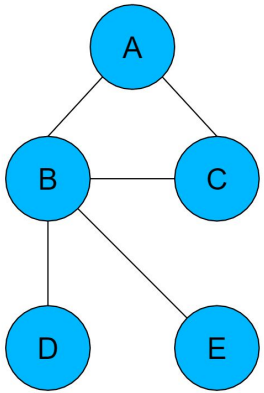
Recitation 9

Graphs and GNNs

Slides adapted from lecture 15 presented by Manolis Kellis, Neil Band, and Maria Brbic

Graphs

- In the most basic setting, graphs are a pair (V,E)
  - $E \subseteq \{ \{x,y\} \mid x,y \in V \}$
    - For directed graphs, $E \subseteq \{ (x,y) \mid x,y \in V \}$



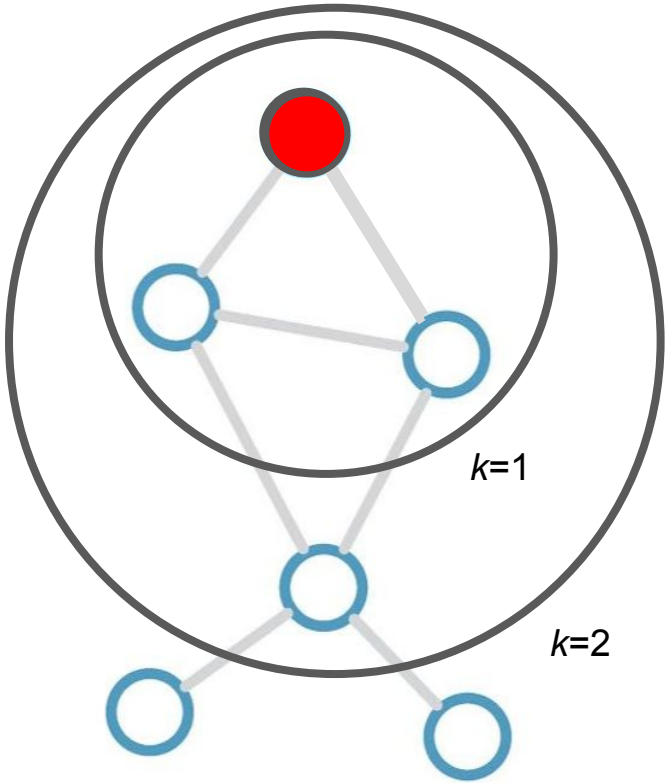
Graphical representation

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	1
C	1	1	0	0	0
D	0	1	0	0	0
E	0	1	0	0	0

Adjacency matrix representation

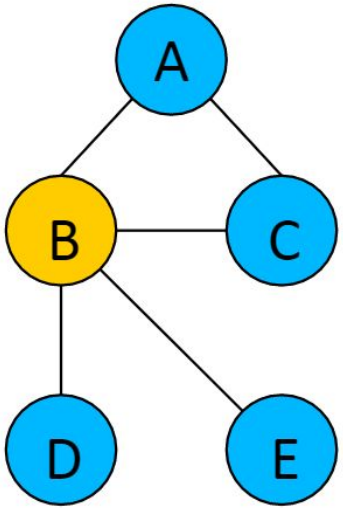
Neighbourhoods

- A **neighbour** of a vertex *v* is any vertex that shares an edge with *v*
- The **neighbourhood** of *v* is the set of neighbours of *v*
- The ***k*-hop neighbourhood** of *v* is the set of vertices that can be reached from *v* by traversing *k* edges
- The **graph distance** between two vertices is the shortest path between them



Importance of a node (centrality)

- The **degree** of a vertex is the number of neighbours it has
  - B has degree 4
- The **betweenness centrality** of a vertex measures how often you travel through it
  - Assume that travel is always along shortest path
  - B has a betweenness centrality of 5
- The **closeness centrality** of a vertex *v* measures how far the average vertex is from *v*
  - B has a closeness centrality of 1

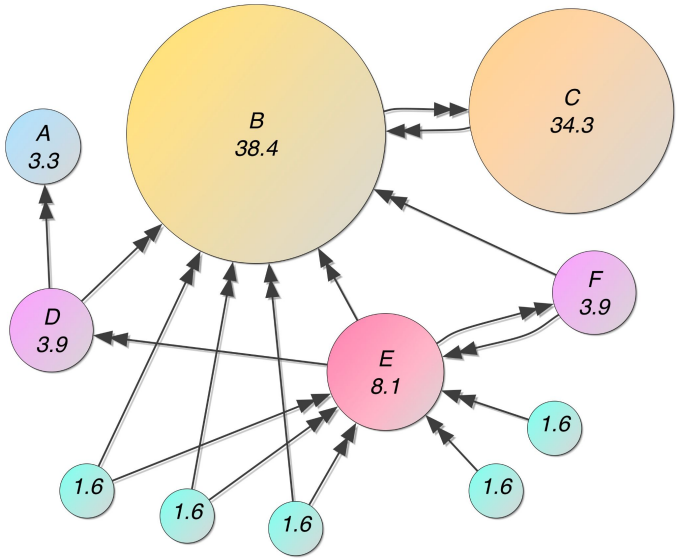


$$BC(k) = \sum_i \sum_j \frac{\rho(i,k,j)}{\rho(i,j)}, \quad i \neq j \neq k$$

$$CC(i) = \frac{N-1}{\sum_j d(i,j)}$$

Random walks

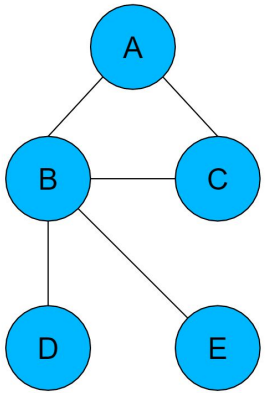
- We generate a random walk by selecting an edge uniformly at random at each step and transitioning to the vertex on the other end
- The average occupancy time for each vertex can be used as a measure of its importance
  - This is the idea behind PageRank



note: The random walk represented here has a random restart allowing escape from A,B, and C.

https://upload.wikimedia.org/wikipedia/en/8/8b/PageRanks-Example.jpg

Random walks



Graph

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

Adjacency Matrix

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/4 & 0 & 1/4 & 1/4 & 1/4 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Random walk/transition matrix

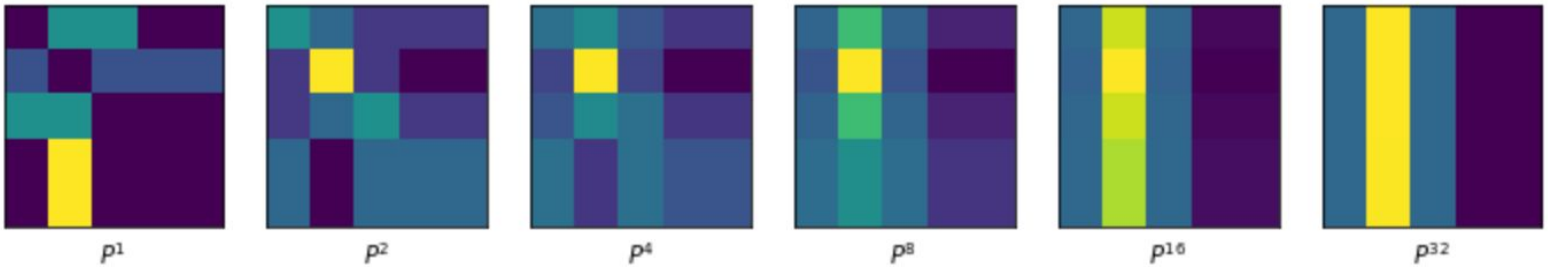
$$\pi_{i+1} = \pi_i P$$

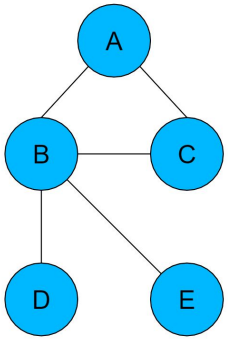
Update distribution for one step

$$\pi_k = \pi_0 P^k$$

Update distribution for k steps

Random walks





$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/4 & 0 & 1/4 & 1/4 & 1/4 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$P^{\infty} = \frac{1}{10} \begin{bmatrix} 2 & 4 & 2 & 1 & 1 \\ 2 & 4 & 2 & 1 & 1 \\ 2 & 4 & 2 & 1 & 1 \\ 2 & 4 & 2 & 1 & 1 \\ 2 & 4 & 2 & 1 & 1 \end{bmatrix}$$

In general, we want to find the eigenvector of P corresponding to λ=1

Graph Laplacian (motivation)

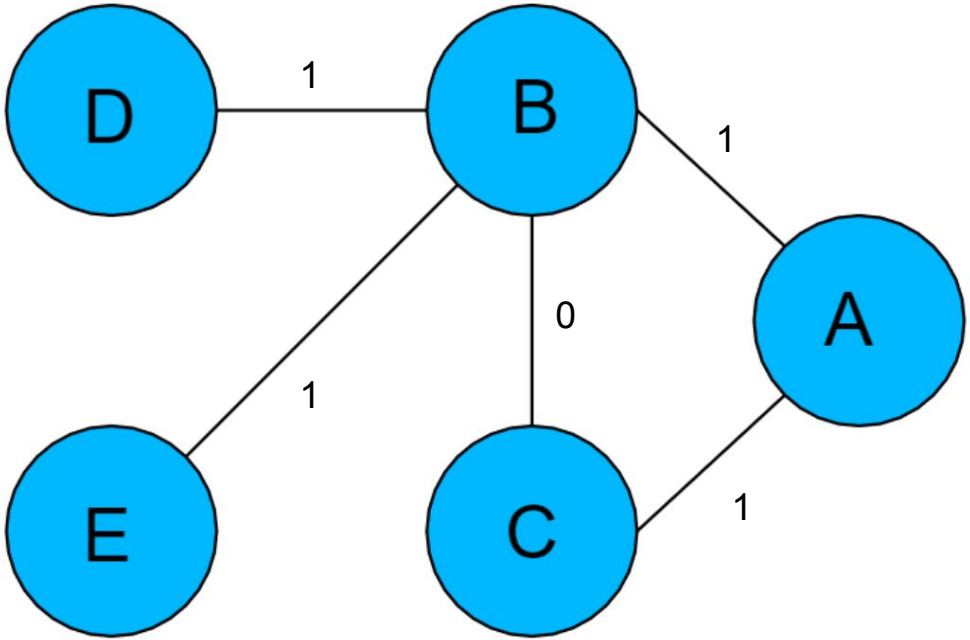
Suppose we have a function f:V→ℝ that maps vertices to numbers

Suppose we want to characterize the “variability” of the function

$$U[f] = \sum_{\{v_1, v_2\} \in E} (f(v_1) - f(v_2))^2$$

Alternatively: U is related to the potential energy in a system connected by springs (these springs are relaxed at length 0)

$$U[f] = \sum_{\{v_1, v_2\} \in E} (f(v_1) - f(v_2))^2$$

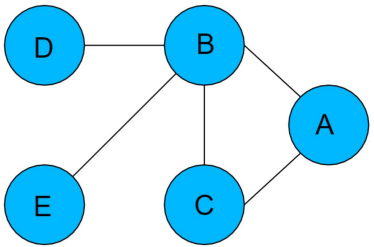


v	f(v)
A	0
B	1
C	1
D	0
E	0

$$U[f] = \sum_{\{v_1,v_2\}\in E} (f(v_1) - f(v_2))^2$$
$$= \sum_{\{v_1,v_2\}\in E} f(v_1)^2 + f(v_2)^2 - 2f(v_1)f(v_2)$$

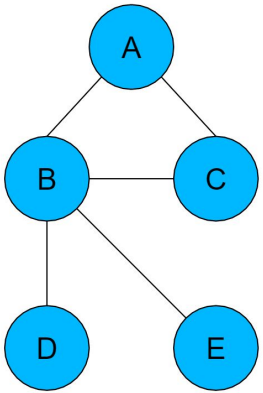
Rearranging the terms in the sum in the shape of a matrix:

$$\begin{bmatrix} deg(A)f(A)^2 & -f(A)f(B) & -f(A)f(C) & 0 & 0 \\ -f(B)f(A) & deg(B)f(B)^2 & -f(B)f(C) & -f(B)f(D) & -f(B)f(E) \\ -f(C)f(A) & -f(C)f(B) & deg(C)f(C)^2 & 0 & 0 \\ 0 & -f(D)f(C) & 0 & deg(D)f(D)^2 & 0 \\ 0 & -f(E)f(C) & 0 & 0 & deg(E)f(E)^2 \end{bmatrix}$$



$$f^T \begin{bmatrix} deg(A) & -1 & -1 & 0 & 0 \\ -1 & deg(B) & -1 & -1 & -1 \\ -1 & -1 & deg(C) & 0 & 0 \\ 0 & -1 & 0 & deg(D) & 0 \\ 0 & -1 & 0 & 0 & deg(E) \end{bmatrix} f \qquad f = \begin{bmatrix} f(A) \\ f(B) \\ f(C) \\ f(D) \\ f(E) \end{bmatrix}$$

Laplacian matrix of a graph



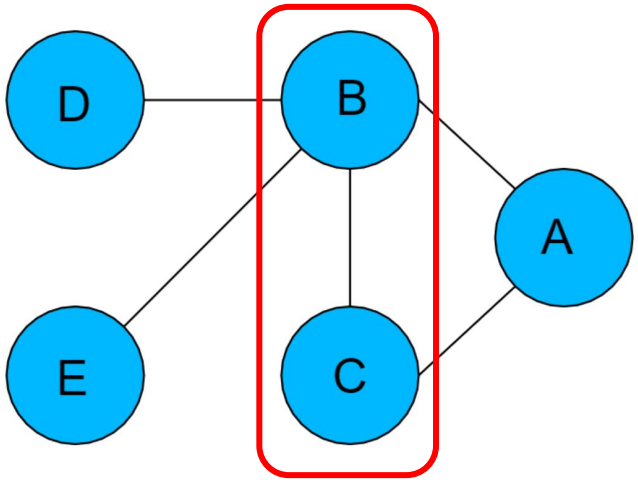
Negative adjacency matrix with the degrees
of the vertices along the diagonal

$$\mathcal{L} = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

$$U[f] = f^T \mathcal{L} f$$

Laplacian matrix has non-negative eigenvalues
Smallest eigenvalue is always 0, and corresponds to the vector of all 1s

The Laplacian captures edge crossings of graph partitions



$$U = \sum_{\{v_1,v_2\}\in E} (f(v_1) - f(v_2))^2$$
$$= f^T \mathcal{L} f$$

v	f(v)
A	0
B	1
C	1
D	0
E	0

Indicator function.

Assume WLOG that the number of vertices that evaluate to 1 is at most the number of vertices that evaluate to 0

v	$f_n(v)$
A	-2/5
B	3/5
C	3/5
D	-2/5
E	-2/5

Normalizing doesn't change U

v	1(v)
A	1
B	1
C	1
D	1
E	1

$f_n \perp 1$

$$\begin{aligned}
\text{edge crossings} &= f_n^T \mathcal{L} f_n \\
&= \sum_i \sum_j u_j^T \mathcal{L} u_i \\
&= \sum_i \sum_j u_j^T u_i \lambda_j \\
&= \sum_i \|u_i\|^2 \lambda_j \\
&\geq \sum_i \|u_i\|^2 \lambda_2 \\
&= \|f_n\|^2 \lambda_2
\end{aligned}$$

λ_2 is the smallest non-zero eigenvector. Remember that f_n has mean 0 so it has no component in the 1 direction.

$$\begin{aligned}
\|f_n\|^2 &= |V|\text{Var}[X] \\
&= |V|p(1-p) \\
&\geq \frac{1}{2}|V|p \\
&= \frac{1}{2}\text{size of partition}
\end{aligned}$$

X is a random variable that denotes whether a randomly selected vertex belongs to the partition. X is true with probability p.

$$\text{edge crossings} \geq \frac{\lambda_2}{2}\text{size of partition}$$

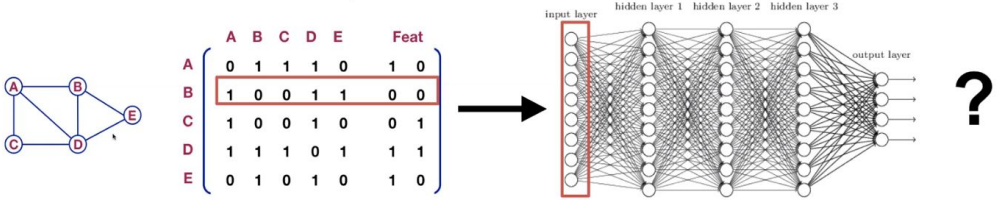
$$2\frac{\text{edge crossings}}{\text{size of partition}} \geq \lambda_2$$

Deep learning on graphs

How do you input a graph to a deep learning network? Representation?

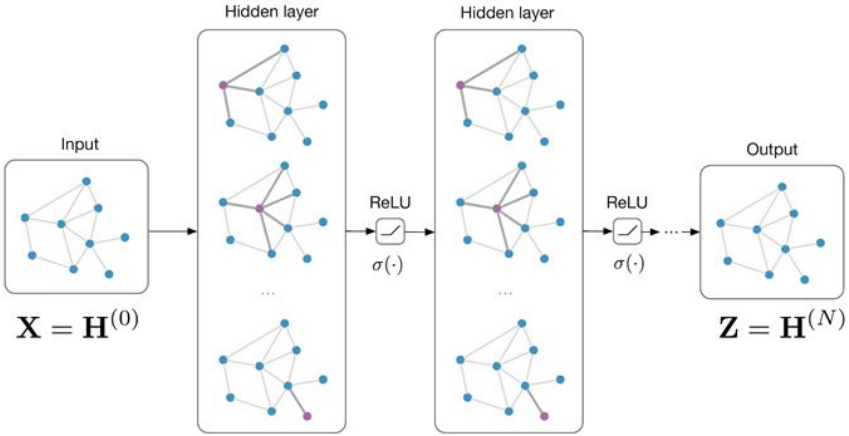
A Naïve Approach

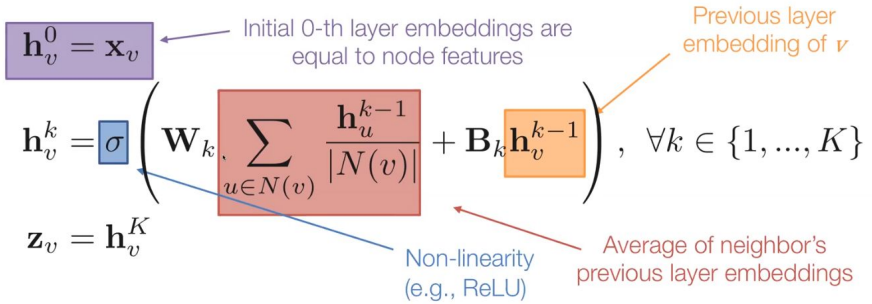
- Join adjacency matrix and features
- Feed them into a deep neural net:



- Issues with this idea:
  - $O(|V|)$ parameters
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

Graph neural networks





Aggregation should respect symmetry of neighbours

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Variants of Aggregation

Mean: Take a weighted average of neighbors

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

Pool: Transform neighbor vectors and apply symmetric vector function

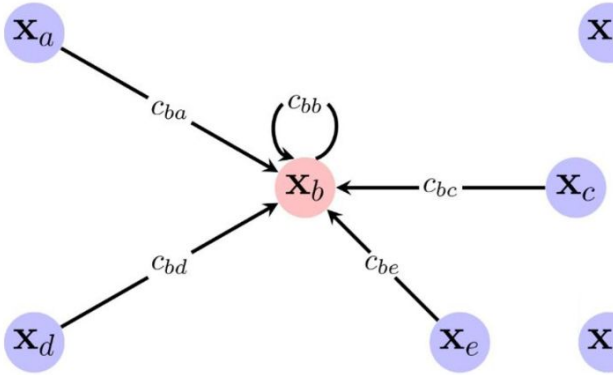
element-wise mean/max

$$\text{AGG} = \boxed{\gamma}\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

LSTM: Apply LSTM to reshuffled of neighbors

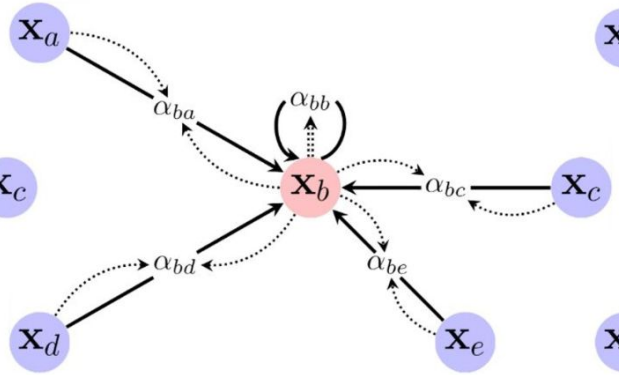
$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

Variations on aggregation



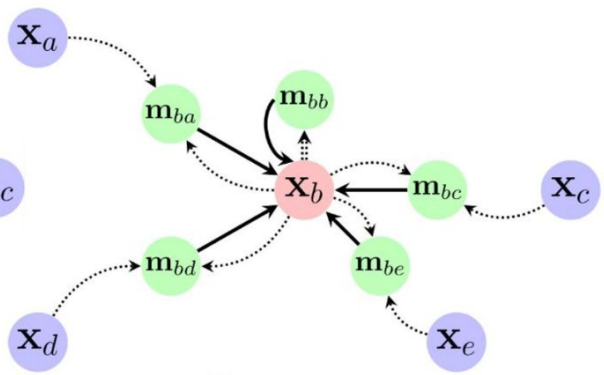
Convolutional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$



Attentional

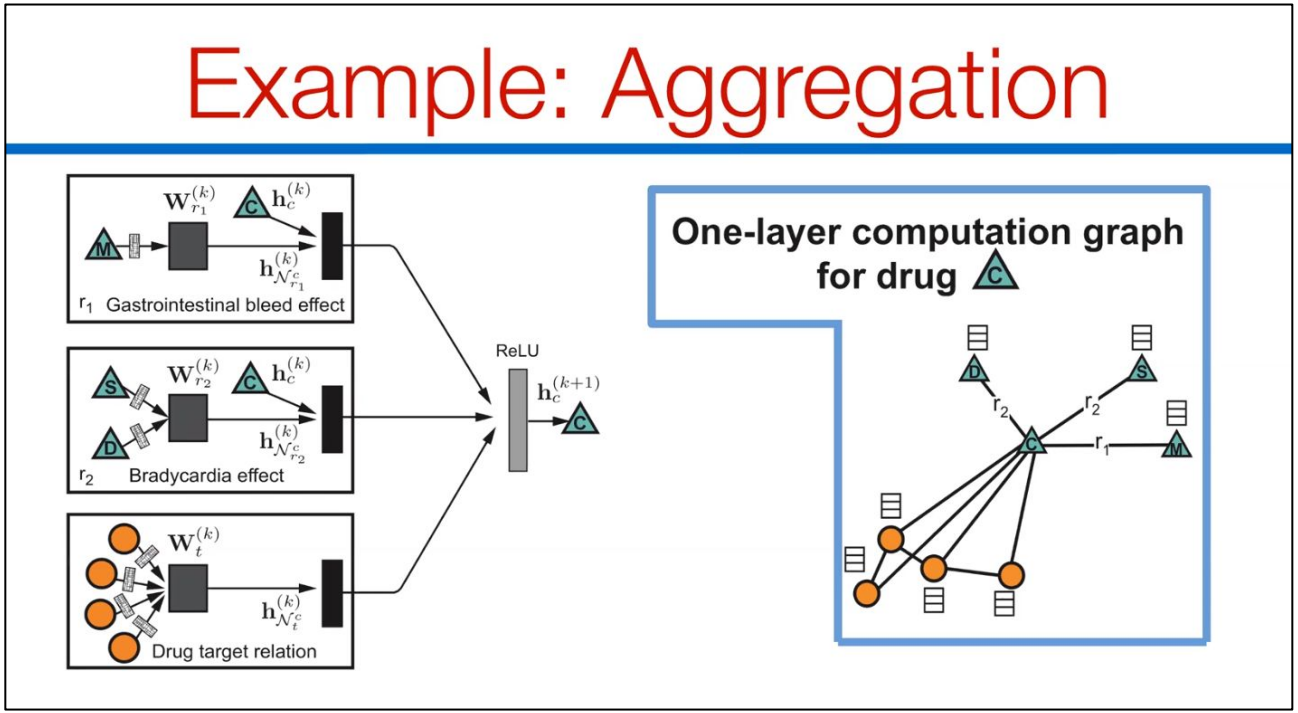
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$



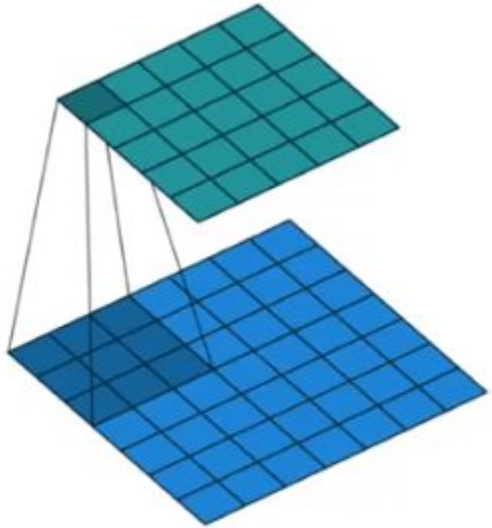
Message-passing

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

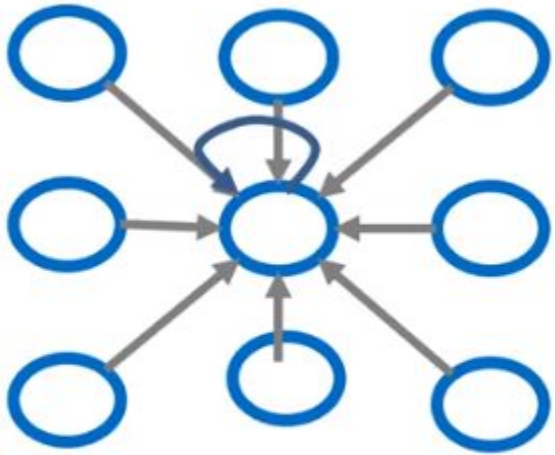
Heterogeneous aggregation



Relation to convolutional layers



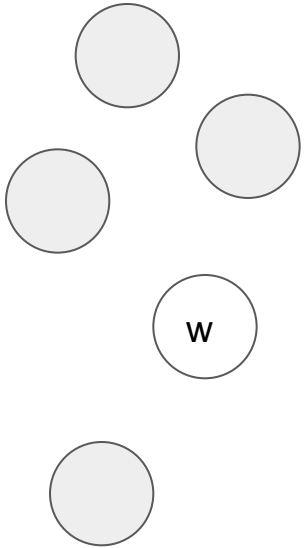
Image



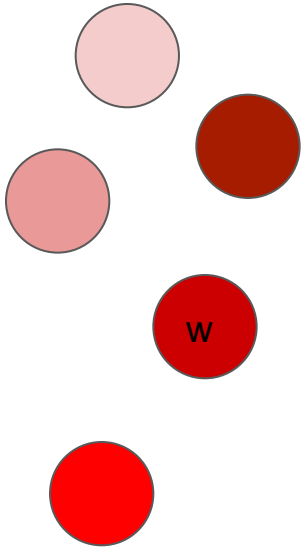
Graph

Refresher on attention

Q(),K(),V() are trainable

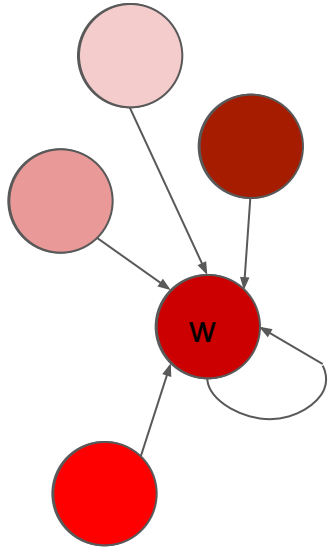


Bag of embeddings



Calculate query-key similarity relative to **w** for all embeddings **e**
(Q(w) · K(e))

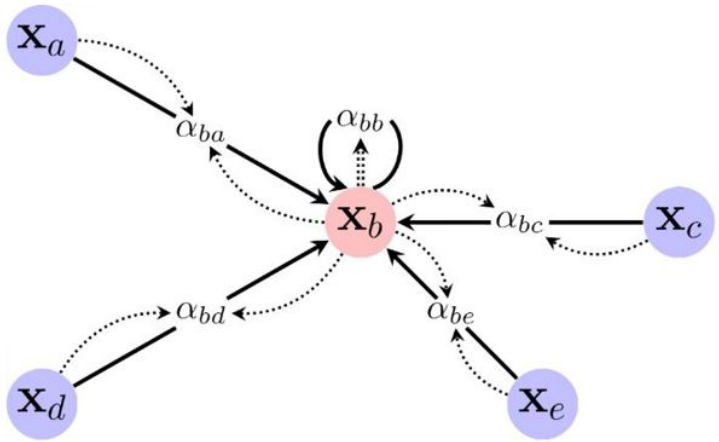
softmax over query-key similarities to get the attention weights a(w,e)

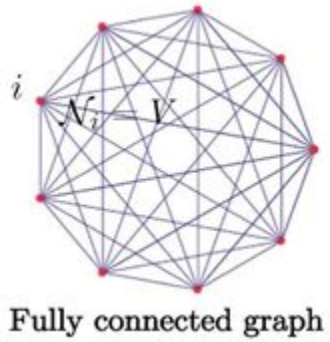


Aggregate over all embeddings e to get the new embedding for **w**
Σ a(w,e) * V(e)

Relation to attention layers

- This is essentially a graph neural network over a fully connected network
  - Remark: Attention layers don't leverage sequence structure architecturally speaking. In fact, sequence structure has to be encoded and explicitly fed into the attention layer

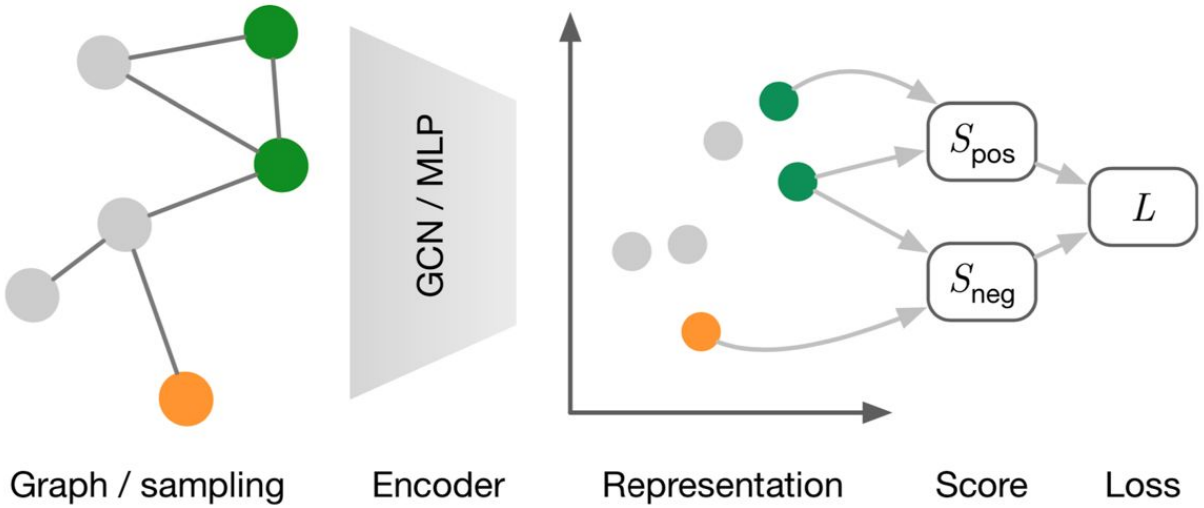




Fully connected graph

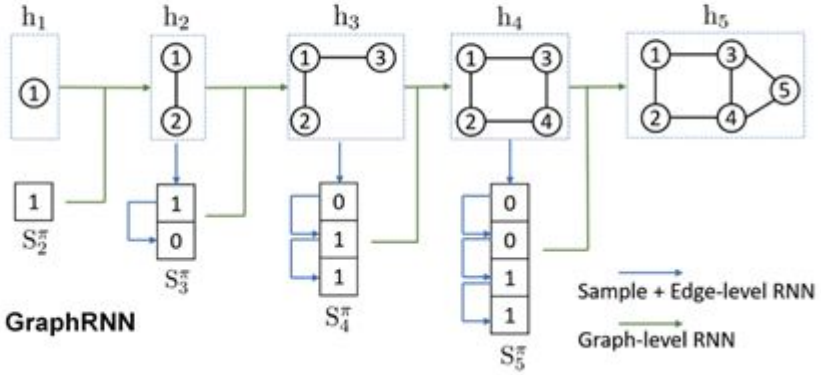
Unsupervised learning

Learning node embeddings for downstream tasks

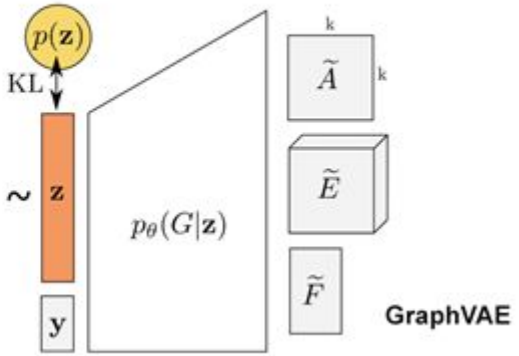


Graph generation

Sequentially:

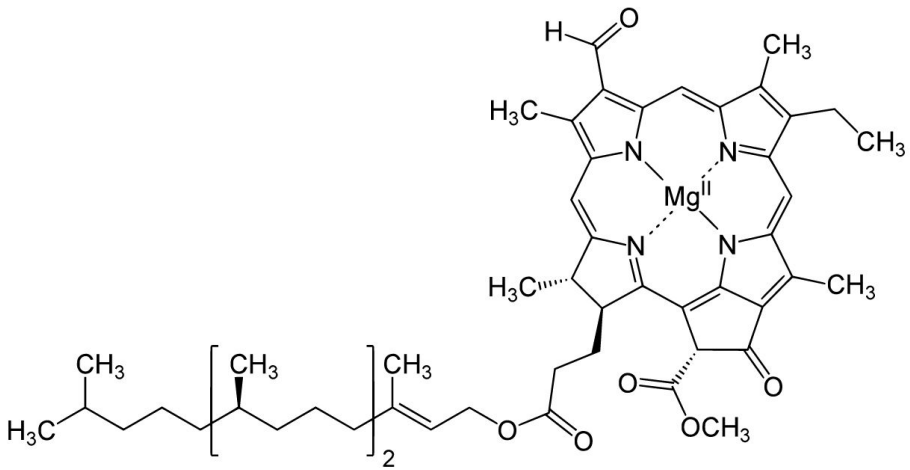


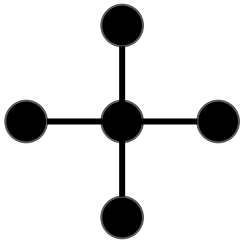
Or in a single step:



A problem (for fun)

Can a GNN identify interesting network structures?





https://upload.wikimedia.org/wikipedia/commons/3/3a/Chlorophyll_d_structure.svg

The 3-SAT problem

Given a formula in conjunctive normal form:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

Figure out an assignment

$$x_1, x_2, x_3 \in \{T, F\}$$

Such that the formula evaluates to True

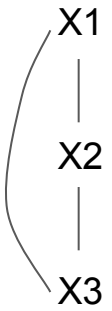
Conjecture: **Any** algorithm solving 3-SAT runs in $\omega(n^k)$ for any k

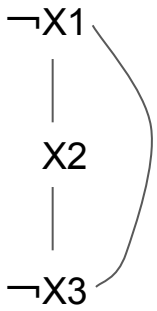
$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

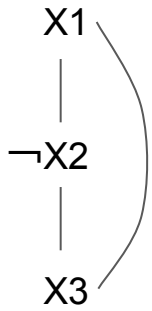
X1	¬X1	X1
X2	X2	¬X2
X3	¬X3	X3

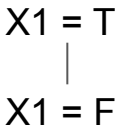
X1 = T	X2 = T	X3 = T
X1 = F	X2 = F	X3 = F

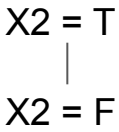
$$x_1, x_2, x_3 \in \{T, F\}$$

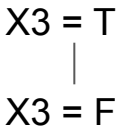

X1
X2
X3

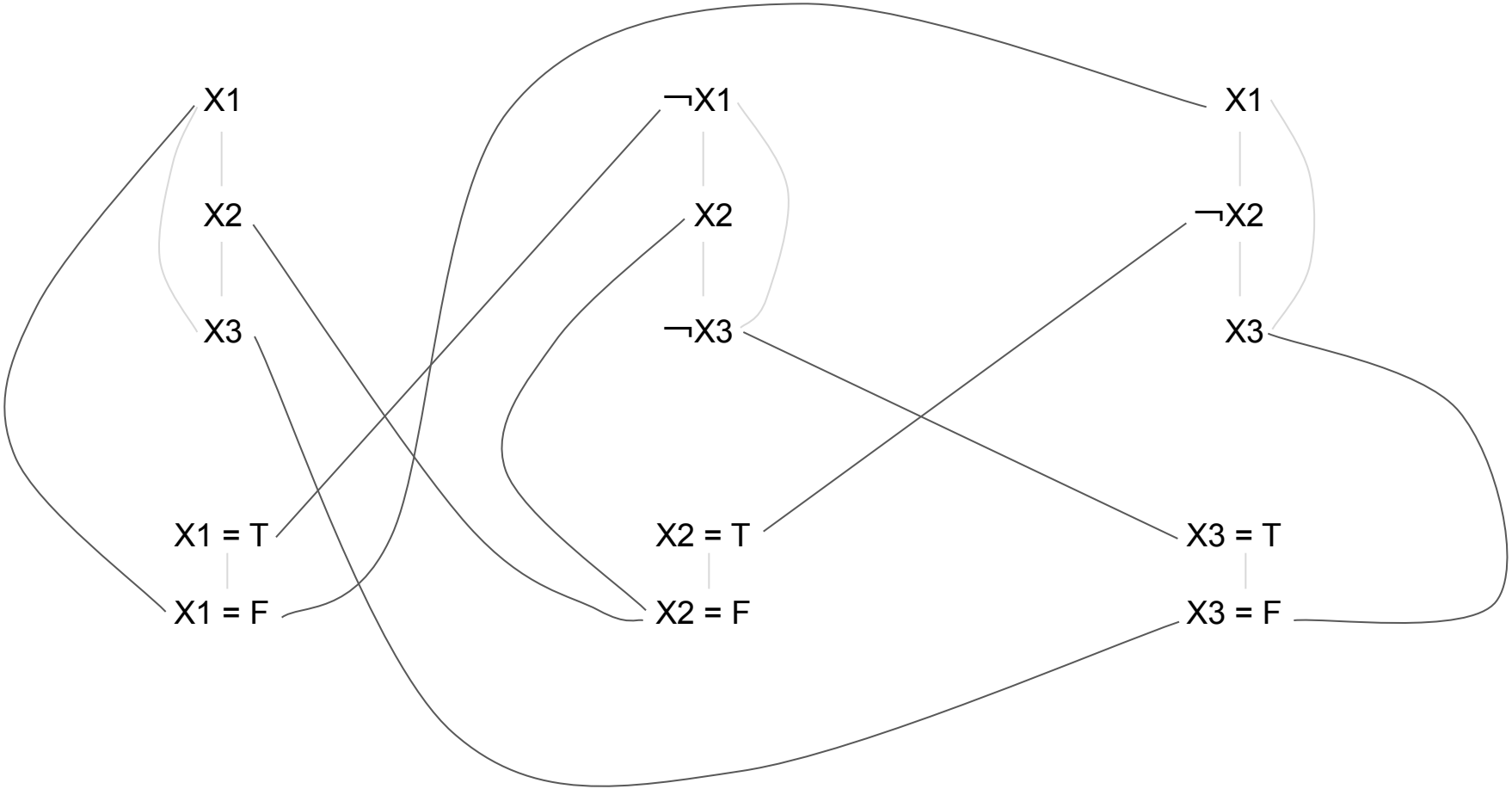

¬X1
X2
¬X3


X1
¬X2
X3

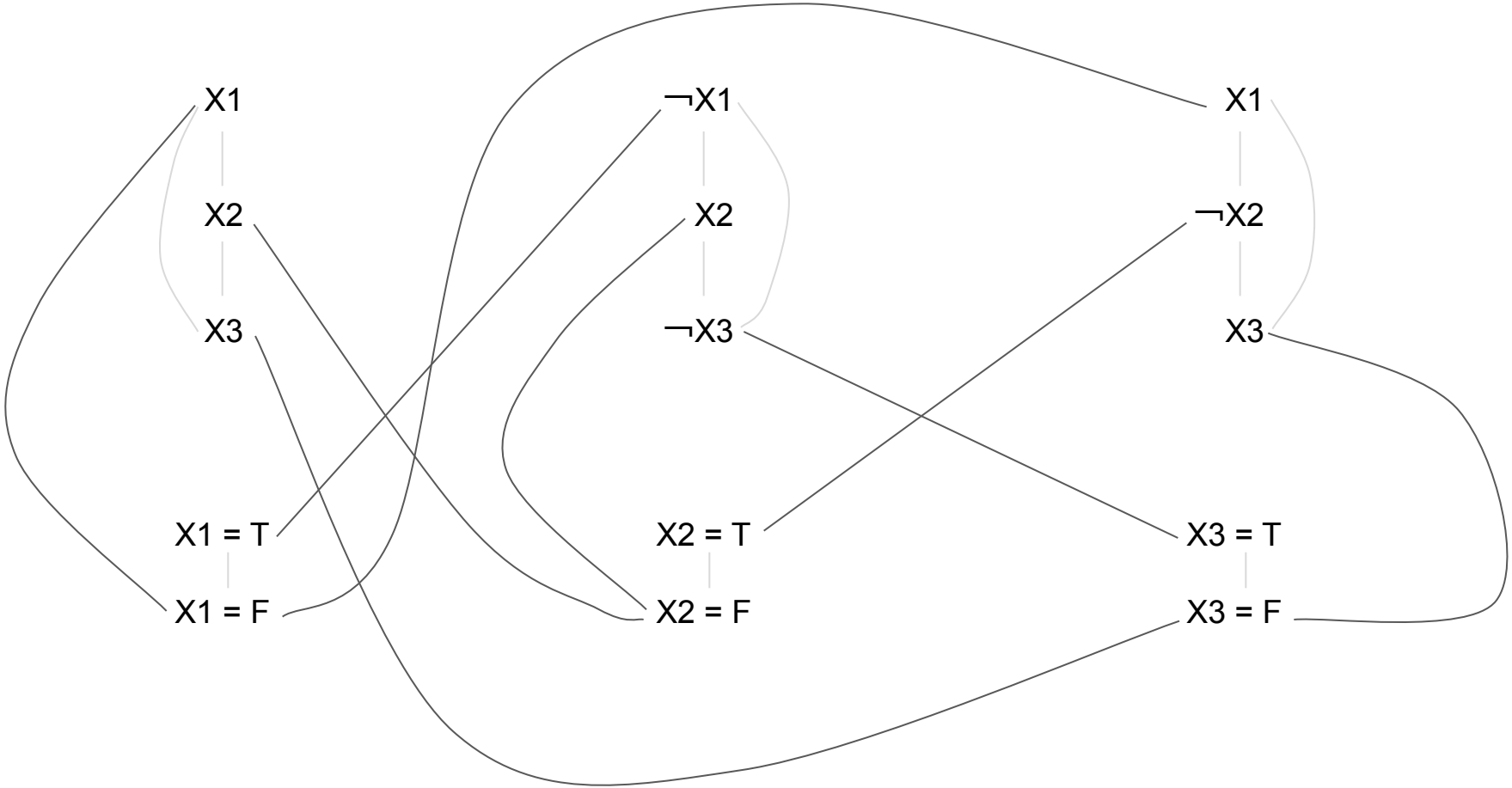

X1 = T
X1 = F

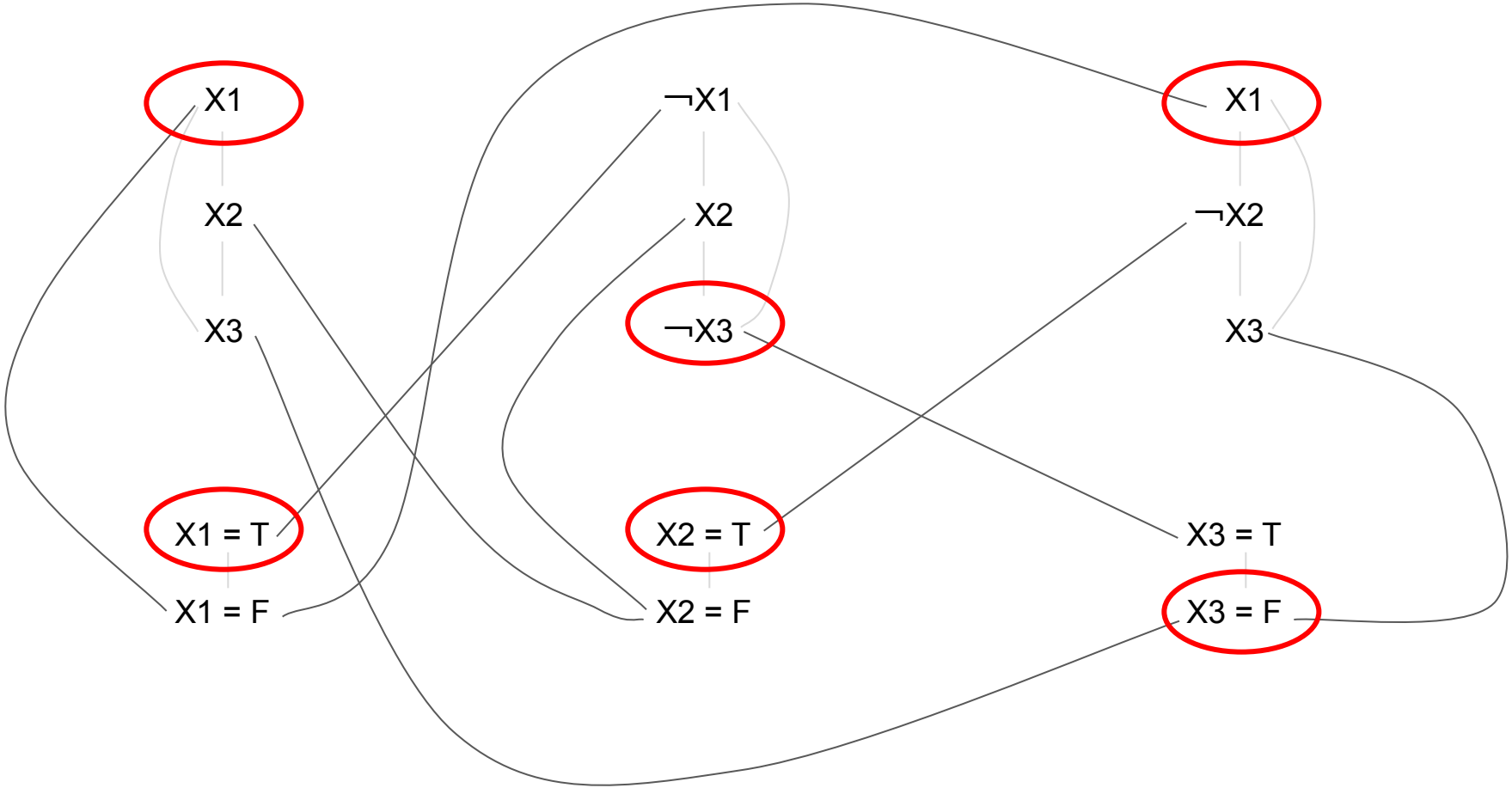

X2 = T
X2 = F


X3 = T
X3 = F

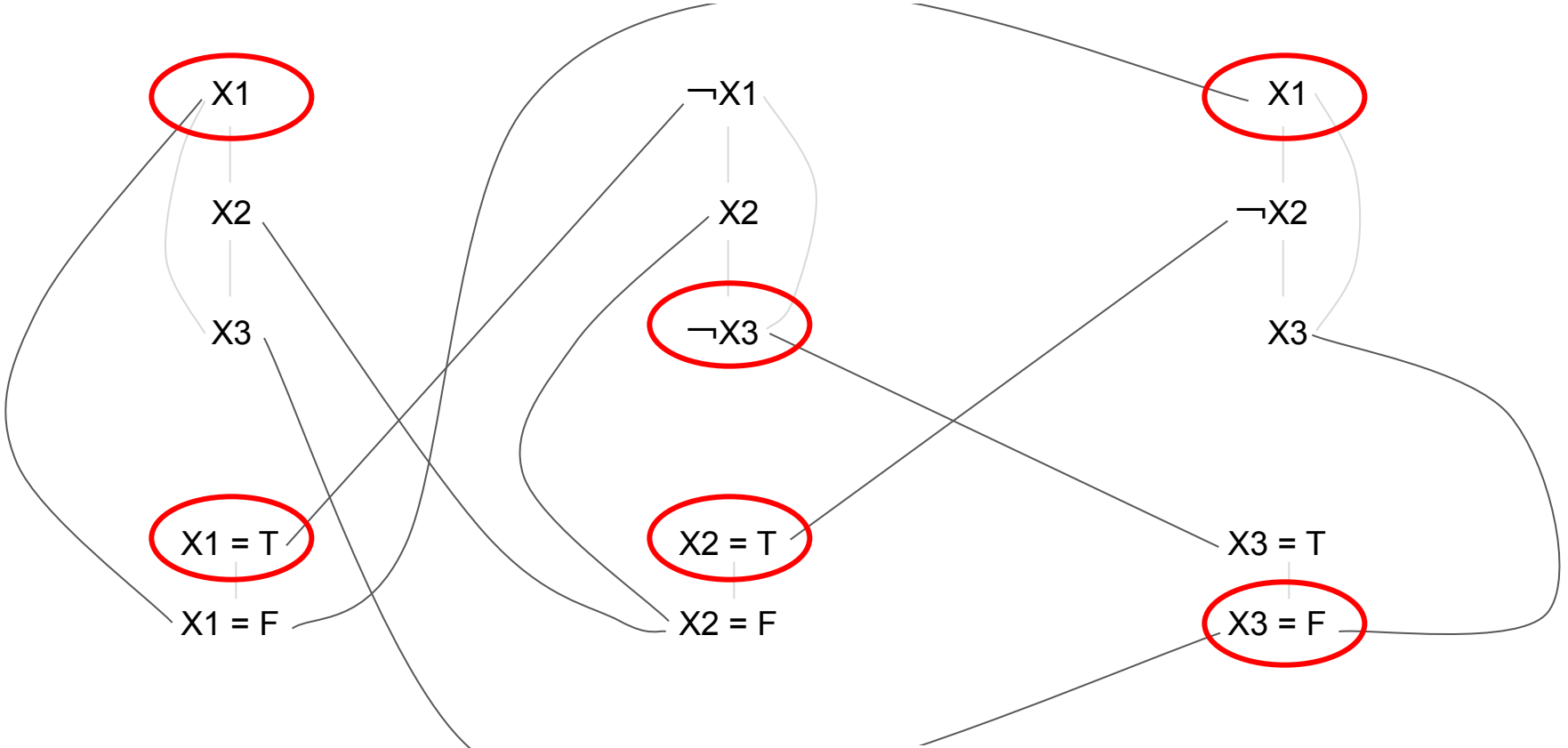

X1
X2
X3
X1 = T
X1 = F
¬X1
X2
¬X3
X2 = T
X2 = F
X1
¬X2
X3
X3 = T
X3 = F

Structure to find: 6 vertices that don’t share any edges



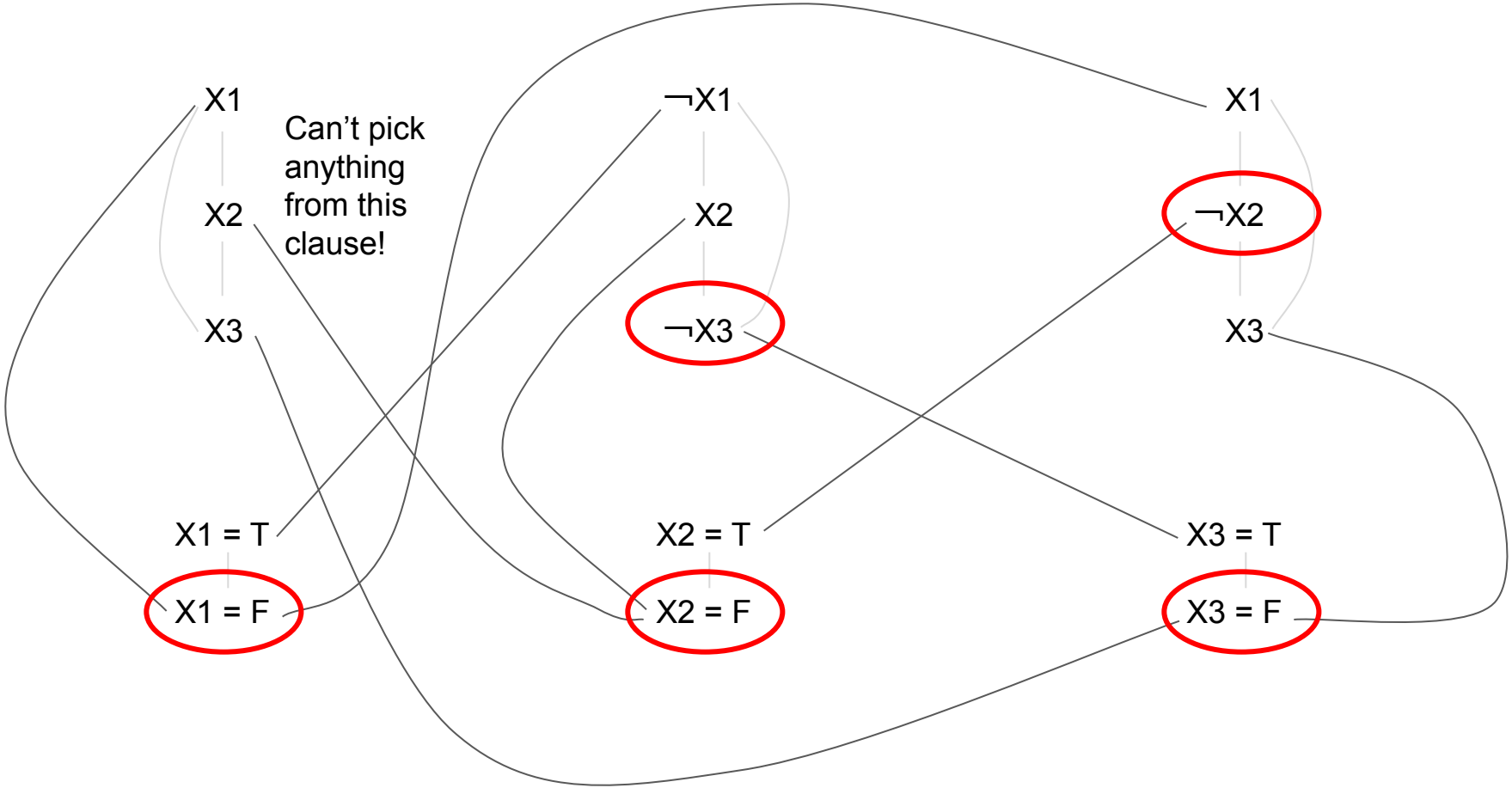

X1
X2
X3
X1 = T
X1 = F
¬X1
X2
¬X3
X2 = T
X2 = F
X1
¬X2
X3
X3 = T
X3 = F

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$



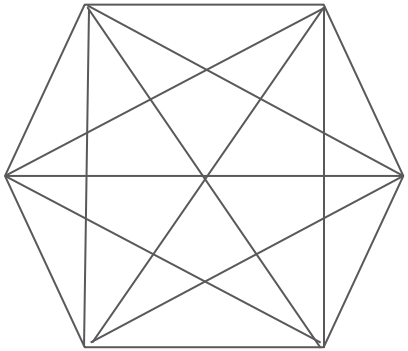
$$x_1 = T, x_2 = T, x_3 = F$$

Any solution will always correspond to a solution of the original 3SAT. Otherwise...



Minor detail

- “6 vertices that don’t share an edge” is not really a “structure”
- You can always flip the graph:
  - Remove all edges that exist
  - Put an edge wherever an edge doesn’t exist
- Find “6 vertices that don’t share an edge” now becomes find “6 vertices that are all connected to each other”



Complexity bound on GNNs

- GNNs will not be able to identify interesting substructures 100% of the time in a reasonable amount of time
  - unless P = NP
- Many interesting graph problems are locked behind this complexity barrier
  - Can you color a graph with at most *k* colors?
  - Is there a path in the graph that traverses all vertices?
  - Is there a set of vertices with at least *k* edges pointing out?
- This is in theory. In practice, GNNs may be able to give very good solutions to many of these problems