Advanced Deep Learning

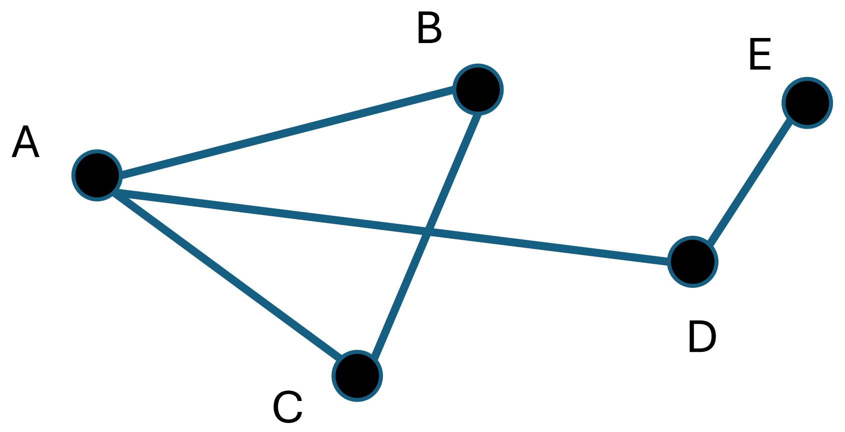
DATA.ML.230

Graph Neural Networks

From: Simon J. D. Prince, Chapter 13 – Understanding Deep Learning, MIT Press (19 May 2025)

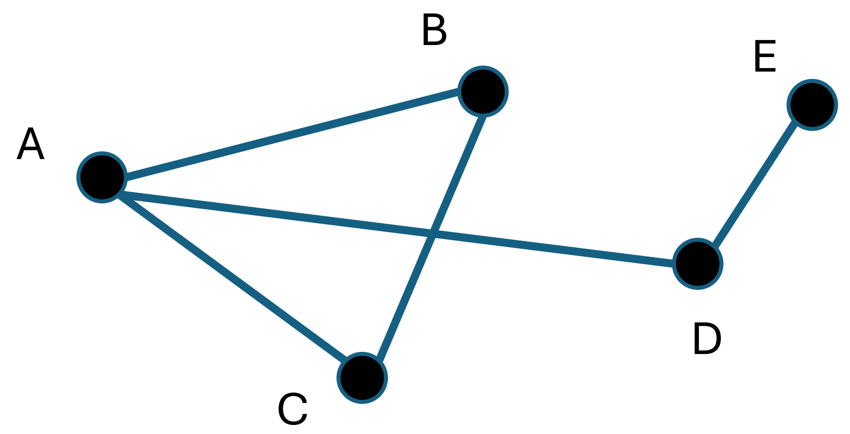
Graphs

- General structure formed by *nodes* and *edges*
- Graphs appear naturally when describing real world objects.



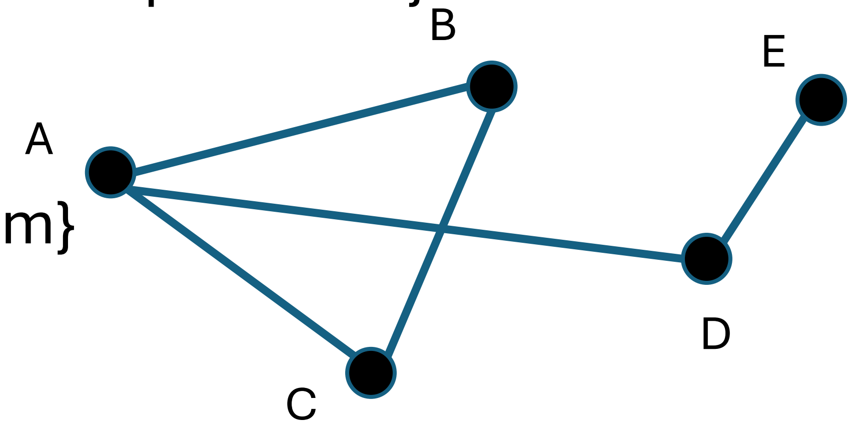
Graphs

- General structure formed by *nodes* and *edges*
- Graphs appear naturally when describing real world objects
- Graphs are a tool that can describe many different types of data



Graphs

- General structure formed by *nodes* and *edges*
- Graphs appear naturally when describing real world objects
- Graphs are a tool that can describe many different types of data. E.g.:
  - Social networks → {people, connections between them}
  - Scientific literature → {articles/papers, citations}
  - Wikipedia → {articles, hyperlinks between them}
  - Computer programs → {syntax tokens/variables, computations}
  - Point clouds → {3D point, connections}
  - Protein interactions → {proteins, interactions}
  - Any set → {objects, all connections between them}



Graphs

- General structure formed by *nodes* and *edges*
- Graphs appear naturally when describing real world objects
- Graphs are a tool that can describe many different types of data. E.g.:

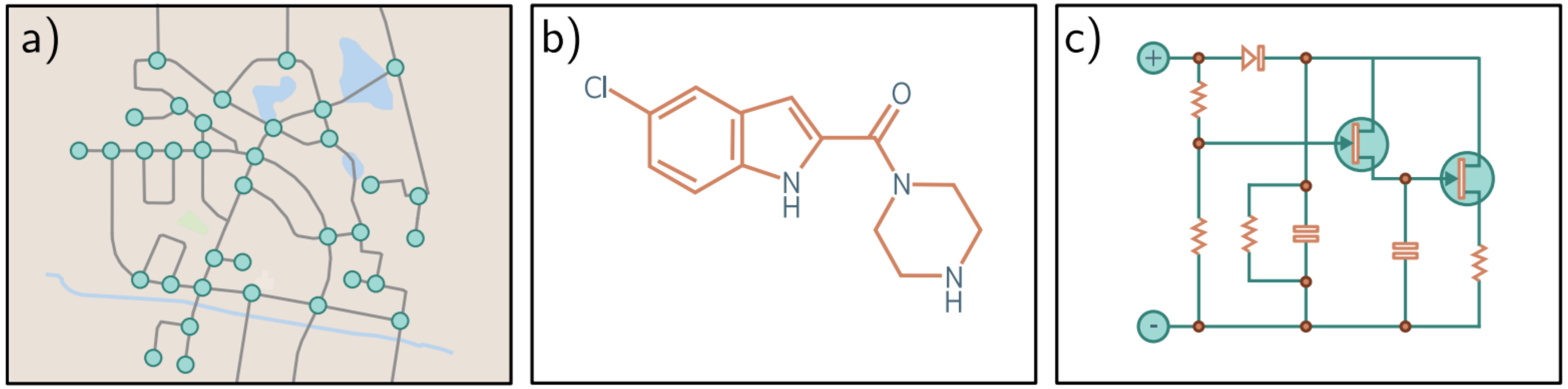
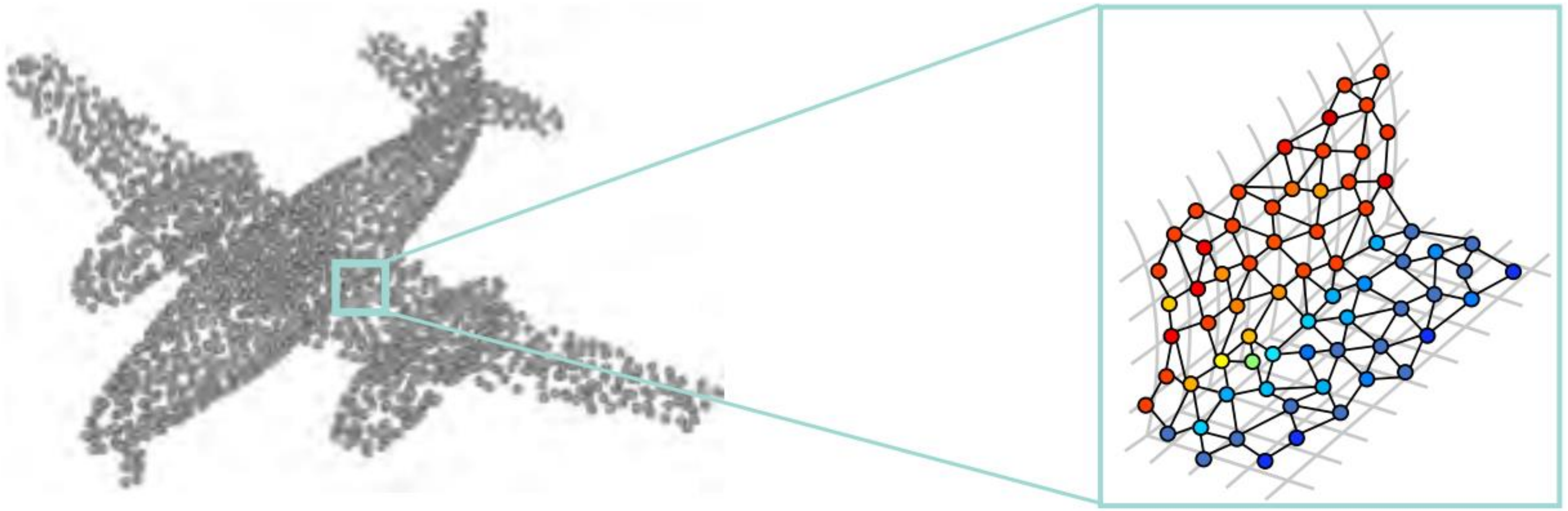


Figure 13.1 Real-world graphs. Some objects, such as a) road networks, b) molecules, and c) electrical circuits, are naturally structured as graphs.

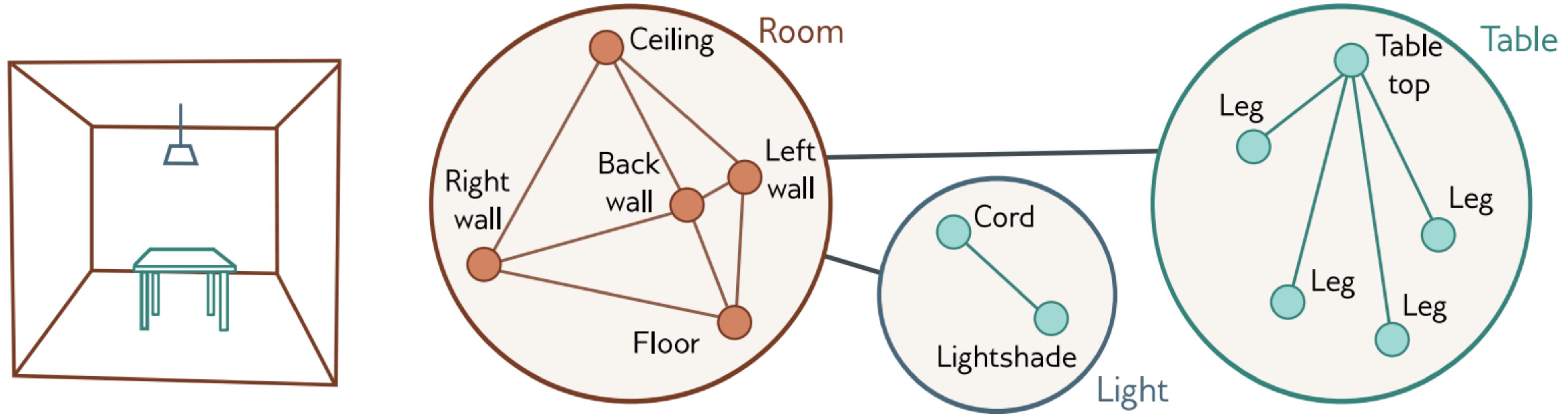
Graphs

- General structure formed by *nodes* and *edges*
- Graphs appear naturally when describing real world objects
- Graphs are a tool that can describe many different types of data. E.g.:



Graphs

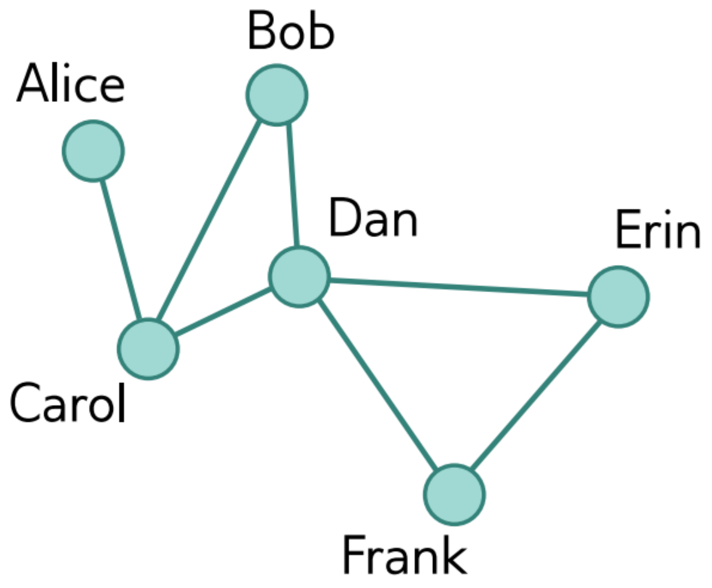
- General structure formed by *nodes* and *edges*
- Graphs appear naturally when describing real world objects
- Graphs are a tool that can describe many different types of data. E.g.:



Graphs

Depending on the type of edges connecting the nodes:

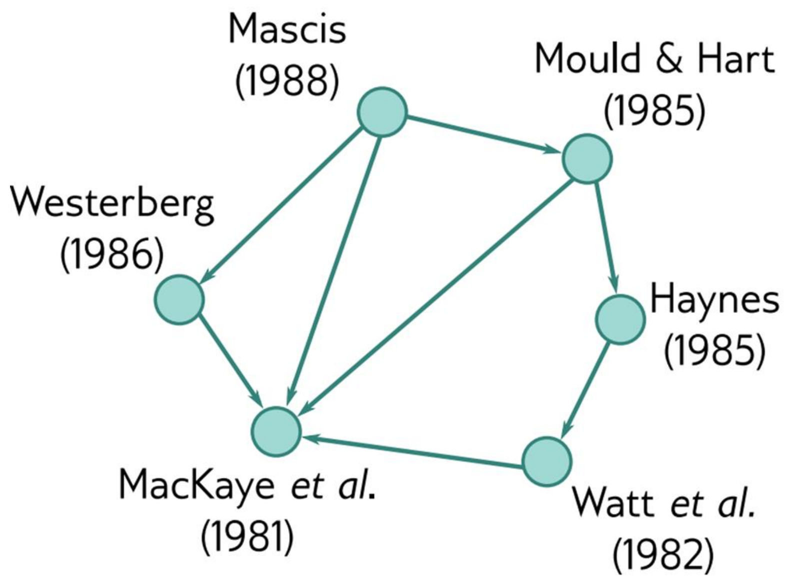
- Undirected graph → Edges do not show any direction from one node to another



Graphs

Depending on the type of edges connecting the nodes:

- Undirected graph
- Directed graph → Edges encode the direction of the connection between two nodes

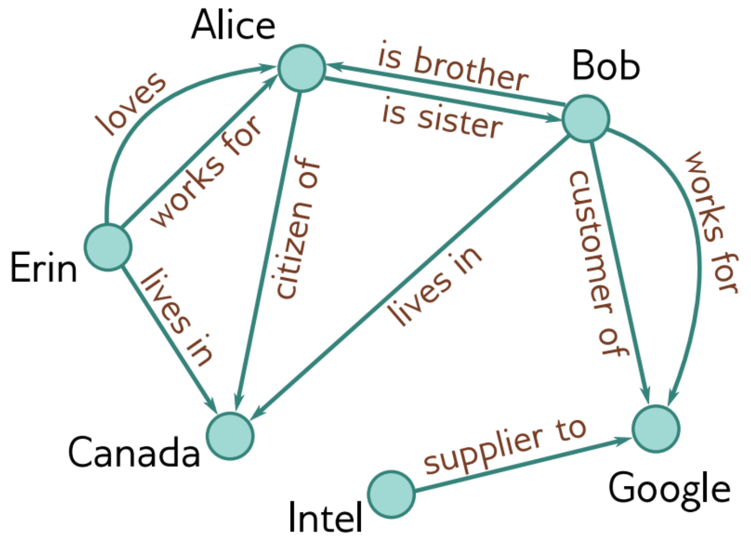


Graphs

Depending on the type of edges connecting the nodes:

- Undirected graph
- Directed graph
- Heterogeneous multigraph or hypergraph → nodes represent different types of entities and edges encode different types of connections

Knowledge graph

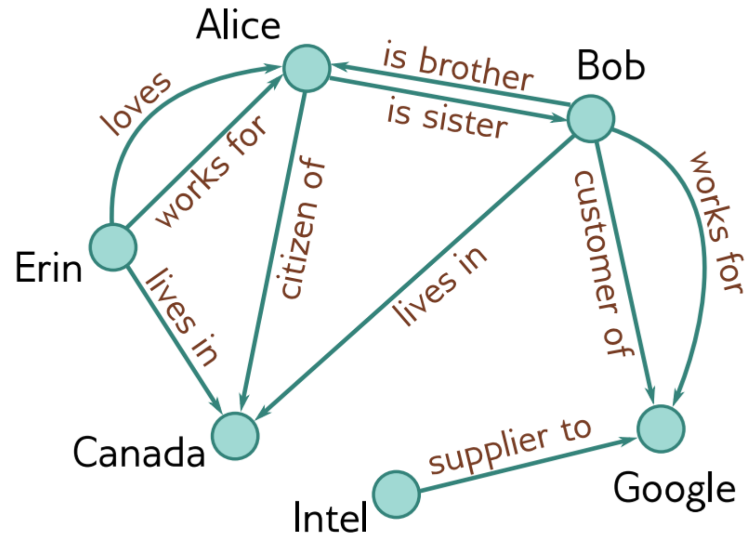


Graphs

Depending on the type of edges connecting the nodes:

- Undirected graph
- Directed graph
- Heterogeneous multigraph or hypergraph → nodes represent different types of entities and edges encode different types of connections

We will focus on undirected graphs

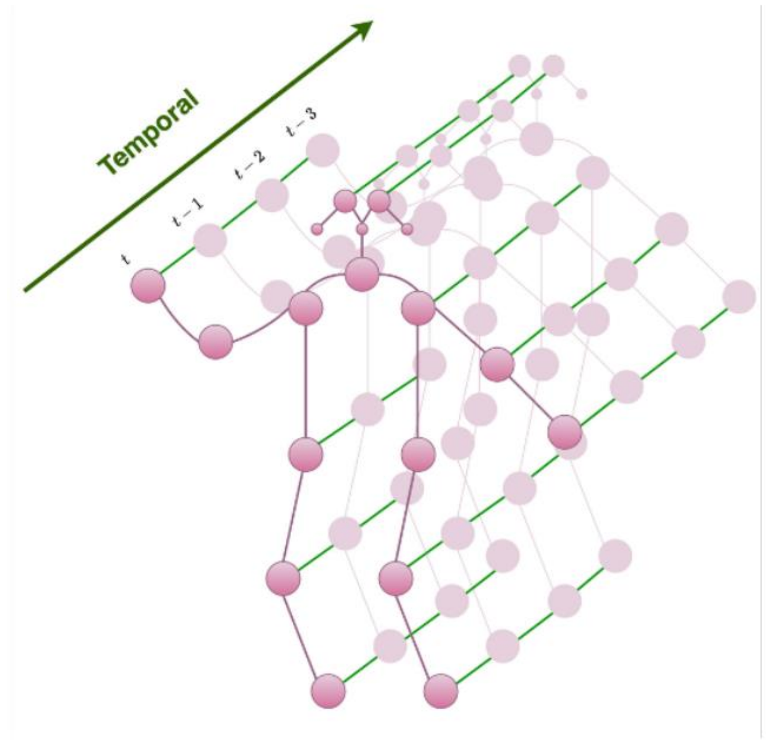


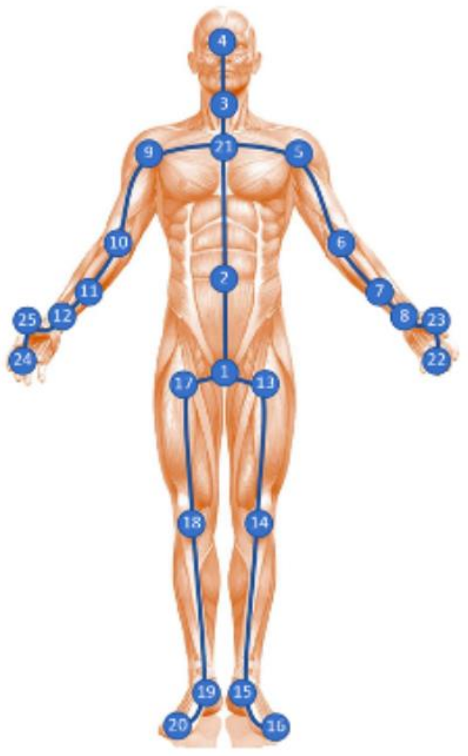
Spatio-temporal graphs

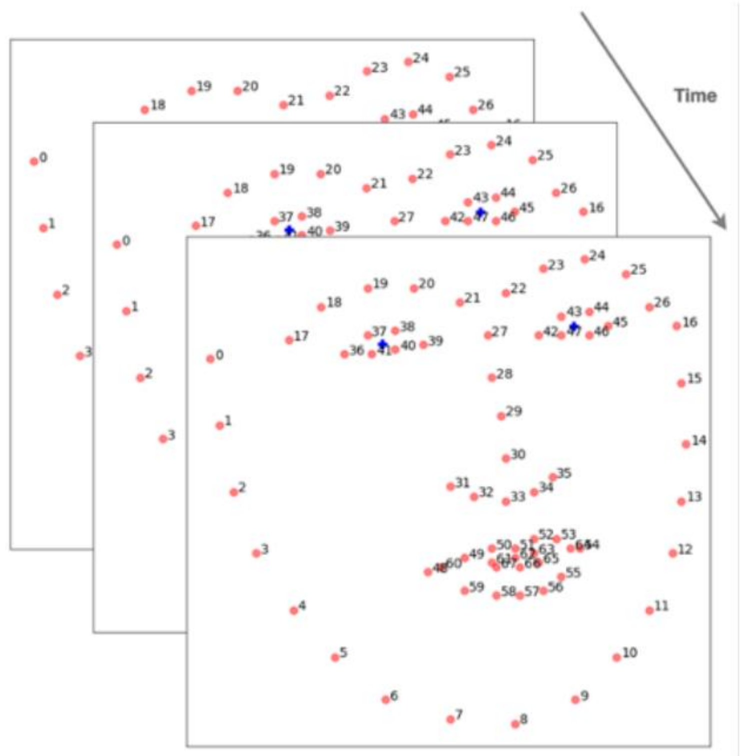
- Graphs which change over time

Spatio-temporal graphs

- Graphs which change over time. E.g.:







Processing graphs

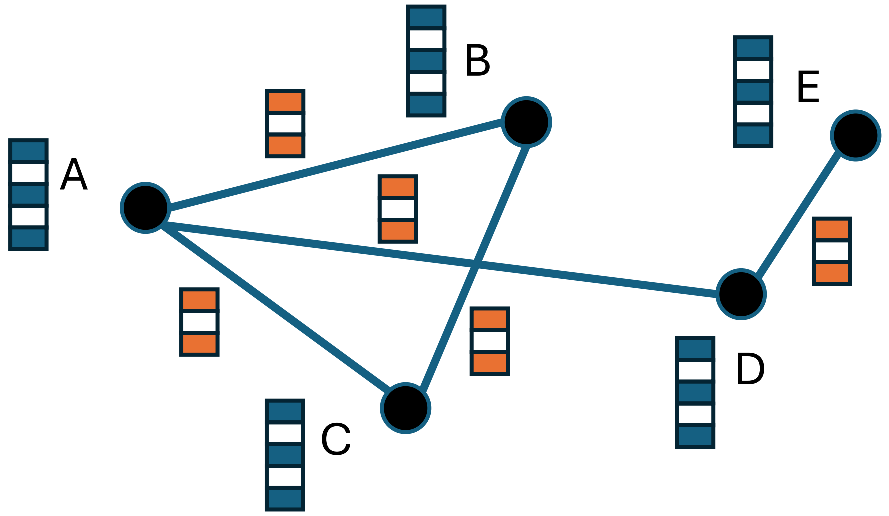
Challenges:

- **Variable topology:** hard to design networks that are sufficiently expressive and deal with this variation
- **Size:** some graphs are enormous (e.g., social networks)
- **Dataset size:** sometimes there is just one large graph to analyze and learn from (not multiple i.i.d. graph samples)

Graph representation

Two types of information *can be* available for a graph:

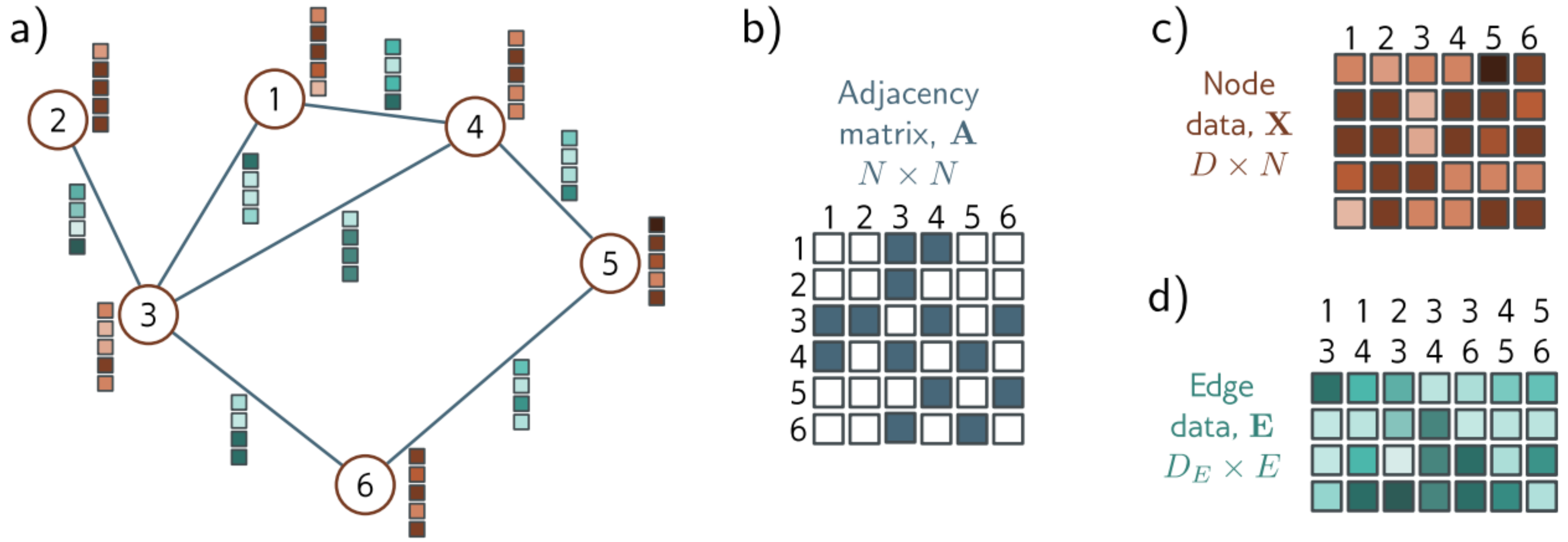
- Properties of each node (stored in the *node embedding*)
- Properties of each edge (stored in the *edge embedding*)



Graph representation

A graph consists of a set of N nodes connected by a set of E edges:

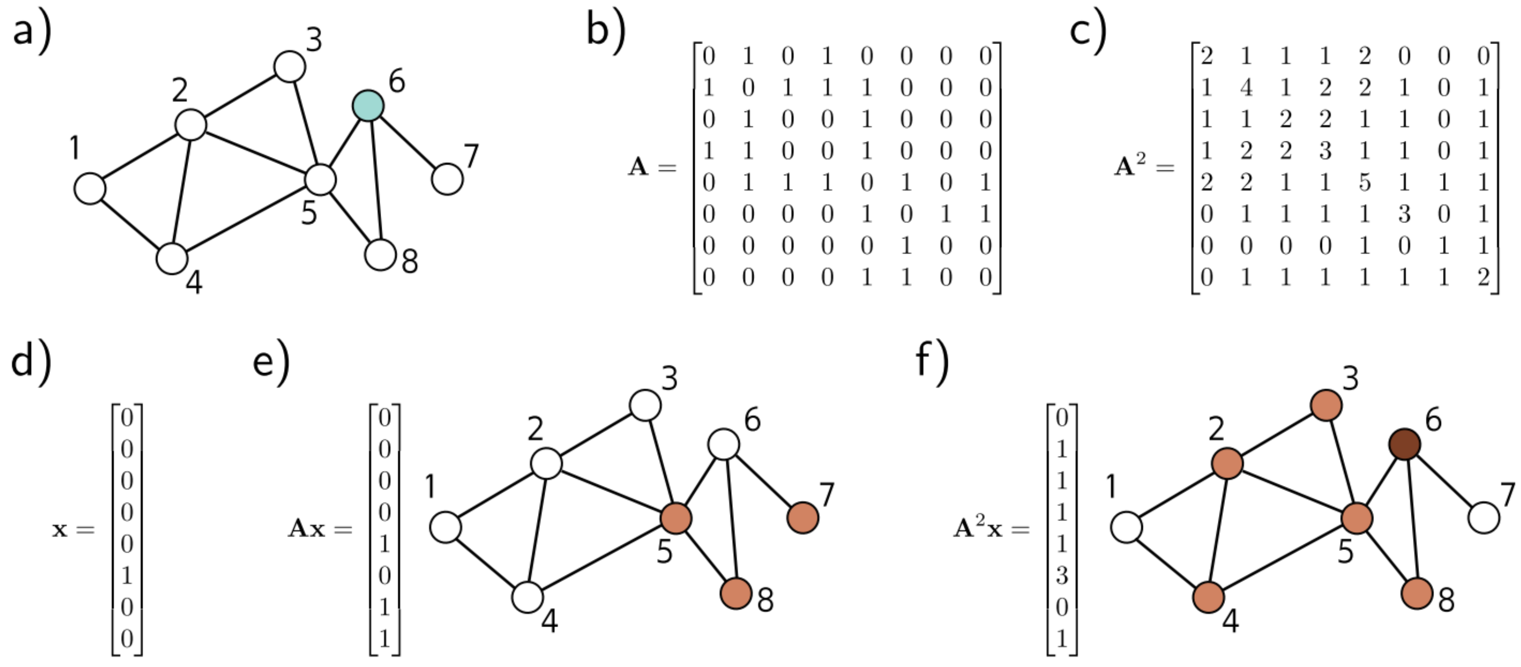
- Adjacency matrix **A** (N x N matrix) → representing the graph structure
- Node embeddings **X** (D x N matrix) → representing node properties
- Edge embeddings **E** (D_e x N matrix) → representing edge properties



Adjacency matrix

A can be used to find a node's neighbors using linear algebra:

- Raise the Adjacency matrix to the power of L → $\mathbf{A}^L$
- $\mathbf{A}^L(m,n)$: number of unique walks of length L from node m to node n
- Use the one-hot vector to represent node n → $\mathbf{x}_n$
- $\mathbf{A}^L\mathbf{x}_n$ has as non-zero elements the Lth order neighbors of node n



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

$$\mathbf{A}^2 = \begin{bmatrix} 2&1&1&1&2&0&0&0\\ 1&4&1&2&2&1&0&1\\ 1&1&2&2&1&1&0&1\\ 1&2&2&3&1&1&0&1\\ 2&2&1&1&5&1&1&1\\ 0&1&1&1&1&3&0&1\\ 0&0&0&0&1&0&1&1\\ 0&1&1&1&1&1&1&2 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} 0\\0\\0\\0\\0\\1\\0\\0 \end{bmatrix} \quad \mathbf{Ax} = \begin{bmatrix} 0\\0\\0\\0\\1\\0\\1\\1 \end{bmatrix} \quad \mathbf{A}^2\mathbf{x} = \begin{bmatrix} 0\\1\\1\\1\\1\\3\\0\\1 \end{bmatrix}$$

Figure 13.4 Properties of the adjacency matrix. a) Example graph. b) Position (m, n) of the adjacency matrix $\mathbf{A}$ contains the number of walks of length one from node m to node n. c) Position (m, n) of the squared adjacency matrix $\mathbf{A}^2$ contains the number of walks of length two from node m to node n. d) One hot vector representing node six, which was highlighted in panel (a). e) When we pre-multiply this vector by $\mathbf{A}$, the result contains the number of walks of length one from node six to each node; we can reach nodes five, seven, and eight in one move. f) When we pre-multiply this vector by $\mathbf{A}^2$, the resulting vector contains the number of walks of length two from node six to each node; we can reach nodes two, three, four, five, and eight in two moves, and we can return to the original node in three different ways (via nodes five, seven, and eight).

Adjacency matrix

The graph representation is permutation invariant:

- Changing the order of the nodes leads to an equivalent graph
- The Adjacency matrix is obtained by performing the corresponding left- and right-side permutations:

$$\begin{aligned}\mathbf{X}' &= \mathbf{X}\mathbf{P} \\ \mathbf{A}' &= \mathbf{P}^T\mathbf{A}\mathbf{P}\end{aligned}$$

Adjacency matrix

Let’s assume that the node embeddings of a graph **X** have real-valued measurements of properties of its nodes:

- What is the effect of multiplying the Adjacency matrix with the representation $\mathbf{x}_n$ of node n?

Graph convolutional networks

(Spatial) Graph convolutional networks:

- Receive as input the node embeddings **X** and Adjacency matrix **A**
- Pass them through a series of K layers **F[•, A,** ϕ**]**
- Produce an output, the structure of which depends on the task

$$\begin{aligned}
\mathbf{H}_1 &= \mathbf{F}[\mathbf{X}, \mathbf{A}, \phi_0] \\
\mathbf{H}_2 &= \mathbf{F}[\mathbf{H}_1, \mathbf{A}, \phi_1] \\
\mathbf{H}_3 &= \mathbf{F}[\mathbf{H}_2, \mathbf{A}, \phi_2] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{F}[\mathbf{H}_{K-1}, \mathbf{A}, \phi_{K-1}]
\end{aligned}$$

Graph convolutional networks

(Spatial) Graph convolutional networks:

- Receive as input the node embeddings **X** and Adjacency matrix **A**
- Pass them through a series of K layers **F[•, A,** ϕ]
- Produce an output, the structure of which depends on the task

$$\begin{aligned}\mathbf{H}_1 &= \mathbf{F}[\mathbf{X}, \mathbf{A}, \phi_0] \\ \mathbf{H}_2 &= \mathbf{F}[\mathbf{H}_1, \mathbf{A}, \phi_1] \\ \mathbf{H}_3 &= \mathbf{F}[\mathbf{H}_2, \mathbf{A}, \phi_2] \\ \vdots &= \vdots \\ \mathbf{H}_K &= \mathbf{F}[\mathbf{H}_{K-1}, \mathbf{A}, \phi_{K-1}]\end{aligned}$$

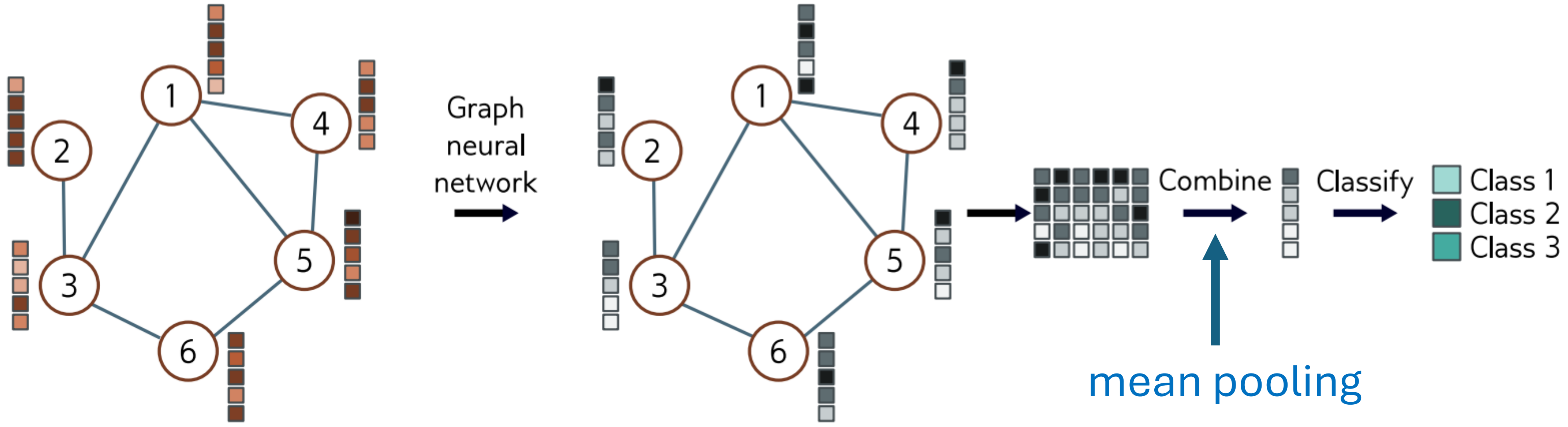
GCN layers are permutation invariant

$$\mathbf{H}_{k+1}\mathbf{P} = \mathbf{F}[\mathbf{H}_k\mathbf{P}, \mathbf{P}^T\mathbf{A}\mathbf{P}, \phi_k]$$

Graph learning tasks

Supervised graph problems:

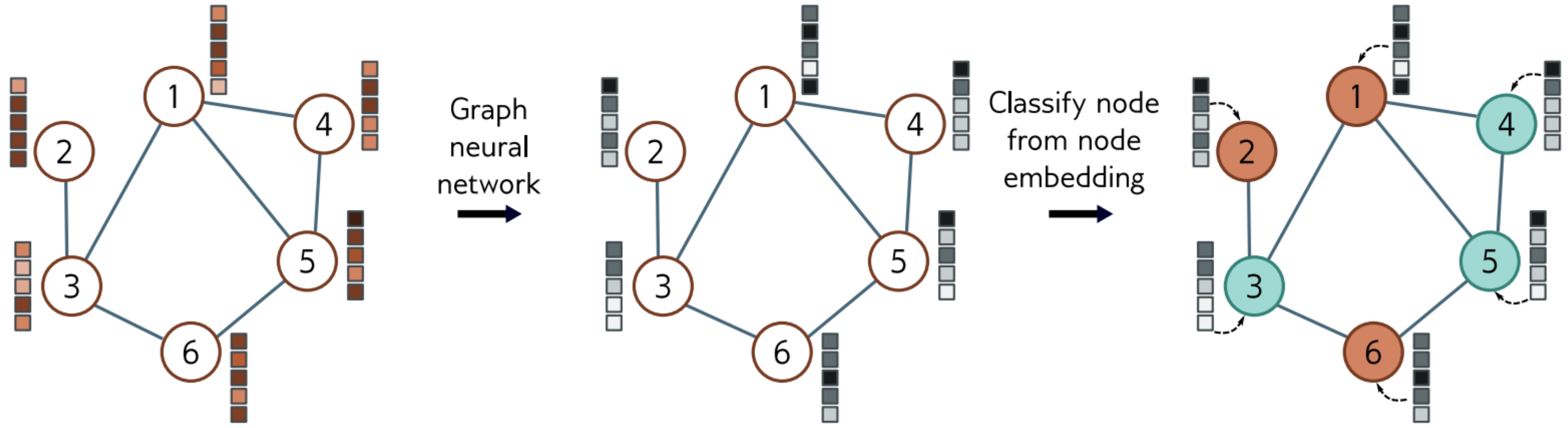
- Graph-level tasks: Assign one label (or regress to one value/vector) to the entire graph, exploiting both the graph structure and the node embeddings



Graph learning tasks

Supervised graph problems:

- Node-level tasks: Assign one label (or regress to one value/vector) to each node of the graph, exploiting both the graph structure and the node embeddings

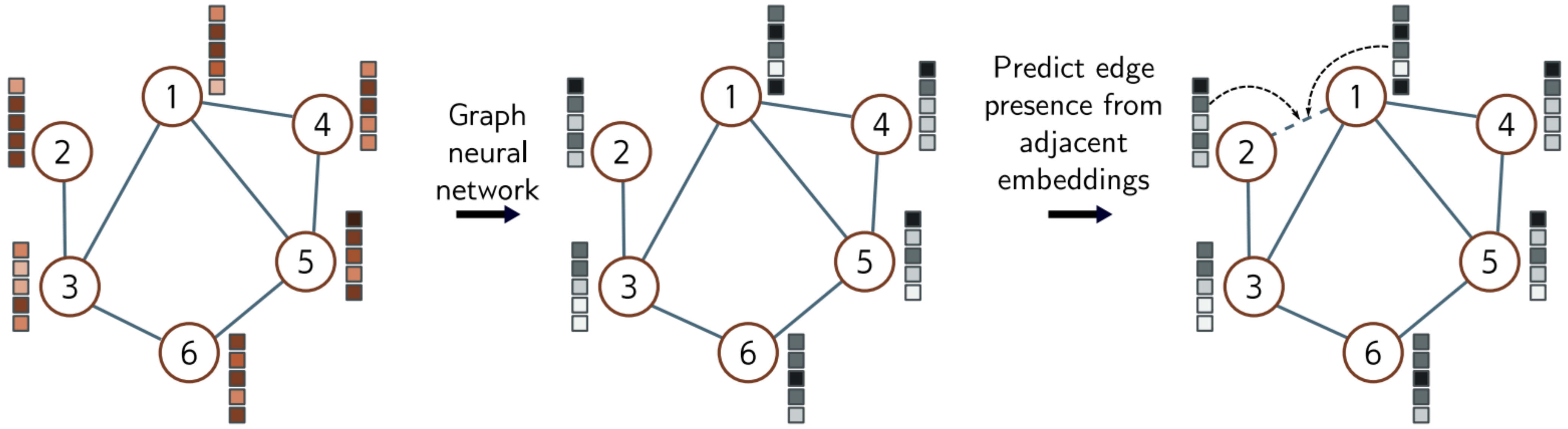


Graph learning tasks

Supervised graph problems:

- Edge prediction tasks: Predict whether there should be an edge between nodes n and m

➔ Binary classification task



Graph convolutional networks

GGN layer:

- At each node n in layer k aggregate information from neighbors:

$$\mathbf{agg}[n,k] = \sum_{m\in\mathrm{ne}[n]} \mathbf{h}_k^{(m)}$$

- Perform linear transformation $\boldsymbol{\Omega}_k$ to the embedding of node n and the aggregate vector, and add a bias term

$$\mathbf{h}_{k+1}^{(n)} = \mathbf{a}\left[\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \cdot \mathbf{h}_k^{(n)} + \boldsymbol{\Omega}_k \cdot \mathbf{agg}[n,k]\right]$$

Graph convolutional networks

GGN layer:

- At each node n in layer k aggregate information from neighbors:

$$\mathbf{agg}[n,k] = \sum_{m \in \mathrm{ne}[n]} \mathbf{h}_k^{(m)}$$

- Perform linear transformation $\mathbf{\Omega}_k$ to the embedding of node n and the aggregate vector, and add a bias term

$$\mathbf{h}_{k+1}^{(n)} = \mathbf{a}\left[\boldsymbol{\beta}_k + \mathbf{\Omega}_k \cdot \mathbf{h}_k^{(n)} + \mathbf{\Omega}_k \cdot \mathbf{agg}[n,k]\right]$$

Does this remind us of anything?

Graph convolutional networks

GGN layer (compact form):

$$\begin{aligned}\mathbf{H}_{k+1} &= \mathbf{a}\left[\boldsymbol{\beta}_k\mathbf{1}^T + \boldsymbol{\Omega}_k\mathbf{H}_k + \boldsymbol{\Omega}_k\mathbf{H}_k\mathbf{A}\right]\\ &= \mathbf{a}\left[\boldsymbol{\beta}_k\mathbf{1}^T + \boldsymbol{\Omega}_k\mathbf{H}_k(\mathbf{A}+\mathbf{I})\right]\end{aligned}$$

Graph convolutional networks

Graph classification with GCN

Inputs: Node embedding matrix $\mathbf{X} \in \mathbb{R}^{D\text{x}N}$

Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\text{x}N}$

Graph convolutional networks

Graph classification with GCN

Inputs: Node embedding matrix **X** $\in \mathbb{R}^{DxN}$

Adjacency matrix **A** $\in \mathbb{R}^{NxN}$

Node embedding transformations through K GCN layers:

$$\begin{aligned}
\mathbf{H}_1 &= \mathbf{a}\left[\boldsymbol{\beta}_0\mathbf{1}^T + \boldsymbol{\Omega}_0\mathbf{X}(\mathbf{A}+\mathbf{I})\right] \\
\mathbf{H}_2 &= \mathbf{a}\left[\boldsymbol{\beta}_1\mathbf{1}^T + \boldsymbol{\Omega}_1\mathbf{H}_1(\mathbf{A}+\mathbf{I})\right] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{a}\left[\boldsymbol{\beta}_{K-1}\mathbf{1}^T + \boldsymbol{\Omega}_{K-1}\mathbf{H}_{k-1}(\mathbf{A}+\mathbf{I})\right]
\end{aligned}$$

Graph convolutional networks

Graph classification with GCN

Inputs: Node embedding matrix **X** $\in \mathbb{R}^{DxN}$

Adjacency matrix **A** $\in \mathbb{R}^{NxN}$

Node embedding transformations through K GCN layers:

$$\begin{aligned}
\mathbf{H}_1 &= \mathbf{a}\left[\boldsymbol{\beta}_0\mathbf{1}^T + \boldsymbol{\Omega}_0\mathbf{X}(\mathbf{A}+\mathbf{I})\right] \\
\mathbf{H}_2 &= \mathbf{a}\left[\boldsymbol{\beta}_1\mathbf{1}^T + \boldsymbol{\Omega}_1\mathbf{H}_1(\mathbf{A}+\mathbf{I})\right] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{a}\left[\boldsymbol{\beta}_{K-1}\mathbf{1}^T + \boldsymbol{\Omega}_{K-1}\mathbf{H}_{k-1}(\mathbf{A}+\mathbf{I})\right]
\end{aligned}$$

Classification layer output: $\mathrm{f}[\mathbf{X}, \mathbf{A}, \boldsymbol{\Phi}] \quad = \quad \mathrm{sig}\left[\beta_K + \boldsymbol{\omega}_K \mathbf{H}_K \mathbf{1}/N\right]$

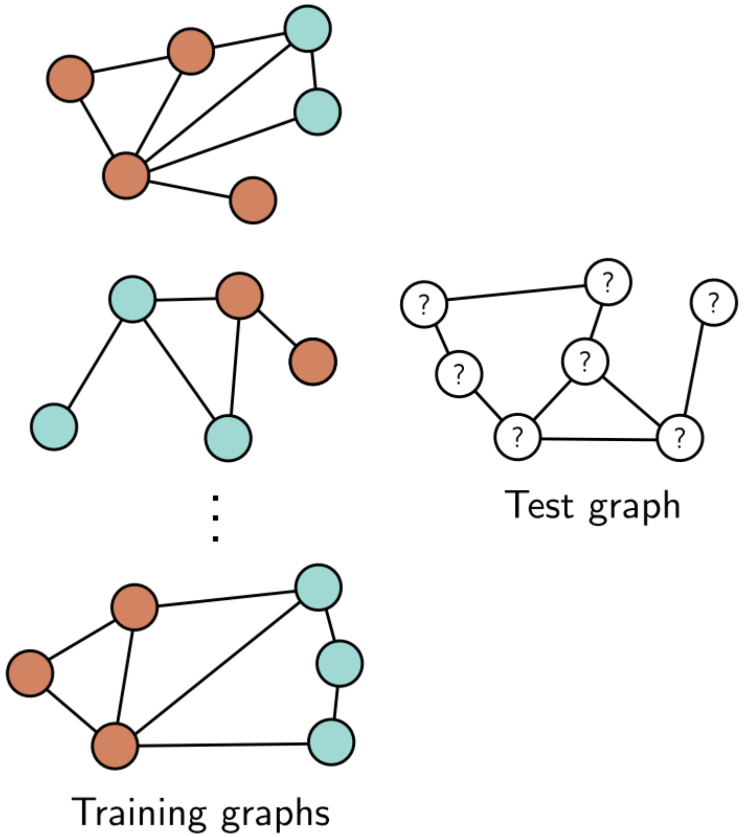
Graph convolutional networks

Graph classification with GCN

Training set: I training graphs $\{\mathbf{X}_i, \mathbf{A}_i\}$, each followed by a label y_i

Parameters to be learned: $\mathbf{\Phi} = \{\boldsymbol{\beta}_k, \mathbf{\Omega}_k\}_{k=0}^{K}$

Loss function: Cross-entropy



Graph convolutional networks

Graph classification with GCN

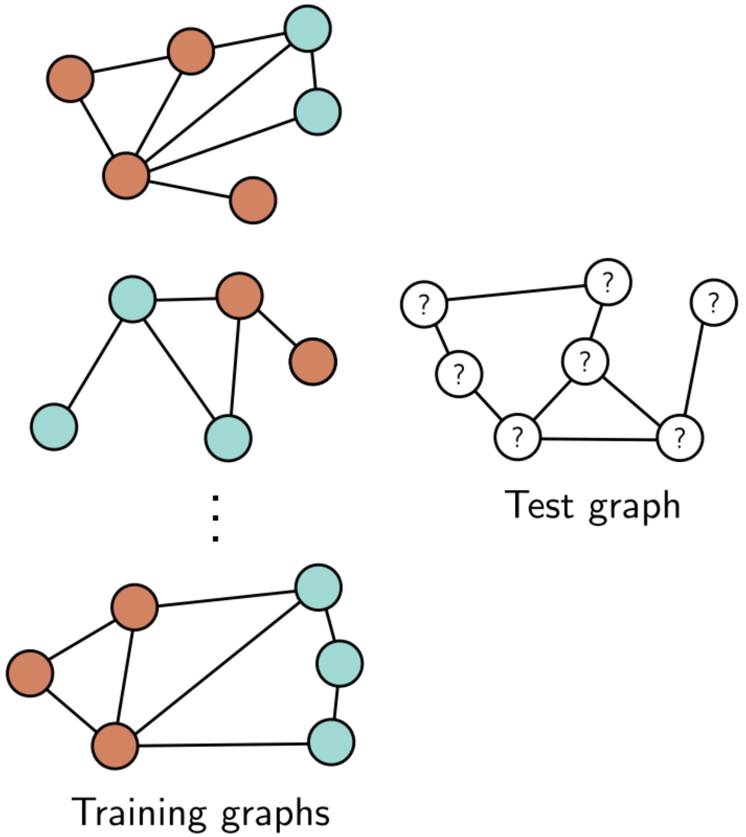
Training set: I training graphs $\{\mathbf{X}_i, \mathbf{A}_i\}$, each followed by a label y_i

Parameters to be learned: $\boldsymbol{\Phi} = \{\boldsymbol{\beta}_k, \boldsymbol{\Omega}_k\}_{k=0}^{K}$

Loss function: Cross-entropy

Trick for creating mini-batches:

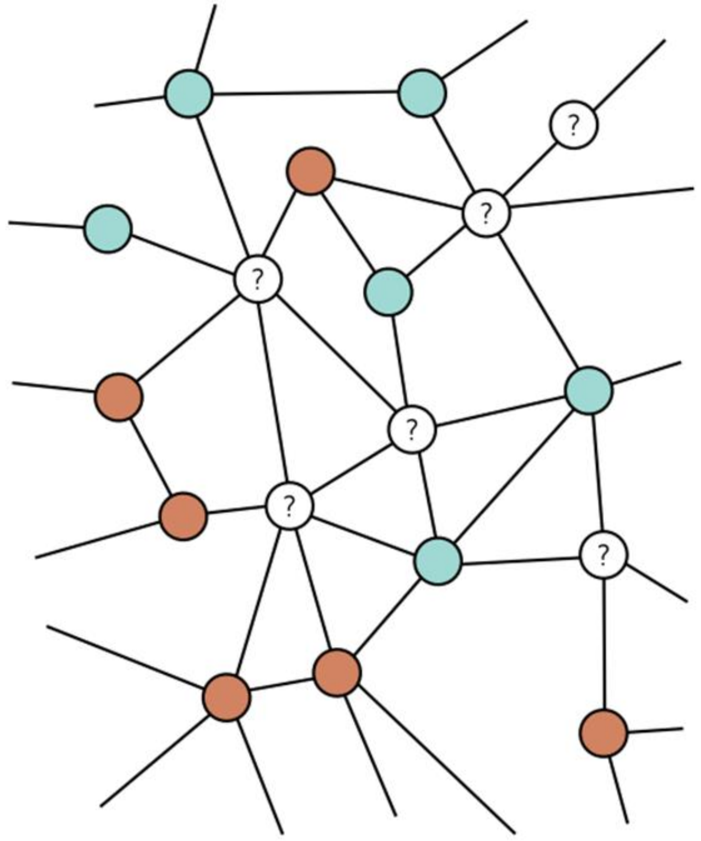
- Create one large graph formed by the graph samples in the mini-batch
- Perform pooling at the last layer for each individual graph sample



Transductive or semi-supervised learning

- Considers both labeled and unlabeled data at the same time
- It produces a labeling for the unknown outputs

Advantage: Unlabeled samples can help in improving performance
Disadvantage: The model needs to be trained again when new unlabeled data are added



Graph convolutional networks

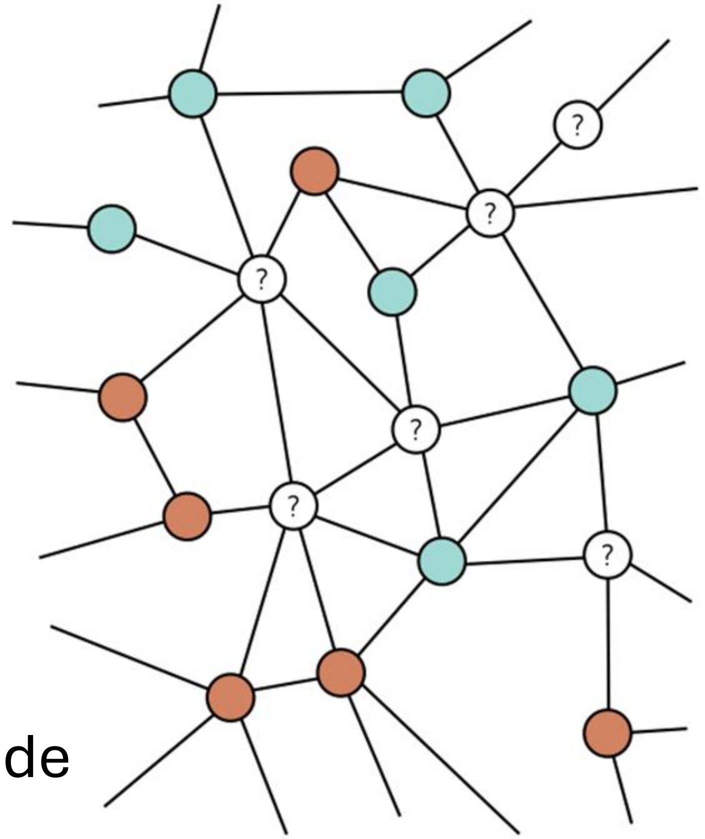
Node classification with GCN

Same GCN structure as the ones used for *inductive learning*:

$$
\begin{aligned}
\mathbf{H}_1 &= \mathbf{a}\left[\boldsymbol{\beta}_0\mathbf{1}^T + \boldsymbol{\Omega}_0\mathbf{X}(\mathbf{A}+\mathbf{I})\right] \\
\mathbf{H}_2 &= \mathbf{a}\left[\boldsymbol{\beta}_1\mathbf{1}^T + \boldsymbol{\Omega}_1\mathbf{H}_1(\mathbf{A}+\mathbf{I})\right] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{a}\left[\boldsymbol{\beta}_{K-1}\mathbf{1}^T + \boldsymbol{\Omega}_{K-1}\mathbf{H}_{k-1}(\mathbf{A}+\mathbf{I})\right]
\end{aligned}
$$

Final layer produces an output vector of size 1xN:

$$
\mathbf{f}[\mathbf{X}, \mathbf{A}, \boldsymbol{\Phi}] = \mathbf{sig}\left[\beta_K\mathbf{1}^T + \boldsymbol{\omega}_K\mathbf{H}_K\right]
$$



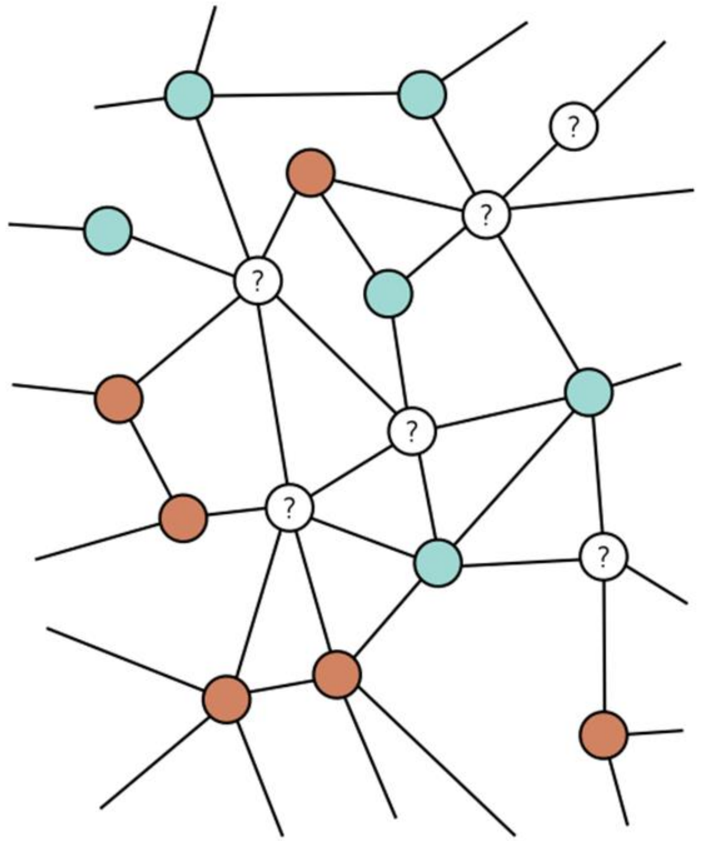
Loss function: Binary cross-entropy for each unlabeled node

Graph convolutional networks

Node classification with GCN

Choosing batches: Using randomly sampled nodes leads to the graph expansion problem

→ Receptive field which reaches the full graph



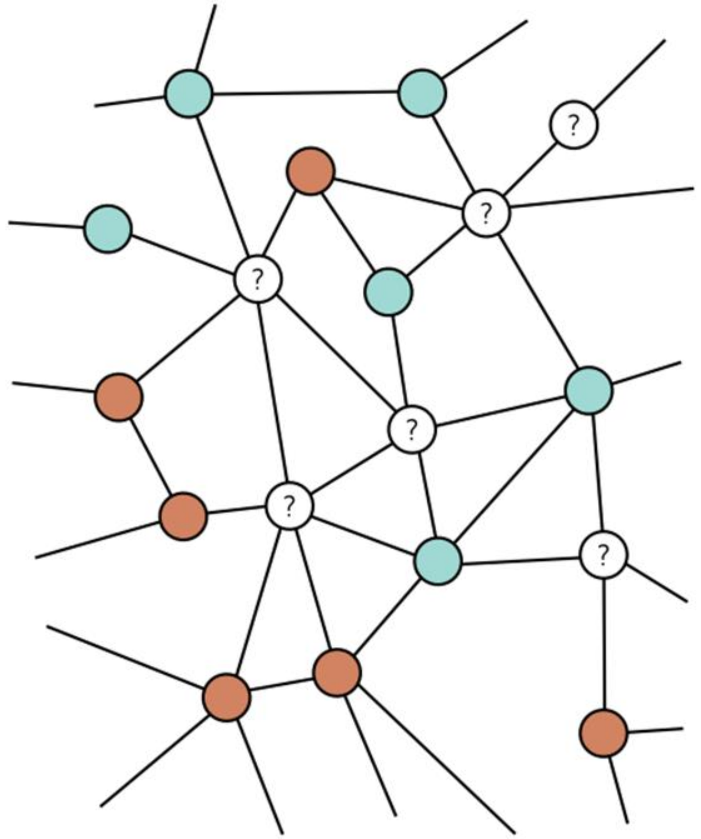
Graph convolutional networks

Node classification with GCN

Choosing batches: Using randomly sampled nodes leads to the graph expansion problem

→ Receptive field which reaches the full graph

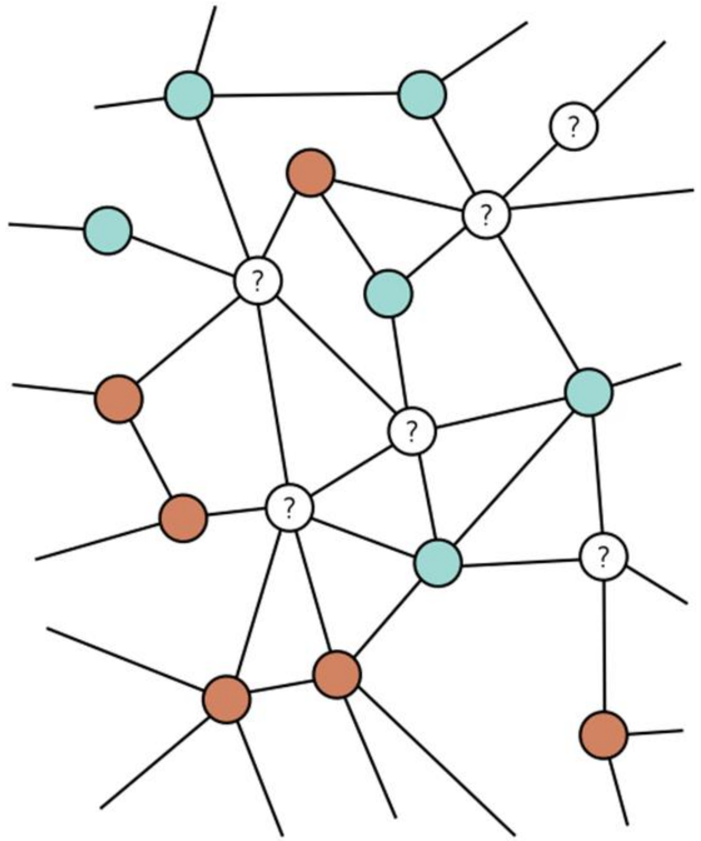
The receptive field region for a node is called *k-hop neighborhood*



Graph convolutional networks

Node classification with GCN

Choosing batches: Using randomly sampled nodes leads to the graph expansion problem



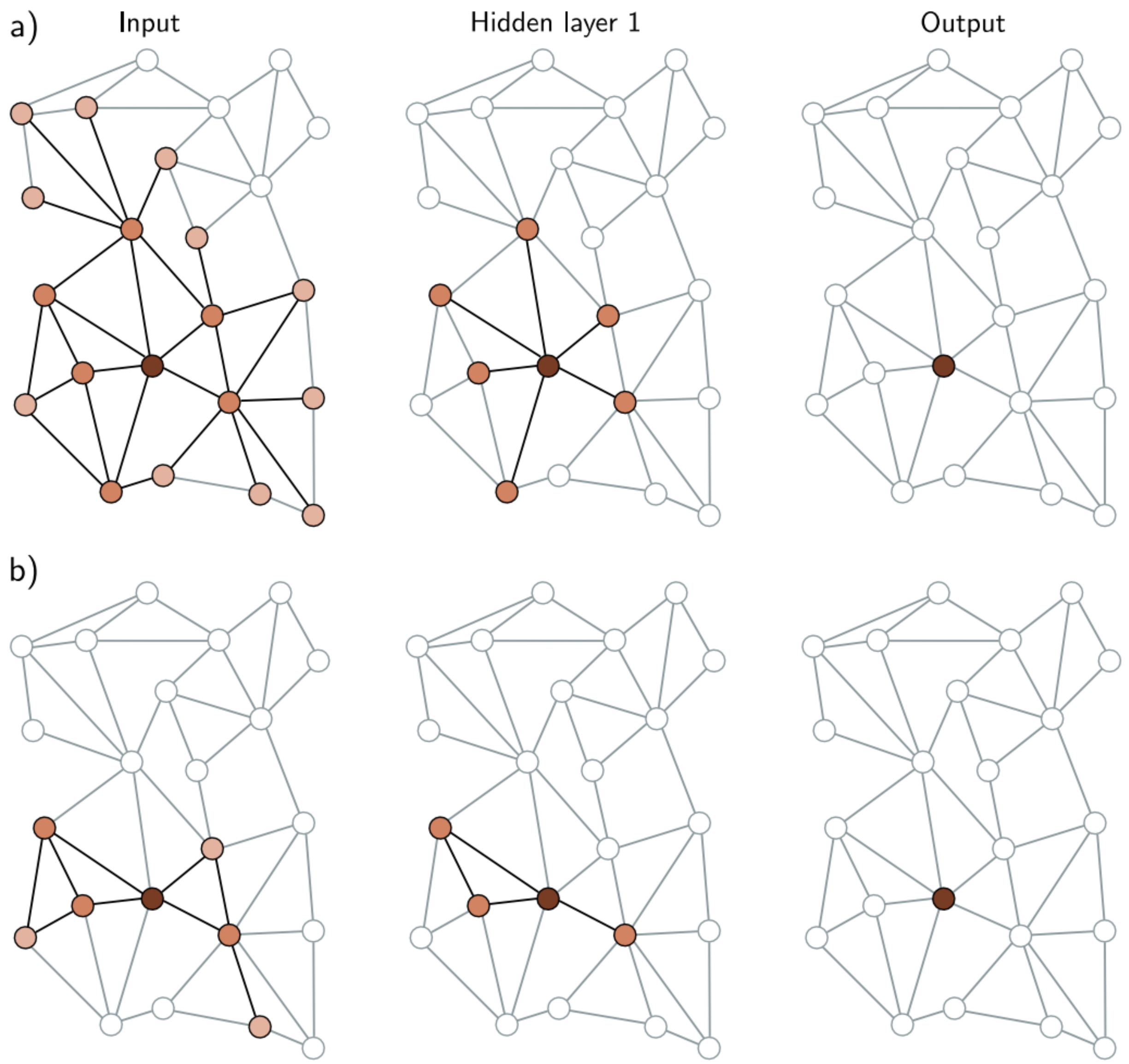
Solutions:

- Neighborhood sampling
- Graph partitioning

Graph convolutional networks

Neighborhood sampling

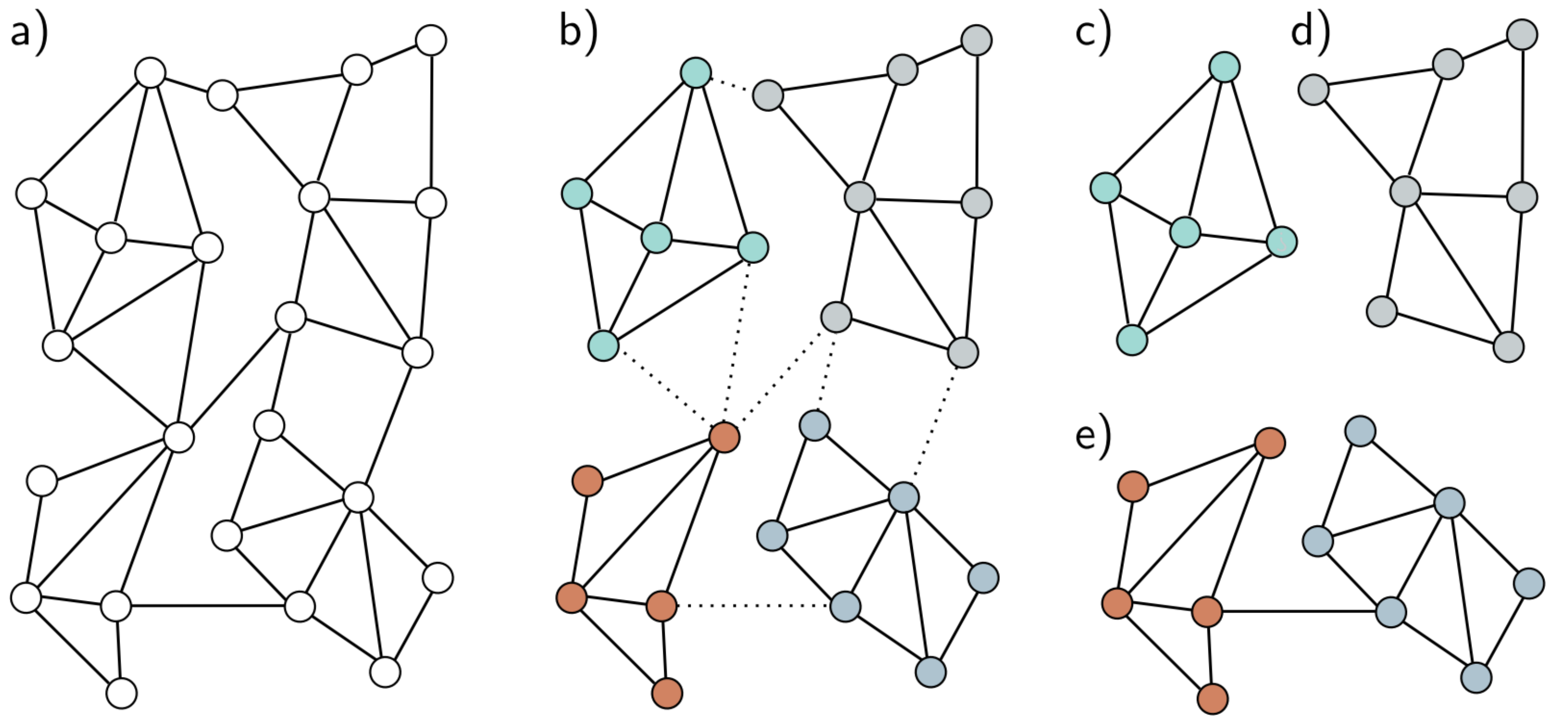
- Sample only a fixed number of neighborhoods for each node in the mini-batch at each layer



Graph convolutional networks

Graph partitioning

- Cluster the original graph into disjoint subsets of nodes
- Each smaller graph (or a random subset of them) is treated as a mini-batch



Layers for GCNs

- Combination of current node and aggregated neighbors (diagonal enhancement):

$$\mathbf{H}_{k+1} = \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k (\mathbf{A} + (1 + \epsilon_k)\mathbf{I})\right]$$

Learned scalar

Layers for GCNs

- Combination of current node and aggregated neighbors (diagonal enhancement):

$$\mathbf{H}_{k+1} = \mathbf{a}\Big[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k (\mathbf{A} + (1+\epsilon_k)\mathbf{I})\Big]$$

- Learn different transformation for the current node:

$$\begin{aligned} \mathbf{H}_{k+1} &= \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k \mathbf{A} + \boldsymbol{\Psi}_k \mathbf{H}_k\right] \\ &= \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \begin{bmatrix} \boldsymbol{\Omega}_k & \boldsymbol{\Psi}_k \end{bmatrix} \begin{bmatrix} \mathbf{H}_k \mathbf{A} \\ \mathbf{H}_k \end{bmatrix}\right] \\ &= \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}'_k \begin{bmatrix} \mathbf{H}_k \mathbf{A} \\ \mathbf{H}_k \end{bmatrix}\right], \end{aligned}$$

where $\boldsymbol{\Omega}'_k = \begin{bmatrix} \boldsymbol{\Omega}_k & \boldsymbol{\Psi}_k \end{bmatrix}$

Layers for GCNs

- <u>Residual connections:</u> the aggregated representation from the neighbors is transformed and passed through the activation function before summation or concatenation with the current node:

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k \mathbf{A}\right] \\ \mathbf{H}_k \end{bmatrix}$$

Layers for GCNs

- <u>Residual connections:</u> the aggregated representation from the neighbors is transformed and passed through the activation function before summation or concatenation with the current node:

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k \mathbf{A}\right] \\ \mathbf{H}_k \end{bmatrix}$$

- <u>Mean aggregation:</u> Use mean for not highlighting nodes with many connections:

$$\mathbf{agg}[n] = \frac{1}{|\mathrm{ne}[n]|} \sum_{m \in \mathrm{ne}[n]} \mathbf{h}_m$$

Degree matrix (diagonal)

$$\mathbf{H}_{k+1} = \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k (\mathbf{A}\mathbf{D}^{-1} + \mathbf{I})\right]$$

Layers for GCNs

- Kipf normalization: normalize the aggregated representation so that information coming from nodes with a very large number of neighbors is down-weighted:

$$\mathbf{agg}[n] = \sum_{m \in \text{ne}[n]} \frac{\mathbf{h}_m}{\sqrt{|\text{ne}[n]||\text{ne}[m]|}}$$

$$\mathbf{H}_{k+1} = \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k (\mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} + \mathbf{I})\right]$$

Layers for GCNs

- Kipf normalization: normalize the aggregated representation so that information coming from nodes with a very large number of neighbors is down-weighted:

$$\mathbf{agg}[n] = \sum_{m \in \mathrm{ne}[n]} \frac{\mathbf{h}_m}{\sqrt{|\mathrm{ne}[n]||\mathrm{ne}[m]|}}$$

$$\mathbf{H}_{k+1} = \mathbf{a}\left[\boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} + \mathbf{I})\right]$$

- Max pooling aggregation: use the per-dimension maximum value of the neighbors:

$$\mathbf{agg}[n] = \max_{m \in \mathrm{ne}[n]} \left[\mathbf{h}_m\right]$$

Layers for GCNs

- Aggregation by attention: Weight the contribution of the neighbors based on the data at the corresponding nodes:

$$\mathbf{H}'_k = \boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{H}_k$$

Calculate the similarity of each transformed node embedding $\mathbf{h}'_m$ to the transformed node embedding $\mathbf{h}'_n$ using a learnable vector $\boldsymbol{\phi}_k$:

$$s_{mn} = \mathrm{a}\left[\boldsymbol{\phi}_k^T \begin{bmatrix} \mathbf{h}'_m \\ \mathbf{h}'_n \end{bmatrix}\right]$$

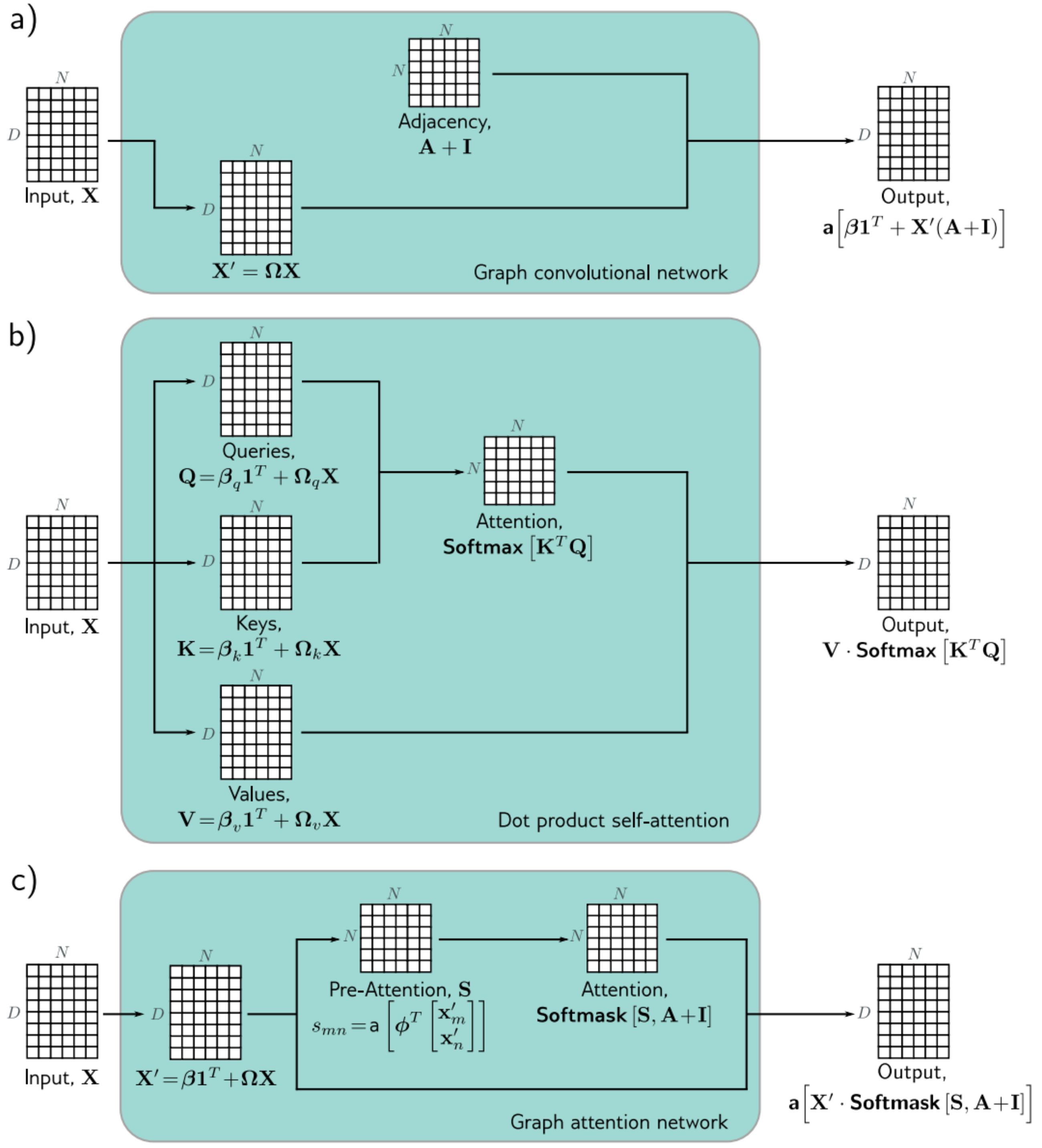
Apply the attention weights to the transformed embeddings:

$$\mathbf{H}_{k+1} = \mathbf{a}\Big[\mathbf{H}'_k \cdot \mathbf{Softmask}[\mathbf{S}, \mathbf{A} + \mathbf{I}]\Big]$$

Layers for GCNs

Softmask[**S**, **A**+**I**]:

- Find the positions (m,n) in which the elements of the matrix (**A**+**I**) $_{(m,n)}$ = 0
- Set the corresponding elements (m,n) of the matrix S(m,n) = -∞
- Applying softmax separately to each column of the resulting matrix **S**


a)
Input, $\mathbf{X}$
$\mathbf{X}' = \mathbf{\Omega X}$
Adjacency, $\mathbf{A}+\mathbf{I}$
Output, $\mathbf{a}\left[\boldsymbol{\beta}\mathbf{1}^T + \mathbf{X}'(\mathbf{A}+\mathbf{I})\right]$
Graph convolutional network
b)
Input, $\mathbf{X}$
Queries, $\mathbf{Q}=\boldsymbol{\beta}_q\mathbf{1}^T+\mathbf{\Omega}_q\mathbf{X}$
Keys, $\mathbf{K}=\boldsymbol{\beta}_k\mathbf{1}^T+\mathbf{\Omega}_k\mathbf{X}$
Values, $\mathbf{V}=\boldsymbol{\beta}_v\mathbf{1}^T+\mathbf{\Omega}_v\mathbf{X}$
Attention, Softmax $\left[\mathbf{K}^T\mathbf{Q}\right]$
Output, $\mathbf{V}\cdot$ Softmax $\left[\mathbf{K}^T\mathbf{Q}\right]$
Dot product self-attention
c)
Input, $\mathbf{X}$
$\mathbf{X}'=\boldsymbol{\beta}\mathbf{1}^T+\mathbf{\Omega X}$
Pre-Attention, $\mathbf{S}$
$s_{mn}=\mathrm{a}\left[\boldsymbol{\phi}^T\begin{bmatrix}\mathbf{x}'_m\\ \mathbf{x}'_n\end{bmatrix}\right]$
Attention, Softmask $[\mathbf{S}, \mathbf{A}+\mathbf{I}]$
Output, $\mathbf{a}\left[\mathbf{X}'\cdot\text{Softmask}[\mathbf{S},\mathbf{A}+\mathbf{I}]\right]$
Graph attention network
N
D

Edge graphs

- Use the *edge graph* (also called *adjoint graph* or *line graph*):
  - Nodes of the edge graph are the edges
  - Every two edges with a common node in the original graph create an edge in the new graph
- It is possible to swap between the two types of graphs
- Use the same techniques described above for the node graph

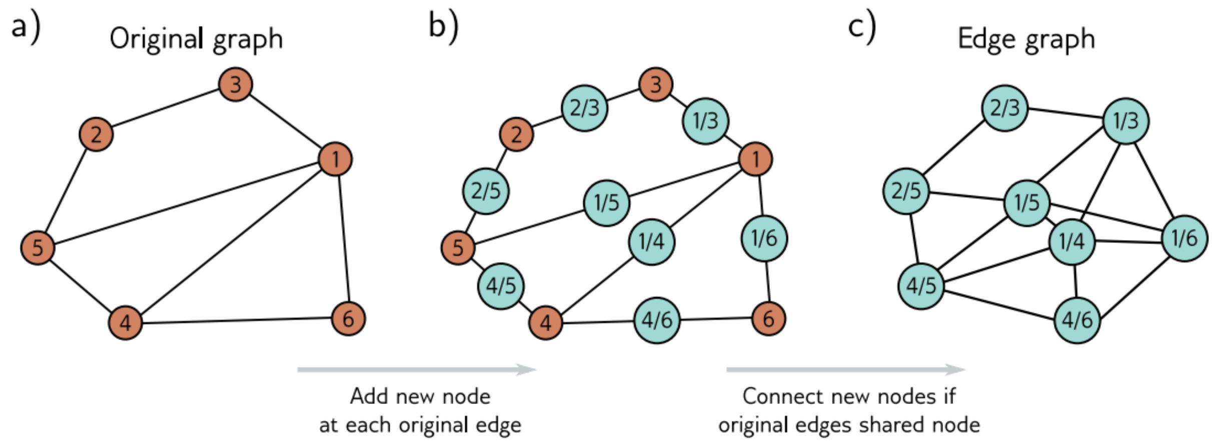
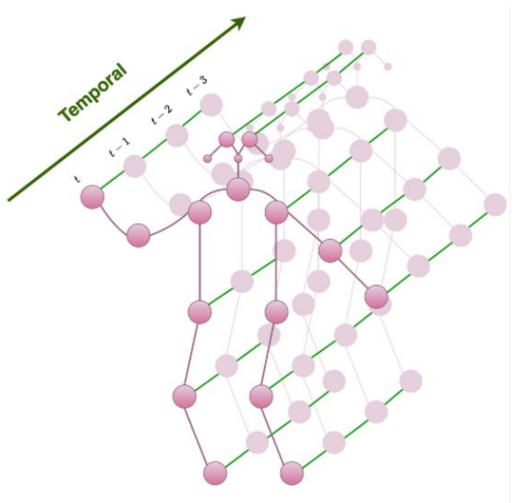

a)
Original graph
b)
c)
Edge graph
Add new node at each original edge
Connect new nodes if original edges shared node

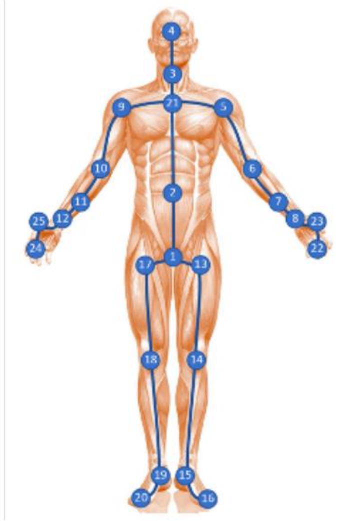
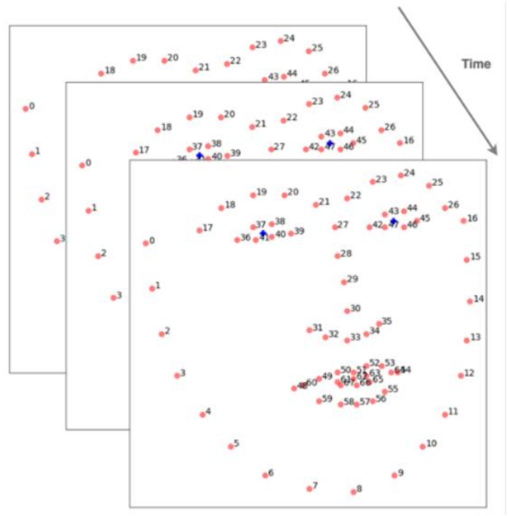
Figure 13.13 Edge graph. a) Graph with six nodes. b) To create the edge graph, we assign one node for each original edge (cyan circles), and c) connect the new nodes if the edges they represent connect to the same node in the original graph.

Spatio-Temporal graph structure

Spatio-Temporal graph:

- Nodes: 2D or 3D joint/landmark coordinates
- Edges: spatial and temporal connections
  - Spatial edges corresponds to the natural connectivity of the nodes
  - Temporal edges connect the same nodes across time frames



Spatio-Temporal GCN (ST-GCN)

The ST-GCN was proposed for the Skeleton-based human action recognition task by modeling the sequence of body poses as a spatio-temporal graph

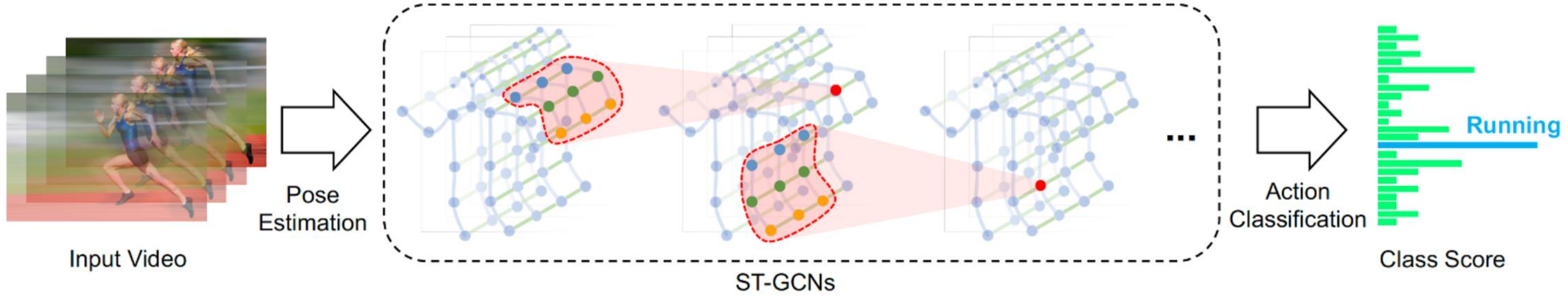
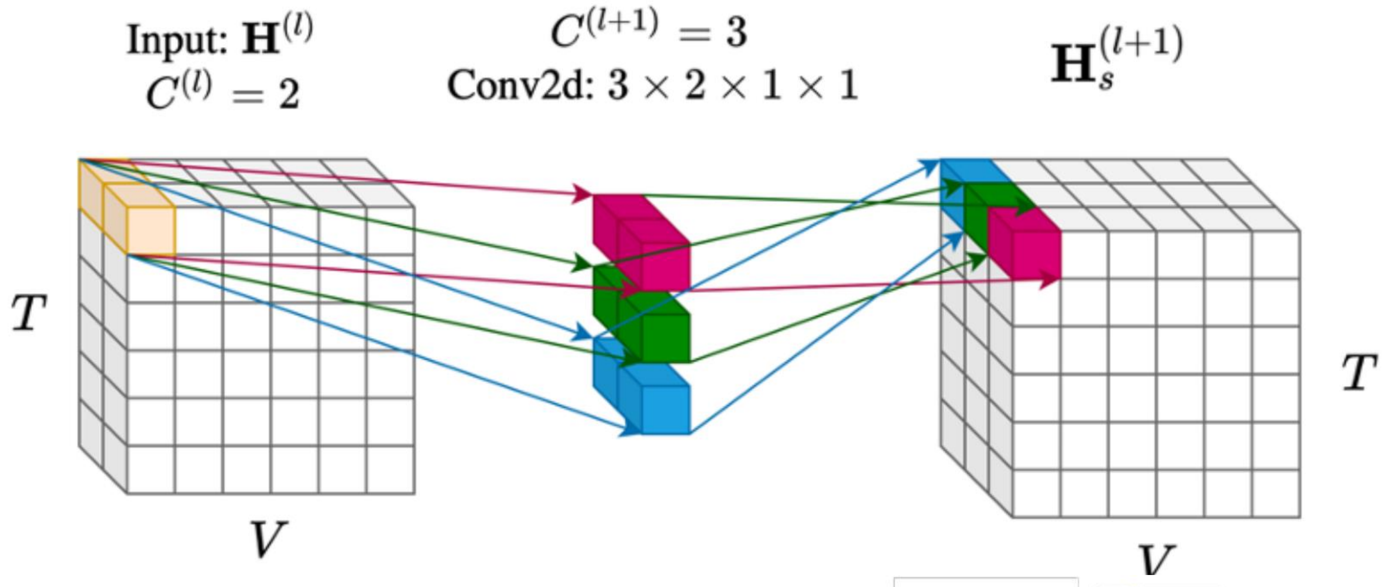


Image taken from: S. Yan, Y. Xiong and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition", AAAI 2018

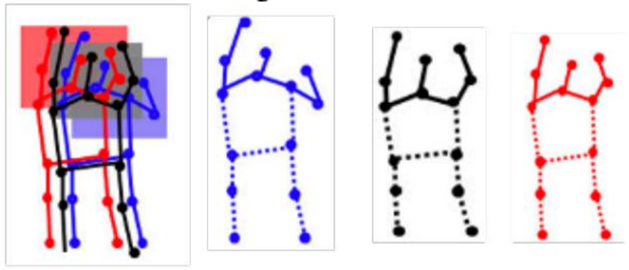
Spatio-Temporal GCN (ST-GCN)

<u>Spatial convolution:</u> Identification of patterns in each individual graph


Input: $\mathbf{H}^{(l)}$
$C^{(l)} = 2$
$C^{(l+1)} = 3$
Conv2d: $3 \times 2 \times 1 \times 1$
$\mathbf{H}_s^{(l+1)}$
T
V
T
V

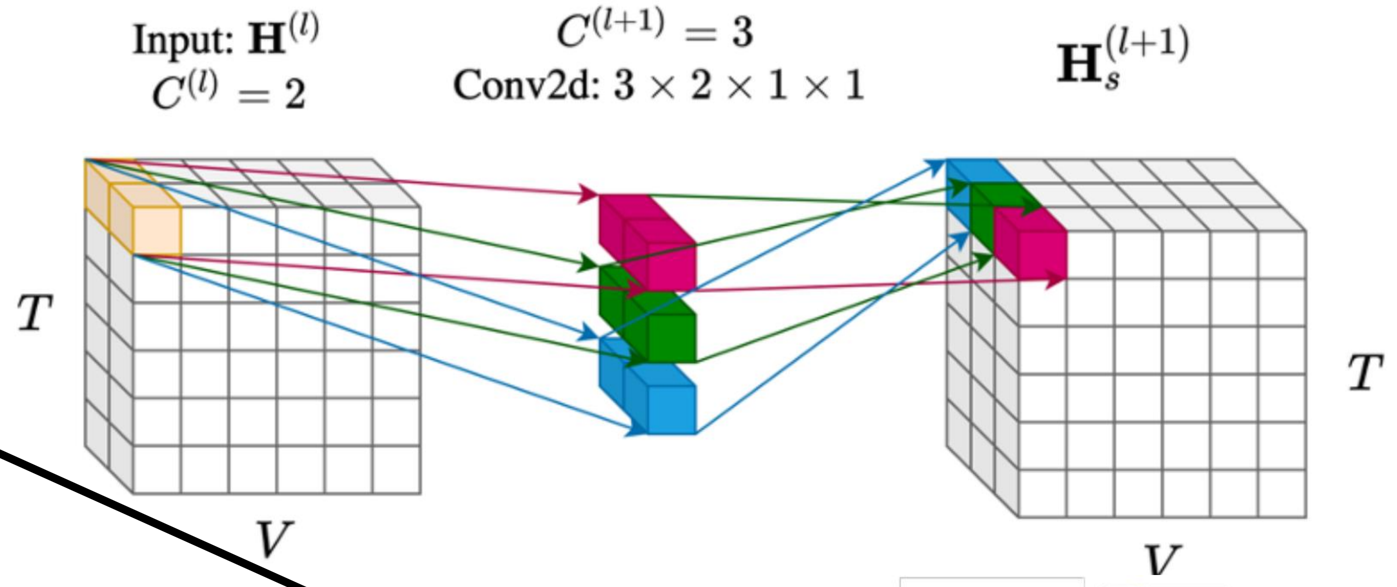
$$H_s^{(l+1)} = ReLU\left(\sum_p \left(\underbrace{\hat{A}_p \times M_p^{(l+1)}}_{\text{Spatial Attention}}\right) \underbrace{H^{(l)} W_p^{(l+1)}}_{\text{2D Convolution}}\right)$$

$$H_s^{(l+1)} \in R^{C^{(l+1)} \times T \times V}$$



Spatio-Temporal GCN (ST-GCN)

<u>Spatial convolution:</u> Identification of patterns in each individual graph

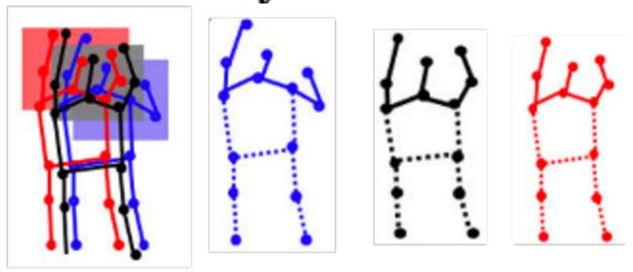


Input: $\mathbf{H}^{(l)}$
$C^{(l)} = 2$
$C^{(l+1)} = 3$
Conv2d: $3 \times 2 \times 1 \times 1$
$\mathbf{H}_s^{(l+1)}$
T
V
T
V

Learnable attention mask $\mathbf{M}_p$

$$H_s^{(l+1)} = ReLU\left(\sum_p \left(\underbrace{\hat{A}_p \times M_p^{(l+1)}}_{\text{Spatial Attention}}\right) \underbrace{H^{(l)} W_p^{(l+1)}}_{\text{2D Convolution}}\right)$$

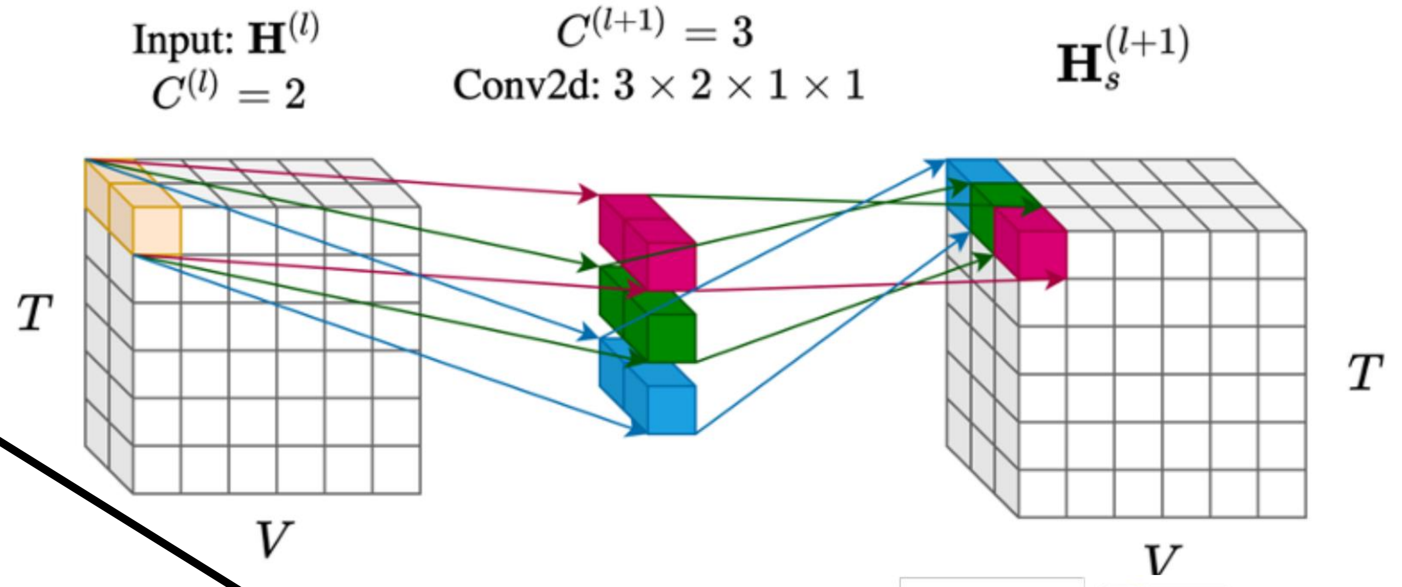
$$H_s^{(l+1)} \in R^{C^{(l+1)} \times T \times V}$$



Spatio-Temporal GCN (ST-GCN)

Spatial convolution: Identification of patterns in each individual graph

The element-wise multiplication can be replaced with summation

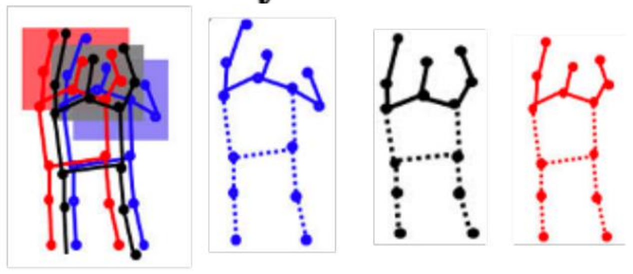

Input: $\mathbf{H}^{(l)}$
$C^{(l)} = 2$
$C^{(l+1)} = 3$
Conv2d: $3 \times 2 \times 1 \times 1$
$\mathbf{H}_s^{(l+1)}$
T
V
T
V

$$H_s^{(l+1)} = ReLU\left(\sum_p \left(\hat{A}_p \times M_p^{(l+1)}\right) H^{(l)} W_p^{(l+1)}\right)$$

Spatial Attention

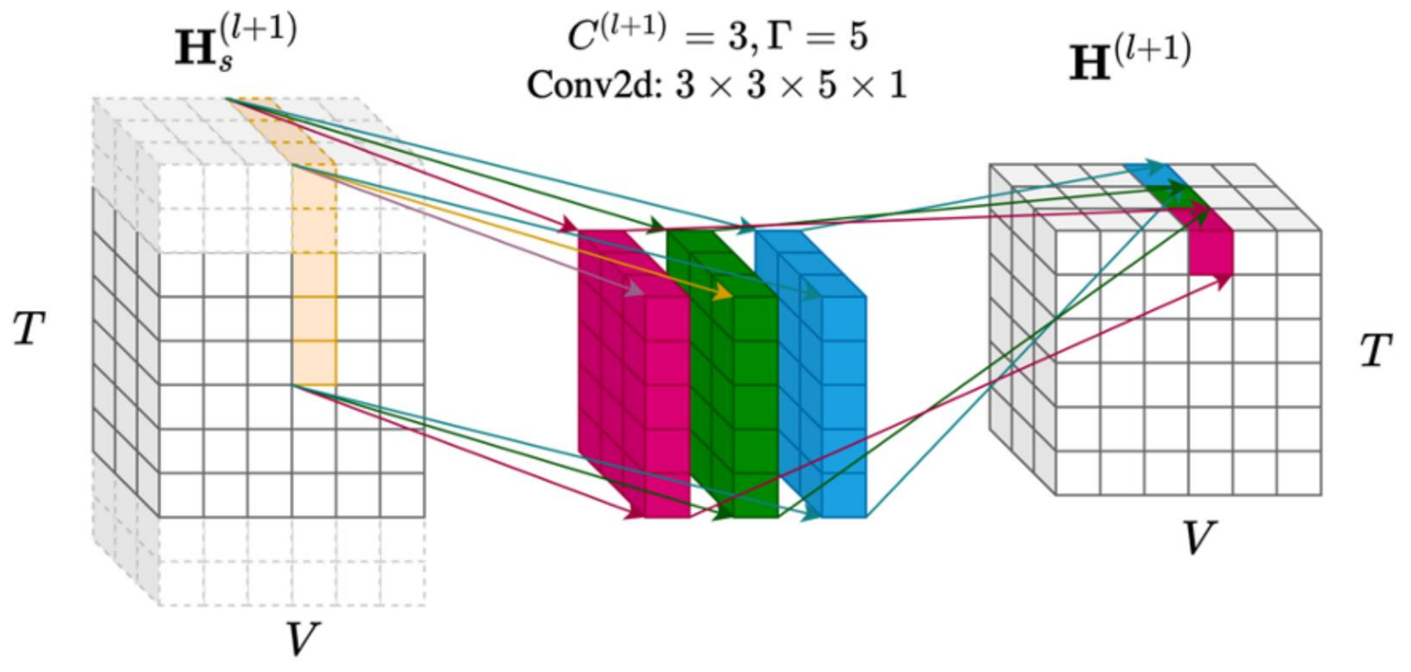
2D Convolution

$$H_s^{(l+1)} \in R^{C^{(l+1)} \times T \times V}$$



Spatio-Temporal GCN (ST-GCN)

Temporal convolution: Identification of motion patterns in temporally-adjacent graphs


$\mathbf{H}_s^{(l+1)}$
$C^{(l+1)} = 3, \Gamma = 5$
Conv2d: $3 \times 3 \times 5 \times 1$
$\mathbf{H}^{(l+1)}$
T
V
T
V

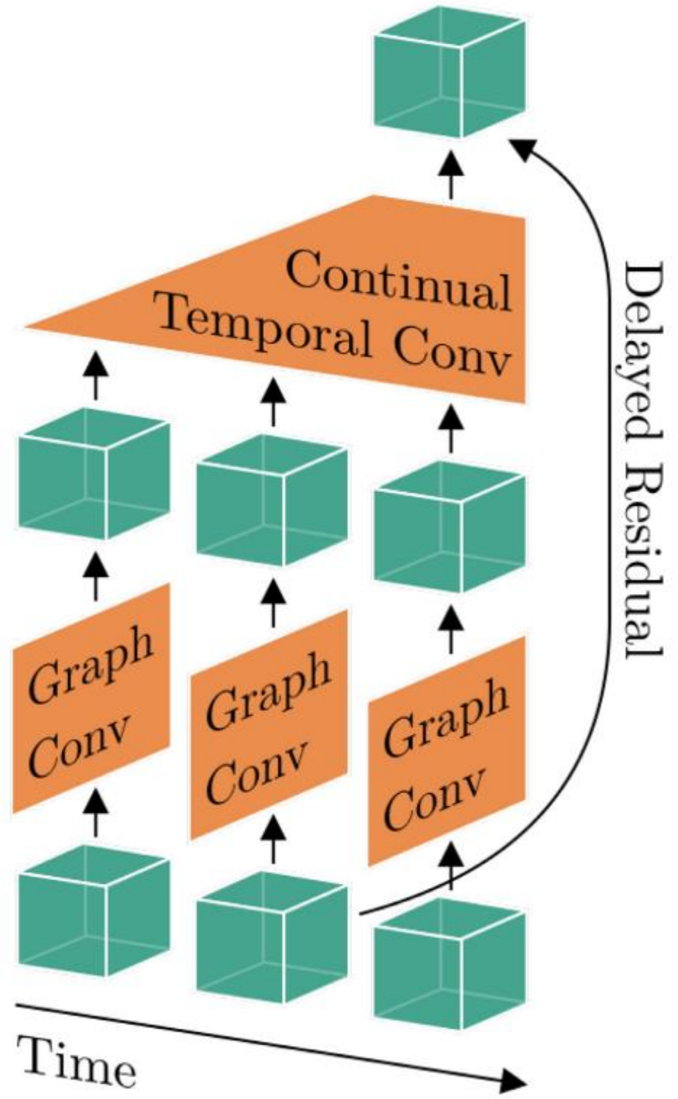
Continual Spatio-Temporal GCN (Co-STGCN)

Observation: In ST-GCNs the spatial and the temporal convolutions are decoupled

Continual operation by:

- Calculating spatial convolution for each input graph
- Perform continual temporal convolution over successive graph representations

Hedegaard, Heidari, Iosifidis, "Continual spatio-temporal graph convolutional networks", Pattern Recognition, 2023


Continual Temporal Conv
Delayed Residual
Graph Conv
Graph Conv
Graph Conv
Time