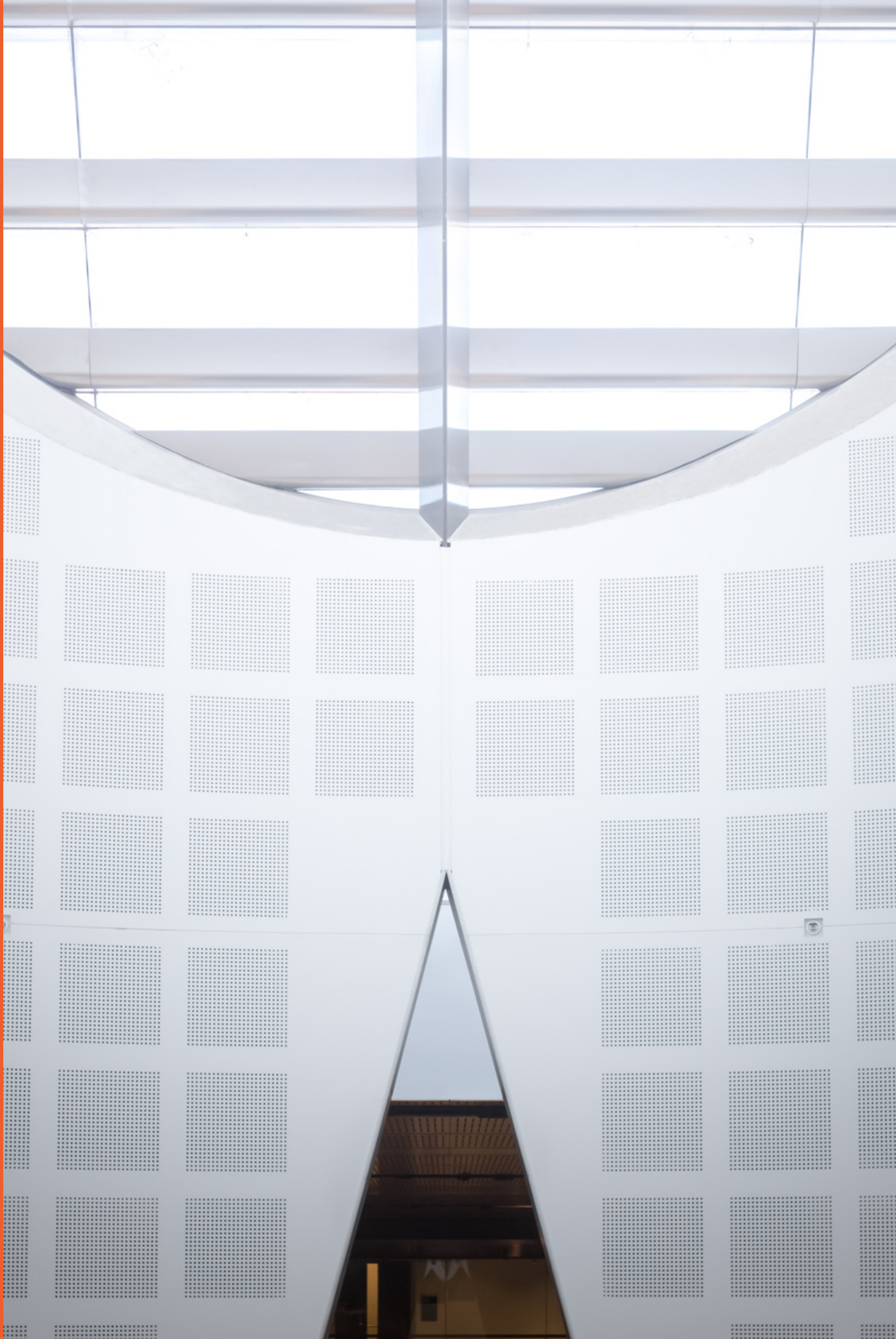
Graph Convolutional Networks

Dr Chang Xu

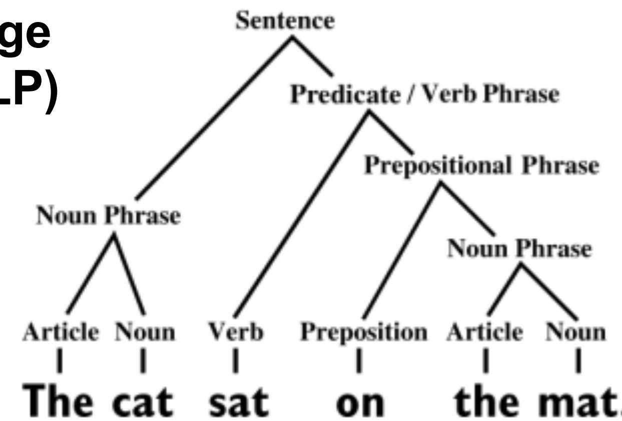
School of Computer Science



Background: Grid structured data

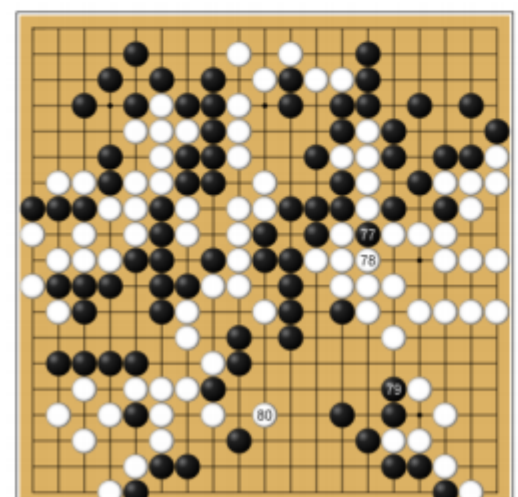
IMAGENET

Natural language processing (NLP)



...

Grid games

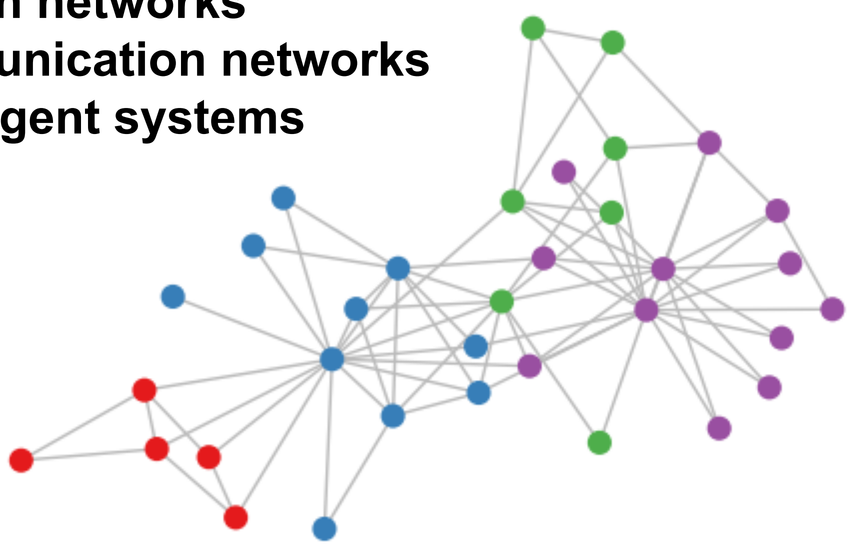


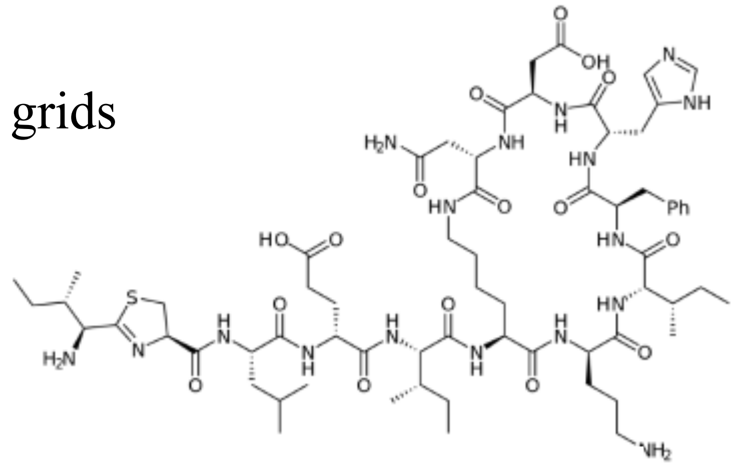
Speech data

Background: Graph data

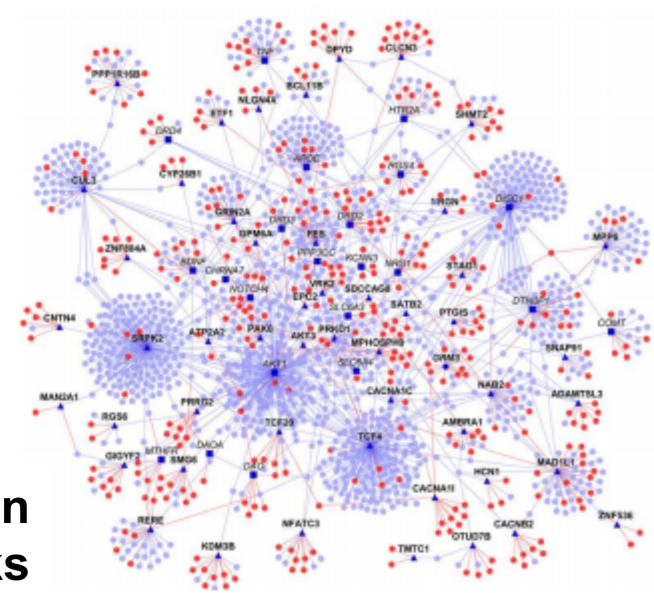
A lot of real-world data does not ‘live’ on grids

Social networks
Citation networks
Communication networks
Multi-agent systems





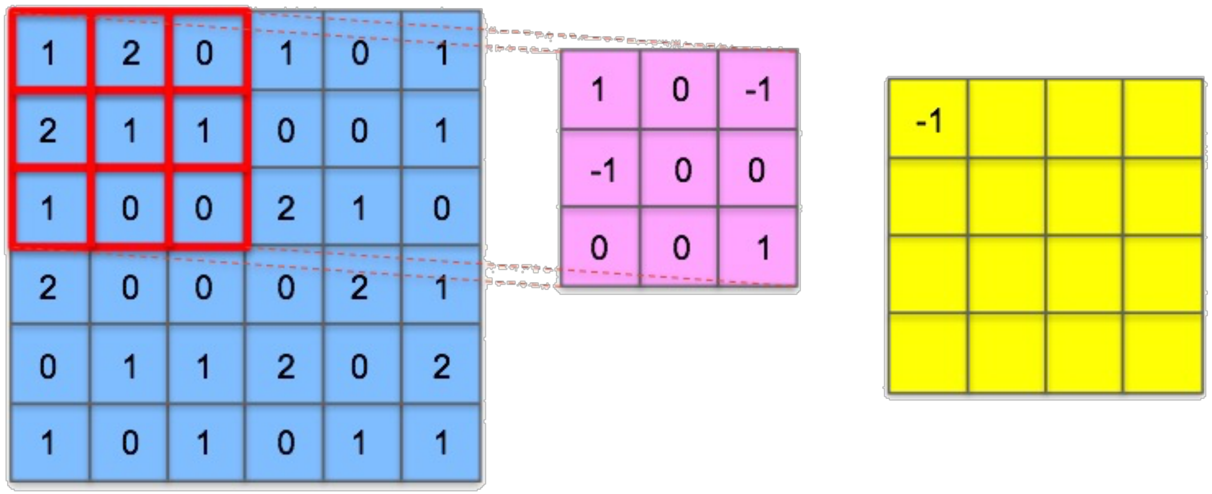
Molecules



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

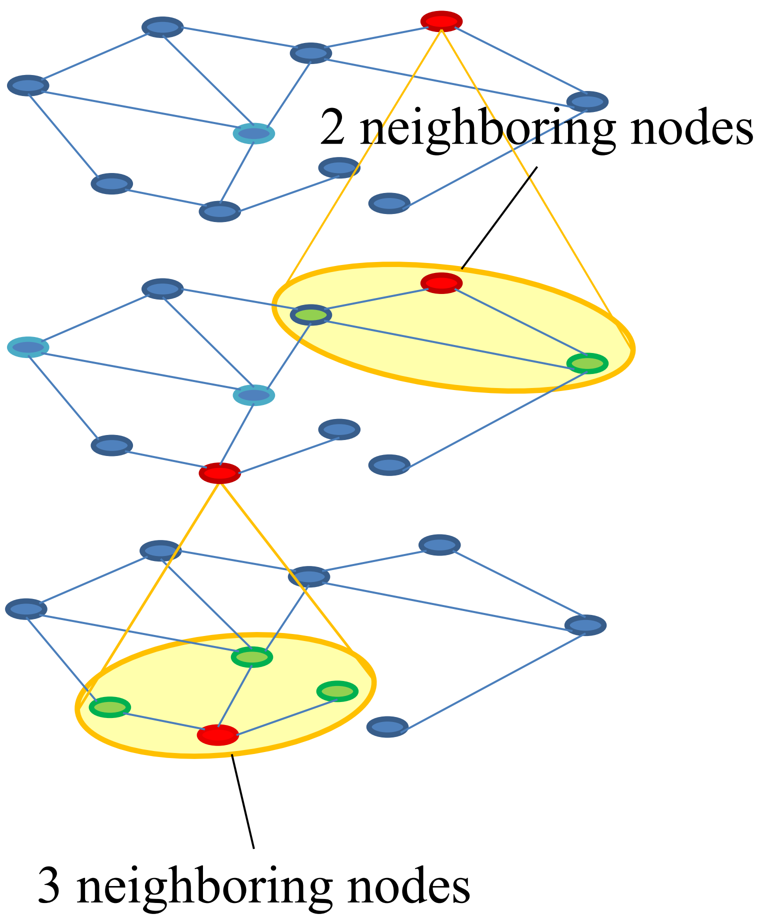
⇒

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

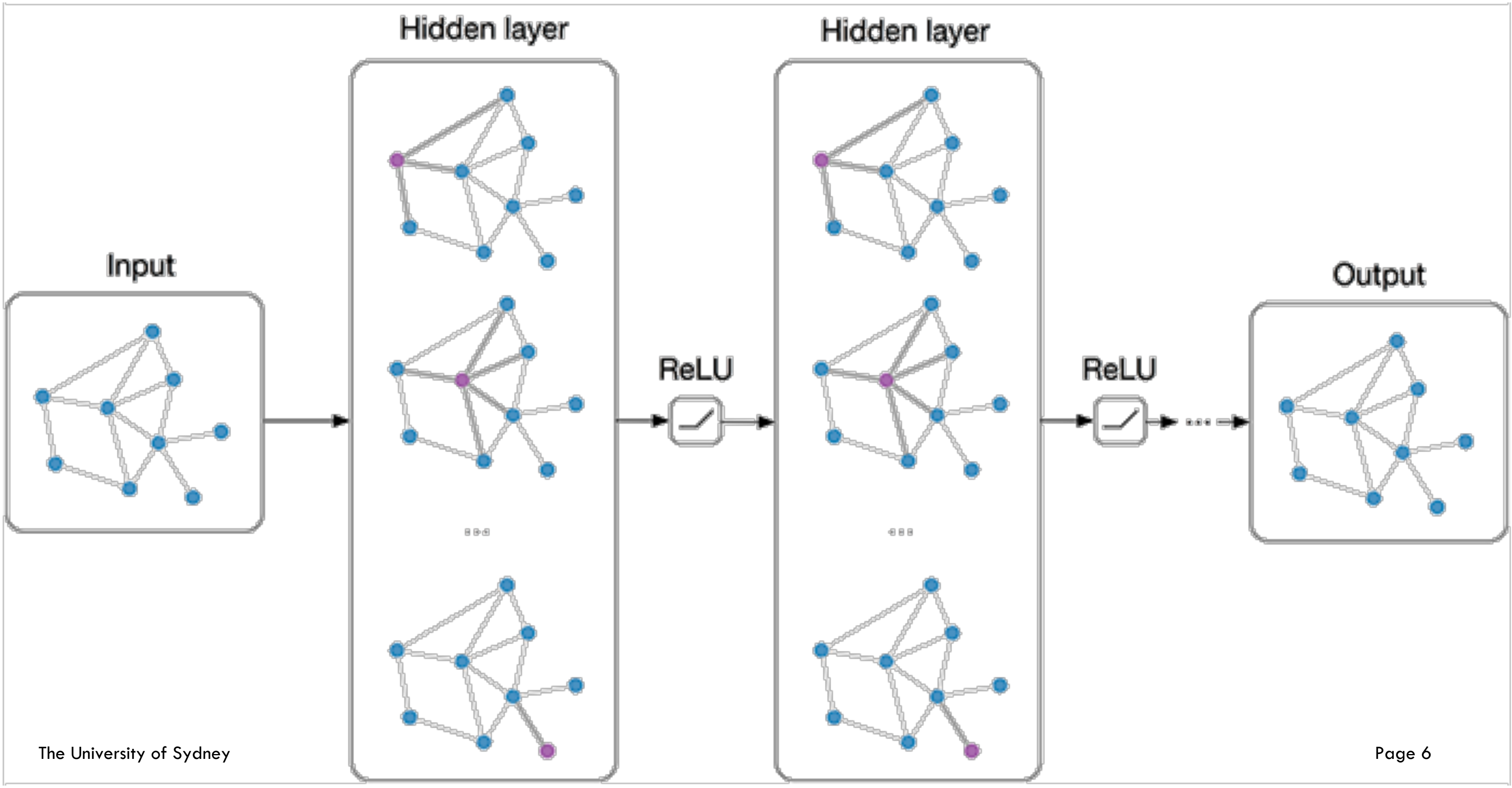
Kernel with varying size

&

Weights with undecided order

Graph Convolutional Networks (GCN)

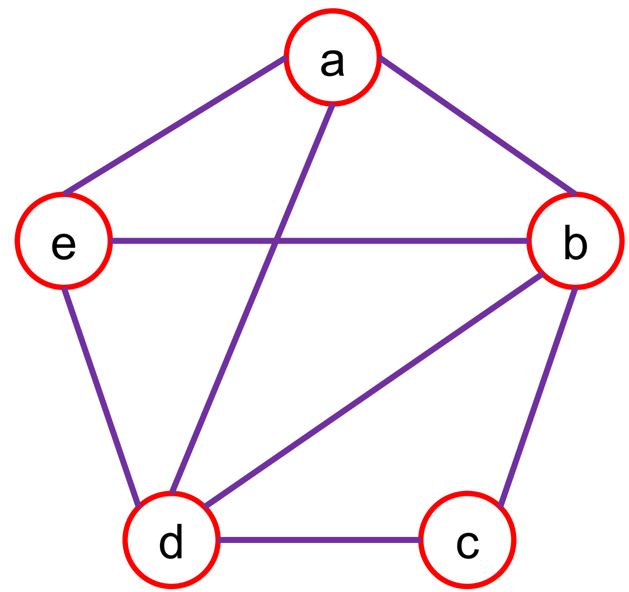
Extract features from the graph data.



Preliminaries: Representing graphs

Graphs consist of

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



G

Formally

G = (V, E)

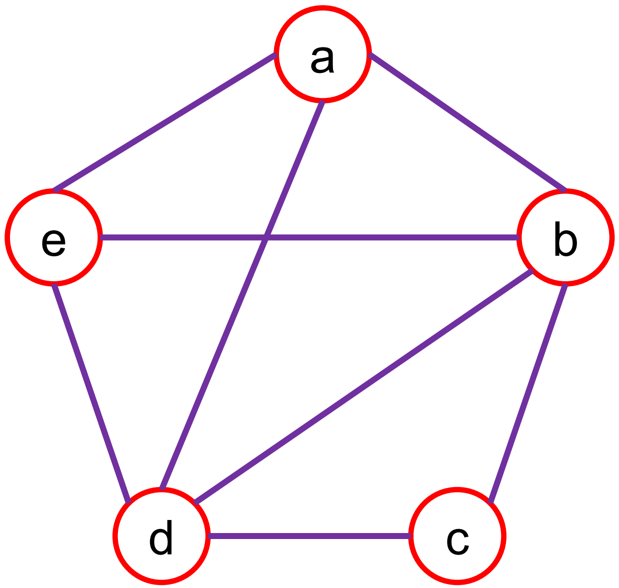
- V = $\{v_i | i = 1 \dots N\}$: Node set, containing |V|=N nodes
- E = $\{e_{ij} | v_i \ is\ connected\ to\ v_j\}$: Edge set

Preliminaries: Representing graphs

Edge representation

- Edge list
- Adjacency matrix
- Incidence matrix

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



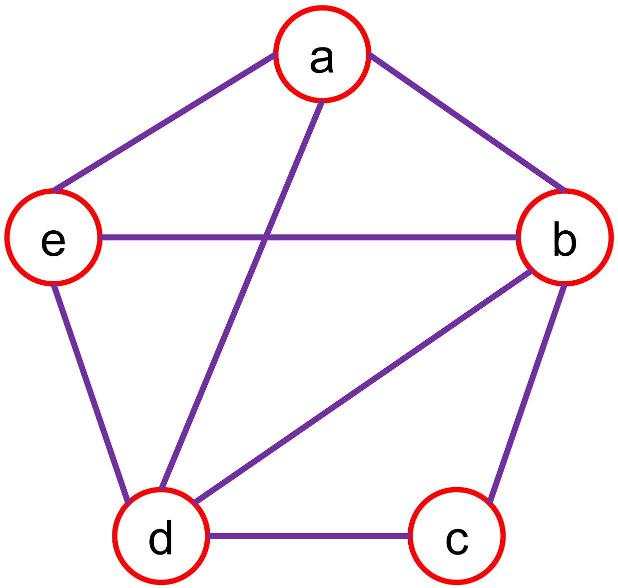
G

Edges:

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

Preliminaries: Representing graphs

2. **Adjacency matrix**



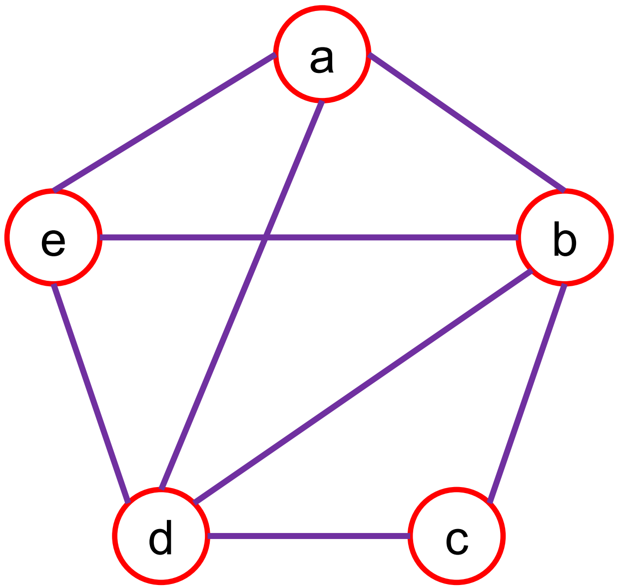
G

$$A= \begin{array}{cc} & \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

$$N = 5$$

Preliminaries: Representing graphs

2. **Adjacency matrix with self connections**



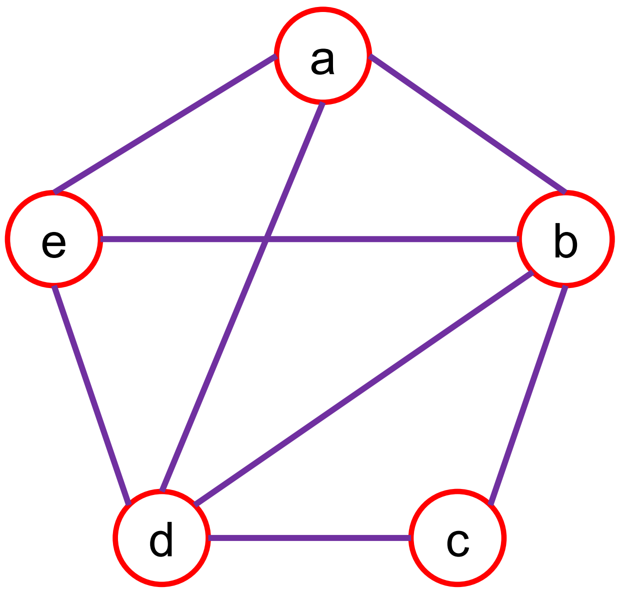
G

$$\hat{A} = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 1 & 1 & 0 & 1 & 1 \\ b & 1 & 1 & 1 & 1 & 1 \\ c & 0 & 1 & 1 & 1 & 0 \\ d & 1 & 1 & 1 & 1 & 1 \\ e & 1 & 1 & 0 & 1 & 1 \end{array}$$

$N = 5$

Preliminaries: Representing graphs

2. **Adjacency matrix (weighted graph)**



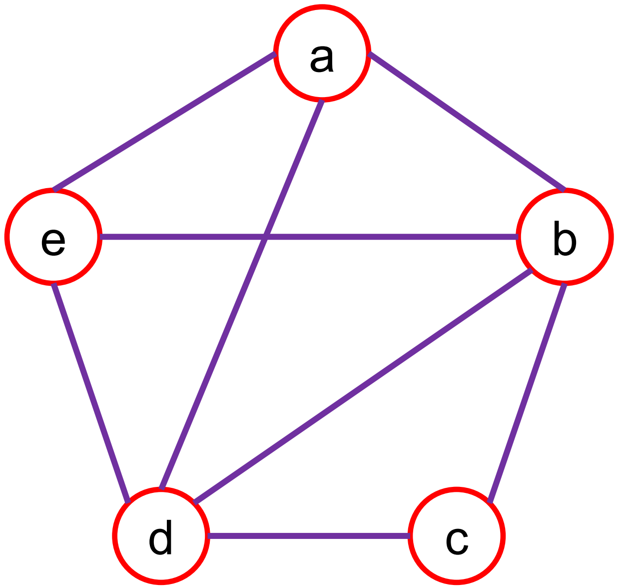
G

$$A_W = \begin{array}{c c} & \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{pmatrix} w_{aa} & w_{ab} & 0 & w_{ad} & w_{ae} \\ w_{ba} & w_{bb} & w_{bc} & w_{bd} & w_{be} \\ 0 & w_{cb} & w_{cc} & w_{cd} & 0 \\ w_{da} & w_{db} & w_{dc} & w_{dd} & w_{de} \\ w_{ea} & w_{eb} & 0 & w_{ed} & w_{ee} \end{pmatrix} \end{array}$$

$N = 5$

Preliminaries: Representing graphs

Degree matrix: Denoting degree of each node



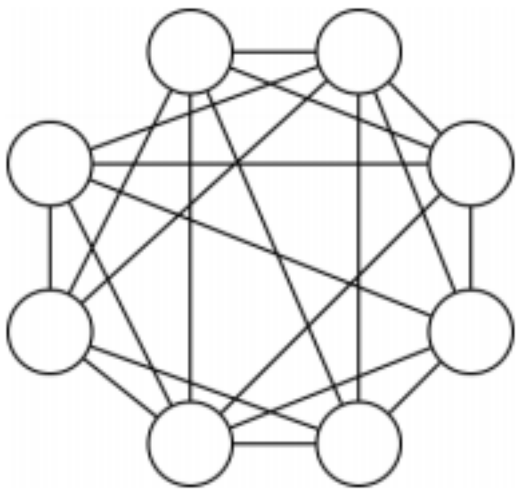
G

$$
D = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{pmatrix}
3 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 0 & 3
\end{pmatrix}
\end{array}
$$

$N = 5$

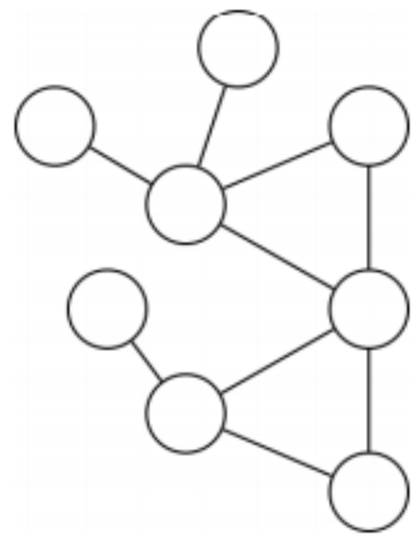
Preliminaries: Basic properties

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



A large fraction of pairs of nodes are connected by edges

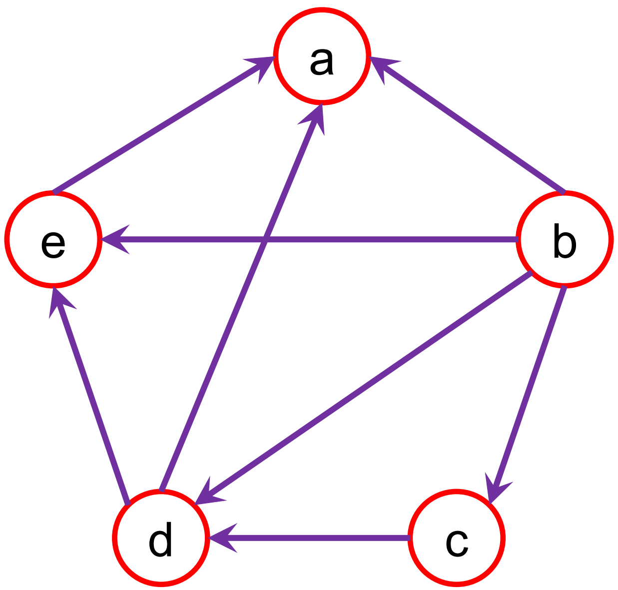
Sparse graph: $|E| \approx |V|$
Or $|E| = O(|V|)$



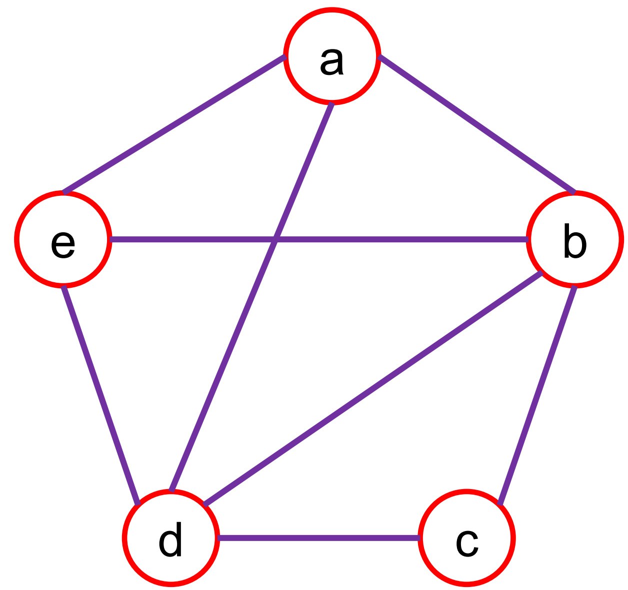
Each node has only a few edges

Preliminaries: Basic properties

Directed graph: Each edge has a direction

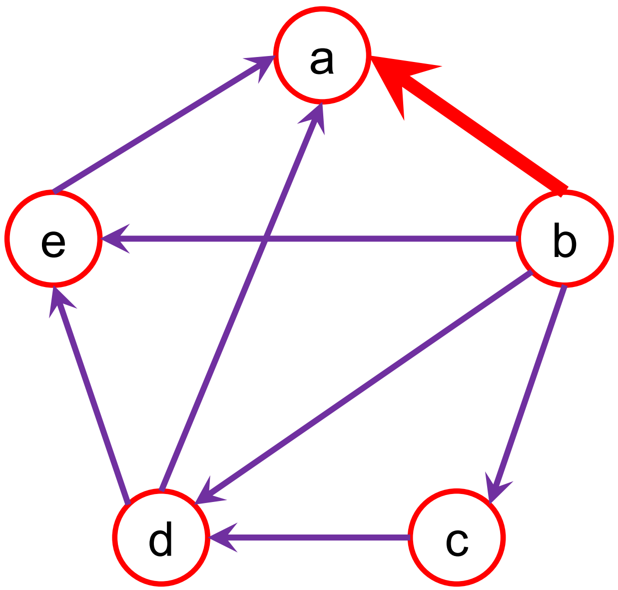


Undirected graph: No direction is associated with edges



Preliminaries: Representing graphs

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



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

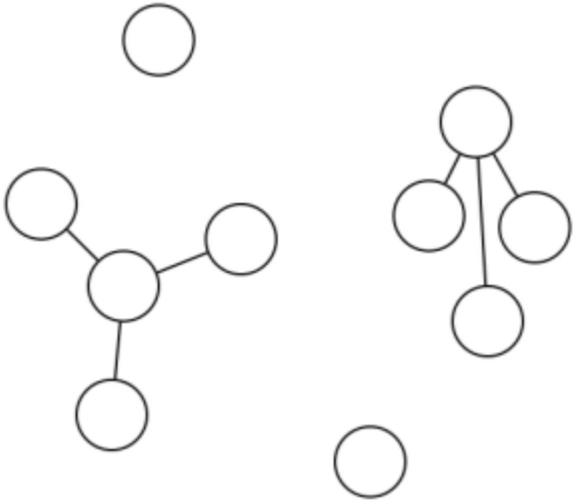
$$N = 5$$

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



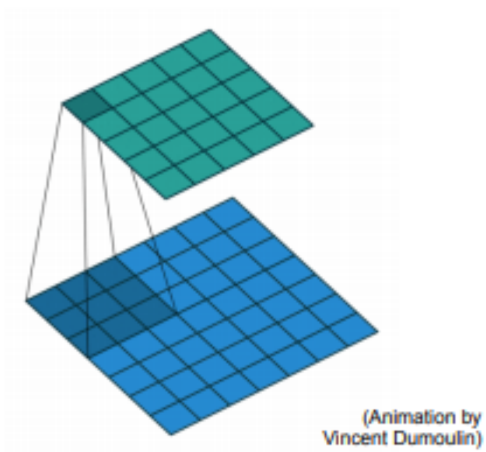
Preliminaries: Representing graphs

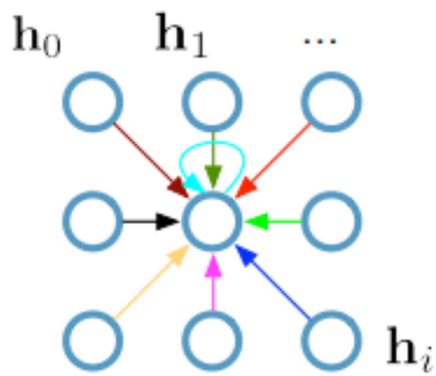
List of basics notations

- G = (V, E)
- V = $\{v_i | i = 1 \ldots N\}$: 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 $\widehat{D} \in R^{N \times N}$ is computed from $\hat{A}$

Convolution in CNN

Single CNN layer with 3x3 filter:





Update for a single pixel:

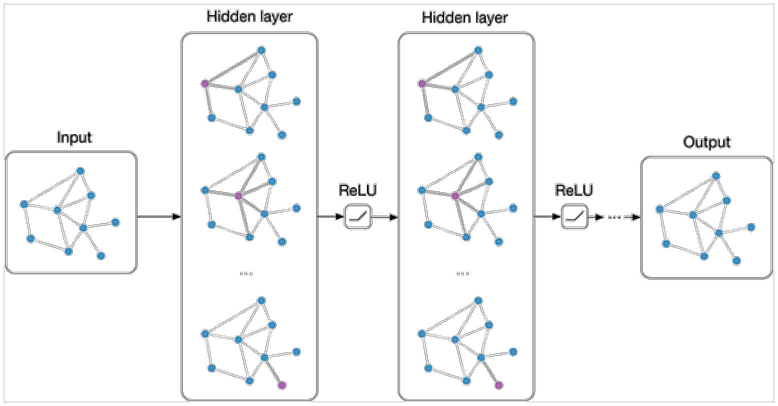
- Transform messages individually $W_i h_i$
- Add everything up $\sum_i W_i h_i$

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

Full update:

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

Convolution in GCN



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

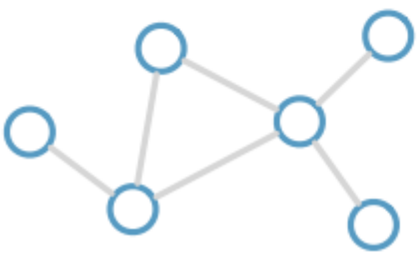
- Adjacency matrix with self-connections: $\hat{A} = A + I_N$
- Degree: number of edges connected to a node
- Degree matrix: $\widehat{D} \in R^{N\times N}$ (computed from $\hat{A}$)

Why

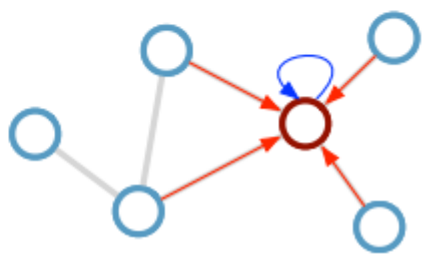
$$H^{(l+1)} = \sigma(\widehat{D}^{-\frac{1}{2}}\hat{A}\widehat{D}^{-\frac{1}{2}}H^{l}W^{l})$$

Graph convolutional networks (Spatial approach)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

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

N_i: neighbor indices

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

Desirable properties:

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

Limitations:

- Requires gating mechanism / residual connections for depth
- Only indirect support for edge features

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

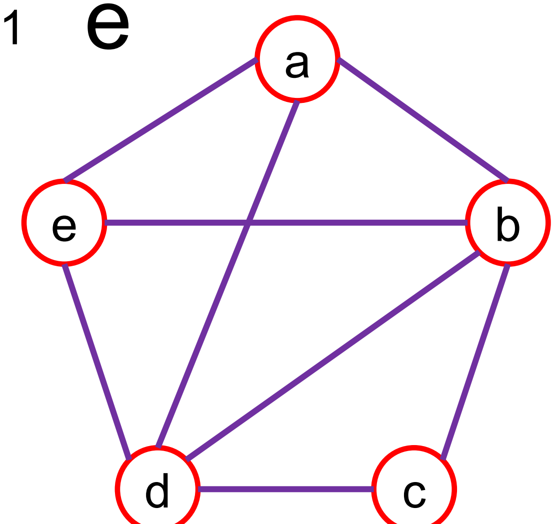
Graph convolutional networks: A concrete model

$$\begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}}_{A} \times \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \Rightarrow \begin{matrix} 0 & \cancel{a} \\ 1 & b \\ 0 & \cancel{c} \\ 1 & d \\ 1 & e \end{matrix} \Rightarrow a' = b + d + e$$

Focus on updated *a’*

Only neighboring nodes of a are retained

Unconnected nodes are masked out



G

Graph convolutional networks: A concrete model

$$\begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \Rightarrow \begin{matrix} 1 & a \\ 1 & b \\ 0 & \cancel{c} \\ 1 & d \\ 1 & e \end{matrix} \Rightarrow a‘=b+d+e$$

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

Self-loops are added into adjacency matrix

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

Central nodes are included in convolution

Normalizing the Feature Representations

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

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

$$\begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0 & 0.33 & 0.33 & 0.33 & 0 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.25 & 0.25 & 0 & 0.25 & 0.25 \end{pmatrix}$$

Normalizing the Feature Representations

$$\begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}^{-\frac{1}{2}} \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}^{-\frac{1}{2}}$$

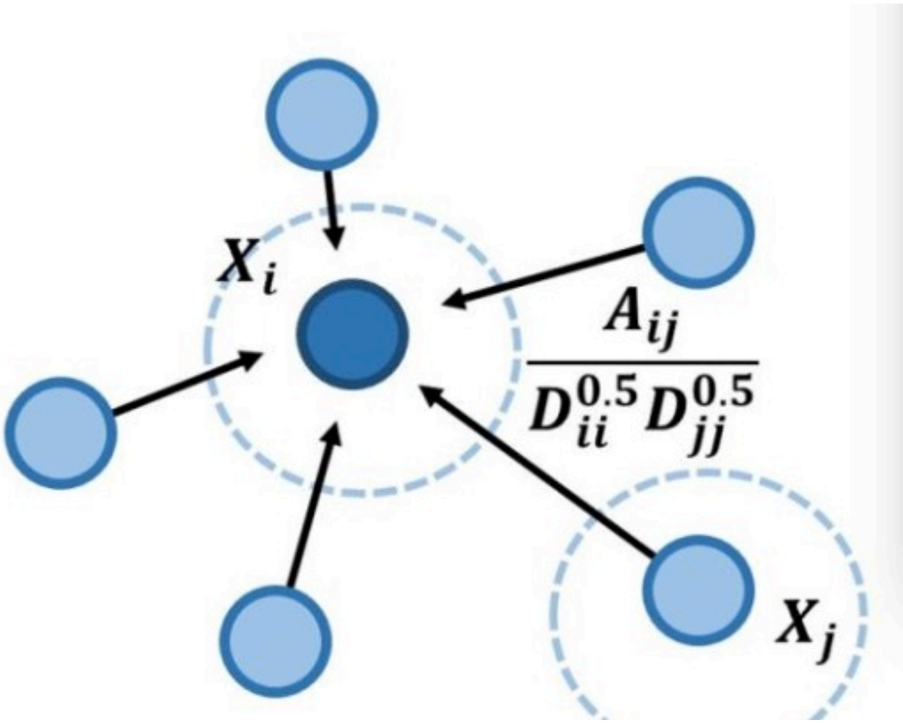
$$\Rightarrow \begin{pmatrix} 0.25 & 0.22 & 0 & 0.22 & 0.25 \\ 0.22 & 0.20 & 0.26 & 0.20 & 0.22 \\ 0 & 0.26 & 0.33 & 0.26 & 0 \\ 0.22 & 0.20 & 0.26 & 0.20 & 0.22 \\ 0.25 & 0.22 & 0 & 0.22 & 0.25 \end{pmatrix}$$

A symmetric normalization

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

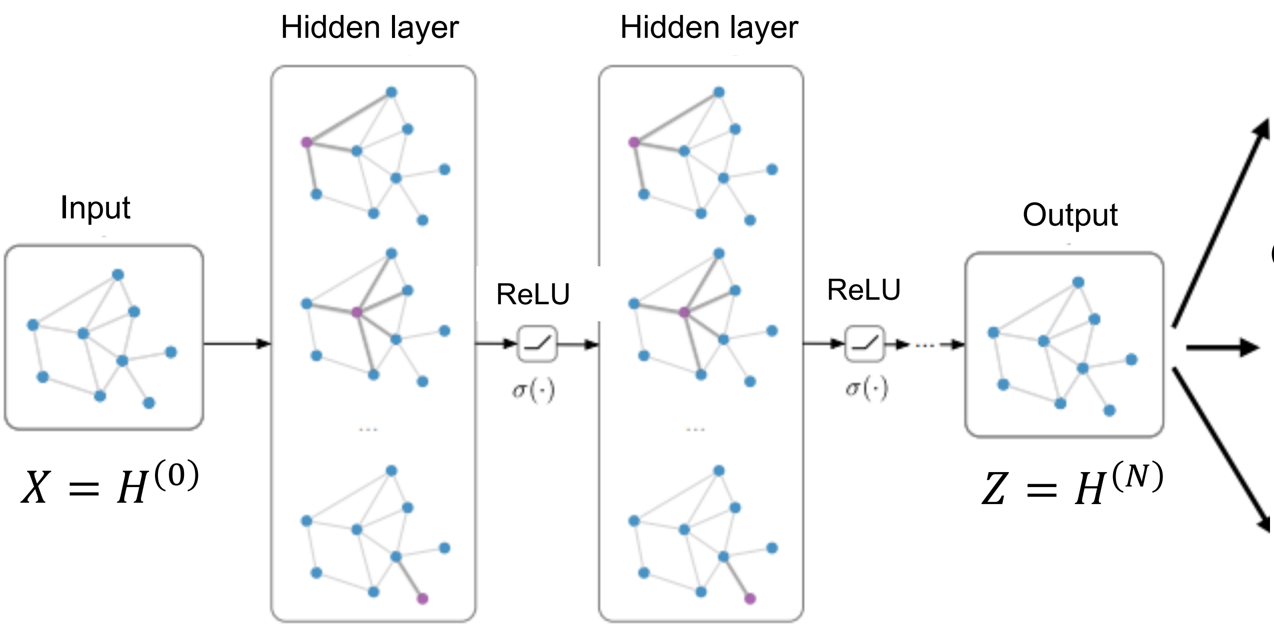
Normalizing the Feature Representations

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



Graph convolutional networks: A concrete model

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



Node classification:

$$softmax(z_n)$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$softmax(\sum_n z_n)$$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

$$p(A_{ij}) = \sigma(z_i^T z_j)$$

Kipf & Welling (NIPS BDL 2016)
"Graph Auto-Encoders"

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

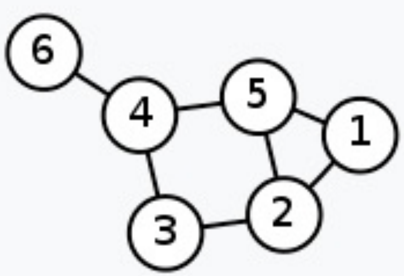
Semi-Supervised Classification with Graph Convolutional Networks (Kipf & Welling)

Spectral approach

Graph convolutional networks (Spectral approach)

Graph Laplacian: $L = D - A$

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

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

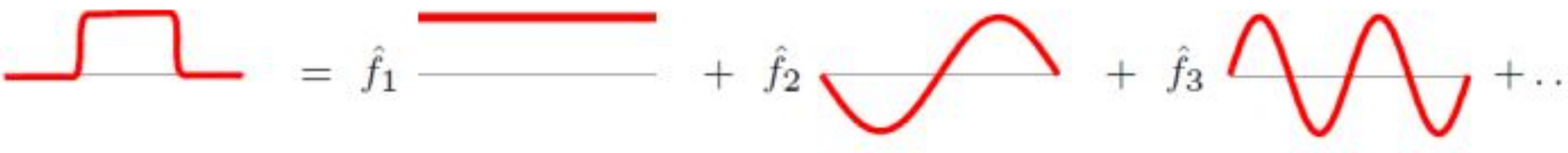
Fourier transform

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

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

Fourier transform



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 $\triangle$:

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

$$Lu = \lambda u$$

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 :

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

Similarly,

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

The inverse graph Fourier transform is then given by:

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

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

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

$$\hat{f} = U^T f \qquad \hat{h} = U^T h \qquad f = U\hat{f}$$

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

Elementwise multiplication

Graph convolutional network

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

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

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

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

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

Graph convolutional network

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

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

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

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

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

$$U g_\theta(\Lambda) U^T = U \sum_{j=0}^{K} \alpha_j \Lambda^j U^T = \sum_{j=0}^{K} \alpha_j L^j$$

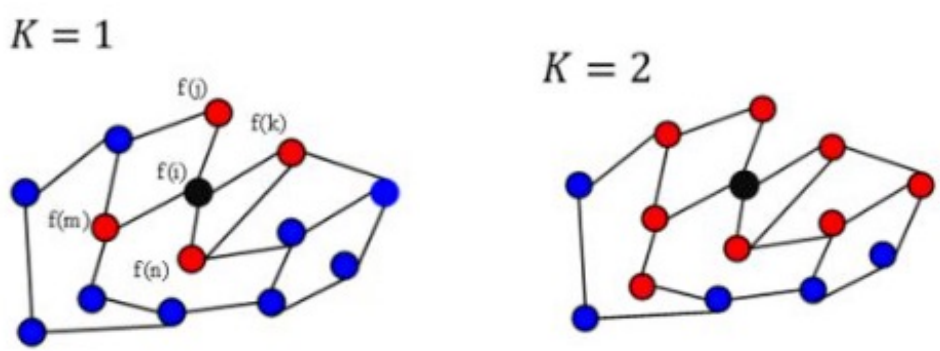
Graph convolutional network

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

- There are only K parameters to train, and usually $K \ll n$. The complexity has been decreased.

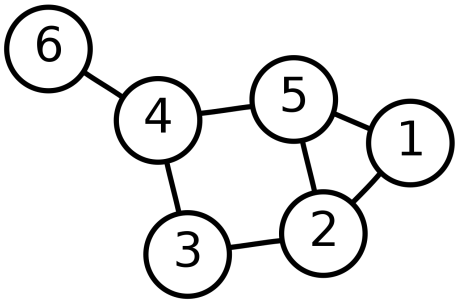
- No need to conduct eigen decomposition. Explicitly depend on the Laplacian matrix L.

- Spatial localization. In particular, K denotes the receptive field.



Graph convolutional network

Local Connectivity + Parameter Sharing



K=1, the convolution filter is

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

K=2, 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=3, the convolution filter is

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

Graph convolutional network

A Recursive formulation for fast filtering

Recall:

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

With Chebyshev expansion, we have

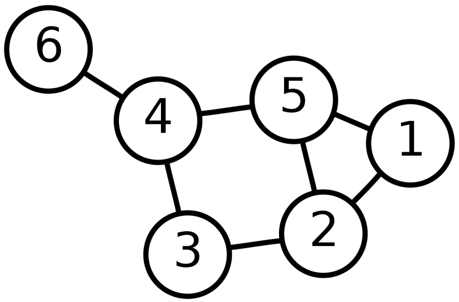
$$g_{\theta'} \approx \sum_{j=0}^{K} \theta'_j T_j(\widetilde{\Lambda}) \qquad \text{where} \qquad \widetilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I_N$$

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

$$y = \sigma(U g_\theta(\Lambda) U^T x) = \sigma\left(U \sum_{j=0}^{K} \theta'_j T_j(\widetilde{\Lambda})\, U^T x\right) = \sigma\left(\sum_{j=0}^{K} \theta'_j T_j(U\widetilde{\Lambda}U^T)\, x\right)$$

$$= \sigma\left(\sum_{j=0}^{K} \theta'_j T_j(\tilde{L})\, x\right)$$

Graph convolutional network



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

K=1, 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=2, the convolution filter is

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

- 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!