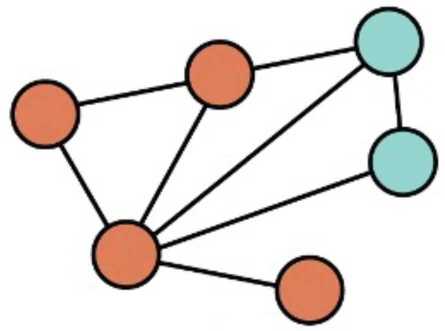
BOSTON UNIVERSITY



Graph Neural Networks

DL4DS – Spring 2025

April/May Dates

You are here

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
		April 1	2	3 Xformers Part 2	4	5
6	7	8 Industry Talk	9	10 Improving LLMs	11	12
13	14	15 GANs	16	17 VAEs	18	19
20	21	22 Diffusion Models	23	24 Graph NNs	25	26
27	28	29 **★ Project Presentations 1 ★**	30	May 1 **★ Project Presentations 2★**	2	3
4	5 Project Reports Due	6	7	8	8	10

Finals Week

Project Presentations

https://dl4ds.github.io/sp2025/miniconf.html

Starts promptly at 3:30 both days – get here early!

Format: Timing is Strict!!
≤ 4 minutes presentation, video or combo
~1 minute Q&A and changeover

April 29 – 75 minutes

3:30 - 3:34 PM : Real-Time Localized Image Enhancement via YOLOv8 and Lightweight Super Resolution Model
-- Huihao Xing, Youran Geng, Yuhao Zhang

3:35 - 3:39 PM : AI-Based Car Crash Detection Using Deep Learning
-- Yuchen Li, Shiyi Chen, Yuanchen Yin

3:40 - 3:44 PM : Enhanced Relationformer for Linear Detection
-- Chen-Yu(Erioe) Liu, Yuanhao Shen

3:45 - 3:49 PM : AI-Powered Storyboard Generator Using Prompt-Based Diffusion Models
-- Rishabh Reddy Suravaram, Amruth Devineni

3:50 - 3:54 PM : Bone fractures Graphical Chatbot
-- Caslow Chien, Serena Theobald, Sindhuja Kumar

3:55 - 3:59 PM : Legal Contract Dataset
-- Ethan Chang, Heng Chang, Josh Yip

4:00 - 4:04 PM : Smart Multimodal Classroom Video Recorder
-- Bhavya Surana, Grace Chong, William Clavier

4:05 - 4:09 PM : Machine Learning And Physics
-- Hieu Nguyen, Yiren Wang, Mi-Ru Youn, Gukai Chen

4:10 - 4:14 PM : Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning
-- Carlos Garcia, Anthony Huang, Daniel Strick

4:15 - 4:19 PM : Cricket Commentary Generation Using LLaVA
-- Bhuvan Shivalingaiah Gowda, Sumanth Hosdurg Kamath, Samritha Aadhi Ravikumar

4:20 - 4:24 PM : Deep Learning Approach on Music Recommendation System
-- Christine Sangphet, Namika Takada, Ann Liang

May 1 – 75 minutes

3:30 - 3:34 PM : Explaining Deepseek with Manim
-- Declan Young

3:35 - 3:39 PM : Enhancing AI-Generated Image Detection: A Comparative Study of CNNs, Transformers, and Contrastive Learning
-- Viktoria Zruttova, Junhui Cho, Cordell Cheng

3:40 - 3:44 PM : Reproduction of DeepSeek R1 Zero: A case study in Math Task
-- Ziqi Tang, Mingyu Chen

3:45 - 3:49 PM : Fine-Grained Visual Classification of Bird Species
-- Kunshu Yang, Shiheng Xu, Renjie Fan

3:50 - 3:54 PM : Efficient Open-Vocabulary Models for Low-Power Computer Vision
-- Hsiang Yu Huang, Zainab Alhaddad, Winni Tai

3:55 - 3:59 PM : EfficientMonocular Depth Estimation
-- Neel Gangrade, Rachita Singh

4:00 - 4:04 PM : Efficient Computer Vision: Pushing the Frontier of Edge Computing
-- Zachary Gentile, Michael Krah, Alex Lavaee

4:05 - 4:09 PM : Predicting Alzheimer's Disease using Structural MRIs: Activation Map Analysis of Memory Related Brain Regions
-- Rajdeep Singh, John Salloum, Atul Aravind Das

4:10 - 4:14 PM : ProtMotifGen: Protein Motif Generation
-- Sicheng Yi

4:15 - 4:19 PM : ChainThought: Enhancing Reasoning Capabilities in Language Models Through Step-by-Step Problem Solving
-- Eric Gulotty, Brandon Wong, Derek Laboy

4:20 - 4:24 PM : Deep Learning for Autism Screening Using Transfer Learning on Video Data
-- Zhamshidbek Abdulkhamidov, Zhansaya Taszhanova, Saya Atchibay

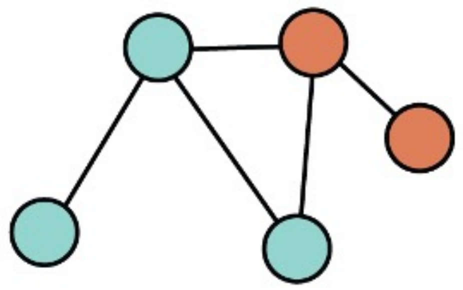
4:25 - 4:29 PM : Kilter Board Beta and Problem Generation
-- Kailen Richards

Project Video Submissions

- Post on YouTube
- Upload to this Google Drive folder
- Send me link at least day before your presentation if you want me to play it

Google Drive

Graph Neural Networks



Neural architectures that process graphs.

Three challenges:

1. Variable topology
2. Size (billions of nodes)
3. Single monolithic graph

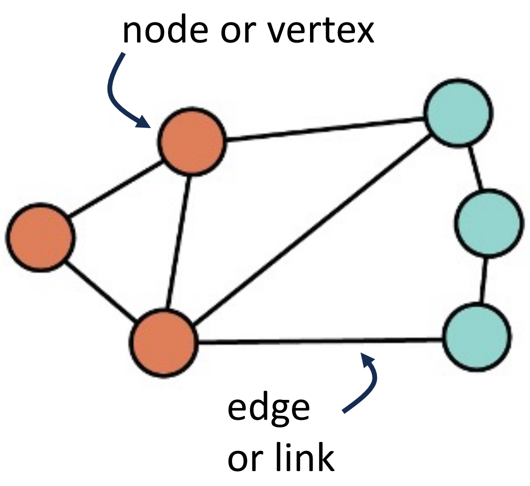
Topics

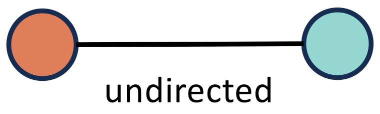
Topics

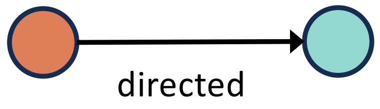
- Basic definition and examples
- Graph representation
- Properties of Adjacency Matrix
- Graph neural network, tasks and loss functions
- Graph convolutional network
- Graph & Node classification
- Edge graphs

Graph (Network)

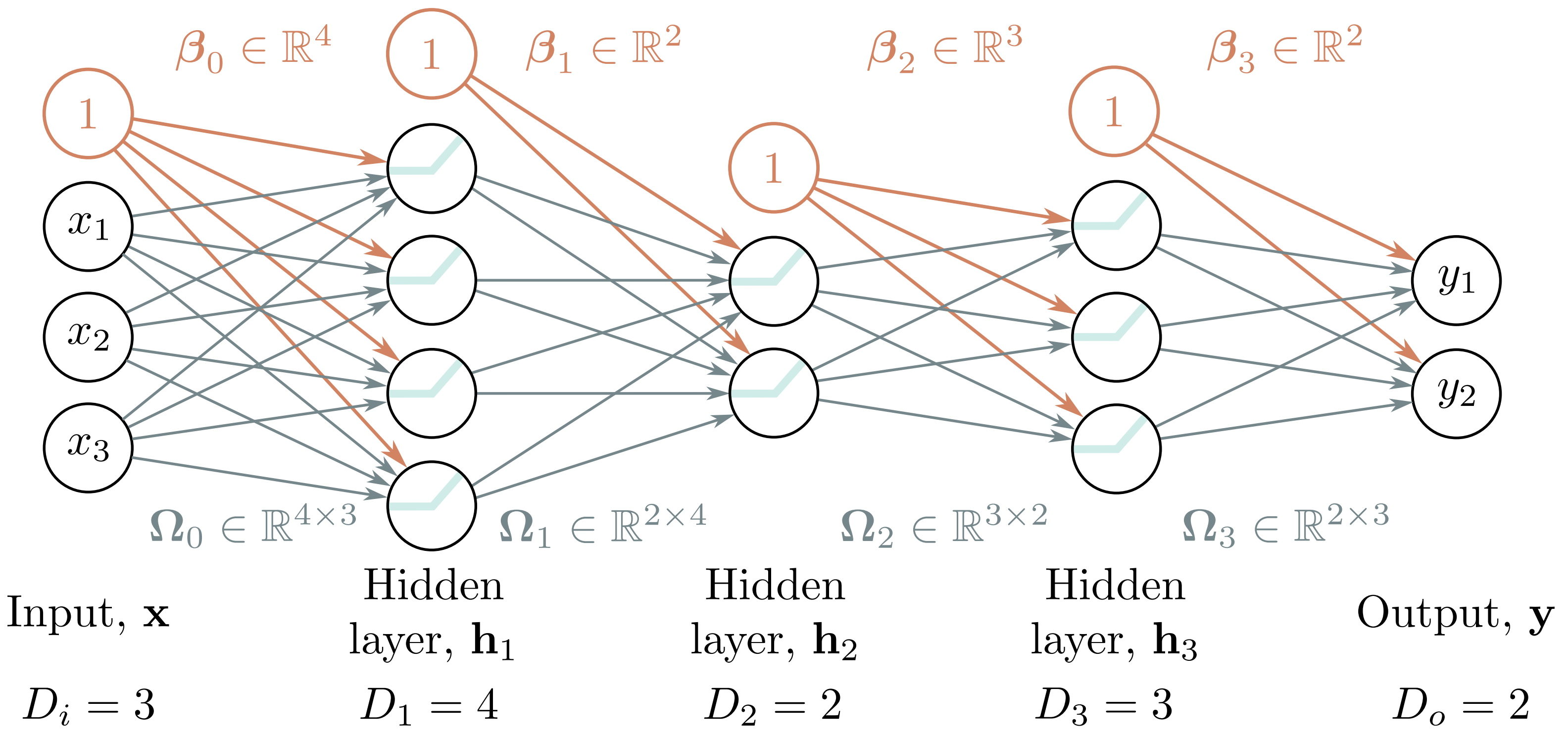
- general structure composed of *nodes* (vertices) and *edges* (links)
- edges can be *undirected* or *directed*
- a graph with directed edges and no cycles (no loops) is called *directed acyclic graph* (DAG)





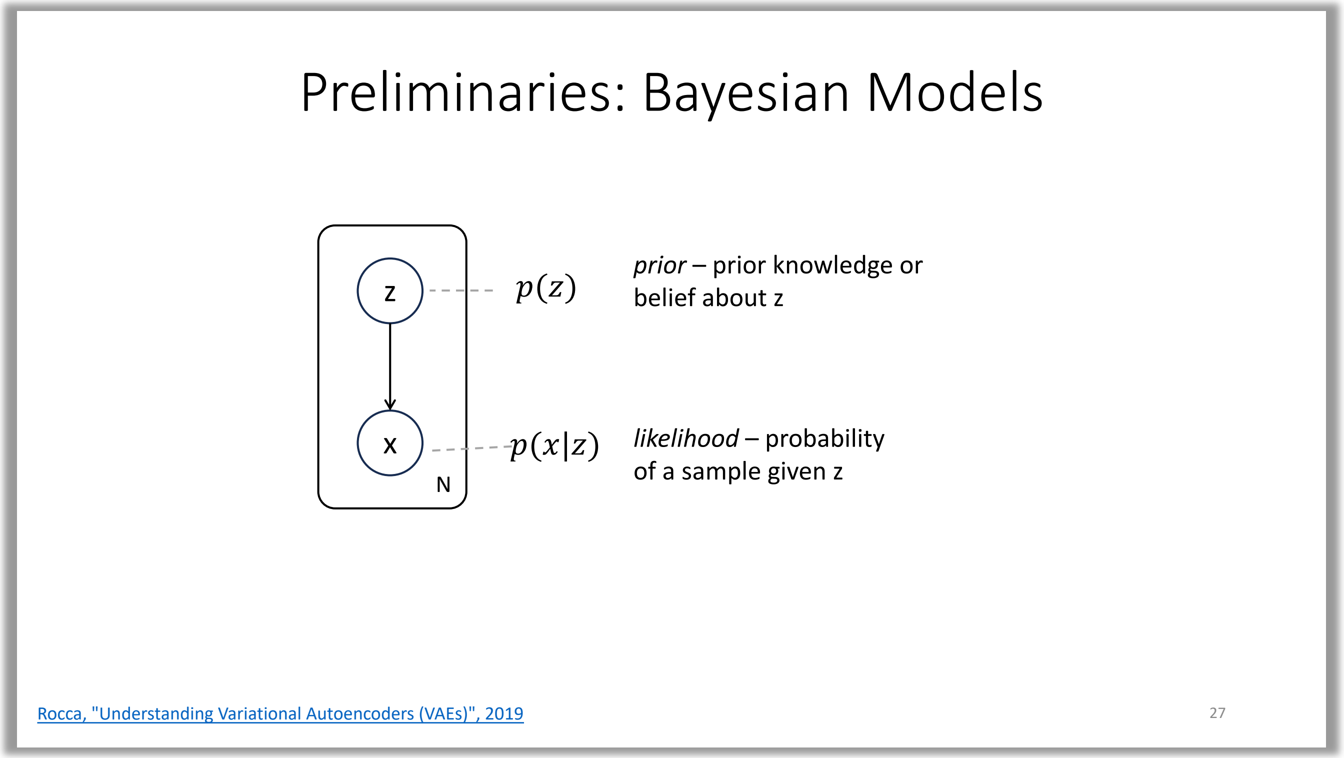


Directed Example – Feed Forward Network

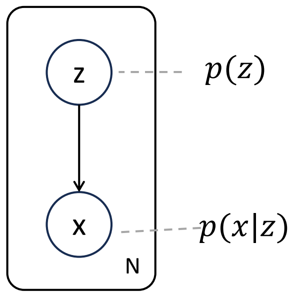


$\boldsymbol{\beta}_0 \in \mathbb{R}^4$
$\boldsymbol{\beta}_1 \in \mathbb{R}^2$
$\boldsymbol{\beta}_2 \in \mathbb{R}^3$
$\boldsymbol{\beta}_3 \in \mathbb{R}^2$
1
1
1
1
x_1
x_2
x_3
y_1
y_2
$\boldsymbol{\Omega}_0 \in \mathbb{R}^{4\times 3}$
$\boldsymbol{\Omega}_1 \in \mathbb{R}^{2\times 4}$
$\boldsymbol{\Omega}_2 \in \mathbb{R}^{3\times 2}$
$\boldsymbol{\Omega}_3 \in \mathbb{R}^{2\times 3}$
Input, $\mathbf{x}$
Hidden layer, $\mathbf{h}_1$
Hidden layer, $\mathbf{h}_2$
Hidden layer, $\mathbf{h}_3$
Output, $\mathbf{y}$
$D_i = 3$
$D_1 = 4$
$D_2 = 2$
$D_3 = 3$
$D_o = 2$

Directed Example – Bayesian Graphical Model



Preliminaries: Bayesian Models



prior – prior knowledge or belief about z

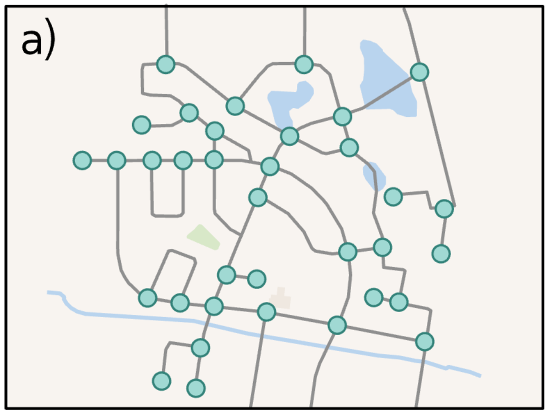
likelihood – probability of a sample given z

Rocca, "Understanding Variational Autoencoders (VAEs)", 2019

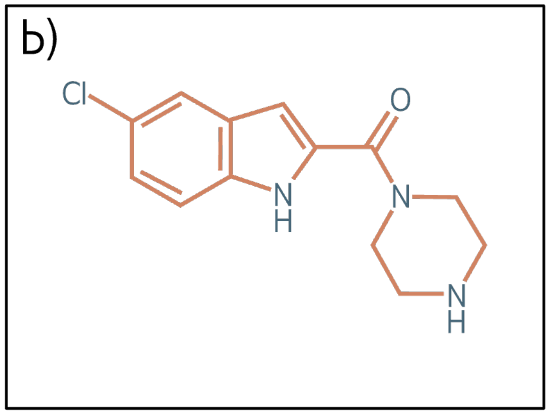
27

From lecture 18 – Variational Autoencoders

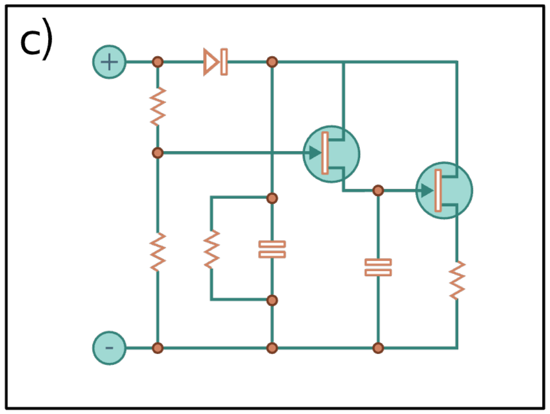
Undirected Examples

a)


road networks
nodes: physical locations or landmarks
edges: connecting roads

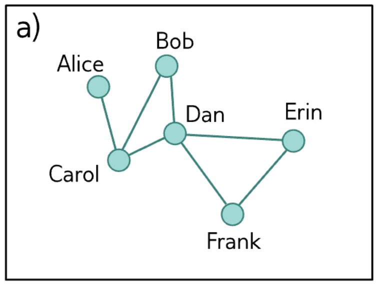
b)


chemical molecules
nodes: atoms
edges: chemical bonds

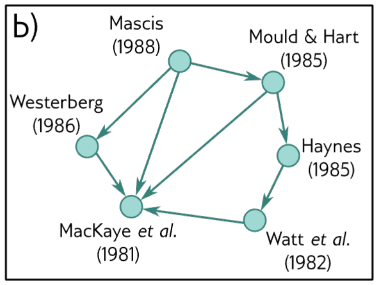
c)


electrical circuits
nodes: components or junctions
edges: wires/electrical connections

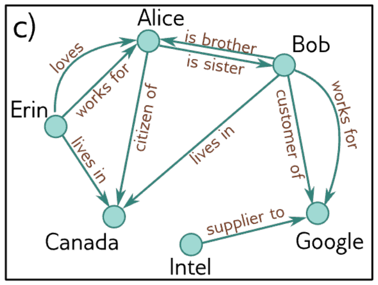
Examples



social networks
nodes: people
edges: friendships
(undirected)

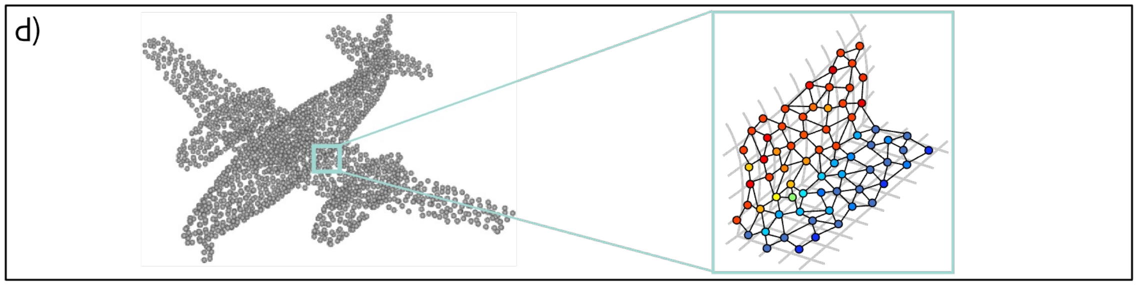


science literature
nodes: papers
edges: citations
(acyclic directed)



knowledge graph
nodes: objects
edges: named relationship
(cyclic directed)

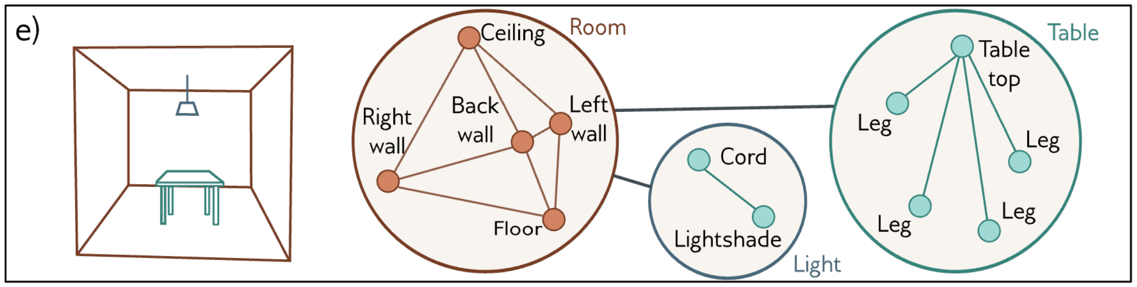
Example – Geometric Point Cloud



nodes: positions in 3D space (vertex in 3D graphics)
edges: connections to nearby points
(undirected)

https://en.wikipedia.org/wiki/Vertex_(computer_graphics)

Example – Scene Graph



hierarchical graph showing relationship between objects in a 3D scene

nodes: composite graphs or objects in 3D space
edges: connections to nearby points
(undirected)

Fernandez-Madrigal and Gonzalez, "Multi-hierarchical graph search," 2002
Armeni et al, “3D Scene Graph: A structure for unified semantics, 3D space and camera,” 2020?
Wald et al, “Learning 3D Semantic Scene Graphs with Instance Embeddings,” 2022

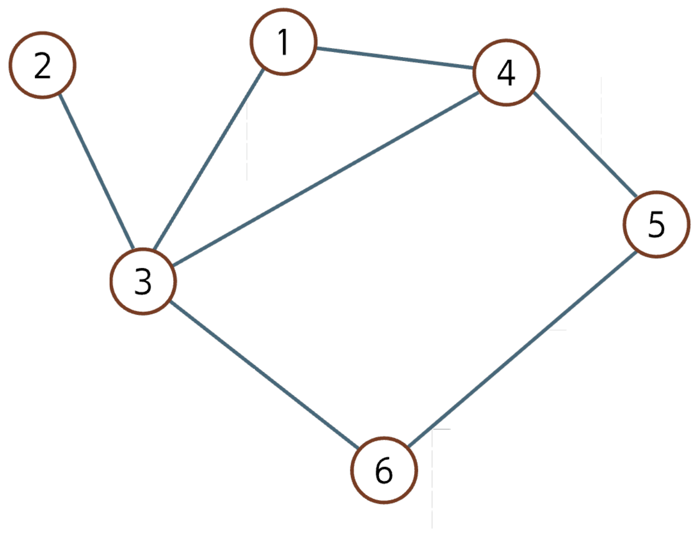
Other examples

- Wikipedia – nodes are articles, edges are hyperlinks between articles
- Computer programs – nodes are syntax tokens, edges are computation between tokens (tensor graph from Gradients lecture)
- Protein interactions – nodes are proteins, edges exist where two proteins interface
- Set or list – every element is connected to every other element
- image – each pixel is a node with edges to the eight adjacent pixels

Topics

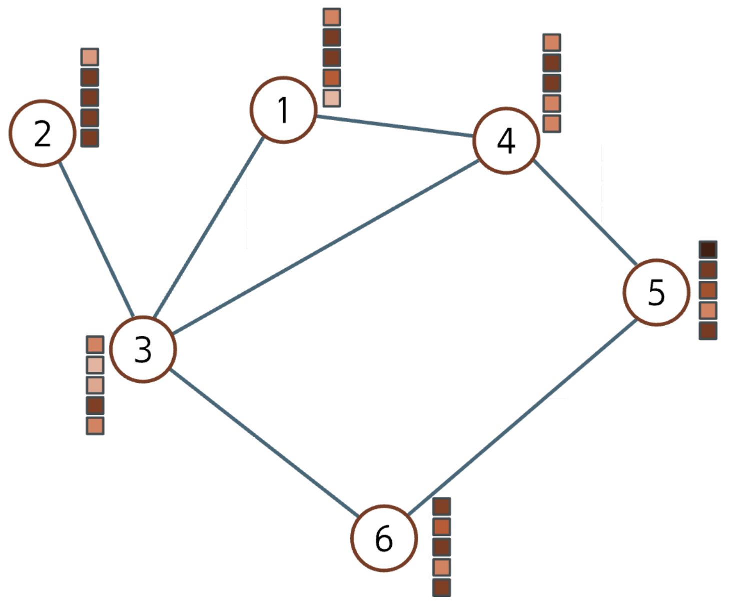
- Basic definition and examples
- Graph representation
- Properties of Adjacency Matrix
- Graph neural network, tasks and loss functions
- Graph convolutional network
- Graph & Node classification
- Edge graphs

Graph representation



Example undirected graph with 6 nodes

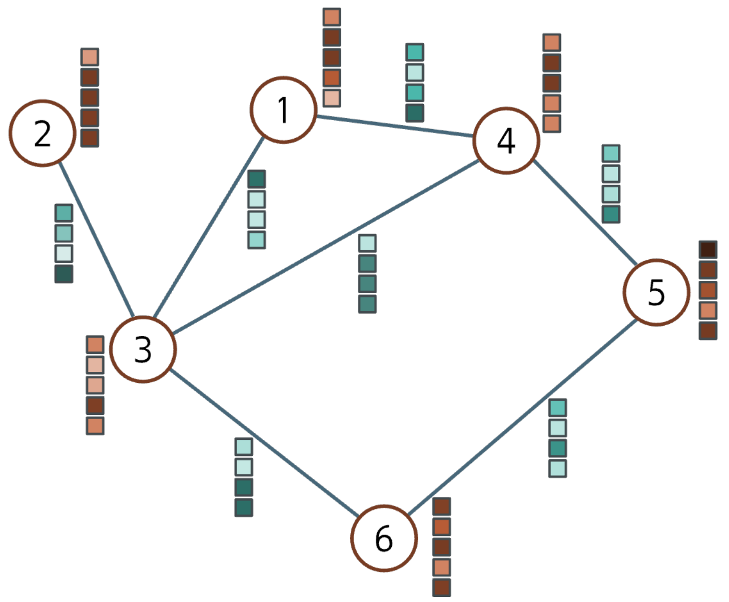
Graph representation – node embedding



Example undirected graph with 6 nodes

Information about a node is stored in a *node embedding*

Graph representation – edge embedding

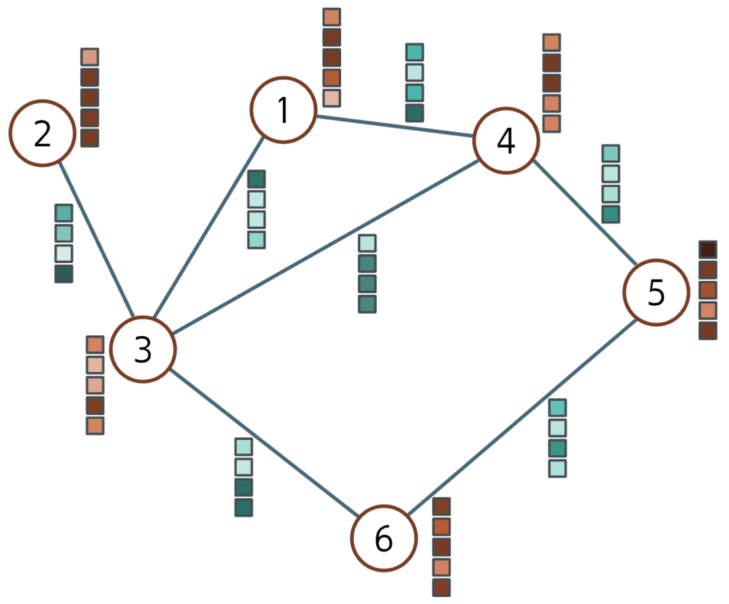


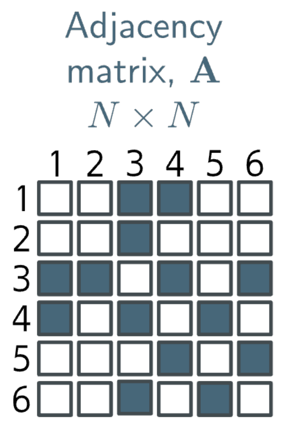
Example undirected graph with 6 nodes

Information about a node is stored in a *node embedding*

Information about an edge is stored in an *edge embedding*

Graph representation – adjacency matrix




Adjacency matrix, $\mathbf{A}$
$N \times N$

	1	2	3	4	5	6
1	□	□	■	■	□	□
2	□	□	■	□	□	□
3	■	■	□	■	□	■
4	■	□	■	□	■	□
5	□	□	□	■	□	■
6	□	□	■	□	■	□

Assume we have N nodes

The graph connections can be represented by an *adjacency matrix*

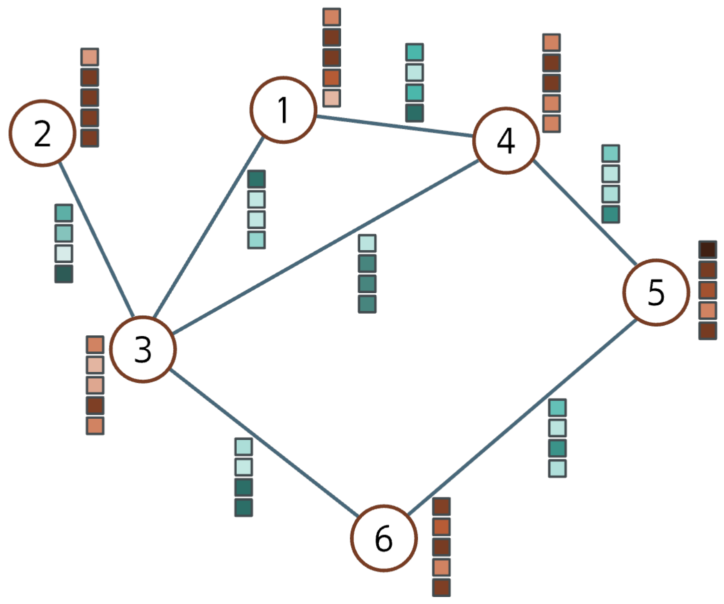
Where a value of 1 at (m, n) represents a connection between nodes m and n.

For undirected graphs the matrix is always symmetric about the diagonal

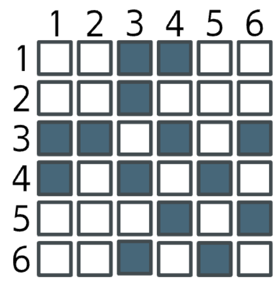
Diagonal is zero – no edge to itself

Can be very sparse

Graph representation – node data matrix

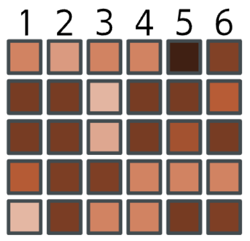


Adjacency matrix, $\mathbf{A}$
$N \times N$



	1	2	3	4	5	6
1	□	□	■	■	□	□
2	□	□	■	□	□	□
3	■	■	□	■	□	■
4	■	□	■	□	■	□
5	□	□	□	■	□	■
6	□	□	■	□	■	□

Node data, $\mathbf{X}$
$D \times N$

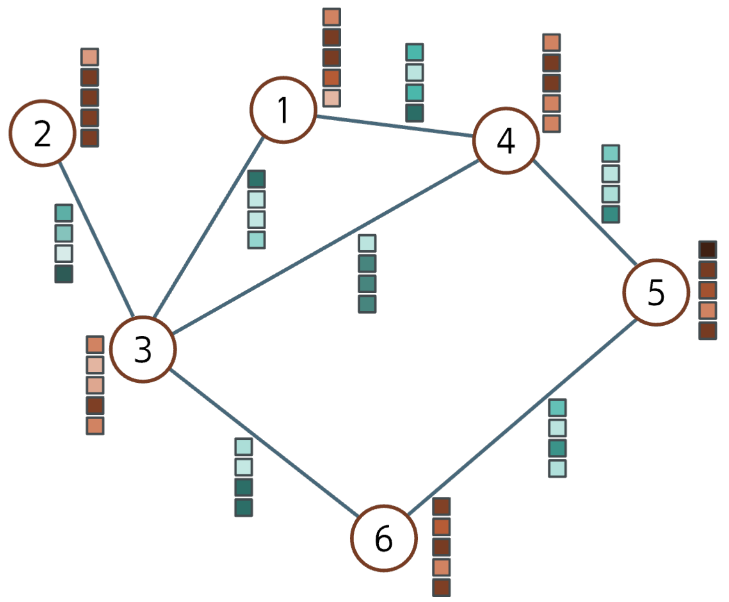


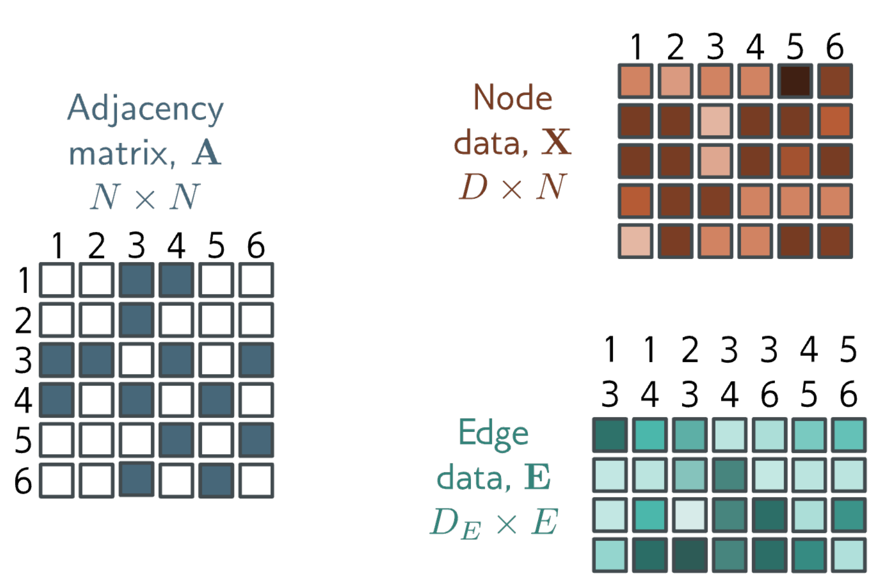
All the node data in the form of node embeddings can represented by a *Node data matrix*

Where D is the dimension of the note embedding and

N is the number of nodes

Graph representation – edge data matrix



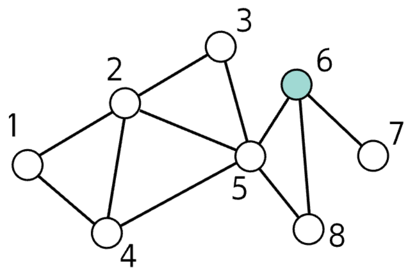


Similarly, all the edge embedding information can be stored in an *Edge data matrix*, where:
D_E is the dimension of the edge embedding vector and
E is the number of edges

Topics

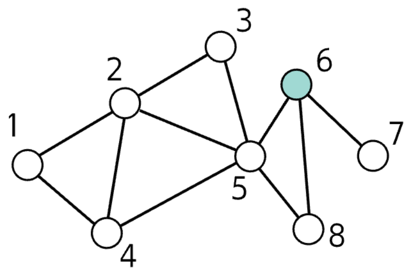
- Basic definition and examples
- Graph representation
- Properties of Adjacency Matrix
- Graph neural network, tasks and loss functions
- Graph convolutional network
- Graph & Node classification
- Edge graphs

Adjacency Matrix



Assume we have an 8-node undirected graph

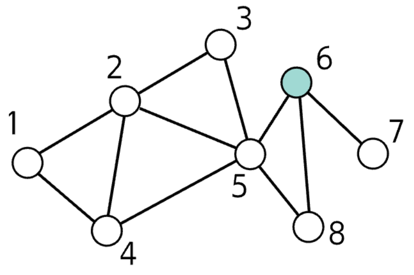
Adjacency Matrix



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

Adjacency matrix for this graph.

Adjacency Matrix



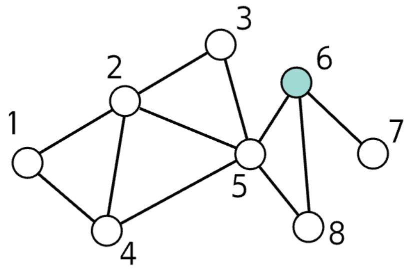
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

adjacency matrix

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

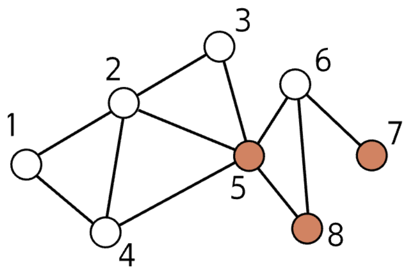
We can one hot encode representation of node 6

Adjacency Matrix



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

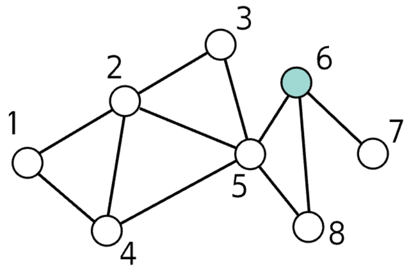
$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



If we pre-multiply the one-hot encoded data node vector x by adjacency matrix A we get the 6th column of A indicating direct connections to other nodes

One-hot encoding vector of all nodes directly connected node 6

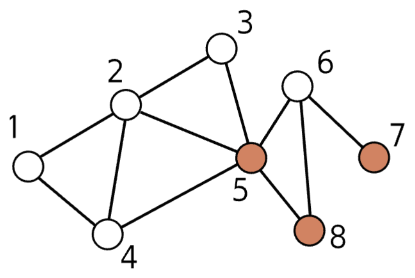
Adjacency Matrix



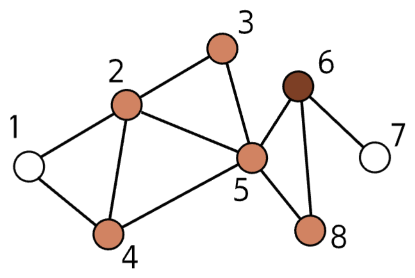
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

If we pre-multiply again by A, we get a vector showing the number of times we can get to each node in 2 steps.

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



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

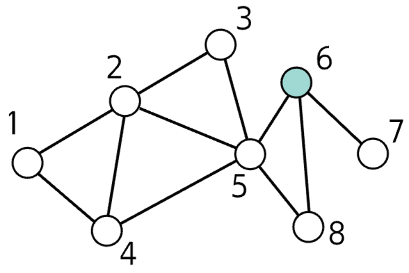


Graph showing all nodes that can be reached in *exactly* 2 steps.

Adjacency Matrix

Pre-multiplying x by A twice is equivalent to the matrix A^2

Shows how many times you can get from node m to node n in 2 steps

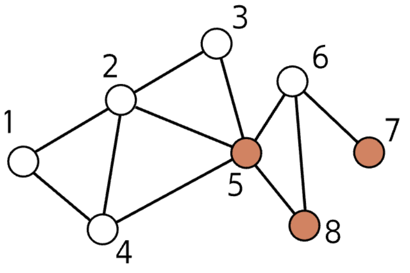


$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

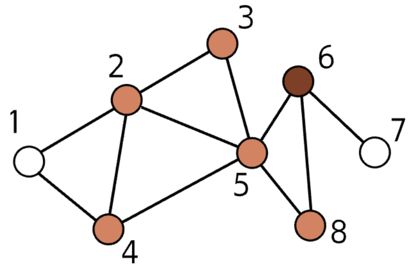
$$\boxed{\mathbf{A}^2 = \begin{bmatrix} 2 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 2 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 & 0 & 1 \\ 1 & 2 & 2 & 3 & 1 & 1 & 0 & 1 \\ 2 & 2 & 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2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

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

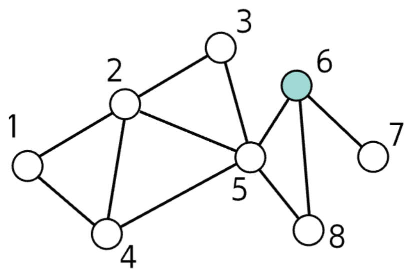
$$\mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



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



Adjacency Matrix



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

Example for $L = 2$

When you raise the adjacency matrix to the power of L (pre-multiply L-1 times),

the entry at position (m, n) of $\mathbf{A}^L$ contains the number of unique walks of length L from node n to node m

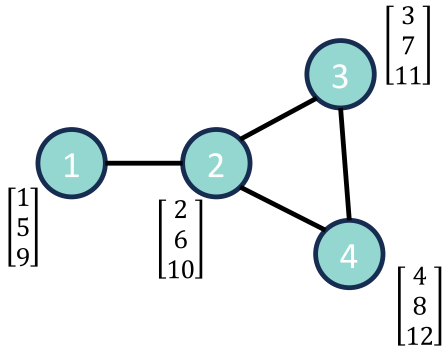
Note: this is not the same as the number of unique paths since it includes routes that visit the same node more than once.

a non-zero entry at position (m, n) indicates that the distance from m to n must be less than or equal to L.

See Notebook 13.1 – Encoding Graphs

Permutation of node indices

Since node indexing is arbitrary, we can permute the node indices


1
2
3
4
$\begin{bmatrix}1\\5\\9\end{bmatrix}$
$\begin{bmatrix}2\\6\\10\end{bmatrix}$
$\begin{bmatrix}3\\7\\11\end{bmatrix}$
$\begin{bmatrix}4\\8\\12\end{bmatrix}$

$$\mathbf{X} = \begin{array}{c} \begin{matrix}(1 & 2 & 3 & 4)\end{matrix} \\ \begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\\9 & 10 & 11 & 12\end{bmatrix}\end{array}$$

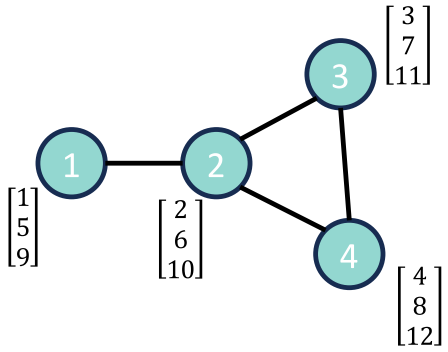
node data

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

adjacency matrix

Permutation of node indices

Since node indexing is arbitrary, we can permute the node indices



1
2
3
4
$\begin{bmatrix}1\\5\\9\end{bmatrix}$
$\begin{bmatrix}2\\6\\10\end{bmatrix}$
$\begin{bmatrix}3\\7\\11\end{bmatrix}$
$\begin{bmatrix}4\\8\\12\end{bmatrix}$

$$\mathbf{X} = \begin{array}{c} \begin{matrix}(1 & 2 & 3 & 4)\end{matrix} \\ \begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\\9 & 10 & 11 & 12\end{bmatrix}\end{array}$$

node data

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

adjacency matrix

$$\mathbf{P} = \begin{array}{c} \begin{matrix}(3 & 4 & 2 & 1)\end{matrix} \\ \begin{bmatrix}0 & 0 & 0 & 1\\0 & 0 & 1 & 0\\1 & 0 & 0 & 0\\0 & 1 & 0 & 0\end{bmatrix}\end{array}$$

We can express this mathematically with a permutation matrix, $\mathbf{P}$

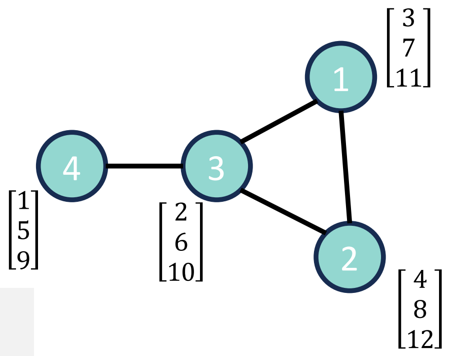
New: $(1 \quad 2 \quad 3 \quad 4)$
Old: $(3 \quad 4 \quad 2 \quad 1)$

$$\mathbf{X}' = \mathbf{XP} = \begin{bmatrix}3 & 4 & 2 & 1\\7 & 8 & 6 & 5\\11 & 12 & 10 & 9\end{bmatrix}$$

Permute the columns of the Node data matrix

$$\mathbf{A}' = \mathbf{P}^{\mathrm{T}}\mathbf{AP} = \begin{bmatrix}0 & 1 & 1 & 0\\1 & 0 & 1 & 0\\1 & 1 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

Permute both the rows and column of the Adjacency matrix



1
2
3
4
$\begin{bmatrix}3\\7\\11\end{bmatrix}$
$\begin{bmatrix}1\\5\\9\end{bmatrix}$
$\begin{bmatrix}2\\6\\10\end{bmatrix}$
$\begin{bmatrix}4\\8\\12\end{bmatrix}$

Topics

Graph Neural Network

- A graph neural network is a model that takes the node embeddings $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$ as inputs and passes them through a series of K layers.
- The node embeddings are updated at each layer to create intermediate “hidden” representations $\mathbf{H}_K$ before finally computing output embeddings $\mathbf{H}_K$.
- At the start of this network, each column of the input node embeddings $\mathbf{X}$ just contains information about the node itself.
- At the end, each column of the model output $\mathbf{H}_K$ includes information about the node and its context within the graph.
- This is like word embeddings passing through a transformer network. These represent words at the start but represent the word meanings in the context of the sentence at the end.

Graph Level Tasks

Determine

- class categories, e.g. molecule is poisonous
- regression values, e.g. molecure boiling and freezing point

based on graph structure and node embeddings

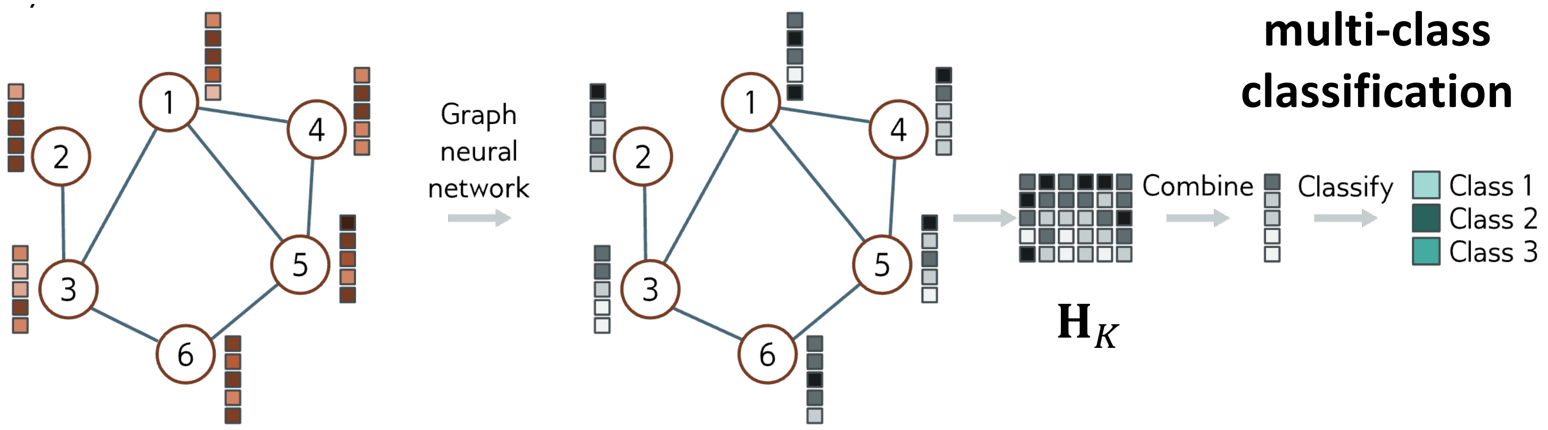
For graph-level tasks, the output node embeddings are combined (e.g., by averaging), and the resulting vector is mapped via a linear transformation or neural network to a fixed-size vector

Typical Three Types of Models

- Graph level regression & classification
- Node level regression & classification
- Edge prediction

Look at prediction heads first.

Graph level regression & classification



Last layer (Regression): $\Pr(y|\mathbf{X},\mathbf{A}) = \beta_K + \omega_K \mathbf{H}_K \mathbf{1}\ /N$

Last layer (Classification): $\Pr(y=1|\mathbf{X},\mathbf{A}) = \text{sigmoid}[\beta_K + \omega_K \underbrace{\mathbf{H}_K \mathbf{1}\ /N}_{\text{Mean pooling}}]$

β_K is scalar

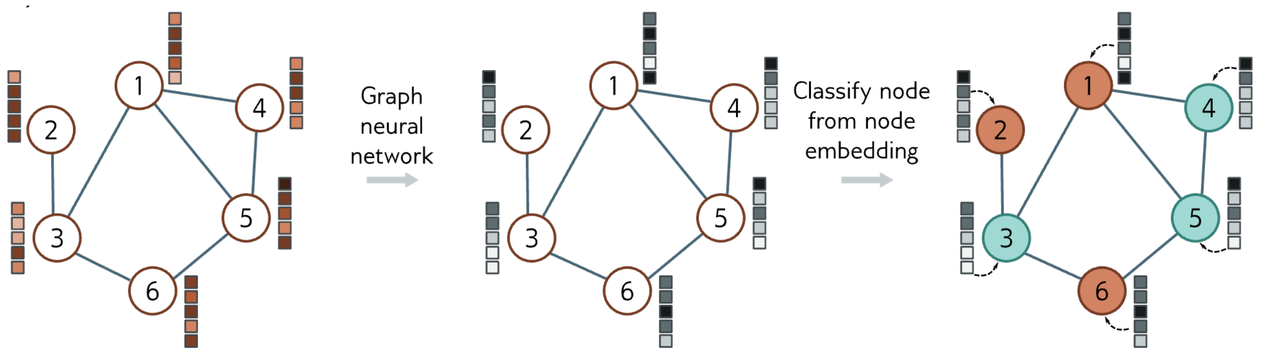
ω_K is $1\times D$ row vector

$\mathbf{H}_K$ is the $D\times N$ output embedding matrix

$\mathbf{1}$ is an $N\times 1$ column vector of 1s

Regression Loss Function: Least Squares Loss
Classification Loss Function: (Binary) Cross Entropy

Node level binary regression & classification



Last layer (Regression): $\Pr\left(y^{(n)} \middle| \mathbf{X}, \mathbf{A}\right) = \beta_K + \omega_K \mathbf{h}_K^{(n)}$

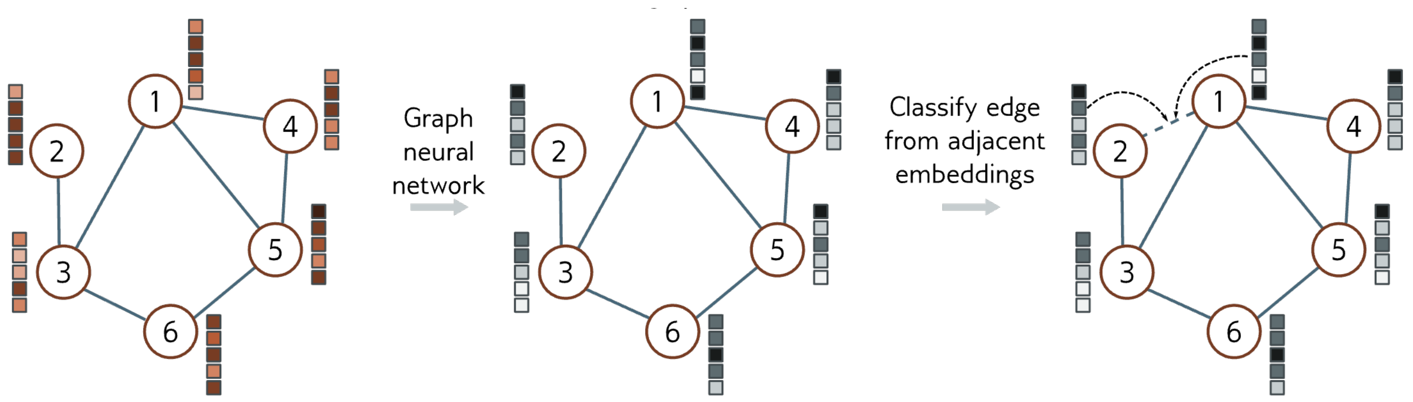
Last layer (Classification): $\Pr\left(y^{(n)} = 1 \middle| \mathbf{X}, \mathbf{A}\right) = \text{sigmoid}[\beta_K + \omega_K \mathbf{h}_K^{(n)}]$

$\mathbf{h}_K^{(n)}$ is the $D \times 1$ output embedding vector node for n

Regression Loss Function: Least Squares Loss
Classification Loss Function: (Binary) Cross Entropy

Edge prediction (classification)

Predict whether edge should exist or not.



Last layer: $\Pr\left(y^{(mn)} = 1 \middle| \mathbf{X}, \mathbf{A}\right) = \text{sigmoid}[\mathbf{h}_K^{(m)T}\mathbf{h}_K^{(n)}]$

$$[1\times D][D\times 1]$$

Classification Loss Function: Binary Cross Entropy

Topics

- Basic definition and examples
- Graph representation
- Properties of Adjacency Matrix
- Graph neural network, tasks and loss functions
- Graph convolutional network
- Graph & Node classification
- Edge graphs

Graph convolutional network

These models are convolutional in that they update each node by aggregating information from nearby nodes.

As such, they induce a relational inductive bias (i.e., a bias toward prioritizing information from neighbors).

$$\begin{array}{rcl}
\mathbf{H}_1 & = & \mathbf{F}[\mathbf{X}, \mathbf{A}, \boldsymbol{\phi}_0] \\
\mathbf{H}_2 & = & \mathbf{F}[\mathbf{H}_1, \mathbf{A}, \boldsymbol{\phi}_1] \\
\mathbf{H}_3 & = & \mathbf{F}[\mathbf{H}_2, \mathbf{A}, \boldsymbol{\phi}_2] \\
\vdots & = & \vdots \\
\mathbf{H}_K & = & \mathbf{F}[\mathbf{H}_{K-1}, \mathbf{A}, \boldsymbol{\phi}_{K-1}],
\end{array}$$

A function $F[\cdot]$ with parameters ϕ_i that takes the node embeddings and adjacency matrix and outputs new node embeddings

Equivariance and Invariance

Every layer should be *equivariant* to index permutations

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

And for node classification and edge prediction the output should be *invariant* to index permutations

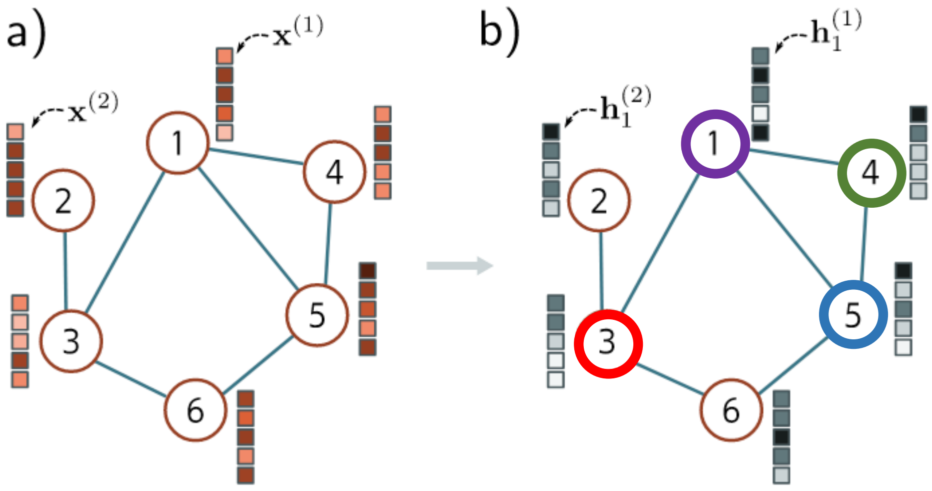
$$y = \text{sigmoid}[\beta_K + \omega_K \mathbf{H}_K \mathbf{1}/N\ ] = \text{sigmoid}[\beta_K + \omega_K \mathbf{H}_K \mathbf{P}\mathbf{1}/N\ ]$$

Example Graph Convolution Network (GCN) layer

At each node n in layer k, aggregate information from neighboring nodes

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

where $\text{ne}[n]$ returns the set of indices of the neighbors of node n.



$$\text{ne}[1] = \{4, 5, 3\}$$

$$\text{agg}[n=1, k=1] = \mathbf{h}_1^{(4)} + \mathbf{h}_1^{(5)} + \mathbf{h}_1^{(3)}$$

Example Graph Convolution Network (GCN) layer

At each node n in layer k, aggregate information from neighboring nodes

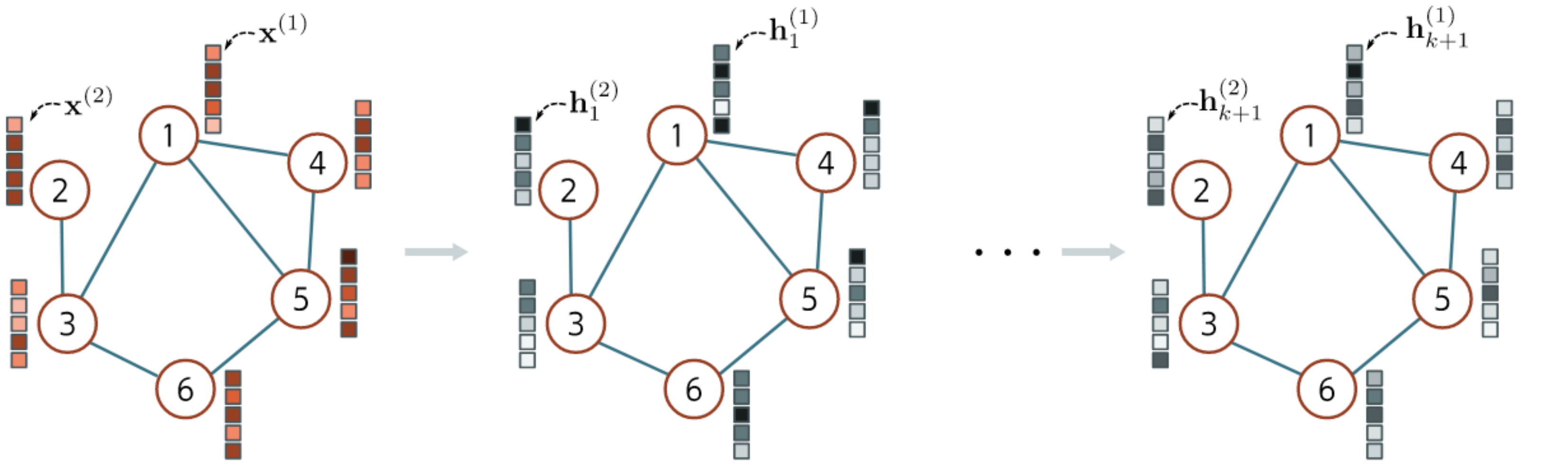
$$\mathrm{agg}[n,k] = \sum_{m\in \mathrm{ne}[n]} \mathbf{h}_k^{(m)}$$

where $\mathrm{ne}[n]$ returns the set of indices of the neighbors of node n.

Then a linear transform to the current node vector and the aggregate for the current node and add a bias.

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

Graph convolution layers



$$\mathbf{h}_1^{(n)} = \mathbf{a}\left[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}_1^{(n)} + \boldsymbol{\Omega}_0\mathbf{agg}[n]\right]$$

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

Input

1st Layer

k+1st Layer

Example Graph Convolution Network (GCN) layer

We apply the following equation

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

to the entire node hidden layers matrix, $\mathbf{H}_k$, by noting that $\mathbf{H}_k\mathbf{A}$ produces a matrix where the n^{th} column is $\mathrm{agg}[n, k]$.

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

Example Graph Convolution Network (GCN) layer

We apply the following equation

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

to the entire node hidden layers matrix, $\mathbf{H}_k$, by noting that $\mathbf{H}_k\mathbf{A}$ produces a matrix where the n^{th} column is agg$[n, k]$.

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

Note that this is (1) equivariant to permutations, (2) handles arbitrary number of neighbors, (3) exploits graph structure and (4) share parameters

Topics

- Basic definition and examples
- Graph representation
- Properties of Adjacency Matrix
- Graph neural network, tasks and loss functions
- Graph convolutional network
- Graph & Node classification
- Edge graphs

Graph classification example

We can put it all together and add a sigmoid layer

$$
\begin{aligned}
\mathbf{H}_1 &= \mathbf{a}\left[\boldsymbol{\beta}_0 \mathbf{1}^T + \boldsymbol{\Omega}_0 \mathbf{X}(\mathbf{A}+\mathbf{I})\right] \\
\mathbf{H}_2 &= \mathbf{a}\left[\boldsymbol{\beta}_1 \mathbf{1}^T + \boldsymbol{\Omega}_1 \mathbf{H}_1(\mathbf{A}+\mathbf{I})\right] \\
\vdots &= \vdots \\
\mathbf{H}_K &= \mathbf{a}\left[\boldsymbol{\beta}_{K-1} \mathbf{1}^T + \boldsymbol{\Omega}_{K-1} \mathbf{H}_{k-1}(\mathbf{A}+\mathbf{I})\right] \\
\mathrm{f}[\mathbf{X}, \mathbf{A}, \boldsymbol{\Phi}] &= \mathrm{sig}\left[\beta_K + \boldsymbol{\omega}_K \mathbf{H}_K \underbrace{\mathbf{1}/N}_{\text{Mean pooling}}\right],
\end{aligned}
$$

For classification on molecules,

$X \in \mathbb{R}^{118\times N}$: one hot encoding of 118 elements

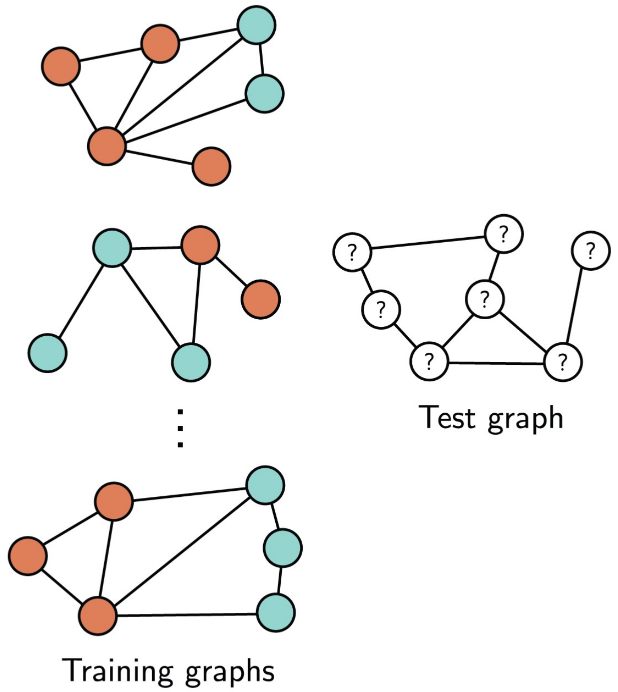
$\Omega_0 \in \mathbb{R}^{D\times 118}$: convert to D-dimensional embeddings

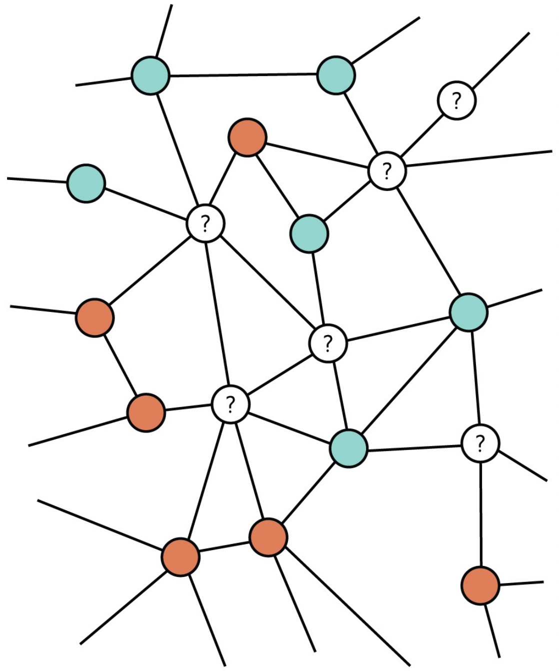
β_K: is a scalar

ω_K: a $1\times D$ parameters row vector

See notebook 13.2.

Inductive vs. Transductive





supervised learning: train with the labeled graphs and then run inference on the unlabeled (test) graphs

semi-supervised learning: train with the labeled nodes, then run inference to determine label for unlabeled nodes

Node classification example

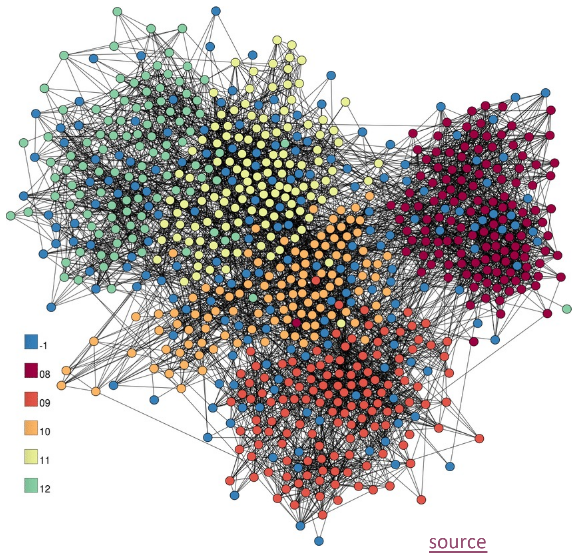
Assume *transductive* binary node *classification* with millions of nodes, *partially labeled*.

Same network body as graph classification, but different head:

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

No mean pooling. Output is $1 \times N$.

Train with binary cross-entropy loss on nodes with labels.



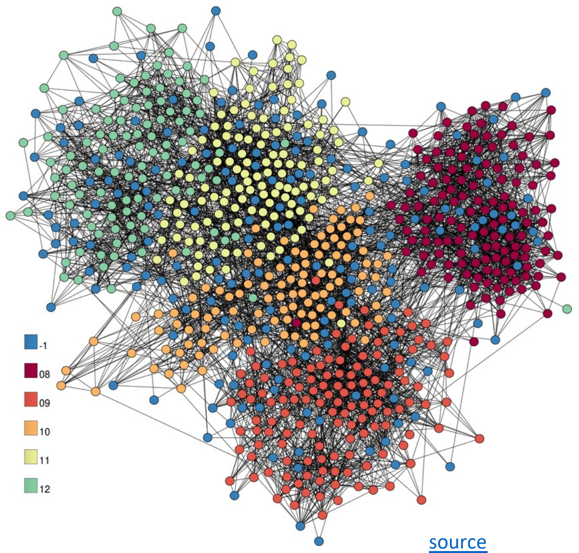
source

Node classification example

Assume *transductive* binary node *classification* with millions of nodes, *partially labeled*.

Challenges:

1. memory limitations: need to store every node and hidden layer embedding during training
2. how to perform SGD with basically one batch!



source

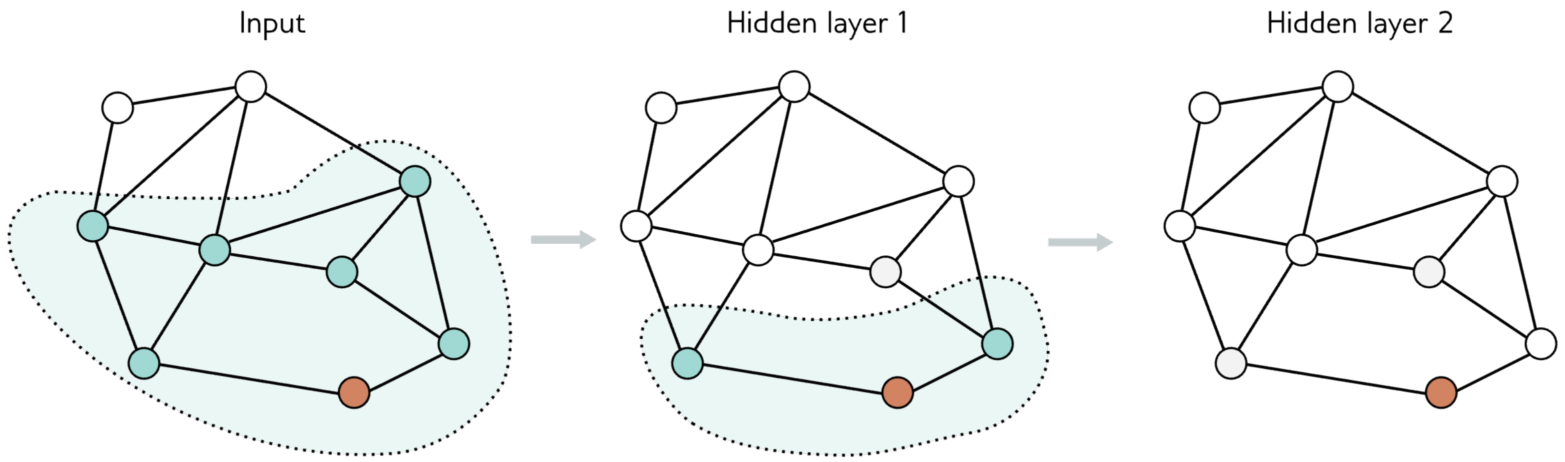
Solutions: Choosing batches for graphs

1. Choose random subset of nodes

2. Neighborhood sampling

3. Graph partitioning

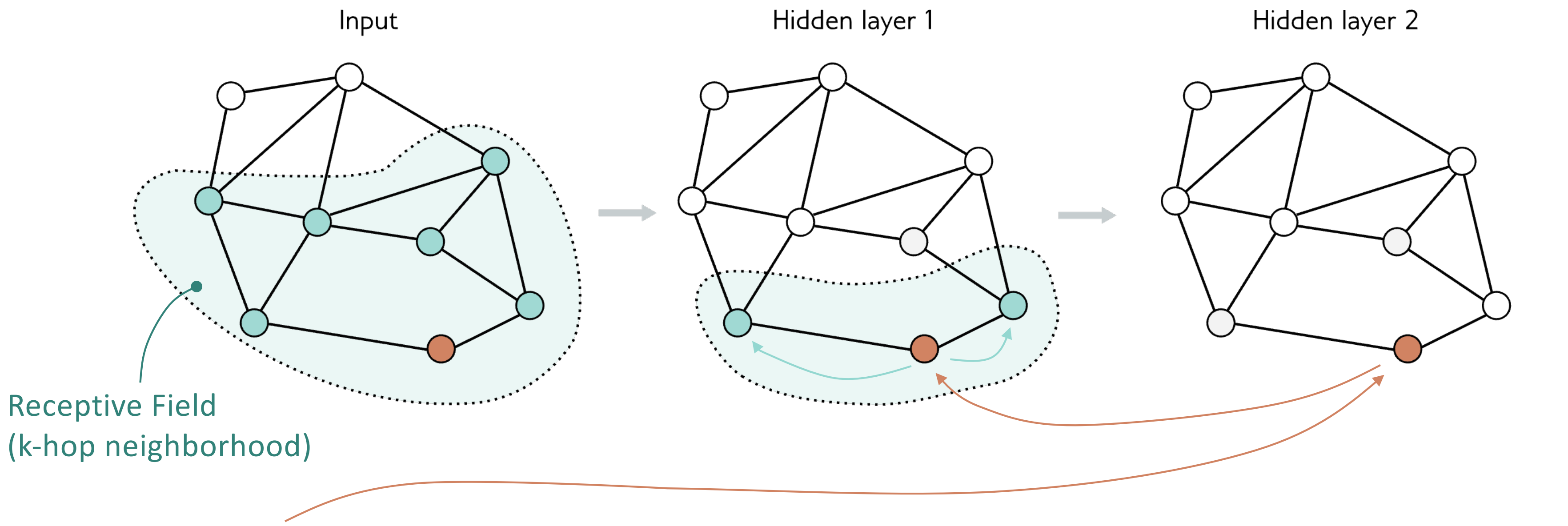
Batches: Random subset



You can pick a random batch of labeled nodes at each training step,

And only include them and their “k-hop neighborhoods”.

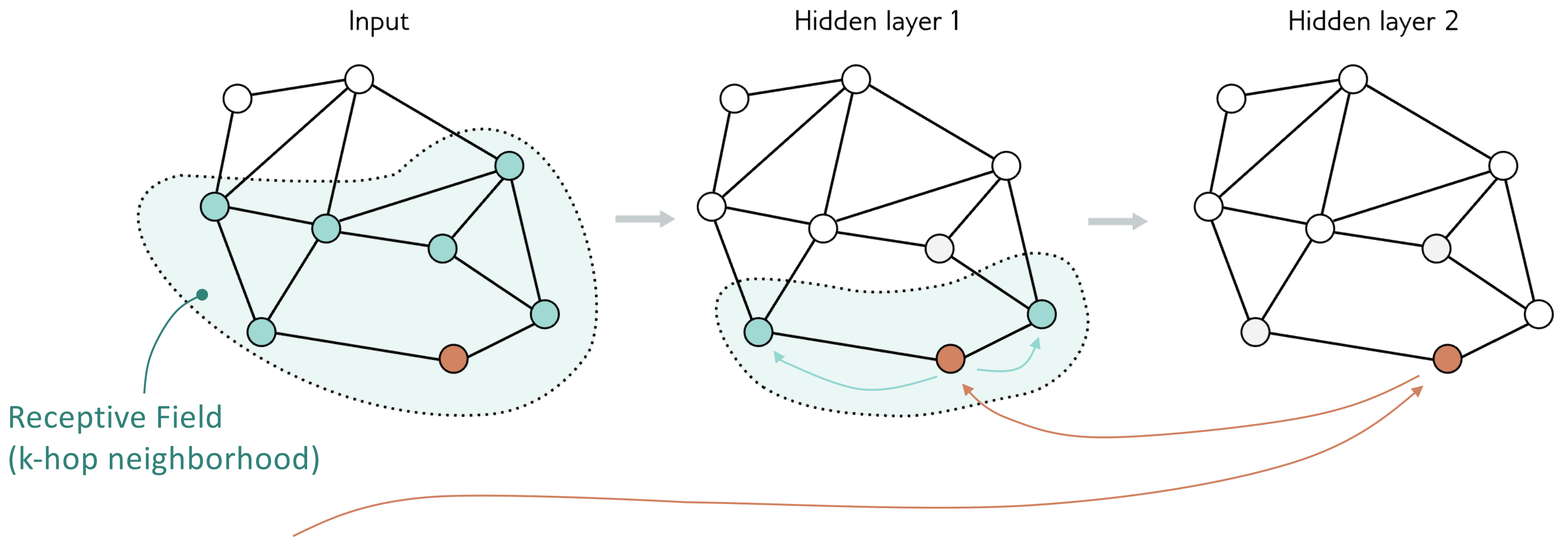
Batches: Random subset



Each node is dependent on the same node in the previous layer and its neighbors because of agg[]

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

Batches: Random subset

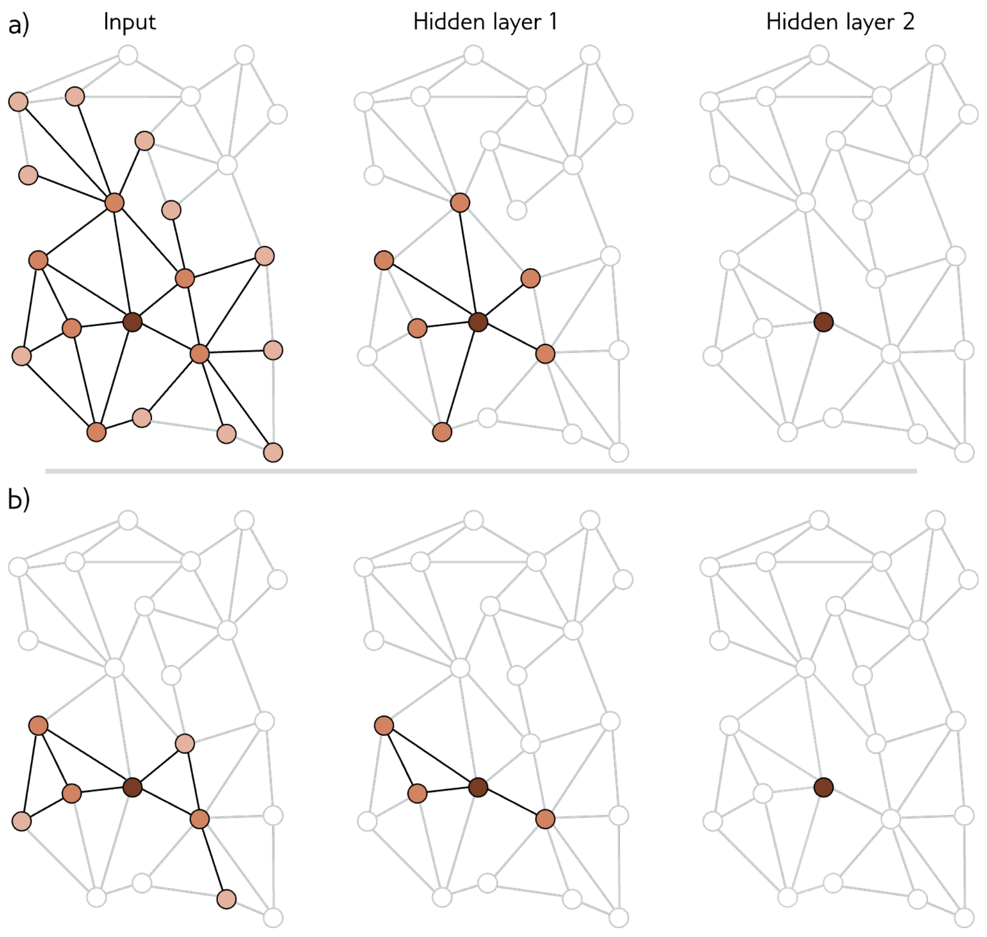


Each node is dependent on the same node in the previous layer and its neighbors because of agg[].

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

With many layers and dense connection, it can quickly expand to encompass every node.

Neighborhood Sampling



Random Sampling:

Use all the neighbors

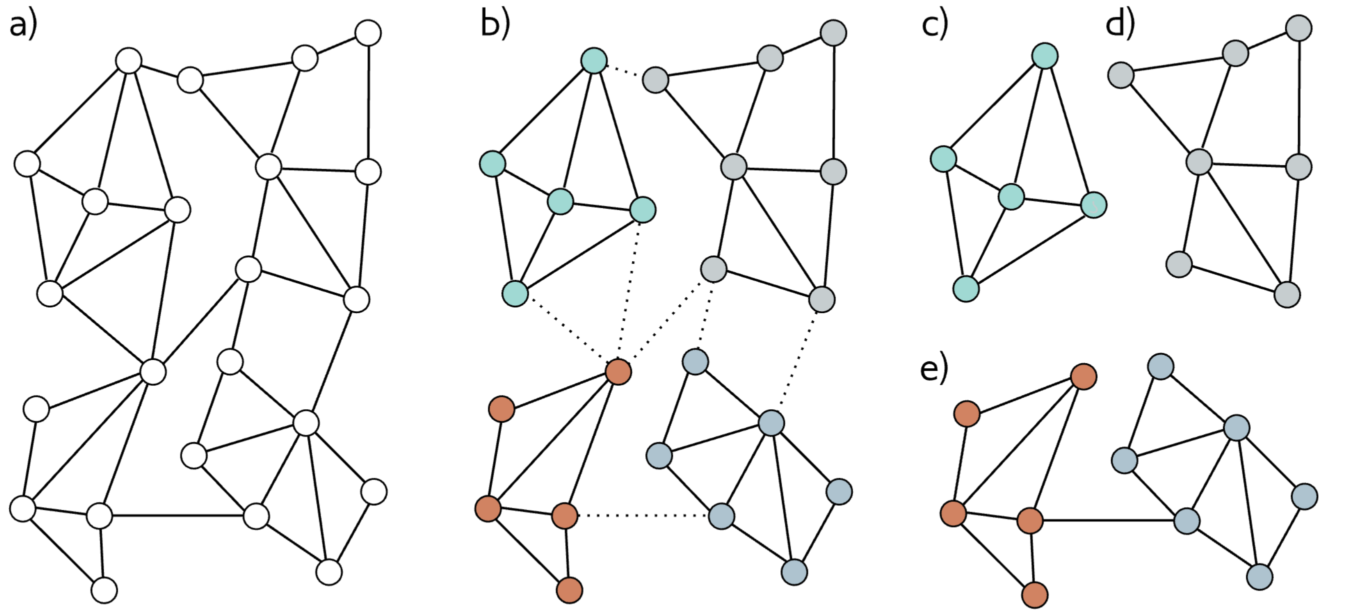
Neighborhood Sampling:

Use max n of the neighbors.

Here $n = 3$.

See Notebook 13.3

Graph Partitioning



Disconnect edges of the original to create maximally connected disjoint subsets

Split into train, test and validation sets and train just like in the inductive setting.

Alternatives to Mean Pooling for Node Combinations

- **Diagonal enhancement**: current node is multiplied by $(1+\epsilon_k)$, where ϵ_k is a learned scalar for each layer

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

- **Residual connections**: Include the current node in the sum

$$\mathbf{H}_{k+1} = \mathbf{a}[\beta_k \mathbf{1}^T + \mathbf{\Omega}_k \mathbf{H}_k \mathbf{A})] + \mathbf{H}_k$$

- **Mean aggregation**: take average instead of sum of neighbors

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

- **Kipf normalization**: downweight neighboring nodes with a lot of neighbors

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

- **Max pool aggregation**: element-wise max of all neighbors to current node

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

Aggregation by Attention

Weights depend on data at the nodes.

Apply linear transform to current node:

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

Then the similarity s_{mn} of each transformed node embedding $\mathbf{h'}_m$ to the transformed node embedding $\mathbf{h'}_n$ is computed by concatenating the pairs, taking a dot product with a column vector ϕ_k of learned parameters, and applying an activation function:

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

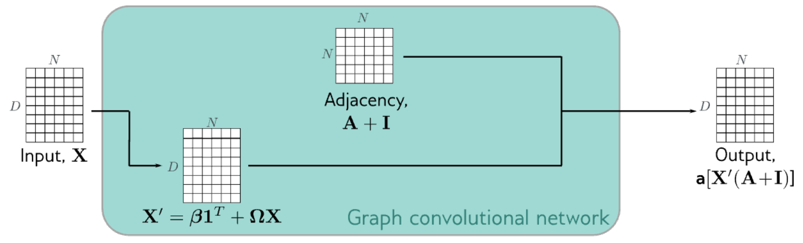
$$\mathbf{H}_{k+1} = \mathbf{a}[\mathbf{H'}_k \cdot \mathrm{Softmask}[\mathbf{S}, \mathbf{A} + \mathbf{I}]$$

Softmask[S, A+I]

The function Softmask[S, A+I]

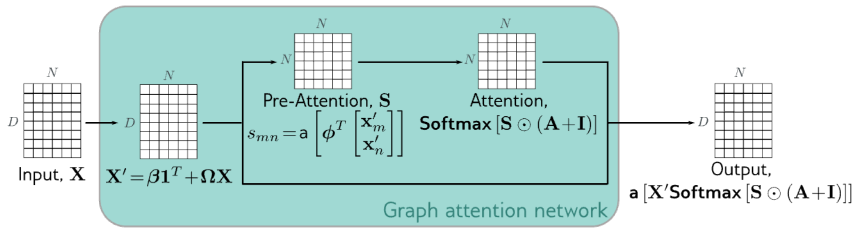
- computes the attention values by applying softmax operation separately to each column of its first argument S,
- but only after setting values where the second argument A + I is zero to negative infinity, so they do not contribute.
- This ensures that the attention to non-neighboring nodes is zero.

Graph Attention



N
D
Input, $\mathbf{X}$
N
N
Adjacency, $\mathbf{A}+\mathbf{I}$
N
D
$\mathbf{X}' = \boldsymbol{\beta}\mathbf{1}^T + \boldsymbol{\Omega}\mathbf{X}$
Graph convolutional network
N
D
Output, $\mathbf{a}[\mathbf{X}'(\mathbf{A}+\mathbf{I})]$

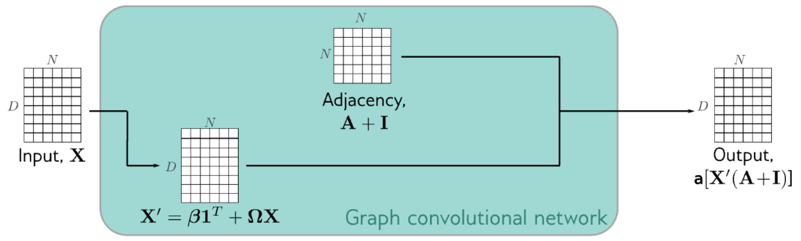
Regular graph convolution



N
D
Input, $\mathbf{X}$
N
D
$\mathbf{X}'=\boldsymbol{\beta}\mathbf{1}^T+\boldsymbol{\Omega}\mathbf{X}$
N
N
Pre-Attention, $\mathbf{S}$
$s_{mn}=\mathbf{a}\left[\boldsymbol{\phi}^T\begin{bmatrix}\mathbf{x}'_m\\ \mathbf{x}'_n\end{bmatrix}\right]$
N
N
Attention, $\textbf{Softmax}[\mathbf{S}\odot(\mathbf{A}+\mathbf{I})]$
Graph attention network
N
D
Output, $\mathbf{a}[\mathbf{X}'\textbf{Softmax}[\mathbf{S}\odot(\mathbf{A}+\mathbf{I})]]$

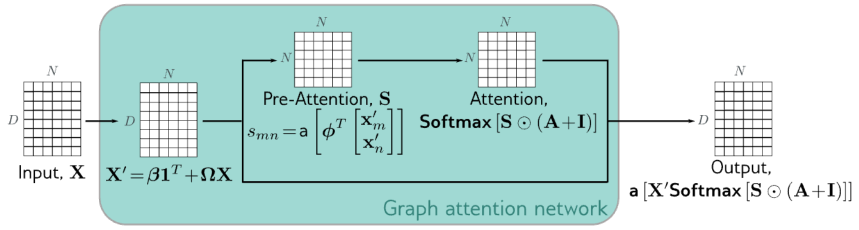
Graph attention

Graph Attention



N
D
Input, $\mathbf{X}$
N
N
Adjacency, $\mathbf{A}+\mathbf{I}$
N
D
$\mathbf{X}' = \beta\mathbf{1}^T + \Omega\mathbf{X}$
Graph convolutional network
N
D
Output, a$[\mathbf{X}'(\mathbf{A}+\mathbf{I})]$

Regular graph convolution

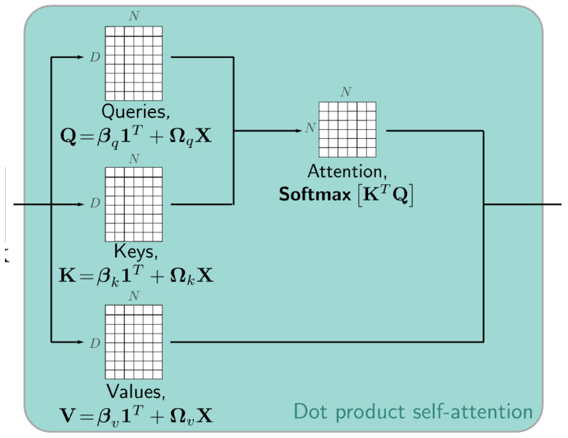


N
D
Input, $\mathbf{X}$
N
D
$\mathbf{X}'=\beta\mathbf{1}^T+\Omega\mathbf{X}$
N
N
Pre-Attention, $\mathbf{S}$
$s_{mn}=\text{a}\left[\phi^T\begin{bmatrix}\mathbf{x}'_m\\\mathbf{x}'_n\end{bmatrix}\right]$
N
N
Attention, Softmax$[\mathbf{S}\odot(\mathbf{A}+\mathbf{I})]$
Graph attention network
N
D
Output, a$[\mathbf{X}'\text{Softmax}[\mathbf{S}\odot(\mathbf{A}+\mathbf{I})]]$

Graph attention

Similar to Transformer Self Attention, *except*

- K, Q and V are all the same
- Different similarity measure
- Only attends to neighbors

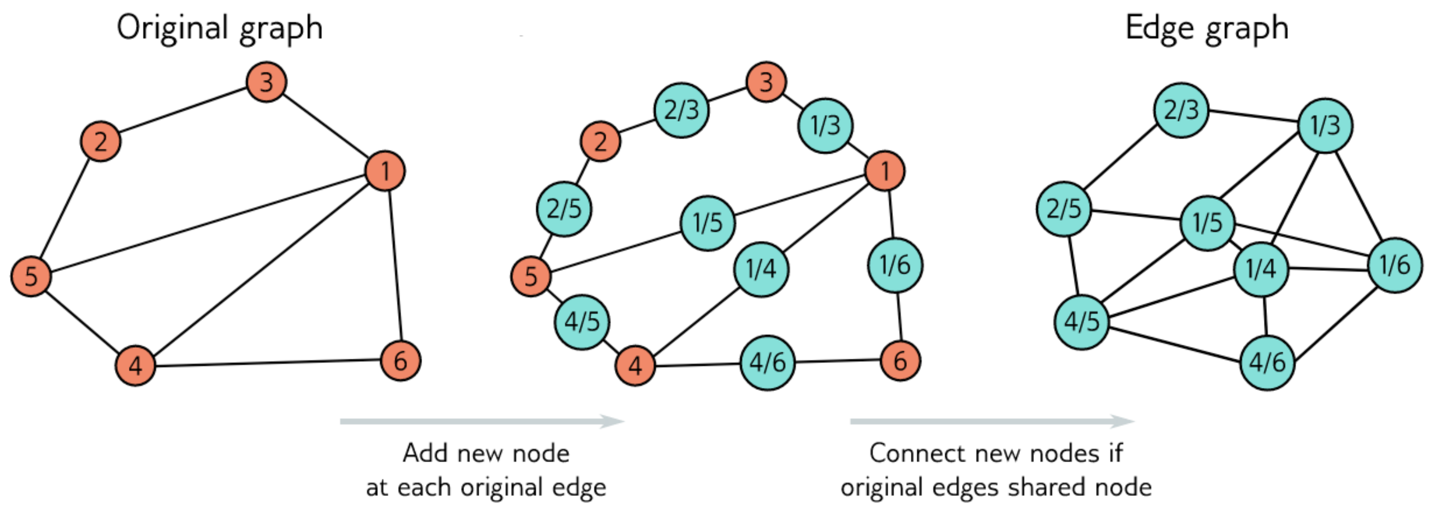


N
D
Queries, $\mathbf{Q}=\beta_q\mathbf{1}^T+\Omega_q\mathbf{X}$
N
D
Keys, $\mathbf{K}=\beta_k\mathbf{1}^T+\Omega_k\mathbf{X}$
N
D
Values, $\mathbf{V}=\beta_v\mathbf{1}^T+\Omega_v\mathbf{X}$
N
N
Attention, Softmax$[\mathbf{K}^T\mathbf{Q}]$
Dot product self-attention

Topics

- Basic definition and examples
- Graph representation
- Properties of Adjacency Matrix
- Graph neural network, tasks and loss functions
- Graph convolutional network
- Graph & Node classification
- Edge graphs

Edge Graphs



Handled by simple transformation from node graphs.

Then process as node graph.

Transform back to edge graph.

Next

- Project Presentations
- Course Evaluations!!

- You’re encouraged to explore notebooks:
  - 13.1 Graph representation
  - 13.2 Graph classification
  - 13.3 Neighborhood Sampling
  - 13.4 Graph Attention Networks

Feedback?

Link