges a node is connected to. $d_u = \sum_{v \in V} \mathbf{A}[u, v].$
- Node centrality:
  - Considers how important a node's neighbours are.
  - There are many different centrality measures (e.g. eigenvector centrality, betweenness centrality).
  - Eigenvector centrality: "we define a node's eigenvector centrality via a recurrence relation in which the node's centrality is proportional to the average centrality of its neighbours" [GRL Book]. By Perron-Frobenius Theorem the vector of centrality values is given by the eigenvector corresponding to the largest eigenvalue of **A**. Use power iteration to find that! **See GRL Book.**

$$e_u = \frac{1}{\lambda} \sum_{v \in V} \mathbf{A}[u, v] e_v \; \forall u \in \mathcal{V} \qquad \lambda \mathbf{e} = \mathbf{A} \mathbf{e}.$$

e: vector of node centralities
λ: constant

https://networkx.org/documentation/stable/reference/algorithms/centrality.html

Node-level Statistics

- Clustering coefficient:
  - Measures how tightly clustered a node's neighbourhood is.
  - Numerator: Number of edges between neighbours.
  - Denominator: Number of possible edges between the neighbours.

$$c_u = \frac{|(v_1, v_2) \in \mathcal{E} \; : \; v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}}$$

$$C_i = \frac{T_i}{\binom{d_i}{2}} = \frac{(A^3)_{ii}}{d_i(d_i - 1)}$$

https://networkx.org/documentation/stable/reference/algorithms/centrality.html

Graph-level Statistics

- Aggregate node-level statistics (e.g. histograms, means).

Edge Prediction

- Assume we know a subset of all edges.

Edge Prediction

- Assume we know a subset of all edges.
- The features discussed so far are not very useful for the task of relation prediction.

Edge Prediction

- Assume we know a subset of all edges.
- The features discussed so far are not very useful for the task of relation prediction.
- For edge prediction we use **neighbourhood overlap measures**.

Edge Prediction

- Assume we know a subset of all edges.
- The features discussed so far are not very useful for the task of relation prediction.
- For edge prediction we use **neighbourhood overlap measures**.

- Count the number of neighbours that two nodes share or a normalised version of that!

$$\mathbf{S}[u,v] = |\mathcal{N}(u) \cap \mathcal{N}(v)| \qquad \mathbf{S}_{\text{Sorenson}}[u,v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{d_u + d_v} \qquad \mathbf{S}_{\text{Salton}}[u,v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{\sqrt{d_u d_v}}$$

Edge Prediction

- Assume we know a subset of all edges.
- The features discussed so far are not very useful for the task of relation prediction.
- For edge prediction we use **neighbourhood overlap measures**.

- Count the number of neighbours that two nodes share or a normalised version of that!

$$\mathbf{S}[u,v] = |\mathcal{N}(u) \cap \mathcal{N}(v)| \qquad \mathbf{S}_{\text{Sorenson}}[u,v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{d_u + d_v} \qquad \mathbf{S}_{\text{Salton}}[u,v] = \frac{2|\mathcal{N}(u) \cap \mathcal{N}(v)|}{\sqrt{d_u d_v}}$$

- We can also consider the importance of the common neighbors! (Resource Allocation, Adamic-Adar Index)

$$\mathbf{S}_{\text{RA}}[v_1,v_2] = \sum_{u \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)} \frac{1}{d_u} \qquad \mathbf{S}_{\text{AA}}[v_1,v_2] = \sum_{u \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)} \frac{1}{\log(d_u)}$$

Edge Prediction

- There also exist global overlap measures like the Katz index that count the number of paths of all lengths between a pair of nodes.

$$\mathbf{S}_{\text{Katz}}[u,v] = \sum_{i=1}^{\infty} \beta^i \mathbf{A}^i[u,v]$$

β is a hyperparameter controlling how much weight is given to short versus long paths

- A theorem allows us to calculate that.

$$\mathbf{S}_{\text{Katz}} = (\mathbf{I} - \beta\mathbf{A})^{-1} - \mathbf{I}$$

The Problem with Traditional Approaches

- “The approaches discussed are limited due to the fact that **they require careful, hand-engineered statistics** and measures. These hand-engineered features are **inflexible**—i.e., they cannot adapt through a learning process—and designing these features can be a **time-consuming and expensive** process.” GRL Book

Degree Normalisation

- If we use the standard adjacency matrix for ML/DL the nodes with high degrees will “overwhelm” the model. We need to do some type of normalization!

$$\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

- What does the above calculation do exactly? Let’s look at an example.

Degree Normalisation

$$D^{-\frac{1}{2}}A.$$

- The **left** multiplication multiplies each **row** by the number in the diagonal. Here, we divide each edge by the (sqrt) degree of the **left** node.

$D^{-1/2}$

0.58	0	0	0	0	0	0	0
0	0.58	0	0	0	0	0	0
0	0	0.5	0	0	0	0	0
0	0	0	0.5	0	0	0	0
0	0	0	0	0.5	0	0	0
0	0	0	0	0	0.5	0	0
0	0	0	0	0	0	0.58	0
0	0	0	0	0	0	0	0.58

Adjacency Matrix

0	1	1	0	0	0	0	1
1	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0
0	1	1	0	1	1	0	0
0	0	1	1	0	1	1	0
0	0	0	1	1	0	1	1
0	0	0	0	1	1	0	1
1	0	0	0	0	1	1	0

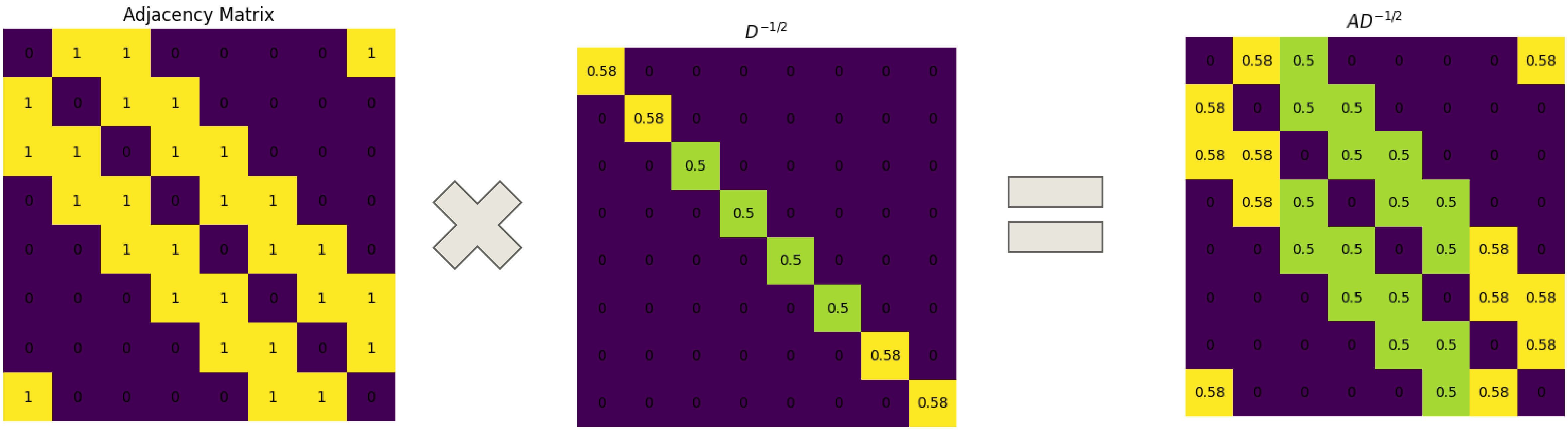
$D^{-1/2}A$

0	0.58	0.58	0	0	0	0	0.58
0.58	0	0.58	0.58	0	0	0	0
0.5	0.5	0	0.5	0.5	0	0	0
0	0.5	0.5	0	0.5	0.5	0	0
0	0	0.5	0.5	0	0.5	0.5	0
0	0	0	0.5	0.5	0	0.5	0.5
0	0	0	0	0.58	0.58	0	0.58
0.58	0	0	0	0	0.58	0.58	0

Degree Normalisation

$$AD^{-\frac{1}{2}}$$

- The **right** multiplication multiplies each **column** by the number in the diagonal. Here, we divide each edge by the (sqrt) degree of the **right** node.



Degree Normalisation

- So, this is equivalent to $$\frac{A_{ij}}{\sqrt{d_i}\sqrt{d_j}}$$

$D^{-1/2}AD^{-1/2}$

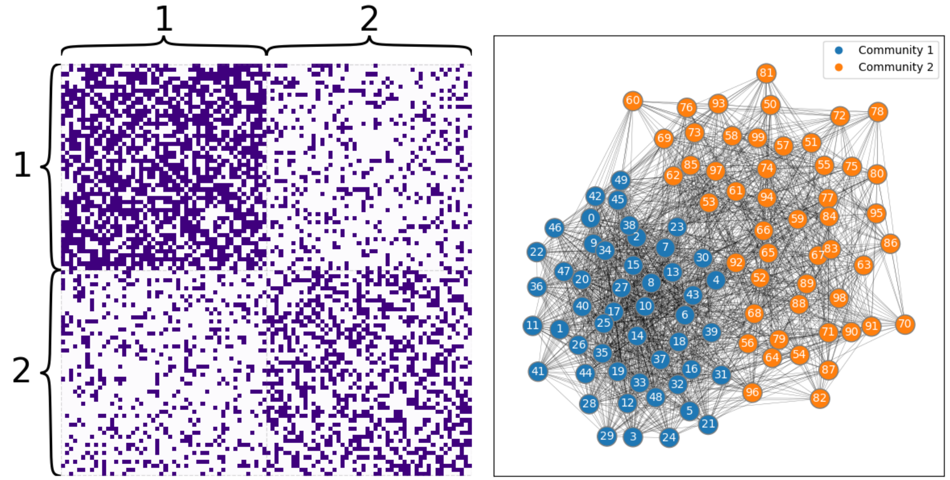
$$\begin{bmatrix}
0 & 0.33 & 0.29 & 0 & 0 & 0 & 0 & 0.33 \\
0.33 & 0 & 0.29 & 0.29 & 0 & 0 & 0 & 0 \\
0.29 & 0.29 & 0 & 0.25 & 0.25 & 0 & 0 & 0 \\
0 & 0.29 & 0.25 & 0 & 0.25 & 0.25 & 0 & 0 \\
0 & 0 & 0.25 & 0.25 & 0 & 0.25 & 0.29 & 0 \\
0 & 0 & 0 & 0.25 & 0.25 & 0 & 0.29 & 0.29 \\
0 & 0 & 0 & 0 & 0.29 & 0.29 & 0 & 0.33 \\
0.33 & 0 & 0 & 0 & 0 & 0.29 & 0.33 & 0
\end{bmatrix}$$

Sneak Peak for Next Week 👀

- What about node clustering?
- What about learning embeddings for each node?
- Graph Laplacians and Spectral Methods
- Unnormalised Laplacian Matrix: $\mathbf{L} = \mathbf{D} - \mathbf{A}$

D is the degree matrix (diagonal entries, diagonal matrix)

A, a sample from an $SBM_n(z, B)$ Simulation



https://docs.neurodata.io/graph-stats-book/representations/ch6/spectral-embedding.html

Key Takeaways

- We can include inductive biases to improve our models.
- Graphs are everywhere!
- The adjacency matrix of a matrix has many cool properties.
- For traditional ML, graph tasks require hand-crafter features which are time-consuming and inflexible.

THANK YOU

QUESTIONS?