Lecture 15:

Graph Neural Networks (GNN)

Markus Hohle

University California, Berkeley

Bayesian Data Analysis and Machine Learning for Physical Sciences

Course Map

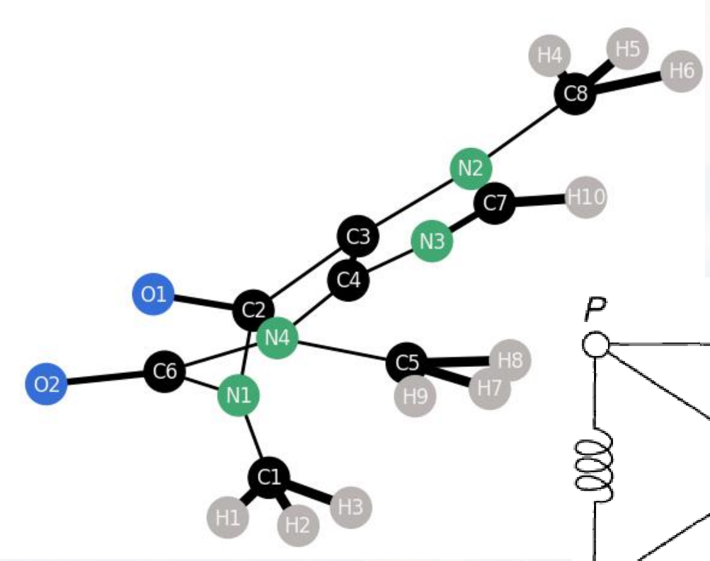
Outline

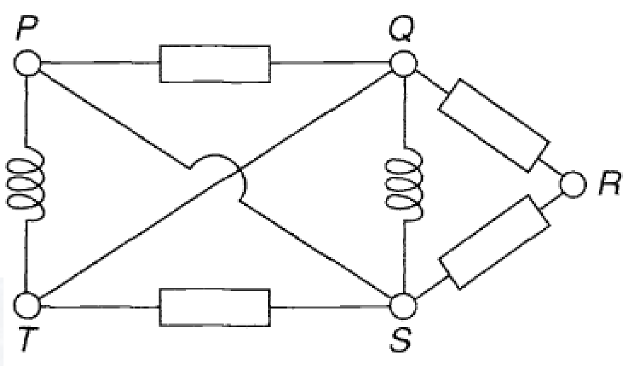
- What is a Graph
- The ANN Part
- PyTorch Example

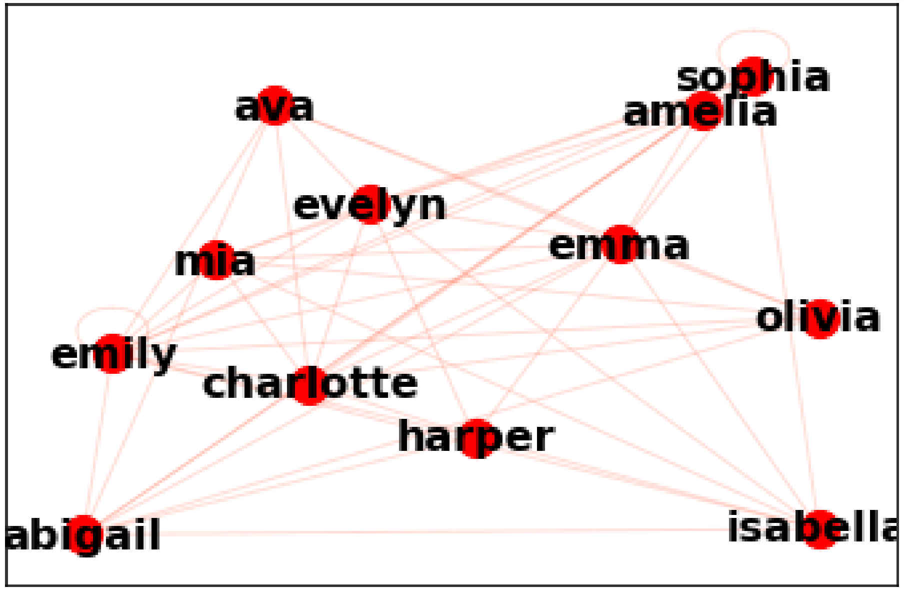
Outline

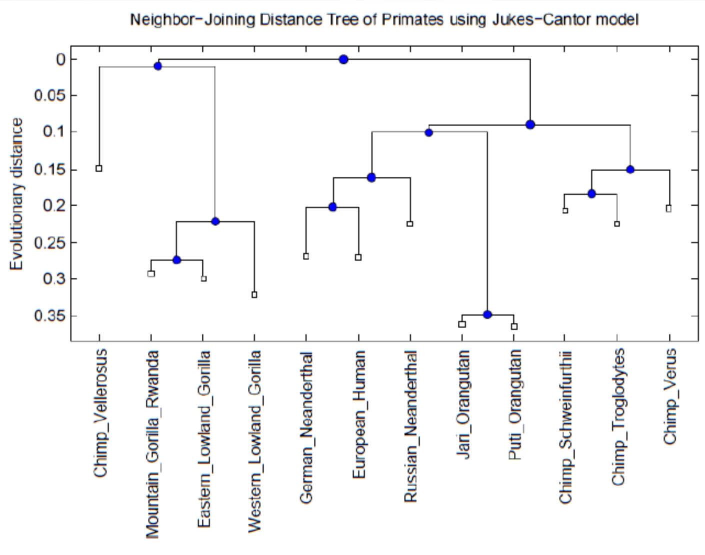
- **What is a Graph**
- The ANN Part
- PyTorch Example

GNN: What is a Graph

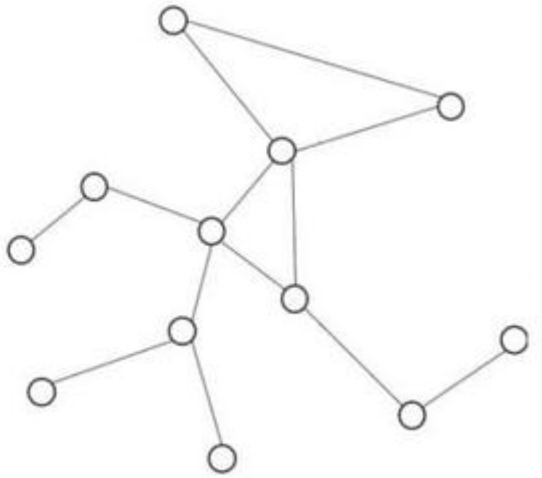


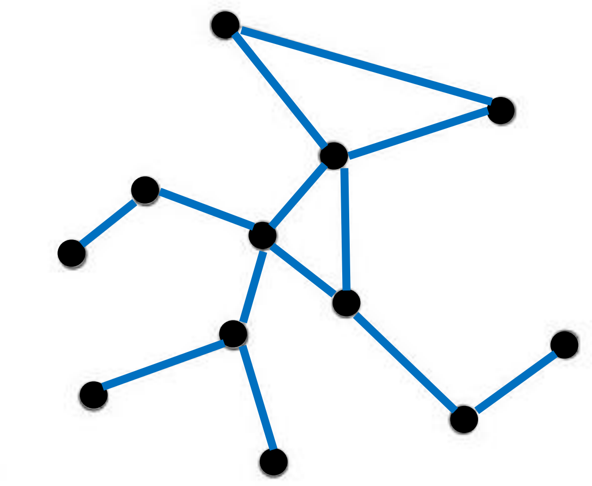






https://doi.org/10.1016/j.aiopen.2021.01.001





Graph G

nodes N (vertices V)

edges E

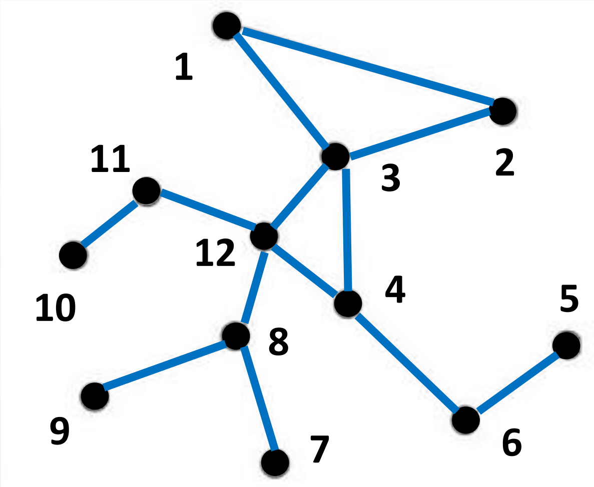
$G = G(N, E)$

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

- 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)

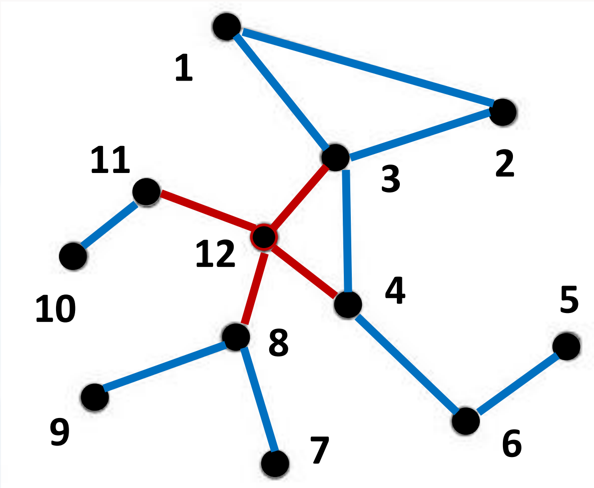
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)

edges E

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



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

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

(nodes $\boldsymbol{n_i}$ and $\boldsymbol{n_j}$ have a common edge)

$A_{ij} = 0$ else

Graph $G = G(N, E)$

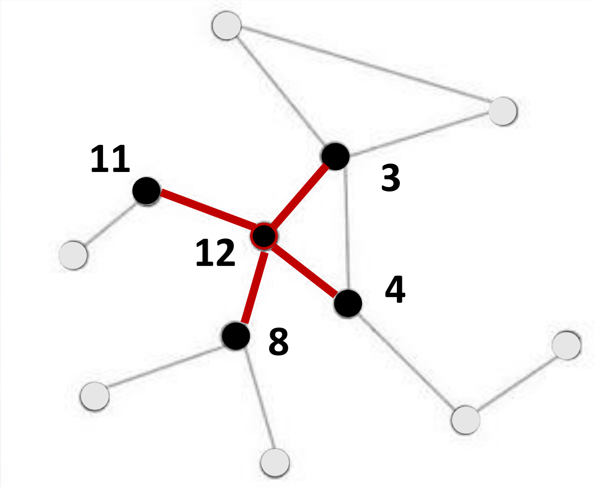
nodes N (vertices V)
edges E

node 12 has four first degree neighbors

degree *d* of a node

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

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



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

node 12 has four first degree neighbors

degree *d* of a node

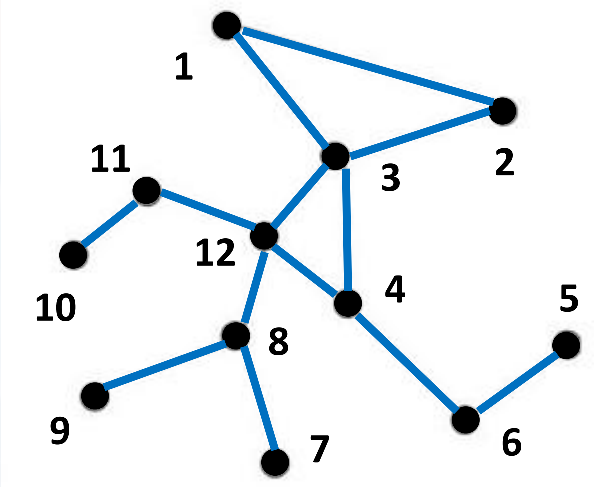
$$d(n_i) = \sum_j A_{ij}$$

first degree neighborhood $\mathcal{N}$

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

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

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

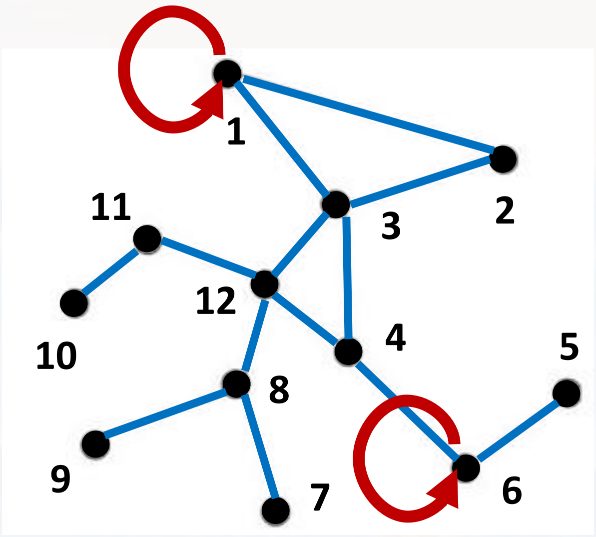
Graph $G = G(N, E)$

nodes N (vertices V)
edges E

A graph can have **loops**

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

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

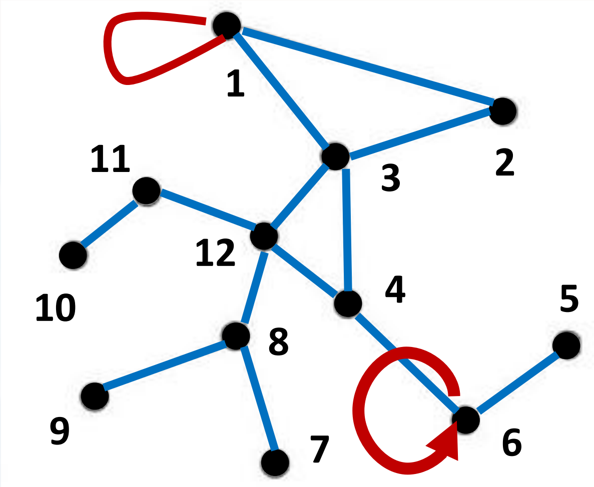
Graph $G = G(N, E)$

nodes N (vertices V)
edges E

A graph can have **loops**

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

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 $G = G(N, E)$

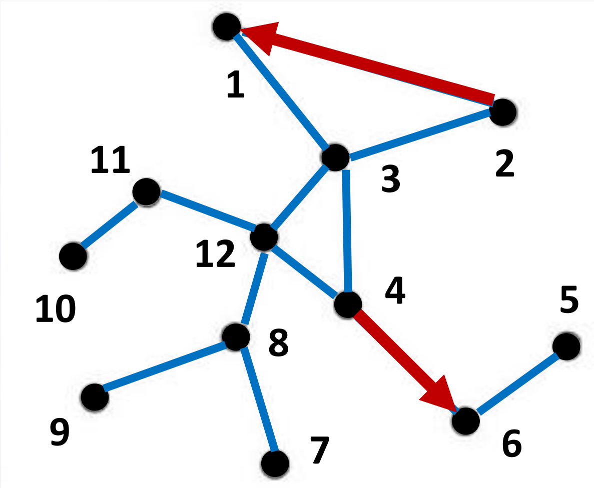
nodes N (vertices V)
edges E

A graph can have **loops**

note:

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

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



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 $G = G(N, E)$

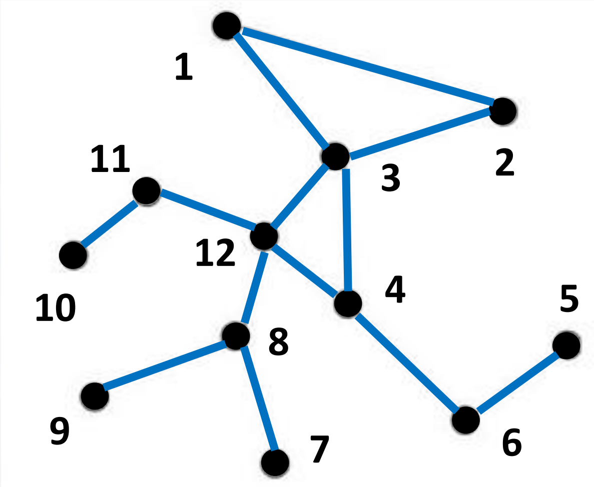
nodes N (vertices V)

edges E

A graph can be **directed**

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

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

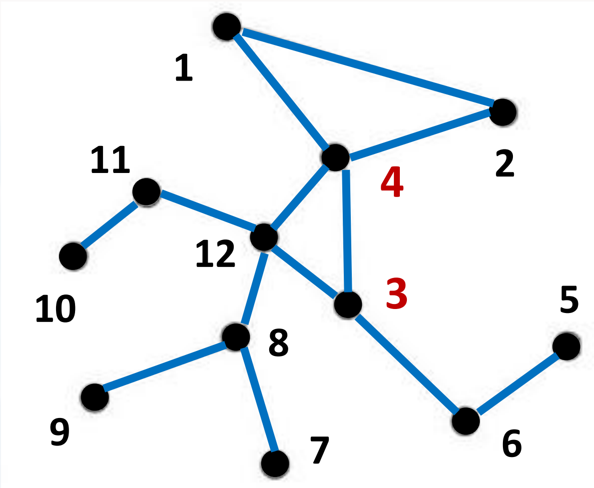
Graph $G = G(N, E)$

nodes N (vertices V)

edges E

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

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



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 enumerating the nodes is not relevant! (**permutation invariance**)

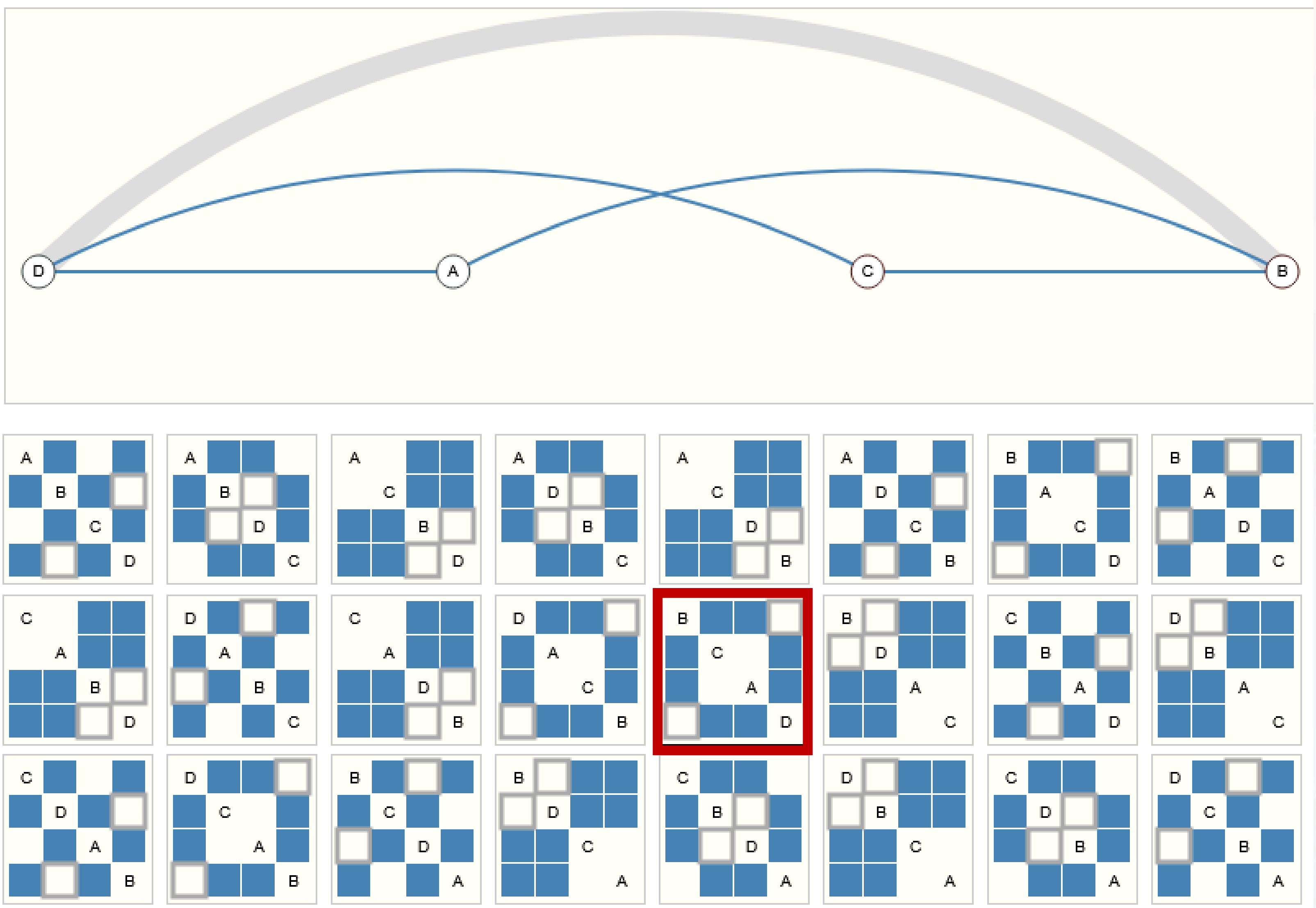
Each graph can be represented by **N!** adjacency matrices!

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

GNN:

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

animation here

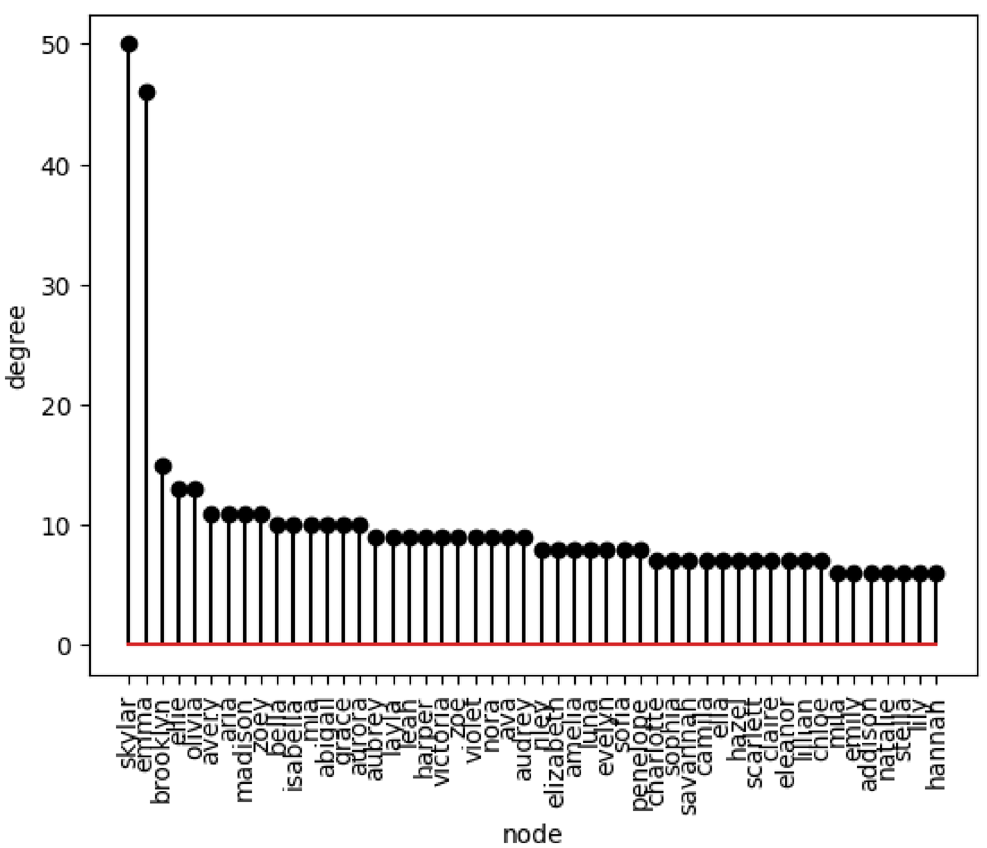


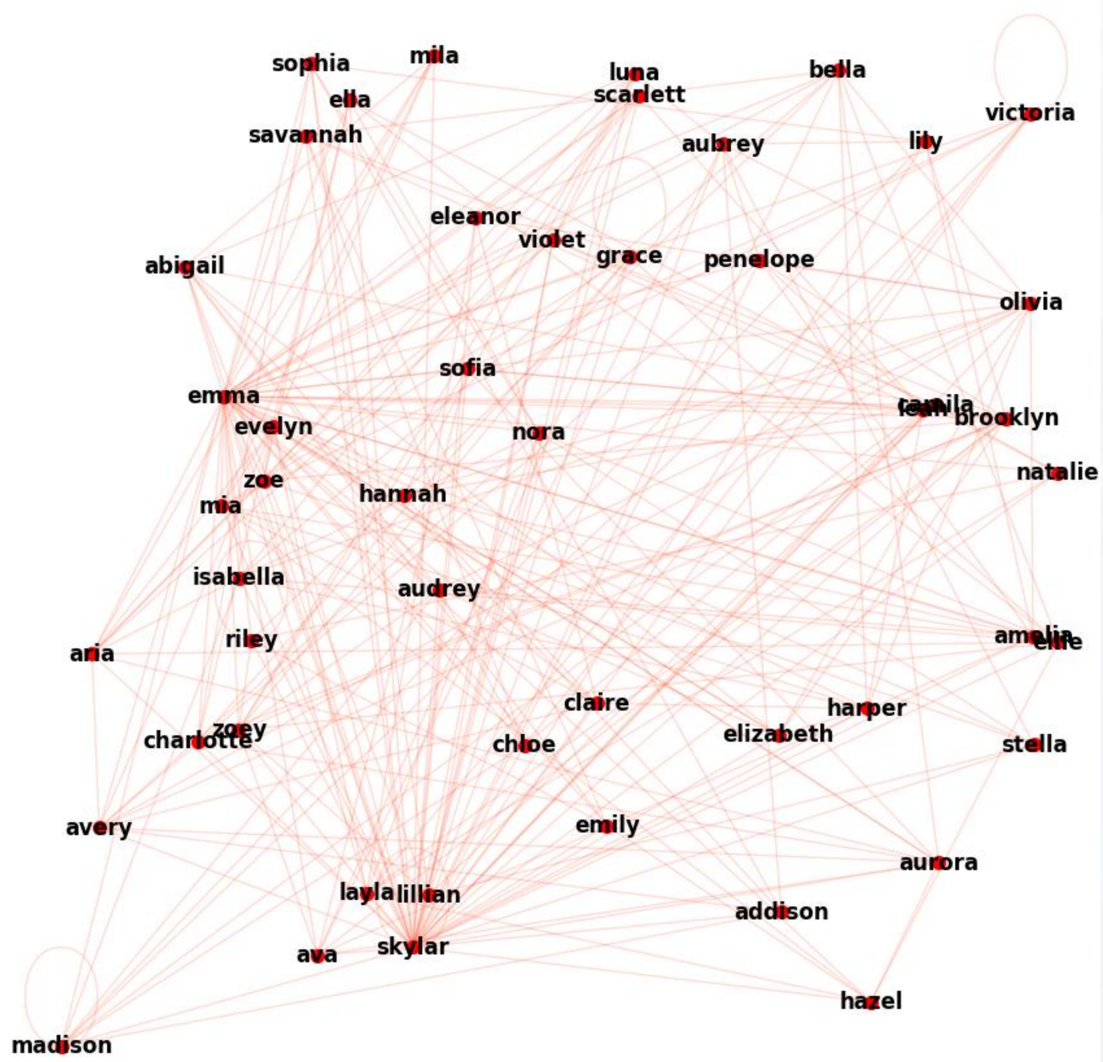
GNN:

visualizing a graph:

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

see: `Graph_I.ipynb`



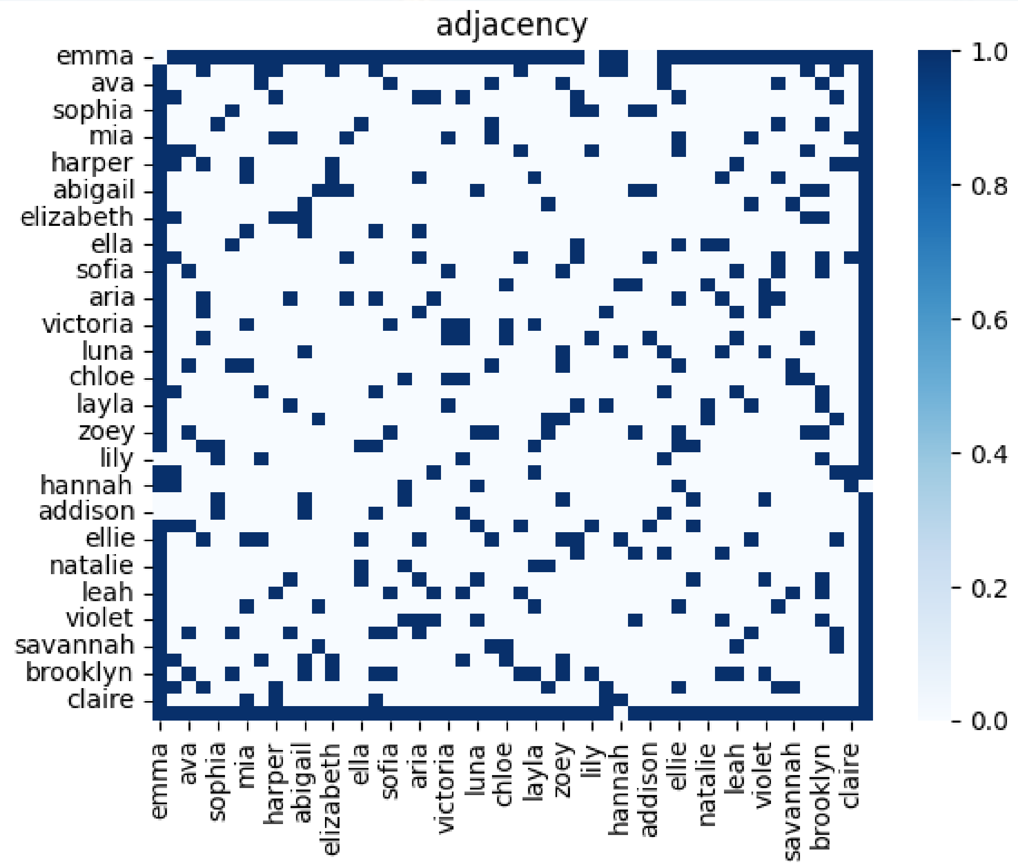


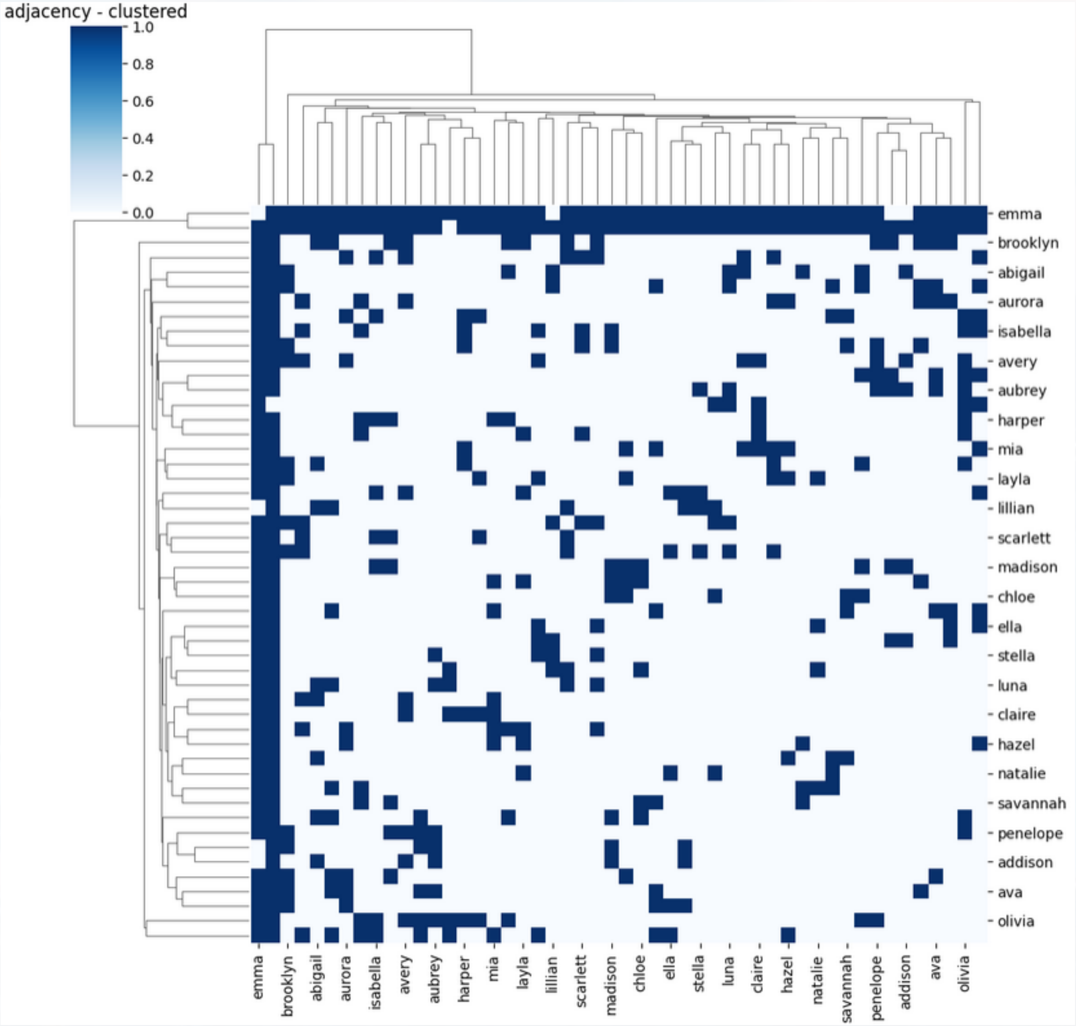
GNN:

visualizing a graph:

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

see: Graph_I.ipynb





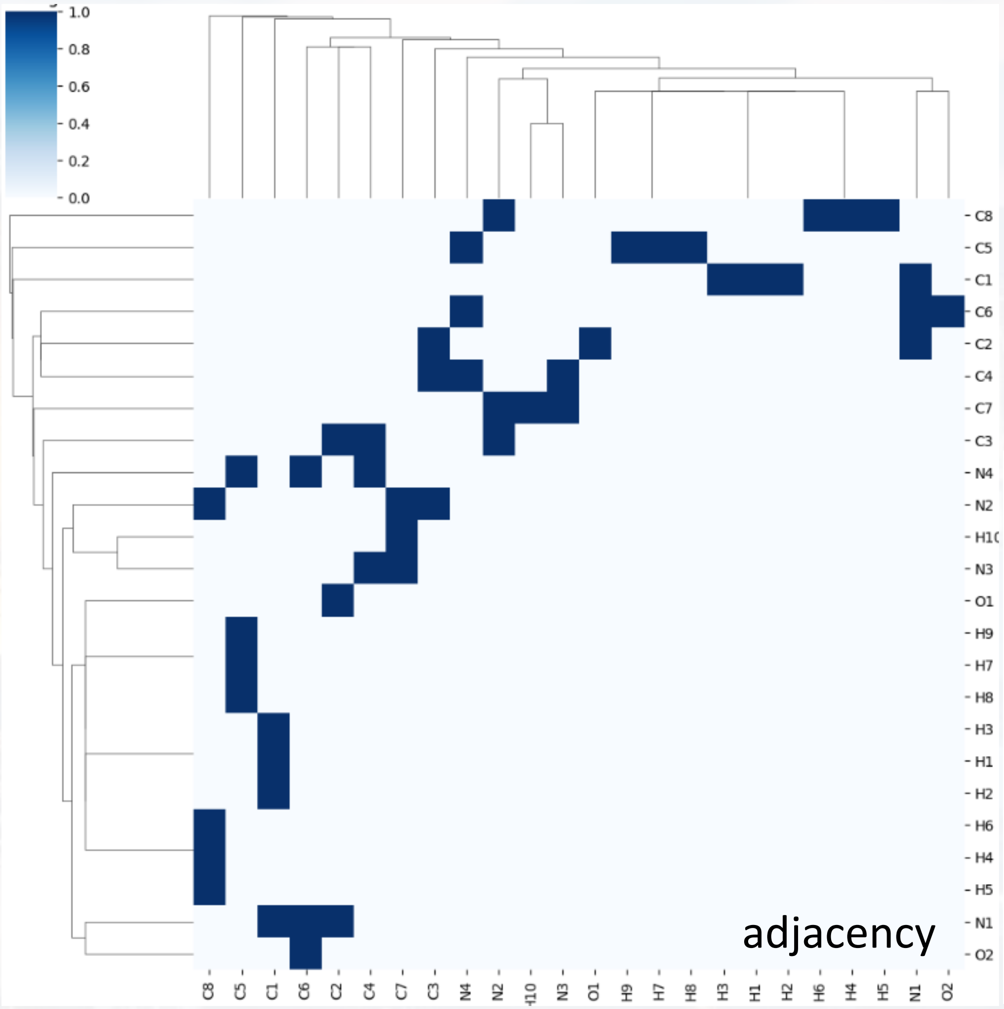
GNN: What is a Graph

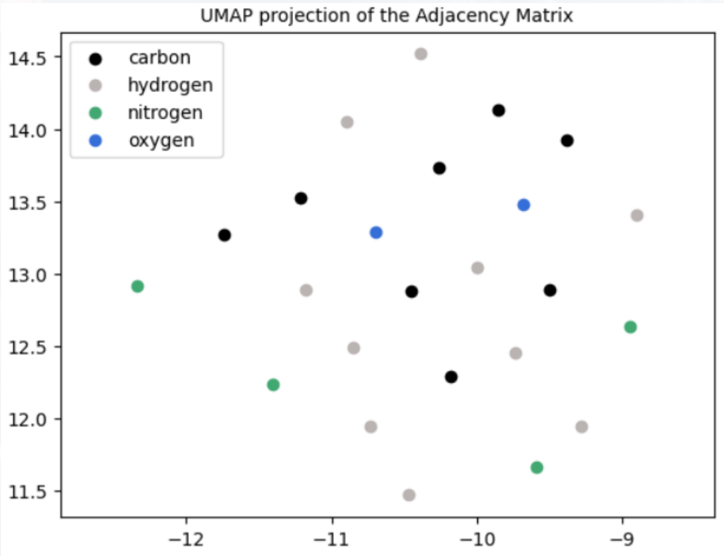
building and visualizing a **weighted** graph:

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

see: `Graph_II.ipynb`

Caffein molecule





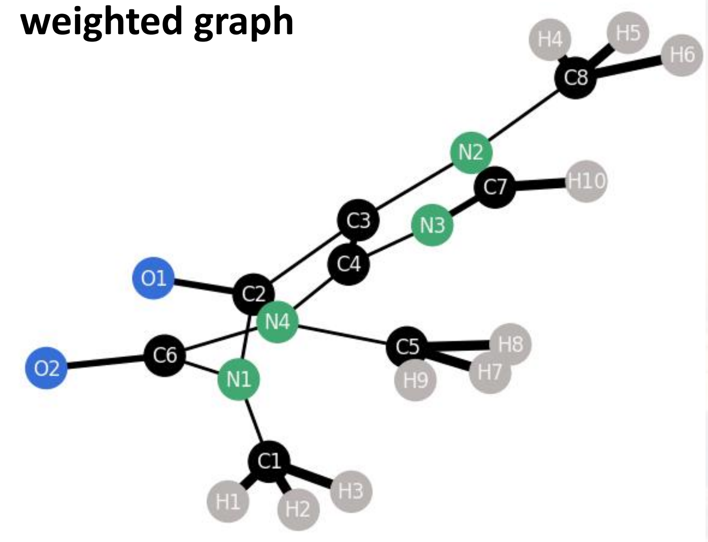
building and visualizing a **weighted** graph:

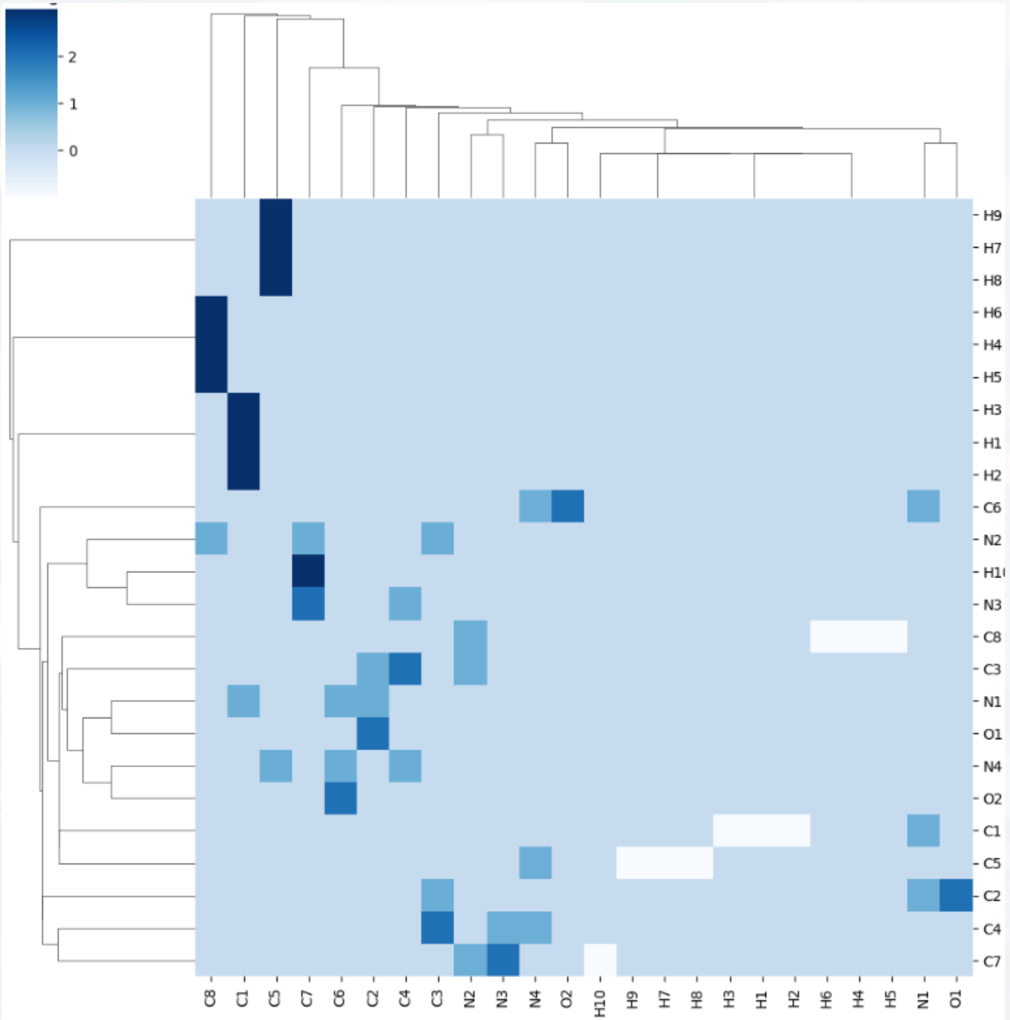
```
import networkx as nx #pip install networkx
```

see: `Graph_II.ipynb`

Caffein molecule

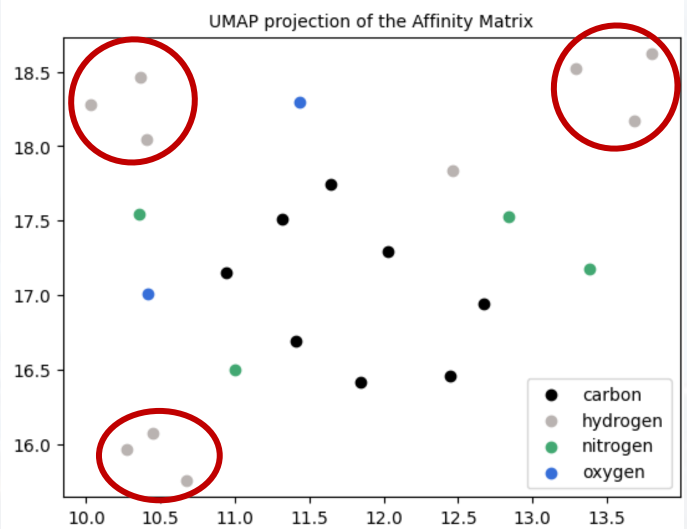
weighted graph





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.

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

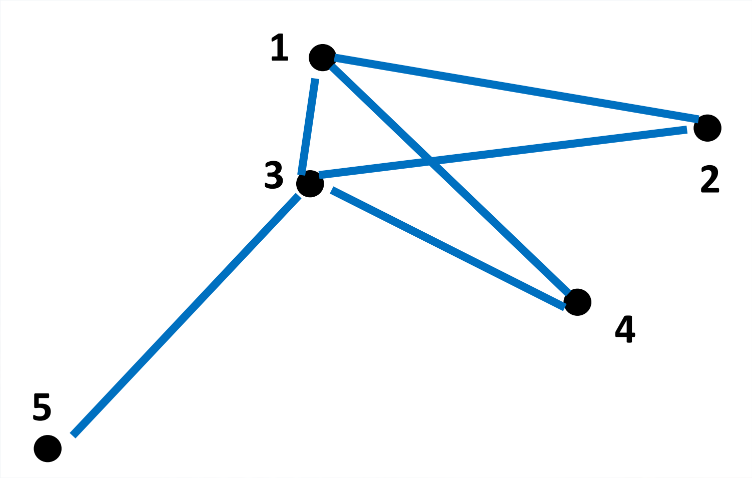
ratio for neighbors nodes n_i and n_j have in common.

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

more about A_{ij} :

$$\vec{e}_0 := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \qquad \boxed{\vec{e}_{t+1} = A\,\vec{e}_t}$$

then $\vec{e}_t$ is the number of length t paths arriving at each node
→ dynamics of the system (diffusion, “message passing”)



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \vec{e}_0 := \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\vec{e}_1 = A\,\vec{e}_0 = \begin{pmatrix} 3 \\ 2 \\ 4 \\ 2 \\ 1 \end{pmatrix}$$

Just the degree of each node!

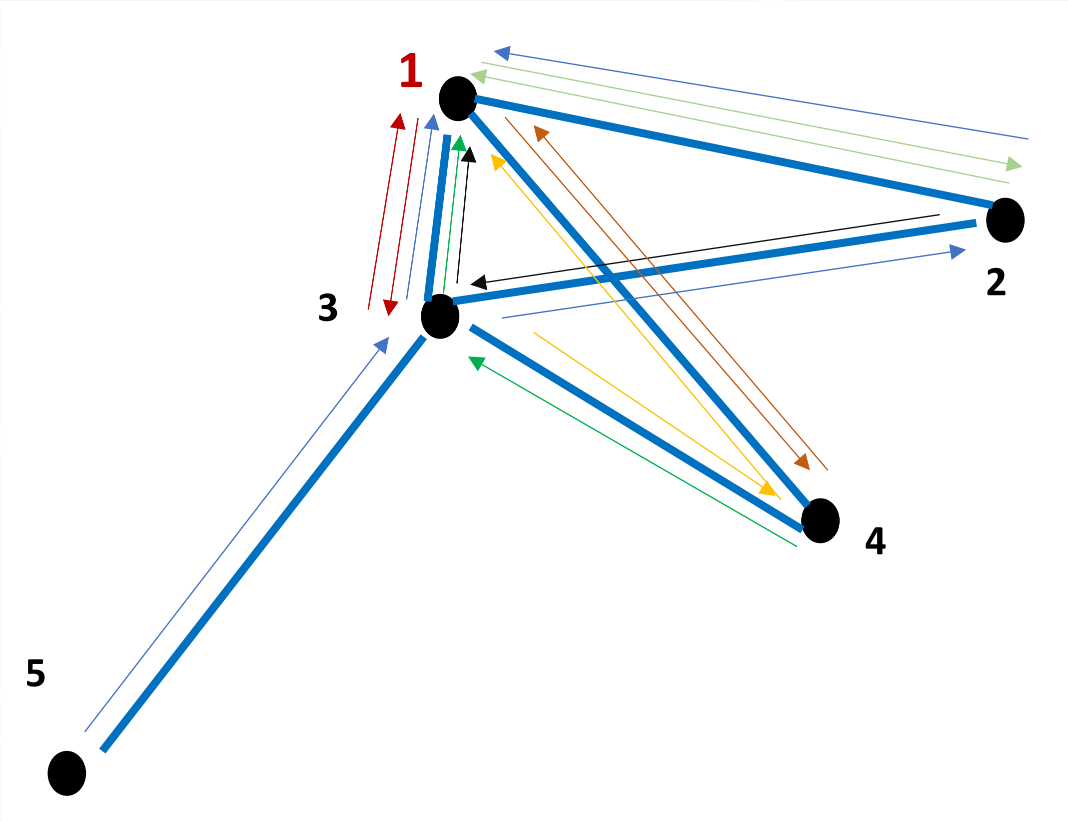
$$\vec{e}_2 = A\,\vec{e}_1 = \begin{pmatrix} 8 \\ 7 \\ 8 \\ 7 \\ 4 \end{pmatrix}$$

number of length $t = 2$ paths arriving at each node

more about A_{ij} :

$$\vec{e}_0 := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \qquad \boxed{\vec{e}_{t+1} = A\,\vec{e}_t}$$

then $\vec{e}_t$ is the number of length t paths arriving at each node
→ dynamics of the system (diffusion, “message passing”)



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

$$\vec{e}_2 = A\,\vec{e}_1 = \begin{pmatrix} 8 \\ 7 \\ 8 \\ 7 \\ 4 \end{pmatrix}$$

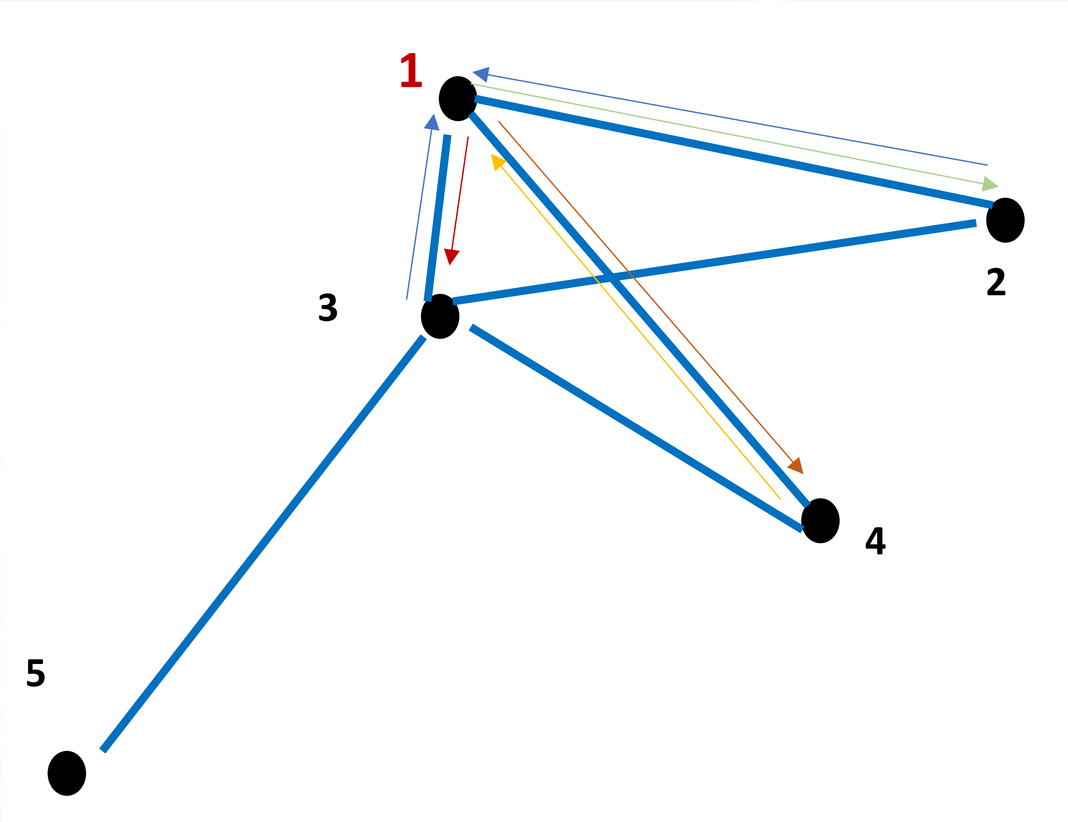
number of length $t = 2$ paths arriving at each node

more about A_{ij} :

$$\vec{e}_0 := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \qquad \boxed{\vec{e}_{t+1} = A\,\vec{e}_t}$$

then $\vec{e}_t$ is the number of length t paths arriving at each node
→ dynamics of the system (diffusion, "message passing")

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \vec{e}_1 = A\,\vec{e}_0 = \begin{pmatrix} \mathbf{3} \\ 2 \\ 4 \\ 2 \\ 1 \end{pmatrix}$$

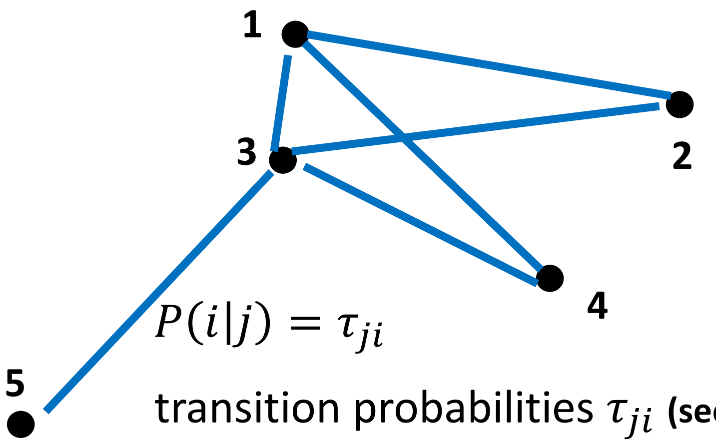


probabilistic point of view:

$$\vec{e}_1 \to \vec{P}_1 = \begin{pmatrix} \boldsymbol{P(1|2)P(2) + P(1|3)P(3) + P(1|4)P(4)} \\ P(2|1)P(1) + P(2|3)P(3) \\ P(3|1)P(1) + \ \dots etc \\ \dots \\ \dots \end{pmatrix}$$

more about A_{ij} : probabilistic point of view:

$$\vec{e}_1 \rightarrow \vec{P}_1 = \begin{pmatrix} P(1|2)P(2) + P(1|3)P(3) + P(1|4)P(4) \\ P(2|1)P(1) + P(2|3)P(3) \\ P(3|1)P(1) + \ \dots etc \\ \dots \\ \dots \end{pmatrix}$$



$$P(i|j) = \tau_{ji}$$

transition probabilities τ_{ji} **(see module 7 “HMMs” and “stochastic processes”)**

master equation

$$\frac{dP(n_i,t)}{dt} = \sum_{j=1}^{J} \tau_{ji}\, P(n_j,t) - \sum_{j=1}^{J} \tau_{ij} P(n_i,t)$$

$$= \sum_{j=1}^{J} \tau_{ji}\, P(n_j,t) - P(n_i,t) \sum_{j=1}^{J} \tau_{ij}$$

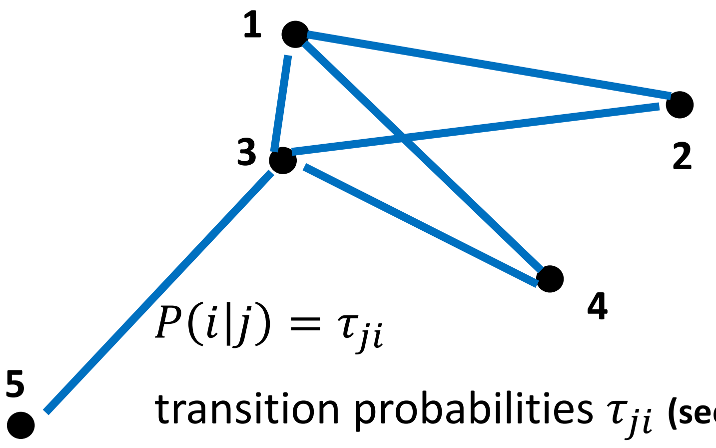
$$= \sum_{j=1}^{J} \tau_{ji}\, A_{ji} P(n_j,t) - P(n_i,t) \sum_{j=1}^{J} \tau_{ij}\, A_{ij}$$

$$A_{ij} = \begin{cases} 1, \text{if } (n_i, n_j) \in E \\ 0, \quad else \end{cases}$$

$$\tau_{ij} = \begin{cases} \tau_{ij}, \text{if } (n_i, n_j) \in E \\ 0, \quad else \end{cases}$$

more about A_{ij} : probabilistic point of view:

$$\vec{e}_1 \to \vec{P}_1 = \begin{pmatrix} P(1|2)P(2) + P(1|3)P(3) + P(1|4)P(4) \\ P(2|1)P(1) + P(2|3)P(3) \\ P(3|1)P(1) + \ \ldots etc \\ \ldots \\ \ldots \end{pmatrix}$$



$P(i|j) = \tau_{ji}$

transition probabilities τ_{ji} **(see module 7 “HMMs” and “stochastic processes”)**

master equation

$$\frac{dP(n_i,t)}{dt} = \sum_{j=1}^{J} \tau_{ji}\, P(n_j,t) - P(n_i,t) \sum_{j=1}^{J} \tau_{ij}$$

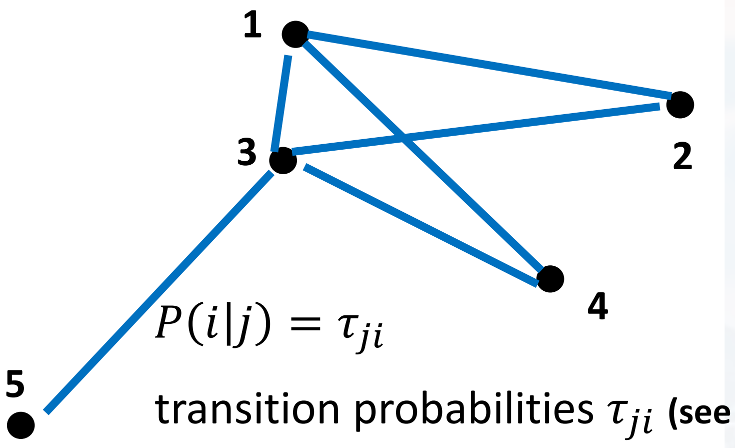
$$\tau_{ij} = \begin{cases} \tau_{ij}, \text{if } (n_i,n_j) \in E \\ 0, \quad else \end{cases}$$

$$\frac{d\,\vec{P}(n,t)}{dt} = \tau\,\vec{P}(n,t) - \vec{P}(n,t) * [\tau\,\vec{P}(n,t)]$$

$*$: element wise multiplication

$$\frac{d\,\vec{P}(n,t)}{dt} = \vec{P}(n,t) * \left(\tau \ - \ [\tau\,\vec{P}(n,t)]\right)$$

more about A_{ij} : probabilistic point of view:



$P(i|j) = \tau_{ji}$

transition probabilities τ_{ji} **(see module 7 “HMMs” and “stochastic processes”)**

$$\vec{e}_1 \rightarrow \vec{P}_1 = \begin{pmatrix} P(1|2)P(2) + P(1|3)P(3) + P(1|4)P(4) \\ P(2|1)P(1) + P(2|3)P(3) \\ P(3|1)P(1) + \ldots etc \\ \ldots \\ \ldots \end{pmatrix}$$

$$\tau_{ij} = \begin{cases} 1, \text{if } (n_i, n_j) \in E \\ 0, \quad else \end{cases}$$

master equation $$\frac{d\,\vec{P}(n,t)}{dt} = \tau\,\vec{P}(n,t) - \vec{P}(n,t) * [\tau\,\vec{P}(n,t)]$$

$*$: element wise multiplication

$$\frac{d\,\vec{P}(n,t)}{dt} = c(D-A)\vec{P}(n,t)$$

for $\tau_{ij} = c$

graph Laplacian $\mathcal{L}$

$$D = \begin{pmatrix} d(n_1) & 0 & 0 \\ 0 & d(n_2) & 0 \\ 0 & 0 & \ldots \end{pmatrix}$$

Outline

- What is a Graph

- The ANN Part

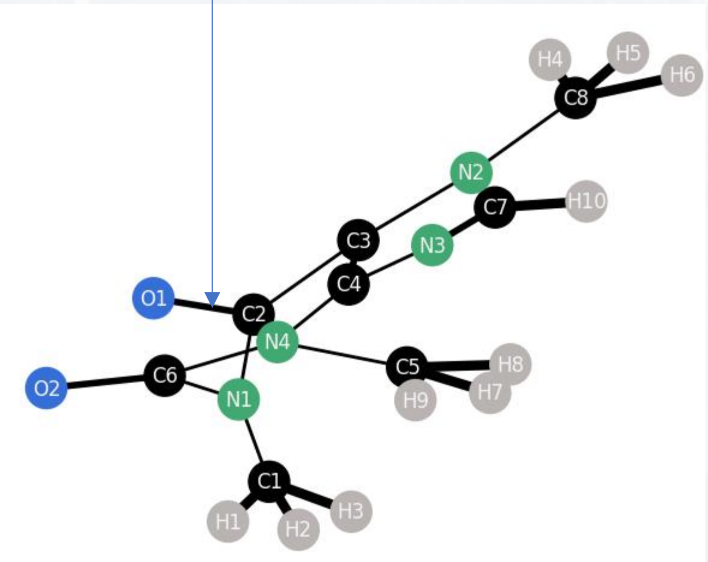
- PyTorch Example

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

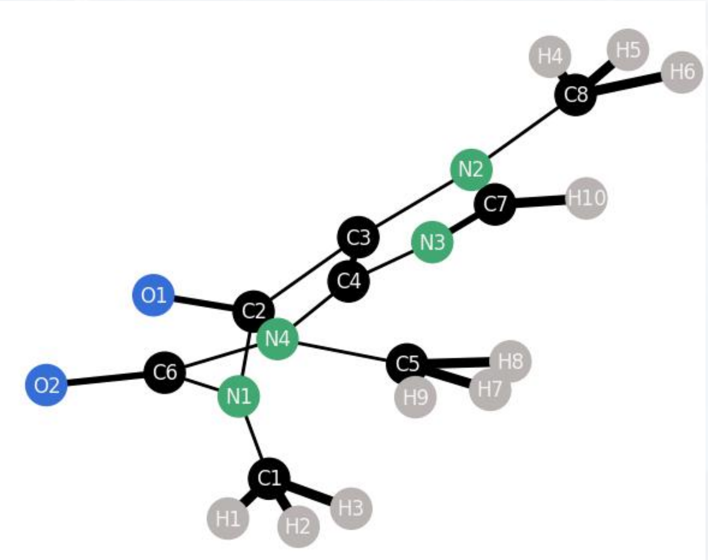
weight: bond strength



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**

graph level tasks

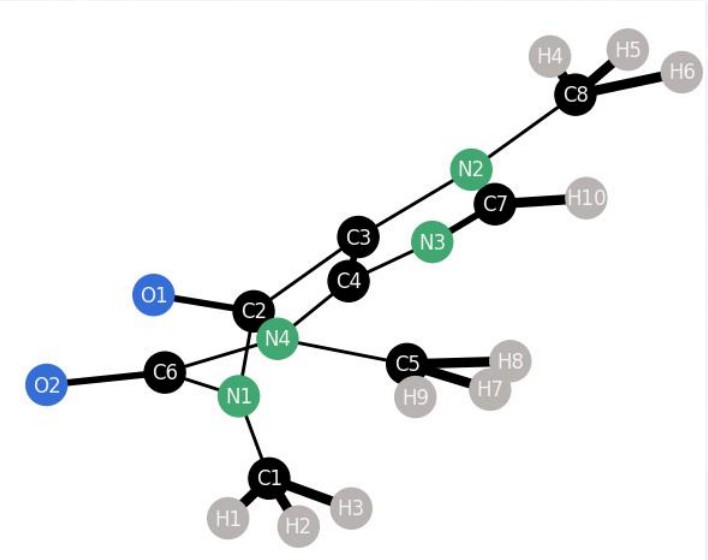


GNN:

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

node (edge) level tasks



GNN:

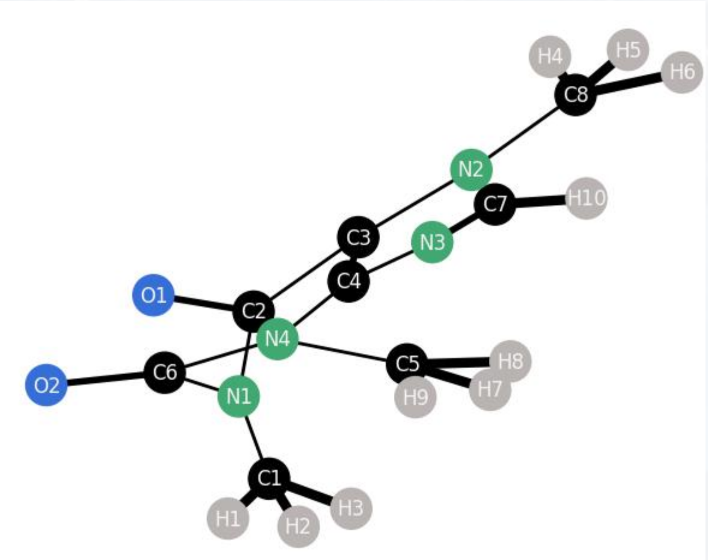
What we can learn:

- 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



GNN:

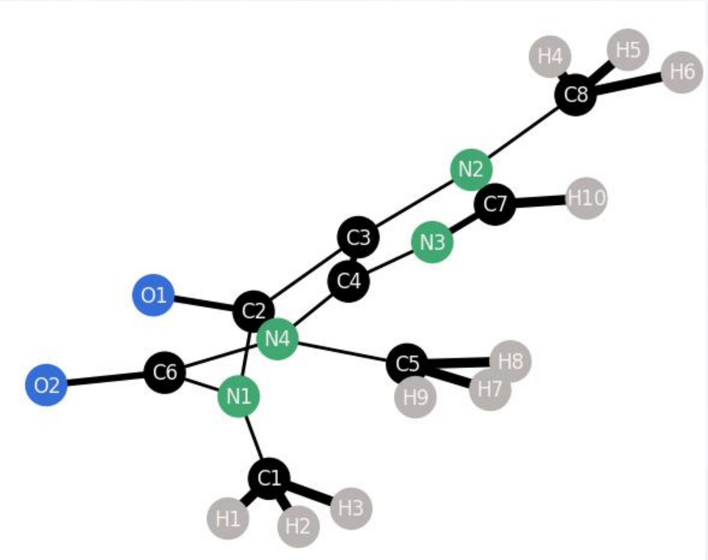
What we can learn:

- 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

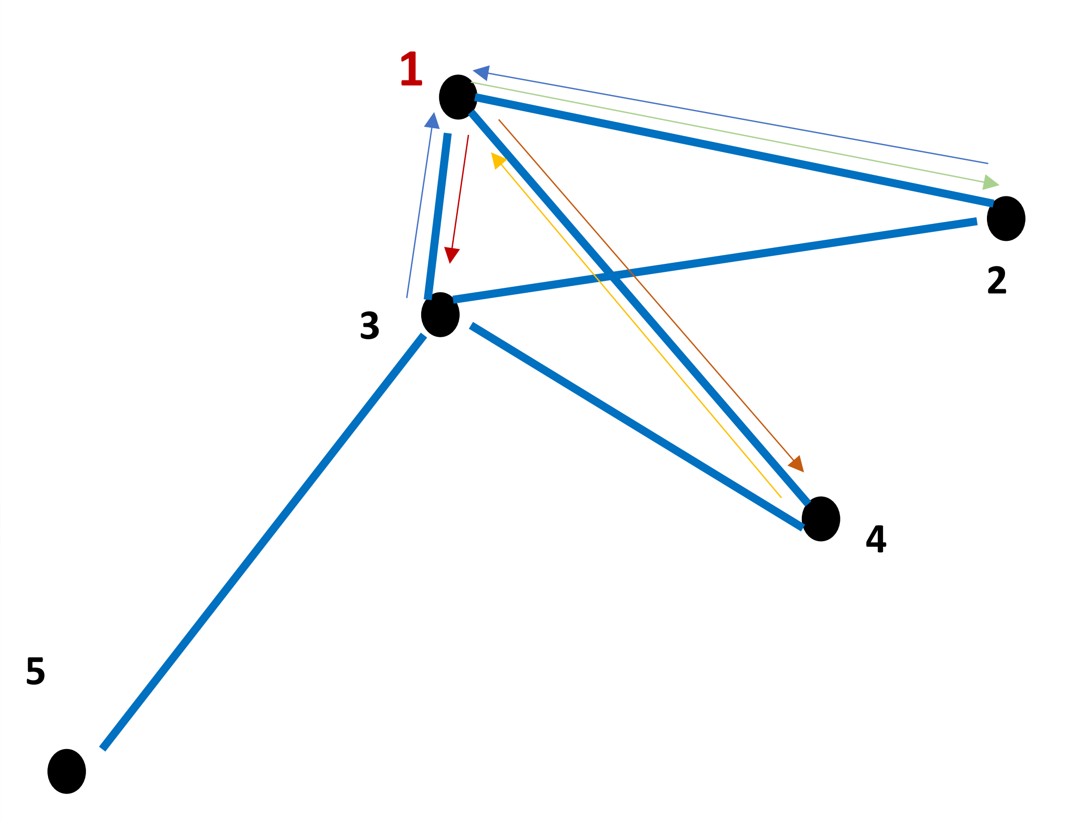
information flow from one node to another:
message passing

different ways how:

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



GNN: The ANN Part



$$\vec{e}_1 \rightarrow \vec{P}_1 = \begin{pmatrix} \boldsymbol{P(1|2)P(2) + P(1|3)P(3) + P(1|4)P(4)} \\ P(2|1)P(1) + P(2|3)P(3) \\ P(3|1)P(1) + \ \dots etc \\ \dots \\ \dots \end{pmatrix}$$

passing information from one node to the others
"message passing"
summing the information from neighbor nodes: **"aggregation"**

$h_{i,t}$: embedding vector of node i at time t

$z_{i,t}$: aggregate from i's **neighbors**

$$z_{i,t} = aggregate(h_{m,t}: m \in \mathcal{N}(n_i))$$

updating values based on new information

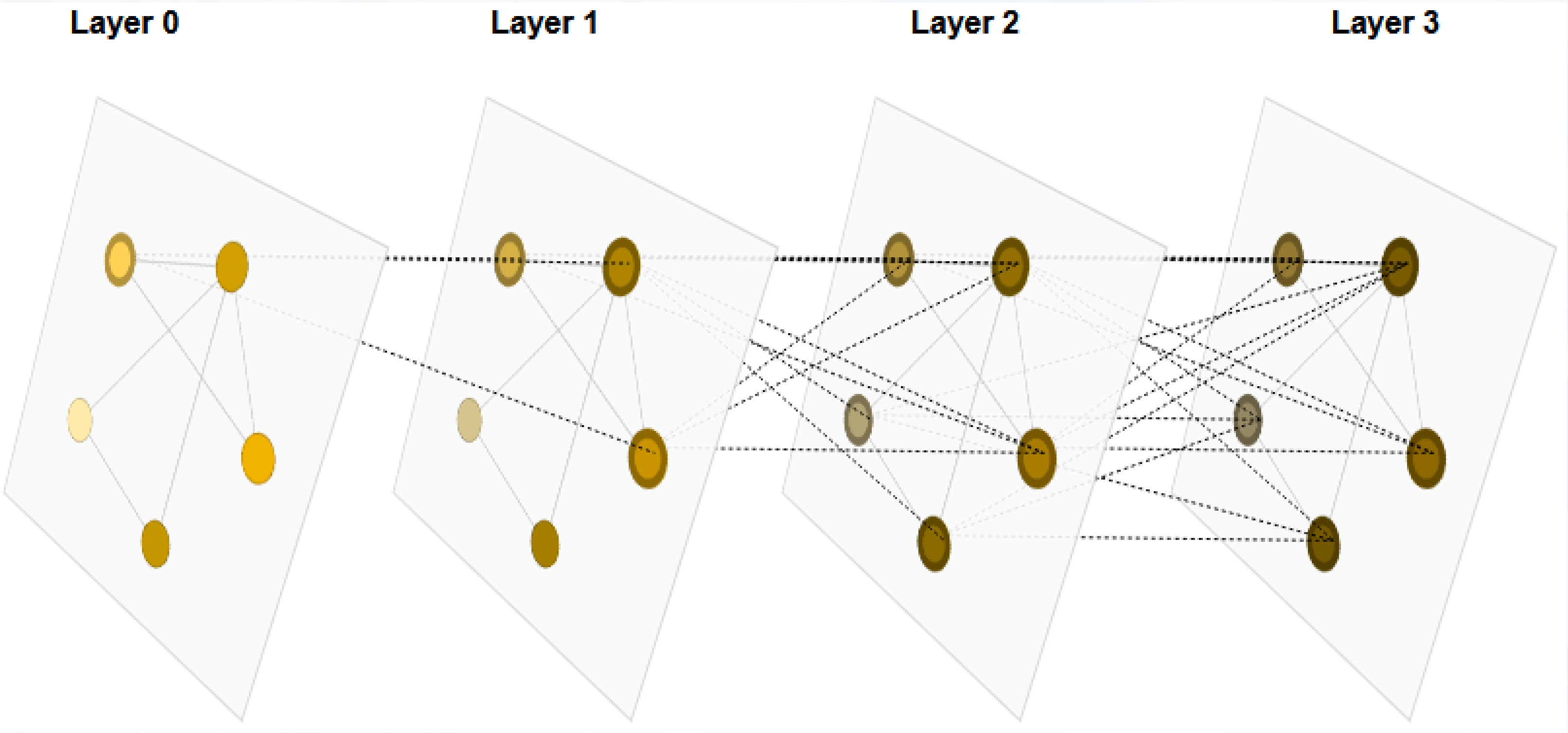
$$update(h_{i,t}, z_{i,t}) = f_{nonlin}(w_{self}\ h_{i,t} + w_{neigh}\ z_{i,t})$$

$w_{self}; w_{neigh}$: trainable weights

GNN: The ANN Part

Graph Convolution

note: time t can be interpreted as different layers!

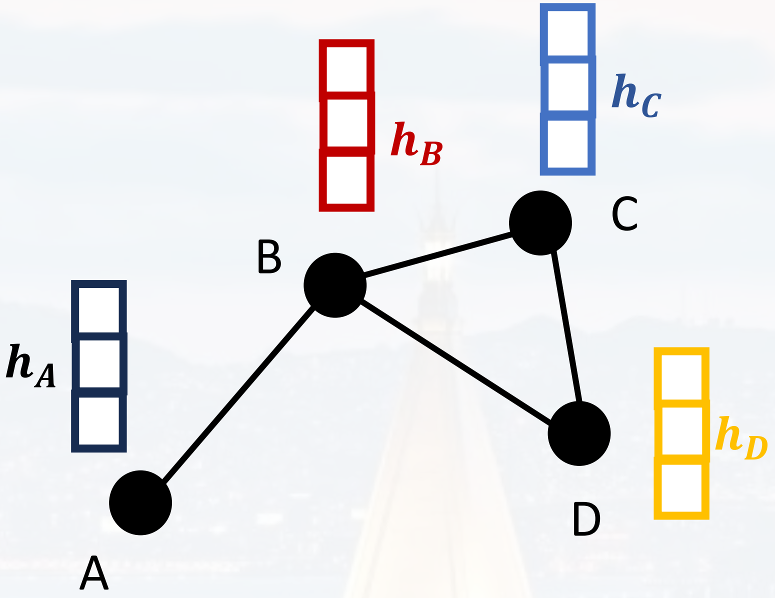


1st layer: one-hop neighborhood
2nd layer: two-hop neighborhood
etc

animation here

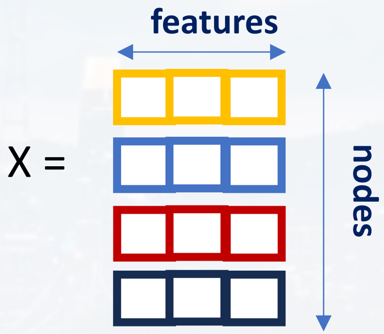
GNN: The ANN Part

Graph Convolution

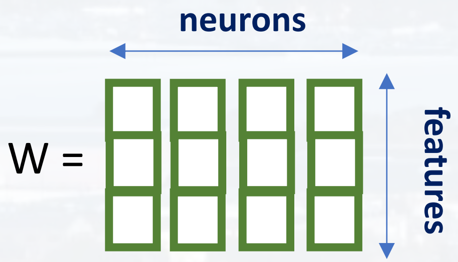


each node i has a **feature vector** h_i

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



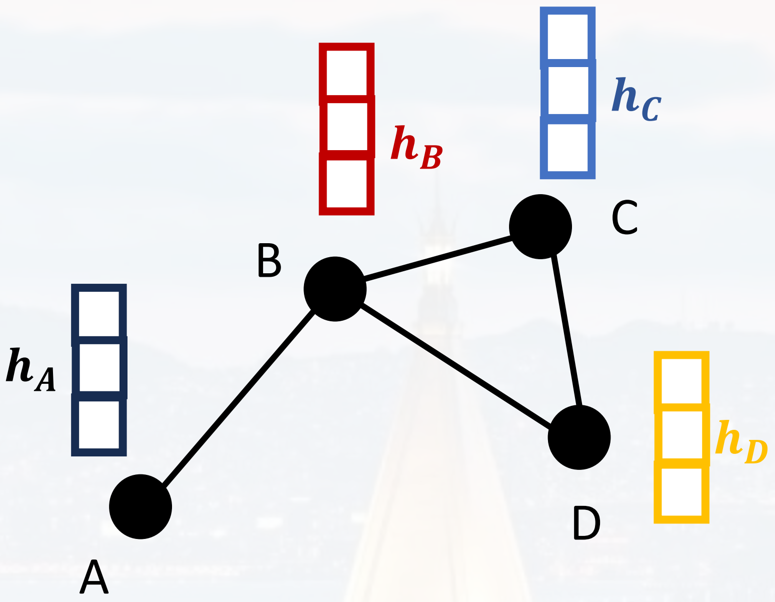
weight matrix W of shape
(number of node features, number of neurons)

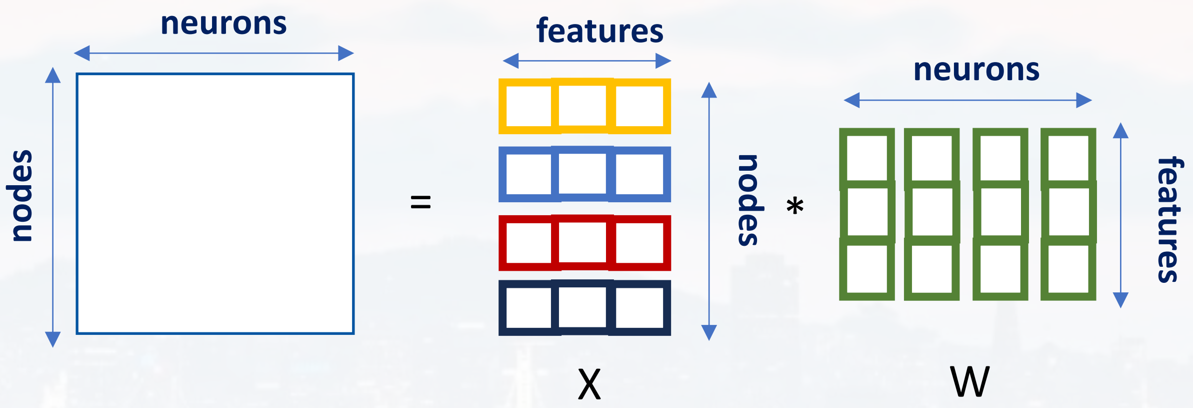


note: only **one** W for the entire graph
W is a **learnable**

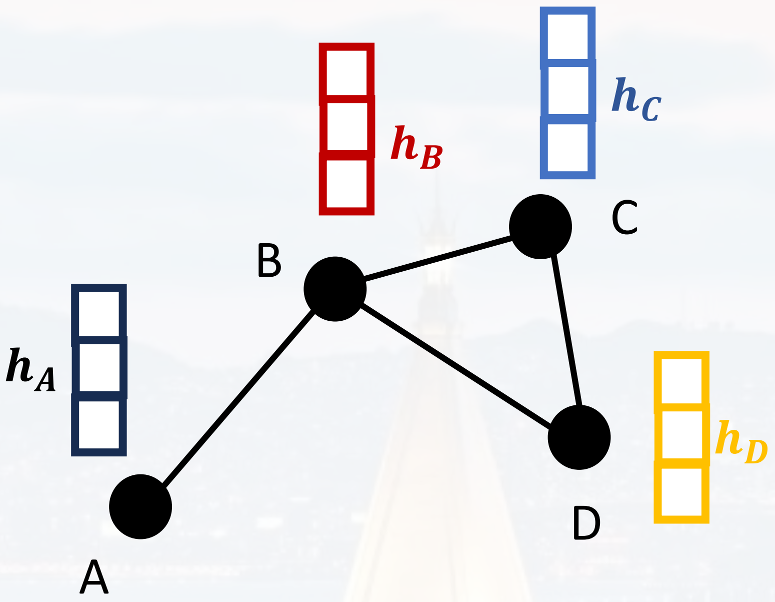
Graph Convolution

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

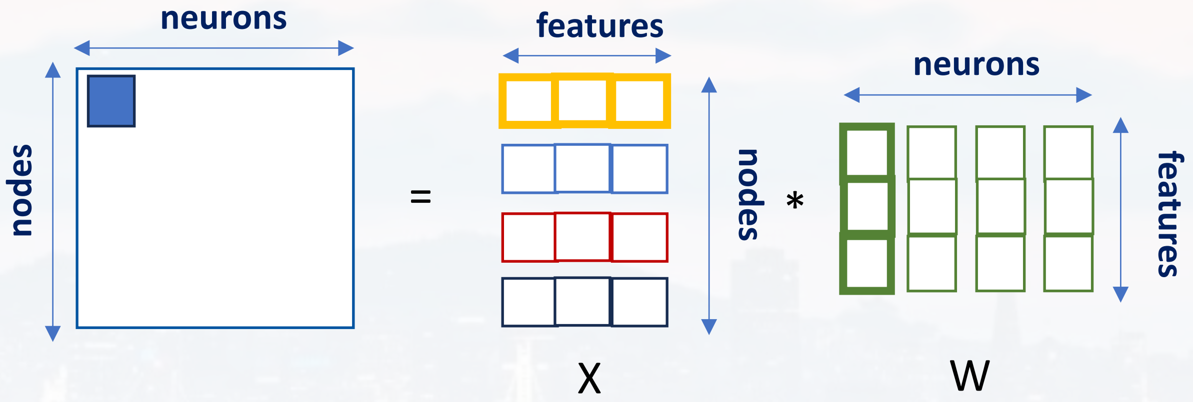




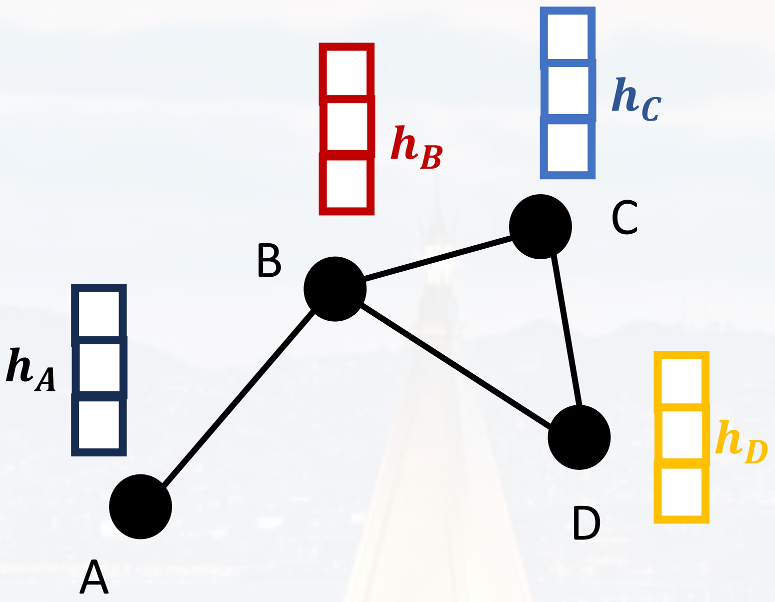
Graph Convolution



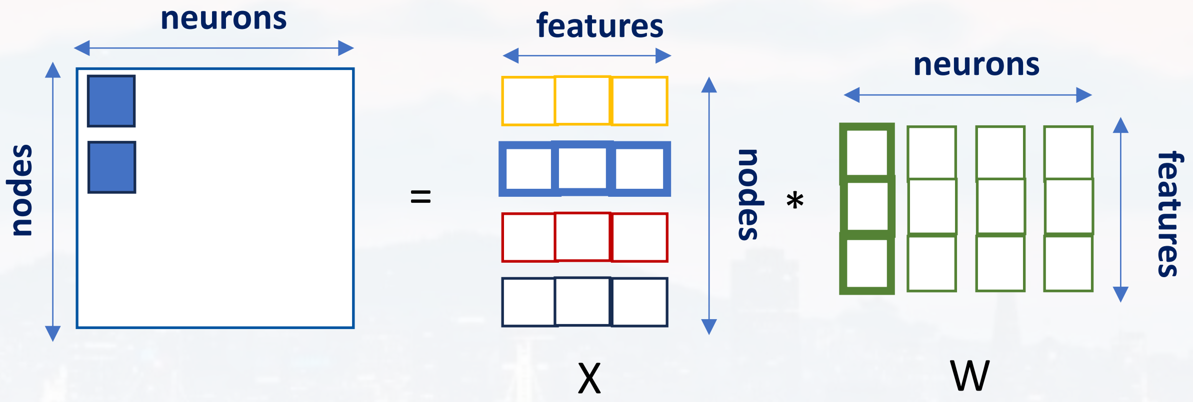
each node i has a **feature vector** h_i



Graph Convolution

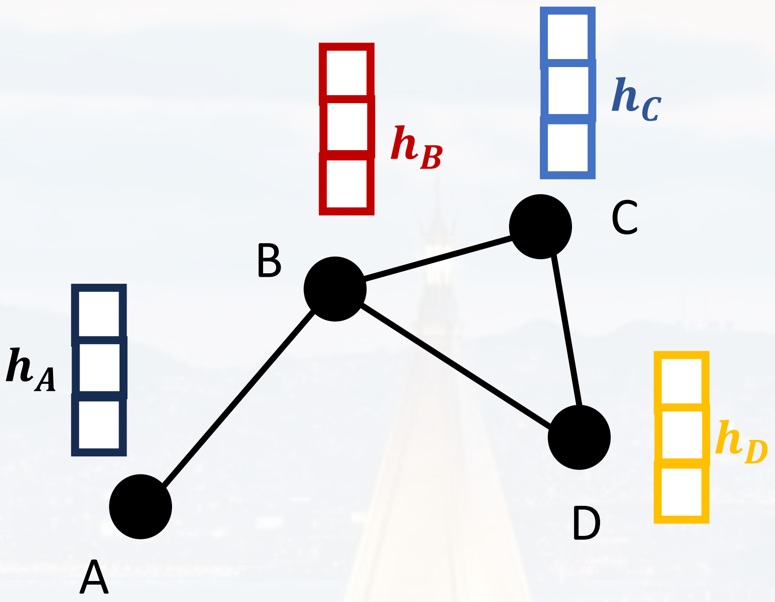


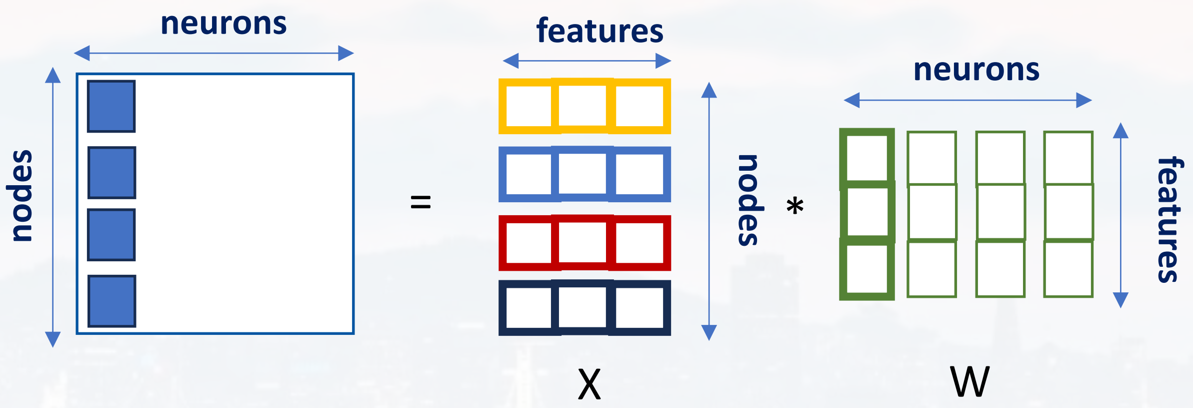
each node i has a **feature vector** h_i



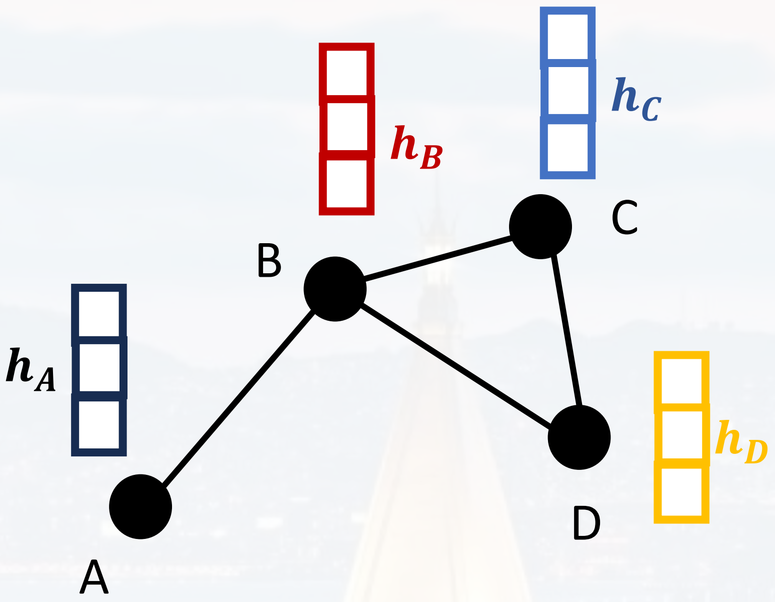
Graph Convolution

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

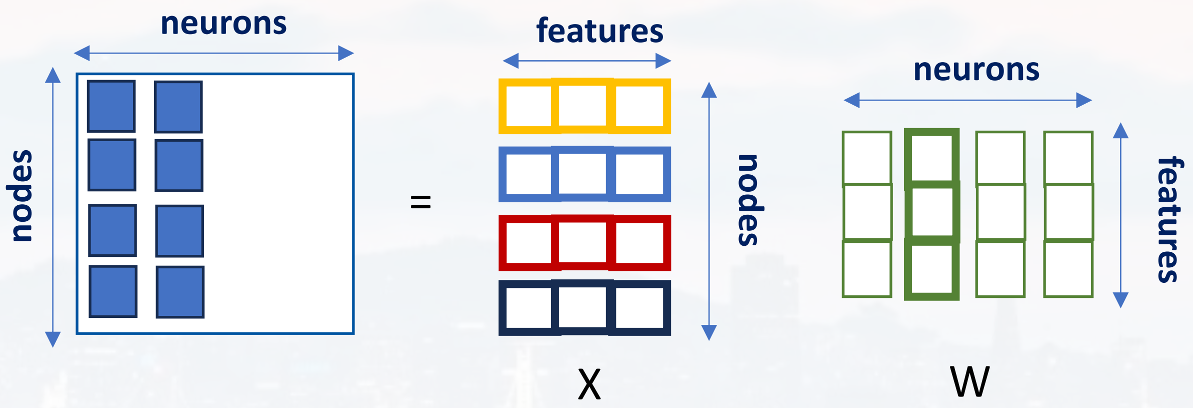




Graph Convolution

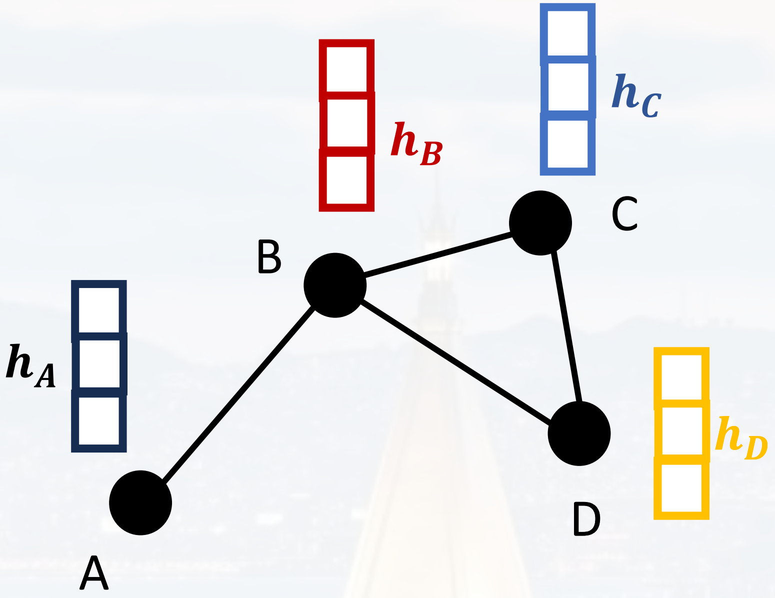


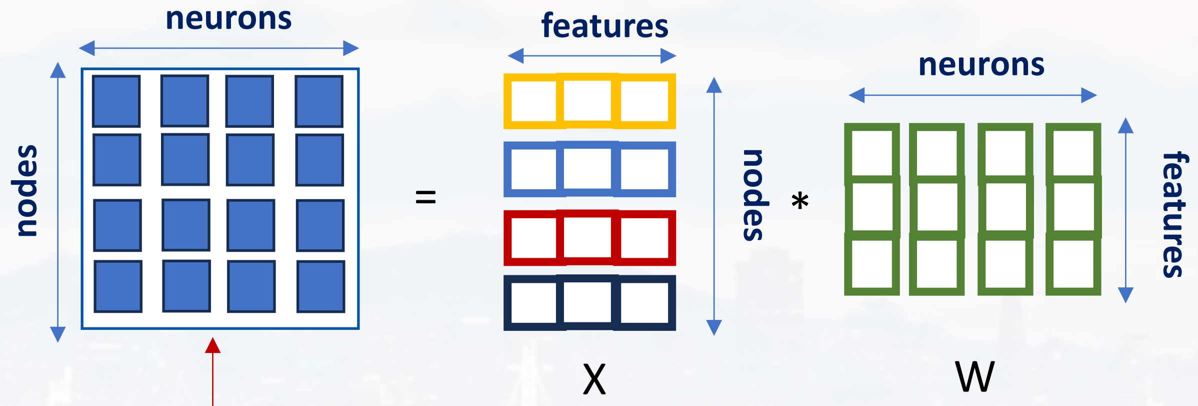
each node i has a **feature vector** h_i

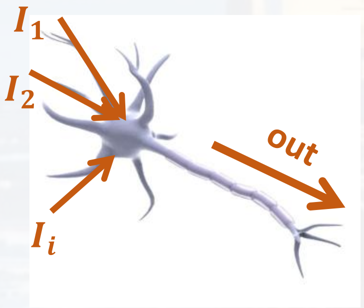


Graph Convolution

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





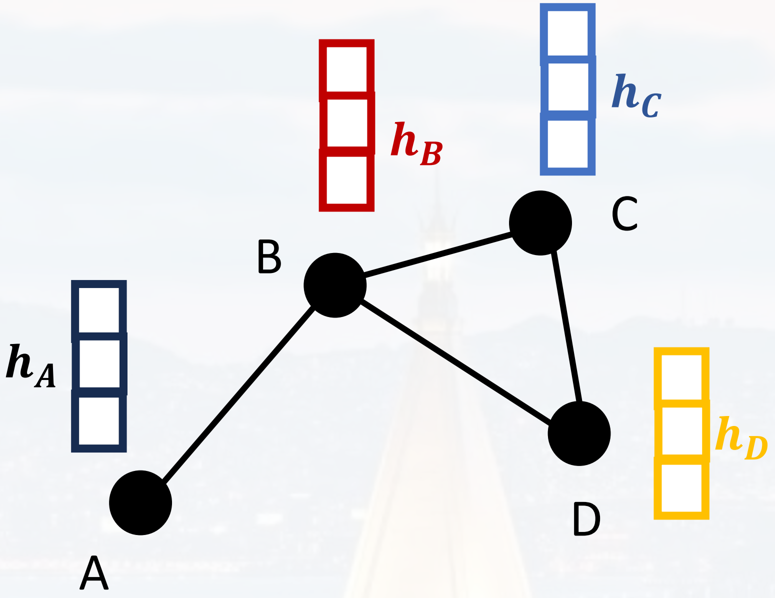


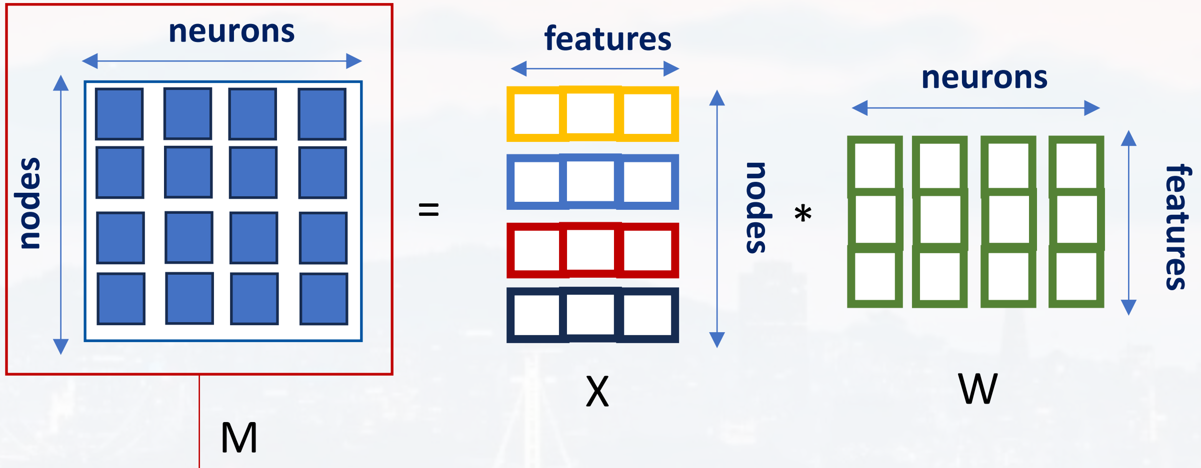
$$net = \sum_i I_i \cdot w_i + b$$

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

Graph Convolution

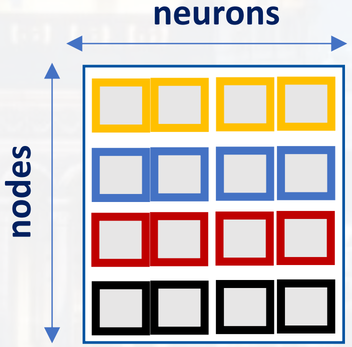
each node *i* has a **feature vector** h_i





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

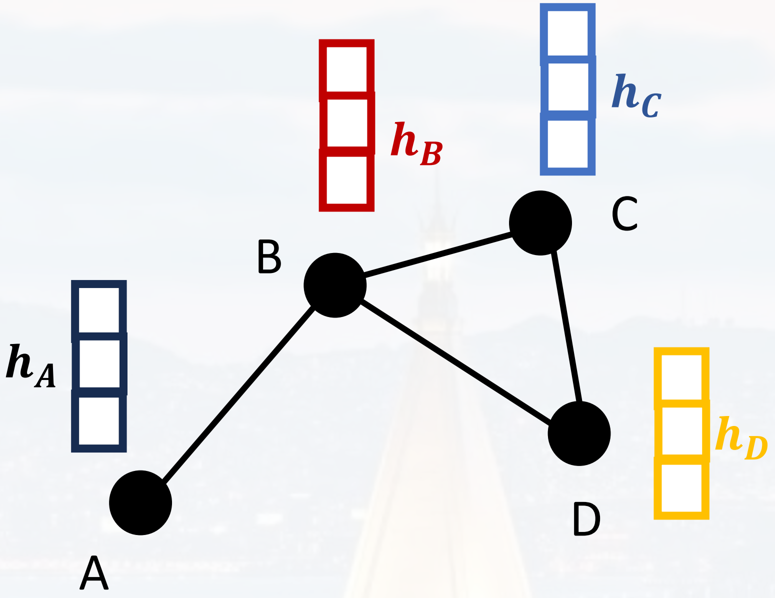
depending on W the output features may have different lengths then the input features



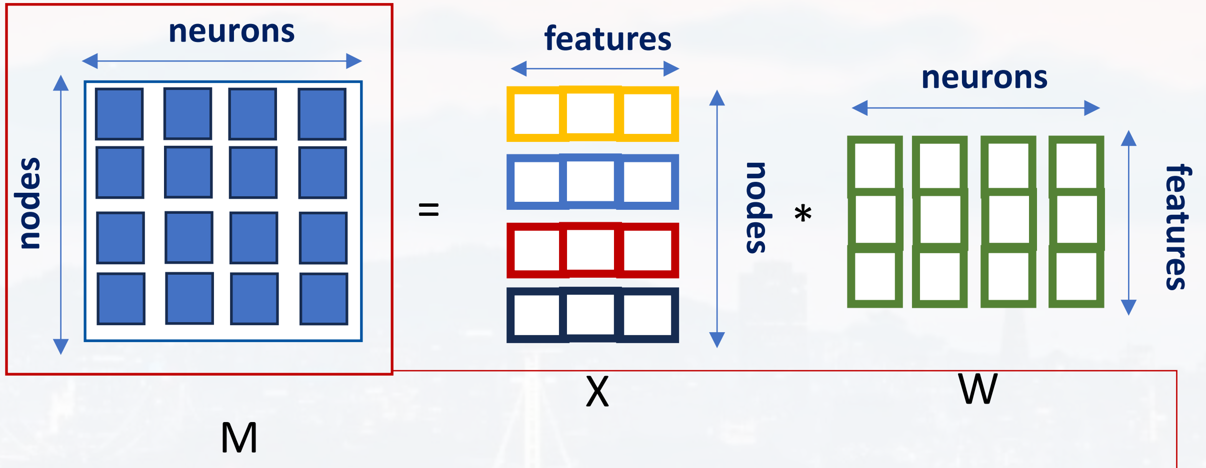
adjacency A

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

Graph Convolution

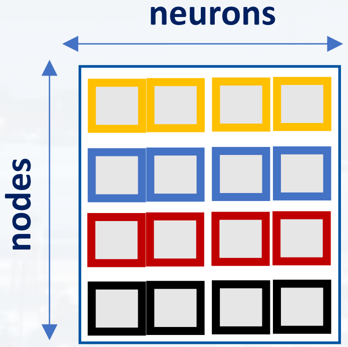


each node *i* has a **feature vector** h_i



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

(aka second convolution layer)

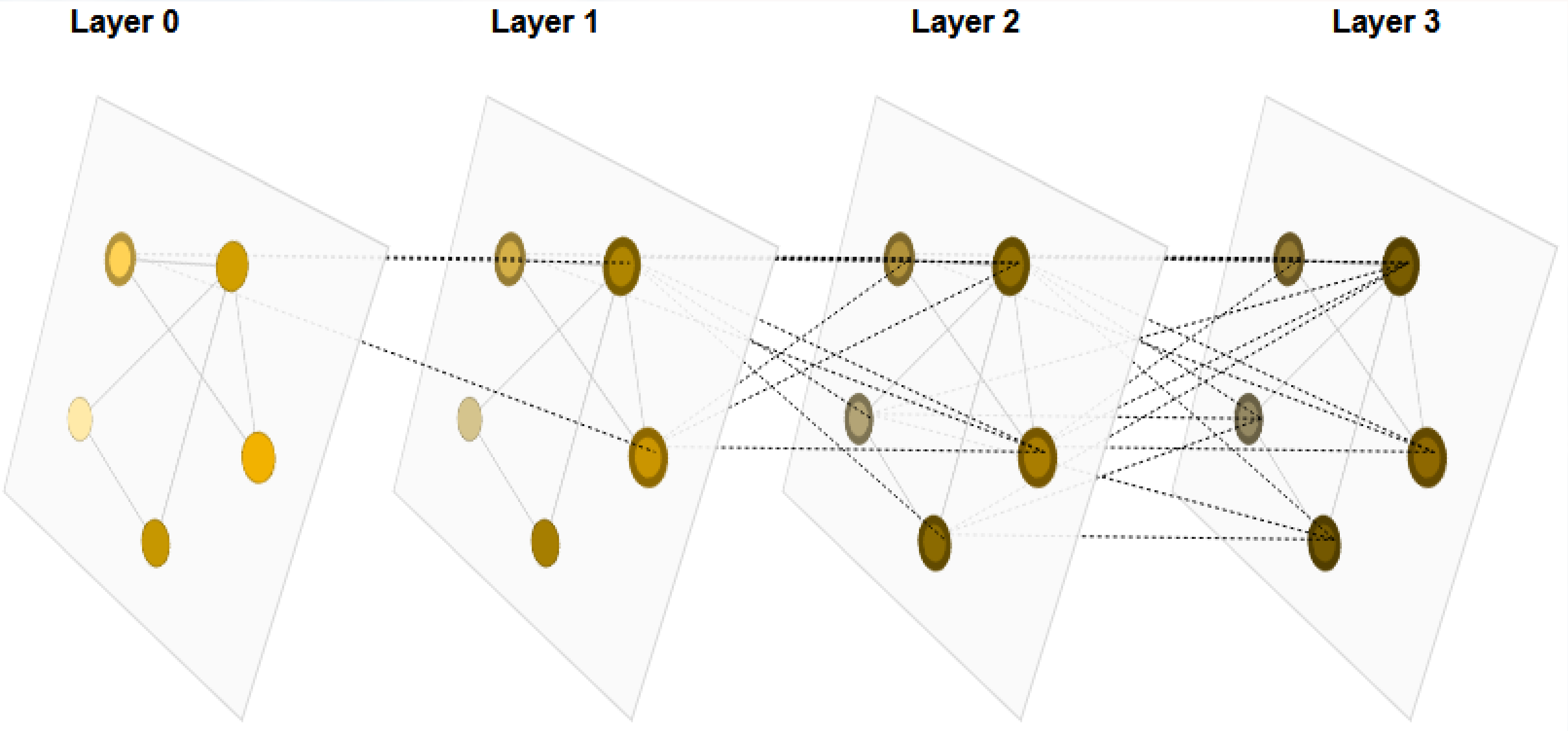


adjacency A

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

GNN: The ANN Part

Graph Convolution

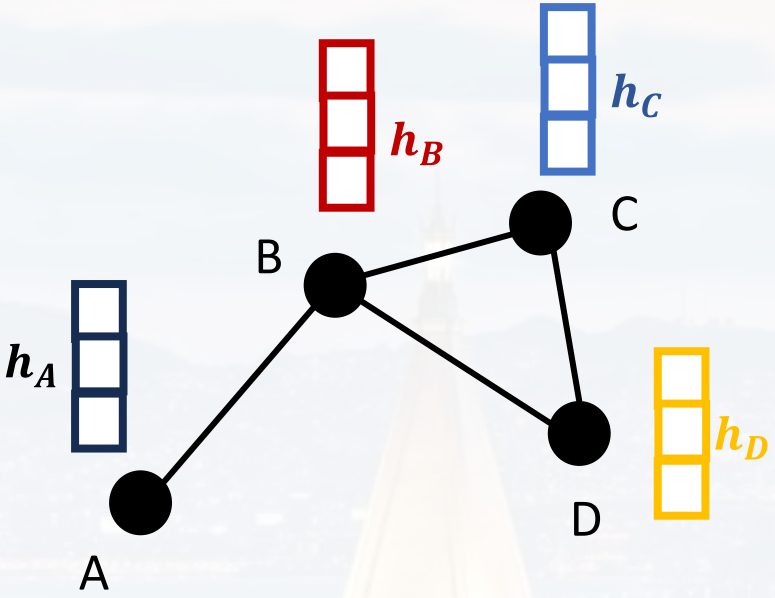


1st layer: one-hop neighborhood
2nd layer: two-hop neighborhood
etc

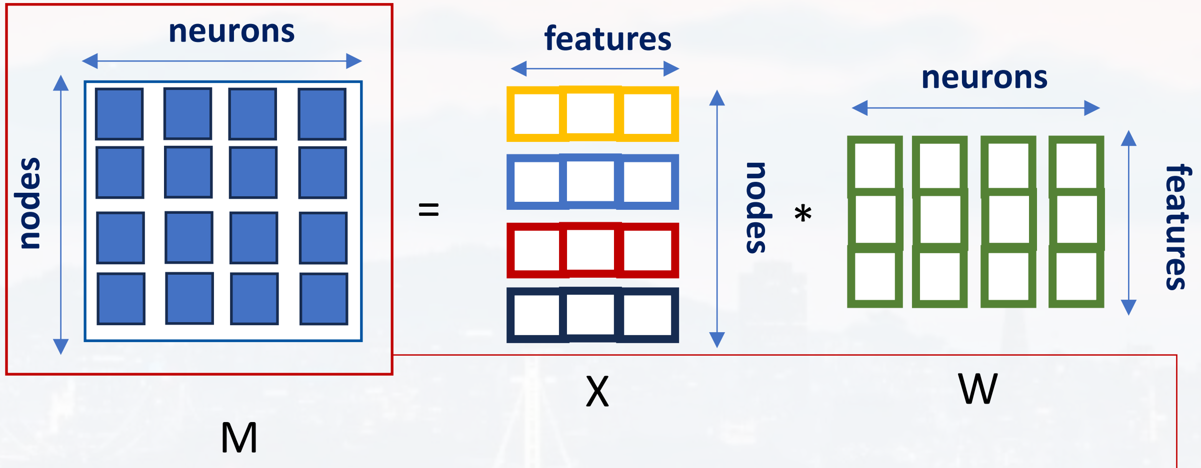
animation here

GNN: The ANN Part

Graph Convolution

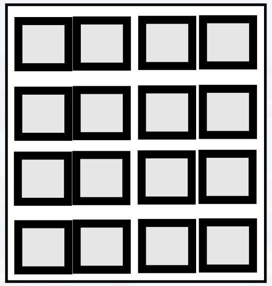


each node i has a **feature vector** h_i

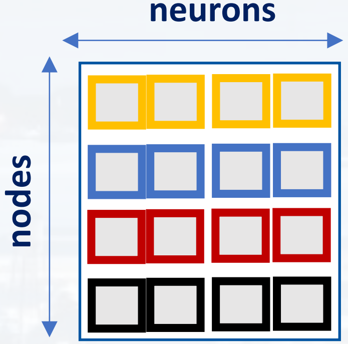


node classification

probabilities



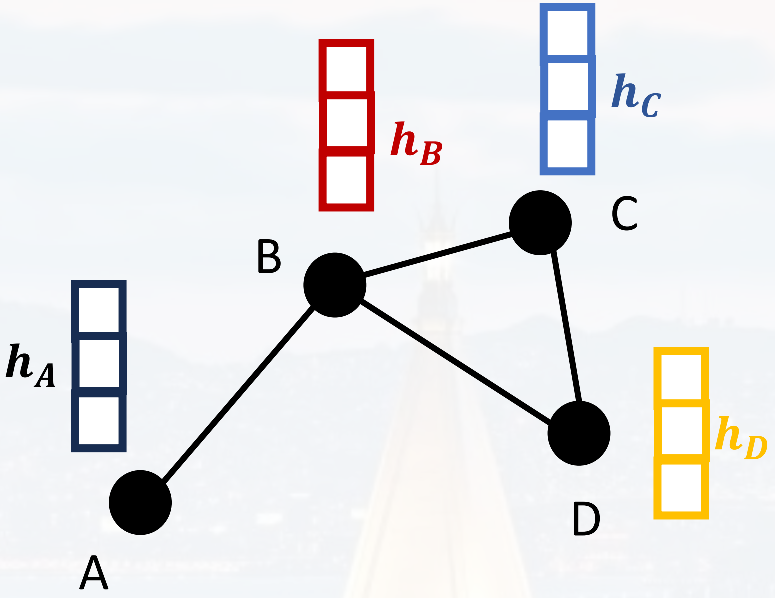
pass through a softmax layer



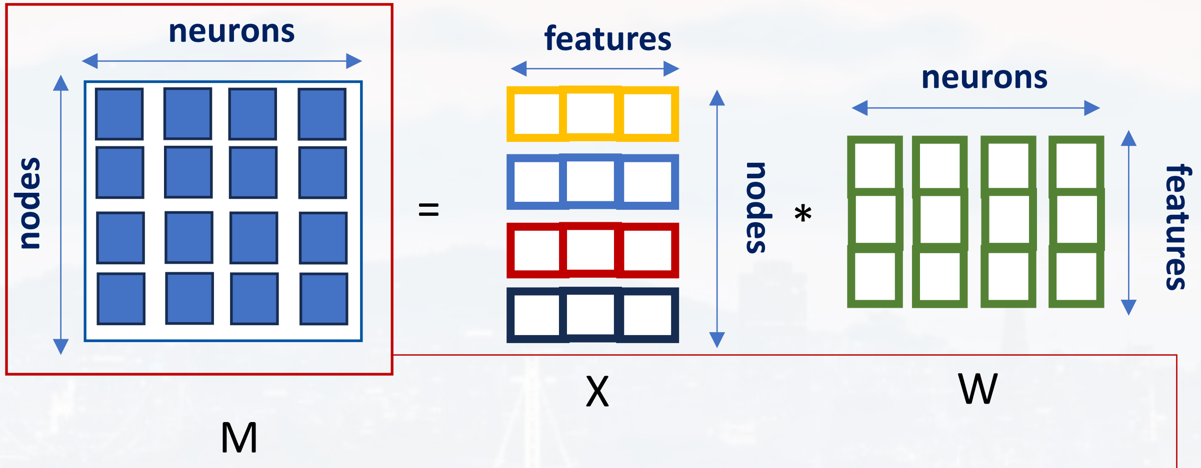
adjacency A

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

Graph Convolution



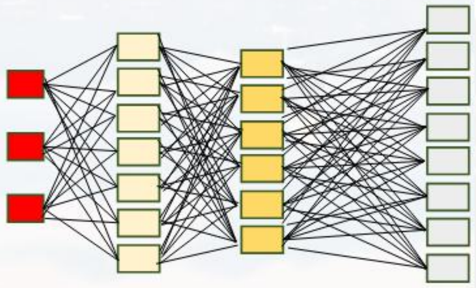
each node i has a **feature vector** h_i

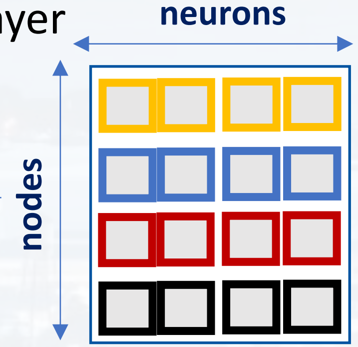


node regression

e.g. 3D coordinates, angles etc.

pass through a dense layer



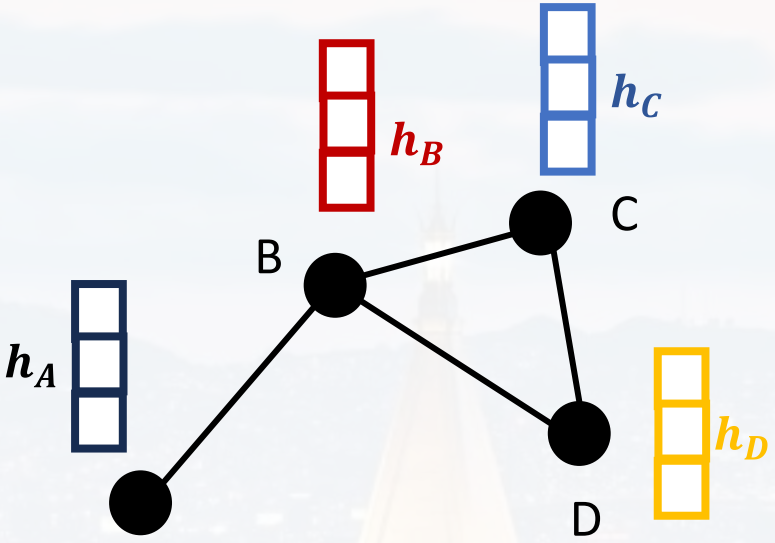


adjacency A

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

GNN: The ANN Part

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

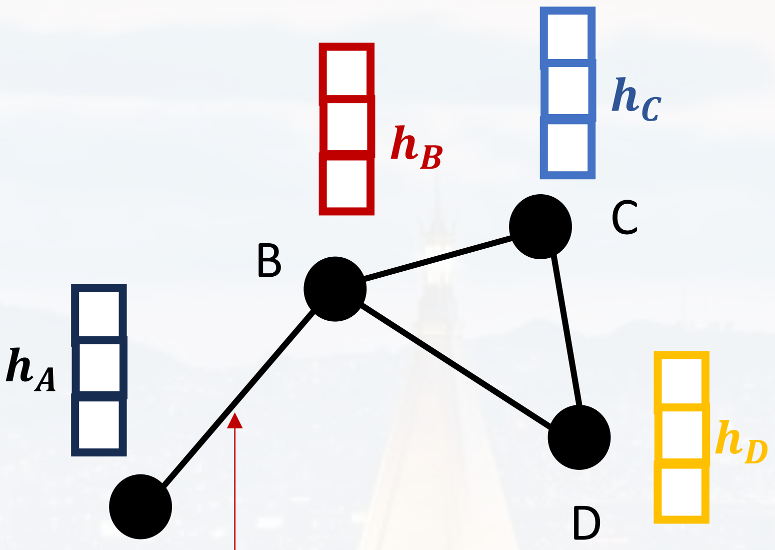
→ normalizing by $\frac{1}{\sqrt{d(n_i)}}$ and $\frac{1}{\sqrt{d(n_j)}}$

more information here


Matthew N. Bernstein

GNN: The ANN Part

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

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

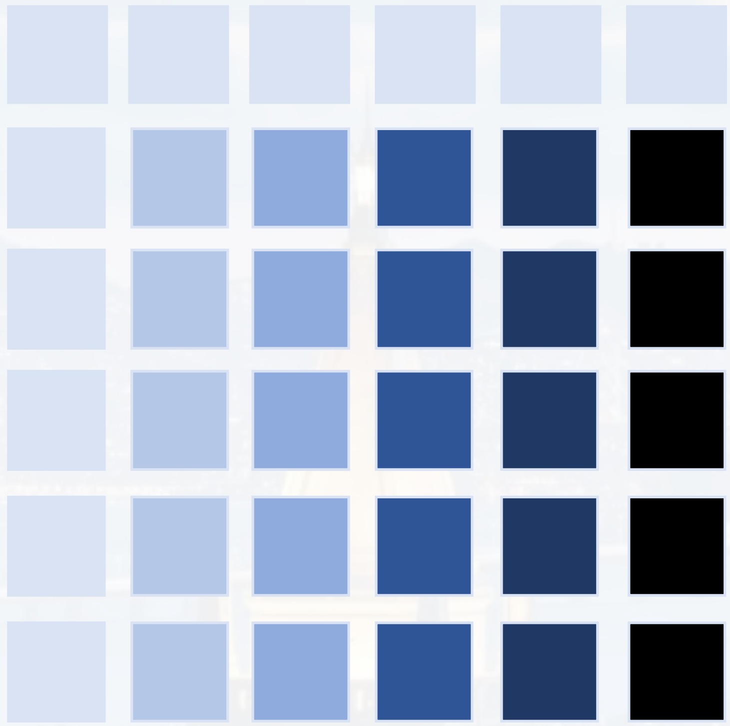
h_i: embedding vector of node i

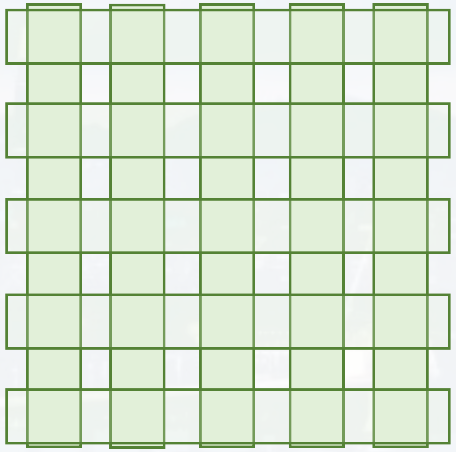
edge prediction (= classification), probability of an edge $p_{edge} = f(h_i, h_j)$

where f can be a **sigmoid** or **dense layer**

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)
```

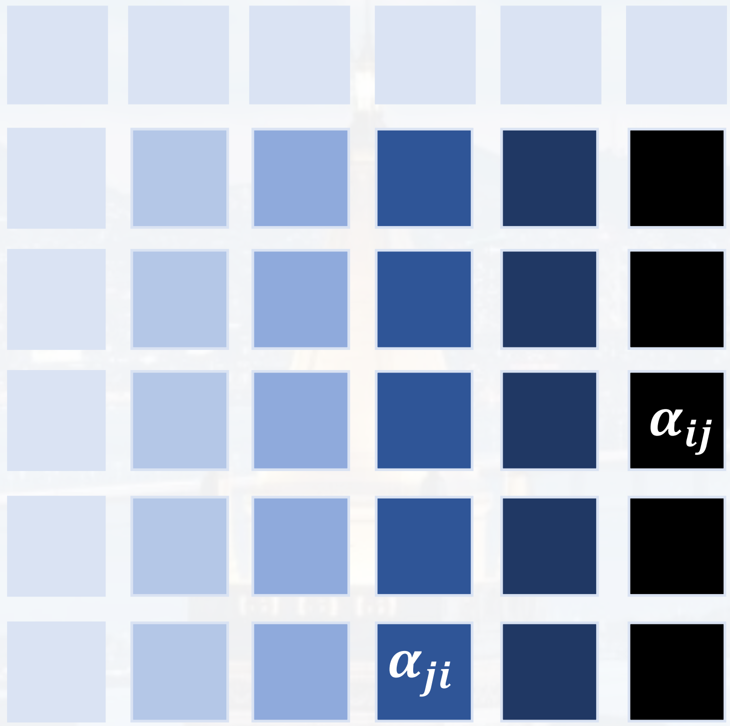
actual attention:

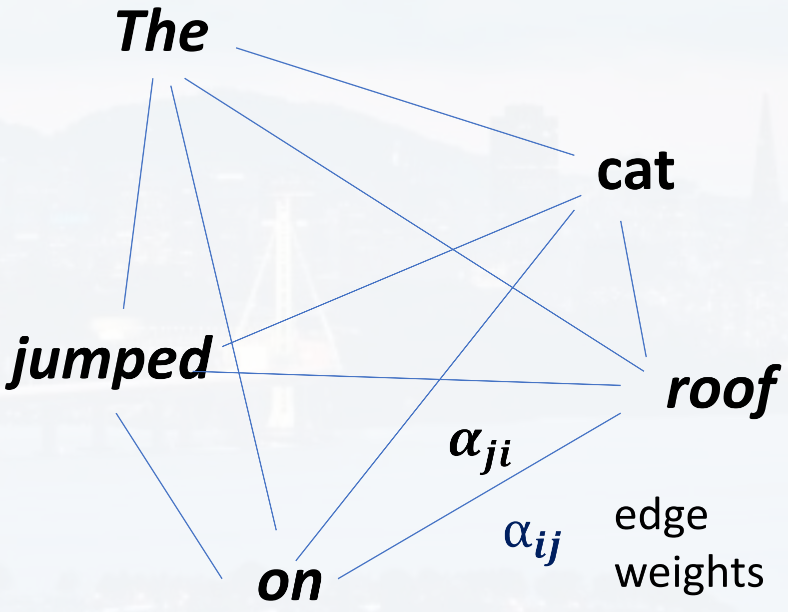
these weights are learnable,

no kernel assumed!

Graph Attention

“The cat jumped on the roof.”

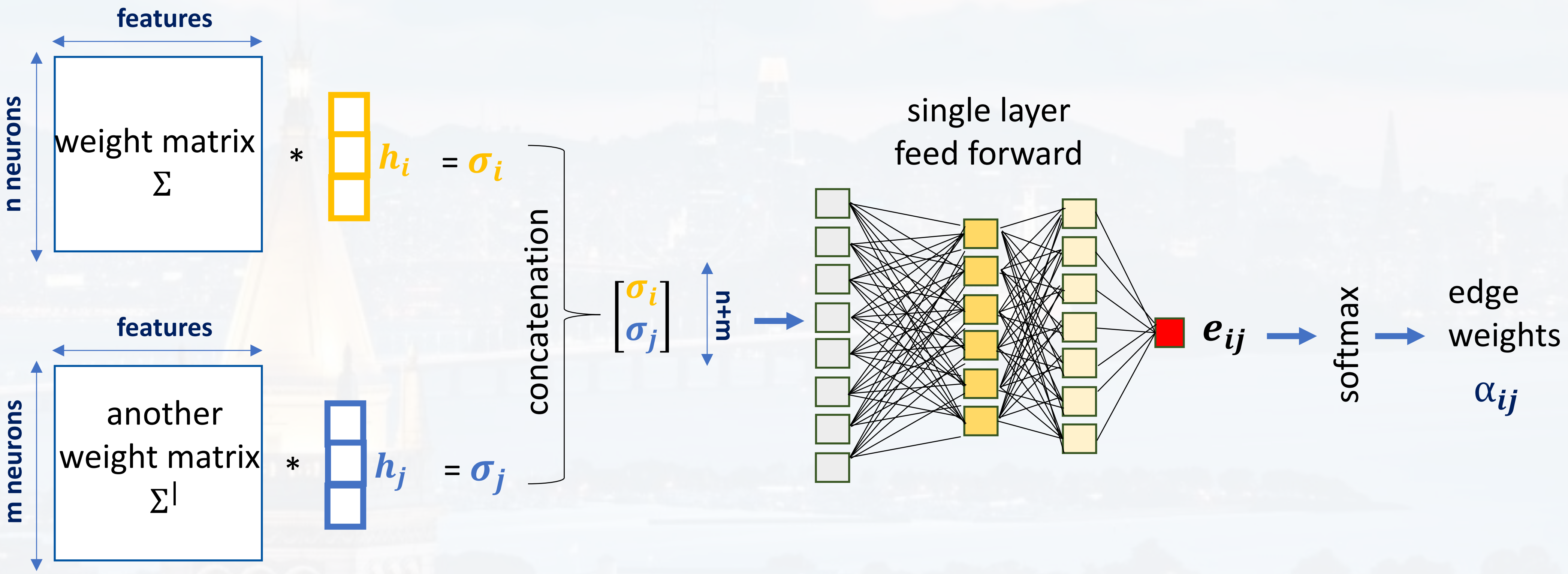




GNN: The ANN Part

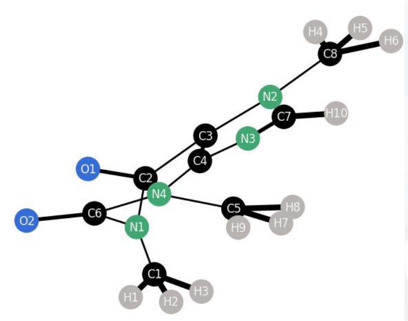
Graph Attention

Learning the weights!
(edge attributes)



GNN: The ANN Part

Graph Attention for graph classification/regression



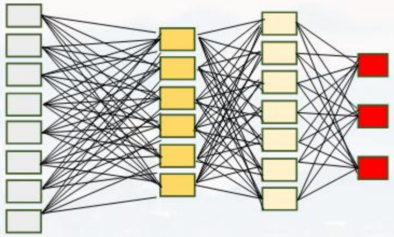
→ convolution GNN → node embeddings h_i

Each node contributes to the classification, but how?
→ **weighted** (= scalar) **average** of the embedding vectors h_i
→ **number of nodes** can **differ** between different graphs!

output $$\boxed{v} = \sum_i \alpha_i h_i = \sum_i \boldsymbol{softmax}\,[\boldsymbol{a}^T \boldsymbol{tanh}(\boldsymbol{W}\boldsymbol{h_i})]h_i$$

$\boldsymbol{a}^T$ and $\boldsymbol{W}$ are trainable

dense layer+ softmax for classification
or
dense layer for regression



Outline

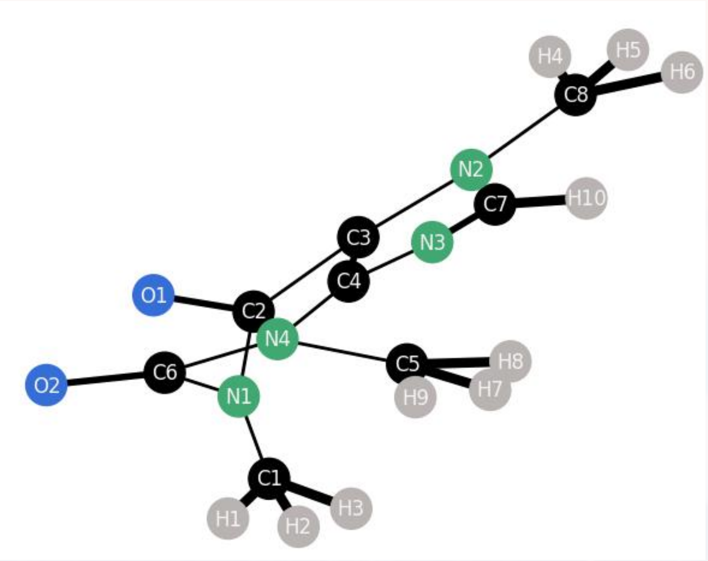
- What is a Graph
- The ANN Part
- **PyTorch Example**

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

```
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%
```

see `Graph_III.ipynb`

```
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!