

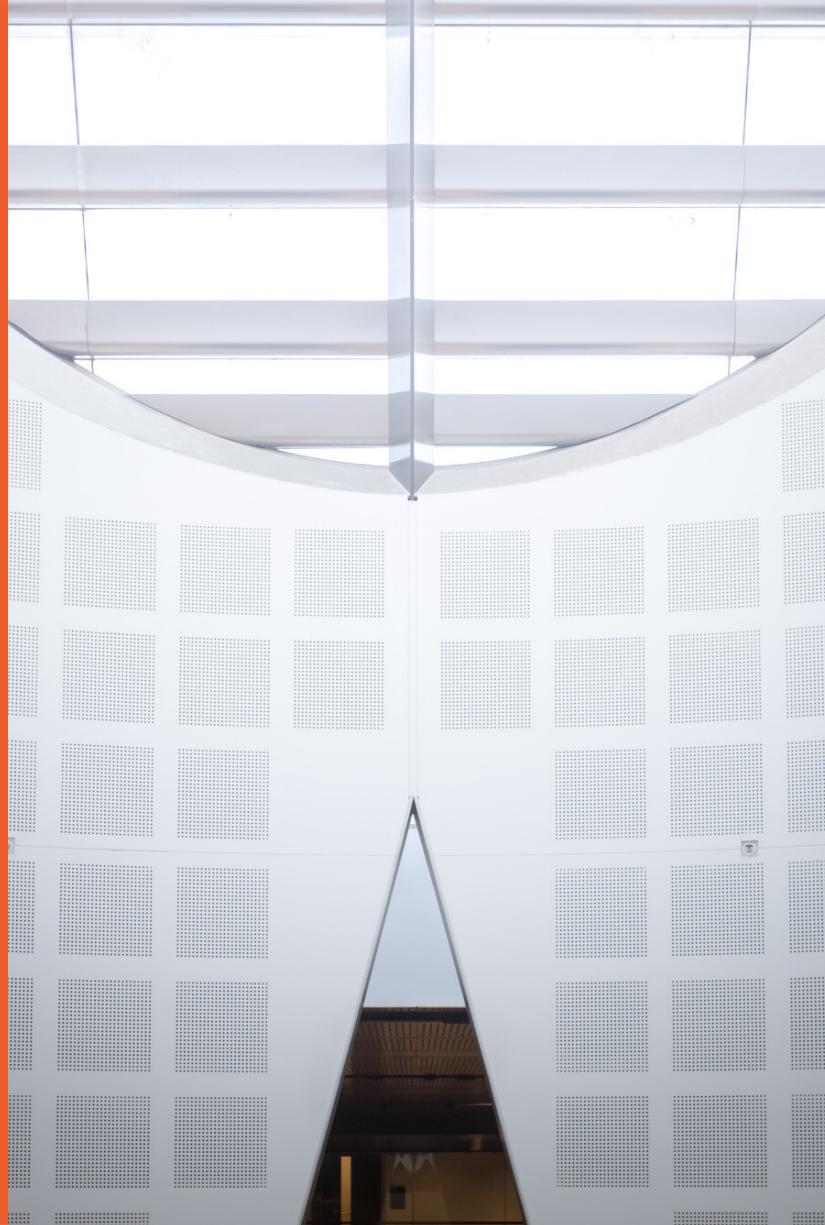
Graph Convolutional Networks

Dr Chang Xu

School of Computer Science



THE UNIVERSITY OF
SYDNEY



Background: Grid structured data

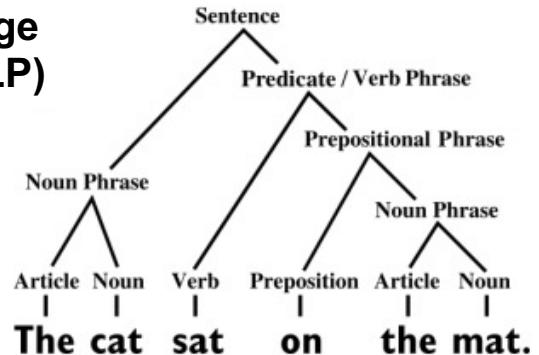


Speech data

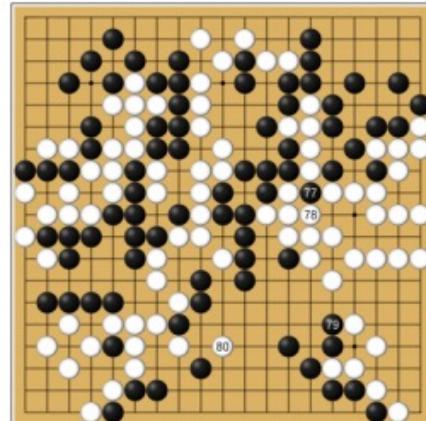


Natural language processing (NLP)

• • •



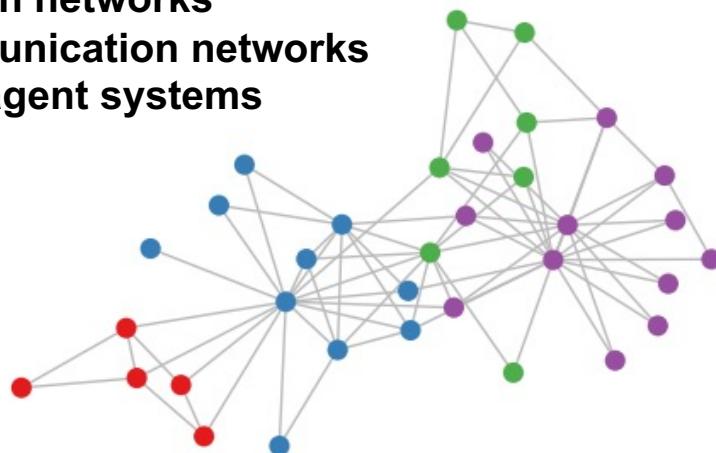
Grid games



Background: Graph data

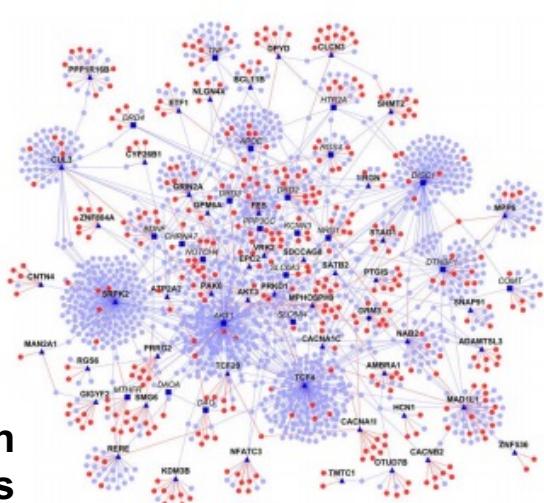
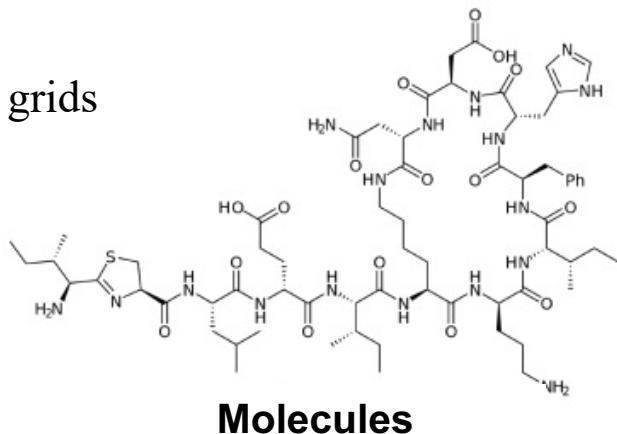
A lot of real-world data does not ‘live’ on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

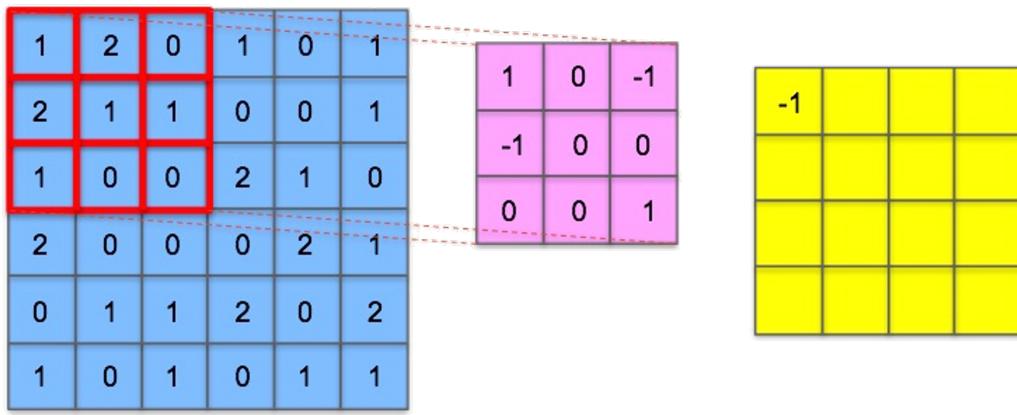


**Protein interaction
networks**

**Standard deep learning architectures
like CNNs and RNNs don’t work here!**



Background: Difficulties on graph data processing



Fixed number of neighboring pixels

&

Implicit spatial order of neighboring pixels

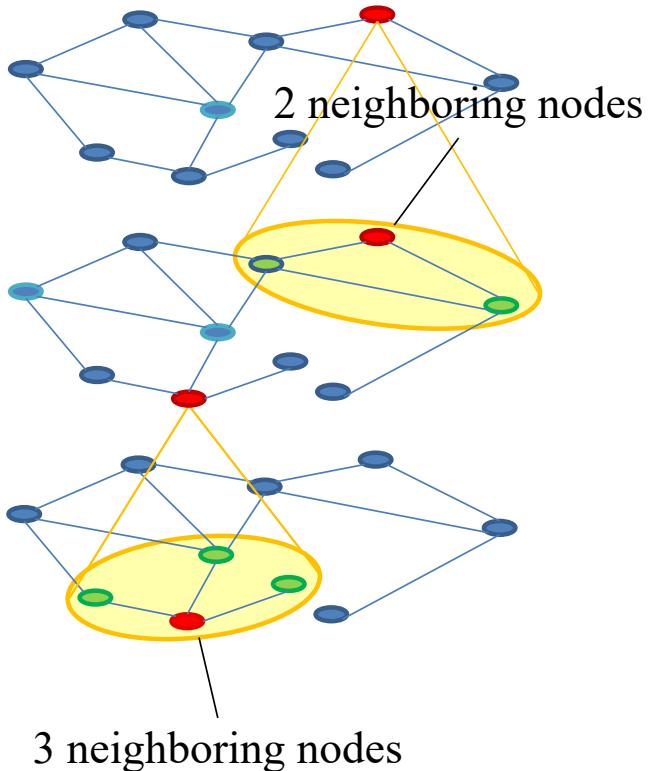


Kernel with fixed size

&

Weights with implicit order

Background: Difficulties on graph data processing



Varying number of neighboring nodes

&

Non-implicit spatial order of neighboring nodes



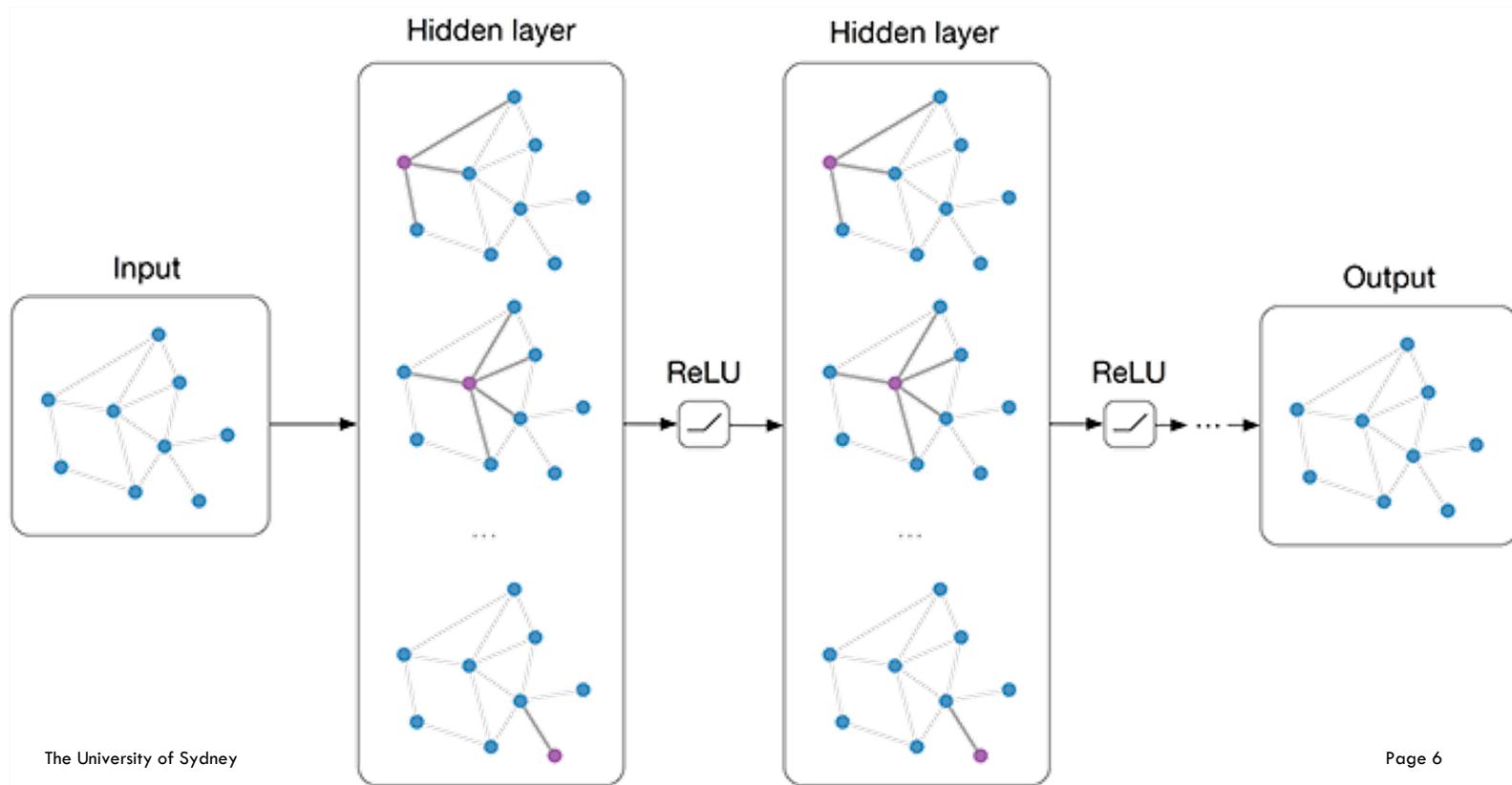
Kernel with varying size

&

Weights with undecided order

Graph Convolutional Networks (GCN)

Extract features from the graph data.



Preliminaries: Representing graphs

Graphs consist of

- ◆ Nodes (or vertices)
- ◆ Edges connecting pairs of nodes

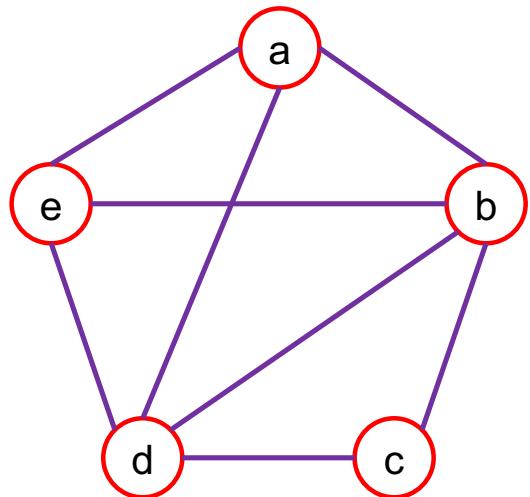
Formally

$$G = (V, E)$$

- ◆ $V = \{v_i | i = 1 \dots N\}$: Node set, containing $|V|=N$ nodes

G

- ◆ $E = \{e_{ij} | v_i \text{ is connected to } v_j\}$: Edge set



Preliminaries: Representing graphs

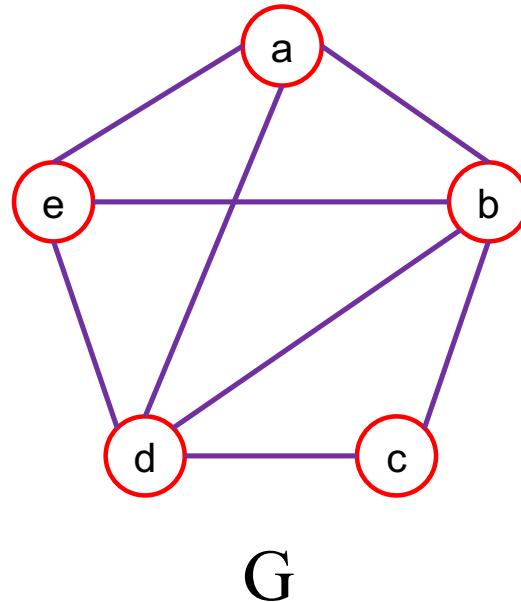
Edge representation

- ◆ Edge list
- ◆ Adjacency matrix
- ◆ Incidence matrix

1. **Edge list:** list of all edges

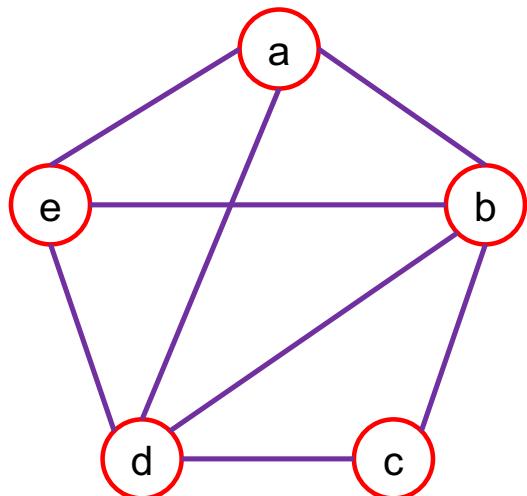
Edges:

(a,b),(a,d),(a,e),(b,c),(b,d),(b,e),(c,d),(d,e)



Preliminaries: Representing graphs

2. Adjacency matrix



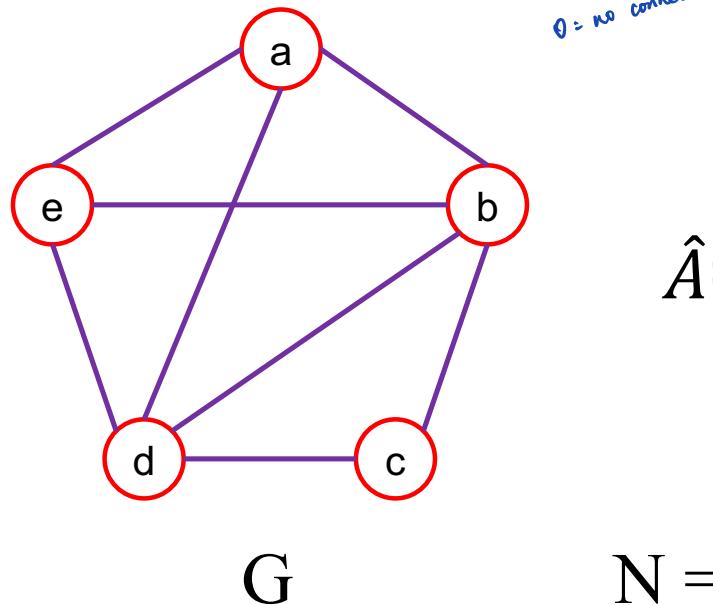
$$A =$$

	a	b	c	d	e
a	0	1	0	1	1
b	1	0	1	1	1
c	0	1	0	1	0
d	1	1	1	0	1
e	1	1	0	1	0

$$N = 5$$

Preliminaries: Representing graphs

2. Adjacency matrix with self connections



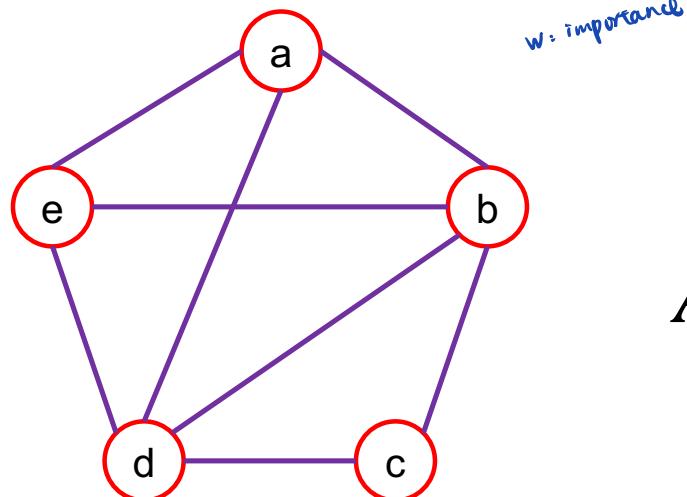
$$\hat{A} =$$

	a	b	c	d	e
a	1	1	0	1	1
b	1	1	1	1	1
c	0	1	1	1	0
d	1	1	1	1	1
e	1	1	0	1	1

$$N = 5$$

Preliminaries: Representing graphs

2. Adjacency matrix (weighted graph)



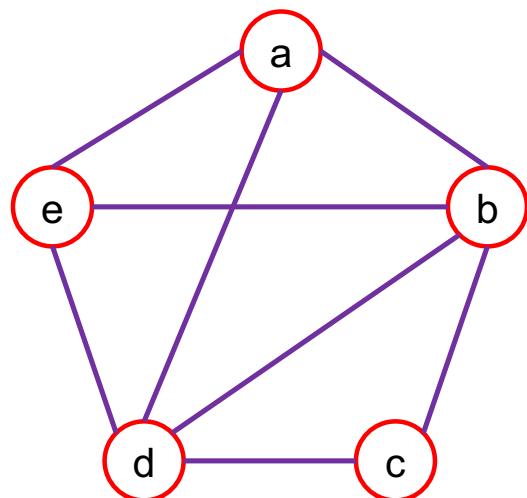
$$A_W =$$

$$N = 5$$

	a	b	c	d	e
a	w_{aa}	w_{ab}	0	w_{ad}	w_{ae}
b	w_{ba}	w_{bb}	w_{bc}	w_{bd}	w_{be}
c	0	w_{cb}	w_{cc}	w_{cd}	0
d	w_{da}	w_{db}	w_{dc}	w_{dd}	w_{de}
e	w_{ea}	w_{eb}	0	w_{ed}	w_{ee}

Preliminaries: Representing graphs

Degree matrix: Denoting degree of each node

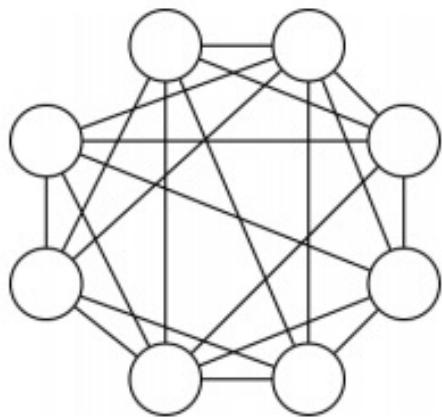


$$D = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \left(\begin{matrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{matrix} \right) \end{matrix}$$

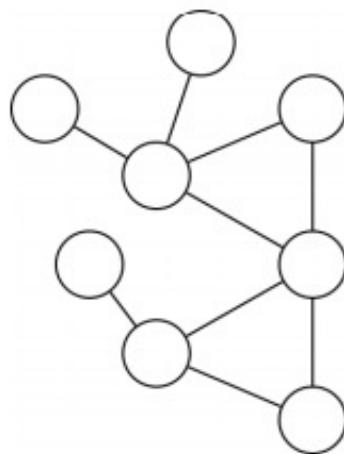
The matrix D is labeled with a, b, c, d, e both as row and column labels. A blue bracket above the matrix is labeled "non-zero". The matrix itself is a 5x5 matrix where the diagonal elements represent the degree of each node: a has degree 3, b has degree 4, c has degree 2, d has degree 4, and e has degree 3.

Preliminaries: Basic properties

Dense graph: $|E| \approx |V|^2$
Or $|E| = O(|V|^2)$



Sparse graph: $|E| \approx |V|$
Or $|E| = O(|V|)$

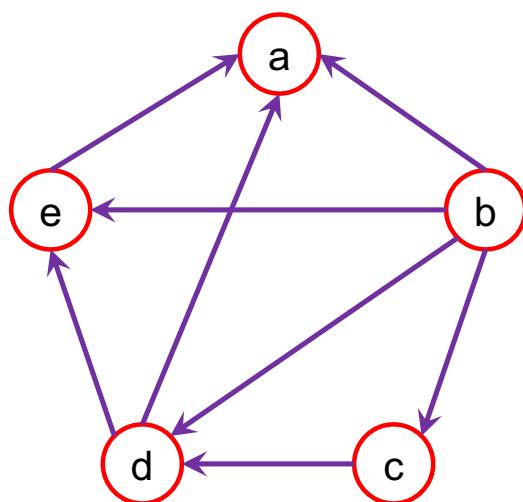


A large fraction of pairs of nodes
are connected by edges

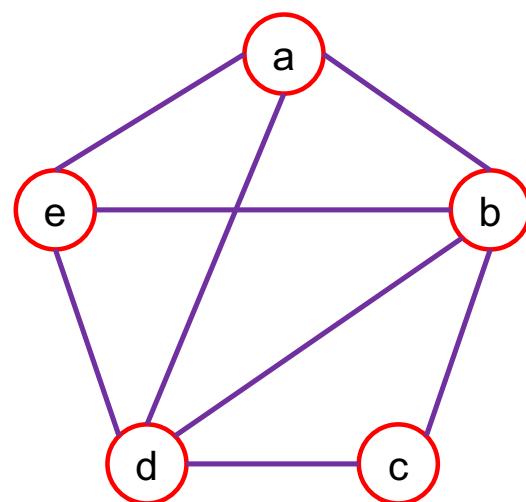
Each node has only a few edges

Preliminaries: Basic properties

Directed graph: Each edge has a direction

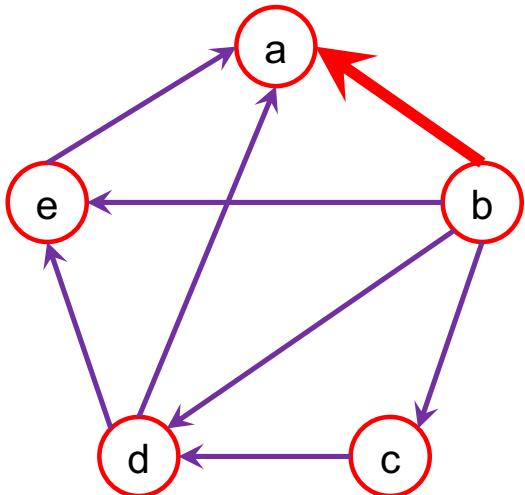


Undirected graph: No direction is associated with edges



Preliminaries: Representing graphs

Adjacency matrix of directed graph could be **asymmetric**



$$A =$$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	1	1
c	0	0	0	1	0
d	1	0	0	0	1
e	1	0	0	0	0

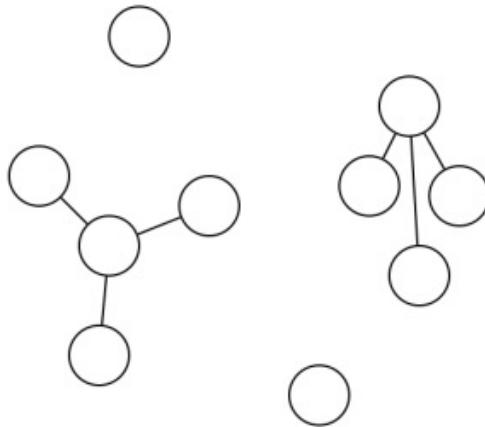
$$N = 5$$

Preliminaries: Basic properties

Connected component: A subgraph in which any two nodes are connected to each other by paths.

Theorem: The nodes of a graph G can be partitioned into connected components so that a node v is reachable from w if and only if they are in the same connected components

4 connected components exist in the graph below



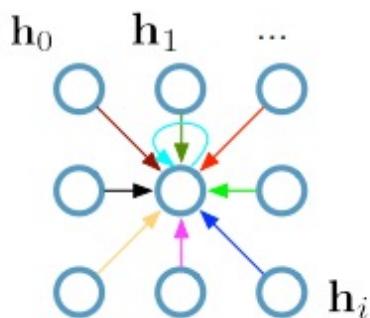
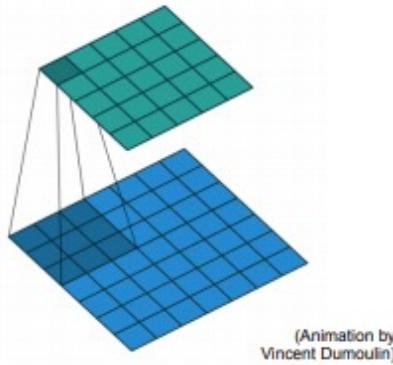
Preliminaries: Representing graphs

List of basics notations

- $G = (V, E)$
- $V = \{v_i | i = 1 \dots N\}$: Node set, containing $|V|=N$ nodes
- $E = \{e_{ij} | v_i \text{ is connected to } v_j\}$: Edge set
- $X \in R^{N \times d}$: Node attribute matrix
- **Adjacency matrix**: $A \in R^{N \times N}, A_{ij} \in \{0,1\}$, denoting the existence of e_{ij}
- I_N : **Identity matrix**, denoting self-connections
- **Adjacency matrix with self-connections**: $\hat{A} = A + I_N$
- Degree: number of edges connected to a node
- **Degree matrix**: $D \in R^{N \times N}$ (computed from A), diagonal matrix denoting the degree of each node. And $\hat{D} \in R^{N \times N}$ is computed from \hat{A}

Convolution in CNN

Single CNN layer
with 3x3 filter:



Update for a single pixel:

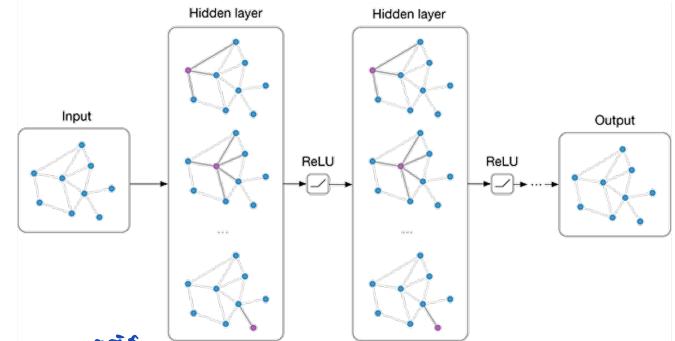
- Transform messages individually $W_i h_i$
- Add everything up $\sum_i W_i h_i$

$h_i \in R^F$ are (hidden layer) activations of a pixel/node

Full update:

$$h_4^{(l+1)} = \sigma(W_0^{(l)} h_0^{(l)} + W_1^{(l)} h_1^{(l)} + \dots + W_8^{(l)} h_8^{(l)})$$

Convolution in GCN



$$H^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^l W^l)$$

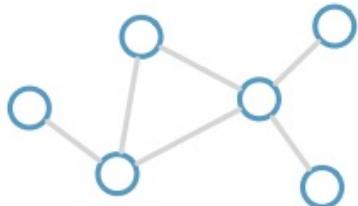
*adjacency matrix
+ self connection*

- Adjacency matrix with self-connections: $\hat{A} = A + I_N$
- Degree: number of edges connected to a node
- Degree matrix: $\hat{D} \in R^{N \times N}$ (computed from \hat{A})

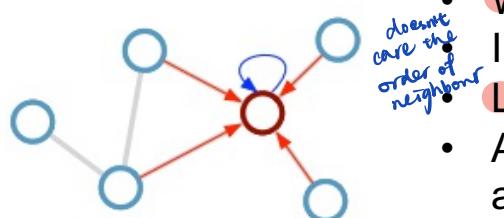
Why $H^{(l+1)} = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^lW^l)$

Graph convolutional networks (Spatial approach)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$h_i^{(l+1)} = \sigma(h_i^{(l)} W_0^{(l)} + \sum_{j \in N_i} \frac{1}{c_{ij}} h_j^{(l)} W_1^{(l)})$$

↑
features
central node ↑
feature in neighbour node

N_i : neighbor indices

c_{ij} : norm. constant
(fixed/trainable)

Desirable properties:
transform all features of its neighbouring nodes

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity $O(E)$
- Applicable both in transductive and inductive settings

Limitations:

- whether we know the testing data*
- Requires gating mechanism / residual connections for depth
 - Only indirect support for edge features
- transductive: specific training case to specific testing case*
- inductive: to general rules*

Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Graph convolutional networks: A concrete model

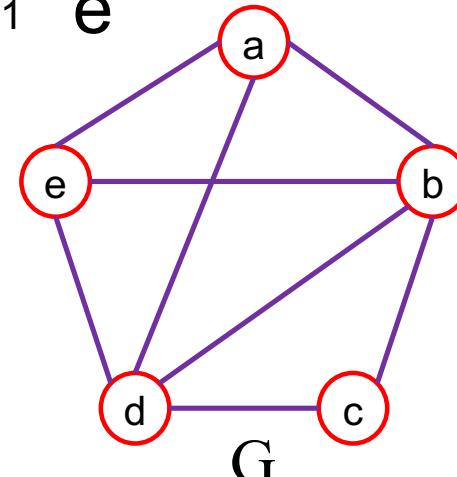
$$\begin{bmatrix} a' \\ b' \\ c' \\ d' \\ e' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$$

A

Focus on updated a'

Only neighboring nodes of a are retained

Unconnected nodes are masked out



$a' = b + d + e$

Graph convolutional networks: A concrete model

$$\begin{bmatrix} a' \\ b' \\ c' \\ d' \\ e' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$$

$a' = b + d + e$

$$\hat{A} = A + I_N$$

Self-loops are added into adjacency matrix

$$H^{(l+1)} = \sigma(\hat{A}H^lW^l)$$

Central nodes are included in convolution

Normalizing the Feature Representations

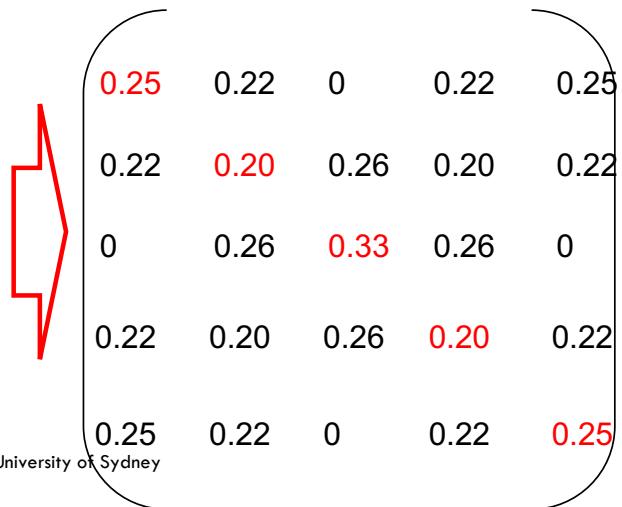
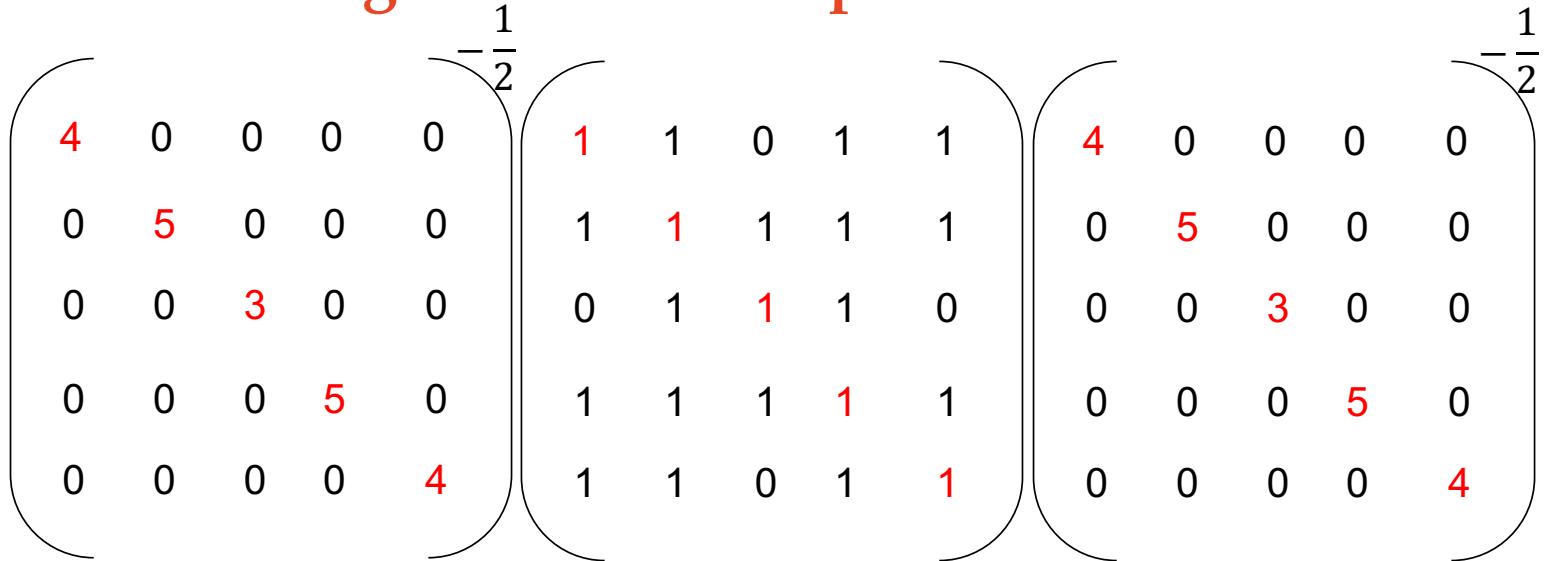
$$H^{(l+1)} = \sigma(\hat{A}H^lW^l) \rightarrow H^{(l+1)} = \sigma(\hat{D}^{-1}\hat{A}H^lW^l)$$

The feature representations can be normalized by node degree by transforming the adjacency matrix \hat{A} by multiplying it with the inverse degree matrix \hat{D} .

problem:
After normalization,
not symmetric

$$\begin{matrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{matrix} \begin{matrix} -1 \\ \times \end{matrix} \begin{matrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{matrix} \begin{matrix} \times \\ \times \end{matrix} \begin{matrix} 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0 & 0.33 & 0.33 & 0.33 & 0 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.25 & 0.25 & 0 & 0.25 & 0.25 \end{matrix}$$

Normalizing the Feature Representations

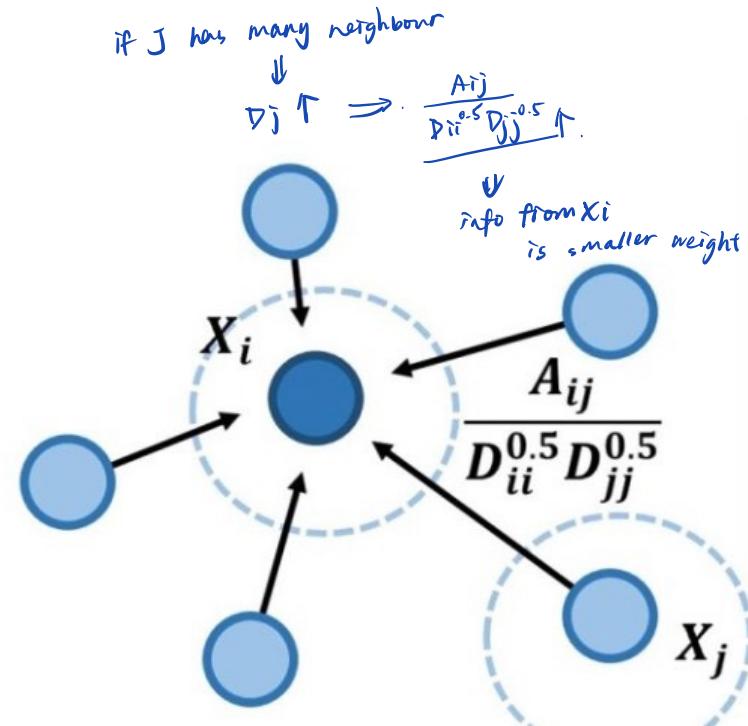


A symmetric normalization

$$H^{(l+1)} = \sigma(\widehat{D}^{-\frac{1}{2}} \widehat{A} \widehat{D}^{-\frac{1}{2}} H^l W^l)$$

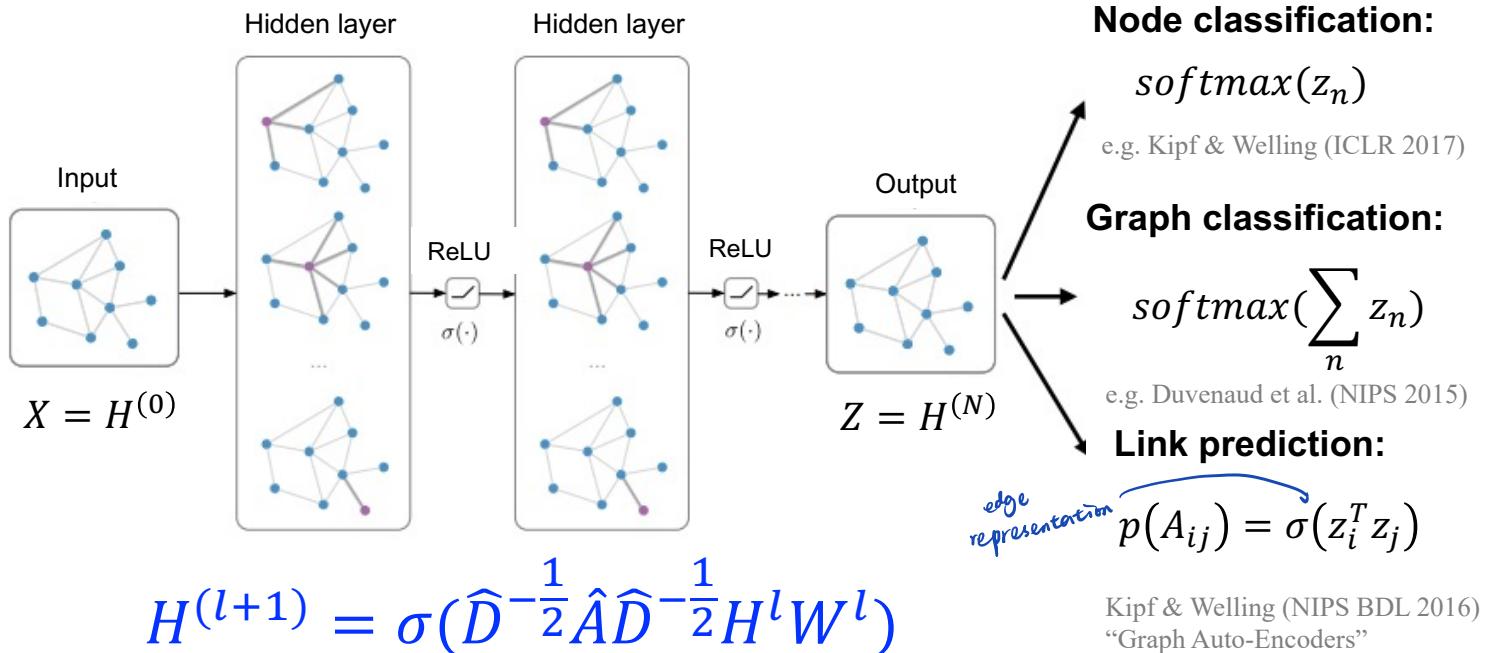
Normalizing the Feature Representations

$$\begin{aligned} & (\widehat{D}^{-0.5} \widehat{A} \widehat{D}^{-0.5} H)_i \\ &= (\widehat{D}^{-0.5} \widehat{A})_i \widehat{D}^{-0.5} H \\ &= \left(\sum_k \widehat{D}_{ik}^{-0.5} \widehat{A}_i \right) \widehat{D}^{-0.5} H \\ &= \widehat{D}_{ii}^{-0.5} \sum_j \widehat{A}_{ij} \sum_k \widehat{D}_{ik}^{-0.5} H_j \\ &= \widehat{D}_{ii}^{-0.5} \sum_j \widehat{A}_{ij} \widehat{D}_{jj}^{-0.5} H_j \\ &= \sum_j \frac{1}{\sqrt{\widehat{D}_{ii} \widehat{D}_{jj}}} \widehat{A}_{ij} H_j \end{aligned}$$



Graph convolutional networks: A concrete model

Input: Feature matrix $X \in R^{N \times E}$, preprocessed adjacency matrix \hat{A}



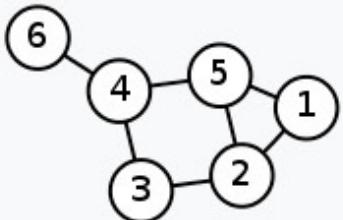
Semi-Supervised Classification with Graph Convolutional Networks (Kipf & Welling)

Spectral approach

Graph convolutional networks (Spectral approach)

Graph Laplacian: $L = D - A$

Normalized Graph Laplacian: $L' = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

https://en.wikipedia.org/wiki/Laplacian_matrix

Fourier transform

Convolution theorem: Under suitable conditions, the Fourier (Laplace) transform of a convolution of two signals is the pointwise product of their Fourier (Laplace) transforms.

Given a graph f and a convolutional filter (with trainable parameters) h , the convolution can be calculated by

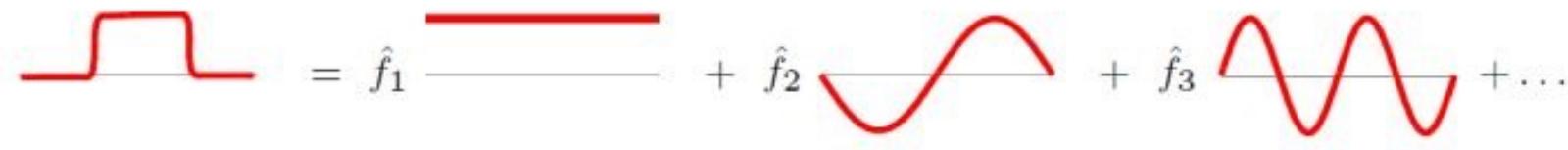
$$f * h = \mathcal{F}^{-1}[\hat{f}(\omega)\hat{h}(\omega)]$$

\downarrow

\hat{f}, \hat{h} denotes Fourier transform

Fourier transform

combine different basic frequencies



The classic Fourier transform

$$\hat{f}(\xi) = \langle f, e^{2\pi i \xi t} \rangle = \int_{\mathbb{R}} f(t) e^{2\pi i \xi t} dt$$

is the expansion of f in terms of the complex exponentials ($e^{2\pi i \xi t}$);
the expansion results are the eigenfunctions of 1-d Laplace operator
 Δ :

$$-\Delta(e^{2\pi i \xi t}) = -\frac{\partial^2}{\partial t^2} e^{2\pi i \xi t} = (2\pi\xi)^2 e^{2\pi i \xi t}$$

$$Lu = \lambda \underset{\substack{\text{eigen values} \\ \text{eigen vector}}}{u}$$

Fourier transform on the graph

The classic Fourier transform

$$\hat{f}(\xi) = \langle f, e^{2\pi i \xi t} \rangle = \int_{\mathbb{R}} f(t) e^{2\pi i \xi t} dt$$

Graph Fourier transform \hat{f} : of any $f \in \mathbb{R}^N$, of all vertices of G , expansion of f :

$$\hat{f}(\lambda_l) = \langle f, u_l \rangle = \sum_i^N f(i) u_l^*(i) \quad \hat{f} = U^T f$$

Similarly,

$$\hat{h}(\lambda_l) = \langle h, u_l \rangle = \sum_i^N h(i) u_l^*(i) \quad \hat{h} = U^T h$$

The inverse graph Fourier transform is then given by:

$$f(i) = \sum_{l=0}^{N-1} \hat{f}(\lambda_l) u_l(i) \quad f = U \hat{f}$$

Graph convolution

Convolution theorem: Under suitable conditions, the Fourier (Laplace) transform of a convolution of two signals is the pointwise product of their Fourier (Laplace) transforms.

Given a graph f and a convolutional filter (with trainable parameters) h , the convolution can be calculated by

$$f * h = \mathcal{F}^{-1}[\hat{f}(\omega)\hat{h}(\omega)]$$
$$\hat{f} = U^T f \quad \hat{h} = U^T h \quad f = U\hat{f}$$
$$(f * h)_G = U((U^T f) \odot (U^T h))$$

Elementwise multiplication

Graph convolutional network

$$(f * h)_G = U((U^T f) \odot (U^T h))$$

Version 1.0 (Spectral Networks and Locally Connected Networks on Graphs)

$$y = \sigma(U g_\theta(\Lambda) U^T x)$$

↑ feature map ↗ corresponds to h

$$g_\theta(\Lambda) = \begin{bmatrix} \theta_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \theta_n \end{bmatrix}$$

- Calculate the multiplication between U , $g_\theta(\Lambda)$, and U^T in each feedforward.
- No spatial localization
- #param = n
↑
graph nodes

Graph convolutional network

- No eigen decomposition
- Spatial localization
- #param = K

$$(f * h)_G = U((U^T f) \odot (U^T h))$$

Version 2.0 (Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering)

$$y = \sigma(U g_\theta(\Lambda) U^T x)$$

$K \leq n$ (# nodes)

$$g_\theta(\Lambda) = \begin{bmatrix} \sum_{j=0}^K \alpha_j \lambda_1^j & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sum_{j=0}^K \alpha_j \lambda_n^j \end{bmatrix}$$

λ eigenvalues of $L = D - A$ (determined by graph)

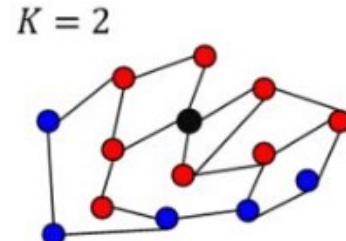
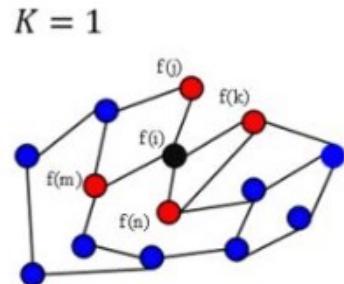
$$\begin{aligned} U g_\theta(\Lambda) U^T &= U \sum_{j=0}^K \alpha_j \Lambda^j U^T \\ &\longrightarrow \\ &= \sum_{j=0}^K \alpha_j L^j \end{aligned}$$

j^{th} power

Graph convolutional network

$$y = \sigma(U g_\theta(\Lambda) U^T x) = \sigma\left(\sum_{j=0}^{K-1} \alpha_j L^j x\right)$$

- There are only K parameters to train, and usually $K \ll n$. The complexity has been decreased.
- No need to conduct eigen decomposition. Explicitly depend on the Laplacian matrix L.
- Spatial localization. In particular, K denotes the receptive field.



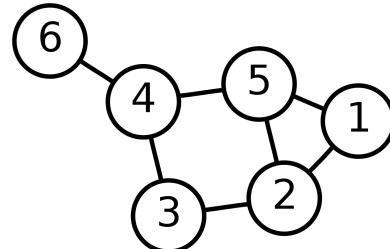
Graph convolutional network

Local Connectivity + Parameter Sharing

K=0, the convolution filter is

$$\begin{bmatrix} \alpha_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha_0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha_0 \end{bmatrix}$$

(only the node itself)



K=1, the convolution filter is

$$\begin{bmatrix} \alpha_0 + 2\alpha_1 & -\alpha_1 & 0 & 0 & -\alpha_1 & 0 \\ -\alpha_1 & \alpha_0 + 3\alpha_1 & -\alpha_1 & 0 & -\alpha_1 & 0 \\ 0 & -\alpha_1 & \alpha_0 + 2\alpha_1 & -\alpha_1 & 0 & 0 \\ 0 & 0 & -\alpha_1 & \alpha_0 + 3\alpha_1 & -\alpha_1 & -\alpha_1 \\ -\alpha_1 & -\alpha_1 & 0 & -\alpha_1 & \alpha_0 + 3\alpha_1 & 0 \\ 0 & 0 & 0 & -\alpha_1 & 0 & \alpha_0 + \alpha_1 \end{bmatrix}$$

K=2, the convolution filter is

2nd order relation (increase the receptive field)

$$\begin{bmatrix} \alpha_0 + 2\alpha_1 + 6\alpha_2 & -\alpha_1 - 4\alpha_2 & \alpha_2 & \alpha_2 & -\alpha_1 - 4\alpha_2 & 0 \\ -\alpha_1 - 4\alpha_2 & \alpha_0 + 3\alpha_1 + 12\alpha_2 & -\alpha_1 - 5\alpha_2 & 2\alpha_2 & -\alpha_1 - 5\alpha_2 & 0 \\ \alpha_2 & -\alpha_1 - 5\alpha_2 & \alpha_0 + 2\alpha_1 + 6\alpha_2 & -\alpha_1 - 5\alpha_2 & 2\alpha_2 & \alpha_2 \\ \alpha_2 & 2\alpha_2 & -\alpha_1 - 5\alpha_2 & \alpha_0 + 3\alpha_1 + 12\alpha_2 & -\alpha_1 - 6\alpha_2 & -\alpha_1 - 4\alpha_2 \\ -\alpha_1 - 4\alpha_2 & -\alpha_1 - 5\alpha_2 & 2\alpha_2 & -\alpha_1 - 6\alpha_2 & \alpha_0 + 3\alpha_1 + 12\alpha_2 & \alpha_2 \\ 0 & 0 & \alpha_2 & -\alpha_1 - 4\alpha_2 & \alpha_2 & \alpha_0 + \alpha_1 + 2\alpha_2 \end{bmatrix}$$

Graph convolutional network

A Recursive formulation for fast filtering

Recall:

$$y = \sigma(Ug_\theta(\Lambda)U^T x)$$

With Chebyshev expansion, we have

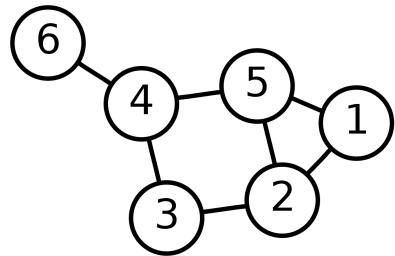
$$g_\theta \approx \sum_{j=0}^K \theta'_j T_j(\tilde{\Lambda}) \quad \text{where} \quad \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I_N$$

range $\in [-1, 1]$

Chebyshev polynomials: $T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x)$,
with $T_0(x) = 1$ and $T_1(x) = x$

$$\begin{aligned} y &= \sigma(Ug_\theta(\Lambda)U^T x) = \sigma\left(U \sum_{j=0}^K \theta'_j T_j(\tilde{\Lambda}) U^T x\right) = \sigma\left(\sum_{j=0}^K \theta'_j T_j(\tilde{\Lambda} U U^T) x\right) \\ &= \sigma\left(\sum_{j=0}^K \theta'_j T_j(\tilde{L}) x\right) \end{aligned}$$

Graph convolutional network



$$L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

K=0, the convolution filter is

$$\begin{bmatrix} \beta_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \beta_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta_0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta_0 \end{bmatrix}$$

K=1, the convolution filter is

\uparrow small \Rightarrow we should focus more on node itself rather than neighbour

$$\begin{bmatrix} \beta_0 + 0.07\beta_1 & -0.44\beta_1 & 0 & 0 & -0.44\beta_1 & 0 \\ -0.44\beta_1 & \beta_0 + 0.07\beta_1 & -0.44\beta_1 & 0 & -0.36\beta_1 & 0 \\ 0 & -0.44\beta_1 & \beta_0 + 0.07\beta_1 & -0.44\beta_1 & 0 & 0 \\ 0 & 0 & -0.44\beta_1 & \beta_0 + 0.07\beta_1 & -0.36\beta_1 & -0.62\beta_1 \\ -0.36\beta_1 & -0.36\beta_1 & 0 & -0.36\beta_1 & \beta_0 + 0.07\beta_1 & 0 \\ 0 & 0 & 0 & -0.62\alpha_1 & 0 & \beta_0 + 0.07\beta_1 \end{bmatrix}$$

- Considering the diagonal entry, the coefficient before β_1 is small.
- β_1 is to control the properties of the one-hop neighbor.

Thank you!