Graph Neural Network

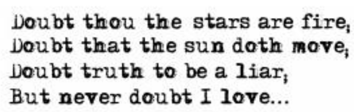
Fang Yuanqiang, 2019/05/18

Graph Neural Network

- Why GNN?
- Preliminary
- Fixed graph
  - Vanilla Spectral Graph ConvNets
  - ChebyNet
  - GCN
  - CayleyNet
  - Multiple graphs
- Variable graph
  - GraphSAGE
  - Graph Attention Network
- Tasks

Why GNN?

- Euclidean domain & Non-Euclidean domain

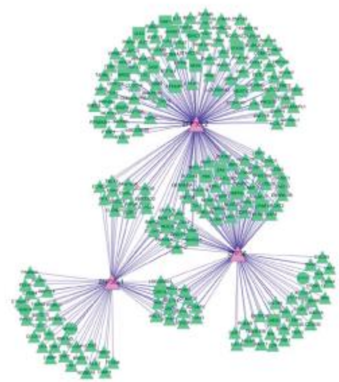


Text

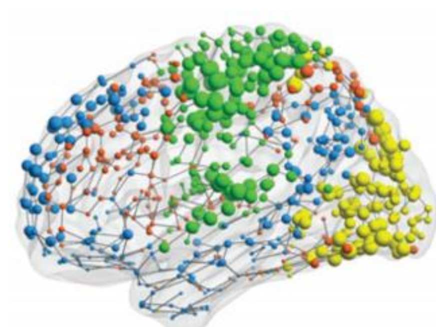
Audio signals

Images

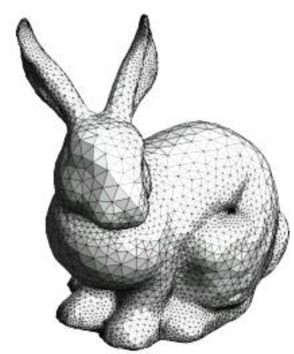
Social networks



Regulatory networks



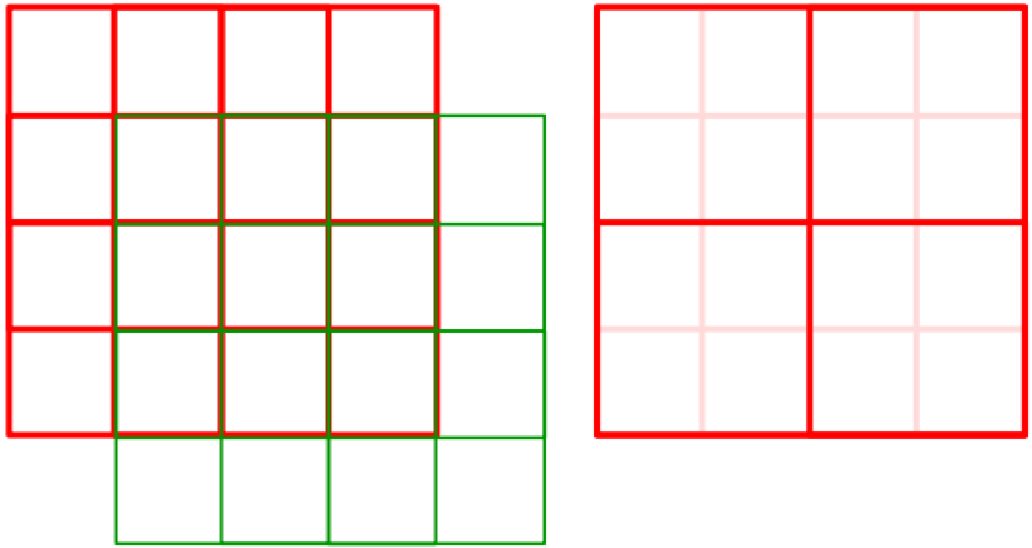
Functional networks



3D shapes

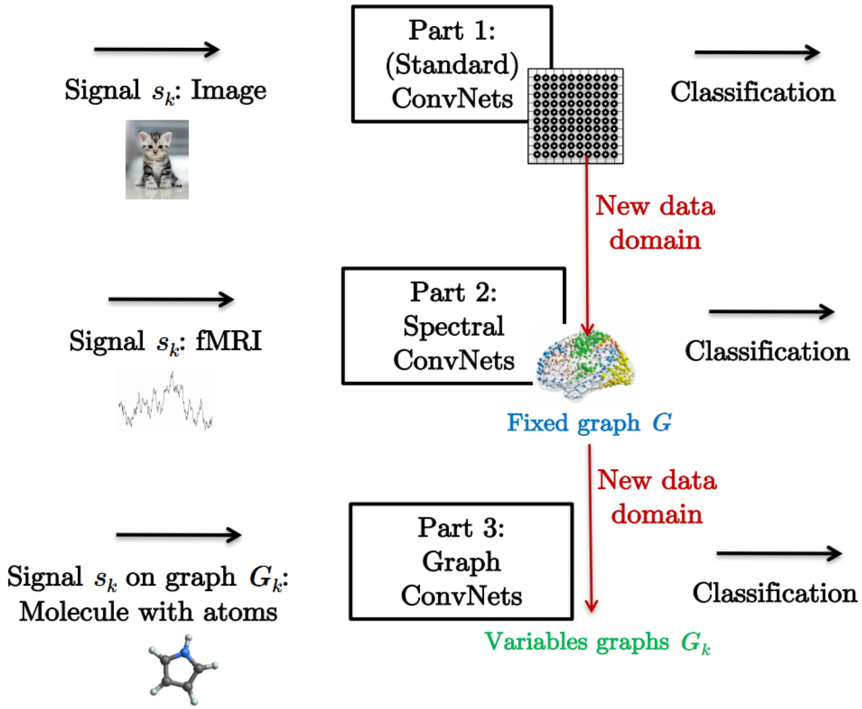
Why GNN?

- ConvNets and Euclidean geometry
  - Data (image, video, sound) are compositional, they are formed by patterns that are:
    - ✓ Local → convolution
    - ✓ Multi-scale (hierarchical) → downsampling/pooling
    - ✓ Stationary → global/local invariance



Why GNN?

- Extend ConvNets to graph-structured data
  - Assumption: Non-Euclidean data are locally stationary and manifest hierarchical structures.
  - How to define compositionality on graphs? (conv. & pooling)
  - How to make them fast? (linear complexity)



Preliminary

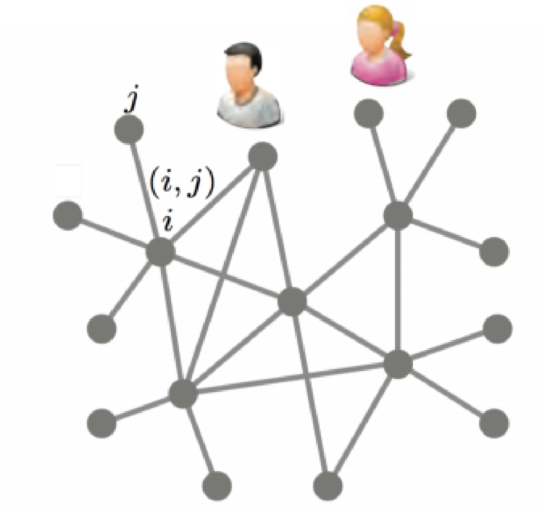
- Theory
  - Graph theory
  - Convolution, spectral convolution
  - Fourier transform
  - Riemannian manifold
  - ……
- Reference
  - http://geometricdeeplearning.com/slides/NIPS-GDL.pdf
  - http://helper.ipam.ucla.edu/publications/dlt2018/dlt2018_14506.pdf
  - https://www.zhihu.com/question/54504471?sort=created

Preliminary

Graph

- **Graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- **Vertices** $\mathcal{V} = \{1, \ldots, n\}$
- **Edges** $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$
- **Vertex weights** $b_i > 0$ for $i \in \mathcal{V}$
- **Edge weights** $a_{ij} \geq 0$ for $(i, j) \in \mathcal{E}$
- **Vertex fields** $L^2(\mathcal{V}) = \{f : \mathcal{V} \rightarrow \mathbb{R}^h\}$
  Represented as $\mathbf{f} = (f_1, \ldots, f_n)$
- **Hilbert space with inner product**

$$\langle f, g \rangle_{L^2(\mathcal{V})} = \sum_{i \in \mathcal{V}} a_i f_i g_i$$



Preliminary

Graph Laplacian

- Represented as a positive semi-definite $n \times n$ matrix
  - Unnormalized Laplacian $\quad \boldsymbol{\Delta} = \mathbf{D} - \mathbf{A}$
  - Normalized Laplacian $\quad \boldsymbol{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
  - Random walk Laplacian $\quad \boldsymbol{\Delta} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$

  where $\mathbf{A} = (a_{ij})$ and $\mathbf{D} = \mathrm{diag}(\sum_{j \neq i} a_{ij})$

  - Eigendecomposition of graph Laplacian:

$$\boldsymbol{\Delta} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top$$

where $\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_n)$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$
$(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} = \mathbf{I})$

Preliminary

- Convolution: continuous
  - Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(x')g(x - x')dx'$$

    - Shift-invariance: $f(x - x_0) \star g(x) = (f \star g)(x - x_0)$
    - Convolution theorem: Convolution can be computed in the Fourier domain as

$$\widehat{(f \star g)} = \hat{f} \cdot \hat{g}$$

    - Efficient computation using FFT: $\mathrm{O}(n\log n)$

Preliminary

- Convolution: discrete
  - Convolution of two vectors $\mathbf{f} = (f_1, \ldots, f_n)^\top$ and $\mathbf{g} = (g_1, \ldots, g_n)^\top$

$$(\mathbf{f} \star \mathbf{g})_i = \sum_m g_{(i-m) \bmod n} \cdot f_m$$

Circular convolution

$$\mathbf{f} \star \mathbf{g} = \underbrace{\begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix}}_{\text{Circulant matrix}} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

diagonalised by Fourier basis (Toeplitz)

Spatial (2-d)
Temporal (1-d)
domain

Spectral domain

inverse Fourier transform

Hadamard product

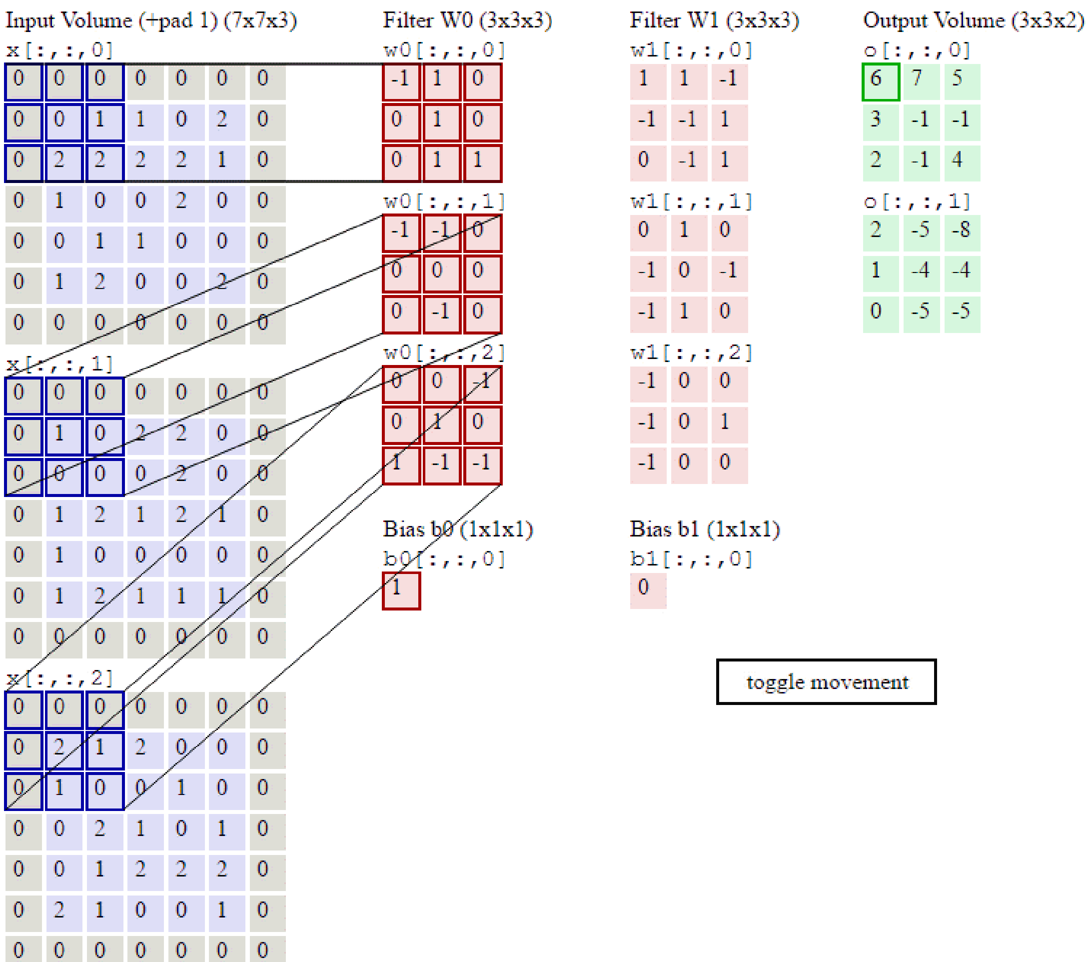
$$= \mathbf{\Phi} \begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix} \mathbf{\Phi}^\top \mathbf{f} = \mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{g} \circ \mathbf{\Phi}^\top \mathbf{f})$$

Fourier transform

$\mathbf{\Phi}$: *DFT matrix/Fourier matrix*

Preliminary: aside

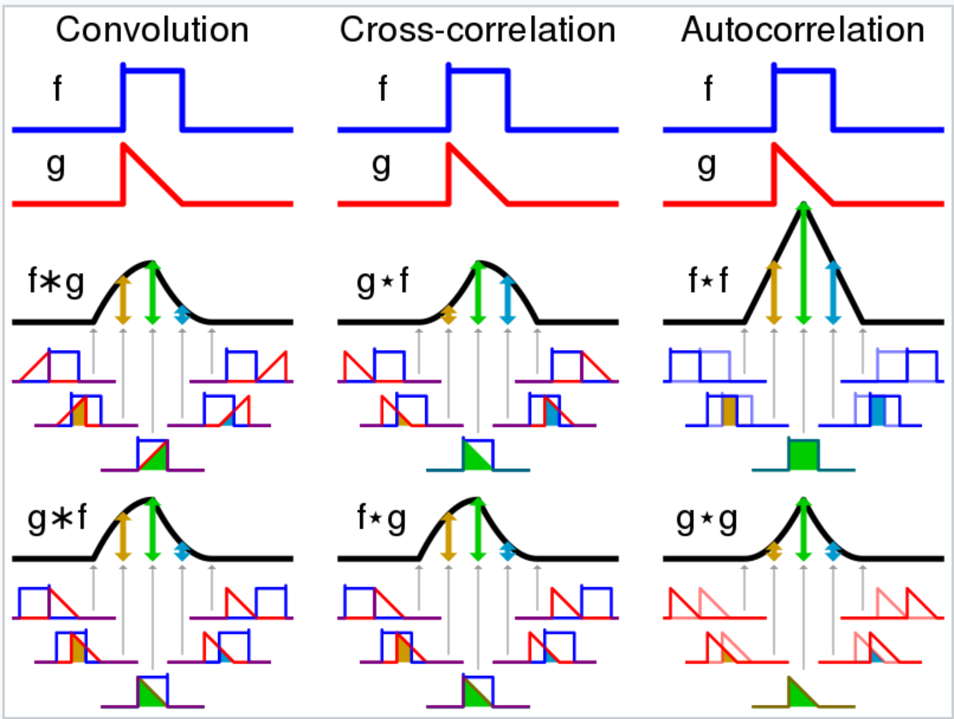
- “Conv” in Deep Neural Networks.



http://cs231n.github.io/convolutional-networks/

Preliminary: aside

- "Conv" in Deep Neural Networks.



Visual comparison of convolution, cross-correlation and autocorrelation. For the operations involving function f, and assuming the height of f is 1.0, the value of the result at 5 different points is indicated by the shaded area below each point. Also, the vertical symmetry of f is the reason $f * g$ and $f \star g$ are identical in this example.

https://en.wikipedia.org/wiki/Cross-correlation

Preliminary: aside

- "Conv" in Deep Neural Networks is actually "Cross-correlation".

class `torch.nn.Conv1d`*(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)* [source]

Applies a 1D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output (N, C_{out}, L_{out}) can be precisely described as:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k)$$

where $\star$ is the valid cross-correlation operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

`stride` controls the stride for the cross-correlation, a single number or a one-element tuple.
`padding` controls the amount of implicit zero-paddings on both
sides for `padding` number of points.
`dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
`groups` controls the connections between inputs and outputs. *in_channels* and *out_channels* must both be divisible by *groups*.
At groups=1, all inputs are convolved to all outputs.
At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated. At groups=`in_channels`, each input channel is convolved with its own set of filters (of size *out_channels // in_channels*).

Note

Depending of the size of your kernel, several (of the last) columns of the input might be lost, because it is a valid cross-correlation, and not a full cross-correlation. It is up to the user to add proper padding.

https://pytorch.org/docs/0.3.1/nn.html#convolution-layers

Preliminary

Convolution: graph

- Spectral convolution of $f, g \in L^2(\mathcal{V})$ can be defined by analogy[11]

$$(f \star g)_i = \underbrace{\sum_{k \geq 1} \underbrace{\langle f, \phi_k \rangle_{L^2(\mathcal{V})} \langle g, \phi_k \rangle_{L^2(\mathcal{V})}}_{\text{product in the Fourier domain}} \phi_{k,i}}_{\text{inverse Fourier transform}}$$

- In matrix-vector notation

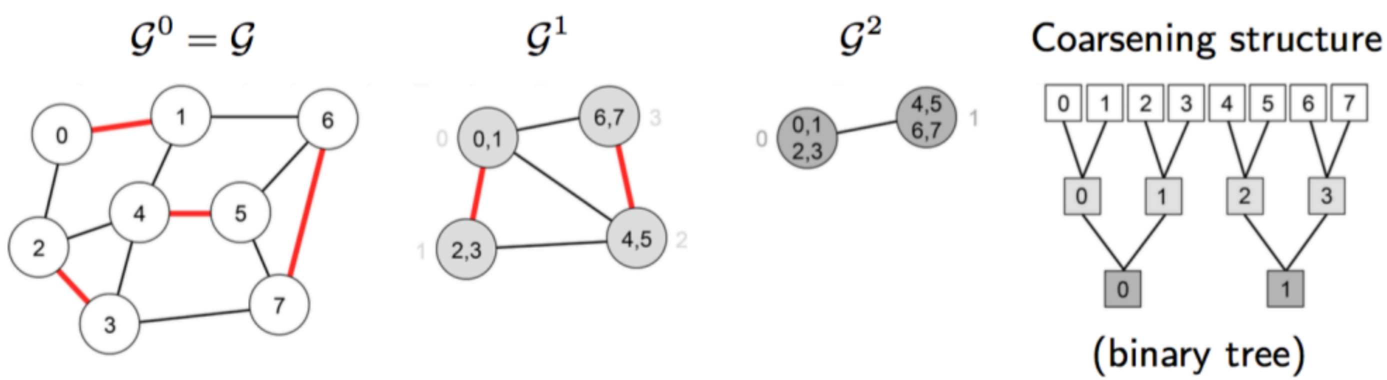
$$\begin{aligned} \mathbf{f} \star \mathbf{g} &= \mathbf{\Phi} (\mathbf{\Phi}^\top \mathbf{g} \circ \mathbf{\Phi}^\top \mathbf{f}) \\ &= \underbrace{\mathbf{\Phi}\, \mathrm{diag}(\hat{g}_1, \ldots, \hat{g}_n) \mathbf{\Phi}^\top}_{\mathbf{G}} \mathbf{f} \\ &= \mathbf{\Phi} \hat{g}(\mathbf{\Lambda}) \mathbf{\Phi}^\top \mathbf{f} = \hat{g}(\mathbf{\Phi} \mathbf{\Lambda} \mathbf{\Phi}^\top) \mathbf{f} = \hat{g}(\mathbf{\Delta}) \mathbf{f} \end{aligned}$$

$\boldsymbol{g}$: *filter*
$\boldsymbol{f}$: *signal*
$\hat{g}(\mathbf{\Lambda})$: *diagonal matrix, function of* $\mathbf{\Lambda}$.

  - Not shift-invariant (G has no circulant structure)
  - Filter coefficients depend on basis $\phi_1, \ldots, \phi_n$
  - Expensive computation (no FFT): $O(n^2)$

Preliminary

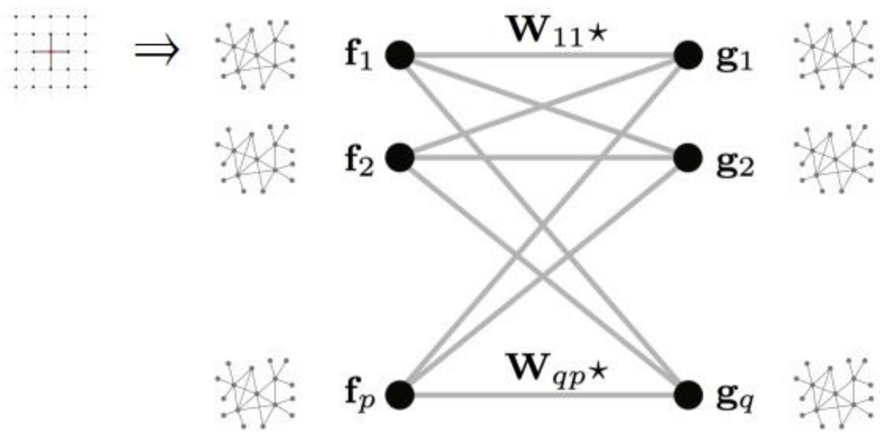
- Graph pooling
  - Produce a sequence of coarsened graphs
  - Max or average pooling of collapsed vertices
  - Binary tree arrangement of node indices



☺ As efficient as 1D-Euclidean grid pooling.

Locally connected networks

$$\begin{aligned} \mathbf{f}_l &= l\text{-th data feature on graphs, } \dim(\mathbf{f}_l) = n \times 1 \\ \mathbf{g}_l &= l\text{-th feature map, } \dim(\mathbf{g}_l) = n \times 1 \end{aligned}$$



$$\mathbf{g}_l = \xi\left(\sum_{l'=1}^{p} \mathbf{W}_{l,l'} \star \mathbf{f}_{l'}\right)$$

Activation, e.g. $\xi(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

Fixed graph: Vanilla Spectral Graph ConvNets

Spectral Networks and Deep Locally Connected Networks on Graphs, 2014, ICLR

- Locally connected networks

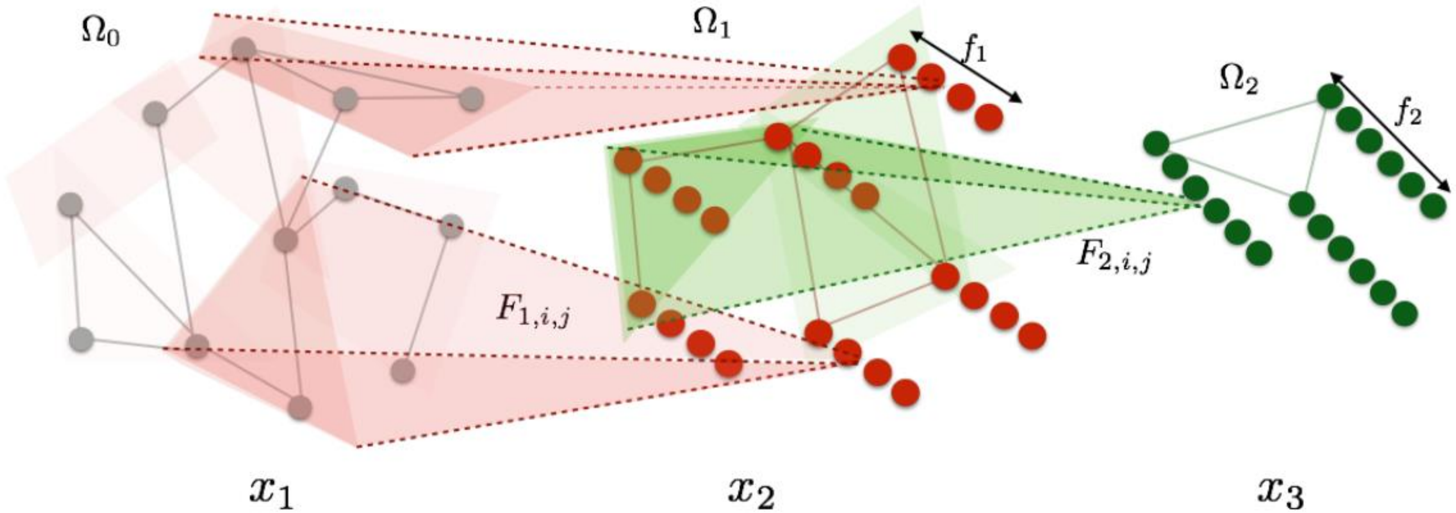


Figure 2: Spatial Construction as described by (2.1), with $K = 2$. For illustration purposes, the pooling operation is assimilated with the filtering stage. Each layer of the transformation loses spatial resolution but increases the number of filters.

- Spectral convolution
  - $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, diagonal matrix of learnable spectral filter coefficients at each layer.

$$\mathbf{g}_l^{(k)} = \xi \left(\sum_{l'=1}^{q^{(k-1)}} \mathbf{\Phi} \mathbf{W}_{l,l'}^{(k)} \mathbf{\Phi}^\top \mathbf{g}_{l'}^{(k-1)} \right)$$

Fixed graph: Vanilla Spectral Graph ConvNets

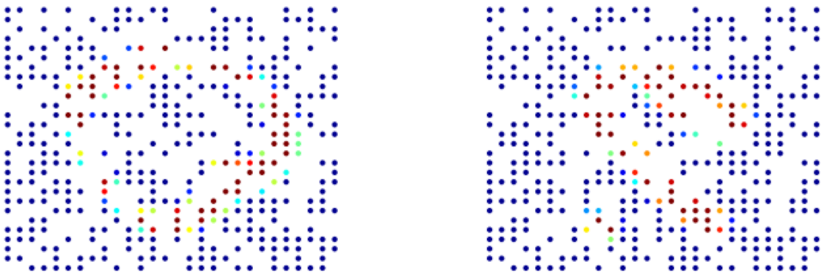
Spectral Networks and Deep Locally Connected Networks on Graphs, 2014, ICLR

Analysis

Table 1: Classification results on MNIST subsampled on 400 random locations, for different architectures. FCN stands for a fully connected layer with N outputs, LRFN denotes the locally connected construction from Section 2.3 with N outputs, MPN is a max-pooling layer with N outputs, and SPN stands for the spectral layer from Section 3.2.

method	Parameters	Error
Nearest Neighbors	N/A	4.11
400-FC800-FC50-10	$3.6 \cdot 10^5$	1.8
400-LRF1600-MP800-10	$7.2 \cdot 10^4$	1.8
400-LRF3200-MP800-LRF800-MP400-10	$1.6 \cdot 10^5$	**1.3**
400-SP1600-10 ($d_1 = 300$, $q = n$)	$3.2 \cdot 10^3$	2.6
400-SP1600-10 ($d_1 = 300$, $q = 32$)	$1.6 \cdot 10^3$	2.3
400-SP4800-10 ($d_1 = 300$, $q = 20$)	$5 \cdot 10^3$	1.8

Each sample is a graph!



(a) (b)

Figure 3: Subsampled MNIST examples.

Fixed graph: Vanilla Spectral Graph ConvNets

Spectral Networks and Deep Locally Connected Networks on Graphs, 2014, ICLR

Analysis

- ☺ First spectral graph CNN architecture
- ☹ No guarantee of spatial localization of filters
- ☹ $O(n)$ parameters per layer
- ☹ $O(n^2)$ computation of forward and inverse Fourier transforms $\phi, \phi^\top$ (no FFT on graphs)
- ☹ Filters are basis-dependent $\Rightarrow$ does not generalize across graphs

Fixed graph: ChebyNet

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, 2016, NIPS

- **Polynomial** parametrization for **localized** filters
  - $y = \mathbf{\Phi} g_\theta(\boldsymbol{\Lambda}) \mathbf{\Phi}^T x, \mathbf{\Phi}^T \mathbf{\Phi} = \boldsymbol{I}$
  - Polynomial filter

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

$$y = \mathbf{\Phi} \sum_{k=0}^{K-1} \theta_k \boldsymbol{\Lambda}^k \mathbf{\Phi}^T x = \sum_{k=0}^{K-1} \theta_k \boldsymbol{L}^k x$$

  - Chebyshev polynomial

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

✓ Cost: $\mathcal{O}(K|\mathcal{E}|) \ll \mathcal{O}(n^2)$

  - Why localized?

$$d_{\mathcal{G}}(i,j) > K \text{ implies } (L^K)_{i,j} = 0$$

Fixed graph: ChebyNet

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, 2016, NIPS

- Experiments
  - MNIST: each digit is a graph
  - Text categorization: 10,000 key words make up the graph.

$$W_{ij} = \exp\left(-\frac{\|z_i - z_j\|_2^2}{\sigma^2}\right)$$

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

Table 2: Accuracies of the proposed graph CNN and other methods on 20NEWS.

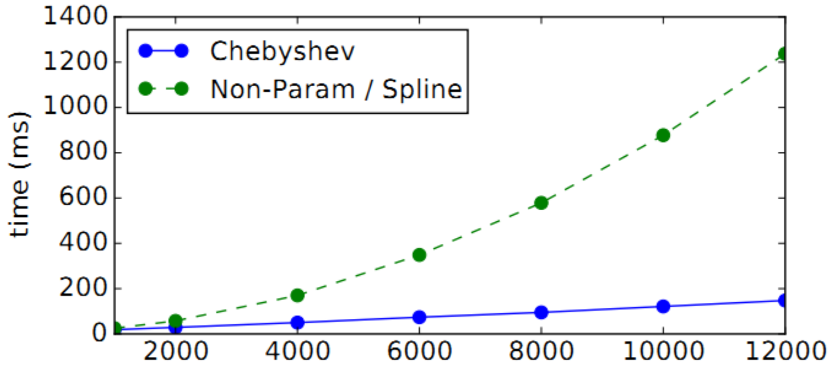


Figure 3: Time to process a mini-batch of $S = 100$ 20NEWS documents w.r.t. the number of words n.

		Accuracy		
Dataset	Architecture	Non-Param (2)	Spline (7) [4]	Chebyshev (4)
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Table 3: Classification accuracies for different types of spectral filters ($K = 25$).

Fixed graph: ChebyNet

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, 2016, NIPS

Analysis

- ☺ Filters are exactly localized in r-hops support
- ☺ O(1) parameters per layer
- ☺ No computation of ϕ, $\phi^\top$ $\Rightarrow$ $O(n)$ computational complexity (assuming sparsely-connected graphs)
- ☺ Stable under coefficients perturbation
- ☹ Filters are basis-dependent $\Rightarrow$ does not generalize across graphs

Fixed graph: GCN

Semi-Supervised Classification with Graph Convolutional Networks, 2017, ICLR

□ Simplification of ChebyNet

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x$$

K=1

$$\approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x$$

$$= \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

$$\approx \theta \underbrace{\left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right)}_{\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}} x$$

Fixed graph: GCN

Semi-Supervised Classification with Graph Convolutional Networks, 2017, ICLR

- Input-output

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

  - $X \in \mathbb{R}^{N \times C}$, C-d feature vector for N nodes.
  - $\Theta \in \mathbb{R}^{C \times F}$, matrix of filter parameters.
  - $Z \in \mathbb{R}^{N \times F}$, F-d output vector for N nodes.

- Two-layer network

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)}\right)$$

- Loss over **labeled examples**

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

Fixed graph: GCN

Semi-Supervised Classification with Graph Convolutional Networks, 2017, ICLR

- Datasets
  - Whole dataset as a graph: $N = N_{train} + N_{val} + N_{test}$

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

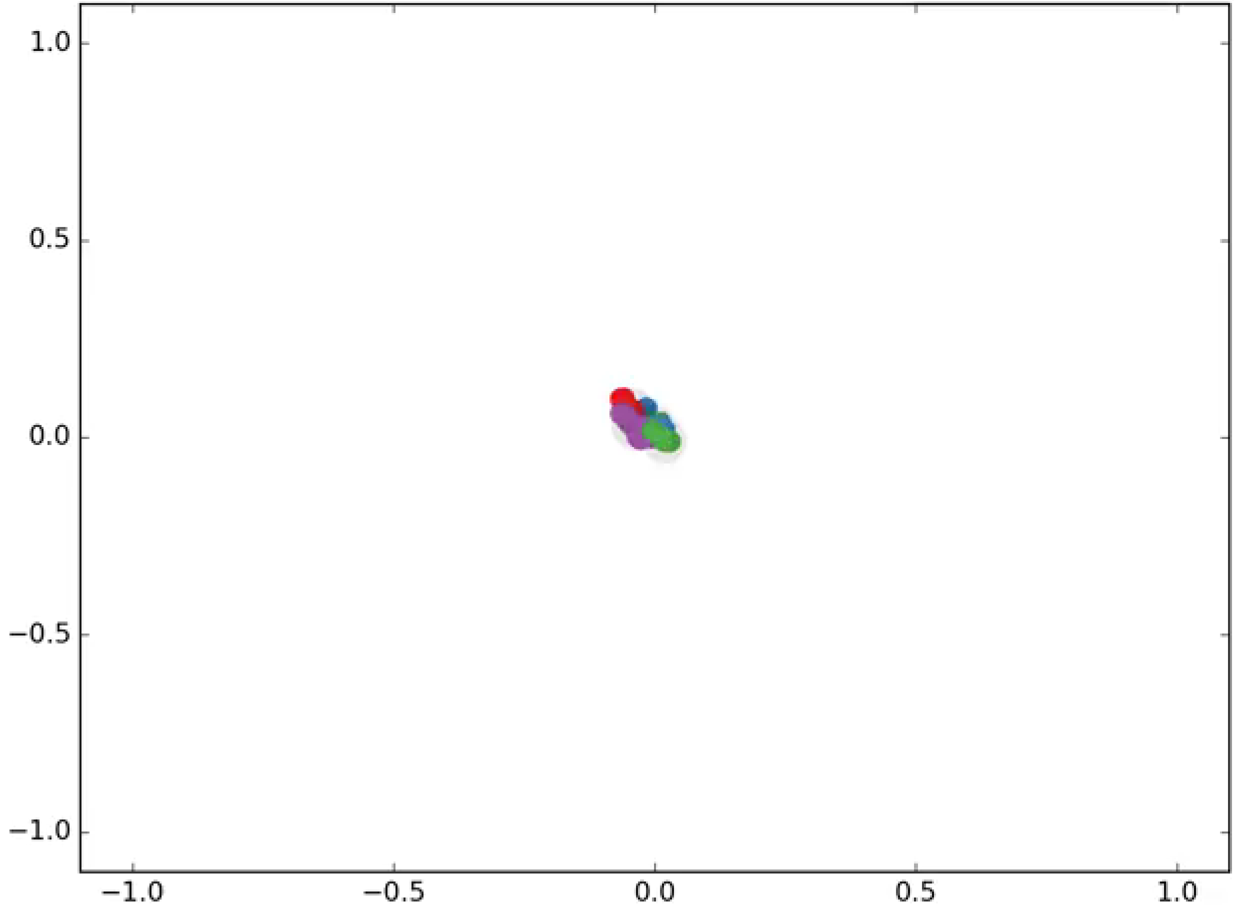
Table 2: Summary of results in terms of classification accuracy (in percent).

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

Fixed graph: GCN

Semi-Supervised Classification with Graph Convolutional Networks, 2017, ICLR

- Visulization (one labeled point for each class)



Fixed graph: CayleyNet

CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters, 2017

- Cayley transform

$$C(x) = \frac{x - i}{x + i}$$

- Cayley polynomial

$$g_{\mathbf{c},h}(\lambda) = c_0 + 2\mathrm{Re}\left\{ \sum_{j=1}^{r} c_j (h\lambda - i)^j (h\lambda + i)^{-j} \right\}$$

- Cayley filter

$$\mathbf{G} = c_0 \mathbf{I} + \sum_{j=1}^{r} c_j \mathcal{C}^j (h\boldsymbol{\Delta}) + \overline{c_j} \mathcal{C}^{-j} (h\boldsymbol{\Delta})$$

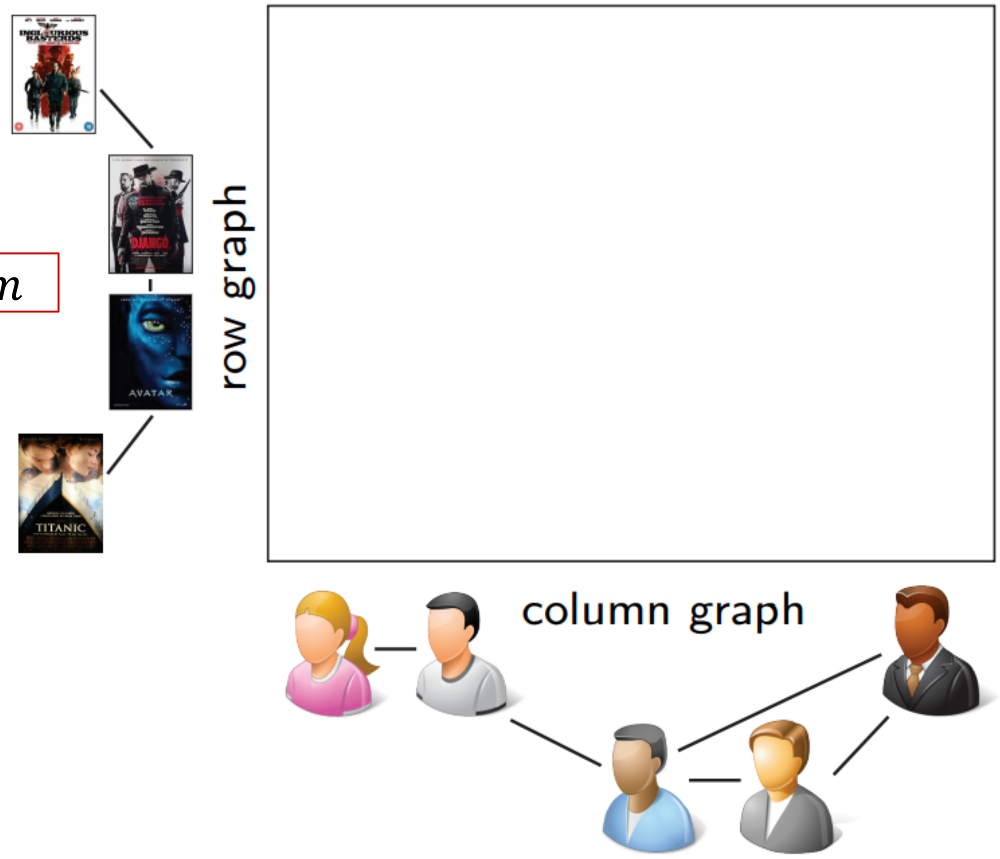
  - Any spectral filter can be formulated as a Cayley filter.

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

- Matrix ($\mathbb{R}^{m \times n}$) completion

$\mathcal{G}_r: row\ graph, m \times m$



$\mathcal{G}_c: col.\ graph, n \times n$

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

- Matrix ($\mathbb{R}^{m\times n}$) completion
  - Problem:

$$\min_{\mathbf{X}} \operatorname{rank}(\mathbf{X}) \quad \text{s.t.} \quad x_{ij} = y_{ij}, \ \forall ij \in \Omega \qquad \text{(NP-hard)}$$

$$\min_{\mathbf{X}} \|\mathbf{X}\|_\star + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|_{\mathrm{F}}^2$$

$\|\cdot\|_\star$: *sum of singular values*
$\|\cdot\|_F$: *Frobenius norm*

  - Geometric matrix completion

$$\min_{\mathbf{X}} \|\mathbf{X}\|_{\mathcal{G}_r}^2 + \|\mathbf{X}\|_{\mathcal{G}_c}^2 + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|_{\mathrm{F}}^2$$

$\|\cdot\|_{\mathcal{G}}$: *Dirichlet norm*
$\|\boldsymbol{X}\|_{\mathcal{G}_r}^2 = trace(\boldsymbol{X}^T\boldsymbol{\Delta}_r\boldsymbol{X})$
$\|\boldsymbol{X}\|_{\mathcal{G}_c}^2 = trace(\boldsymbol{X}\boldsymbol{\Delta}_c\boldsymbol{X}^T)$

  - Factorized model
    - ✓ Low-rank factorization (for large matrix): $\mathbf{X} = \mathbf{W}\mathbf{H}^\top$

$W, m \times r$
$H, n \times r$

$$\min_{\mathbf{W},\mathbf{H}} \frac{1}{2}\|\mathbf{W}\|_{\mathrm{F}}^2 + \frac{1}{2}\|\mathbf{H}\|_{\mathrm{F}}^2 + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{W}\mathbf{H}^\top - \mathbf{Y})\|_{\mathrm{F}}^2$$

Graph-based

$$\min_{\mathbf{W},\mathbf{H}} \frac{1}{2}\|\mathbf{W}\|_{\mathcal{G}_r}^2 + \frac{1}{2}\|\mathbf{H}\|_{\mathcal{G}_c}^2 + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{W}\mathbf{H}^\top - \mathbf{Y})\|_{\mathrm{F}}^2$$

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

- Multi-graph CNNs (MGCNN)
  - ***2-d Fourier transform** of an matrix can be thought of as applying a **1-d Fourier transform to its rows and columns**.*

$$\hat{\mathbf{X}} = \mathbf{\Phi}_r^\top \mathbf{X} \mathbf{\Phi}_c$$

$\Phi_r, eigenvecors\ w.r.t\ \mathcal{G}_r$
$\Phi_c, eigenvecors\ w.r.t\ \mathcal{G}_c$

  - Multi-graph spectral convolution

$$\mathbf{X} \star \mathbf{Y} = \mathbf{\Phi}_r (\hat{\mathbf{X}} \circ \hat{\mathbf{Y}}) \mathbf{\Phi}_c^\top$$

  - p-order Chebyshev polynomial filters

$$\tilde{\mathbf{X}}_l = \xi\left(\sum_{l'=1}^{q'} \mathbf{X}_{l'} \star \mathbf{Y}_{ll'}\right) = \xi\left(\sum_{l'=1}^{q'} \sum_{j,j'=0}^{p} \theta_{jj',ll'} T_j(\tilde{\mathbf{\Delta}}_r) \mathbf{X}_{l'} T_{j'}(\tilde{\mathbf{\Delta}}_c)\right), \quad l = 1, \ldots, q$$

$input\ dim.: m \times n \times q'$
$output\ dim.: m \times n \times q$

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

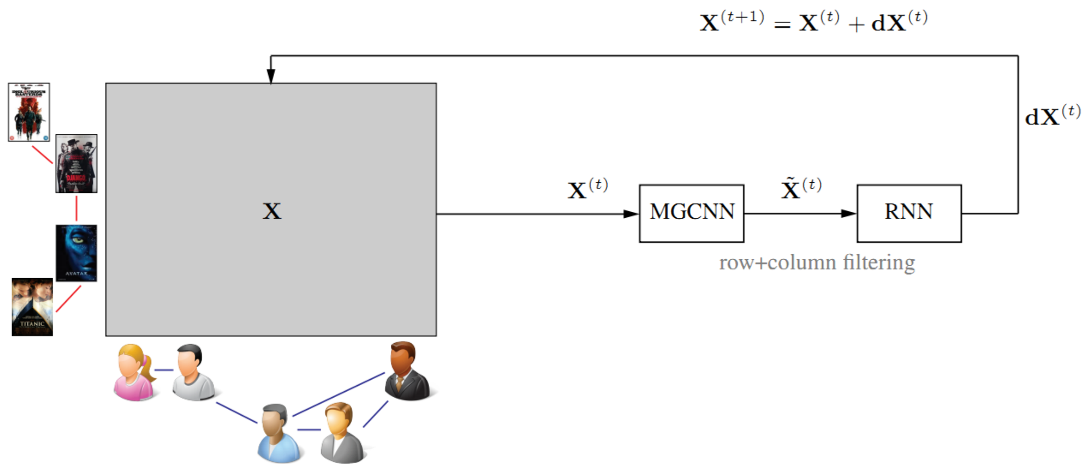
- Separable convolution (sMGCNN)
  - Complexity: $\mathcal{O}(m+n) < \mathcal{O}(mn)$

$$\tilde{\mathbf{w}}_l = \xi\left(\sum_{l'=1}^{q'}\sum_{j=0}^{p}\theta^r_{j,ll'}T_j(\tilde{\mathbf{\Delta}}_r)\mathbf{w}_{l'}\right), \qquad \tilde{\mathbf{h}}_l = \xi\left(\sum_{l'=1}^{q'}\sum_{j'=0}^{p}\theta^c_{j',ll'}T_{j'}(\tilde{\mathbf{\Delta}}_c)\mathbf{h}_{l'}\right)$$

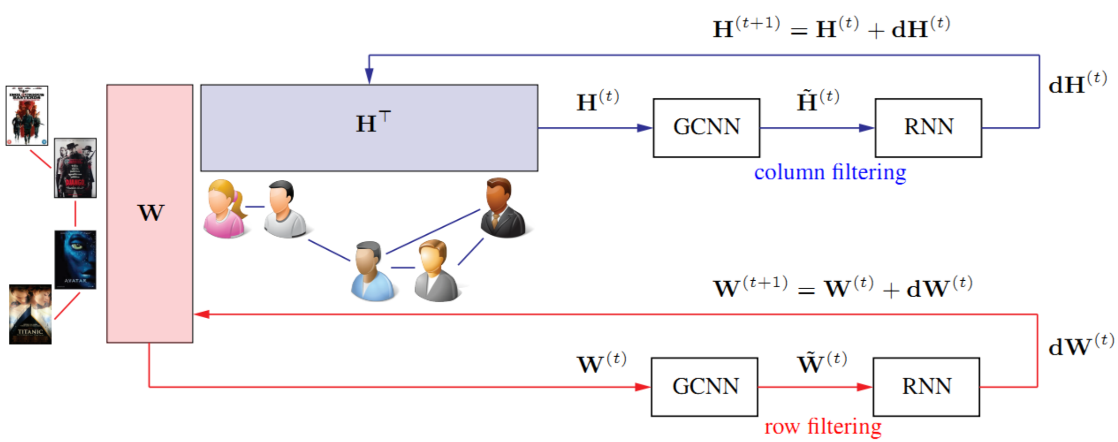
Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

- Architectures
  - RNN: diffuse the score values $\tilde{X}^{(t)}$ progressively.



MGCNN



sMGCNN

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

- Loss
  - $\Theta, \theta_r, \theta_c$: *chebyshev polymial coefficients*
  - σ: *LSTM*, T: *number of iterations*
  - MGCNN

$$\ell(\boldsymbol{\Theta}, \boldsymbol{\sigma}) = \|\mathbf{X}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}^{(T)}\|_{\mathcal{G}_r}^2 + \|\mathbf{X}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}^{(T)}\|_{\mathcal{G}_c}^2 + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{X}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}^{(T)} - \mathbf{Y})\|_{\mathrm{F}}^2.$$

  - sMGCNN

$$\ell(\boldsymbol{\theta}_r, \boldsymbol{\theta}_c, \boldsymbol{\sigma}) = \|\mathbf{W}_{\boldsymbol{\theta}_r,\boldsymbol{\sigma}}^{(T)}\|_{\mathcal{G}_r}^2 + \|\mathbf{H}_{\boldsymbol{\theta}_c,\boldsymbol{\sigma}}^{(T)}\|_{\mathcal{G}_c}^2 + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{W}_{\boldsymbol{\theta}_r,\boldsymbol{\sigma}}^{(T)}(\mathbf{H}_{\boldsymbol{\theta}_c,\boldsymbol{\sigma}}^{(T)})^\top - \mathbf{Y})\|_{\mathrm{F}}^2$$

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

- Algorithm

Algorithm 1 (RMGCNN)

input $m \times n$ matrix $\mathbf{X}^{(0)}$ containing initial values

for $t = 0 : T$ **do**
  Apply the Multi-Graph CNN ($\boxed{13}$) on $\mathbf{X}^{(t)}$ producing an $m \times n \times q$ output $\tilde{\mathbf{X}}^{(t)}$.
  for all elements (i, j) **do**
    Apply RNN to q-dim $\tilde{\mathbf{x}}_{ij}^{(t)} = (\tilde{x}_{ij1}^{(t)}, \ldots, \tilde{x}_{ijq}^{(t)})$ producing incremental update $dx_{ij}^{(t)}$
  end for
  Update $\mathbf{X}^{(t+1)} = \mathbf{X}^{(t)} + \mathbf{dX}^{(t)}$
end for

Fixed graph: Multiple graphs

Geometric matrix completion with recurrent multi-graph neural networks, 2017, NIPS

Results

- MovieLens dataset:
  - ✓ 100,000 ratings (1-5) from 943 users on 1682 movies (6.3%).
  - ✓ Each user has rated at least 20 movies.
  - ✓ User: user id | age | gender | occupation | zip code
  - ✓ Movie: movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | ……

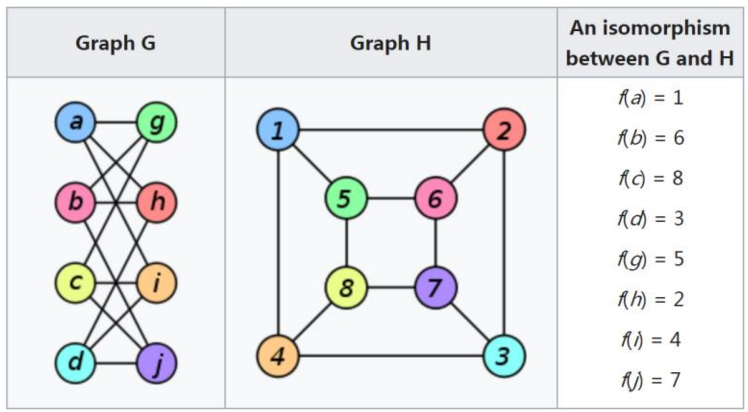
Table 4: Performance (RMS error) of different matrix completion methods on the MovieLens dataset.

METHOD	RMSE
GLOBAL MEAN	1.154
USER MEAN	1.063
MOVIE MEAN	1.033
MC [9]	0.973
IMC [17, 42]	1.653
GMC [19]	0.996
GRALS [33]	0.945
sRMGCNN	**0.929**

Variable graph: GraphSAGE

Inductive Representation Learning on Large Graphs, 2017, NIPS

- Desiderata => well generalized.
  - Invariant to node ordering
    - ✓ No graph isomorphism problem (https://en.wikipedia.org/wiki/Graph_isomorphism)

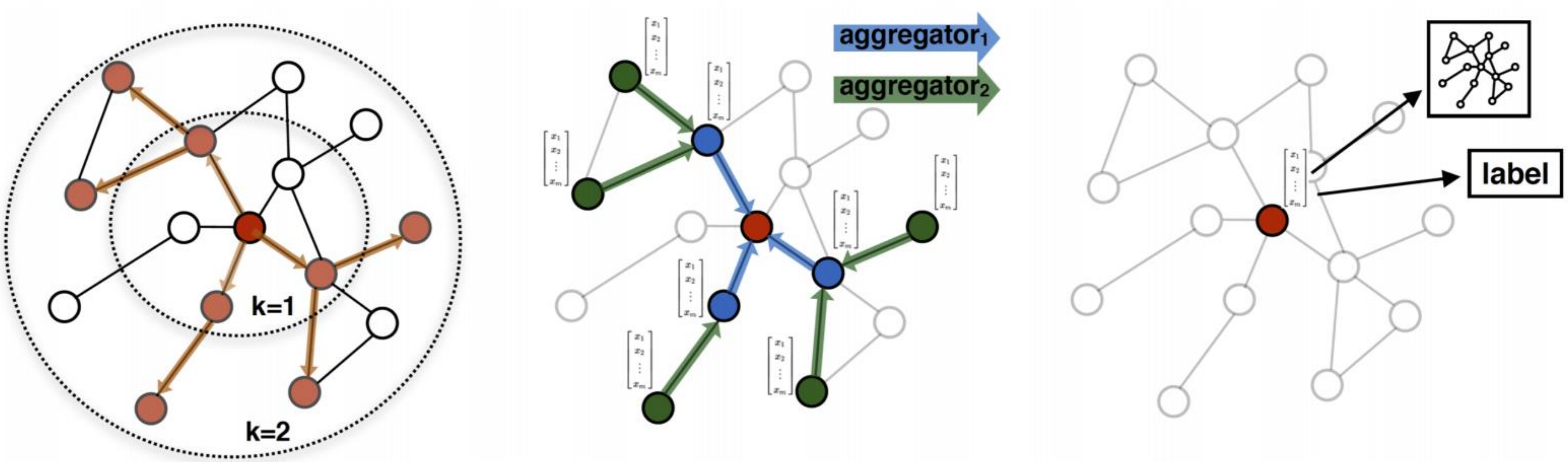


  - Locality
    - ✓ Operations depend on the neighbors of a given node
  - Number of model parameters should be independent of graph size
  - Model should be independent of graph structure and we should be able to transfer the model across graphs.

Variable graph: GraphSAGE

Inductive Representation Learning on Large Graphs, 2017, NIPS

- Learn to propagate information across the graph to compute node features.



1. Sample neighborhood

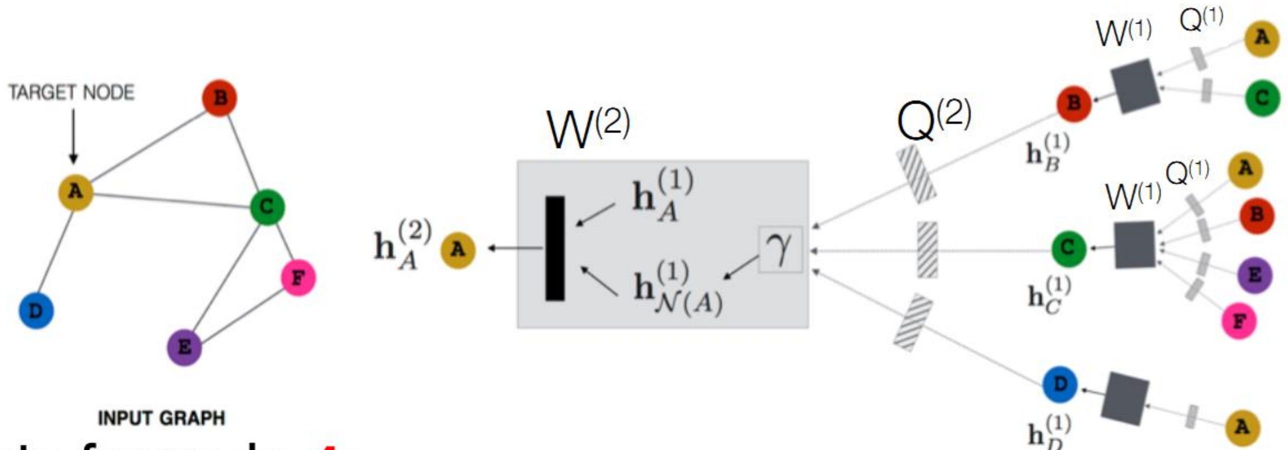
2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Variable graph: GraphSAGE

Inductive Representation Learning on Large Graphs, 2017, NIPS

- Update
  - $h_A^{(0)}: attribute\ of\ node\ A$
  - $\sum(\cdot): aggregator\ function(e.g., avg/lstm/max-pooling)$



Update for node *A*:

$$\underbrace{h_A^{(k+1)}}_{k+1^{th}\text{ level features of node }A} = ReLU\left(\underbrace{W^{(k)} h_A^{(k)}}_{\text{Transform }A\text{'s own features from level }k}, \sum_{n\in\mathcal{N}(A)} \underbrace{\left(ReLU\left(Q^{(k)} h_n^{(k)}\right)\right)}_{\text{Transform and aggregate features of neighbors }n}\right)$$

Variable graph: GraphSAGE

Inductive Representation Learning on Large Graphs, 2017, NIPS

Algorithm

initialize representations as features

K = "search depth"

aggregate information from neighbors

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;

for $k = 1...K$ **do**

  for $v \in \mathcal{V}$ **do**

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

  end

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

concatenate neighborhood info with current representation and propagate

$$J = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - \frac{1}{|Q|} \cdot \sum_{q=1}^{Q} \mathbb{E}_{v_n \sim P_n(v)} \log\left(-\sigma(\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

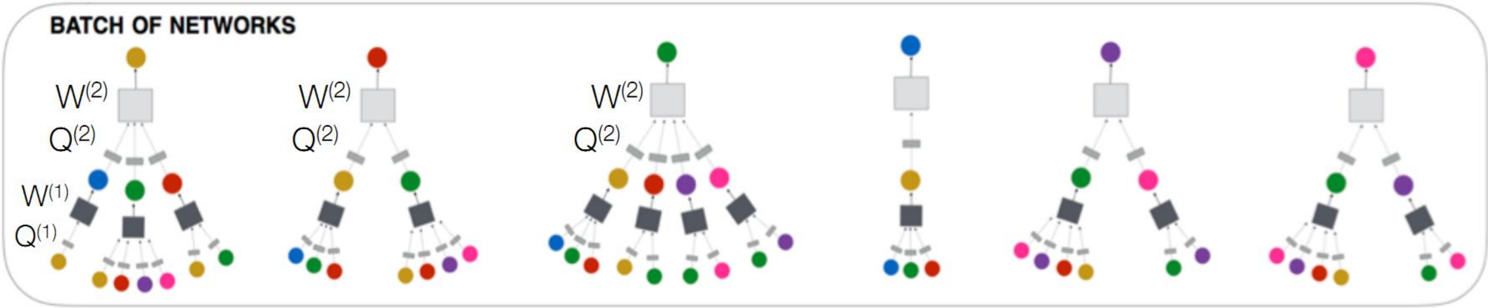
classification (cross-entropy) loss

Variable graph: GraphSAGE

Inductive Representation Learning on Large Graphs, 2017, NIPS

Training

- Batch

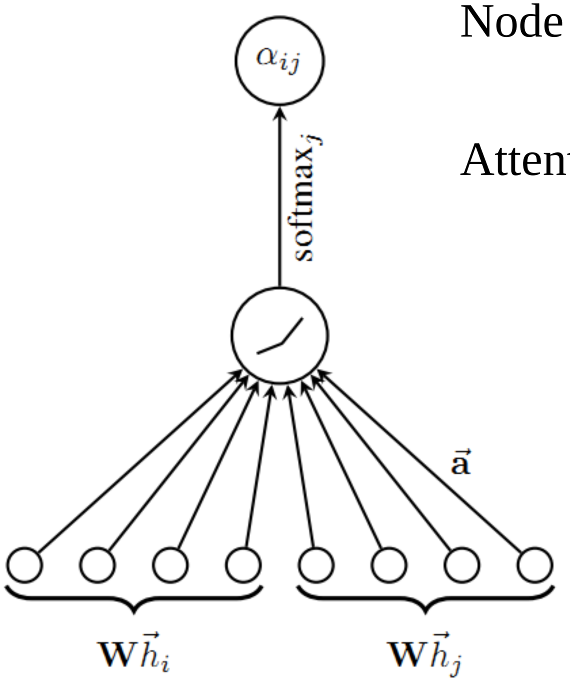


- Learnable parameters
  - ✓ Aggregate function
  - ✓ Matrix W

Variable graph: Graph Attention Network

Graph attention networks, 2018, ICLR

- Specify different weights to different nodes in a neighbor.
  - Self-attention



Node features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$

Attention: importance of node *j* to node *i*.

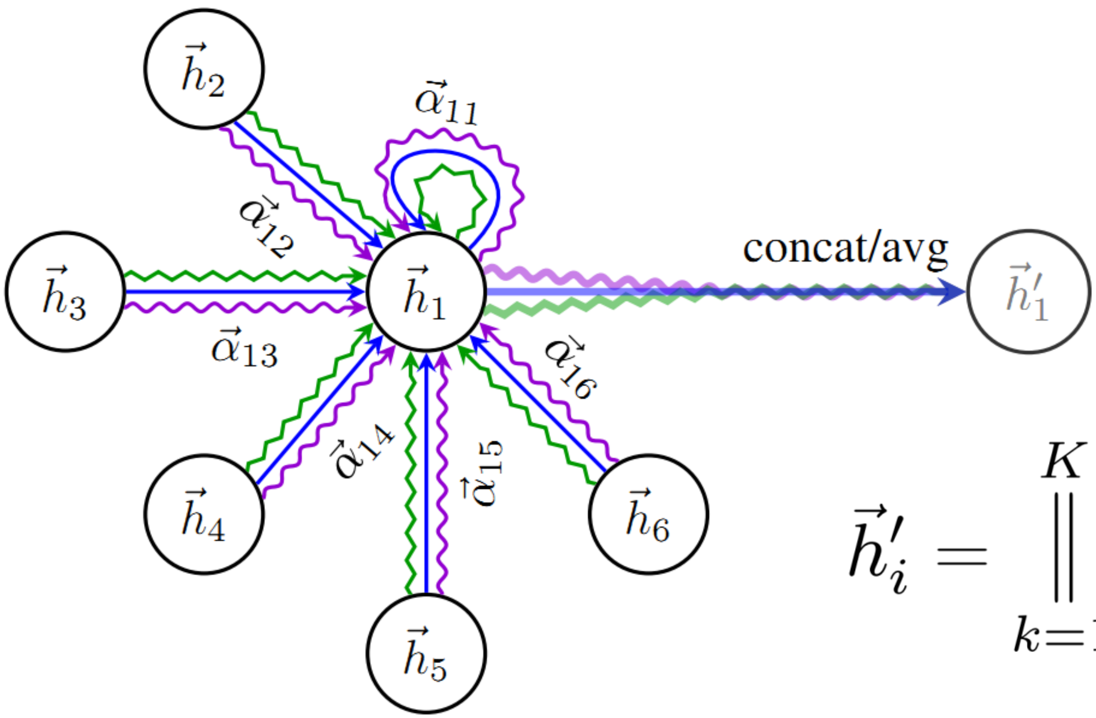
$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j) \quad j \in \mathcal{N}_i$$

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

Variable graph: Graph Attention Network

Graph attention networks, 2018, ICLR

- Specify different weights to different nodes in a neighbor.
  - Aggregation (K-head attention)



$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

Variable graph: Graph Attention Network

Graph attention networks, 2018, ICLR

- Experiments
  - Datasets

Table 1: Summary of the datasets used in our experiments.

	Cora	**Citeseer**	**Pubmed**	**PPI**
Task	Transductive	Transductive	Transductive	Inductive
# Nodes	2708 (1 graph)	3327 (1 graph)	19717 (1 graph)	56944 (24 graphs)
# Edges	5429	4732	44338	818716
# Features/Node	1433	3703	500	50
# Classes	7	6	3	121 (multilabel)
# Training Nodes	140	120	60	44906 (20 graphs)
# Validation Nodes	500	500	500	6514 (2 graphs)
# Test Nodes	1000	1000	1000	5524 (2 graphs)

Variable graph: Graph Attention Network

Graph attention networks, 2018, ICLR

- Experiments
  - Transductive learning (single fixed graph)
  - Inductive learning (unseen nodes / new graph)

Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	$81.7 \pm 0.5\%$	—	$78.8 \pm 0.3\%$
GCN-64*	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	$\mathbf{79.0} \pm 0.3\%$
GAT (ours)	$\mathbf{83.0} \pm 0.7\%$	$\mathbf{72.5} \pm 0.7\%$	$\mathbf{79.0} \pm 0.3\%$

Inductive

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	$\mathbf{0.973} \pm 0.002$

Tasks

- Citation networks
- Recommender systems
- Medical imaging
- Particle physics and Chemistry
- Computer graphics
- ……