Neural Networks on Graphs

[1] T. Kipf and M. Welling, **Semi-supervised classification with graph convolutional networks.** ICLR (2017), citations: 4674.

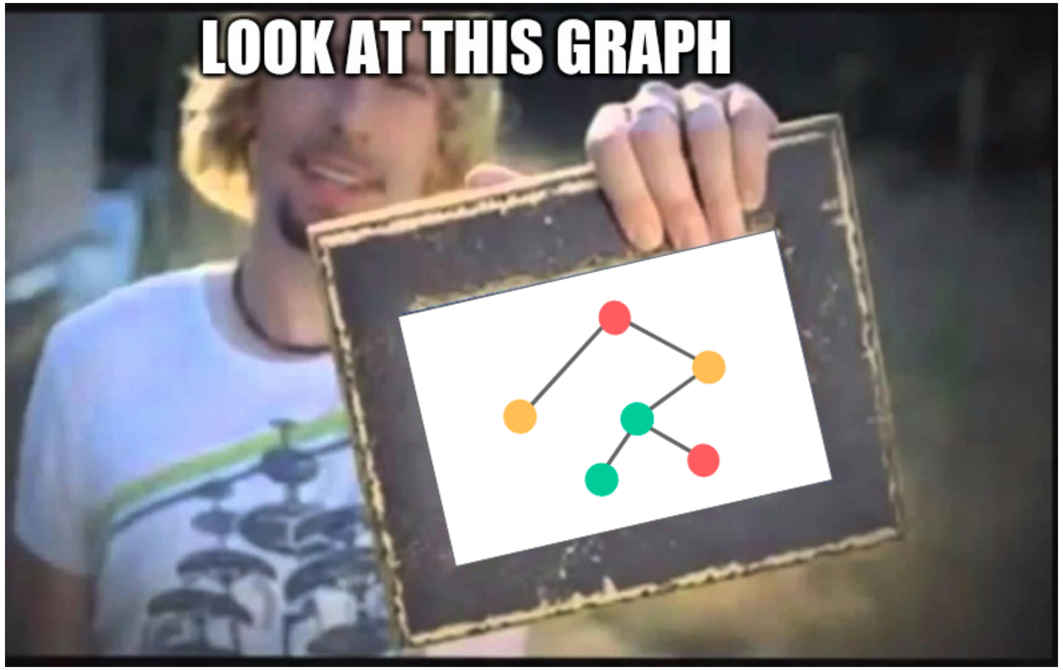
[2] Z. Wu, et al**.** **A comprehensive survey on graph neural networks.** IEEE Transactions on NNLS (2020), citations: 530.

Presenter: Guihong Wan, Sep/2020

Outline

- **Introduction**
  - Graph
  - Learning Tasks on Graphs
  - Graph Networks
- **Part1: Spectral graph convolution**
  - GCN
  - Theory behind GCN
  - More reading
- **Part2: Spatial graph convolution (my next presentation)**
  - GAT ?

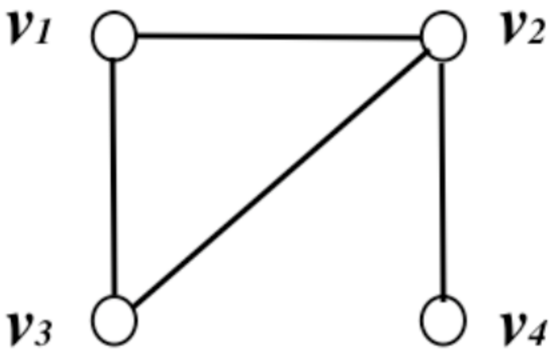
Introduction



Graph

$x_i \in R^F$

N = 4, M = 4



- G(V, E)

V: vertex set with N = |V|;

E: edge set with M = |E|.

- Neighborhood of v: N(v)
- Data matrix: X with size N x F
- Adjacency matrix: A with size N x N
- Degree matrix: D diagonal

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

$$D = \begin{bmatrix} 2 & & & \\ & 3 & & \\ & & 2 & \\ & & & 1 \end{bmatrix}$$

Input of Graph Networks: A, X

Graph (cont.)

- Adjacency matrix: A with size N x N
- Degree matrix: D diagonal
- Laplacian matrix: L = D - A

$$D = \begin{bmatrix} 2 & & & \\ & 3 & & \\ & & 2 & \\ & & & 1 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

degree

Graph (cont.)

- Laplacian matrix: L = D - A
- Normalized Laplacian matrix: $L_1 = D^{-1}\,\mathrm{L}$
- Symmetric Normalized Laplacian matrix:

$$L_2 = D^{-\frac{1}{2}}\,\mathrm{L}D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}\mathrm{A}\,D^{-\frac{1}{2}}$$

$$D = \begin{bmatrix} d_1 & & \\ & d_2 & \\ & & d_3 \end{bmatrix}$$

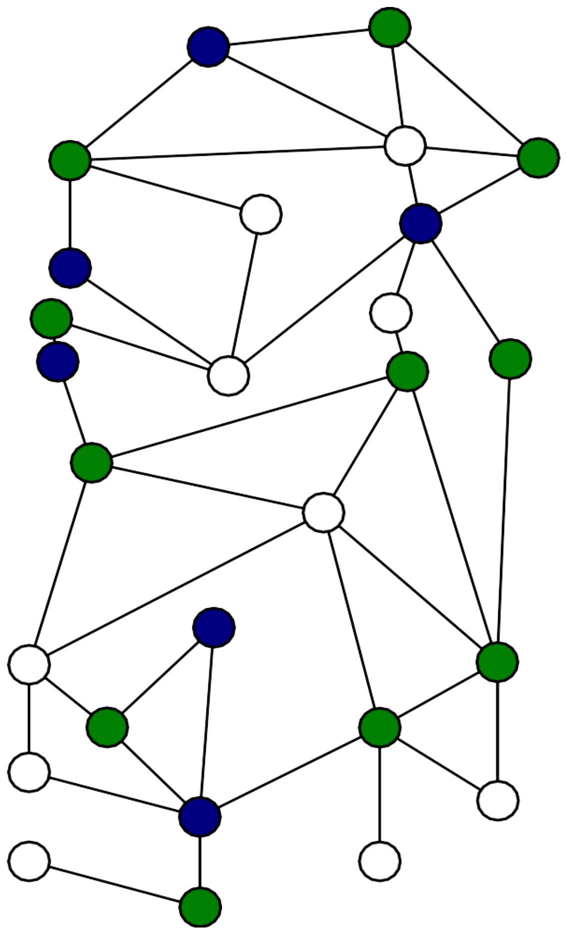
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\begin{bmatrix} \frac{a_{11}}{\sqrt{d_1}\sqrt{d_1}} & \frac{a_{12}}{\sqrt{d_1}\sqrt{d_2}} & \frac{a_{13}}{\sqrt{d_1}\sqrt{d_3}} \\ \frac{a_{21}}{\sqrt{d_2}\sqrt{d_1}} & \frac{a_{22}}{\sqrt{d_2}\sqrt{d_2}} & \frac{a_{23}}{\sqrt{d_2}\sqrt{d_3}} \\ \frac{a_{31}}{\sqrt{d_3}\sqrt{d_1}} & \frac{a_{32}}{\sqrt{d_3}\sqrt{d_2}} & \frac{a_{33}}{\sqrt{d_3}\sqrt{d_3}} \end{bmatrix}$$

Learning Tasks

For example:

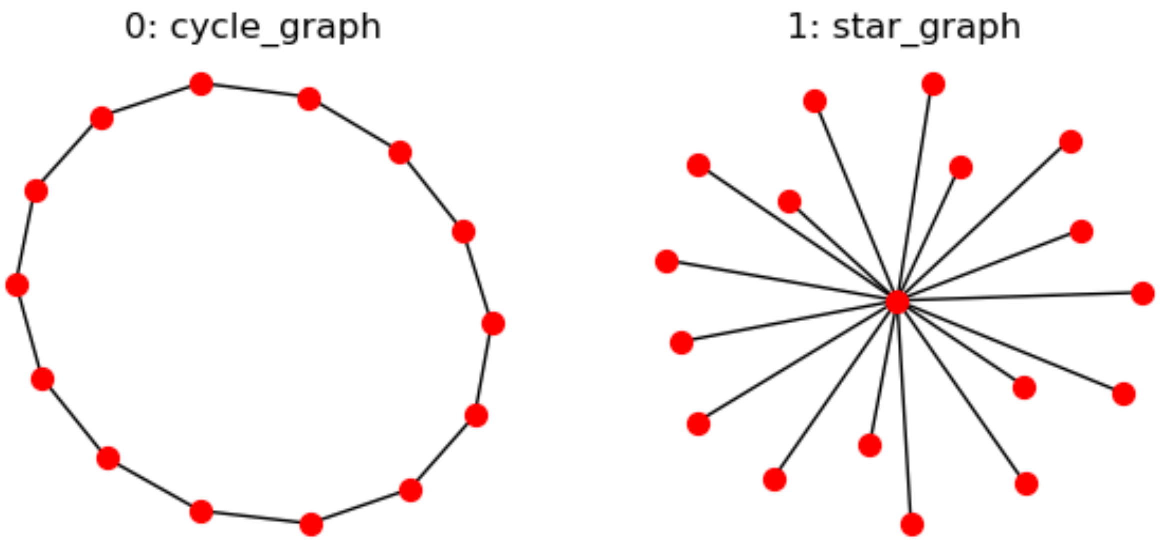
- Node level
  - Node classification
  - Node representation
- Graph level
  - Graph classification
  - Graph representation



Learning Tasks (cont.)

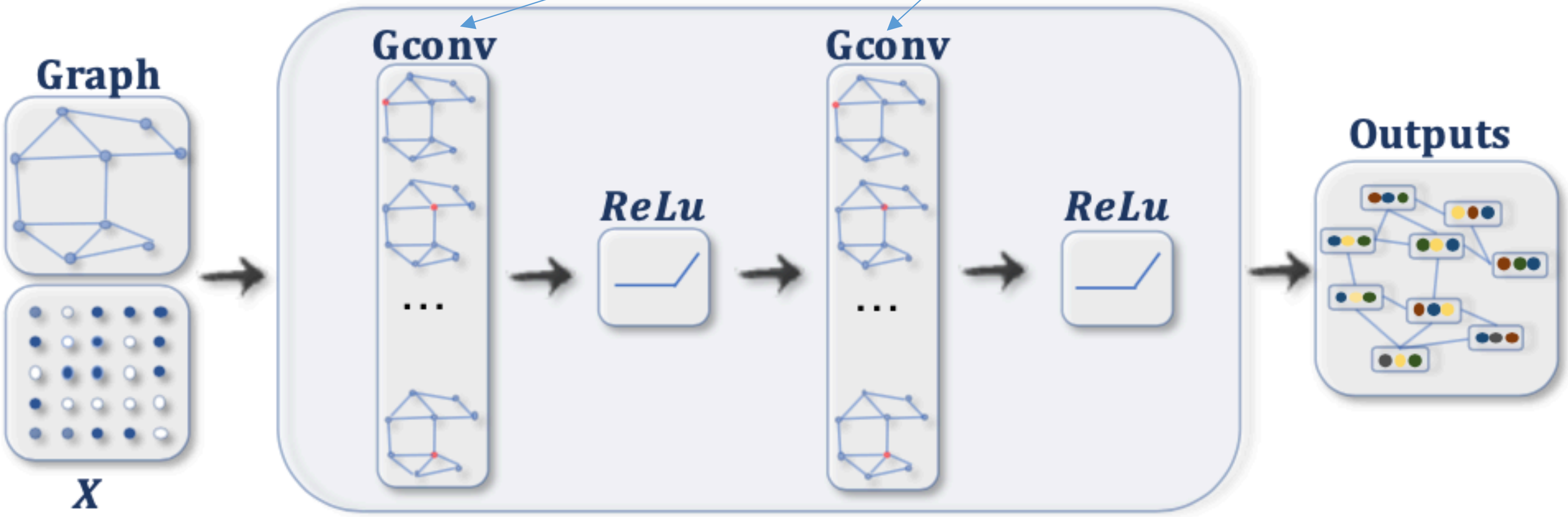
For example:

- Node level
  - Node classification
  - Node representation
- Graph level
  - Graph classification
  - Graph representation



Graph networks

Graph convolution layer

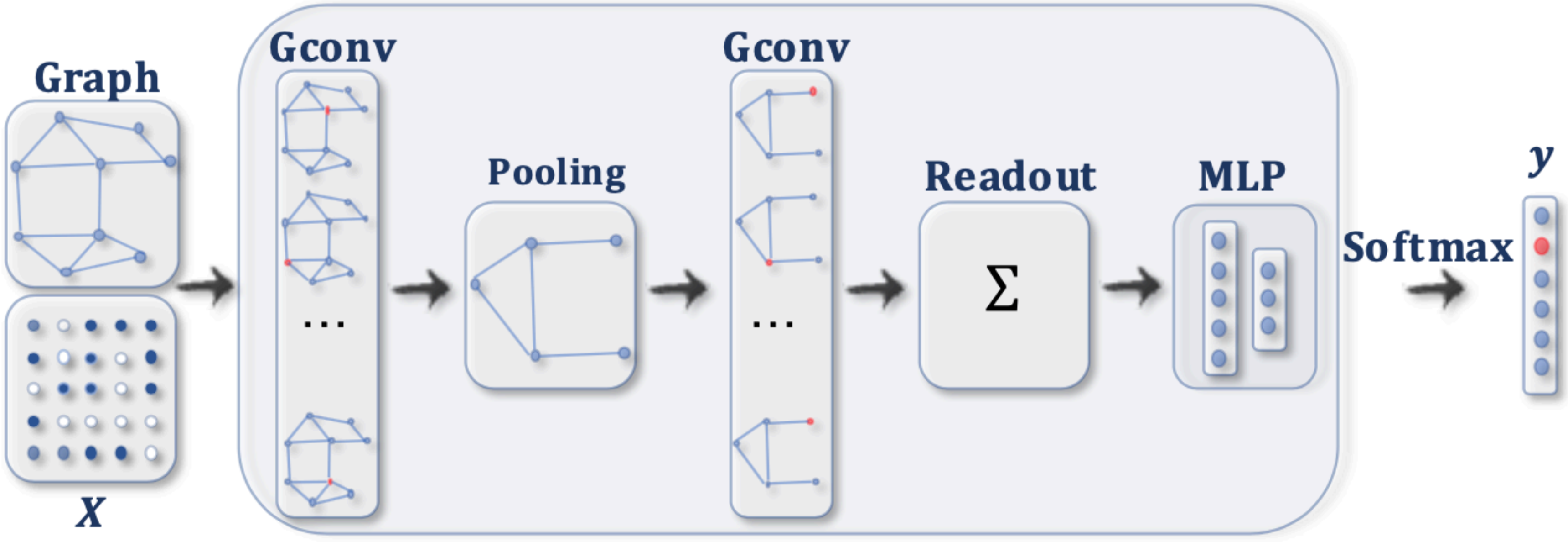


Task : node embedding/classification $\mathbf{f}(A_{N\times N}, X_{N\times F}) \rightarrow \tilde{X}_{N\times R}\ or\ \tilde{Y}_{N\times C}$

Main idea: learn a node v's representation

by aggregating its own feature x_v and its neighbors' feature x_u, for all $u \in N(v)$.

Graph networks (cont.)

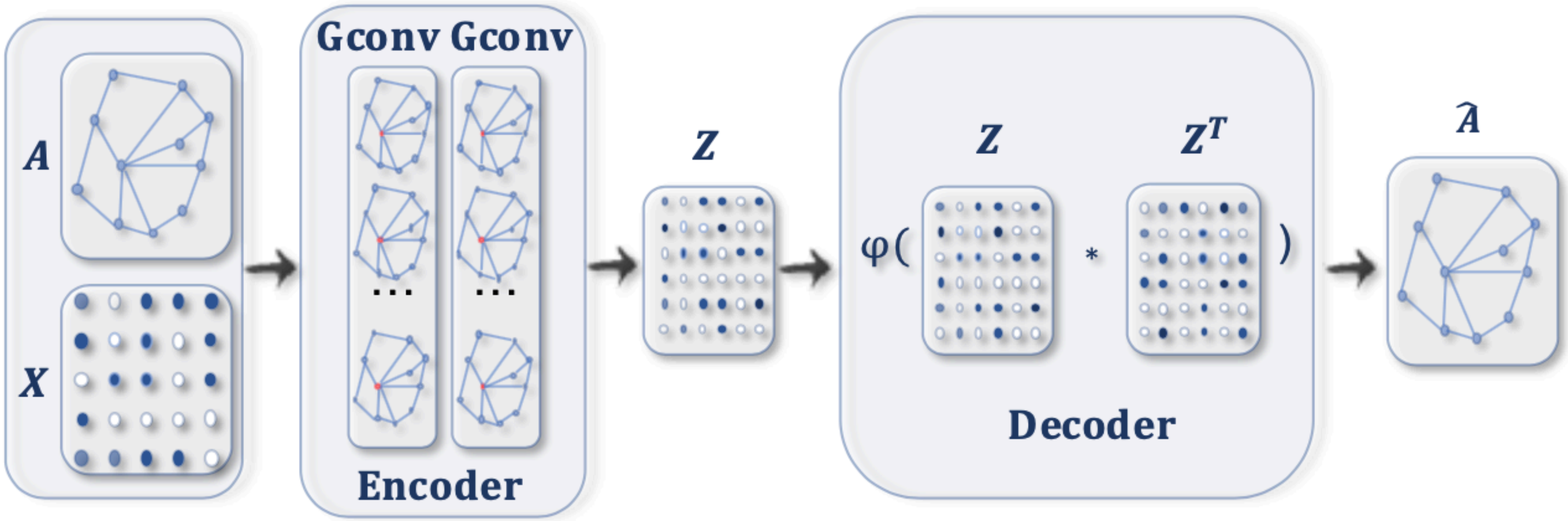


Task : graph classification

Pooling: coarsen the graph

Readout: to summarize to get the graph representation

Graph networks (cont.)



Graph Autoencoder : unsupervised learning

$$f(A_{N\times N}, X_{N\times F}) \rightarrow Z_{N\times R}$$

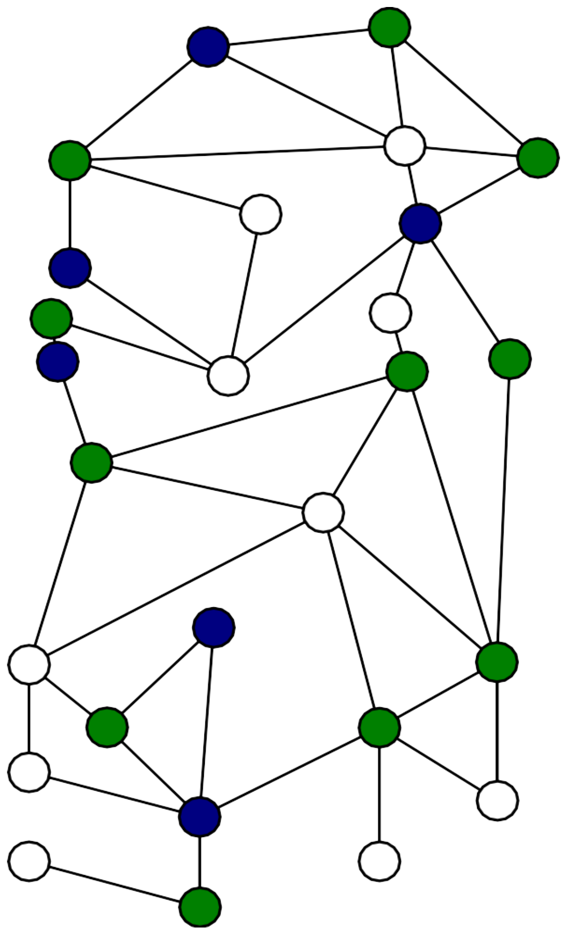
Spectral graph convolution

- **Problem**
- **How to do**
- **Why (theory behind)**

[1] T. Kipf and M. Welling, **Semi-supervised classification with graph convolutional networks.** ICLR (2017), citations: 4674.

Graph Convolution Network (GCN)

- **Task:** Node classification
  - Input: (A, X, partial Y)
  - Output: label of each node
- Semi-supervised learning
- A first-order approximation of spectral graph convolution



GCN (cont.)

The l^{th} GCN Layer: $\boldsymbol{H^{l+1} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{l}W^{l})}$

$\boldsymbol{H^0 = X}$, otherwise the output of previous layer.

$\boldsymbol{\widetilde{A} = A + I_N}$, $\quad \boldsymbol{\widetilde{D} = \sum_j \widetilde{A}_{ij}}$

$\boldsymbol{W}$: what to learn.

GCN (cont.)

The l^{th} **GCN Layer:** $H^{l+1} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{l}W^{l}\right),\ \widetilde{A} = A + I_N$

Understand intuitively:

if there is no normalization, then $H^{l+1} = \sigma(\widetilde{A}H^{l}W^{l})$

if $l = 0$, then $H^{1} = \sigma(\widetilde{A}XW^{0})$

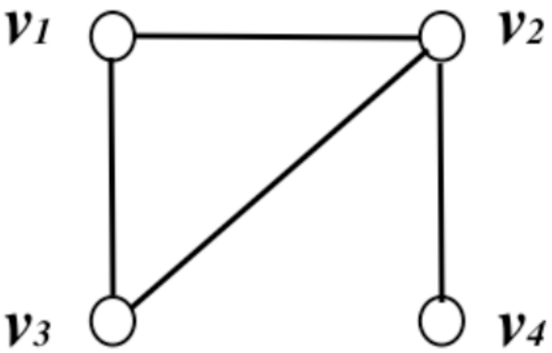
if there is no modification of adjacency matrix , then

$H^{1} = \sigma(AXW^{0})$

GCN (cont.)

Understand intuitively:

$$H^1 = \sigma(AXW^0)$$



$$\overset{A}{\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}} \overset{X}{\begin{bmatrix} x_{11} & x_{12} & \dots x_{1F} \\ x_{21} & x_{22} & \dots x_{2F} \\ x_{31} & x_{32} & \dots x_{3F} \\ x_{41} & x_{42} & \dots x_{4F} \end{bmatrix}} \begin{matrix} \leftarrow x_1 \\ \leftarrow x_2 \\ \leftarrow x_3 \\ \leftarrow x_4 \end{matrix}$$

$$AX = \begin{bmatrix} x_{21} + x_{31} & x_{22} + x_{32} & \dots x_{2F} + x_{3F} \\ & \vdots & \\ & . & \end{bmatrix}$$

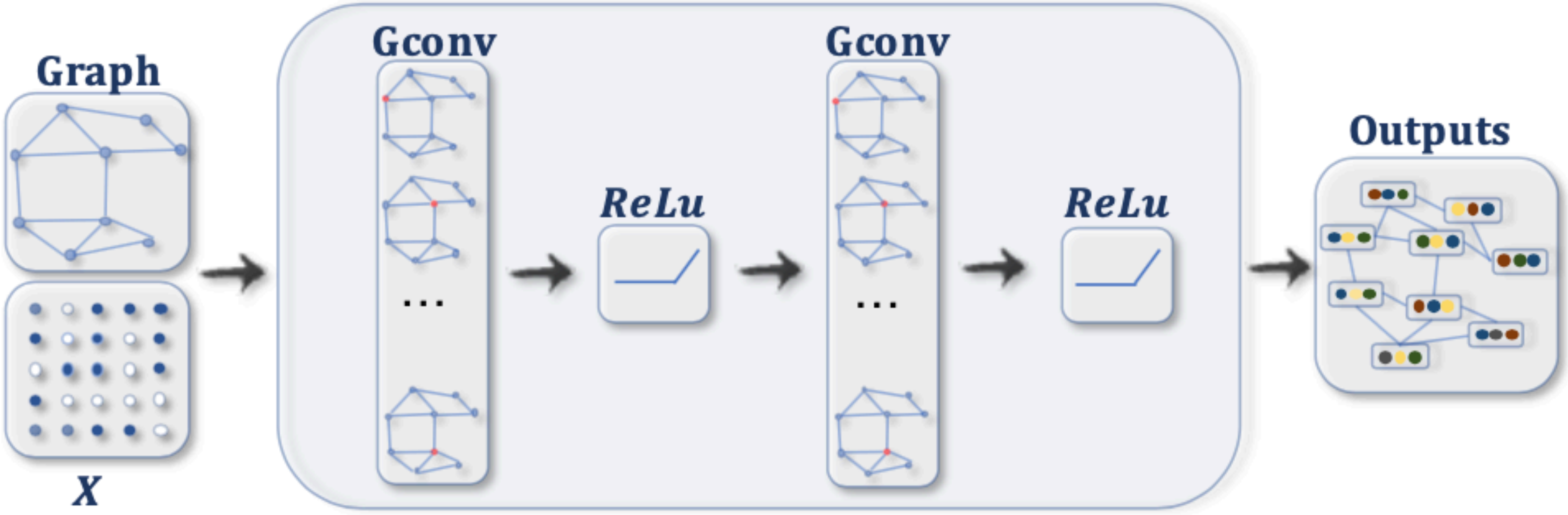
Main idea: learn a node v's representation
by aggregating its own feature x_v and its neighbors' feature x_u,
for all $u \in N(v)$.

Normalization: the multiplication will completely change the scale of the features.

GCN (cont.)

Implementation:

hidden = tf.sparse_tensor_dense_matmul(self.adj_norm, self.attrs @ w) + b



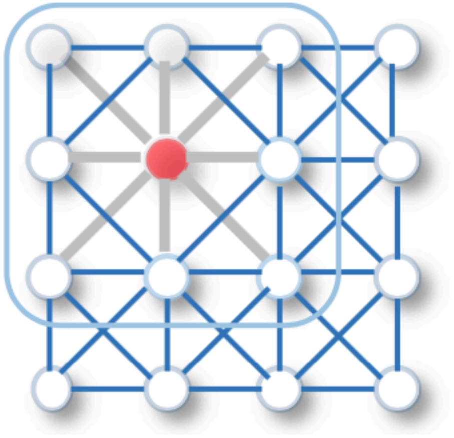
Theory behind GCN

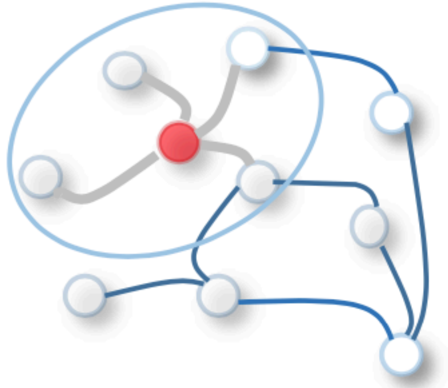
$$H^{l+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}W^{l}\right)$$

- GCN is a spectral-based graph convolution network.
- "spectral": eigen-decomposition of the graph symmetric normalized Laplacian L.
- A first-order approximation of spectral graph convolution.

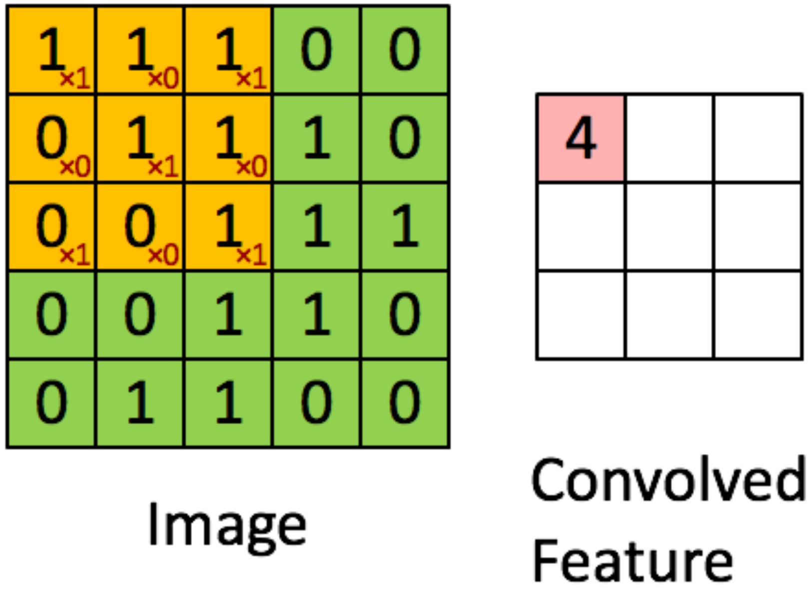
Where is the "spectral" part?

Graph Convolution





What to learn: the filter W.



- Order neighbors VS unordered neighbors
- A fixed size VS variable size

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Spectral Graph Convolution

Given $L = U\Lambda \mathrm{U}^{\mathrm{T}}$, and a signal vector x with size N (a scaler for a node)

- The graph Fourier transform: $\hat{x} = \mathrm{U}^{\mathrm{T}}x$
- The inverse graph Fourier transform: $\mathrm{x} = U\hat{x}$

Given the filter y

- The convolution between x and y is:

  $$x * y = U\left(\mathrm{U}^{\mathrm{T}}x \odot \mathrm{U}^{\mathrm{T}}y\right) = \mathrm{U}(\hat{x} \odot \hat{y})$$

  $\odot$ denotes the element-wise product.

Spectral Graph Convolution

- The convolution between x and y is:
  $x * y$
  $= \mathrm{U}(\hat{x} \odot \hat{y})$
  $$= \mathrm{U}\begin{bmatrix} \hat{x}_1\hat{y}_1 \\ \vdots \\ \hat{x}_N\hat{y}_N \end{bmatrix} = \mathrm{U}\begin{bmatrix} \hat{y}_1 & & \\ & \ddots & \\ & & \hat{y}_N \end{bmatrix}\begin{bmatrix} \hat{x}_1 \\ \vdots \\ \hat{x}_N \end{bmatrix} = U\mathrm{diag}(\hat{y})\,\hat{x}$$
  Hence:
  $x * y = U\mathrm{diag}(\hat{y})\,\hat{x}$
  - No matter what form the filter y is, its representation in the spectral/frequency domain will be a function of frequency $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_N)$: $\hat{y}(\Lambda)$.

Spectral Graph Convolution

- $x * y = U\text{diag}(\hat{y})\ \hat{x}$
- $\hat{y}(\Lambda)$: a function of frequency $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_N)$.

If denote $\mathrm{g}_\theta(\Lambda) = \text{diag}(\hat{y}(\Lambda))$, then

$$x * y = U\ \mathrm{g}_\theta(\Lambda)\ U^T x$$

The spectral-based graph convolution networks follow this definition. The key difference lies in the choice of the filter g_θ.

g_θ for GCN

- The $g_\theta(\Lambda)$ can be approximated by a truncated expansion in terms of Chebyshev Polynomials $T_K()$ up to K^{th} order:

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$$

where $\tilde{\Lambda} = \frac{1}{\lambda_{max}}\Lambda - I_N$, θ_k is the Chebyshev Coefficient.

$$x * y = U\, g_\theta(\Lambda)\, U^T x \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})x$$

g_θ for GCN (cont.)

- The Chebysheve Polynomials are recursively defined as:

$T_k(z) = 2zT_{k-1}(z)$ - $T_{k-2}(z)$, with $T_0(z) = 1$ and $T_1(z)$ = z.

- We could derive: $T_0(\tilde{\Lambda}) = I_N, T_1(\tilde{\Lambda}) = \tilde{\Lambda} = \frac{1}{\lambda_{max}}\Lambda - \mathrm{I_N}$
- GCN uses the first-order approximation: $K = 1, \lambda_{max} = 2$:

$$\mathrm{x} * y \approx [\theta_0 T_0(\tilde{\mathrm{L}}) + \theta_1 T_1(\tilde{\mathrm{L}})]x$$
$$= (\theta_0 - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x$$
$$= \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x \text{ } if \text{ } \theta = \theta_0 = -\theta_1.$$

θ is a vector with size N ← need to be learned

g_θ for GCN (cont.)

$$\boldsymbol{H^{l+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}W^{l}\right)}$$

$$x * y \approx \theta\left(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)x$$

In above formula, x is a vector with size N (a scaler for a node.)
However, X is with size N x F (a vector for a node)
Hence, in matrix form:

$$X * Y \approx \left(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)XW$$

W is a matrix with size N x F' ← need to be learned

More reading

- [1] M. Defferrard, et al, "**Convolutional neural networks on graphs with fast localized spectral filtering**," *NIPS* (2016), citations: 2234.
- [2] J. Bruna, et al, "**Spectral networks and locally connected networks on graphs**," *ICLR* (2014), citations: 1538.
- [3] R. Li, et al, "**Adaptive graph convolutional neural networks**," *AAAI* (2018), citations: 152.

Thank you!