Lecture 15:

Graph Neural Networks (GNN)

Markus Hohle

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Course Map

Lecture 1: Course Overview and Introduction to Machine Learning

classic ML tools & algorithms

Lecture 2: Bayesian Methods in Machine Learning

Lecture 3: Dimensionality Reduction: Principal Component Analysis

Lecture 4: Linear and Non-linear Regression and Classification

Lecture 5: Unsupervised Learning: K-Means, GMM, Trees

Lecture 6: Adaptive Learning and Gradient Descent Optimization Algorithms

ANNs/AI/Deep Learning

Lecture 7: Introduction to Artificial Neural Networks - The Perceptron

Lecture 8: Introduction to Artificial Neural Networks - Building Multiple Dense Layers

Lecture 9: Convolutional Neural Networks (CNNs) - Part I

Lecture 10: CNNs - Part II

Lecture 11: Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTMs)

Lecture 12: Combining LSTMs and CNNs

Lecture 15: Transformer

Lecture 14: Project Presentations

Lecture 15: GNN

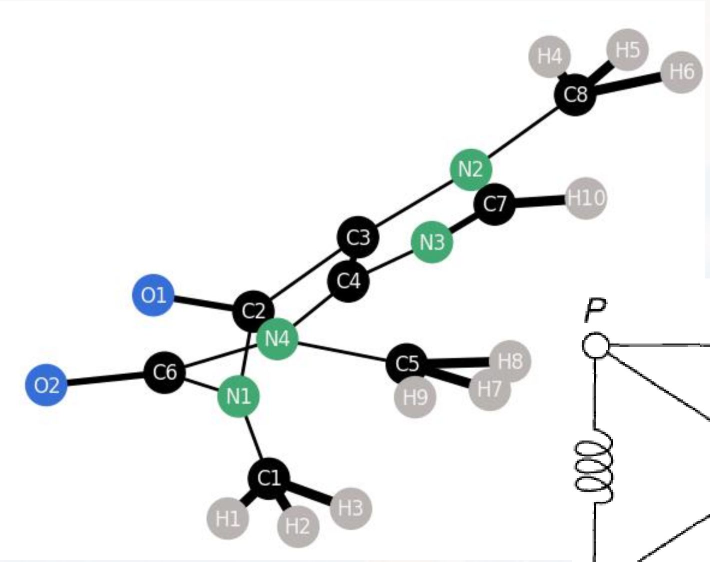
Outline

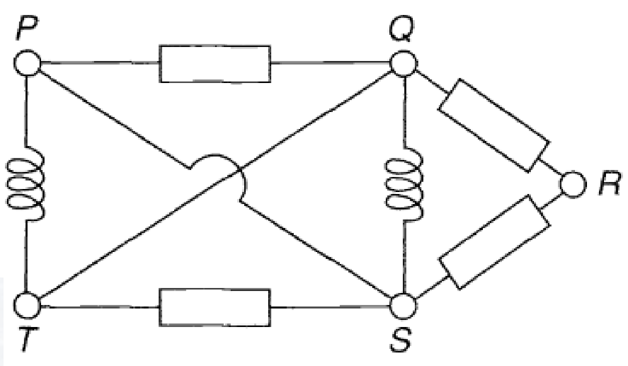
- What is a Graph
- The ANN Part
- PyTorch Example

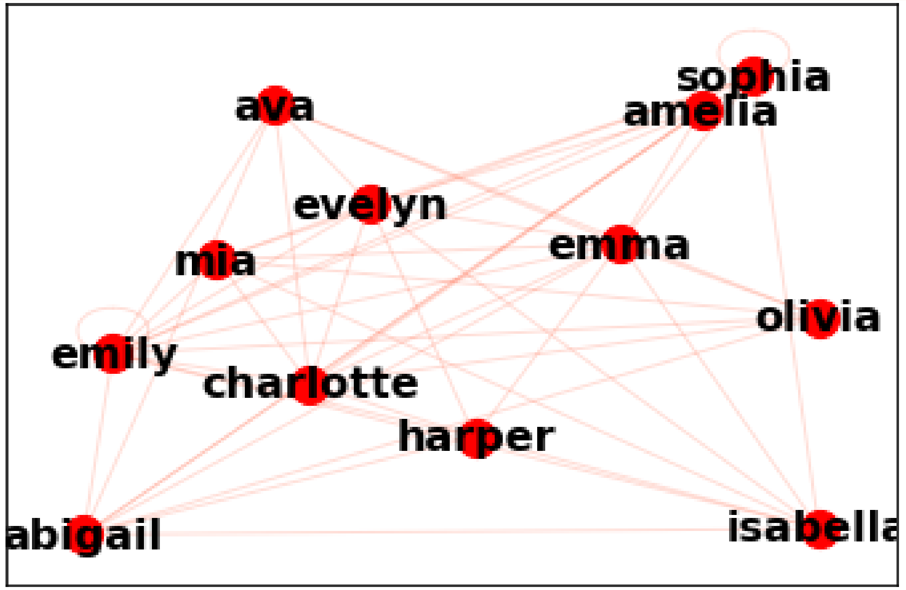
Outline

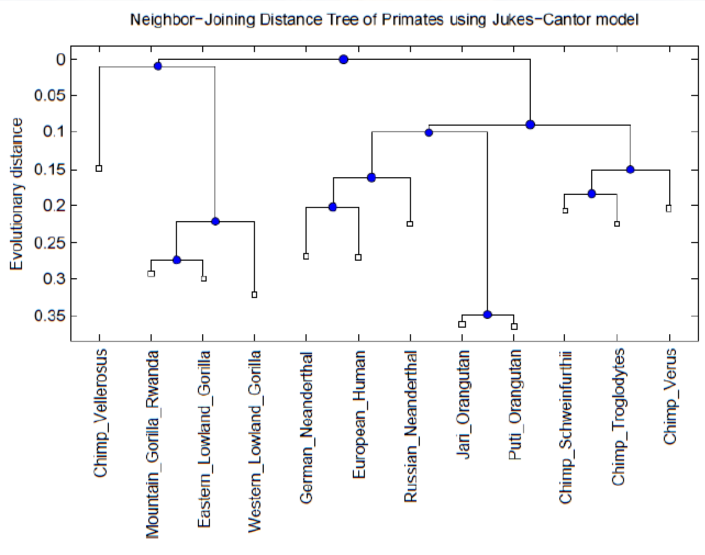
- **What is a Graph**
- The ANN Part
- PyTorch Example

GNN: What is a Graph

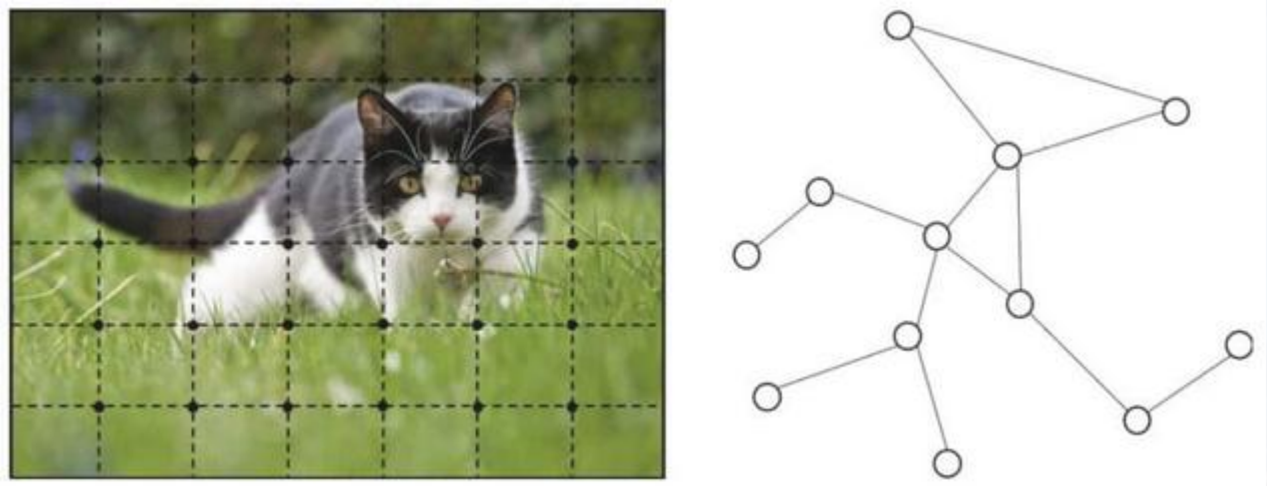


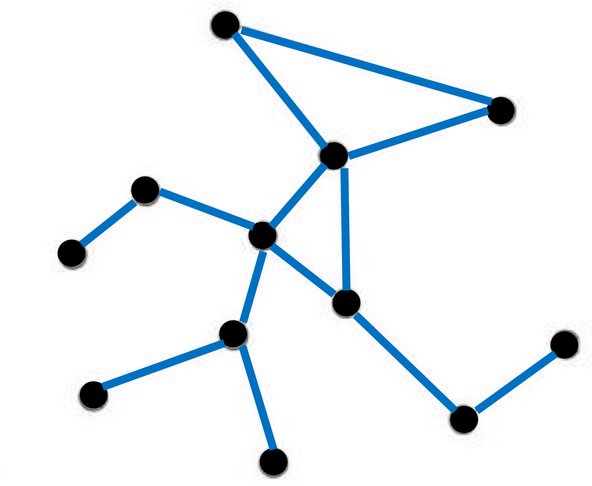






https://doi.org/10.1016/j.aiopen.2021.01.001





Graph G

nodes N (vertices V)

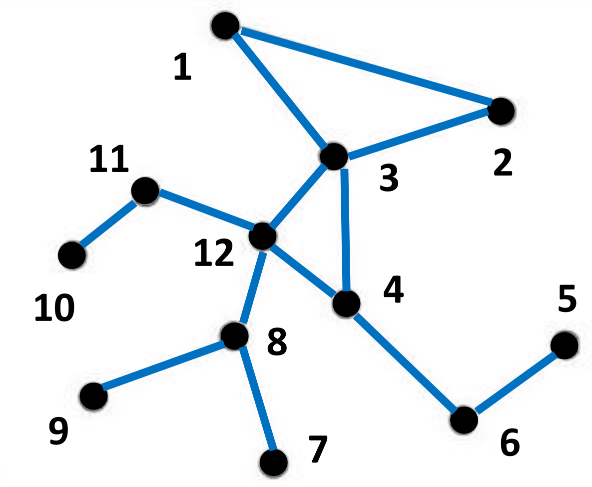
edges E

$G = G(N, E)$

- social networks
- street maps
- workflows/planning
- biological signal pathways
- image processing

- nodes can have **features**
  - molecules: mass/ electronegativity
  - people: age, income, sex, …
- edges can have **attributes**
  - molecules: bond length/strength
  - people: relations (work, friend, family)

GNN: What is a Graph



structural information: **adjacency matrix A**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

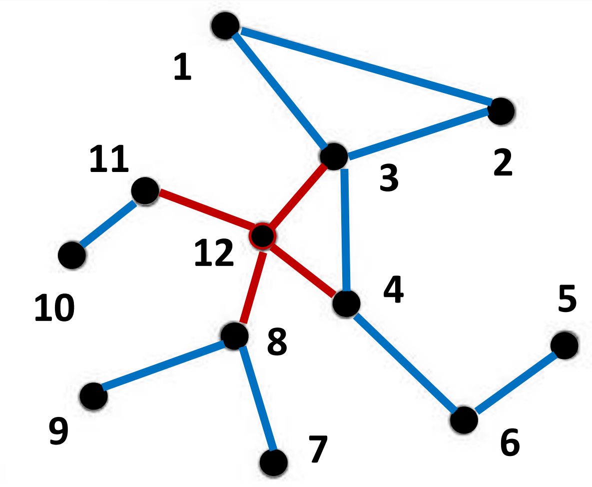
Graph $G = G(N, E)$

nodes N (vertices V)

edges E

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

GNN:



structural information: **adjacency matrix A**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

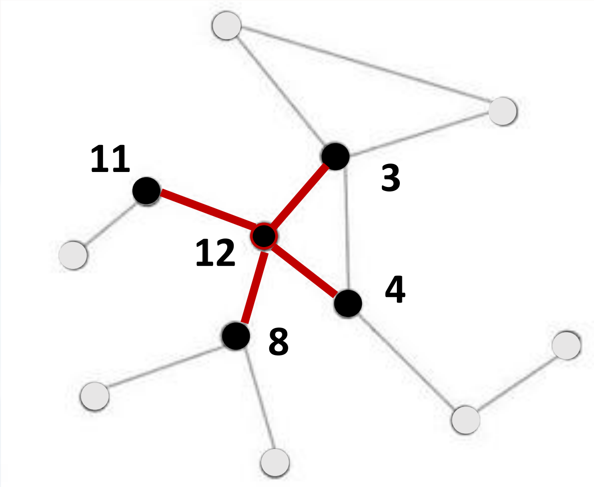
node 12 has four first degree neighbors

degree *d* of a node

$$d(n_i) = \sum_j A_{ij}$$

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

GNN: What is a Graph



structural information: **adjacency matrix A**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)

edges E

node 12 has four first degree neighbors

degree *d* of a node

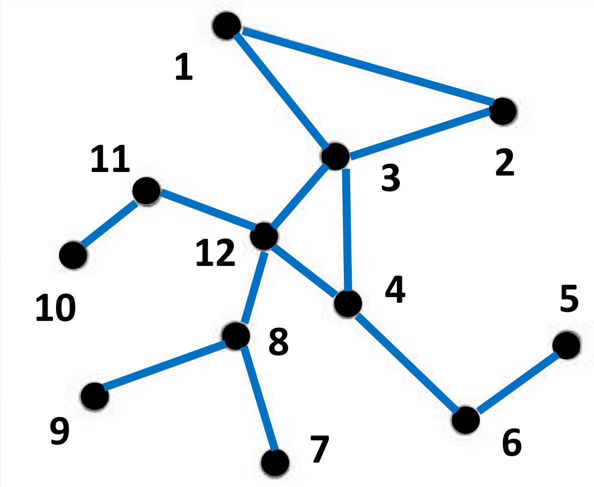
$$d(n_i) = \sum_j A_{ij}$$

first degree neighborhood $\mathscr{N}$

$$\mathscr{N}(n_i) = \{n_j \epsilon N: (n_i, n_j) \in E\}$$

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

GNN: What is a Graph



structural information: **adjacency matrix A**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

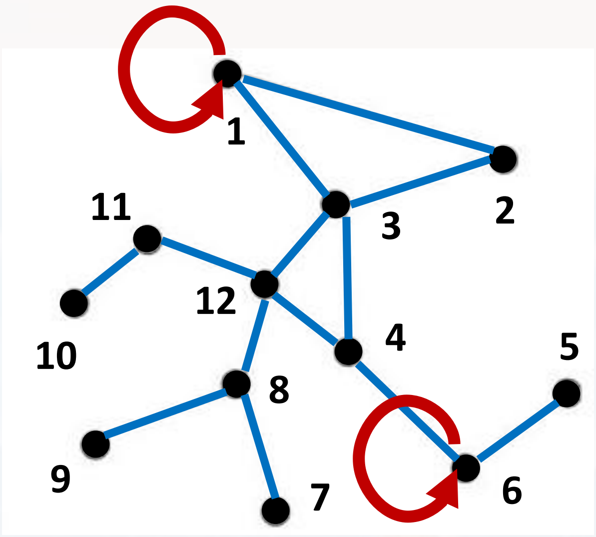
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

A graph can have **loops**

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



structural information: **adjacency matrix A**

Graph $G = G(N, E)$

nodes N (vertices V)

edges E

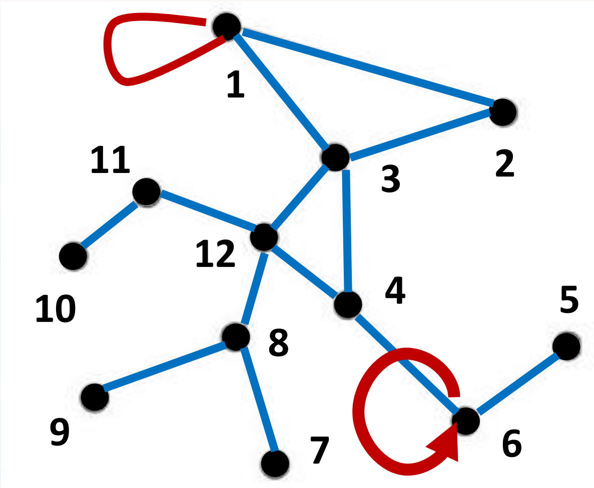
$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

A graph can have **loops**

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



structural information: **adjacency matrix $\boldsymbol{A}$**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

Graph $G = G(N, E)$

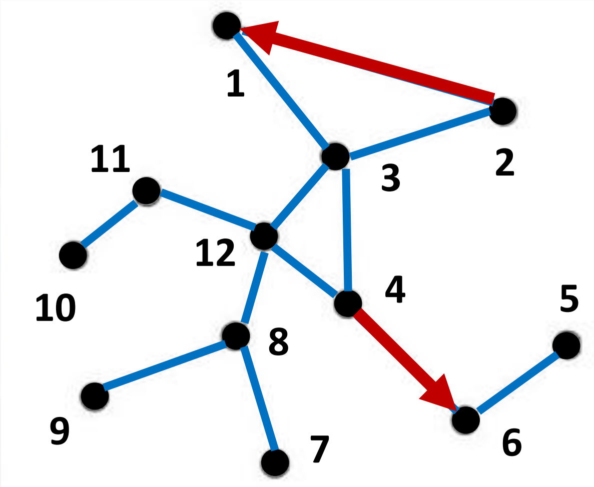
nodes N (vertices V)
edges E

A graph can have **loops**

note:

$d(n_1) = 4$, since loop is **undirected** and hits the node twice!

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$



structural information: **adjacency matrix $\boldsymbol{A}$**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

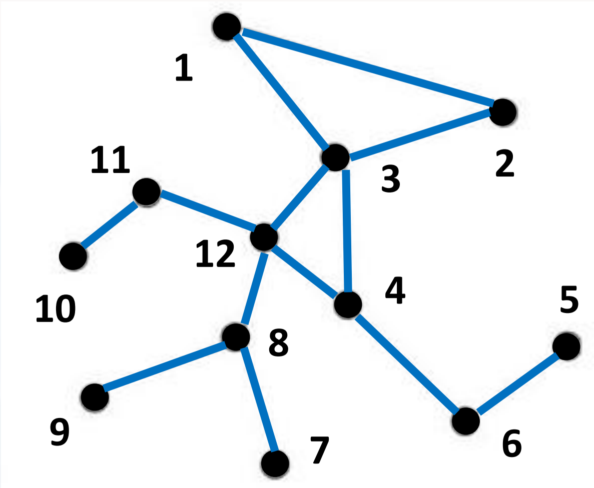
Graph $G = G(N, E)$

nodes N (vertices V)
edges E

A graph can be **directed**

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

GNN: What is a Graph



structural information: **adjacency matrix $\boldsymbol{A}$**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

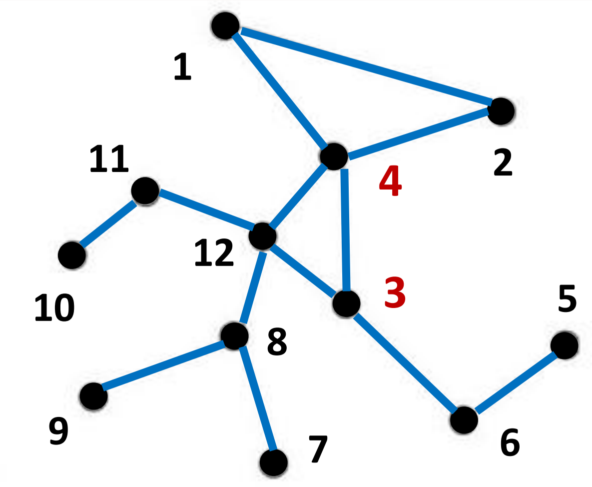
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

The order of counting the nodes is not relevant!
(**permutation invariance**)

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



structural information: **adjacency matrix A**

$A_{ij} = 1$ if $(n_i, n_j) \in E$

(nodes n_i and n_j have a common edge)

$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

The order of counting the nodes is not relevant!
(**permutation invariance**)

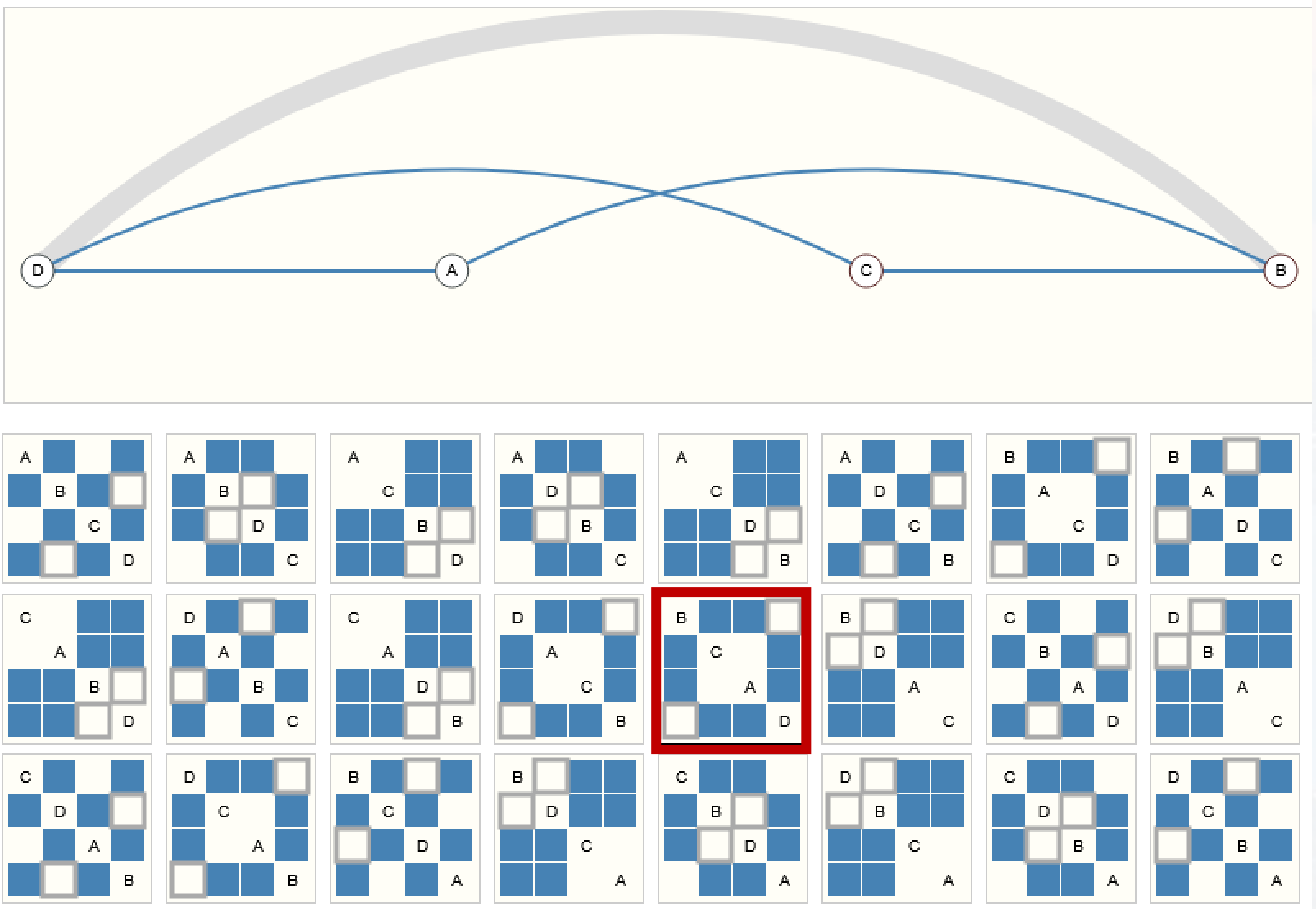
Each graph can be represented by **N!** adjacency matrices!

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

GNN: What is a Graph

Each graph can be represented by **N!** adjacency matrices!

animation here

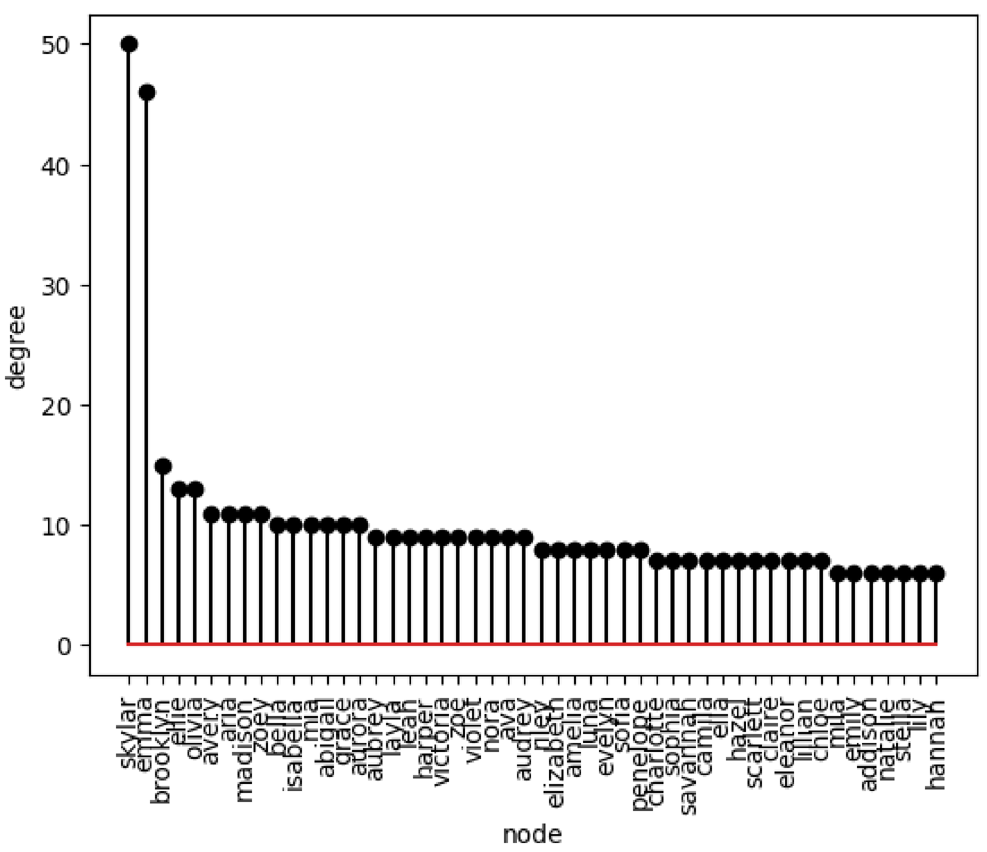


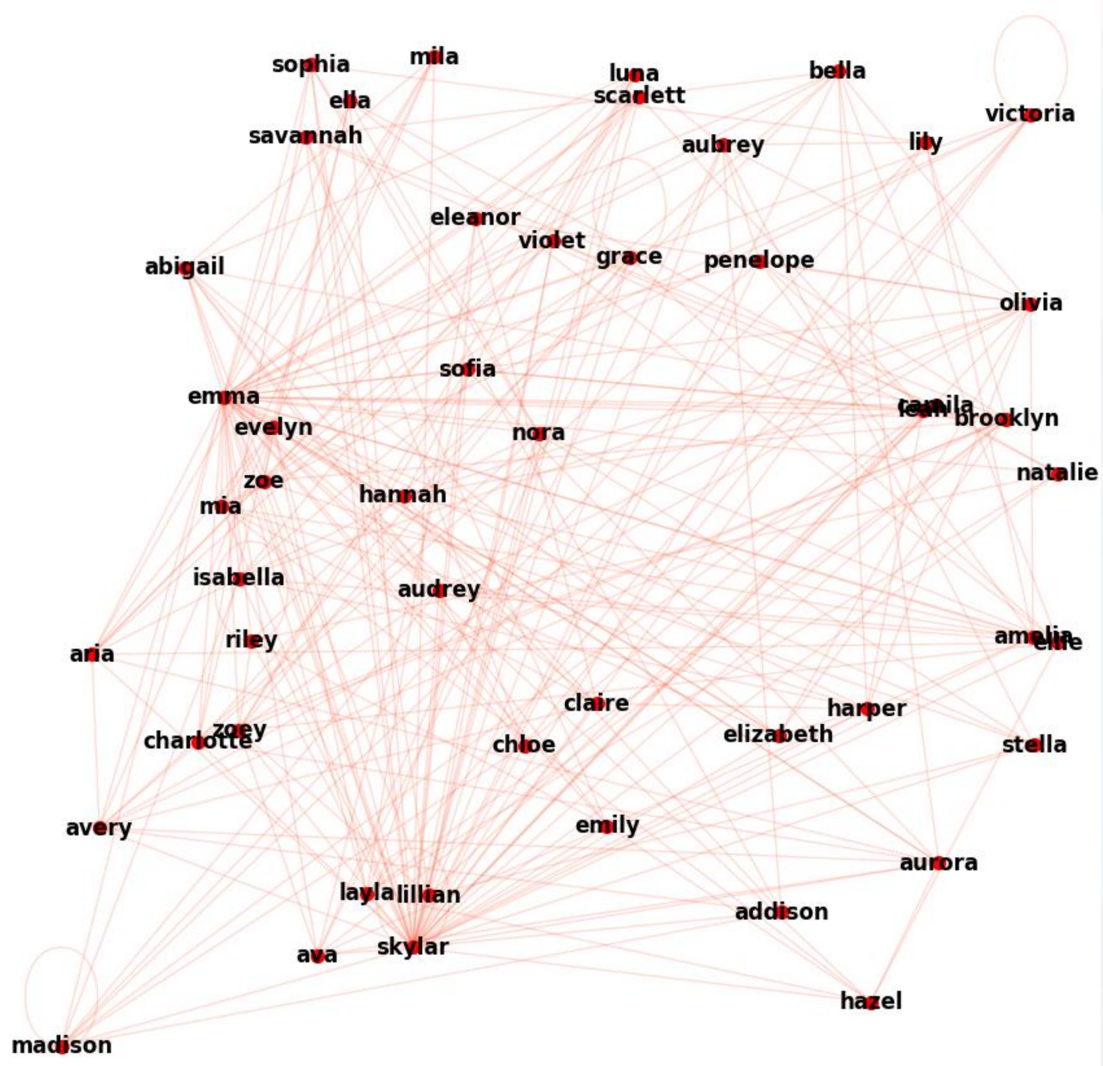
GNN:

visualizing a graph:

```
import networkx as nx #pip install networkx
```

see: `Graph_I.ipynb`



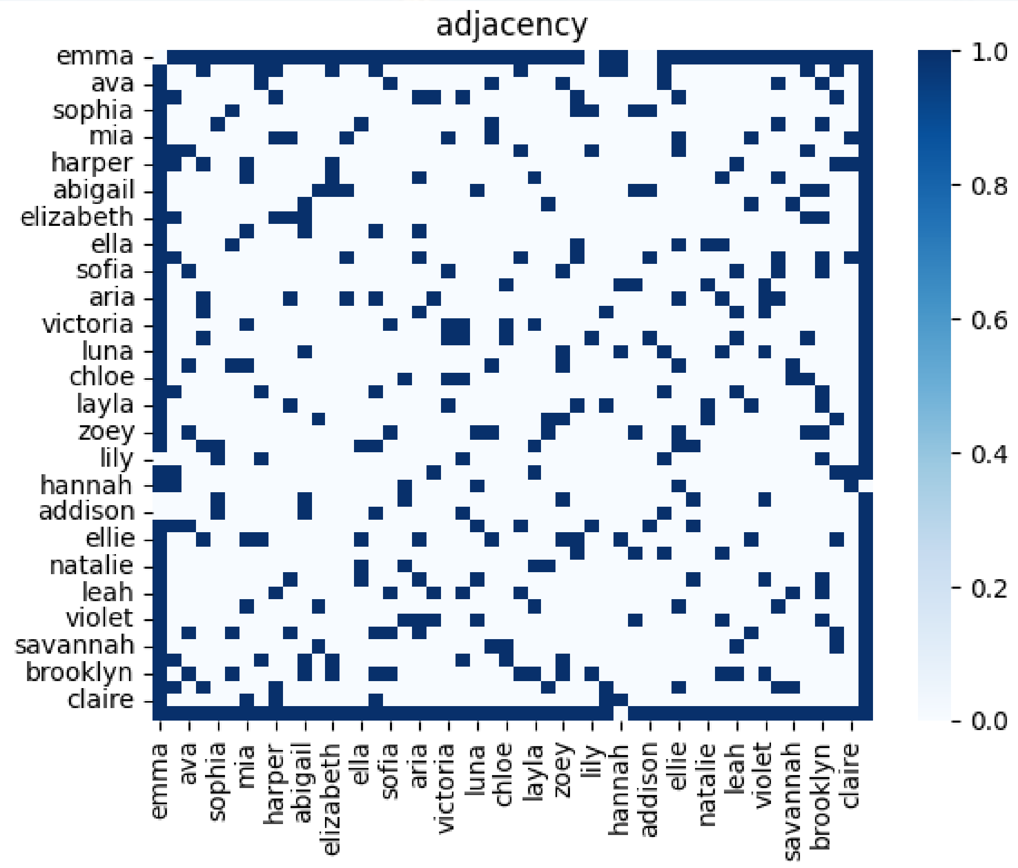


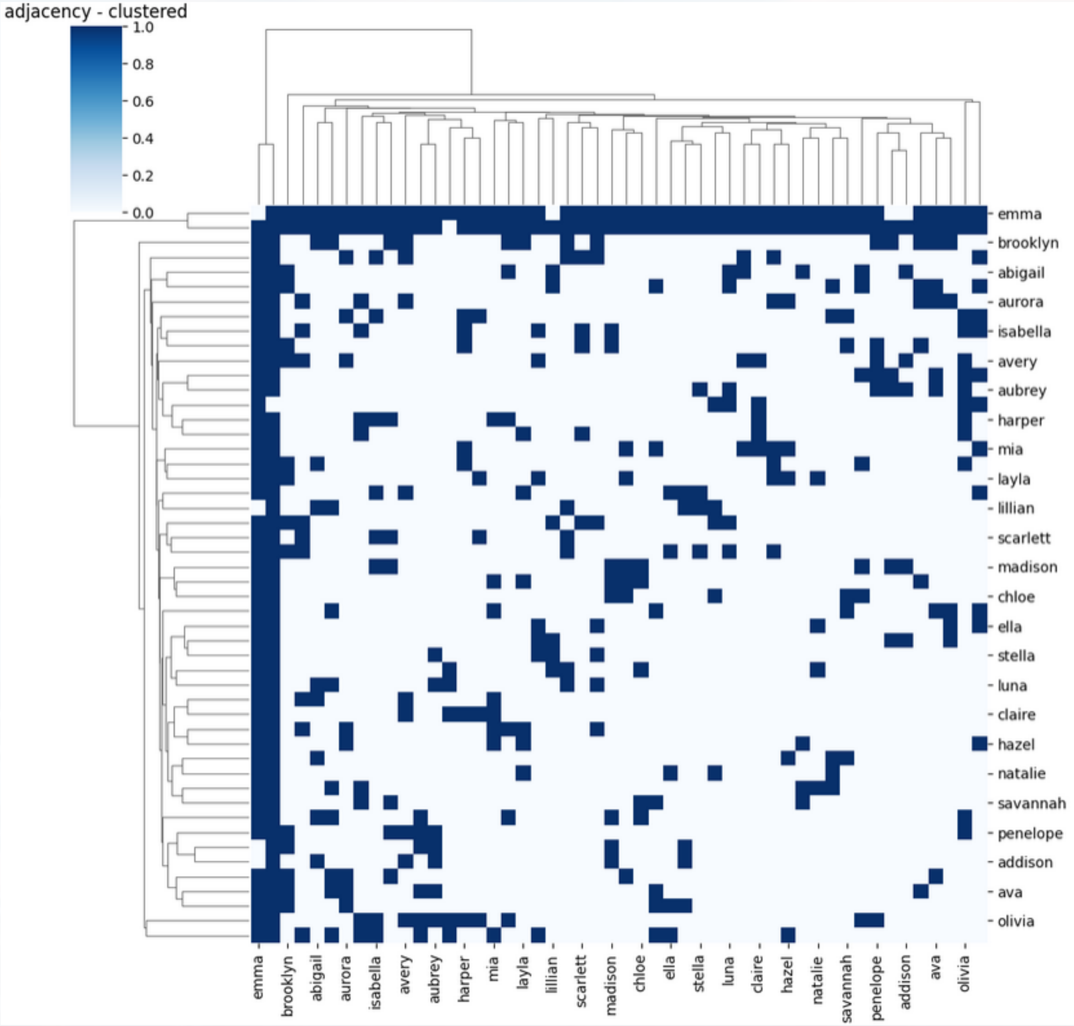
GNN:

visualizing a graph:

```
import networkx as nx #pip install networkx
```

see: `Graph_I.ipynb`



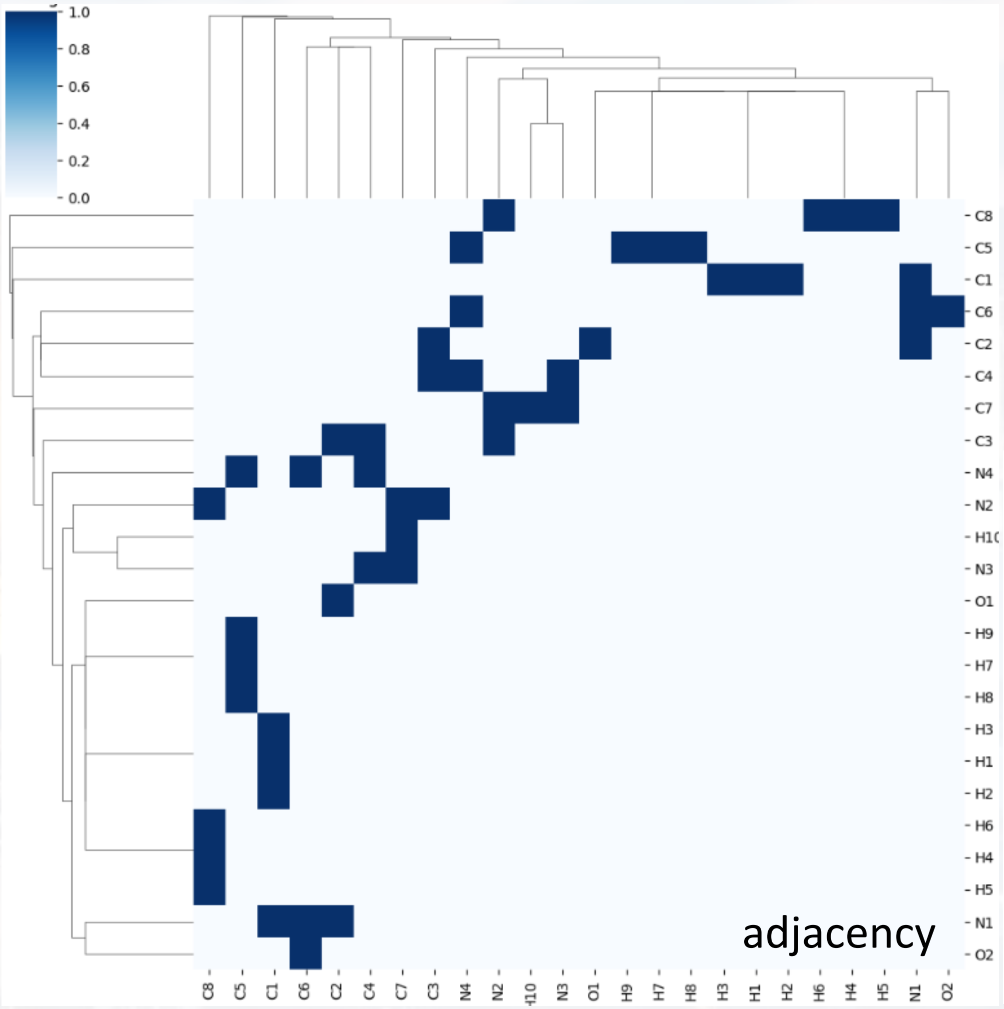


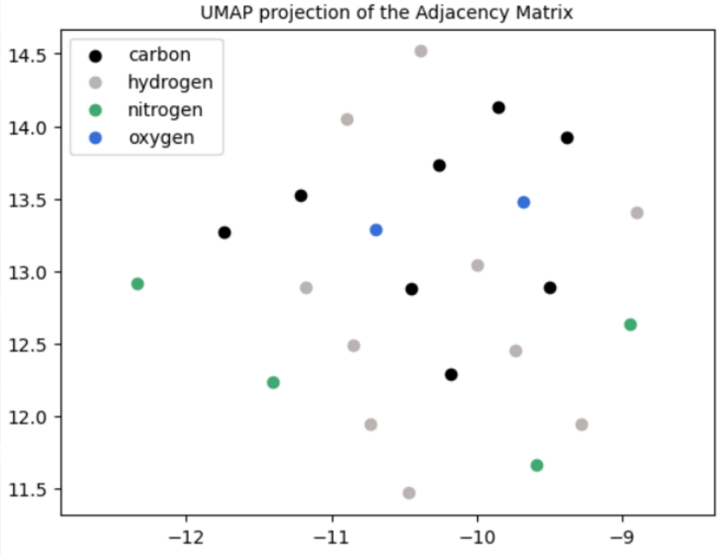
building and visualizing a **weighted** graph:

```
import networkx as nx #pip install networkx
```

see: `Graph_II.ipynb`

Caffein molecule





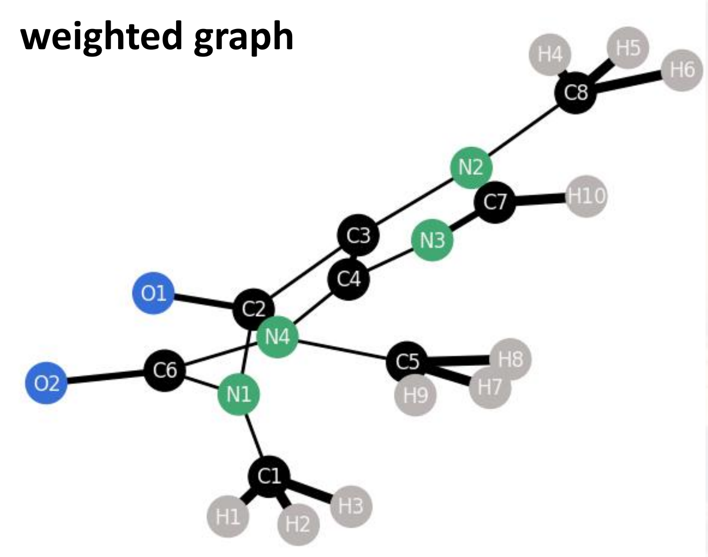
GNN:

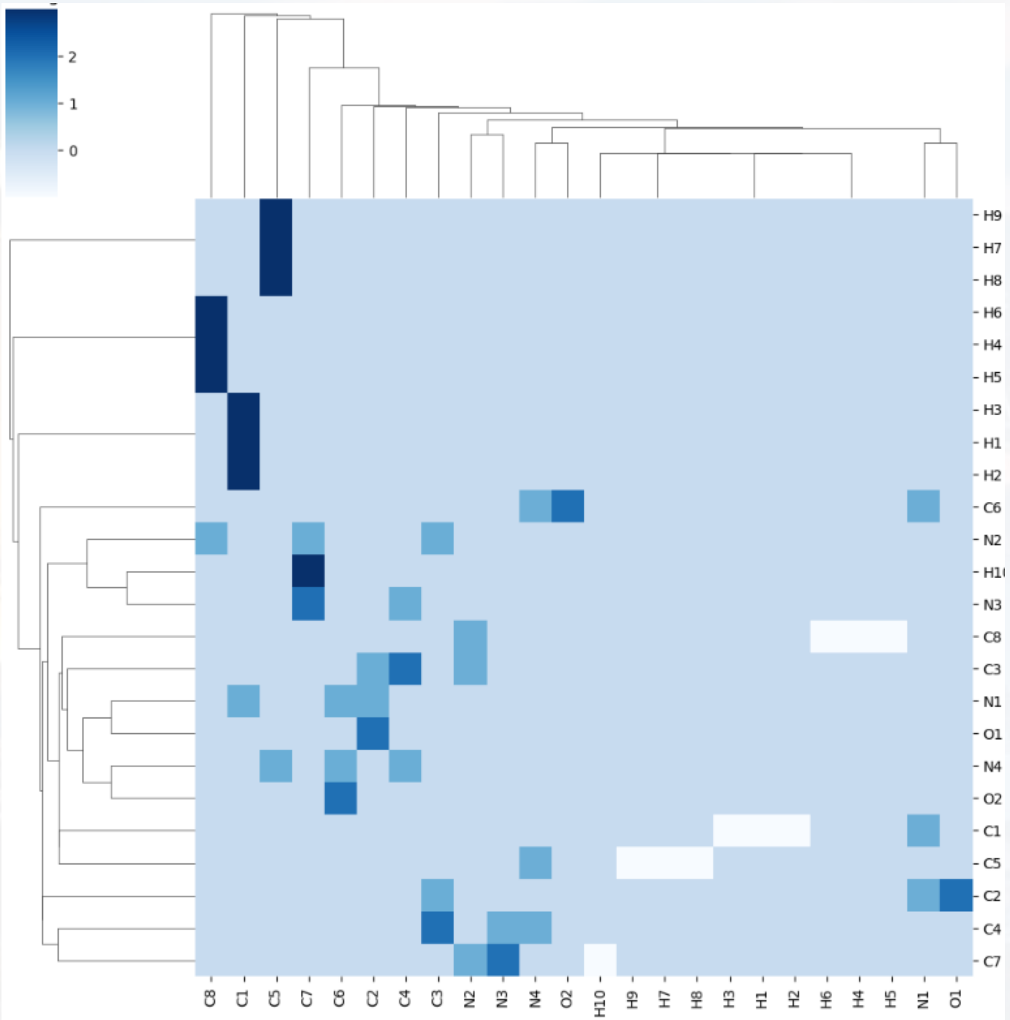
building and visualizing a **weighted** graph:

```
import networkx as nx #pip install networkx
```

see: `Graph_II.ipynb`

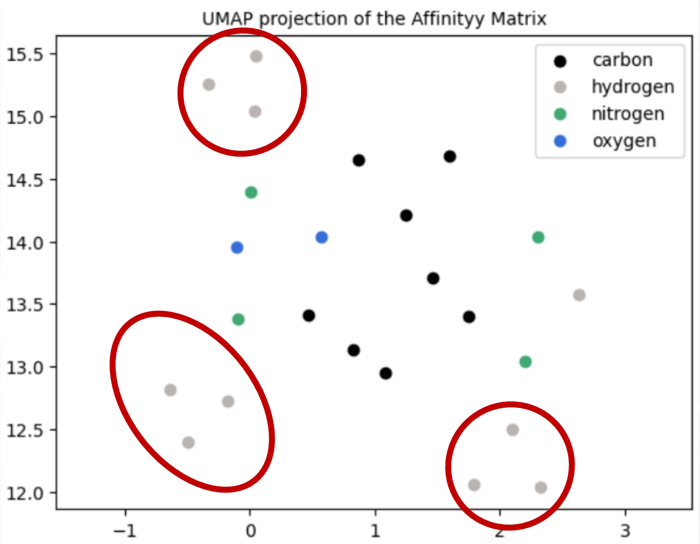
Caffein molecule





binding affinity (weights)

hydrogen atoms are at the edges of the molecule!



more about graphs:

$$d(n_i) = \sum_j A_{ij}$$

degree of node n_i

$$\mathcal{N}(n_i) = \{n_j \in N : (n_i, n_j) \in E\}$$

neighborhood $\mathcal{N}(n_i)$ of node n_i
for first degree neighborhood $|\mathcal{N}(n_i)| = d(n_i)$

$$S_{com} = \left|\mathcal{N}(n_i) \cap \mathcal{N}(n_j)\right|$$

number for neighbors nodes n_i and n_j have in common.

idea: nodes with many common neighbors are more likely to be similar or have a potential connection.

$$S_{rat} = \frac{\left|\mathcal{N}(n_i) \cap \mathcal{N}(n_j)\right|}{\left|\mathcal{N}(n_i) \cup \mathcal{N}(n_j)\right|}$$

ratio for neighbors nodes n_i and n_j have in common.

note:
There are more quantities (“importance”, “centrality” etc.), but they are all a function of A_{ij}.

Outline

- What is a Graph

- **The ANN Part**

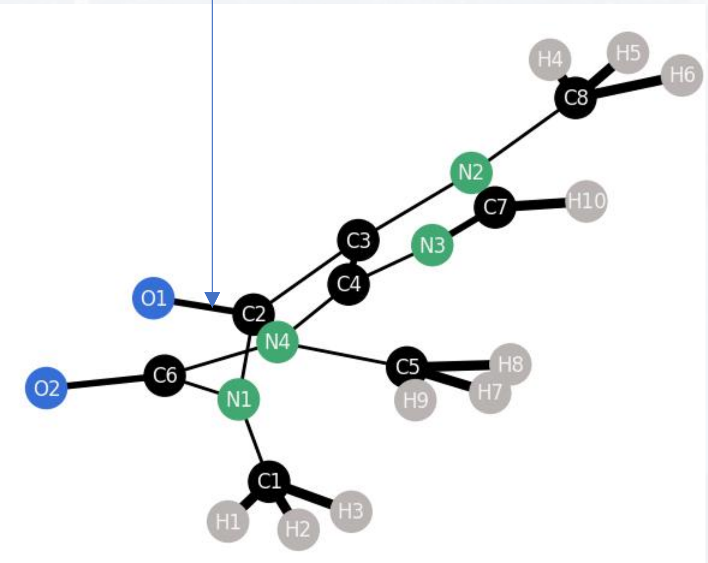
- PyTorch Example

GNN:

What we can learn:

- node classification
- join nodes with similar properties to hyper nodes
- edge attributes, weights (weighted graph)
- edge prediction
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule toxic y/n)
- graph regression (toxicity score)
- graph generation

weight: bond strength

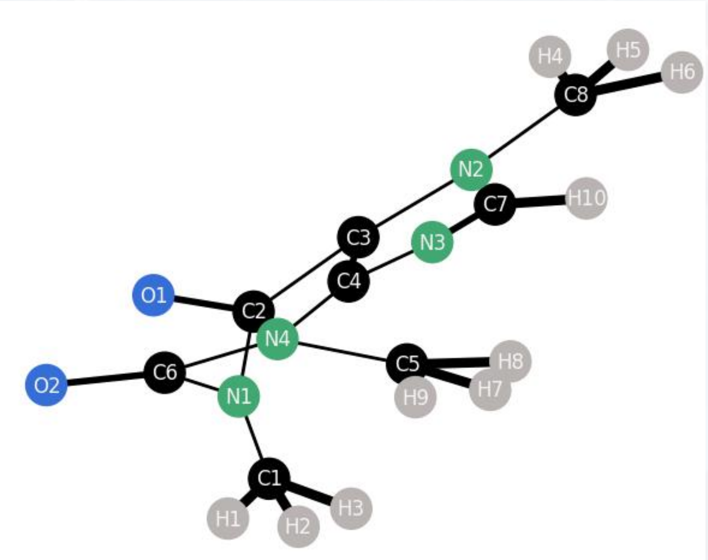


GNN:

What we can learn:

- **node classification**
- **join nodes with similar properties to hyper nodes**
- **edge attributes, weights (weighted graph)**
- **edge prediction**
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule toxic y/n)
- graph regression (toxicity score)
- graph generation

node (edge) level tasks



GNN:

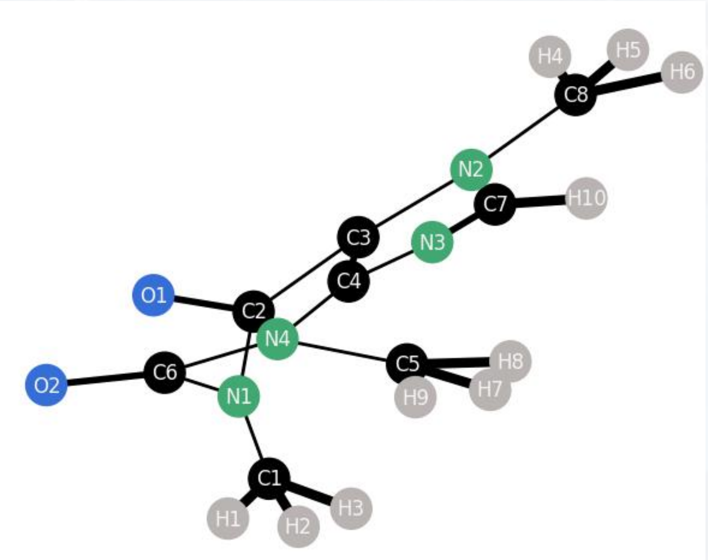
What we can learn:

- node classification
- join nodes with similar properties to hyper nodes
- edge attributes, weights (weighted graph)
- edge prediction
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule toxic y/n)
- graph regression (toxicity score)
- graph generation

information flow from one node to another:
message passing

different ways how:

- local averaging
- graph convolution (aka neighborhood aggregation)
- graph attention



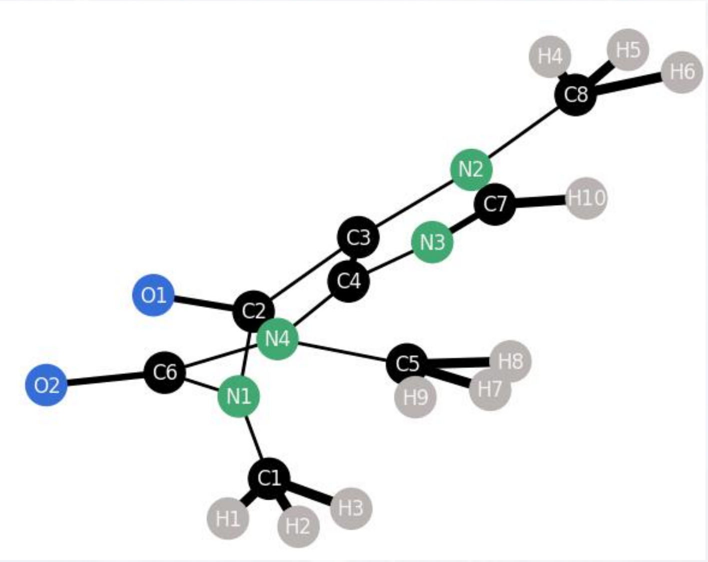
What we can learn:

- node classification
- join nodes with similar properties to hyper nodes
- edge attributes, weights (weighted graph)
- edge prediction
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule toxic y/n)
- graph regression (toxicity score)
- graph generation

information flow from one node to another:
message passing

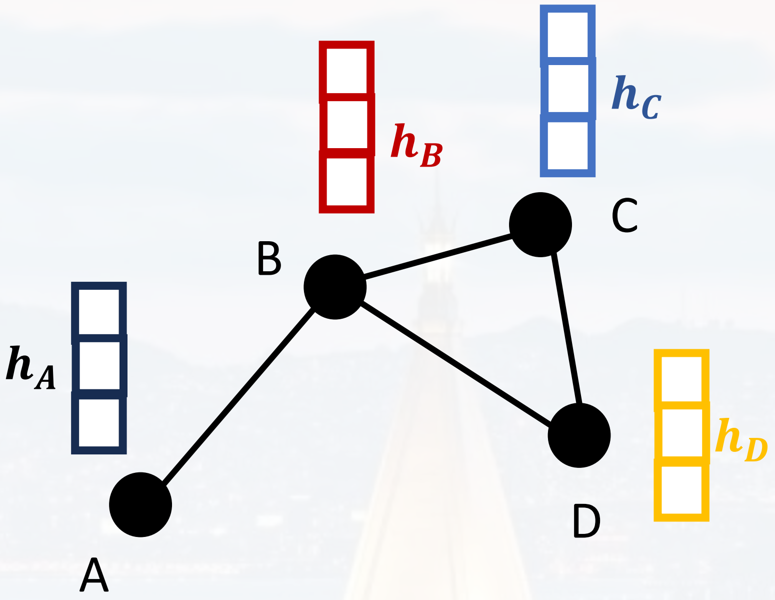
different ways how:

- local averaging
- **graph convolution** (aka neighborhood aggregation)
- **graph attention**



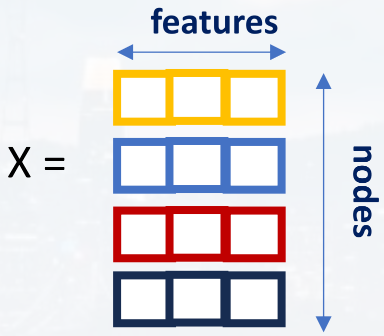
GNN: The ANN Part

Graph Convolution

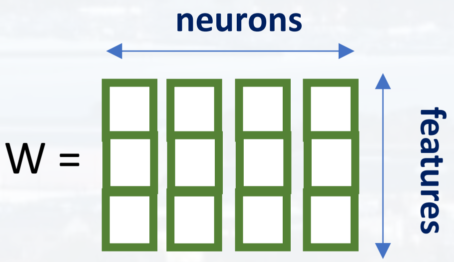


each node i has a **feature vector** h_i

matrix X of shape (number of nodes, number of node features)



weight matrix W of shape
(number of node features, number of neurons)

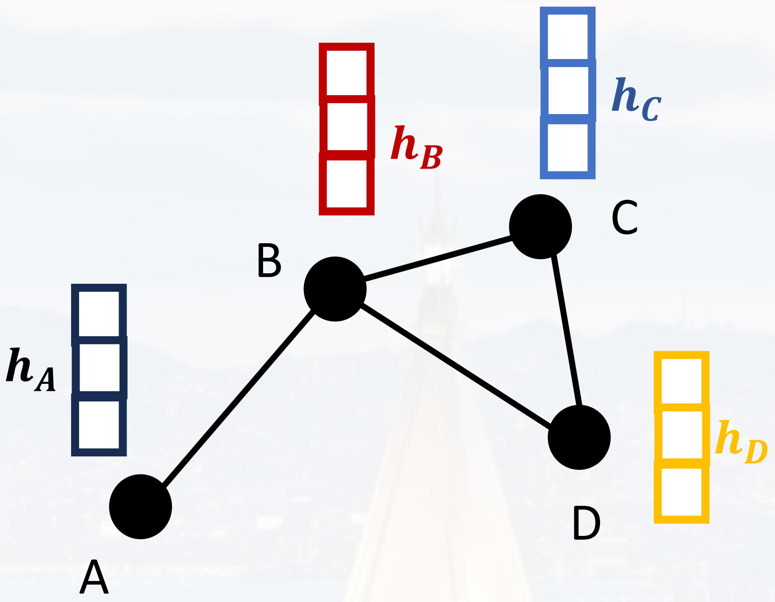


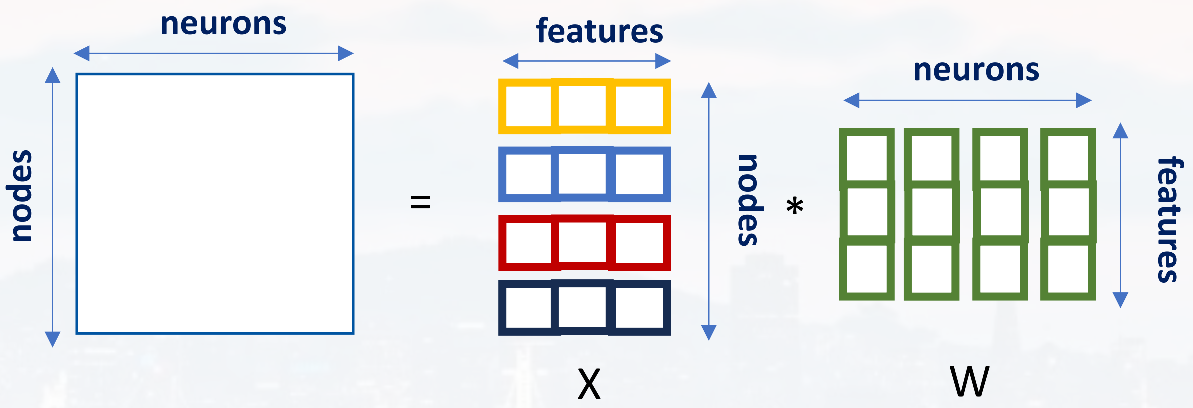
note: only **one** W for the entire graph
W is a **learnable**

GNN:

Graph Convolution

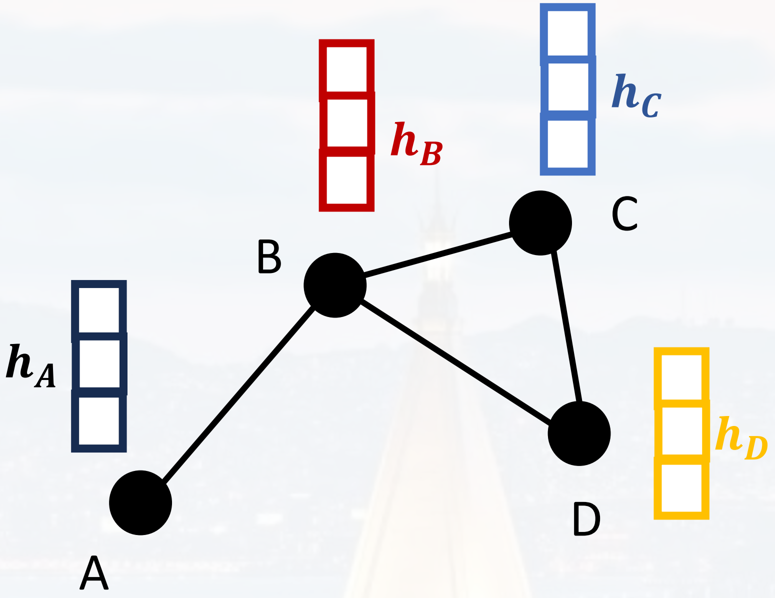
each node i has a **feature vector** h_i

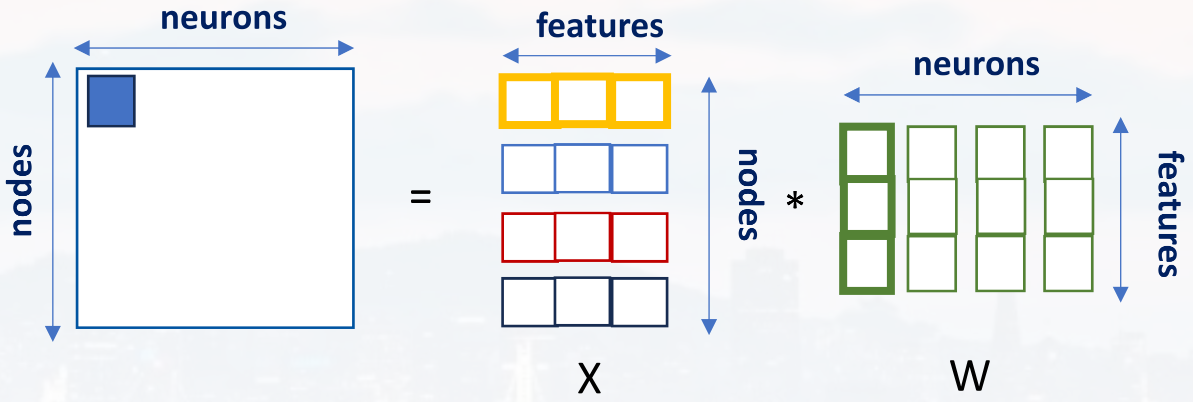




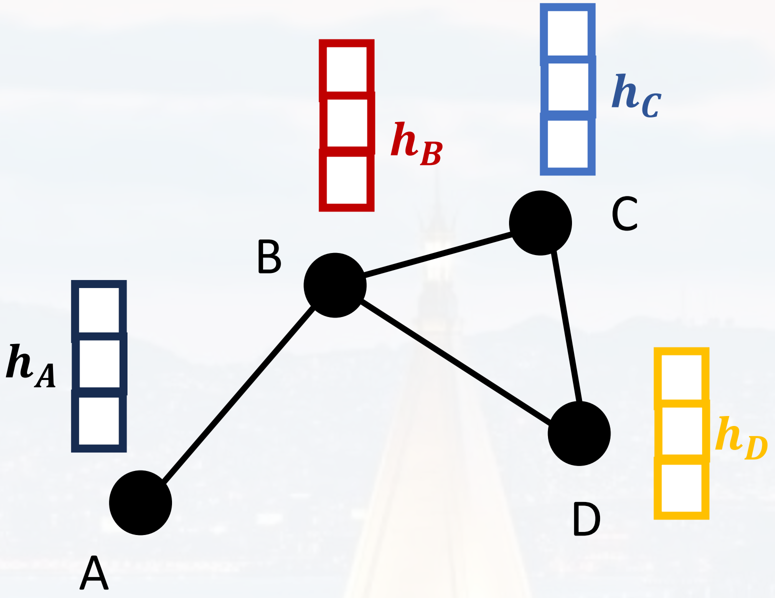
Graph Convolution

each node i has a **feature vector** h_i

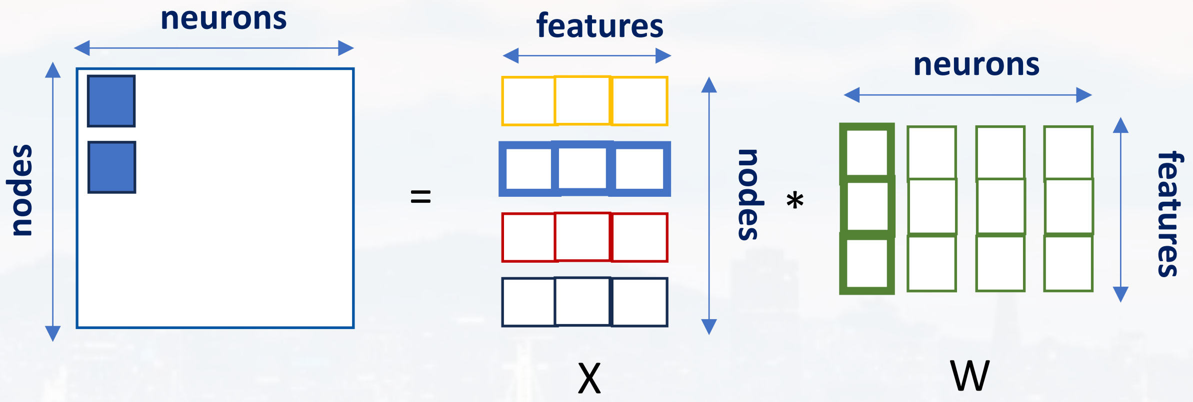




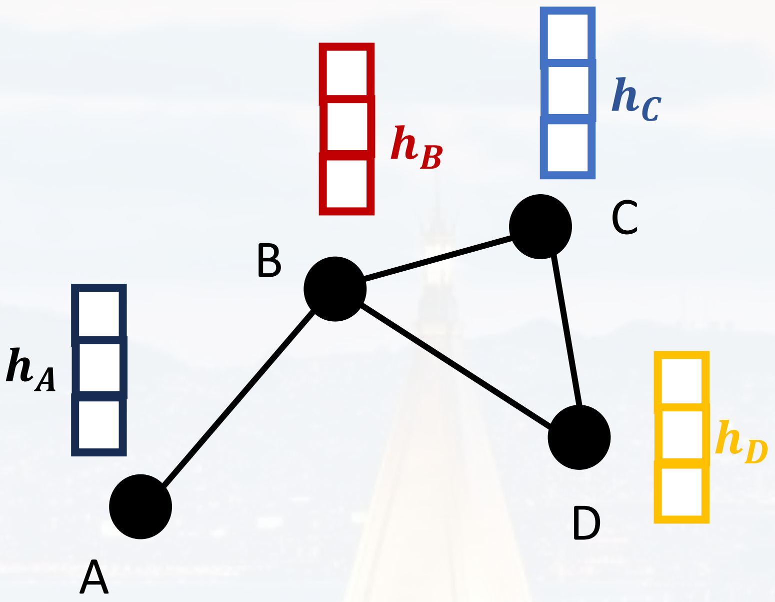
Graph Convolution



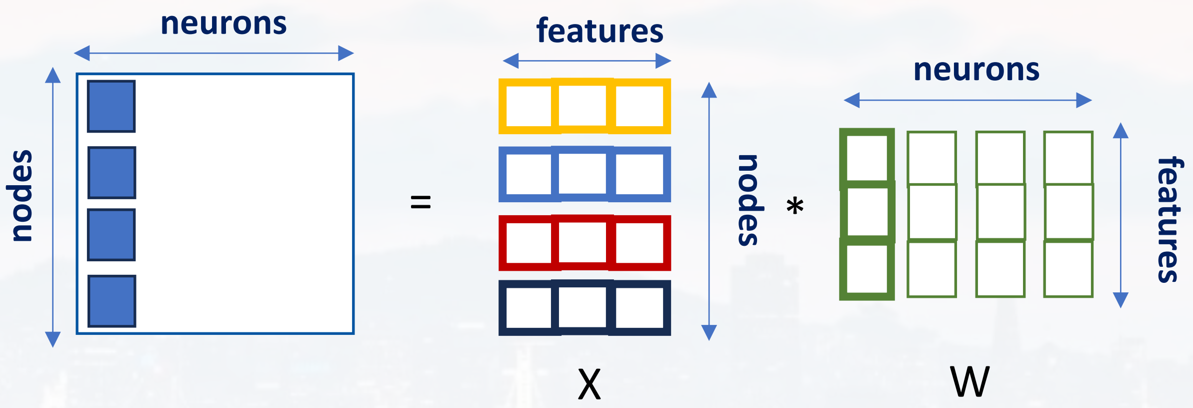
each node i has a **feature vector** h_i



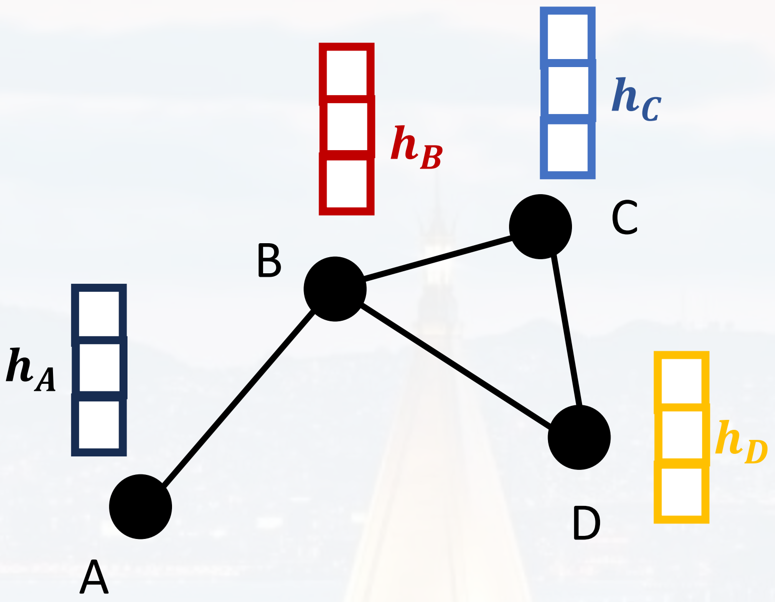
Graph Convolution



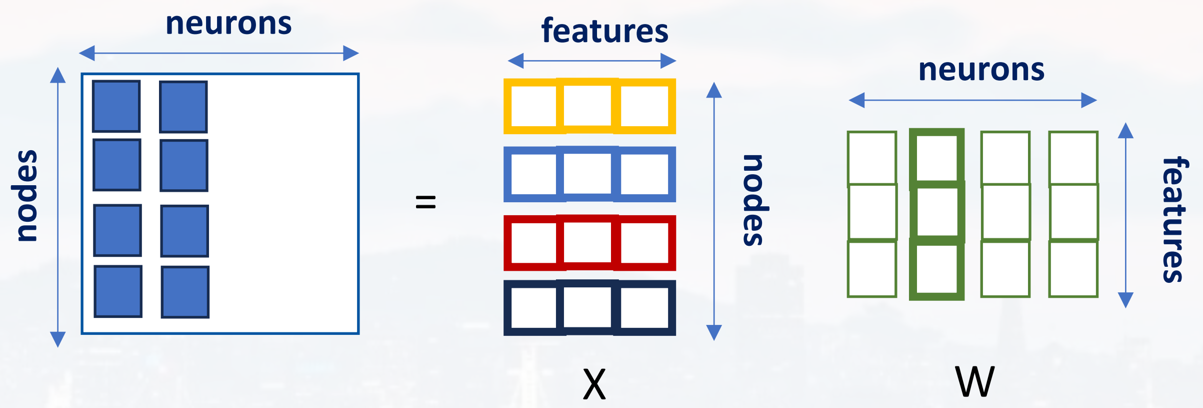
each node i has a **feature vector** h_i



Graph Convolution

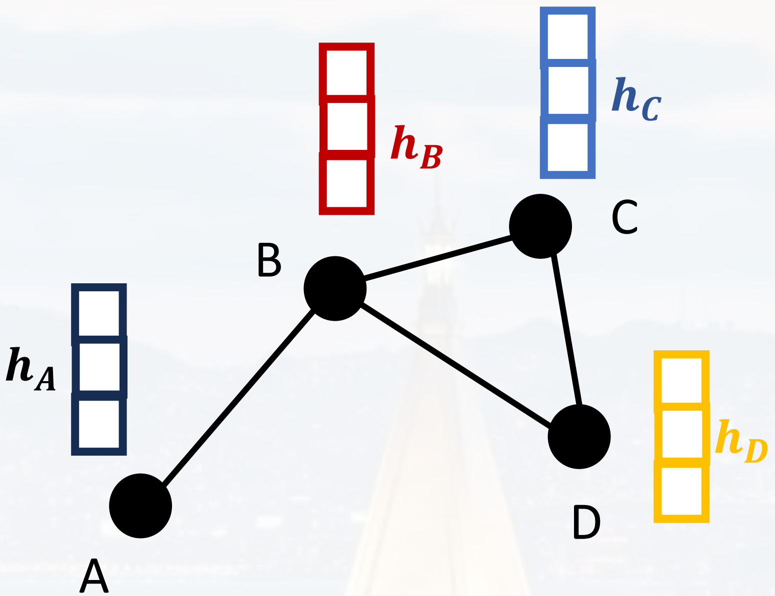


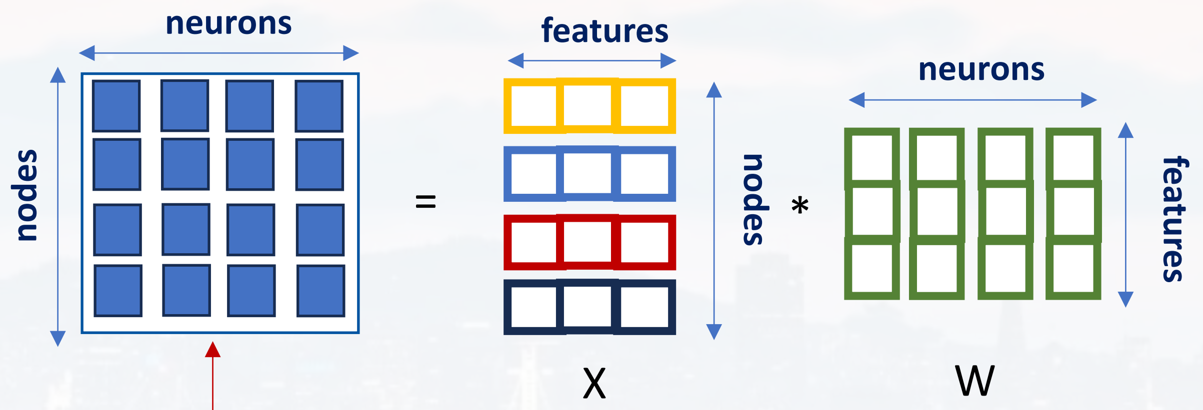
each node i has a **feature vector** h_i

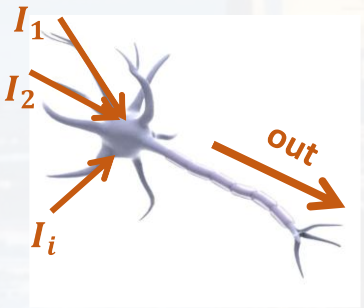


Graph Convolution

each node i has a **feature vector** h_i





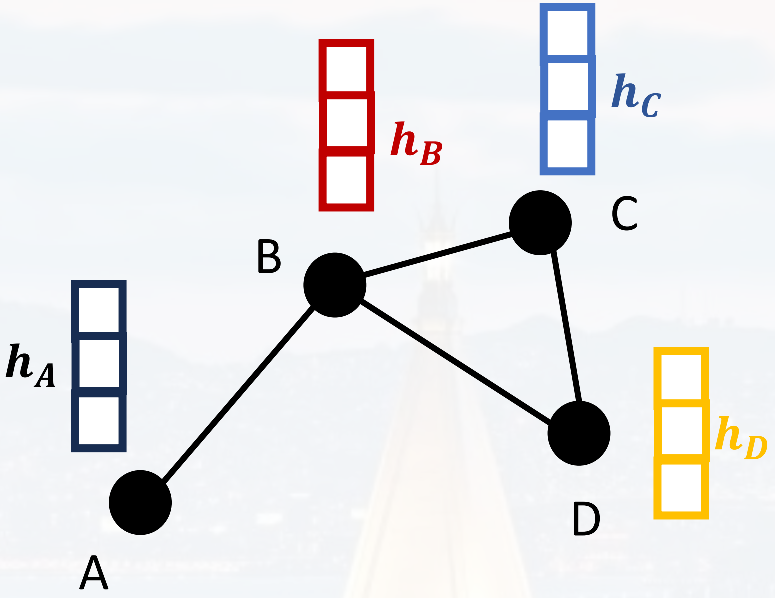


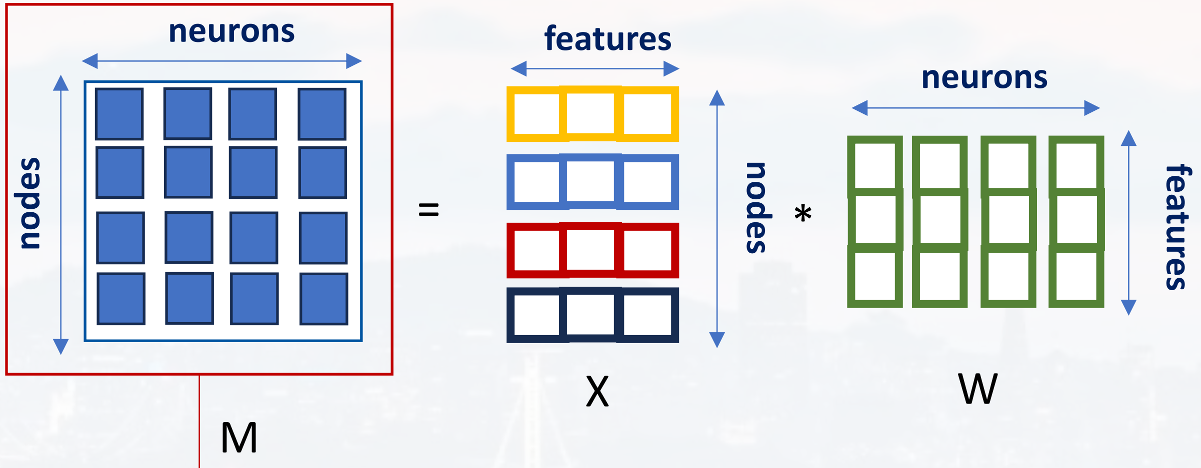
$$net = \sum_i I_i \cdot w_i + b$$

$$m_{jk} = \sum_i w_{ji}\, x_{ik}$$

Graph Convolution

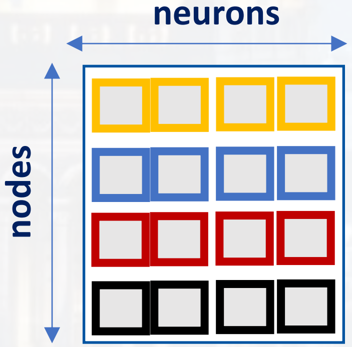
each node i has a **feature vector** h_i





$$m_{jk} = \sum_i w_{ji}\, x_{ik}$$

depending on W the output features may have different lengths then the input features



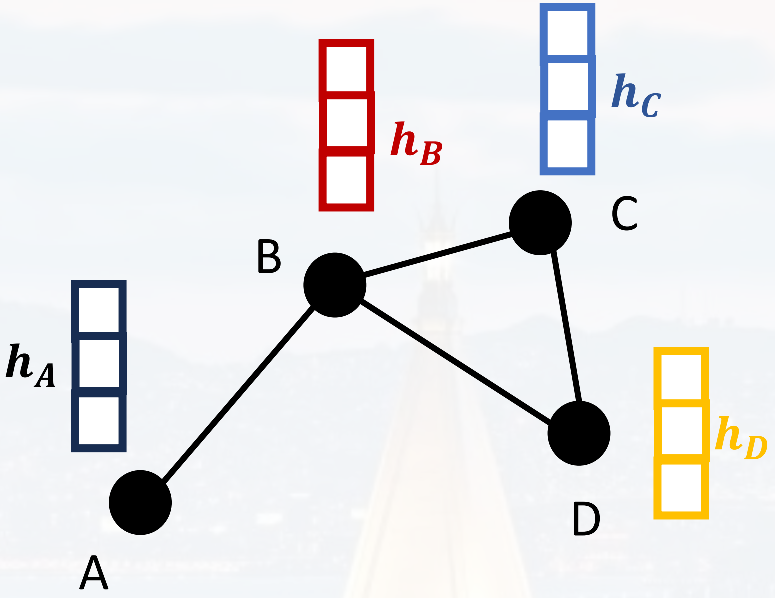
adjacency A

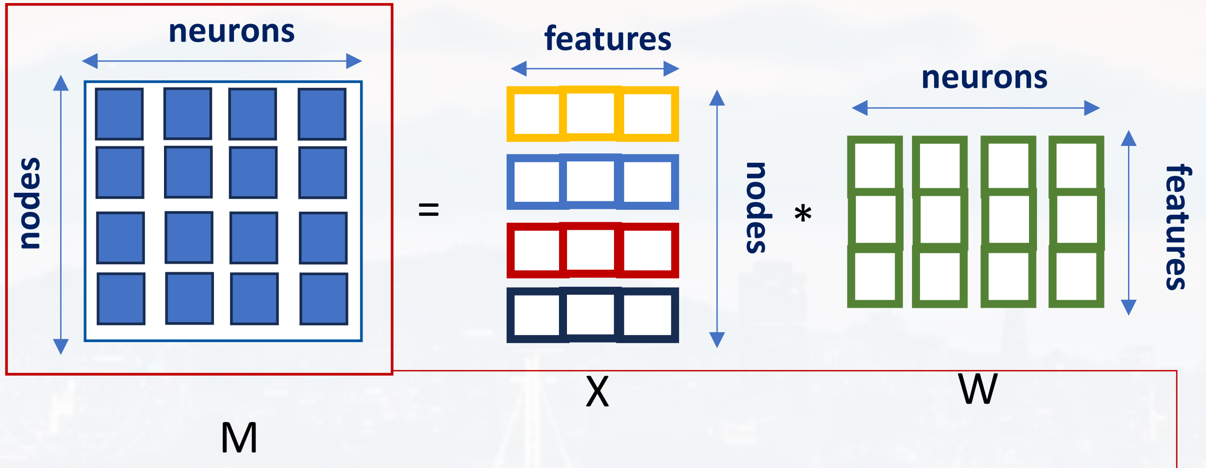
$$= \left[\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right] * \boxed{M}$$

GNN: The ANN Part

Graph Convolution

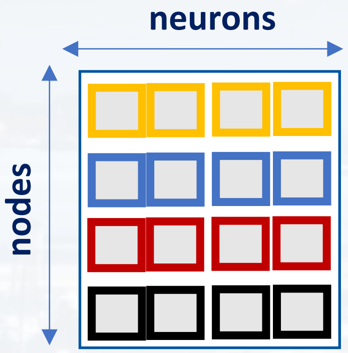
each node *i* has a **feature vector** h_i





pass through a ReLU and/or
repeat the procedure with another W

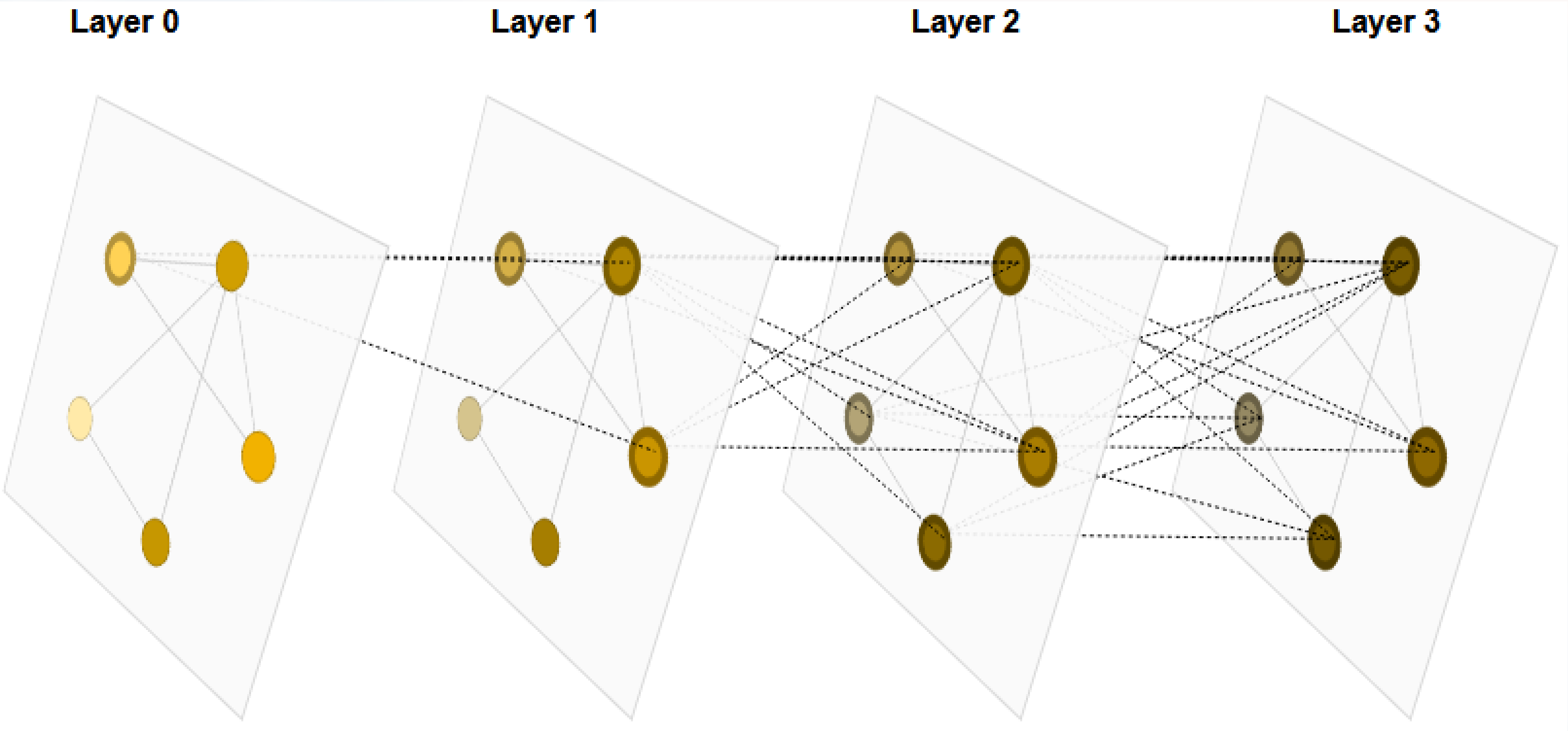
(aka second convolution layer)



adjacency A

$$= \left[\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right] * \boxed{M}$$

Graph Convolution

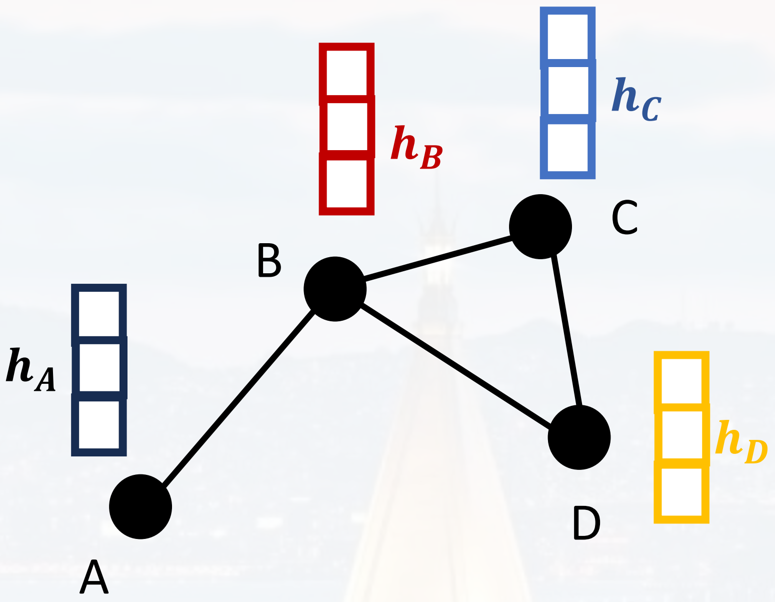


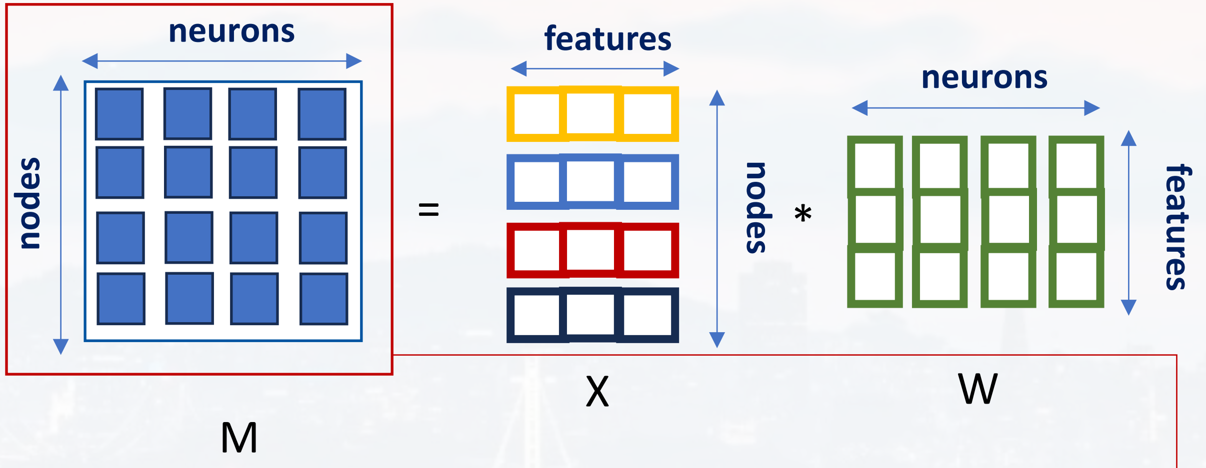
1st layer: one-hop neighborhood
2nd layer: two-hop neighborhood
etc

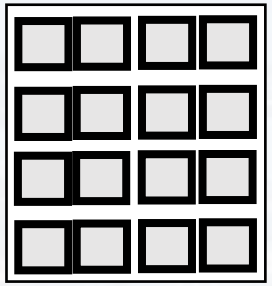
animation here

Graph Convolution

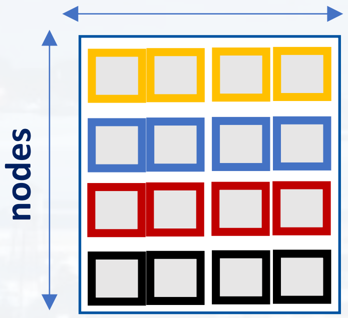
each node i has a **feature vector** h_i







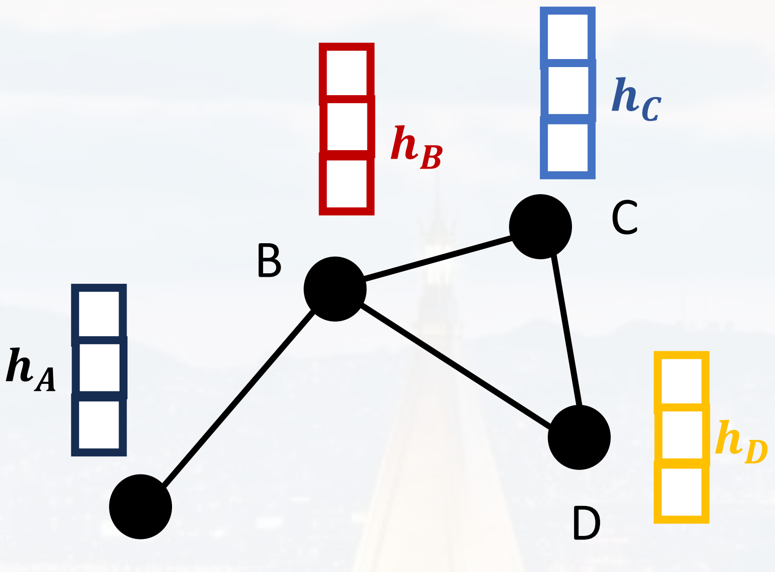
pass through a softmax layer



adjacency A

$$= \left[\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right] * \boxed{M}$$

Graph Convolution



summary

A: adjacency matrix **(number of nodes x number of nodes)**
I: identity matrix **(number of nodes x number of nodes)**
X: node feature matrix **(number of nodes x number of features)**
W: weight matrix **(number of features x number of neurons)**
σ: any activation function
$D^{-1/2}$: diagonal matrix for normalization

$$\boxed{H(embedding) = \sigma[\, \boldsymbol{D}^{-1/2}(A + I)\, \boldsymbol{D}^{-1/2} XW]}$$

However, this would give nodes with higher degree a larger weight

→ normalizing by $\frac{1}{\sqrt{d(n_i)}}$ and $\frac{1}{\sqrt{d(n_j)}}$

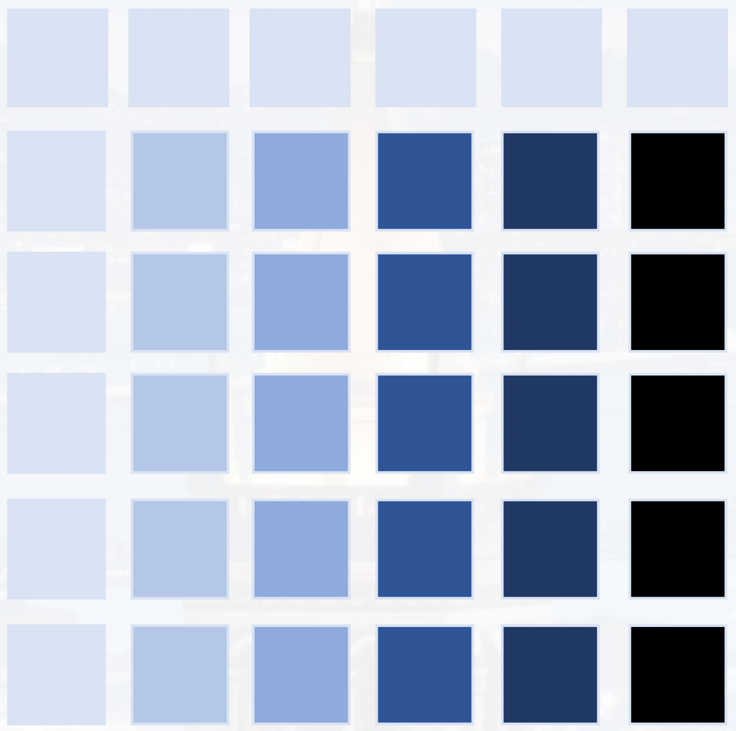
more information here

Matthew N. Bernstein

Graph Attention

see language models (lect 15)

"The cat jumped on the roof."

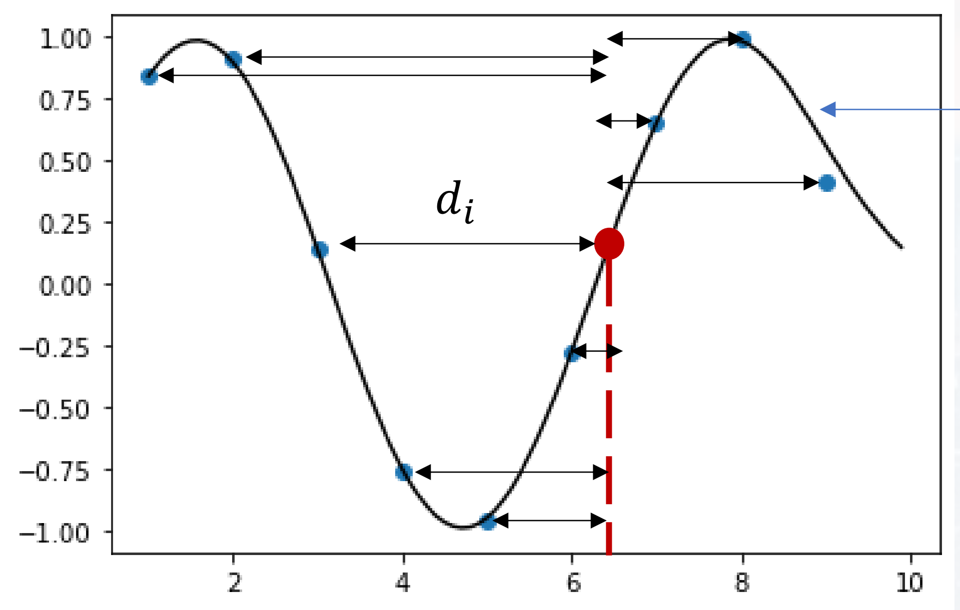


how the first token influences all other token

how the second token influences all other token

…. and so on

Attention



We want to interpolate between the blue dots
→ generating the black line
→ **no curve fitting!**

<u>idea:</u>
- select a point for which we want the interpolation for
- calculate distance d_i to every other point
- each data point should influence the value of the interpolated point
- the closer, the stronger the influence → weighted mean

$$y_{int} \sim \sum_{i=1}^{I} w_i \, y_i \qquad w_i \sim \frac{1}{\boxed{d_i}}$$

GNN: The ANN Part

Attention


```
D = np.tile(V, (1, len(V))) -  np.tile(L.transpose(), (len(V), 1))
```

- each data point should influence the value of the interpolated point
- the closer, the stronger the influence → weighted mean

$$y_{int} \sim \sum_{i=1}^{I} w_i \, y_i$$

Gaussian kernel

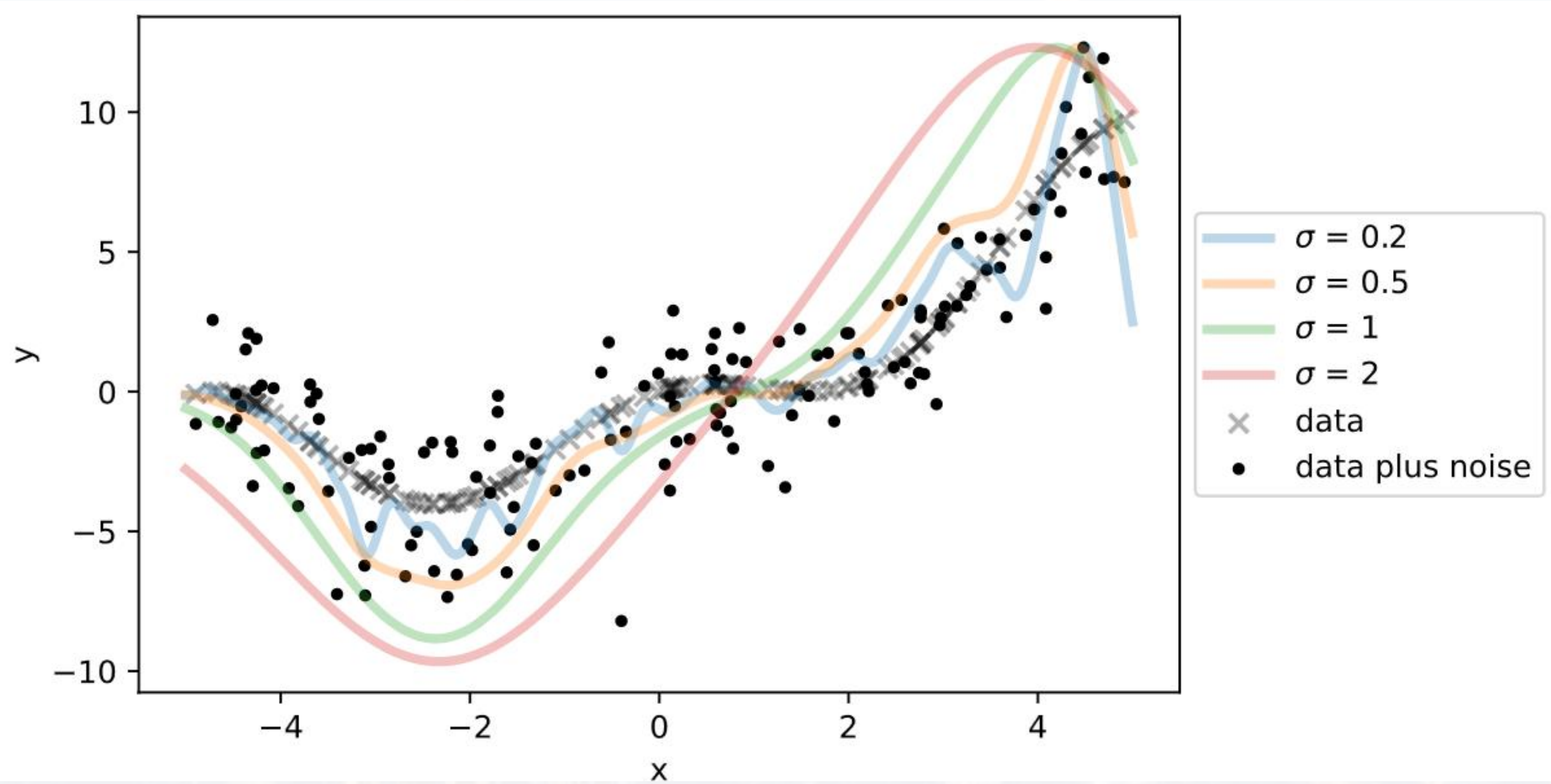
```
W      = np.exp(-(D**2)/(sigma))
W      = W/np.sum(W + 1e-16, axis = 0)
yint   = np.dot(W.transpose(), y)
```

GNN:


```
D = np.tile(V, (1, len(V))) -  np.tile(L.transpose(), (len(V), 1))
```

Gaussian kernel

```
W      = np.exp(-(D**2)/(sigma))
W      = W/np.sum(W + 1e-16, axis = 0)
yint   = np.dot(W.transpose(), y)
```

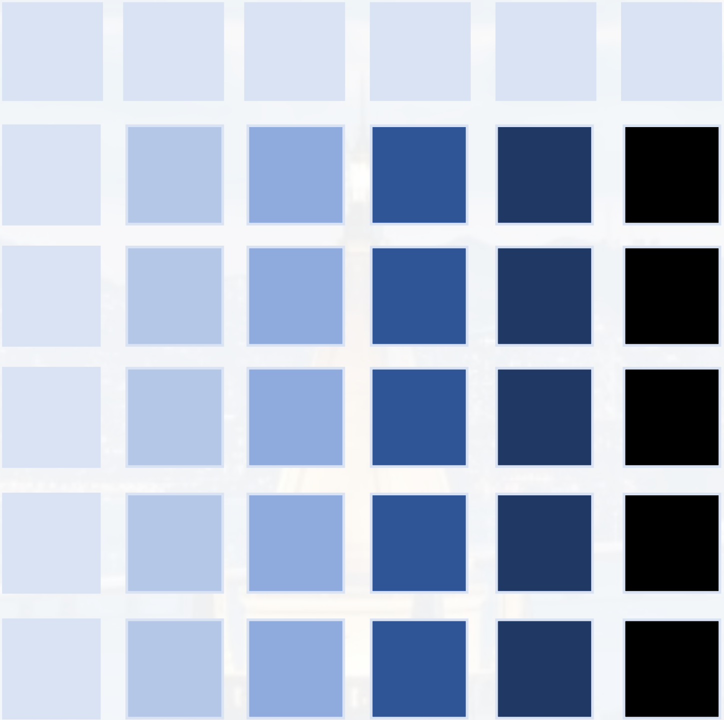


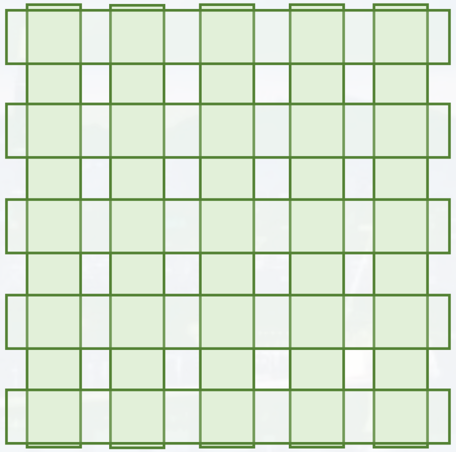
check out:

SmoothGaussKernel.py
SmoothExamples.py

Attention

“The cat jumped on the roof.”





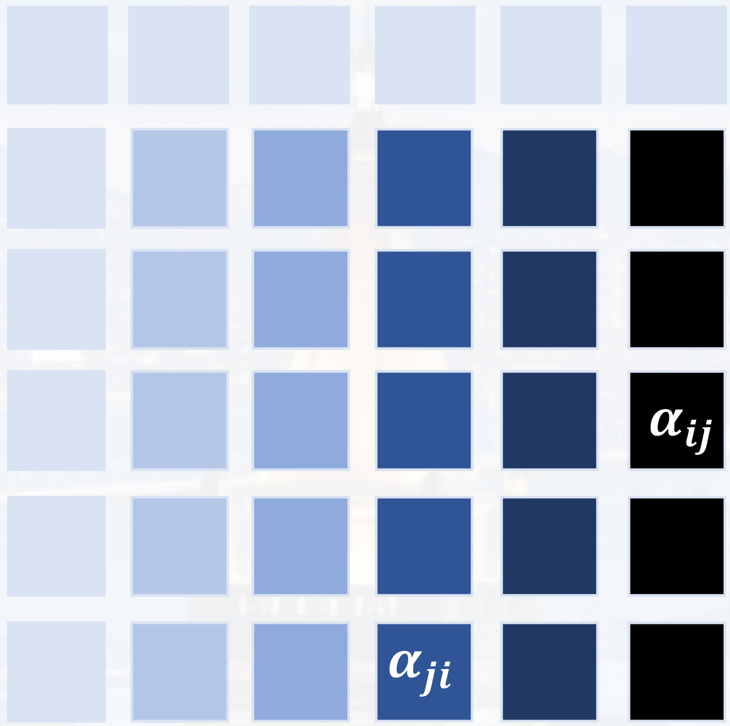
Gaussian kernel

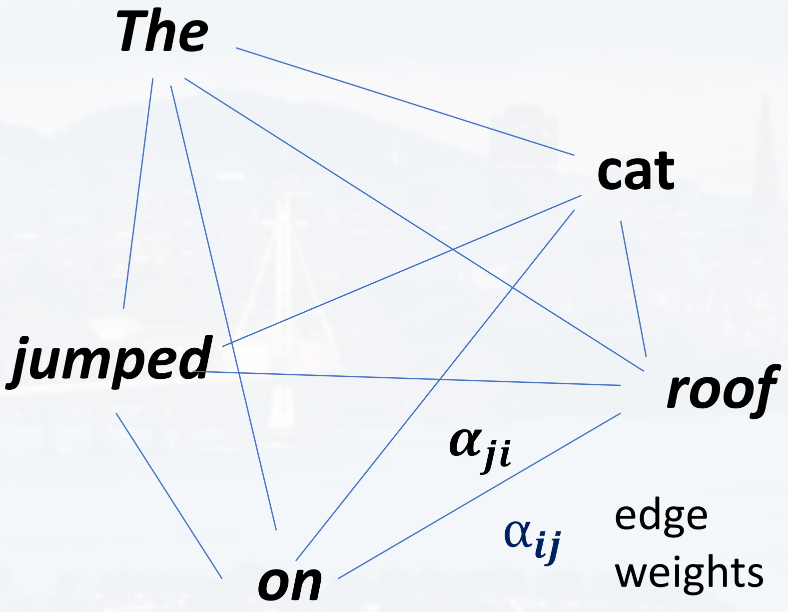
```
W       = np.exp(-(D**2)/(sigma))
W       = W/np.sum(W + 1e-16, axis = 0)
yint    = np.dot(W.transpose(), y)
```

actual attention:
these weights are learnable,
no kernel assumed!

Graph Attention

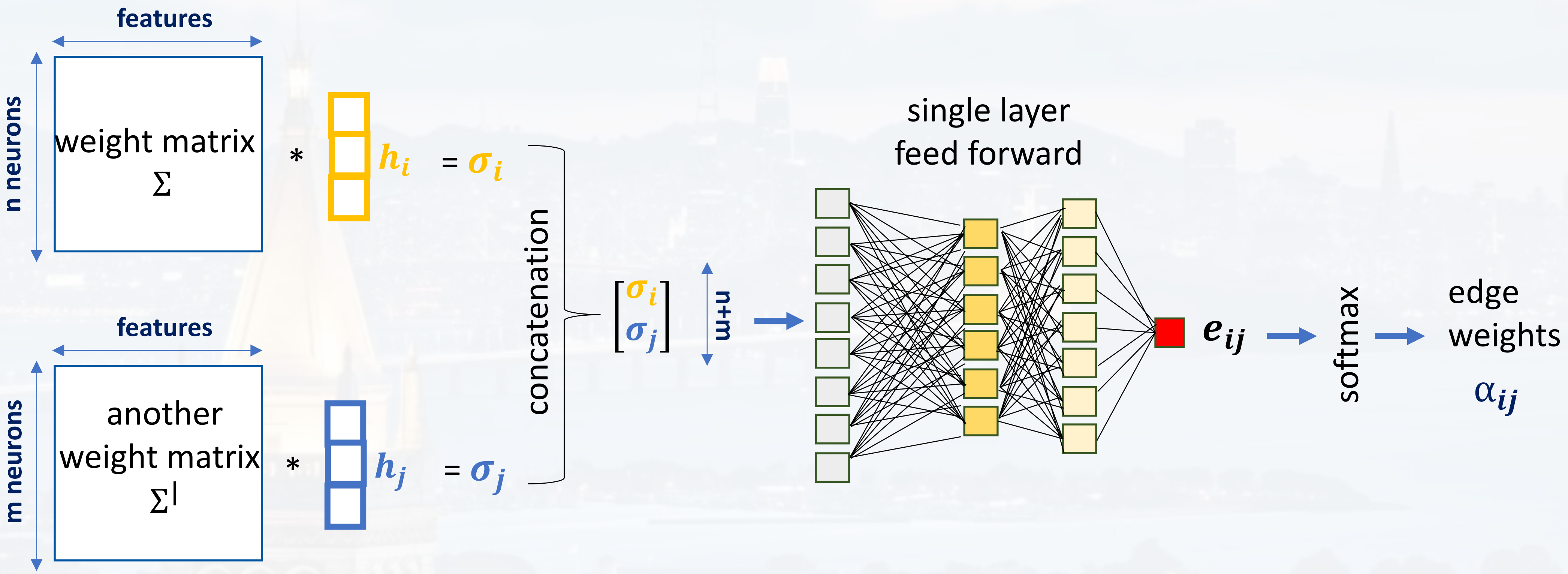
"The cat jumped on the roof."





Graph Attention

Learning the weights!
(edge attributes)



Outline

- What is a Graph

- The ANN Part

- **PyTorch Example**

GNN: PyTorch Example

node classification: **convolution GNN**

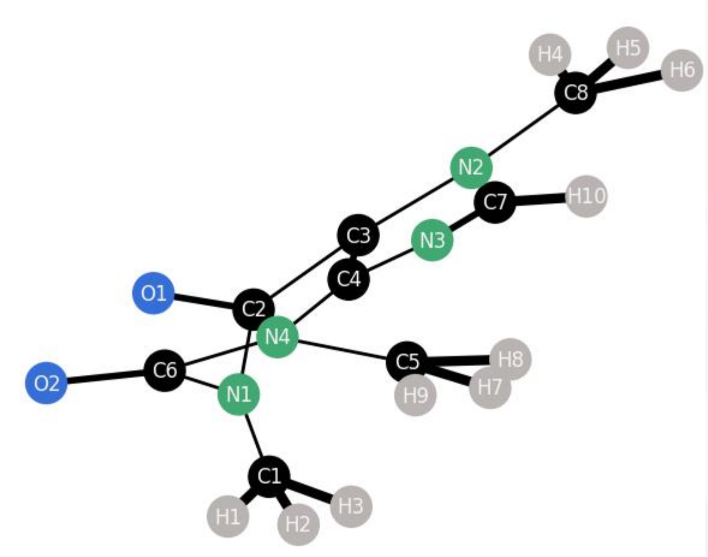
```
self.conv1 = GCNConv(n_node_features, n_neuron)
self.conv2 = GCNConv(n_neuron, n_classes)

log_softmax(x3, dim = 1)
```

- edge weights: binding affinity

see `Graph_III.ipynb`

```
epoch:   0 | loss: 1.49 | accuracy: 66.67%
epoch:  10 | loss: 1.94 | accuracy: 66.67%
epoch:  20 | loss: 0.17 | accuracy: 79.17%
epoch:  30 | loss: 0.13 | accuracy: 79.17%
epoch:  40 | loss: 0.14 | accuracy: 79.17%
epoch:  50 | loss: 0.11 | accuracy: 79.17%
epoch:  60 | loss: 0.11 | accuracy: 79.17%
epoch:  70 | loss: 0.11 | accuracy: 79.17%
epoch:  80 | loss: 0.11 | accuracy: 79.17%
epoch:  90 | loss: 0.11 | accuracy: 79.17%
epoch: 100 | loss: 0.11 | accuracy: 79.17%
epoch: 110 | loss: 0.11 | accuracy: 79.17%
epoch: 120 | loss: 0.10 | accuracy: 79.17%
epoch: 130 | loss: 0.10 | accuracy: 79.17%
epoch: 140 | loss: 0.10 | accuracy: 79.17%
epoch: 150 | loss: 0.10 | accuracy: 79.17%
epoch: 160 | loss: 0.10 | accuracy: 79.17%
```




```
print(Y)
print(Y_pred)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 3. 3.]
tensor([0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 0, 0, 0])
```

Thank you very much for your attention!