Math Behind GCN

Zhiping Xiao

University of California

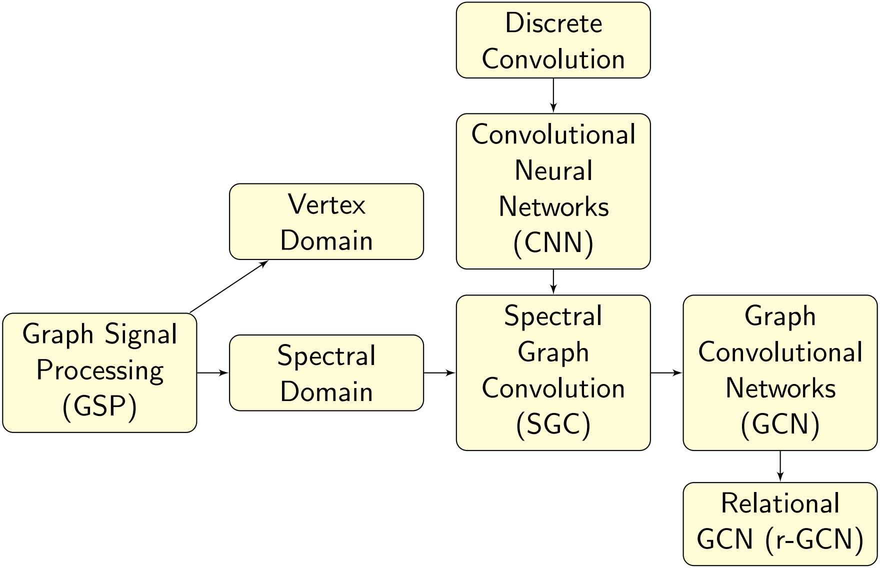
patriciaxiao@g.ucla.edu

December 04, 2018

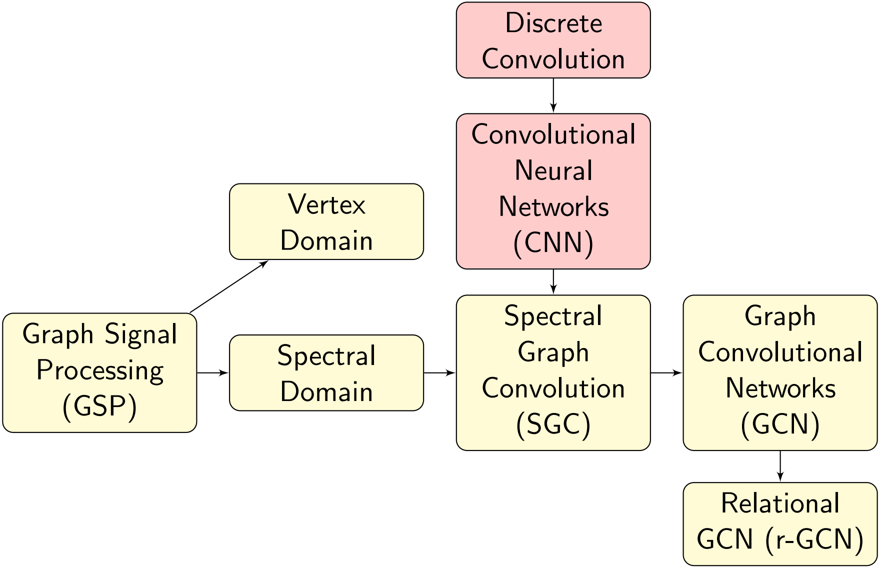
Contents

1. Overview
2. The Emerging Field of Signal Processing on Graphs (IEEE Signal Processing Magazine, Volume: 30 , Issue: 3 , May 2013)
3. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (NIPS 2016)
4. Semi-Supervised Classification with Graph Convolutional Networks (ICLR 2017)
5. Modeling Relational Data with Graph Convolutional Networks (ESWC 2018, Best Student Research Paper)
6. References

Overview

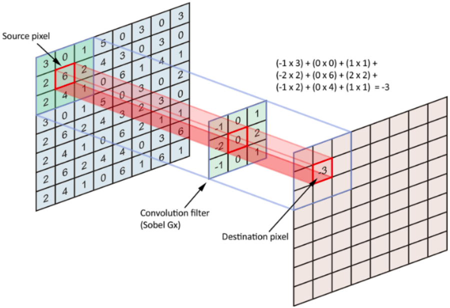

Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

Convolution → CNN


Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

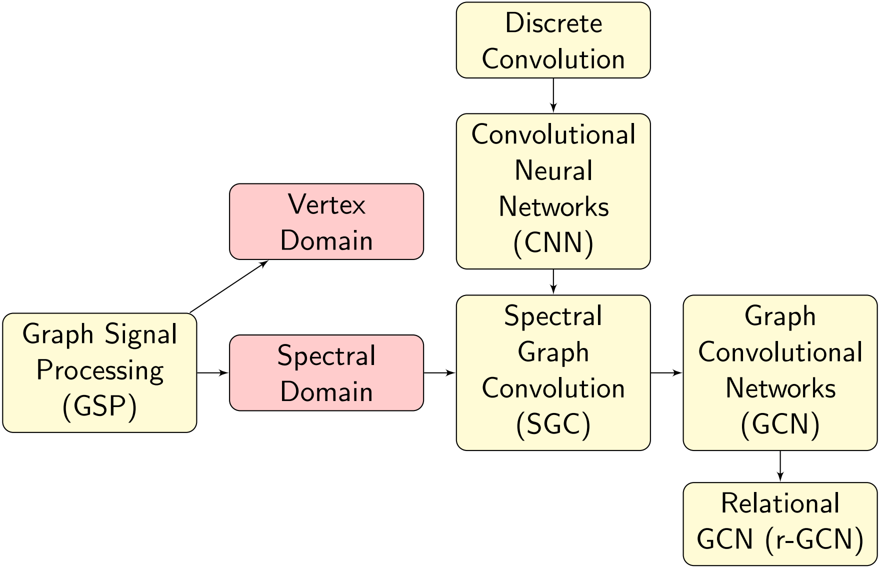
Convolution → CNN

- Convolution: $(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n-\tau)d\tau$
- Discrete Convolution: $(f * g)(a, b) = \sum_h \sum_k f(h, k)g(a-h, b-k)$
    - In a way, **weighed** sum.
- g → the graph; f → the filter (kernel)
- *input* → $\{convolution, activation, pooling\}^*$ → *output*
  (e.g. classification: *output* → *flatten* → *fully connected* → *softmax*)
- Weight to be learned.



(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

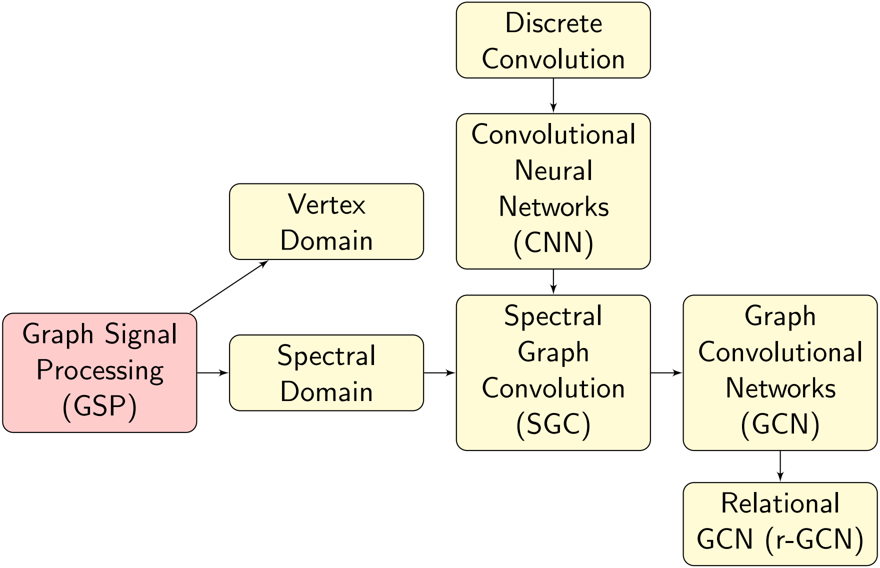
Vertex Domain v.s. Spectral Domain


Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

Vertex Domain v.s. Spectral Domain

- Euclidean structure:
    - matrix
    - e.g. image
    - Applying CNN, feature map could be extracted from the receptive field by using the kernel.
- Non-Euclidean structure:
    - graph $G = \langle V, E \rangle$
    - Couldn't apply CNN: *n_neighbors* is different, no fixed-sized kernel could apply; *translation invariance of discrete convolution is not guaranteed*.
    - How to extract features in general graphs?
        - **Vertex (spacial) domain**: (1) define receptive field (deal with each node); (2) extract features of neighbors (non-convolutional way)
        - **Spectral domain**

Graph Fourier Transformation

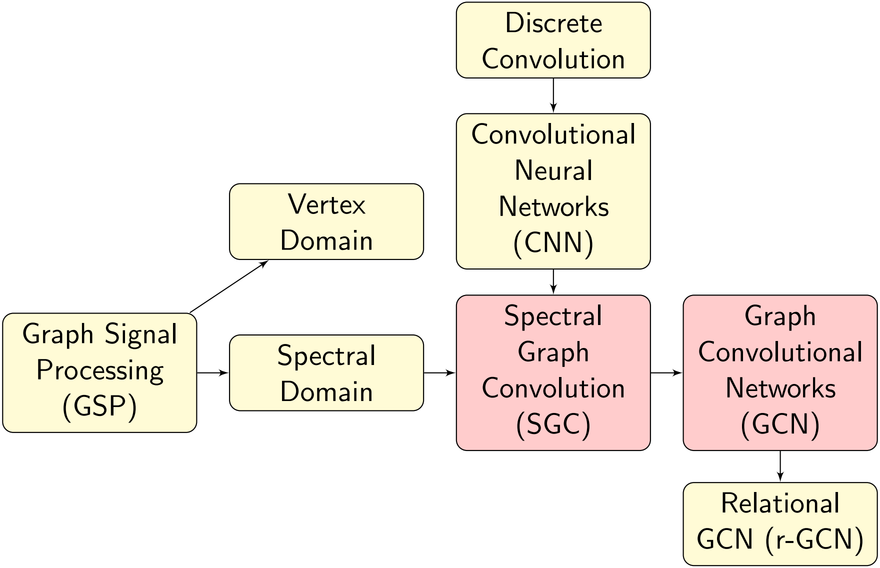

Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

Graph Signal Processing (GSP)

Brief Intro

- Graph Signal Processing makes it possible to do convolution on generalized graphs. e.g. Graph Fourier transformation enables the formulation of fundamental operations on graphs, such as spectral filtering of graph signals.
- **Convolution in the vertex domain is equivalent to multiplication in the graph spectral domain.**

SGC → GCN

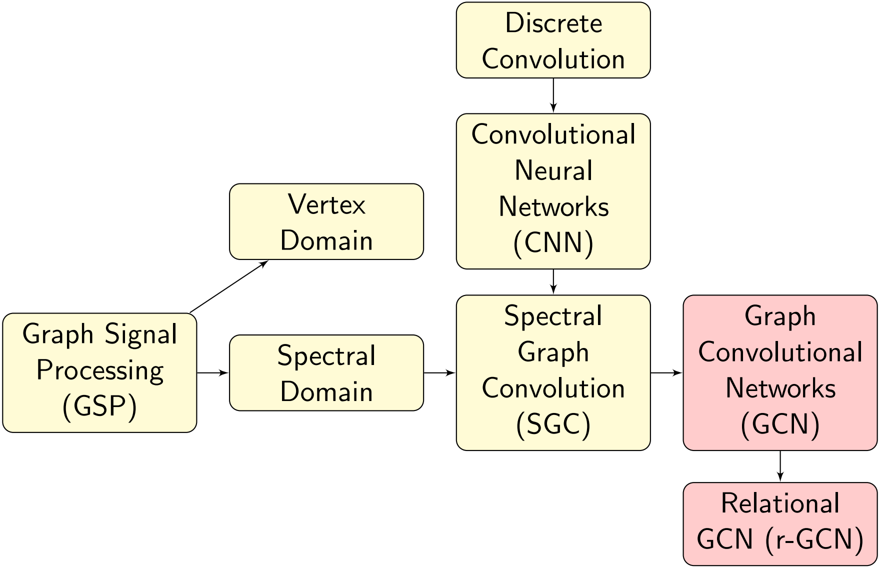

Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

SGC → GCN

Brief Intro

Graph Convolutional Networks (GCN) is a localized first-order approximation of Spectral Graph Convolution.

GCN → r-GCN

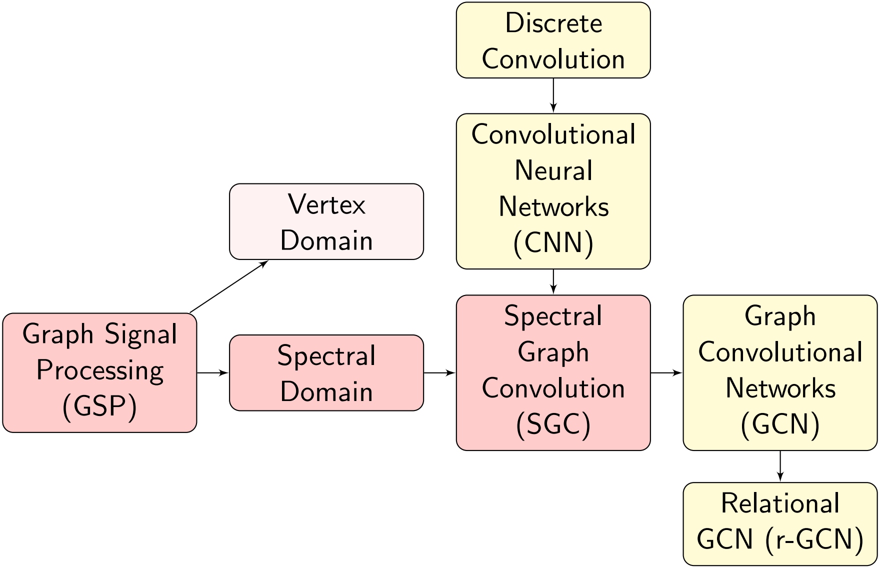

Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

GCN → r-GCN

Brief Intro

Relational Graph Convolutional Networks (r-GCN) is an extension of Graph Convolutional Networks (GCN) on relational data.

The Emerging Field of Signal Processing on Graphs



Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

The Emerging Field of Signal Processing on Graphs

Objective

- To offer a tutorial overview of graph-based data analysis (especially arbitrary graphs, weighted graphs), from a signal-processing perspective.
- Examples of graph signals covers: transportation networks (e.g. epidemic), brain imaging (e.g. fMRI images), machine vision, automatic text classification.
- Irregular, high dimensional data domain; goal is to extracting information efficiently.
- **Constructing, analyzing, manipulating graphs, as opposed to signals on graphs.**

The Emerging Field of Signal Processing on Graphs

What corresponding relations can we get between signal processing tasks and graph data?

Classical Discrete-Time Signal	Graph Signal
N samples with N values	N vertices (/ data points) with N values
a classical discrete-time signal with N samples: vector in $\mathbb{R}^N$	a graph signal with N vertices: vector in $\mathbb{R}^N$

Modulating a signal on the real line by multiplying by a complex exponential corresponds to translation in the Fourier domain[1].

What makes the problem settings different?

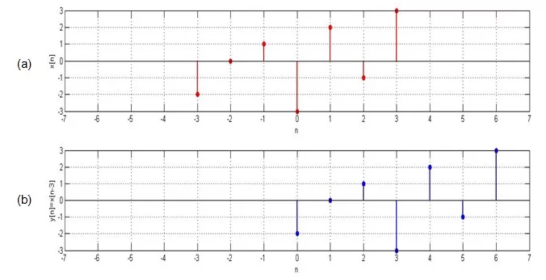
- A discrete-time signal ignores key dependencies in irregular data domain.
- Different in fundamental, non-trivial properties.

[1] Modulation and Sampling. Click here link to some EE slides.

The Emerging Field of Signal Processing on Graphs

Challenge # 1: Shifting

Weighted graphs are irregular structures that lack a shift-invariant notion of translation. $f(t-n)$ no longer makes sense. [a][b]



[a] Special case: ring graph, where Laplacians are circulant and the graphs are highly regular.

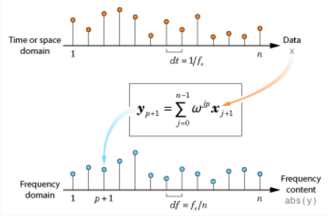
[b] Thus convolution could not be applied directly.

The Emerging Field of Signal Processing on Graphs

Challenge # 2: Transform

The analogous spectrum in the graph setting is discrete and irregularly spaced. Recall that:

> *Modulating a signal on the real line by multiplying by a complex exponential corresponds to translation in the Fourier domain.*

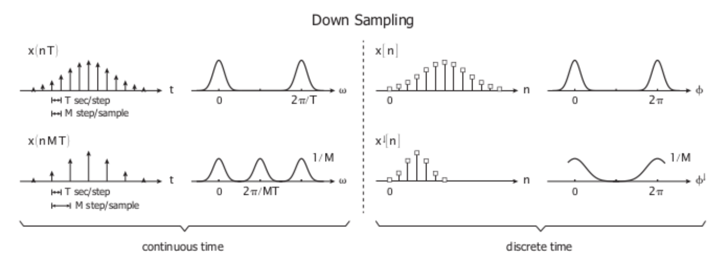


"Transformation" to the graph spectral domain is needed, otherwise convolution could not be applied.

The Emerging Field of Signal Processing on Graphs

Challenge # 3: Downsampling

Downsampling means “delete every other data point” for a discrete-time signal. But what is downsampling for graph signal? What is “every other vertex”?

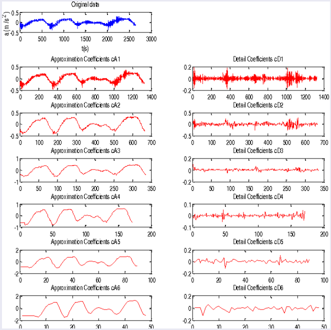


Essential for pooling.

The Emerging Field of Signal Processing on Graphs

Challenge # 4: Multiresolution

With a fixed notion of downsampling, we still need a method to generate coarser version of the graph to achieve multiresolution. And the coarse version should still capture the structural properties embedded in the original graph. Essential for pooling.



Problem Definition (1/2)

- Analyzing signals defined on an undirected, connected, weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{W}\}$, where set of vertices is finite $|\mathcal{V}| = N$, edges is represented by $\mathcal{E}$ and $\mathcal{W}$ is a weighted adjacency matrix.
- Edge $e = (i, j)$ connects vertices v_i and v_j, the entry $\mathcal{W}_{i,j}$ represents the weight of the edge. If $e = (i, j)$ does not exists then $\mathcal{W}_{i,j} = 0$.
- If the graph $\mathcal{G}$ is not fully connected and has $M > 1$ connected components, separate the graph into M parts and independently process the separated signals on each subgraph.
- A signal / function defined on the vertices of the graph: $f : \mathcal{V} \to \mathbb{R}$ may be represented as a vector $f \in \mathbb{R}^N$, where f_i represents the function value at v_i.

The Emerging Field of Signal Processing on Graphs

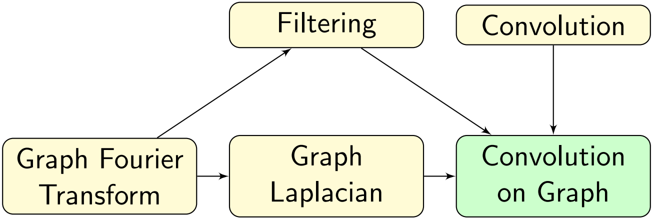
Problem Definition (2/2)

- $\mathcal{W}_{i,j}$ could be naturally given by the application, or defined by ourselves. One common way is the **threshold Gaussian kernel** (for some parameters θ and κ):

$$\mathcal{W}_{i,j} = \begin{cases} \exp -\frac{[dist(i,j)]^2}{2\theta^2} & \text{if } dist(i,j) \leq \kappa \\ 0 & \text{otherwise} \end{cases}$$

Solution: define operations on weighted graphs differently.

The Emerging Field of Signal Processing on Graphs



Other topics mentioned in the paper: Translation, Modulation & Dilation, Graph Coarsening & Downsampling & Reduction (Challenge # 3 and 4), vertex domain design, etc.

The Emerging Field of Signal Processing on Graphs

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[2] of 1-d Laplace operator $\triangle$:

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

or, in other ways,

$$\hat{f}(\omega) = \int f(t) e^{-i\omega t} dt, \qquad \triangle(e^{-i\omega t}) = -\omega^2 e^{-i\omega t}$$

$$e^{i\omega t} = \cos(\omega n) + i \sin(\omega n), \qquad i = \sqrt{-1}$$

[2] Should correspond to eigenvectors in graph settings.

The Emerging Field of Signal Processing on Graphs

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

$$\hat{f}(\lambda_t) = \langle f, u_l \rangle = \sum_{i=1}^{N} f(i) u_l^*(i)$$

$u_l^*(i)$ is the conjugate of $u_l(i)$ in the complex space.
The **inverse** *graph Fourier transform* is then given by:

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

Intuitive understanding of **inverse** Fourier transform[3]: we know all frequency and phase information about a signal then we may reconstruct the original signal precisely.

With $\otimes$ for convolution, $\odot$ for simple multiplication:

$$\mathcal{X} \otimes \mathcal{Y} = Fourier_{inverse}(Fourier(\mathcal{X}) \odot Fourier(\mathcal{Y}))$$

[3] Yunsheng's slides: change of basis

The Emerging Field of Signal Processing on Graphs

Filtering[4] **in the frequency domain**:

$$\hat{f}_{out}(\xi) = \hat{f}_{in}(\xi)\hat{h}(\xi)$$

its inverse Fourier transform:

$$f_{out}(t) = (f_{in} * h)(t) = \int_{\mathbb{R}} \hat{f}_{in}(\xi)\hat{h}(\xi)e^{2\pi i \xi t} d\xi$$

Convolution on Graph: frequency filtering with the complex exponentials replaced by the **graph Laplacian** eigenvectors $(u_l(i))$.

$$f_{out}(i) = (f * h)(i) = \sum_{l=0}^{N-1} \hat{f}(\lambda_l)\hat{h}(\lambda_l)u_l(i)$$

where $\hat{h}(\cdot)$ is the transfer function of the filter, f_{in} omitted in to be f.

[4] Yunsheng's slides: feature aggregation; Defferrard's work: convolution + non-linear activation.

The Emerging Field of Signal Processing on Graphs

1-d Laplacian $\triangle$: the second derivative
Graph Laplacian: A difference operator that satisfies

$$(Lf)(i) = \sum_{j \in \mathcal{N}_i} W_{i,j}[f(i) - f(j)]$$

where $\mathcal{N}_i$ is the set of vertices connected to vertex i by an edge.

Graph Laplacian

What

- D: diagonal matrix whose i^{th} diagnal element d_i is equal to the sum of the weights of all the edges incident to v_i
- **combinatorial graph Laplacian** / non-normalized graph Laplacian: $L = D - W$
- normalized graph Laplacian / symmetric normalized Laplacian: $\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_N - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$.[a]
- asymmetric graph Laplacian: $L_a = I_N - P$, where $P = D^{-1}W$ is the *random walk matrix*[b]

[a] I_N is $N \times N$ identity matrix.

[b] $P_{i,j}$ describes the probability of going from v_i to v_j in one step of a Markov random walk on $\mathcal{G}$.

Graph Laplacian

What - in other words

- $L = D - W$ (when W is A, which means that each edge has weight 1):

$$L(u.v) = \begin{cases} d_v & \text{if } u = v \ (d_v \text{ is the degree of node } v) \\ -1 & \text{if } u \neq v, (u,v) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

- $\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_N - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ (when W is A):

$$\tilde{L}(u.v) = \begin{cases} 1 & \text{if } u = v, d_v \neq 0 \ (d_v \text{ is the degree of node } v) \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \neq v, (u,v) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

The Emerging Field of Signal Processing on Graphs

Graph Laplacian

What - more

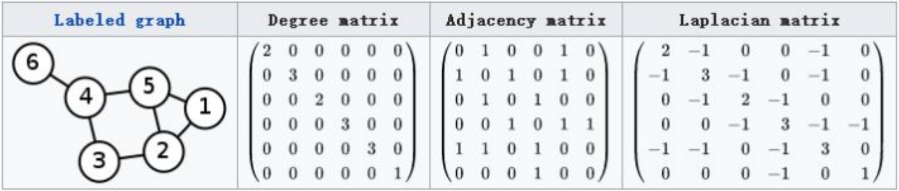
- The normalized and non-normalized graph Laplacians are both examples of *generalized graph Laplacians*[a]. Generalization: any symmetric matrix whose i, j^{th} entry is negative if there's an edge connecting v_i, v_j, zero if $i \neq j$ and v_i, v_j not connected, may be anything when $i = j$.
- L_a has the same set of eigenvalues as $\tilde{L}$[b].
- Laplacian of a graph is also sometimes called: **admittance matrix**, **discrete Laplacian** or **Kirchohoff matrix**.
- It remains no clear answer when should we use which Laplacian matrix.

[a] Also called *discrete Schrödinger operators*.

[b] If $\tilde{u}_l$ is an eigenvector of $\tilde{L}$ associated with $\tilde{\lambda}_l$, then $D^{-\frac{1}{2}}\tilde{u}_l$ is an eigenvector of L_a associated with the eigenvalue $\tilde{\lambda}_l$

The Emerging Field of Signal Processing on Graphs

An example of non-normalized L with $W = A$, $L = D - A$:



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

The Emerging Field of Signal Processing on Graphs

Graph Laplacian properties:

- (normalized / non-normalized) graph Laplacian (L, or $\tilde{L}$) is a real symmetric matrix, with a complete set of orthonormal eigenvectors[5], which we denote by $\{u_l\}$ where $l = 0, 1, \ldots, N-1$.
- $\{u_l\}$ has associated real non-negative eigenvalues $\{\lambda_l\}$.
- $Lu_l = \lambda_l u_l$
- The amount of eigenvalues that appears to be 0 is equal to the number of connected components of the graph.
- Considering connected graphs only, we could order the eigenvalues as $0 = \lambda_0 \leq \lambda_1 \cdots \leq \lambda_{N-1}$.
- The entire spectrum could be denoted as:

$$\sigma(L) = \{\lambda_0, \lambda_1, \ldots, \lambda_{N-1}\}$$

[5] Also known as *graph Fourier modes*.

The Emerging Field of Signal Processing on Graphs

Graph Laplacian another expression:

$$L = U \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1} \end{bmatrix} U^{-1} = U \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1} \end{bmatrix} U^T$$

where λ_i is eigenvalue, and $U = (\vec{u_0}, \vec{u_1}, \ldots, \vec{u_N})$, $\vec{u_\rangle}$ is column vector, and is unit eigenvector. U^{-1} could be replaced by U^T because $UU^T = I_N$.

Fourier transformation could be expressed as:

$$\hat{h}(L) = U \begin{bmatrix} \hat{h}(\lambda_0) & 0 & \cdots & 0 \\ 0 & \hat{h}(\lambda_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{h}(\lambda_{N-1}) \end{bmatrix} U^T, \qquad f_{out} = \hat{h}(L) f_{in}$$

The Emerging Field of Signal Processing on Graphs

Graph Coarsening: $(\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{W}\})$

1. Graph reduction: identify a reduced set of vertices $\mathcal{V}^{reduced}$
2. Graph contraction: assign edges and weights to connect the new set of vertices, $\mathcal{E}^{reduced}$ and $\mathcal{W}^{reduced}$.

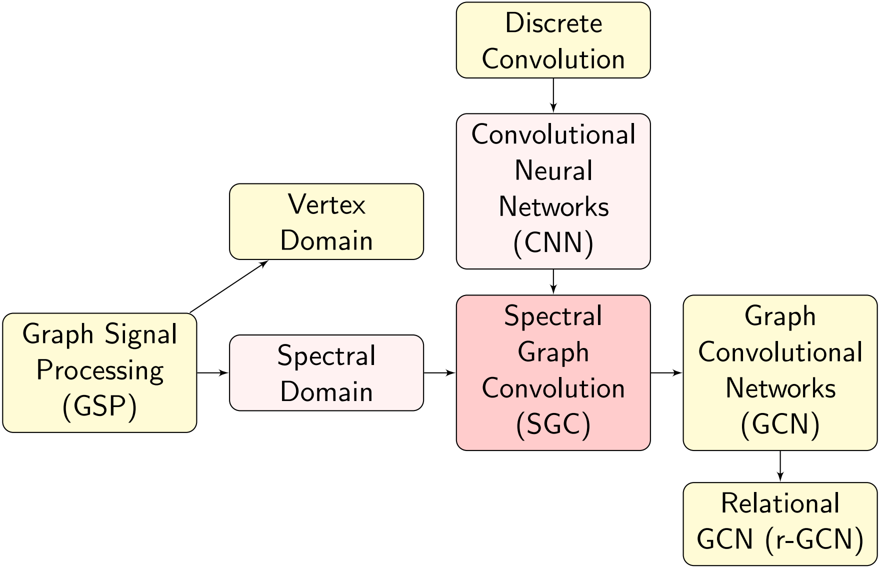
Something else to know

- The convolution theorem defines convolutions as linear operators that diagonalize in the Fourier basis (represented by the eigenvectors of the Laplacian operator)[6].
- Fourier transform localize signals in frequency domain but not in time domain.
- Indeterminacy principles: one cannot be arbitrarily accurate in local both in time and frequency.
- The meaning of localize: to find where the signal is mostly concentrated, and with what precision.
- Vertex domain filtering ($\mathcal{N}(i, k)$ is i's neighbors within K-hop):

$$f_{out}(i) = b_{i,i} f_{in}(i) + \sum_{j \in \mathcal{N}(i,K)} b_{i,j} f_{in}(j)$$

[6] Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (NIPS 2016)

CNN on Graphs with Fast Localized Spectral Filtering


Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

CNN on Graphs with Fast Localized Spectral Filtering

Problem Settings

1. Data sets: MNIST and 20NEWS
2. $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{W}\}$, undirected connected weighted graph.
3. Focus on filtering, other parts remain default (such as the activation, ReLU).

Contributions

1. A spectral graph theoretical formulation of CNNs on graphs;
2. Strictly localized spectral filters, limited to a radius of K (K hops from the central vertex);
3. Low computational complexity, $O(K\mathcal{E})$.
4. Low storage cost, stores the data and Laplacian (sparse, $|\mathcal{E}|$ values).
5. Efficient pooling strategy[a].

[a] Rearrange into binary tree, analog to pooling of 1D signals

CNN on Graphs with Fast Localized Spectral Filtering

Fast localized spectral filters:

Why?

- Why not spatial filter, why spectral filter? - Spatial domain filter is naturally localized, but spectral domain could also achieve localization, while having more mathematically-supported operators, etc.
- Why emphasize "fast"? - a filter defined in the spectral domain is not naturally localized, translations are costly due to the $O(n^2)$ multiplication with the graph Fourier basis.

How?

A **special choice of filter parametrization**.

CNN on Graphs with Fast Localized Spectral Filtering

Starting from signal processing:

- Recall that: The Laplacian is indeed diagonalized by the Fourier basis (the orthonormal eigenvectors) $U = [u_0, u_1, \ldots, u_{N-1}] \in \mathbb{R}^{N \times N}$.
- Let $\Lambda = diag([\lambda_0, \ldots, \lambda_{N-1}])$, then we have $L = U\Lambda U^T$.
- The graph Fourier transform of a signal $x \in \mathbb{R}^N$ is then defined as $\hat{x} = U^T x \in \mathbb{R}^N$, with inverse $x = U\hat{x}$. This transformation enables the formation of fundamental operations, such as filtering, just as on Euclidean spaces.
- Recall that: $\mathcal{X} \otimes \mathcal{Y} = Fourier_{inverse}(Fourier(\mathcal{X}) \odot Fourier(\mathcal{Y}))$
- Definition of convolutional operator:

$$(f * h)(*\mathcal{G}) = U((U^T f) \odot (U^T h))$$

where $\odot$ is the element-wise Hadamard product[7].

[7] Hadamard product: $(A \odot B)_{i,j} = A_{i,j} * B_{i,j}$

CNN on Graphs with Fast Localized Spectral Filtering

- A signal x filtered by g_θ as

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

where a non-parametric filter whose parameters are all free would be defined as $g_\theta(\Lambda) = diag(\theta)$, with $\theta \in \mathbb{R}^N$ be a vector of Fourier coefficients.

- Non-parametric filters are not localized in space and have learning complexity of $O(N)$. Solution: using a polynomial filter:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

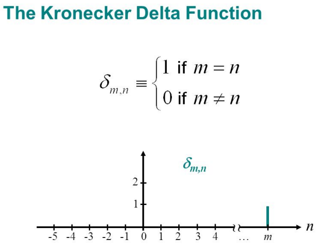
where $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients.

CNN on Graphs with Fast Localized Spectral Filtering

Something we skipped in the first paper[8]:
Kronecker delta function $\delta_i \in \mathbb{R}^N$.

$$\delta_n(i) = \begin{cases} 1 & \text{if } i = n \\ 0 & \text{otherwise} \end{cases}$$

Usage: for localization.



The Kronecker Delta Function

$$\delta_{m,n} \equiv \begin{cases} 1 \text{ if } m = n \\ 0 \text{ if } m \neq n \end{cases}$$

[8] It is in the **translation** part.

CNN on Graphs with Fast Localized Spectral Filtering

What is the value of filter g_θ, at vertex j, centered at vertex i?

$$(g_\theta(L)\delta_i)_j = (g_\theta(L))_{i,j} = \sum_k \theta_k (L^k)_{i,j}$$

CNN on Graphs with Fast Localized Spectral Filtering

- Let $d_{\mathcal{G}}$ denote the shortest path distance, $d_{\mathcal{G}}(i,j) > K$ means that $(L^K)_{i,j} = 0$.
- Spectral filters represented by K^{th}-order polynomials of the Laplacian are exactly K-localized.
- The learning complexity is $O(K)$, the support size of the filter, thus same complexity as classic CNN.

CNN on Graphs with Fast Localized Spectral Filtering

Note that computing $y = Ug_\theta(\Lambda)U^T x$ is still expensive when having to do multiplication with the Fourier basis U.
How to achieve **fast** filtering?

Solution

Parameterize $g_\theta(L)$ as a polynomial function that can be recursively computed from L, then it costs us K multiplications of L, time complexity will be reduced from $O(N^2)$ to $O(K|\mathcal{E}|)$.

- Existing tools in the field of GSP: Chebyshev expansion and Krylov subspace.
- The reason why Chebyshev and not Krylov: simpler.
- Chebyshev expansion: $T_0(x) = 0$, $T_1(x) = x$, $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$.

CNN on Graphs with Fast Localized Spectral Filtering

Recall the parameterized filter:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

With Chebyshev expansion, we have

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

where $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_N$ is a diagonal matrix of scaled eigenvalues lie in $[-1, 1]$. Now $\theta \in \mathbb{R}^K$ becomes a vector of Chebyshev coefficients.
With $\tilde{L} = 2L/\lambda_{max} - I_N$, we have $\bar{x}_0 = x$, $\bar{x}_1 = \tilde{L}x$,
$\bar{x}_k = T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$:

$$y = g_\theta(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x = [\bar{x}_0, \ldots \bar{x}_{K-1}]\theta$$

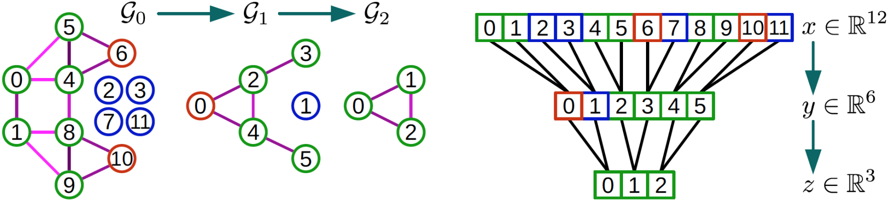
CNN on Graphs with Fast Localized Spectral Filtering

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i} \in \mathbb{R}^N$$

E: loss energy over a mini-batch of S samples. Loss energy E is the cross-entropy with an l_2 regularization on the weights of all FCk (Fully-Connected layer with k hidden units) layers.
Training: back propagation using E's gradients over θ and x.

CNN on Graphs with Fast Localized Spectral Filtering

Pooling: coarsening to create a balanced binary tree (map two unmatched neighbors each time); then the number of vertices is reduced.



Adjacent nodes are hierarchically merged at coarser levels.

CNN on Graphs with Fast Localized Spectral Filtering

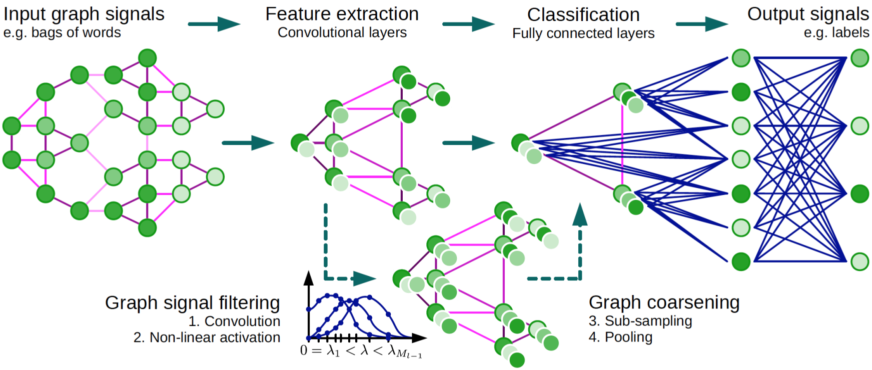
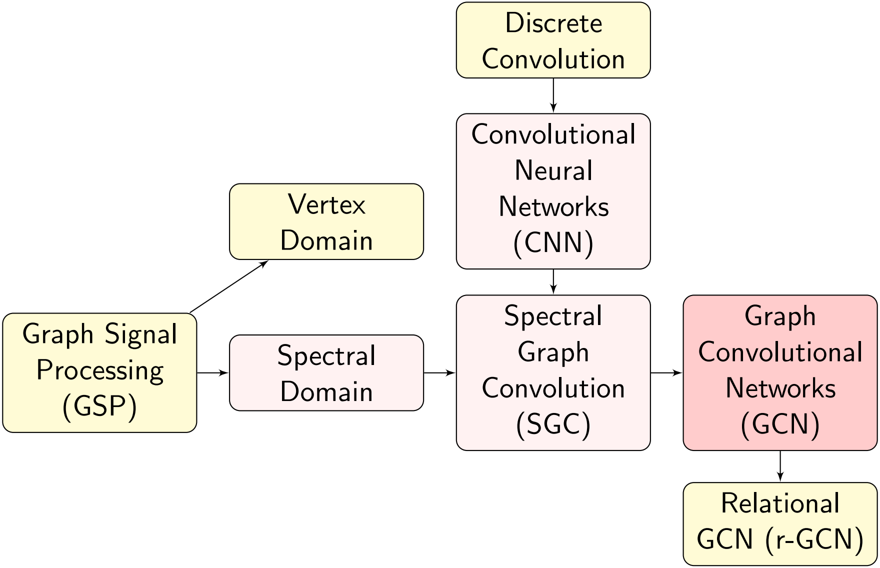


Figure 1: Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer.

Semi-Supervised Classification with GCN



Discrete Convolution

Convolutional Neural Networks (CNN)

Vertex Domain

Graph Signal Processing (GSP)

Spectral Domain

Spectral Graph Convolution (SGC)

Graph Convolutional Networks (GCN)

Relational GCN (r-GCN)

Semi-Supervised Classification with GCN

Problem definition

- Task: semi-supervised entity classification
- Data sets: citation network, knowledge graph, random graph.
- Graph: undirected, weight $= 1$ if connected otherwise 0.

Contribution

- A layer-wise propagation rule, that is simple and well-behaved, working for NN models.
- Semi-supervised classification, that is fast and scalable, using GCN[a].

[a] Previous methods always use graph Laplacian regularization term in the loss function to smooth the graph and make up for the absence of labels. But they have an assumption of obvious similarity of neighbor, which leads to restriction in model capacity

Semi-Supervised Classification with GCN

Staring from graph signal processing again:

- Using normalized graph Laplacian: $\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_N - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ (W is simplified to be A)
- Using the same approximation used in the previous paper (parameterize using Chebyshev polynomials).
- Further limit $K = 1$ (only the direct neighbor is reachable at each layer), and stack multiple layers.
- Add self-connection to each vertices.

Semi-Supervised Classification with GCN

Recall that, previously, with Chebyshev expansion, we have:

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

where $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_N$ is a diagonal matrix of scaled eigenvalues lie in $[-1, 1]$. With $\tilde{L} = 2L/\lambda_{max} - I_N$, we have $\bar{x}_0 = x$, $\bar{x}_1 = \tilde{L}x$, $\bar{x}_k = T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$:

$$y = g_\theta(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x = [\bar{x}_0, \dots \bar{x}_{K-1}]\theta$$

Now we have layer-wise linear model by limit $K = 1$, and further approximate $\lambda_{max} \approx 2$. We are expecting that by stacking multiple such layers the model should perform well.

Semi-Supervised Classification with GCN

$$y = g_\theta(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x = [\bar{x}_0, \dots \bar{x}_{K-1}]\theta$$

becomes:

$$y = g_\theta(L)x \approx \theta_0 x + \theta_1 (L - I_N)x = \theta_0 x - \theta_1 \tilde{L}x$$

where $\tilde{L}$ is the normalized Laplacian ($\tilde{L} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$) in this case. With $\theta^* = \theta_0 - \theta_1$, we have:

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

With $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $\Theta \in \mathbb{R}^{C \times F}$ is the filter parameters, $Z \in \mathbb{R}^{N \times F}$ be the convolved signal matrix, we have:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

Semi-Supervised Classification with GCN

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$$

Adding an activation function, we get the form of:

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

which is the well-known form of GCN. In practice they used ReLU as the non-linear activation function.
In details[9] ($c_{i,j} = \sqrt{d_i d_j}$, $d_i = |\mathcal{N}_i|$):

$$h_i^{(l+1)} = \sigma\Big(\sum_{j\in\mathcal{N}_i}\frac{1}{c_{i,j}}h_j^{(l)}W^{(l)}\Big)$$

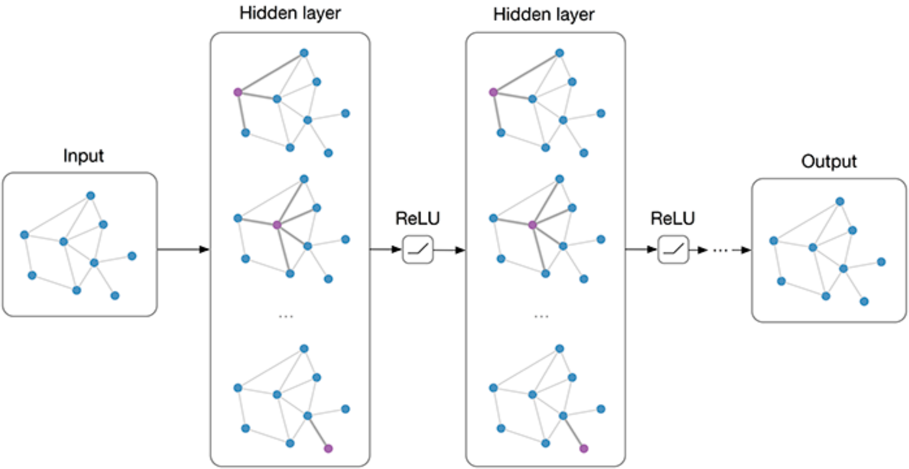
All the expressions could be generalized as[10]:

$$h_i^{(l+1)} = \sigma\Big((\sum_{m\in M_i} g_m(h_i^{(l)}, h_j^{l})\Big)$$

[9] See the Weisfeiler-Lehman algorithm in appendix

[10] Will be seen in r-GCN part.

Semi-Supervised Classification with GCN



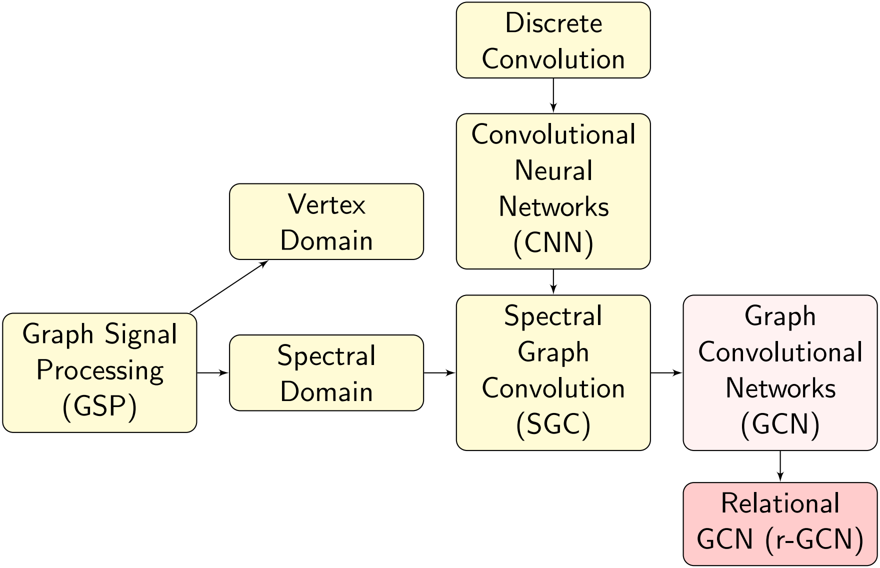
Semi-Supervised Classification with GCN

Downstream task: **semi-supervised classification**
Loss function:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

1. $\mathcal{Y}_L$: set of labeled nodes (indices)
2. F: number of labels of the nodes
3. Z: predicted outcome, softmax of the output of the network.

Modeling Relational Data with GCN (r-GCN)


Discrete Convolution
Convolutional Neural Networks (CNN)
Vertex Domain
Graph Signal Processing (GSP)
Spectral Domain
Spectral Graph Convolution (SGC)
Graph Convolutional Networks (GCN)
Relational GCN (r-GCN)

Modeling Relational Data with GCN (r-GCN)

Problem Settings

- Tasks: entity classification and link prediction.
- Data sets: knowledge graph.
- Relations are directed, but the adjacent matrices we use contains each relation and its reverse.

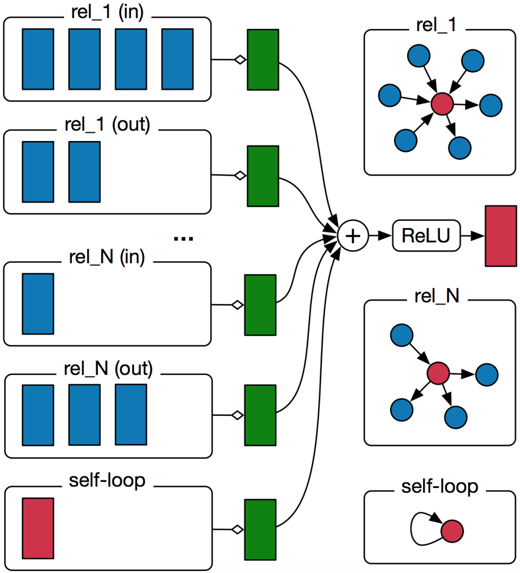
Motivation

Tasks on more general (heterogeneous) graphic structure: knowledge graph completion.

Solution

GCN + weighed sum of neighbors from different relations; regularization applied to weights so as to reduce overfitting problem.

Modeling Relational Data with GCN (r-GCN)



Modeling Relational Data with GCN (r-GCN)

GCN model

$$h_i^{(l+1)} = \sigma\Big(\sum_{m\in M_i} g_m(h_i^{(l)}, h_j^{(l)})\Big)$$

$h_i^{(l)}$	Hidden state of node v_i in the l^{th} layer. size: $\mathbb{R}^{d^{(}(l))}$.
$d^{(l)}$	l^{th} layer's representation's dimension.
$\sigma(\cdot)$	Nonlinear activation function (e.g. ReLU)
$\mathcal{M}_i$	Incoming messages for node v_i, always (chosen to be) identical to the incoming edges.
$g_m(\cdot,\cdot)$	A message-specific NN-like function, or simply a linear transformation ($g(h_i, h_j) = Wh_j$). Kipf and Welling did it the weight matrix way.

r-GCN model

$$h_i^{(l+1)}=\sigma\Big((\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)}+W_0^{(l)}h_i^{(l)}\Big)$$

$\mathcal{R}$	Set of all relations.
$\mathcal{N}_i^r$	Set of neighbor (indices) of node v_i under relation r.
$c_{i,r}$	Problem-specific normalization constant, could be learned or chosen in advance (in their paper, chosen to be $\lvert\mathcal{N}_i^r\rvert$).
$W_0^{(l)}h_i^{(l)}$	A single self-connection of a special relation type to ensure that representation at layer l is passed to corresponding part of layer $l+1$. (In practice, an identity matrix is added when representing the adjacent matrices.)
$W_r^{(l)}$	The weight to be regularized (next slide) and learned (next next slide).

r-GCN: ways of regularizing $W_r^{(l)}$

Basis decomposition

$$W_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} V_b^{(l)}$$

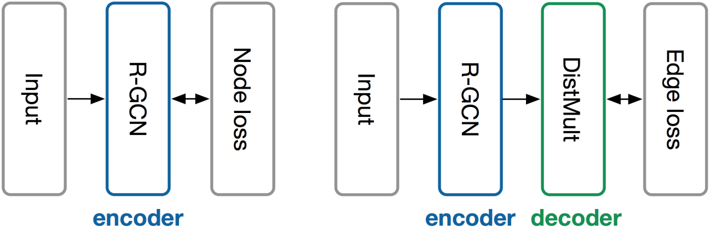
$V_b^{(l)}$	Basis transformations to be learned, similar with traditional weights. $\in \mathbb{R}^{d^{(l+1)}} \times \mathbb{R}^{d^{(l)}}$.
$a_{rb}^{(l)}$	Coefficients, the only parameter that depends on r, dimension related with support and number of basis (B), it is a parameter that is assigned individually to each adjacent matrix ($\|2R+1\|$ matrices in total, also called support).
B	Number of base, user-specified parameter, sometimes called "**number of coefficients**" in their expression

Block-diagonal-decomposition

$$W_r^{(l)} = \bigoplus_{b=1}^{B} Q_{rb}^{(l)}$$

$W_r^{(l)}$	Are block-diagonal matrices: $(Q_{1r}^{(l)}, Q_{2r}^{(l)} \ldots Q_{Br}^{(l)})$.
note∗	Actually *not implemented for node classification task*.
note∗	can be viewed as **sparsity constraints** (whereas basis decomposition can be viewed as **weight sharing**).

r-GCN: downstream tasks (Knowledge Graph Completion)



(a) Entity classification

(b) Link prediction

r-GCN: downstream tasks (Knowledge Graph Completion)

Entity Classification

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{k=1}^{K} t_{ik} \ln h_{ik}^{(L)}$$

$\mathcal{L}$	Categorical cross-entropy loss.
$\mathcal{Y}$	Nodes that have labels.
t_{ik}	Ground truth labels.
$h_{ik}^{(L)}$	k^{th} entry of node i at layer L (Layer L is the output layer.)

Link Prediction

$$\mathcal{L} = -\frac{1}{(1+\omega)|\hat{\varepsilon}|} \sum_{(s,r,o,y) \in \mathcal{T}} y \log l\Big(f(s,r,o)\Big) + (1-y) \log \Big(1 - l\Big(f(s,r,o)\Big)\Big)$$

e_i	$e_i = h_i^{(L)}$, use r-GCN as encoder.
$f(s,r,o)$	**DistMult** factorization is used, $f(s,r,o) = e_s^T \mathcal{R}_r e_o$.
(s,r,o)	s, o are node (indices), r is the relation between them.
y	$y = 1$ for positive triples, 0 for negative triples. Negative sampling is used.
$\mathcal{T}$	All triples (real + corrupted).
$\mathcal{R}$	$\mathcal{R}_r$ is a d*d diagonal matrix to be learned.
l	Logistic sigmoid function.

References

- Yunsheng's Slides on GCN
- Ferenc Huszar's comment on GCN (+ followup discussion by Thomas Kipf)
- Thomas Kipf (the author)'s comment on GCN (+ replying Ferenc Huszar)

The End