Lecture 16:

Graph Neural Networks (GNN)

Markus Hohle

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Fall 2025

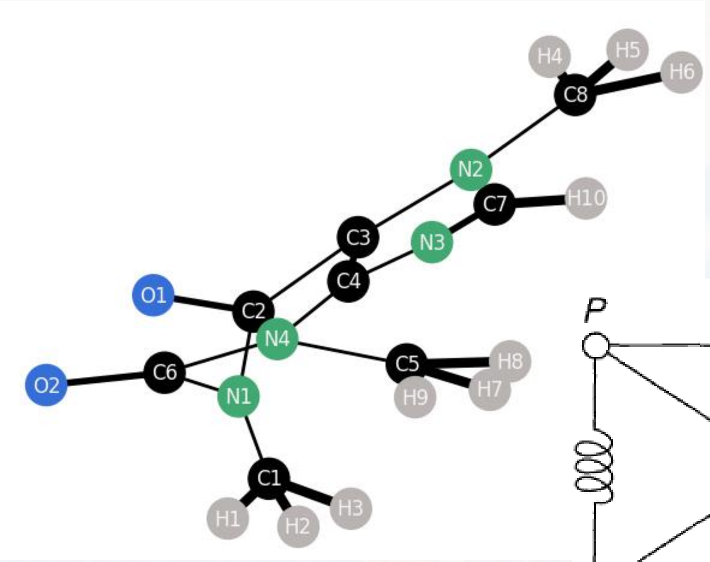
Outline

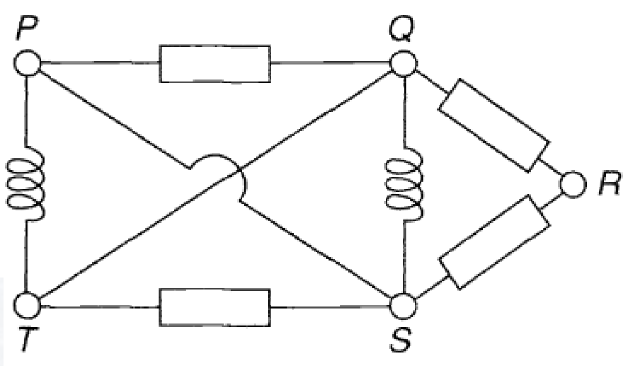
- What is a Graph
- The ANN Part
- PyTorch Example

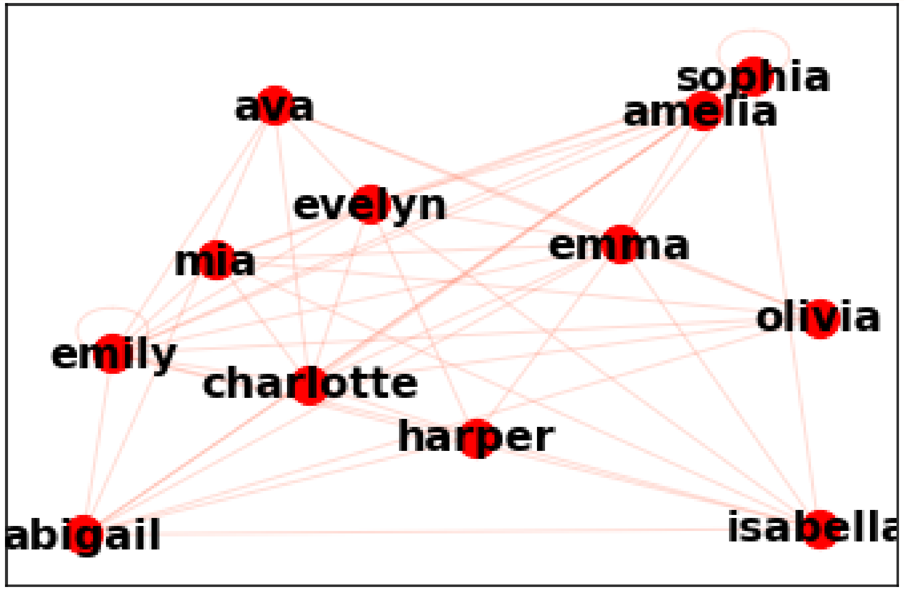
Outline

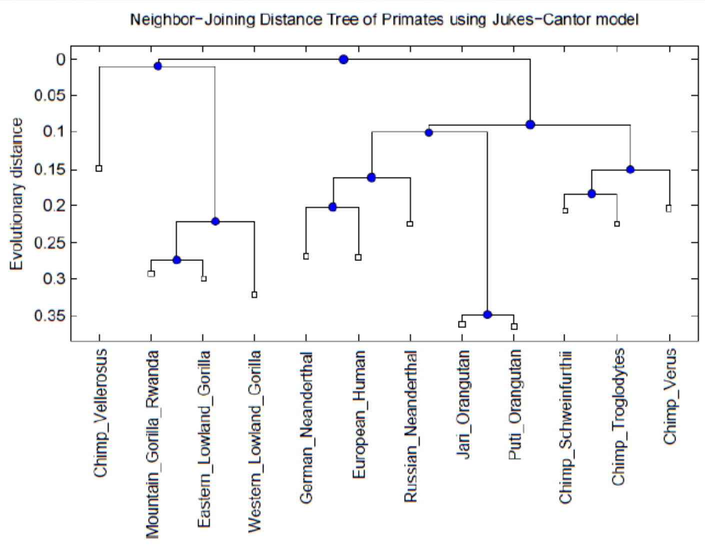
- **What is a Graph**
- The ANN Part
- PyTorch Example

GNN: What is a Graph

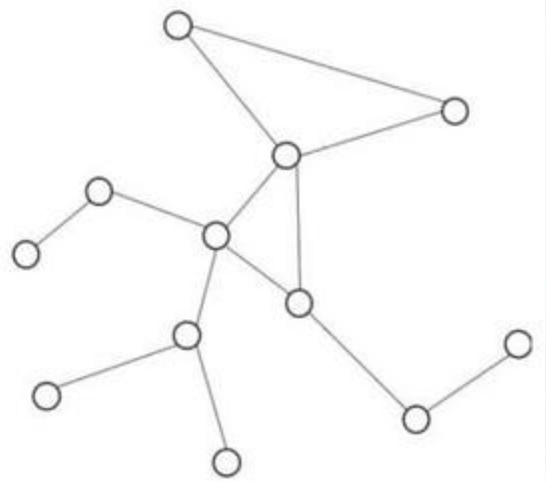


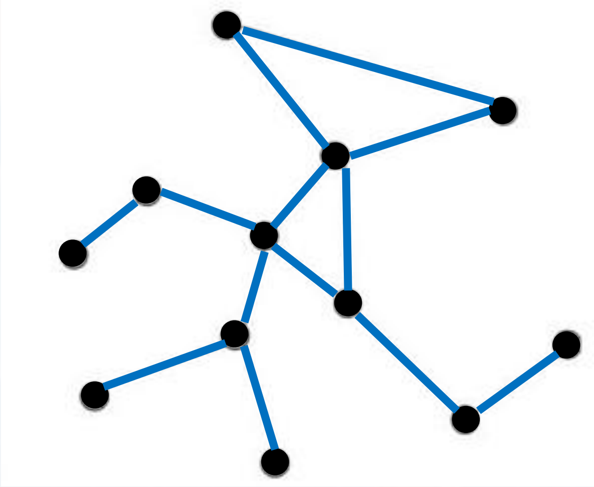






https://doi.org/10.1016/j.aiopen.2021.01.001





Graph G

nodes N (vertices V)

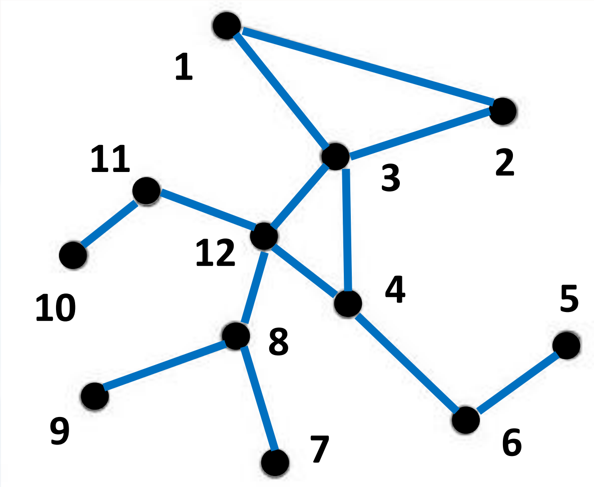
edges E

$G = G(N, E)$

- 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 $\boldsymbol{A}$**

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

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

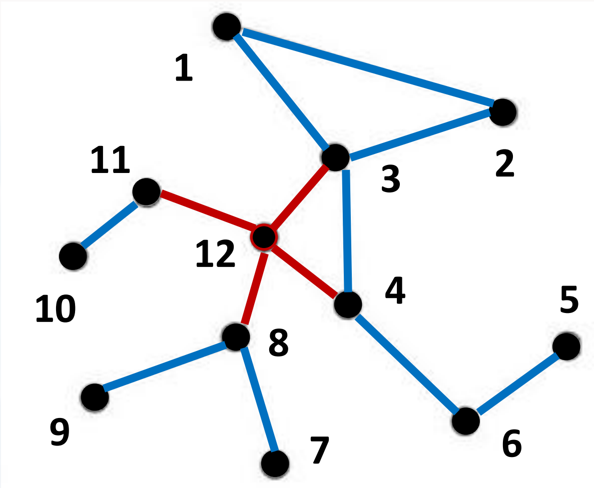
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

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

GNN: What is a Graph



structural information: **adjacency matrix 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)

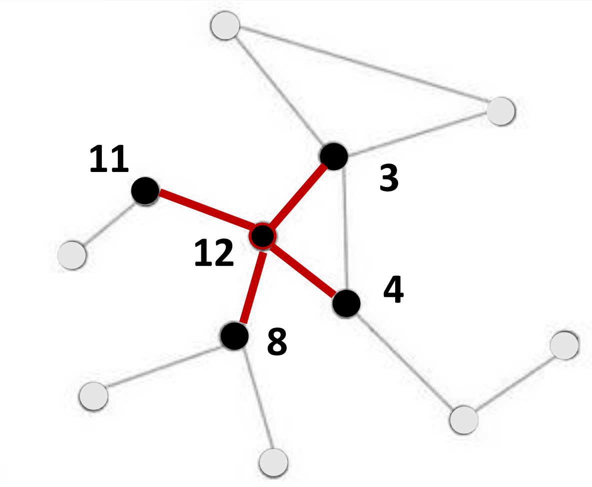
edges E

node 12 has four first degree neighbors

degree *d* of a node

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

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



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

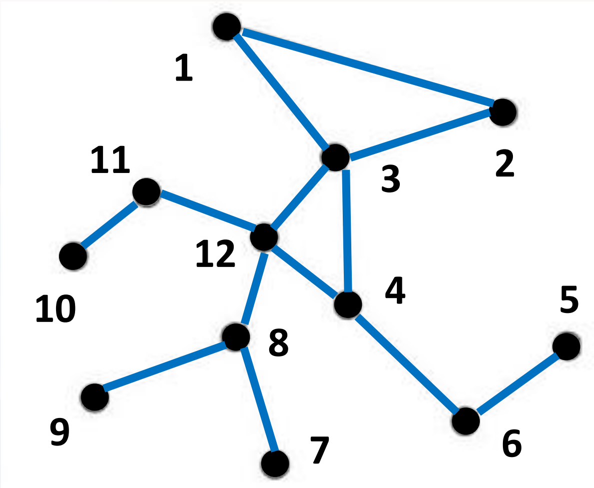
$$d(n_i) = \sum_j A_{ij}$$

first degree neighborhood $\mathscr{N}$

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

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

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

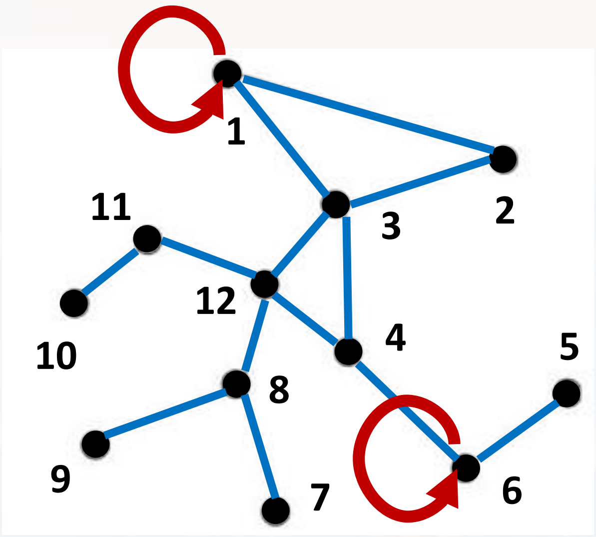
Graph $G = G(N, E)$

nodes N (vertices V)

edges E

A graph can have **loops**

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



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)

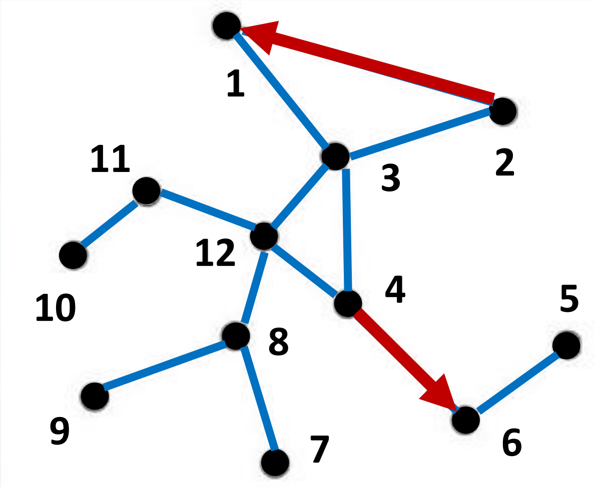
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

A graph can have **loops**

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



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)

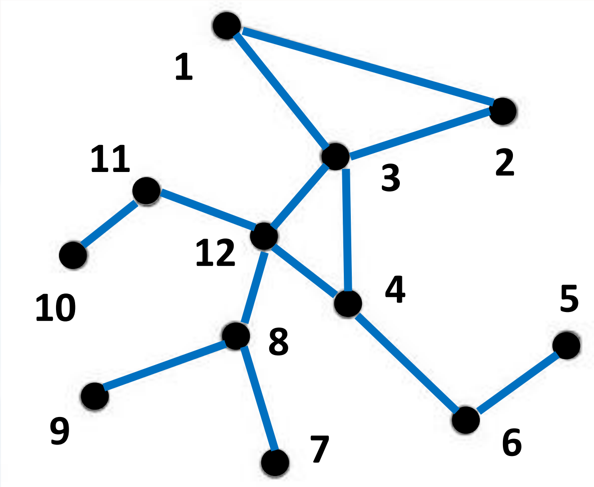
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

A graph can be **directed**

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



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)

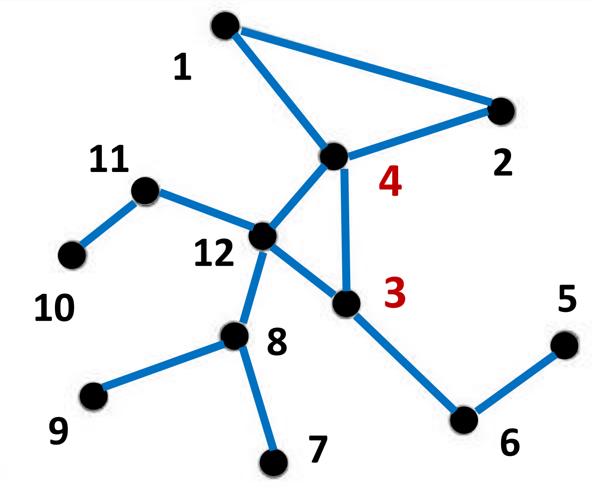
$A_{ij} = 0$ else

Graph $G = G(N, E)$

nodes N (vertices V)
edges E

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

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



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

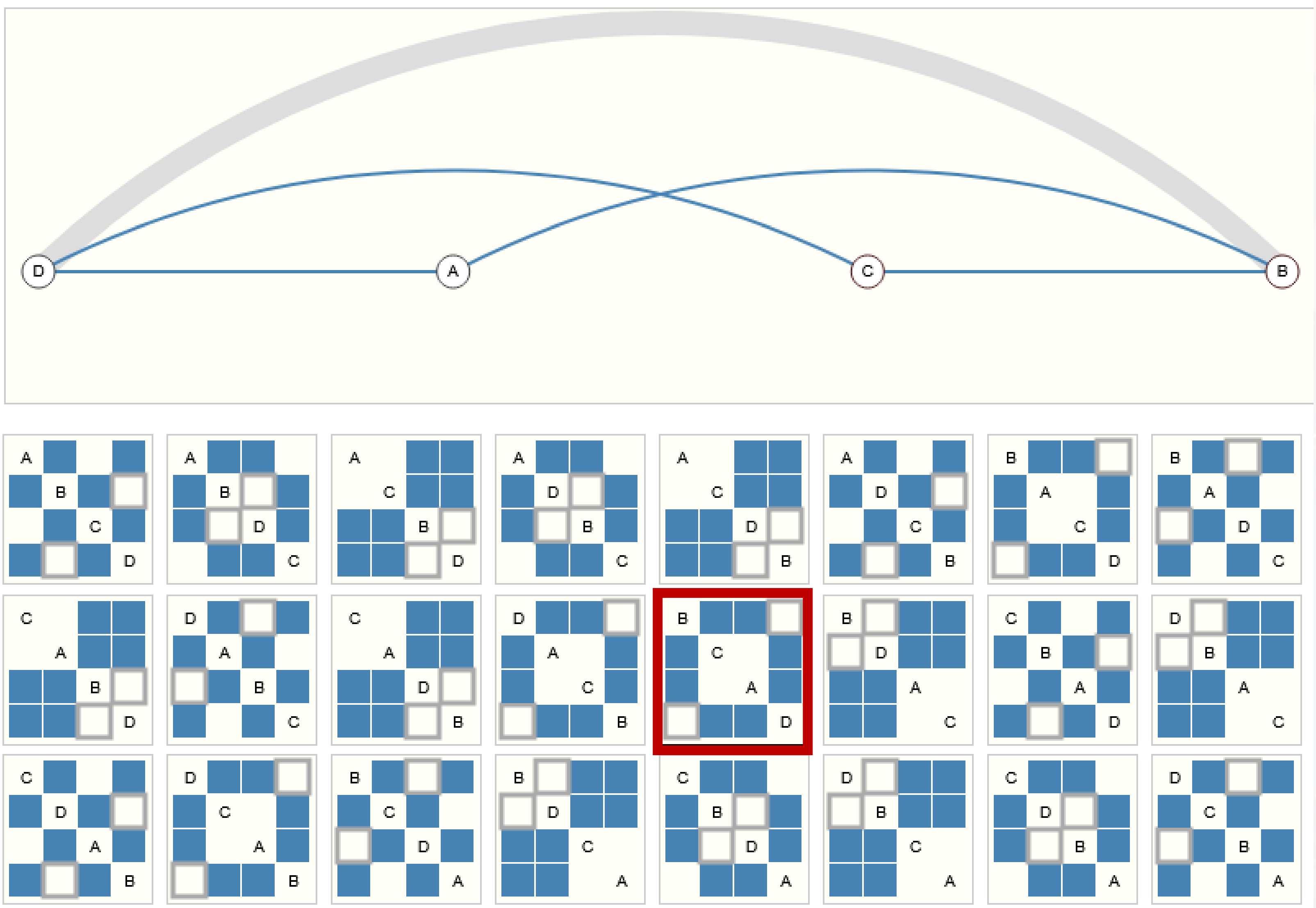
Each graph can be represented by **N!** adjacency matrices!

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

GNN:

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

[animation here]

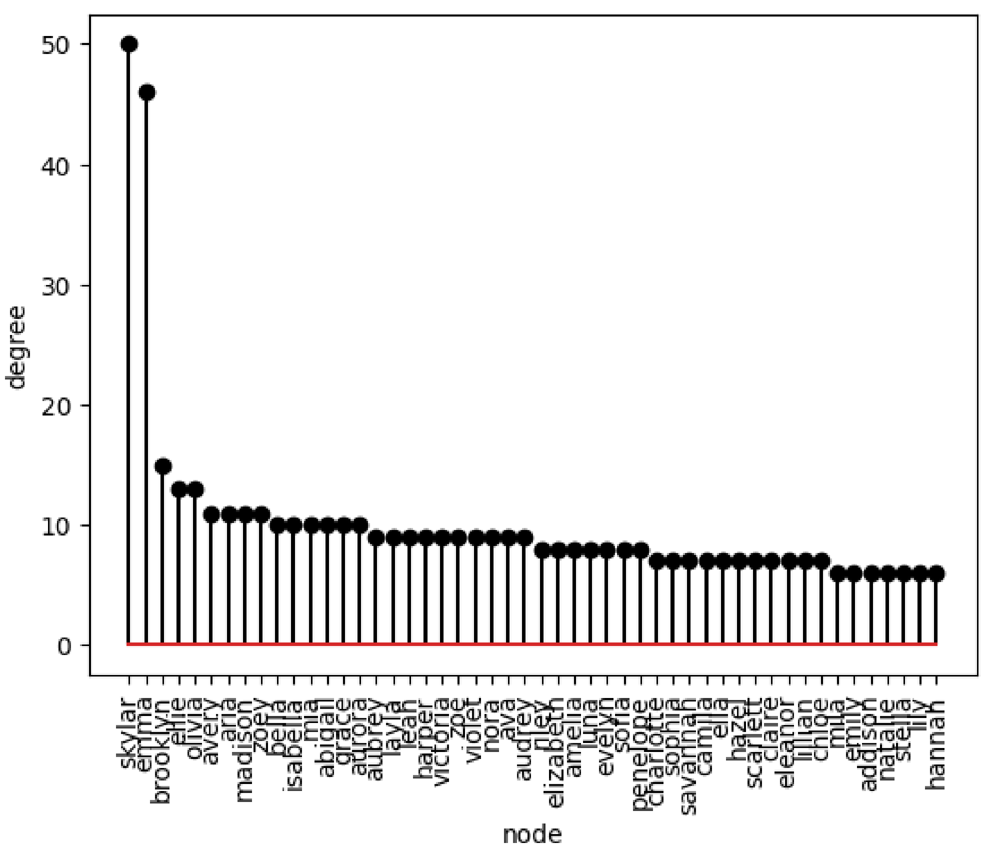


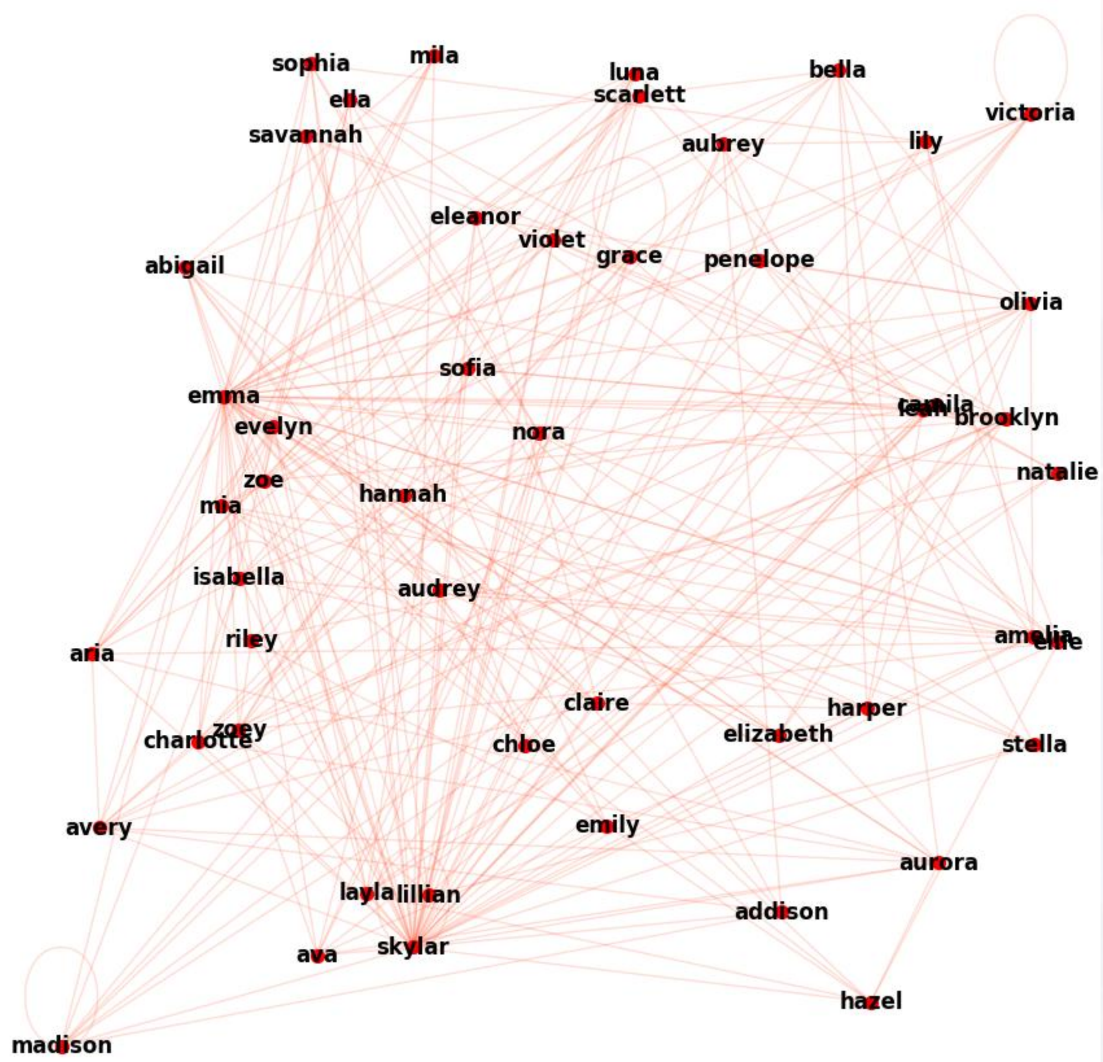
GNN:

visualizing a graph:

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

see: Graph_I.ipynb



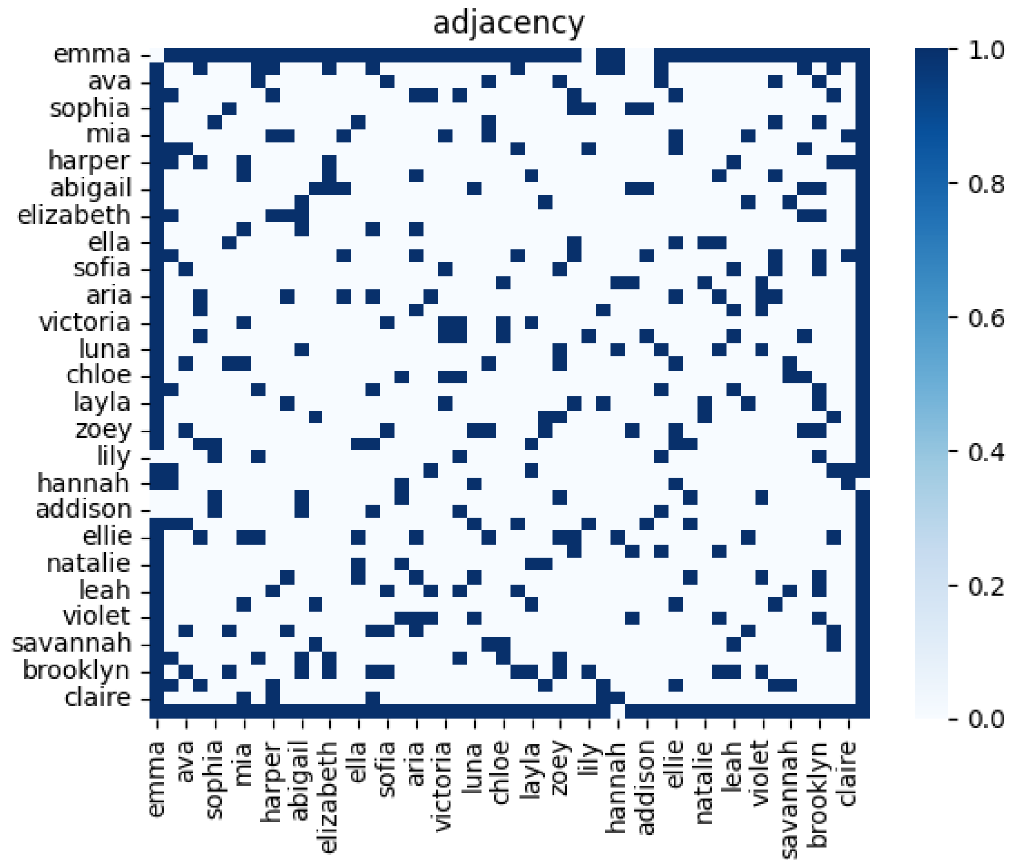


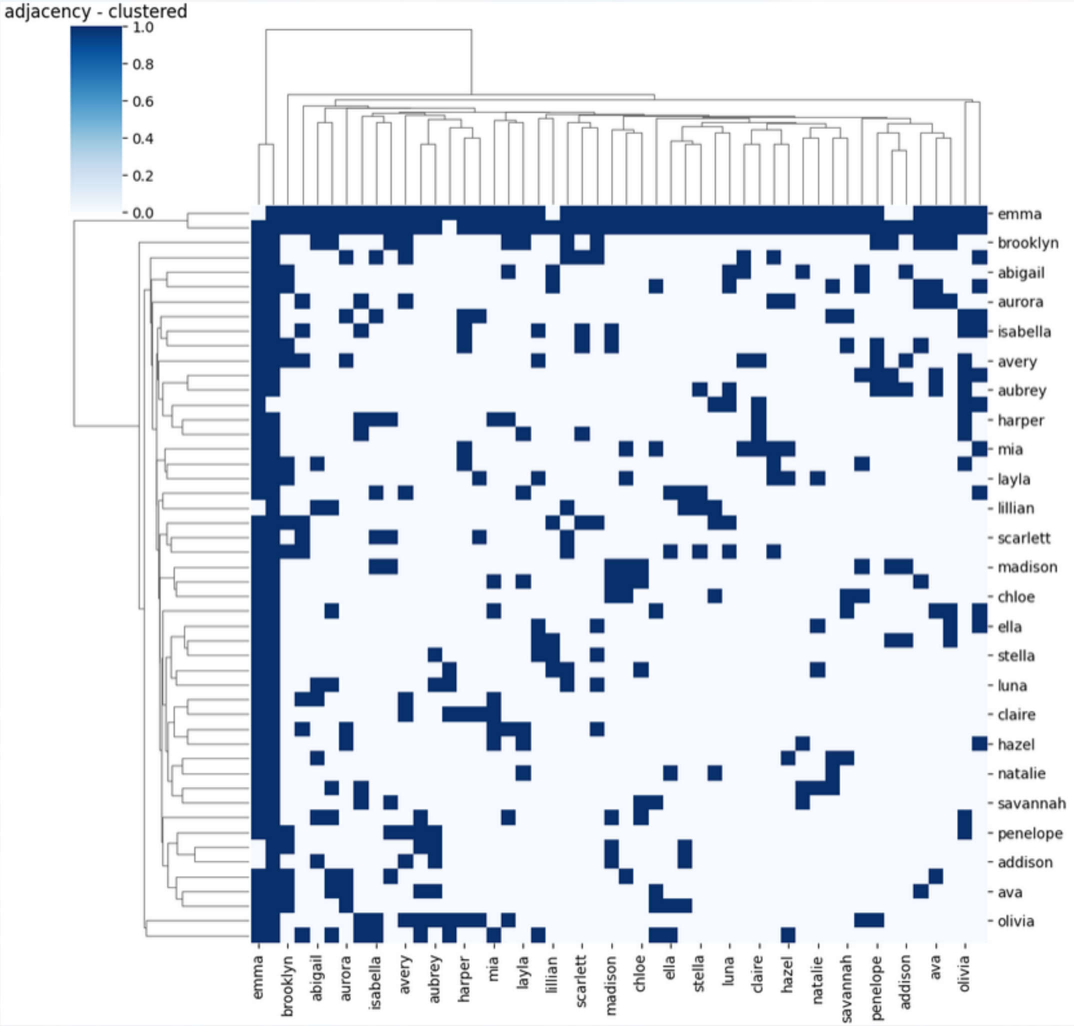
GNN:

visualizing a graph:

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

see: `Graph_I.ipynb`



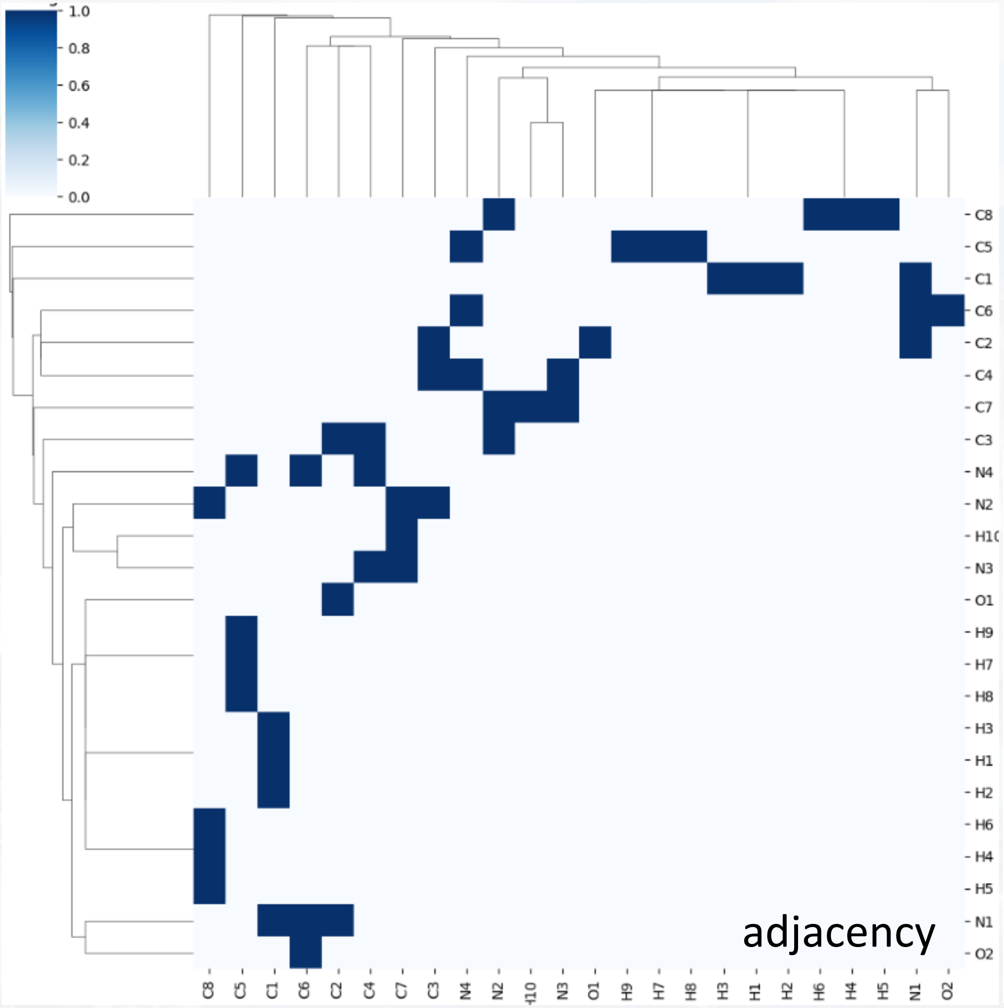


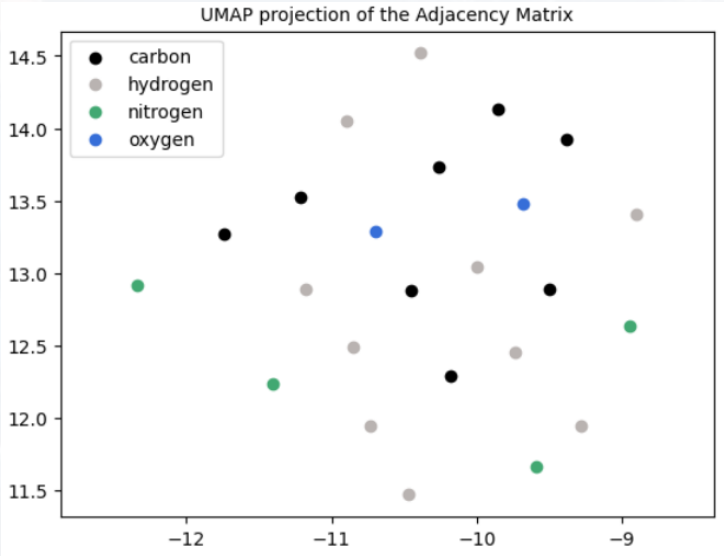
building and visualizing a **weighted** graph:

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

see: `Graph_II.ipynb`

Caffein molecule



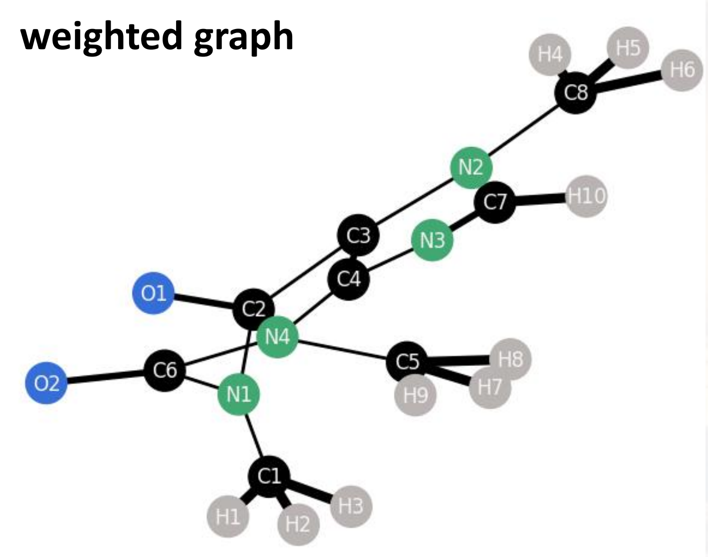


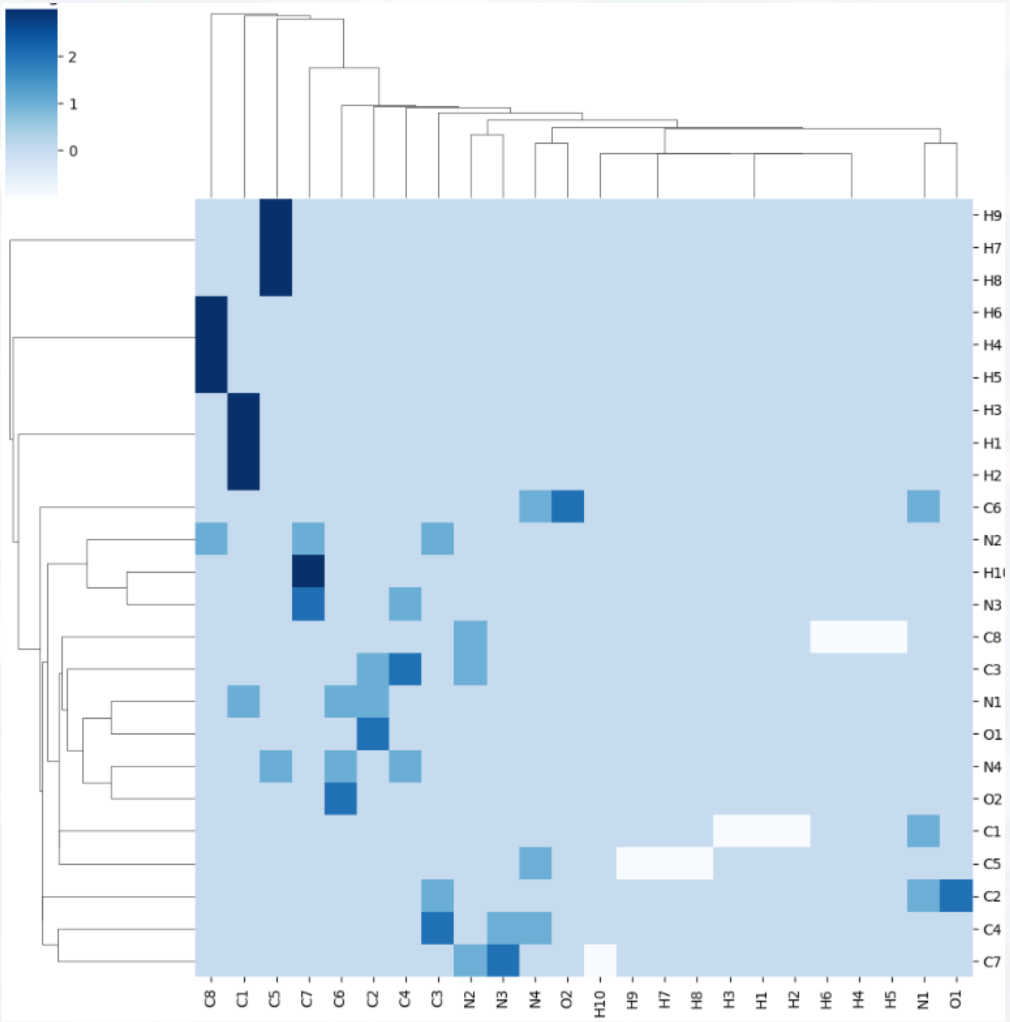
building and visualizing a **weighted** graph:

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

see: `Graph_II.ipynb`

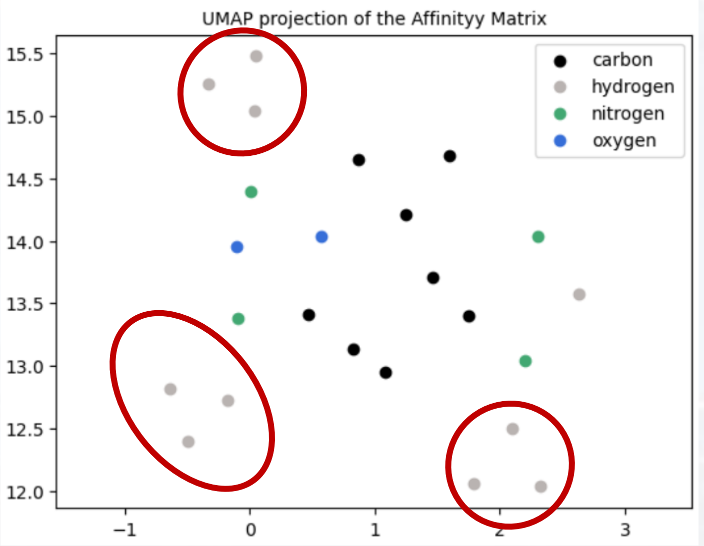
Caffein molecule





binding affinity (weights)

hydrogen atoms are at the edges of the molecule!



Outline

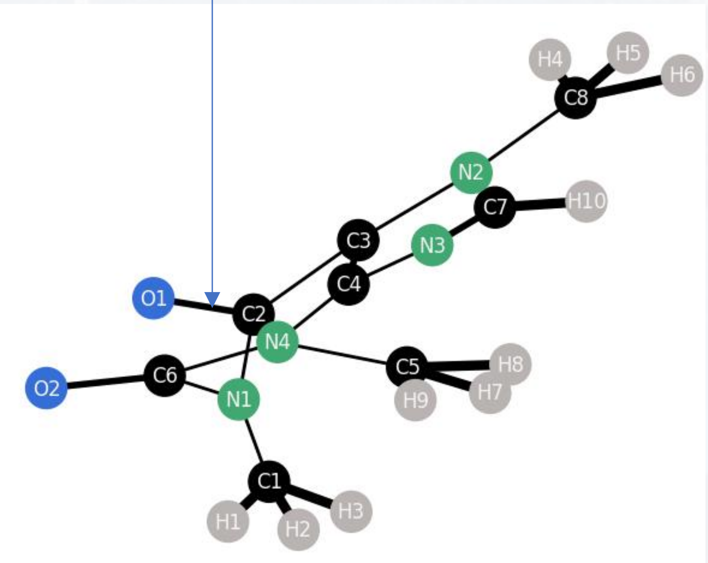
- What is a Graph
- **The ANN Part**
- PyTorch Example

GNN: The ANN Part

What we can learn:

- node classification
- join nodes with similar properties to hyper nodes
- edge attributes, weights (weighted graph)
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule aromatic)

weight: bond strength

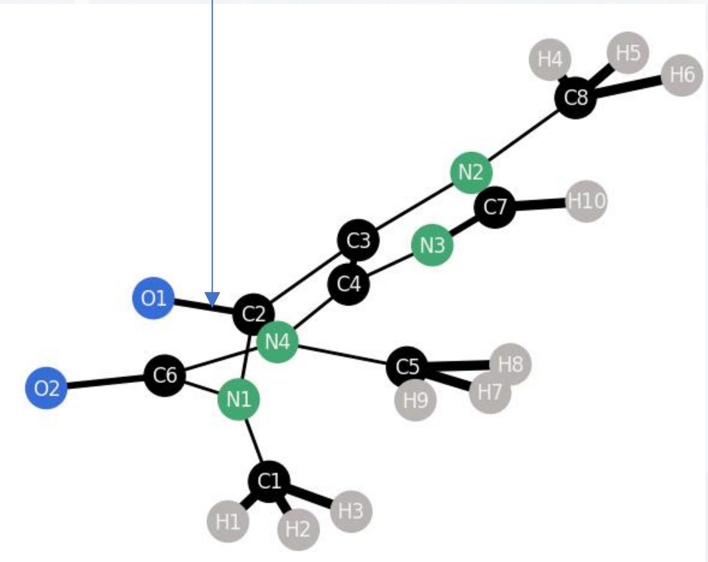


GNN:

What we can learn:

- node classification
- join nodes with similar properties to hyper nodes
- edge attributes, weights (weighted graph)
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule aromatic)

weight: bond strength



information flow from one node to another:
message passing

different ways how:

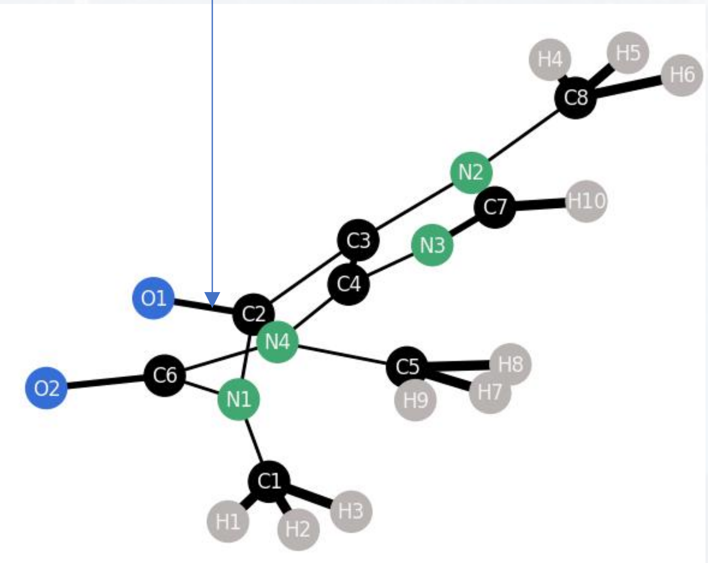
- local averaging
- graph convolution (aka neighborhood aggregation)
- graph attention

GNN:

What we can learn:

- node classification
- join nodes with similar properties to hyper nodes
- edge attributes, weights (weighted graph)
- embedding (eg. 3D structure molecules)
- graph classification (is the molecule aromatic)

weight: bond strength

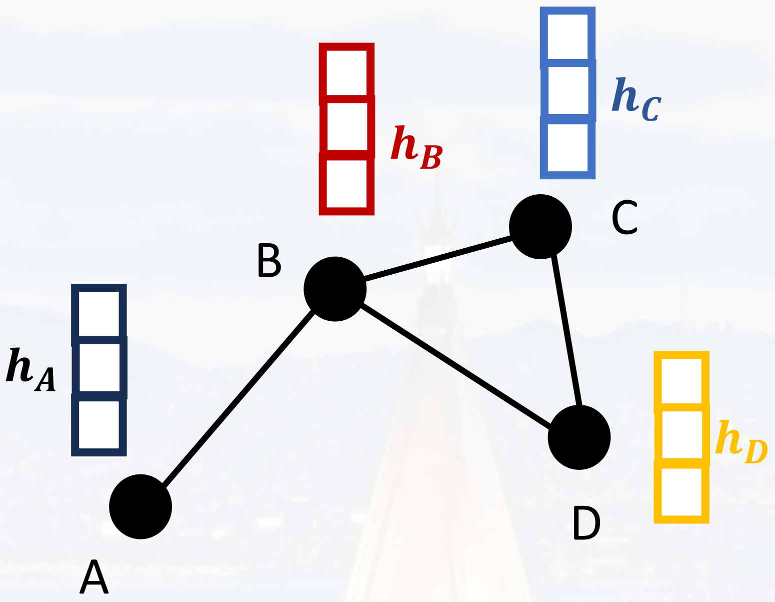


information flow from one node to another:
message passing

different ways how:

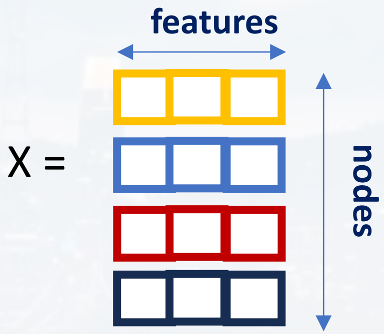
- local averaging
- **graph convolution** (neighborhood aggregation)
- **graph attention**

Graph Convolution

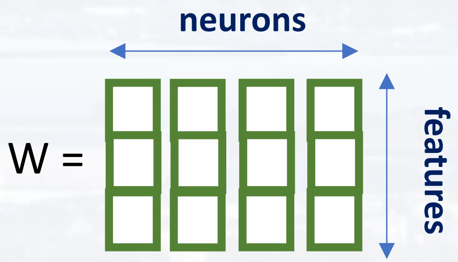


each node i has a **feature vector** h_i

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

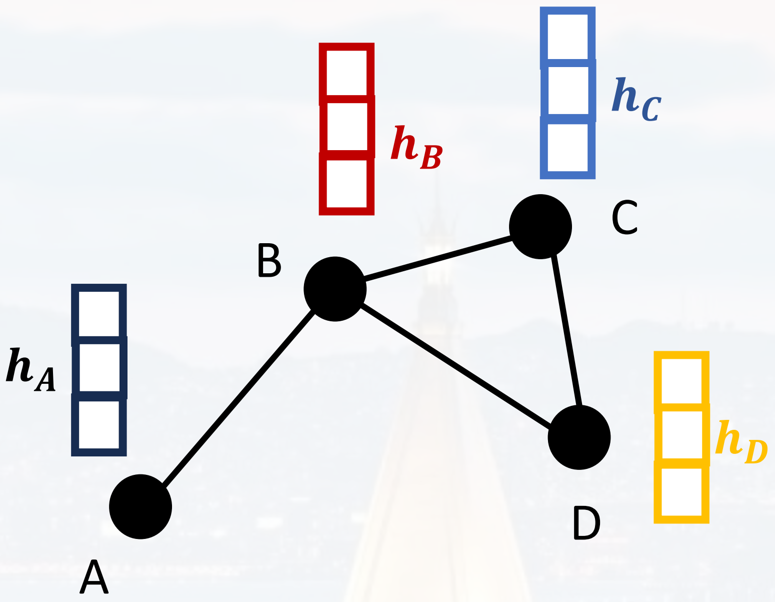


weight matrix W of shape
(number of node features, number of neurons)

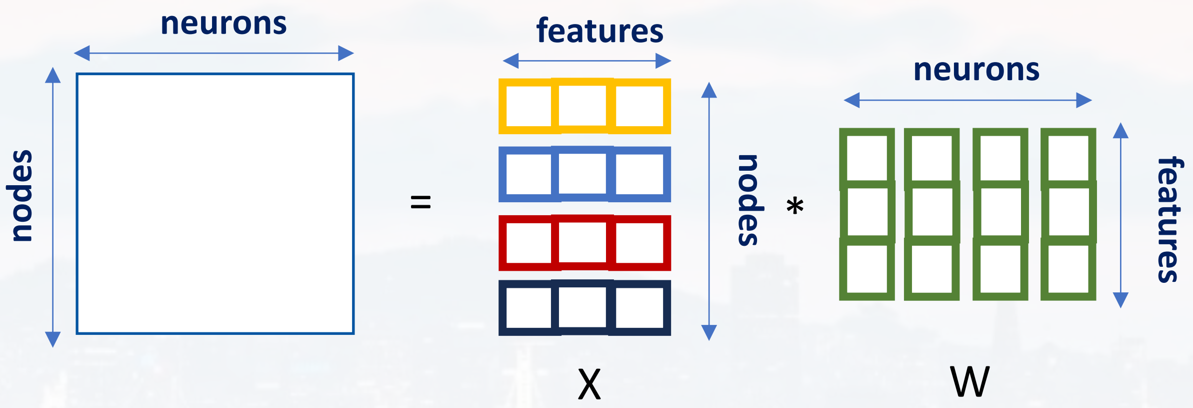


note: only **one** W for the entire graph
W is a **learnable**

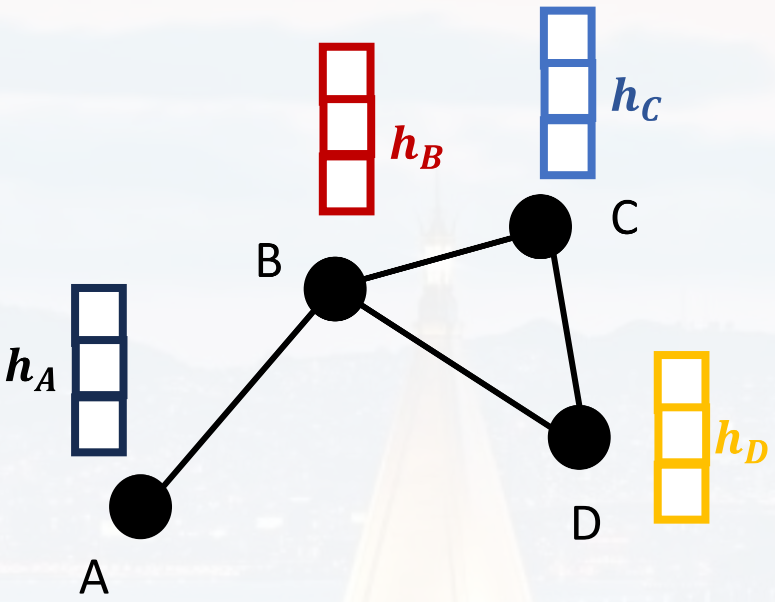
Graph Convolution



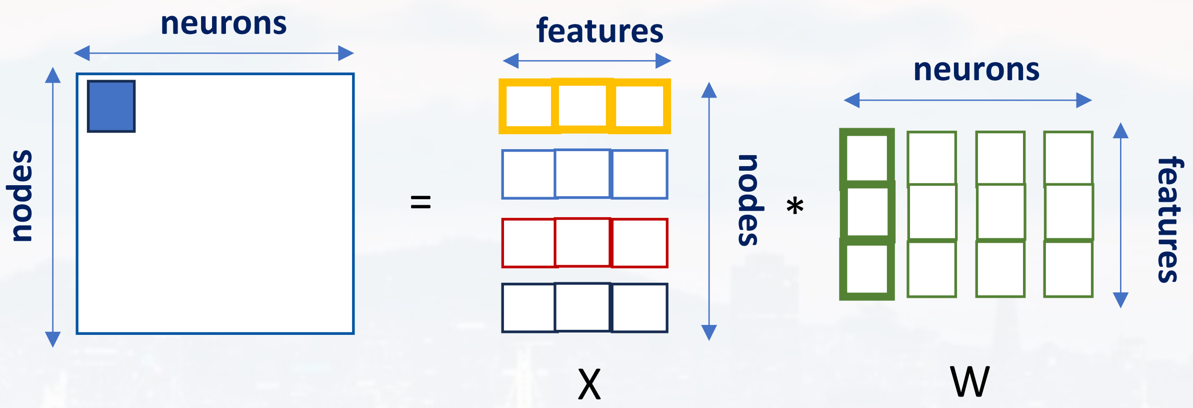
each node i has a **feature vector** h_i



Graph Convolution

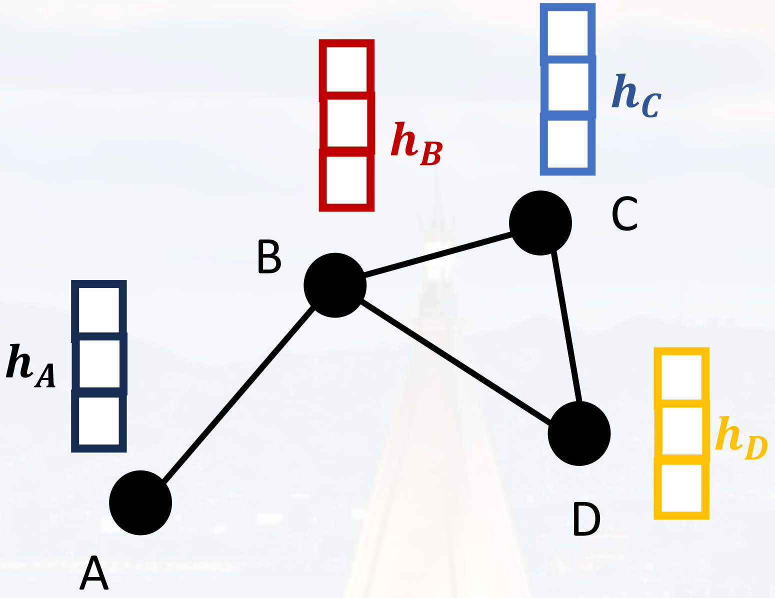


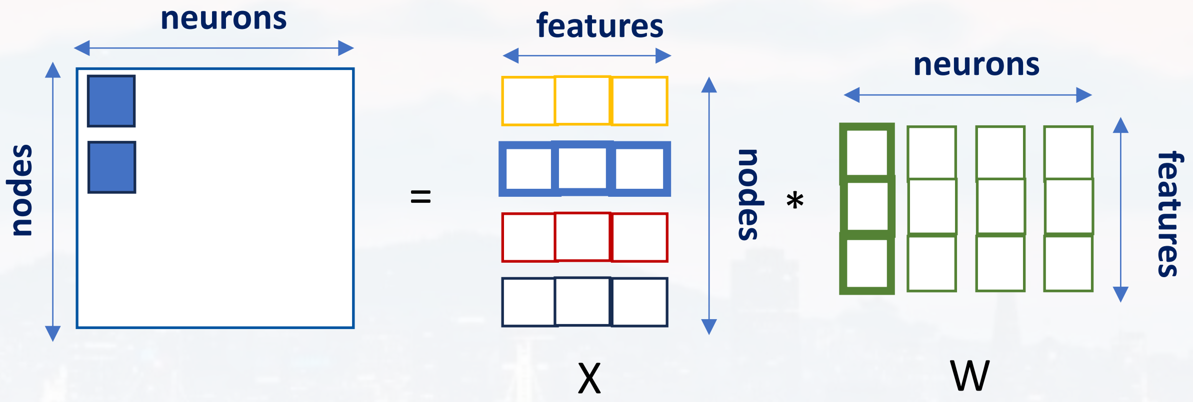
each node i has a **feature vector** h_i



Graph Convolution

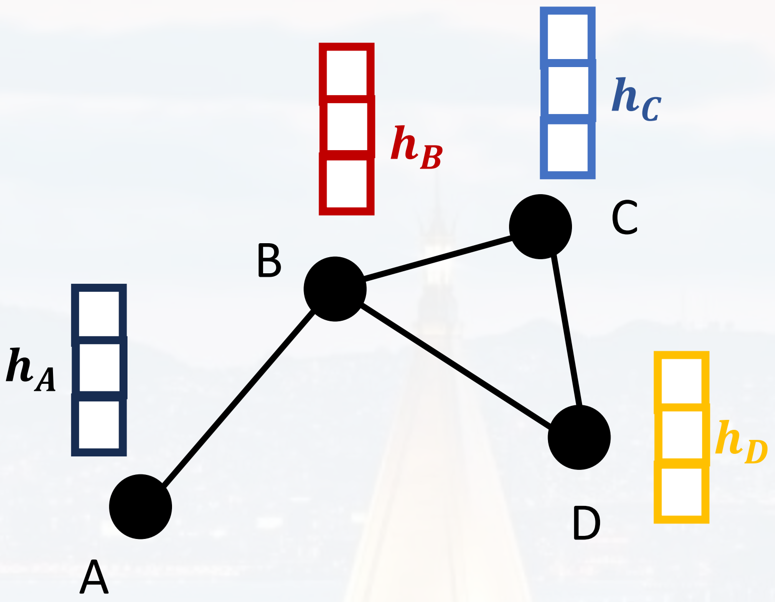
each node *i* has a **feature vector** h_i

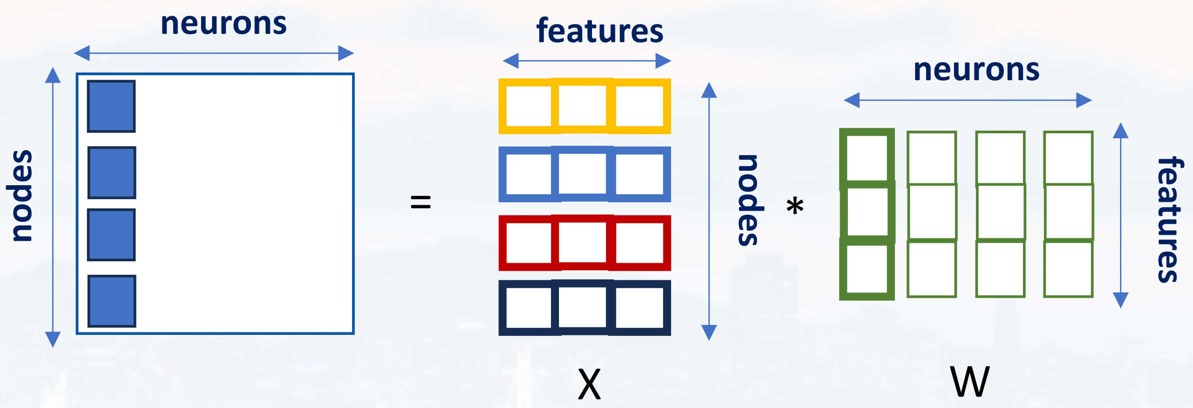




Graph Convolution

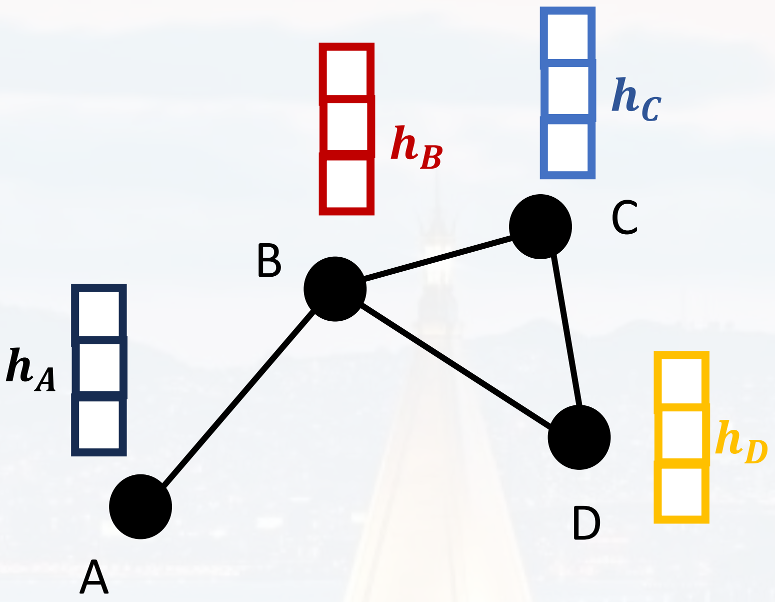
each node i has a **feature vector** h_i

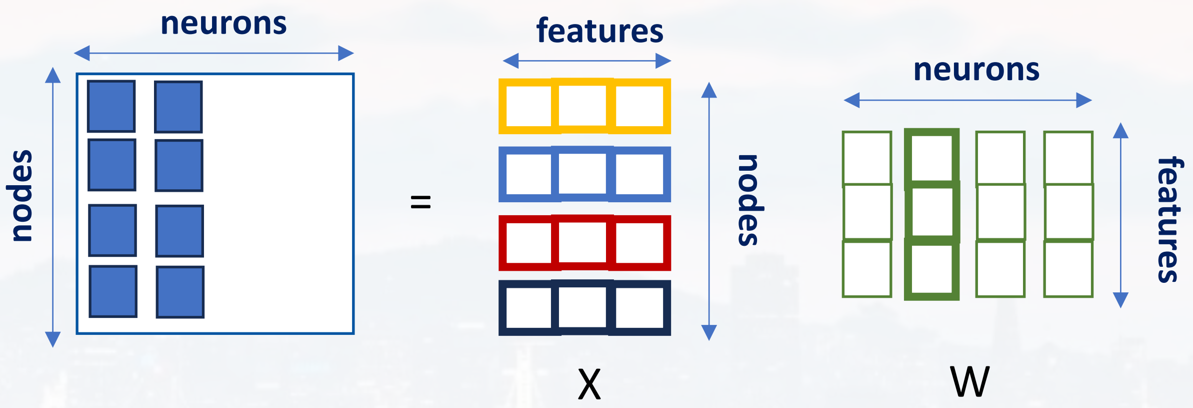




Graph Convolution

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

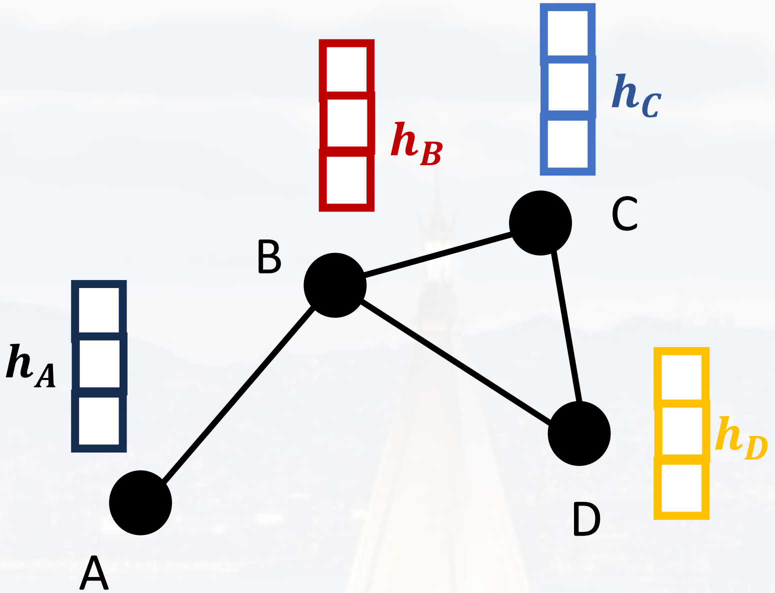


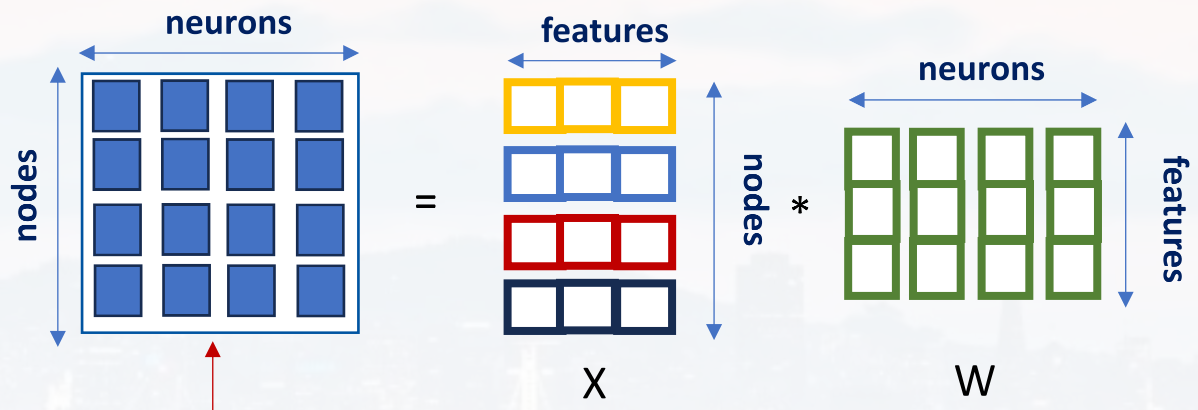


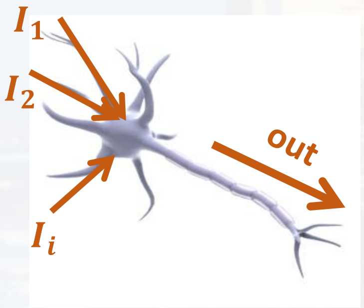
GNN: The ANN Part

Graph Convolution

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



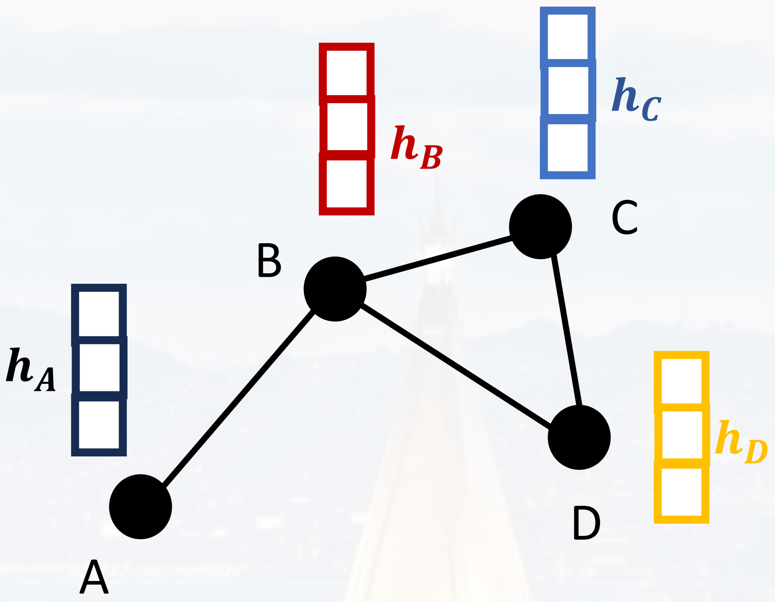




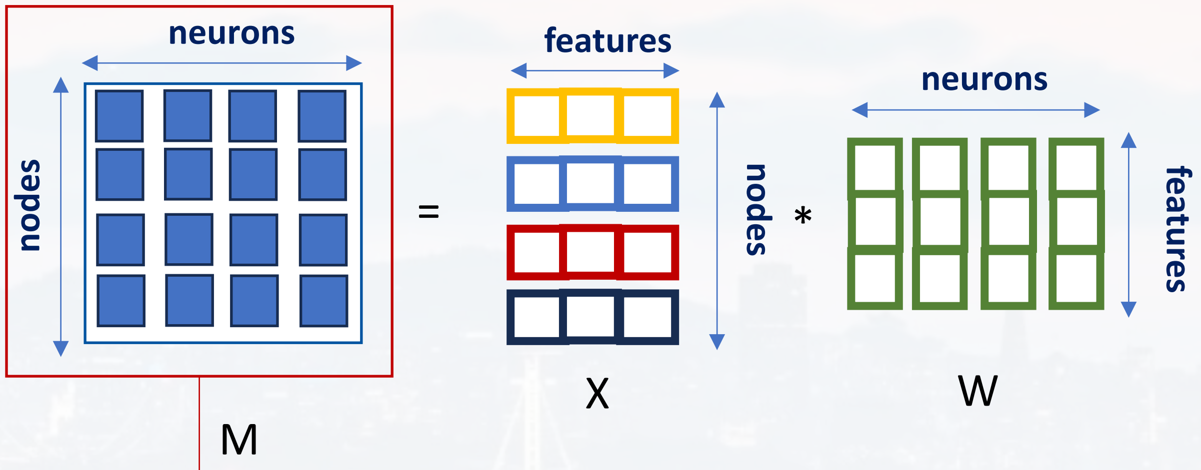
$$net = \sum_i I_i \cdot w_i + b$$

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

Graph Convolution

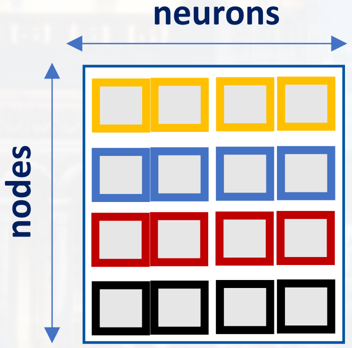


each node i has a **feature vector** h_i



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

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



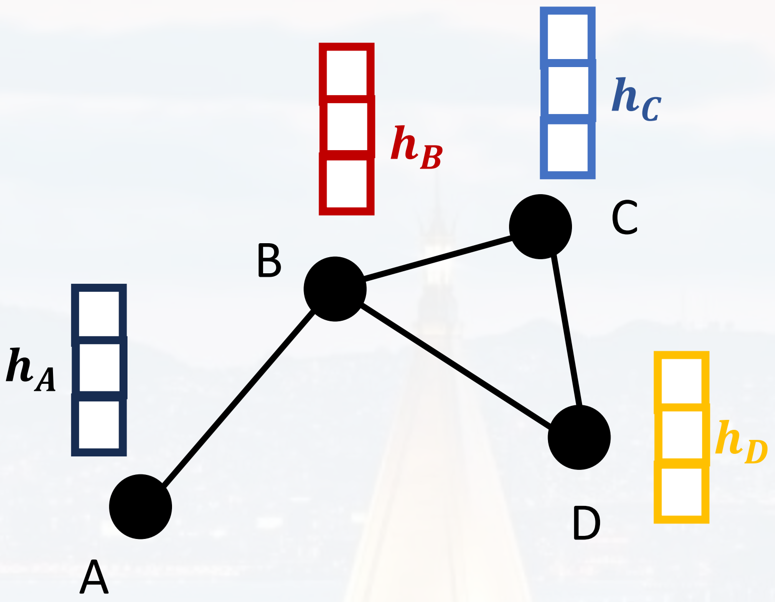
adjacency A

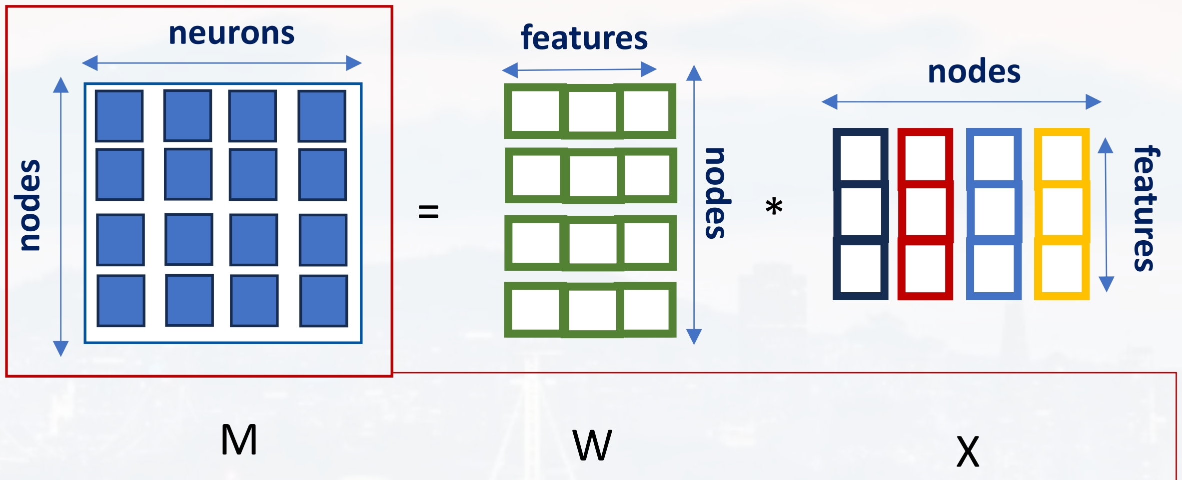
$$= \left[\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right] * \boxed{M}$$

GNN: The ANN Part

Graph Convolution

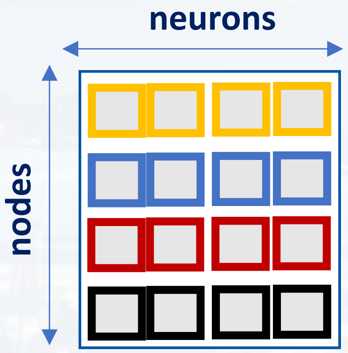
each node i has a **feature vector** h_i





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

(aka second convolution layer)

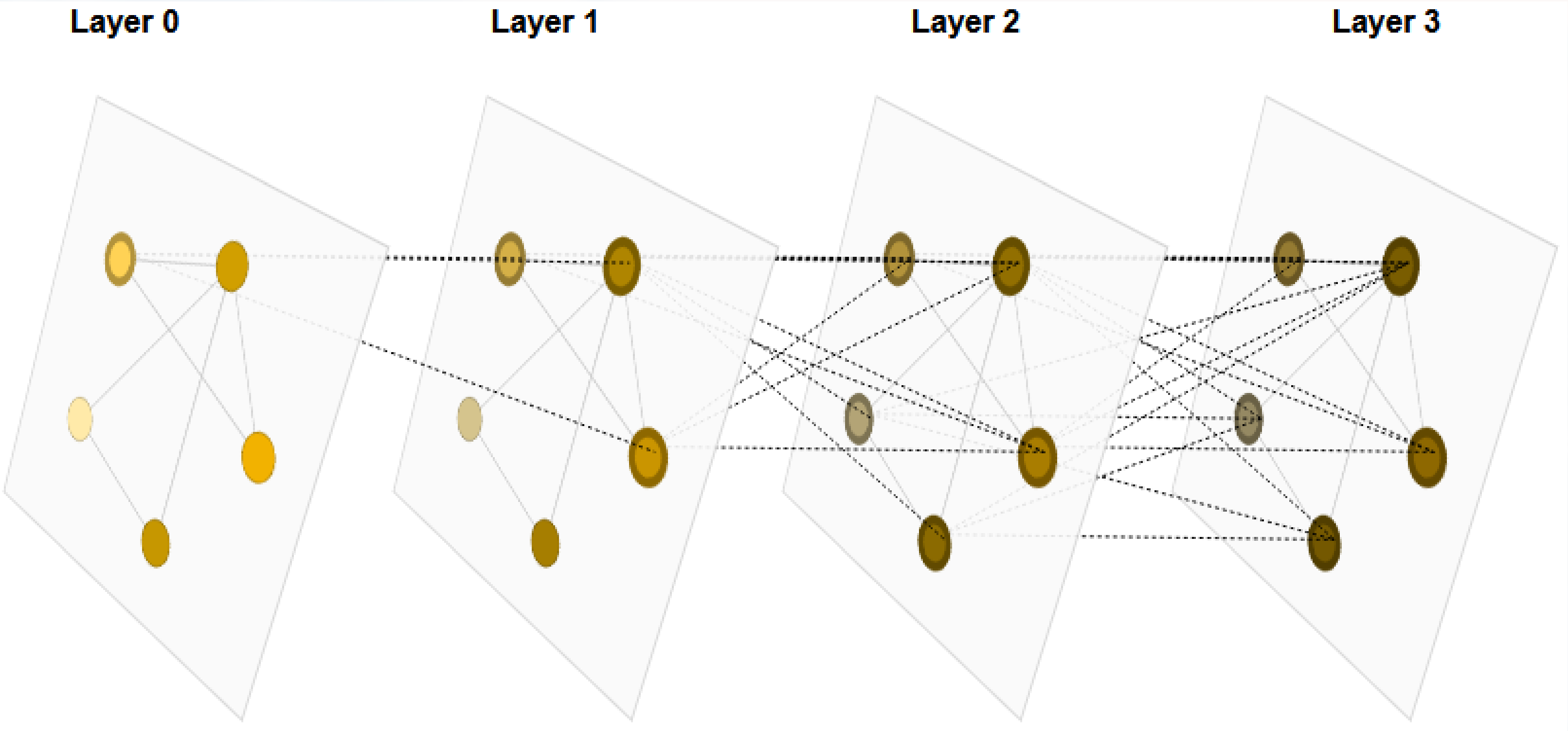


adjacency A

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

GNN: The ANN Part

Graph Convolution

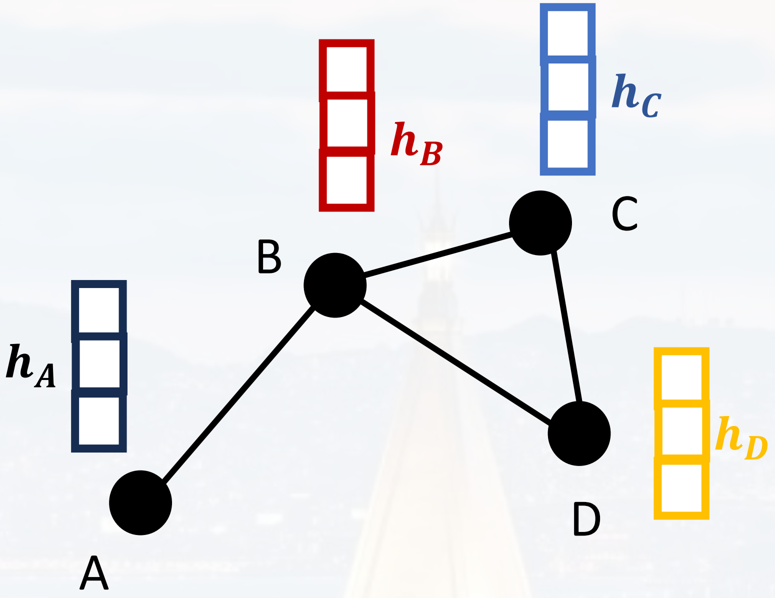


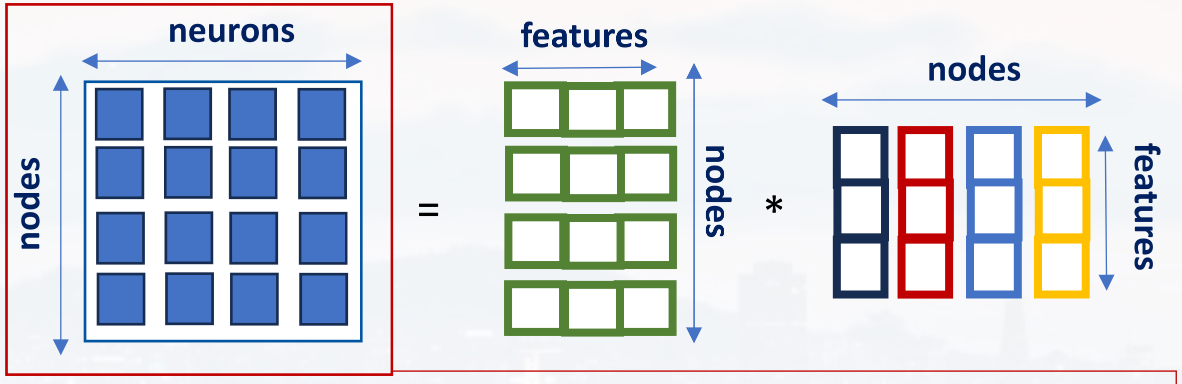
1st layer: one-hop neighborhood
2nd layer: two-hop neighborhood
etc

animation here

Graph Convolution

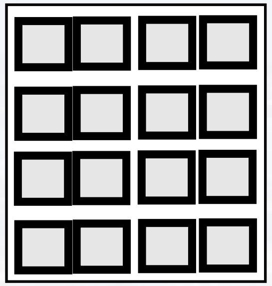
each node *i* has a **feature vector** h_i



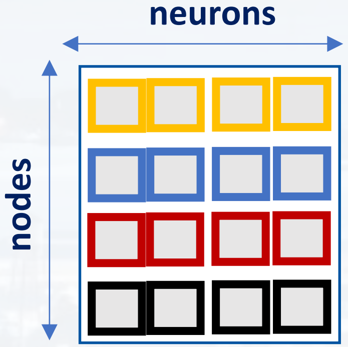


M W X

probabilities



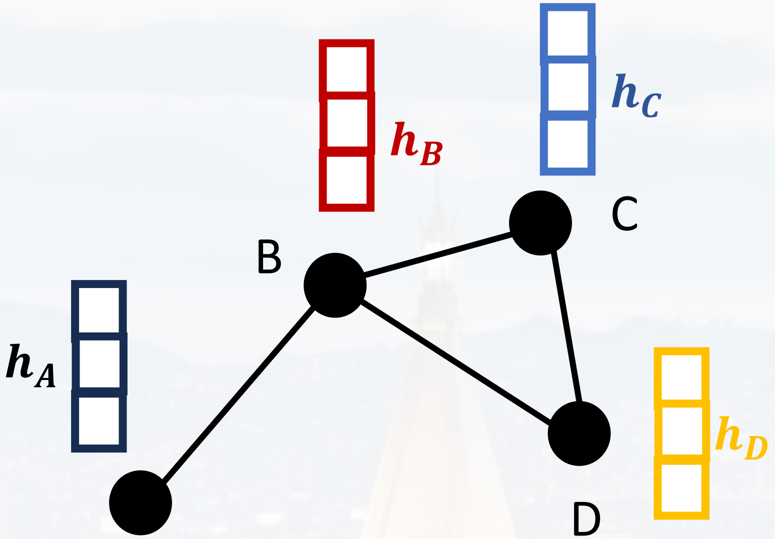
pass through a softmax layer



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



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

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

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

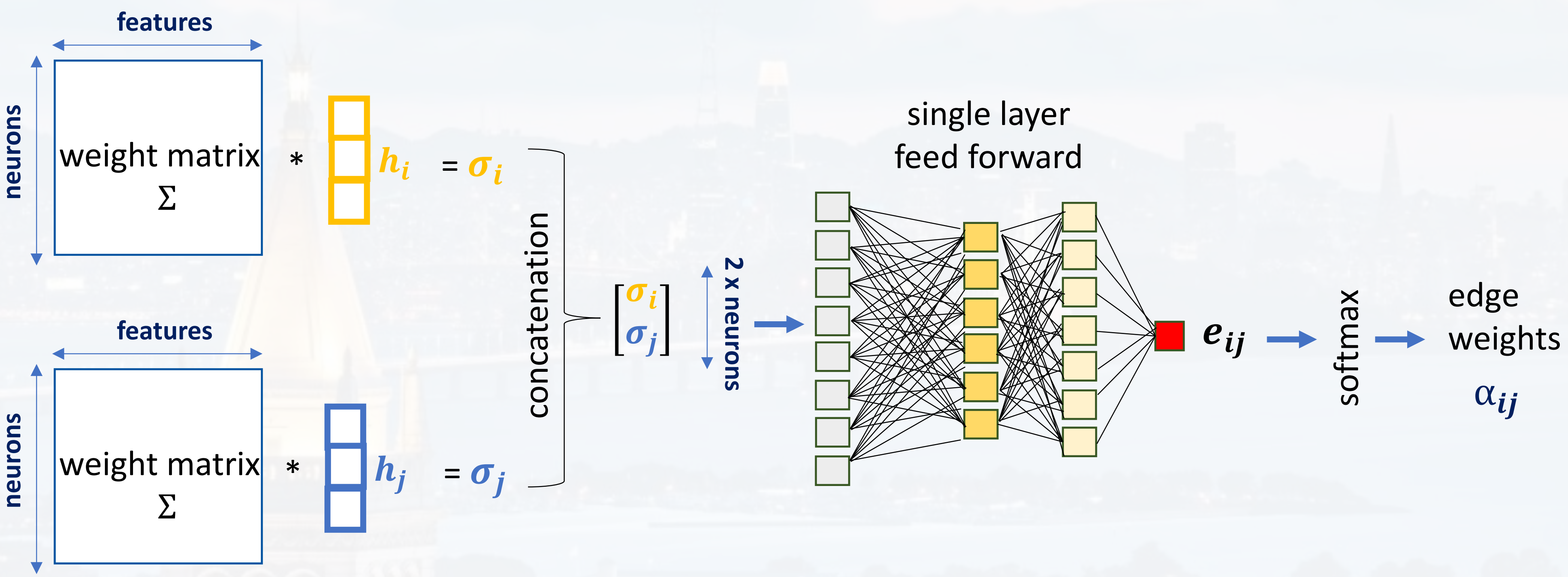
→ normalizing by $\frac{1}{\sqrt{d(n_i)}}$ and $\frac{1}{\sqrt{d(n_j)}}$

more information here


Matthew N. Bernstein

Graph Attention

Learning the weights!
(edge attributes)



Outline

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

node classification: **convolution GNN**

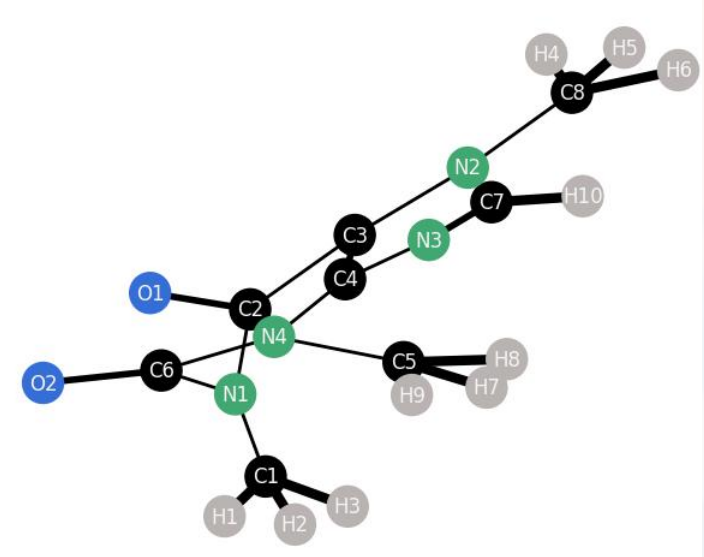
```
self.conv1 = GCNConv(n_node_features, n_neuron)
self.conv2 = GCNConv(n_neuron, n_classes)

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!