Graph Convolutional Networks
Spectral & Spatial Techniques

Xavier Bresson

School of Computer Science and Engineering
Data Science and AI Research Centre
Nanyang Technological University (NTU), Singapore

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

NATIONAL RESEARCH FOUNDATION
PRIME MINISTER'S OFFICE
SINGAPORE

Summer School on Data Science (SSDS 2020)
Sven Loncaric, Tomislav Smuc, Vinko Zlatic, Tomislav Lipic
Sept 11th 2020

Outline

- Part 1: Traditional ConvNets
  - Architecture
  - Graph Domain
  - Convolution
- Part 2: Spectral Graph ConvNets
  - Spectral Convolution
  - Spectral GCNs
- Part 3: Spatial Graph ConvNets
  - Template Matching
  - Isotropic GCNs
  - Anisotropic GCNs
  - Lab on GatedGCNs
- Part 4: GNN Tasks
- Conclusion

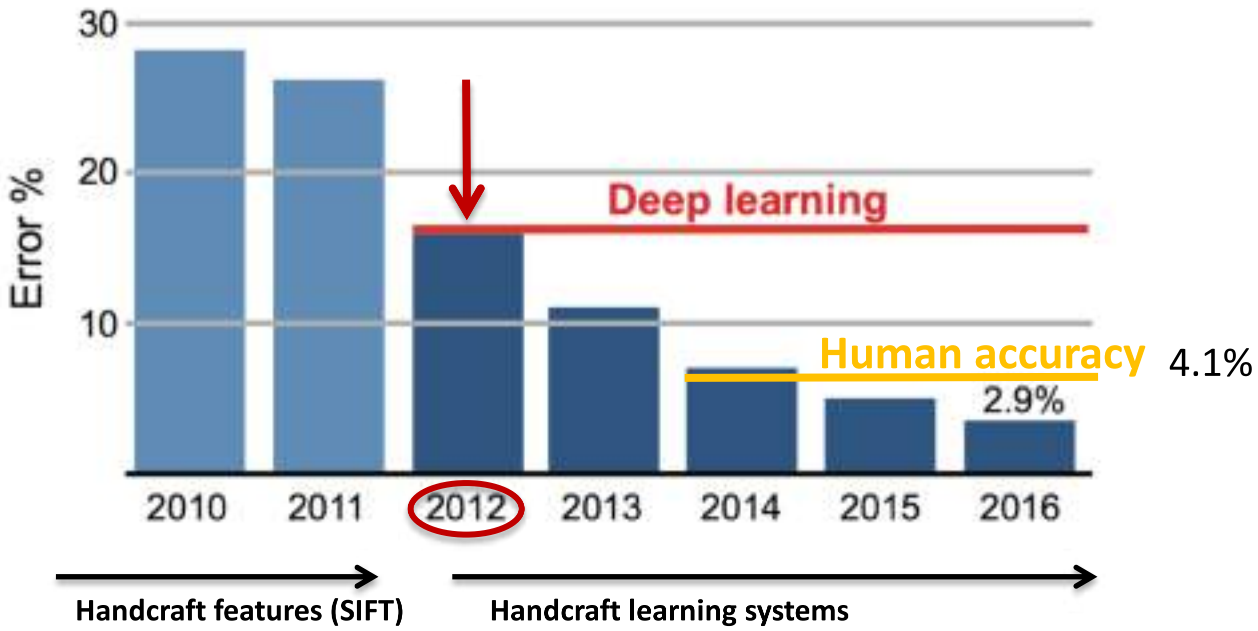
Outline

ConvNets

- A breakthrough in Computer Vision :

  LeCun, Bottou, Bengio, Haffner 1998
  Krizhevsky, Sutskever, Hinton, 2012

IMAGENET

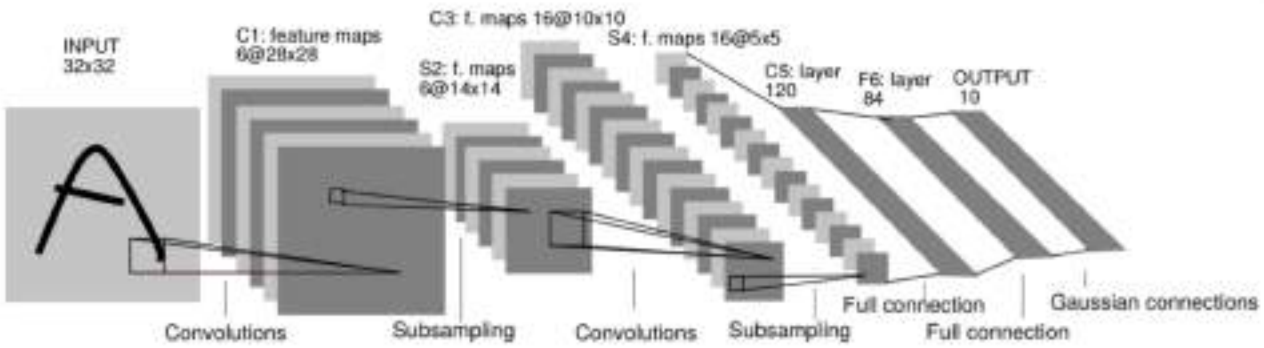


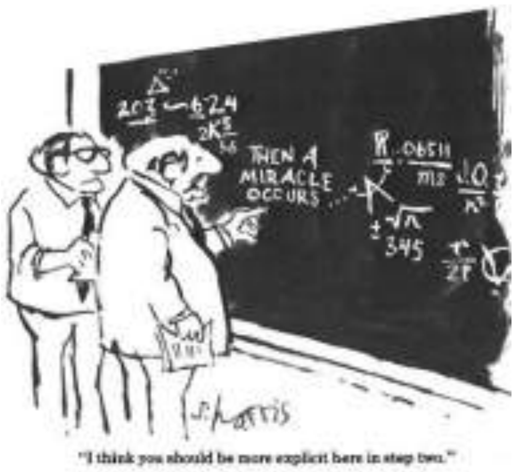
- Also in Speech and Natural Language Processing.

ConvNets

- ConvNets are powerful architectures to solve high-dimensional learning problems.

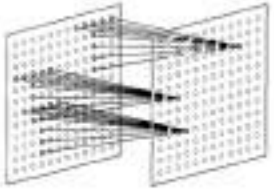
- Curse of dimensionality :
  dim(image) = 1024 x 1024 ≈ 10^6
  For N=10 samples/dim ⇒ $10^{1,000,000}$ points





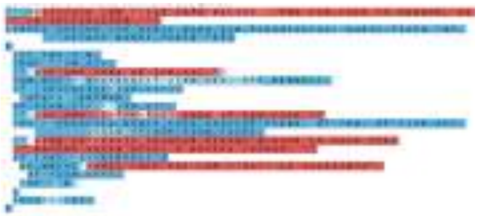
ConvNets

- Main assumption :
  - Data (images, videos, speech) is compositional, it is formed of patterns that are:
    - Local (Hubel-Wiesel 1962)
    - Stationary (shared patterns)
    - Hierarchical (multi-scale)



- ConvNets leverage the compositionality structure :
  - They extract compositional features and feed them to classifier, recommender, etc (end-to-end systems).

Computer Vision

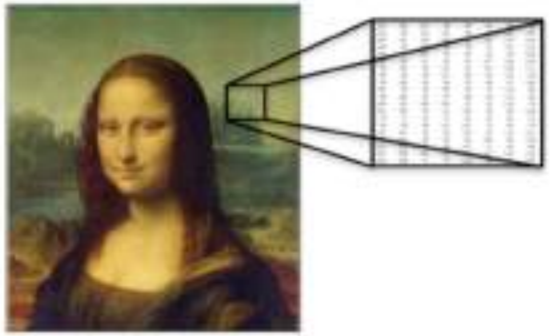

NLP

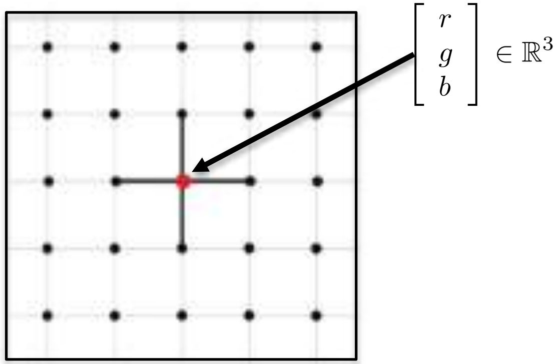

Speech


Game of Go

Outline

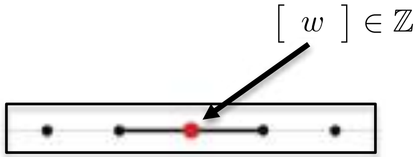
Data Domain





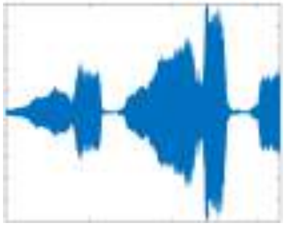
2D grid

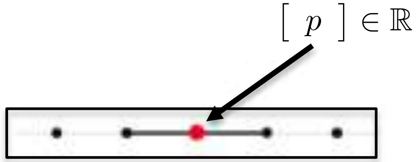
- Images, volumes, videos lie on
  2D, 3D, 2D+1 Euclidean domains (grids)



1D grid

- Sentences, words, speech lie on
  1D Euclidean domain





1D grid

- These domains have strong regular spatial structures.
  - All ConvNet operations are mathematically well defined and fast (convolution, pooling).

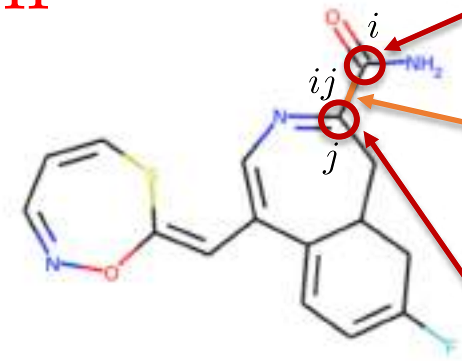
Graph Domain

Social networks (Advertisement/ recommendation)

$\text{User}_\text{i} \begin{bmatrix} \text{messages} \\ \text{images} \\ \text{videos} \end{bmatrix}_i \in \mathbb{R}^d$

$\text{User connection}_\text{ij} \quad A_{ij} = \begin{cases} 1 & \text{if ij friends} \\ 0 & \text{otherwise} \end{cases}$

$\text{User}_\text{j} \begin{bmatrix} \text{messages} \\ \text{images} \\ \text{videos} \end{bmatrix}_j \in \mathbb{R}^d$



Quantum Chemistry (novel molecules for drugs and materials)

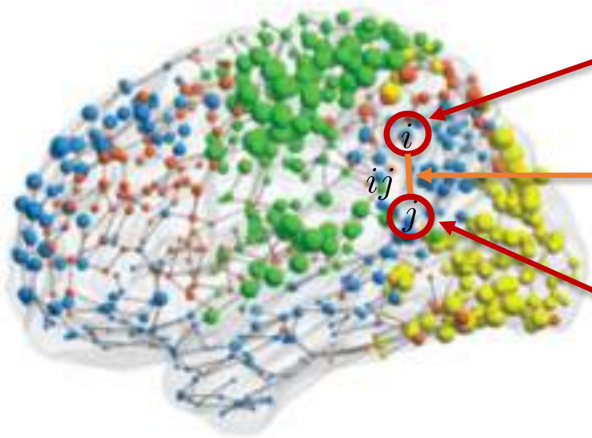
$\text{Atom}_\text{i} \begin{bmatrix} \text{type} \\ \text{coordinates} \\ \text{charge} \end{bmatrix}_i \in \mathbb{R}^{d_v}$

$\text{Bond}_\text{ij} \quad A_{ij} = \begin{cases} 1 & \text{if ij bond} \\ 0 & \text{otherwise} \end{cases}$

$\begin{bmatrix} \text{type} \\ \text{energy} \end{bmatrix}_{ij} \in \mathbb{R}^{d_e}$

$\text{Atom}_\text{j} \begin{bmatrix} \text{type} \\ \text{coordinates} \\ \text{charge} \end{bmatrix}_j \in \mathbb{R}^{d_v}$

Brain connectivity (sMRI)



Brain analysis (Neuroscience/neuro-diseases)

$\text{ROI}_\text{i} \begin{bmatrix} a_1 \\ \vdots \\ a_T \end{bmatrix}_i \in \mathbb{R}^T$

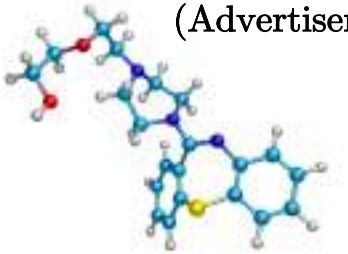
$\text{Cerebral connection}_\text{ij} \quad A_{ij} \in \mathbb{R}_+$

$\text{ROI}_\text{j} \begin{bmatrix} a_1 \\ \vdots \\ a_T \end{bmatrix}_j \in \mathbb{R}^T$

Functional activations (fMRI)

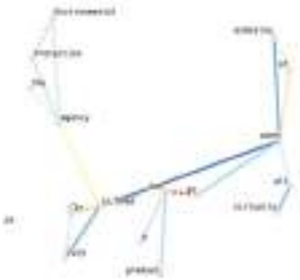
Graph Domain

Social networks (Advertisement)



Drug/Material molecules (Chemistry)

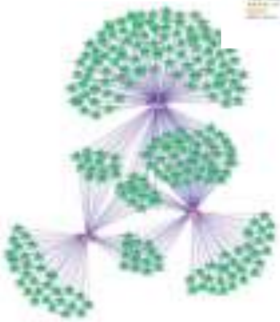
Brain connectivity (Neuroscience)



Words relationships (NLP)

Scene understanding

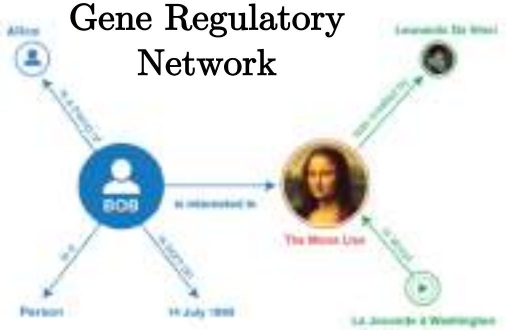
Recommender systems (Amazon, Netflix)



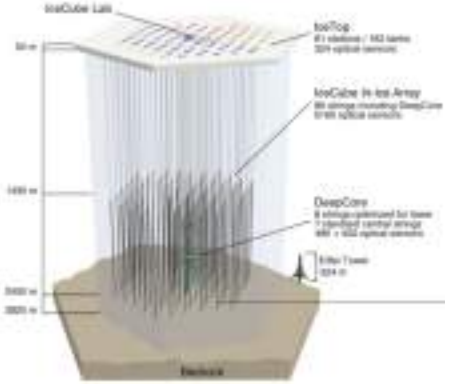
Gene Regulatory Network

3D Meshes (Computer Graphics)

Transportation networks



Knowledge graphs



Neutrino detection (High-energy Physics)

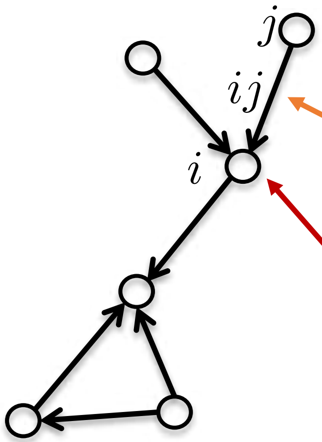
=

Graph

Graph Domain

- Graphs G are defined by :
  - Vertices V
  - Edges E
  - Adjacency matrix A

=



$$h_j = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_v}$$

$$e_{ij} = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_e}$$

$$h_i = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_v}$$

- Graph features :
  - Node features : $h_i,\ h_j$ (atom type)
  - Edge features : e_{ij} (bond type)
  - Graph features : g (molecule energy)

$$\underbrace{\begin{gathered}\mathcal{G} = (V, E, A) \\ V = \{1, .., n\}\end{gathered}}$$

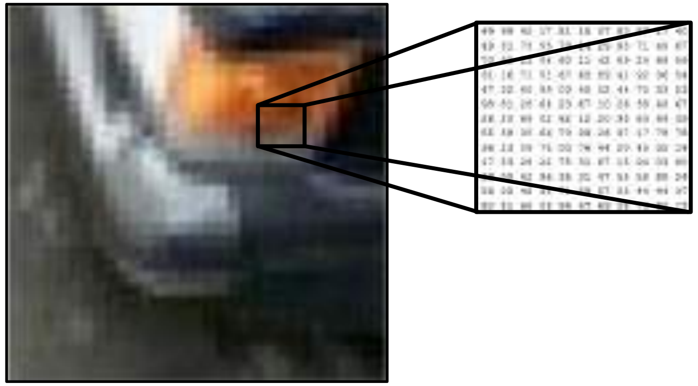
$$g = \begin{bmatrix} \vdots \end{bmatrix} \in \mathbb{R}^{d_g}$$

Outline

Convolution

- Convolutional layer (for grids) :

$$\underbrace{h^{\ell+1}}_{n_1 \times n_2 \times d} = \underbrace{w^{\ell}}_{3 \times 3 \times d} * \underbrace{h^{\ell}}_{n_1 \times n_2 \times d}$$



$*$

$=$

h^{ℓ}

Image/Hidden features

w^{ℓ}

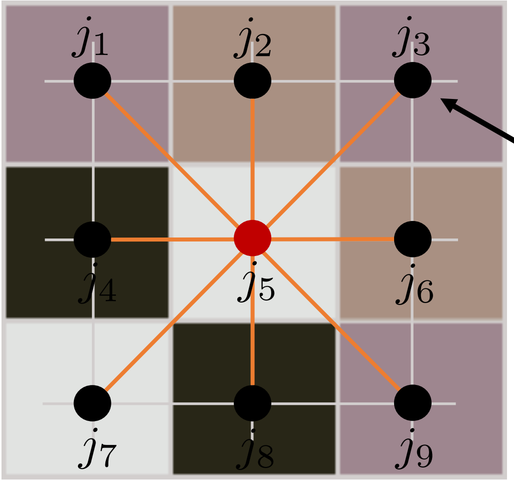
Pattern/kernel (learned by backpropagation)

$h^{\ell+1}$

Convolution

- How to define convolution ?
  - Definition #1 : Convolution as template matching
  - $O(n)$ by parallelization and for compact support patterns

$$
\begin{aligned}
\underset{n_1 \times n_2 \times d}{h_i^{\ell+1}} &= \overset{3 \times 3 \times d}{w^\ell} * h_i^\ell \\
&= \sum_{\underbrace{j \in \Omega}_{j \in \mathcal{N}_i}} \langle w_j^\ell, \underbrace{h_{i-j}^\ell}_{h_{i+j}^\ell = h_{ij}^\ell} \rangle \\
&= \sum_{j \in \mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle \\
&= \sum_{j \in \mathcal{N}_i} \langle \begin{bmatrix} w_j^\ell \end{bmatrix}, \begin{bmatrix} h_{ij}^\ell \end{bmatrix} \rangle
\end{aligned}
$$



Node j_3 is always located at the top-right corner of the pattern.

$$\begin{bmatrix} w_{j_3}^\ell \end{bmatrix} \in \mathbb{R}^d$$

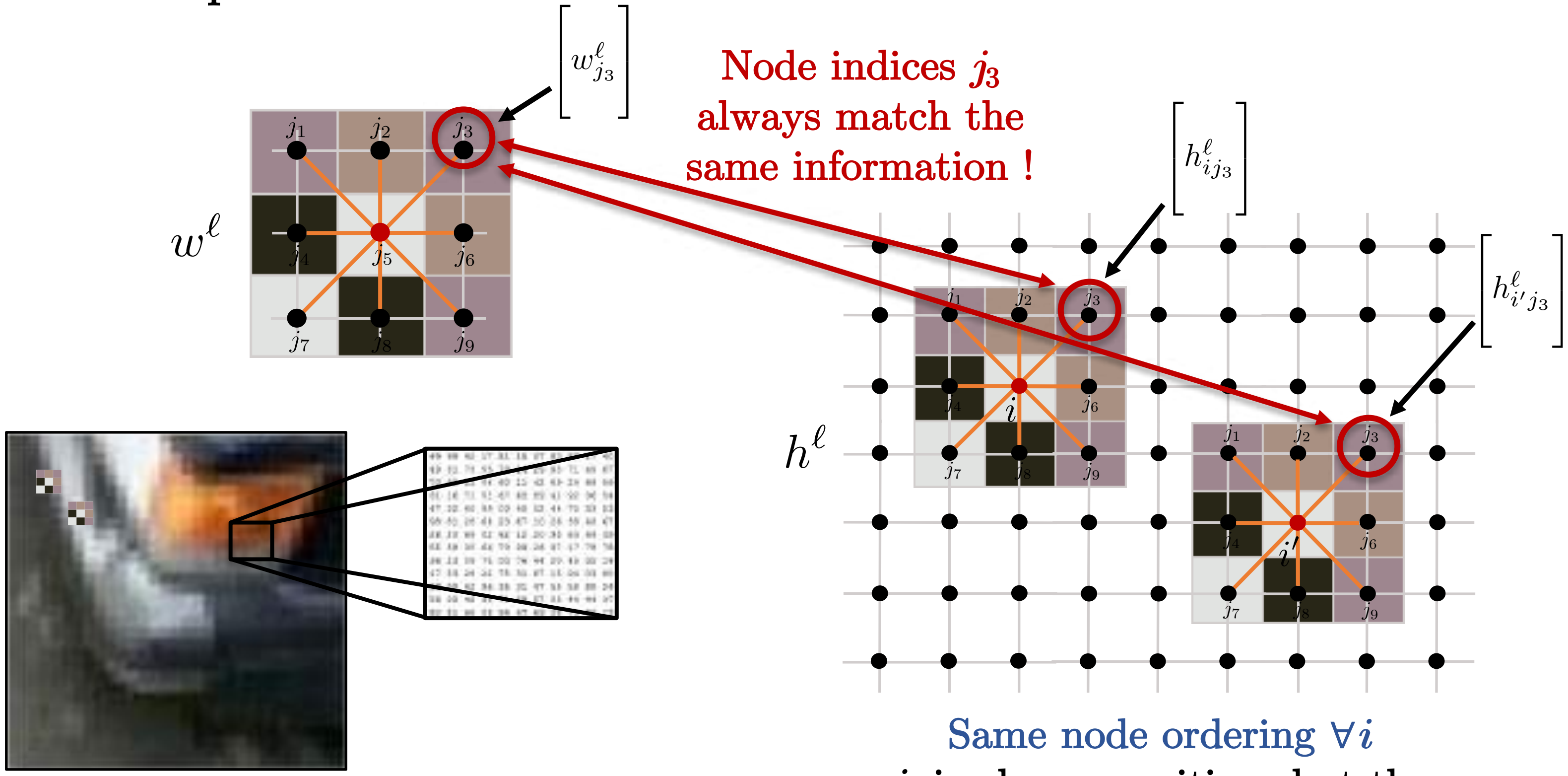
Template features at j_3

All nodes of the template w^l are always ordered/positioned the same way !

w^ℓ

Convolution

- How to define convolution ?
  - Definition #1 : Convolution as template matching
  - $O(n)$ by parallelization for compact support patterns



Same node ordering $\forall i$

j_3 is always positioned at the top-right corner of the template.

$$
\begin{aligned}
h_i^{\ell+1} &= w^\ell * h_i^\ell \\
&= \sum_{j\in\mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle \\
&= \sum_{j\in\mathcal{N}_i} \left\langle \left[w_j^\ell \right], \left[h_{ij}^\ell \right] \right\rangle
\end{aligned}
$$

$$\left\langle \left[w_{j_3}^\ell \right], \left[h_{ij_3}^\ell \right] \right\rangle$$

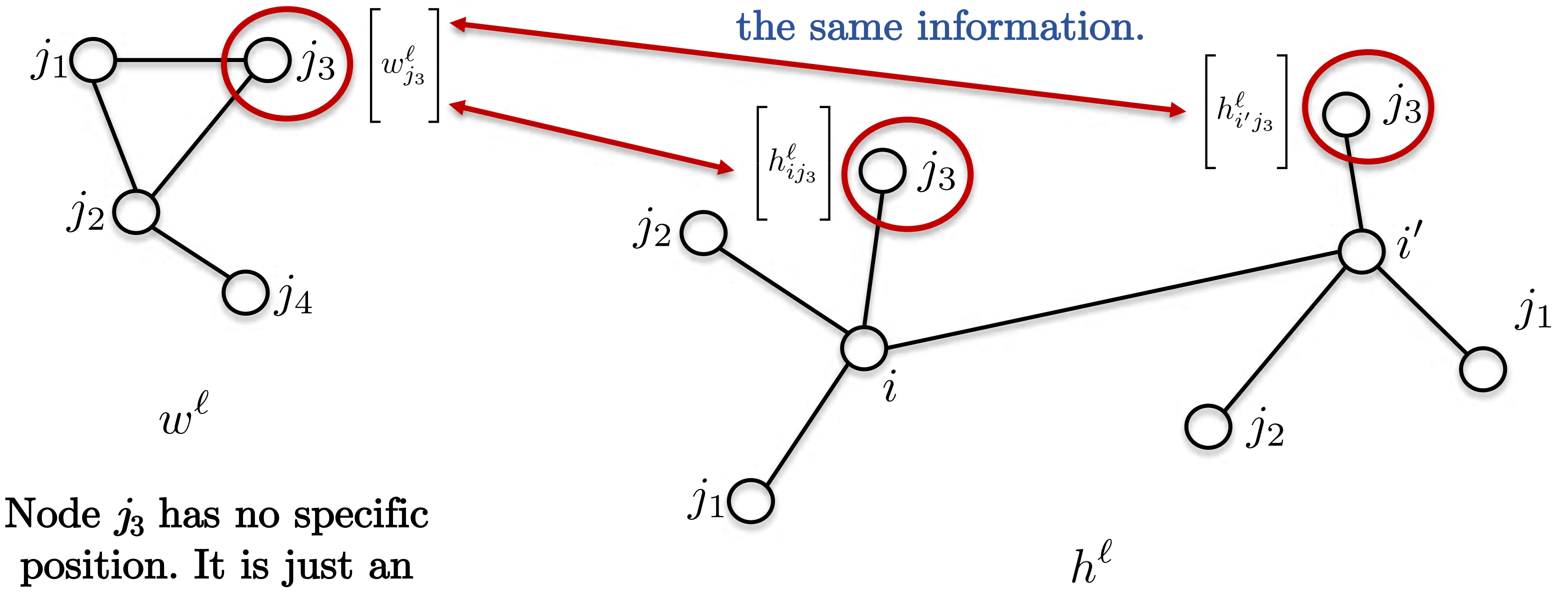
$$\left\langle \left[w_{j_3}^\ell \right], \left[h_{i'j_3}^\ell \right] \right\rangle$$

These matching scores are always for the top-right corner between the template and the image patches.

Graph Convolution

- Can we extend template matching for graphs ?
  - Main issues :
    - No node ordering : How to match template features with data features when nodes have no given position (index is not a position) ?

Node indices j_3 do not match the same information.



Node j_3 has no specific position. It is just an (arbitrary) index.

No node ordering on graphs :
The correspondence of nodes is lost on graphs.
There is no up, down, right and left on graphs.

$$\left\langle \begin{bmatrix} w^\ell_{j_3} \end{bmatrix}, \begin{bmatrix} h^\ell_{ij_3} \end{bmatrix} \right\rangle$$

$$\left\langle \begin{bmatrix} w^\ell_{j_3} \end{bmatrix}, \begin{bmatrix} h^\ell_{i'j_3} \end{bmatrix} \right\rangle$$

These matching scores have no meaning as they do not compare the same information.

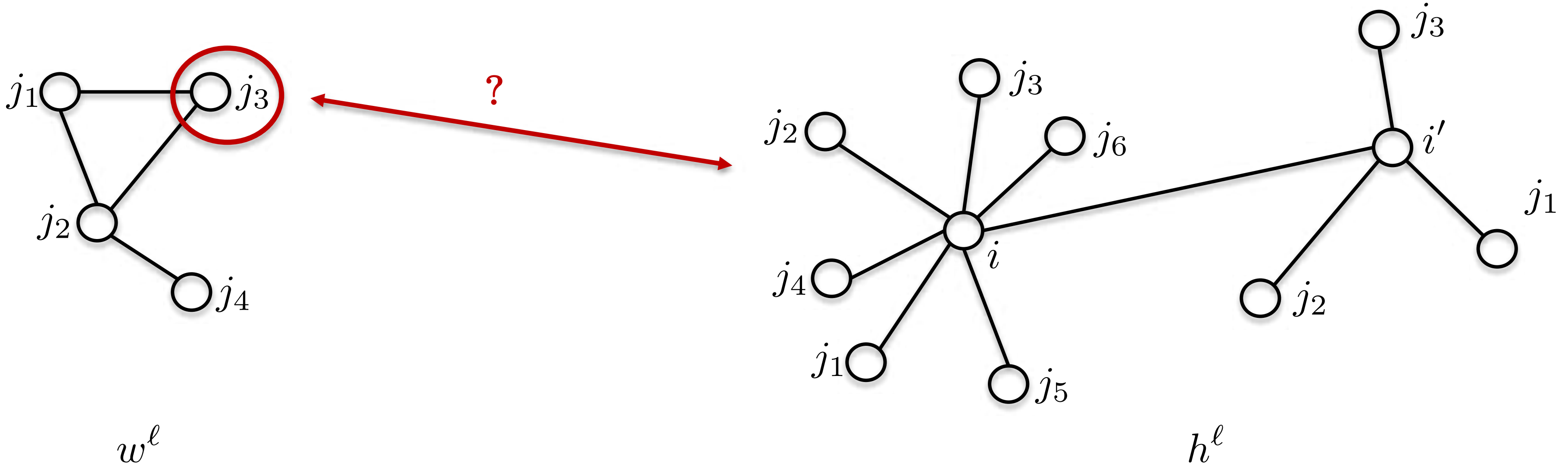
$$h_i^{\ell+1} = w^\ell *_{\mathcal{G}} h_i^\ell$$

$$= \cancel{\sum_{j \in \mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle}$$

$n \times d$

Graph Convolution

- Can we extend template matching for graphs ?
  - Main issues :
    - No node ordering : How to match template features with data features ?
    - Heterogeneous neighborhood : How to deal with different neighborhood sizes ?



Graph Convolution

- How to define convolution ?
  - Definition #1 : Template matching
  - Definition #2 : Convolution theorem
    - Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms

$$\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h) \quad \Rightarrow \quad w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

    - Generic Fourier transform has $O(n^2)$ complexity, but if the domain is a grid then complexity can be reduced to $O(n\log n)$ with FFT[1].

    - Can we extend the Convolution theorem to graphs ?
      - How to define Fourier transform for graphs ?
      - How to compute fast spectral convolutions in $O(n)$ time for compact kernels ?

$$w *_{\mathcal{G}} h \stackrel{?}{=} \mathcal{F}_{\mathcal{G}}^{-1}(\mathcal{F}_{\mathcal{G}}(w) \odot \mathcal{F}_{\mathcal{G}}(h))$$

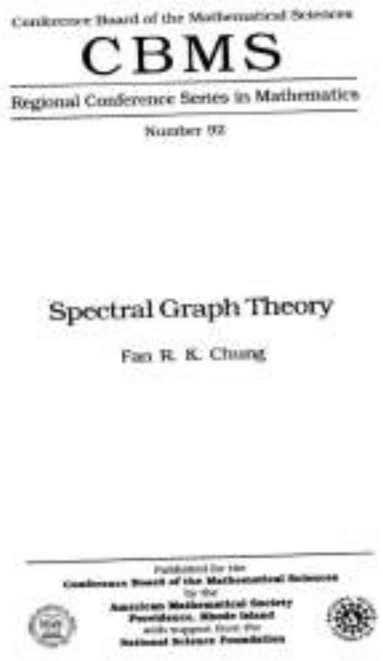
[1] JW Cooley, JW Tukey, An algorithm for the machine calculation of complex Fourier series, 1965

Outline

Spectral Convolution

- Spectral graph theory
  - Book of Fan Chung[1] (harmonic analysis, graph theory, combinatorial problems, optimization)

- How to perform spectral convolution ?
  - Graph Laplacian
  - Fourier functions
  - Fourier transform
  - Convolution theorem



[1] FRK Chung, Spectral graph theory, 1997

Graph Laplacian

- Core operator in Spectral Graph Theory

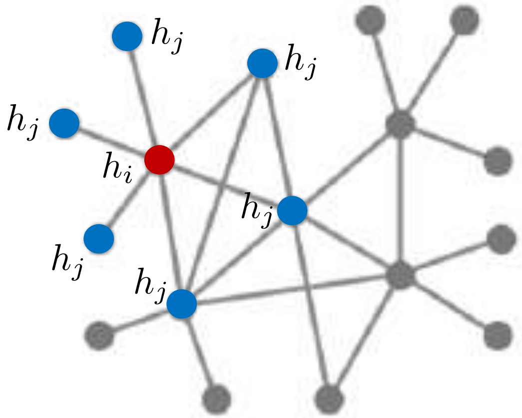
$$\mathcal{G} = (V, E, \underset{n \times n}{A}) \quad \rightarrow \quad \underset{n \times n}{\Delta} = \mathrm{I} - D^{-1/2} A D^{-1/2}$$

Normalized Laplacian

$$\text{where } \underset{n \times n}{D} = \mathrm{diag}\Big(\sum_{j \neq i} A_{ij}\Big)$$

- Interpretation :
  - Measure of smoothness : Difference between local value h_i and its neighborhood average values h_j.

$$(\Delta h)_i = h_i - \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_i d_j}} A_{ij} h_j$$



Fourier Functions

- Eigen-decomposition of graph Laplacian :

Lap Eigenvalues/ Spectrum

Lap Eigenvectors/ Fourier functions

$$\underset{n\times n}{\Delta} = \Phi^T \Lambda \Phi$$

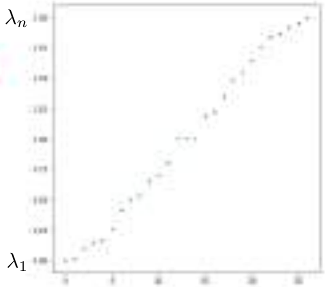
Fourier functions form an orthonormal basis

$$\text{where } \underset{n\times n}{\Phi} = [\phi_1, ..., \phi_n] = \begin{bmatrix} | & & | \\ \phi_1 & ... & \phi_n \\ | & & | \end{bmatrix} \text{ and } \Phi^T\Phi = \mathrm{I},\ \langle \phi_k, \phi_{k'} \rangle = \delta_{kk'}$$

$$\text{where } \underset{n\times n}{\Lambda} = \mathrm{diag}(\lambda_1, ..., \lambda_n) = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix} \text{ and } 0 \leq \lambda_1 \leq ... \leq \lambda_n = \lambda_{\max} \leq 2$$

$$\text{and } \Delta \underset{n\times 1}{\phi_k} = \lambda_k \phi_k, \quad k = 1, ..., n$$

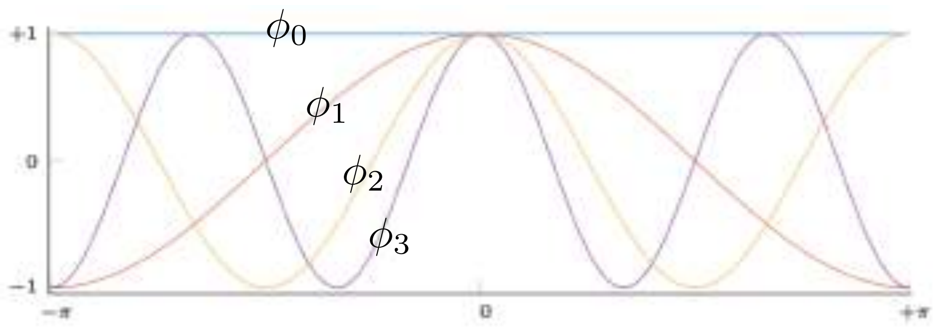
$n = 27$



Spectrum

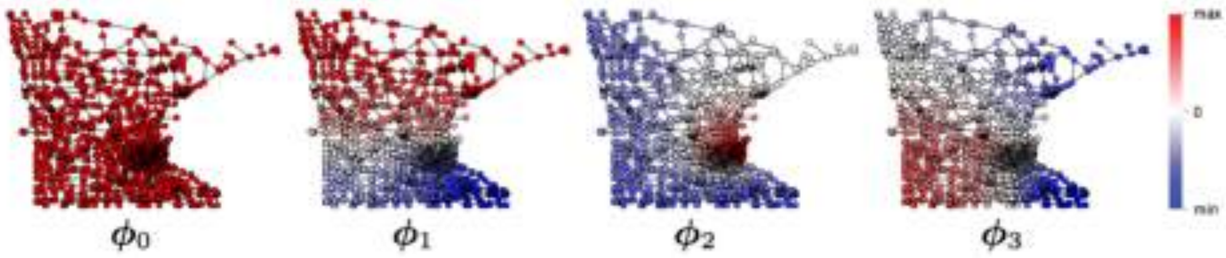
Fourier Functions

- Grid/Euclidean domain :



First eigenvectors of 1D Euclidean Laplacian = standard Fourier basis

- Graph domain :



First Laplacian eigenvectors of a graph

Fourier functions related to graph geometry

(s.a. communities, hubs, etc)

Spectral graph clustering[1]

[1] Von Luxburg, A tutorial on spectral clustering, 2007

Fourier Transform

- Fourier series : Decompose function h with Fourier functions[1] :

$$\underset{n\times 1}{h} = \sum_{k=1}^{n} \underbrace{\langle \phi_k, h\rangle}_{\substack{\mathcal{F}(h)_k=\hat{h}_k=\phi_k^T h \\ \text{scalar}}} \underset{n\times 1}{\phi_k}$$

$$= \sum_{k=1}^{n} \hat{h}_k \phi_k$$

$$= \underbrace{\Phi\hat{h}}_{\mathcal{F}^{-1}(\hat{h})}$$

Fourier transforms are one line of code (linear operations)

$$\underset{n\times 1}{\mathcal{F}(h)} = \Phi^T h = \hat{h}$$

Fourier Transform/ coefficients of Fourier Series

$$\underset{n\times 1}{\mathcal{F}^{-1}(\hat{h})} = \Phi\hat{h}$$

Inverse Fourier Transform

$$= \Phi\Phi^T h = h \text{ as } \mathcal{F}^{-1}\circ\mathcal{F} = \Phi\Phi^T = \mathrm{I}$$

Orthonormal basis/ Invertible transformation

[1] K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, 2011

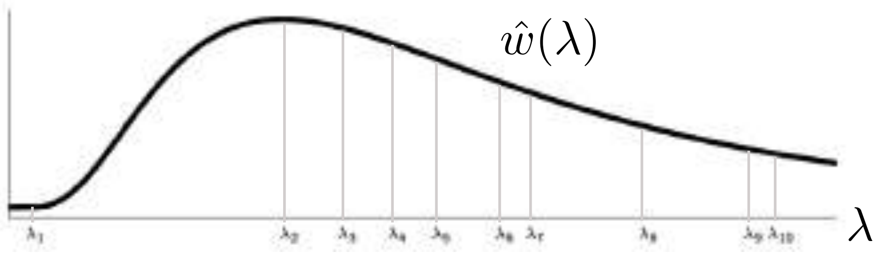
Convolution Theorem

- Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms :

$$
\begin{aligned}
\underset{n\times 1}{w * h} &= \underbrace{\mathcal{F}^{-1}}_{\Phi}\big(\, \underbrace{\mathcal{F}(w)}_{\Phi^T w = \hat{w}} \odot \underbrace{\mathcal{F}(h)}_{\Phi^T h} \,\big) \\
&= \underset{n\times n}{\Phi}\,\big(\, \underset{n\times 1}{\hat{w}} \odot \underset{n\times 1}{\Phi^T h} \,\big) \\
&= \Phi\,\big(\, \underset{n\times n}{\hat{w}(\Lambda)}\; \underset{n\times 1}{\Phi^T h} \,\big) \\
&= \Phi\; \hat{w}(\Lambda)\; \Phi^T h \\
&= \hat{w}(\underbrace{\Phi\Lambda\Phi^T}_{\Delta})\; h \\
&= \underset{n\times n}{\hat{w}(\Delta)}\underset{n\times 1}{h}
\end{aligned}
$$

$$
\underset{n\times 1}{\hat{w}} = \begin{bmatrix} \hat{w}(\lambda_1) \\ \vdots \\ \hat{w}(\lambda_n) \end{bmatrix}
$$

$$
\underset{n\times n}{\hat{w}(\Lambda)} = \mathrm{diag}(\hat{w}) = \begin{bmatrix} \hat{w}(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & \hat{w}(\lambda_n) \end{bmatrix}
$$



Expensive computation $O(n^2)$

No FFT

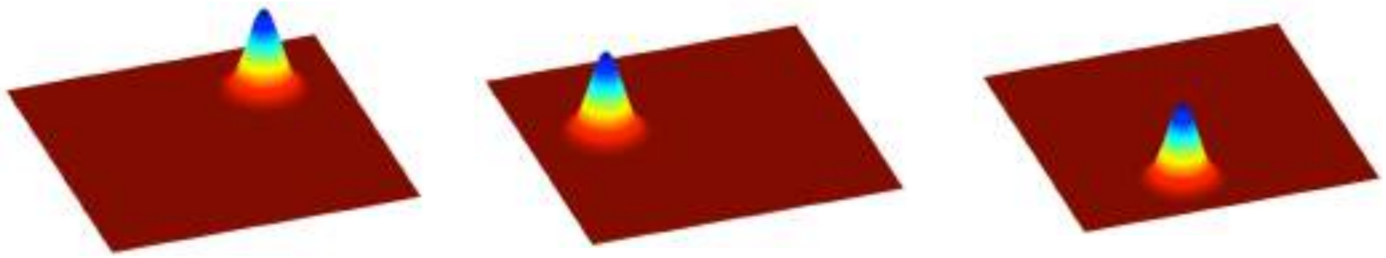
Spectral function/filter

No Shift Invariance for Graphs

- A signal h on graph can be translated to vertex i as follows :

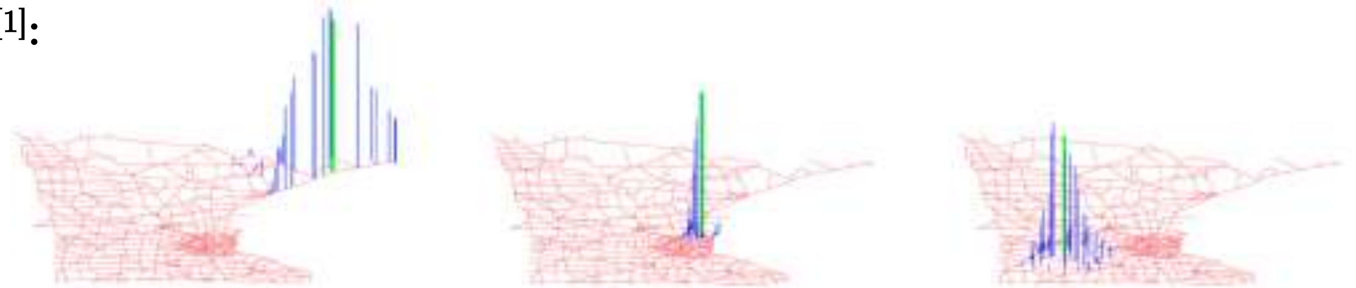
$$\mathrm{T}_i h = \delta_i * h$$

- Grid/Euclidean domain :



$\mathrm{T}_i h$ $\mathrm{T}_{i'} h$ $\mathrm{T}_{i''} h$

- Graph domain[1]:



$\mathrm{T}_i h$ $\mathrm{T}_{i'} h$ $\mathrm{T}_{i''} h$

[1] K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, 2011

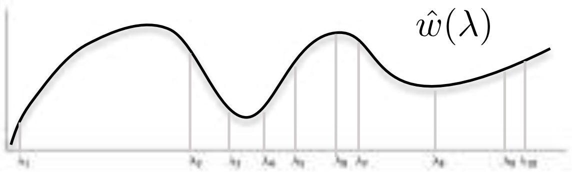
Outline

- Part 1: Traditional ConvNets
  - Architecture
  - Graph Domain
  - Convolution
- Part 2: Spectral Graph ConvNets
  - Spectral Convolution
  - Spectral GCNs
- Part 3: Spatial Graph ConvNets
  - Template Matching
  - Isotropic GCNs
  - Anisotropic GCNs
  - Lab on GatedGCNs
- Part 4: GNN Tasks
- Conclusion

Vanilla Spectral GCN[1]

- Graph spectral convolutional layer :

$$\underset{n\times d}{h^{\ell+1}} = \eta(w^{\ell} * h^{\ell}) = \eta(\hat{w}^{\ell}(\Delta)h^{\ell}) = \eta(\underset{n\times n}{\Phi}\underset{n\times n}{\hat{w}^{\ell}(\Lambda)}\Phi^{T}\underset{n\times d}{h^{\ell}})$$



$$\hat{w}(\Lambda) = \begin{bmatrix} \hat{w}(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & \hat{w}(\lambda_n) \end{bmatrix}$$

- First spectral technique for ConvNets
- Limitations :
  - No guarantee of spatial localization of filters
  - O(n) parameters to learn per layer
  - O(n^2) learning complexity (Fourier transform with full matrix ϕ)

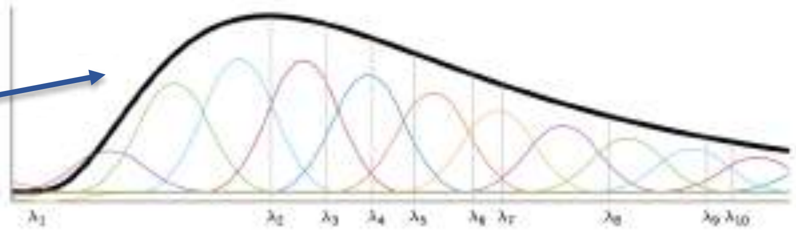
[1] Bruna, Zaremba, Szlam, LeCun, Spectral Networks and Locally Connected Networks on Graphs, 2014

SplineGCNs[1,2]

- Graph spectral convolutional layer :

Smooth spectral filters

$$\underset{n\times d}{h^{\ell+1}} = \eta(\, w^\ell * h^\ell \,)$$
$$= \eta(\, \Phi \hat{w}^\ell(\Lambda)\Phi^T h^\ell \,)$$
$$\underset{n\times n}{\hat{w}^\ell(\Lambda)} = \mathrm{diag}(\underbrace{\underset{n\times K}{B}\,\underset{K\times 1}{w^\ell}}_{n\times n})$$



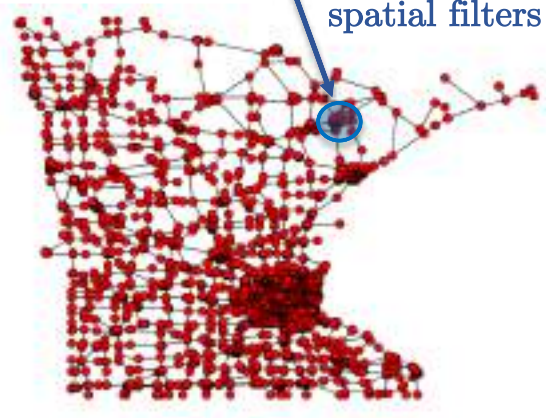
The K coefficients w^l are learned by backpropagation

Smooth spectral filters/ Linear combination of K smooth kernels B (splines)

Parseval's Identity
Localization in space ⇔ Smoothness in frequency domain
(Heisenberg's uncertainty principle)

$$\int |x|^{2k}|w(x)|^2dx = \int |\frac{\partial^k \hat{w}(\lambda)}{\partial \lambda^k}|^2 d\lambda$$

Localized spatial filters



- Localized filters in space (fast-decaying)
- O(*1*) parameters to learn per layer
- O(n^2) learning complexity (Fourier transform with full matrix ϕ)

[1] Bruna, Zaremba, Szlam, LeCun, Spectral Networks and Locally Connected Networks on Graphs, 2014
[2] Henaff, Bruna, LeCun, Deep Convolutional Networks on Graph-Structured Data, 2015

LapGCNs[1]

- How to learn in linear time $O(n)$ (w.r.t. graph size n) ?
  - $O(n^2)$ complexity comes from the direct use of Laplacian eigenvectors :

$$O(\underset{n\times d}{w * h}) = O(\underset{n\times n}{\Phi}\,\underset{n\times n}{\hat{w}^{\ell}(\Lambda)}\,\underset{n\times d}{\Phi^T h}) = O(n^2)$$

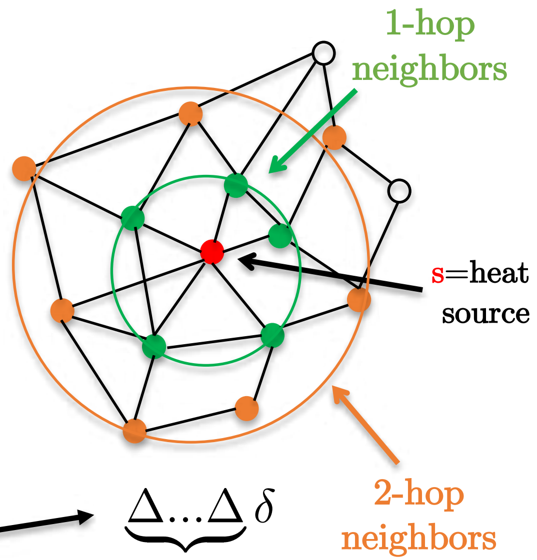
Full matrix

  - How to avoid the eigen-decomposition ?
    - Learn directly functions of the Laplacian !

$$w * h = \hat{w}(\Delta) h$$

$$\underset{n\times n}{\hat{w}(\Delta)} = \sum_{k=0}^{K-1} w_k \underset{n\times n}{\Delta^k}$$

Coefficients w_k are learned by backpropagation.



$$\underbrace{\Delta ... \Delta}_{\Delta^k} \delta$$

Filters are exactly localized in k-hop supports.
(each Laplacian operation increases the support of a function by 1 hop)

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

LapGCNs

- Learning complexity :

$$
\begin{aligned}
w * h &= \hat{w}(\Delta)h \\
&= \sum_{k=0}^{K-1} w_k \Delta^k h \\
&= \sum_{k=0}^{K-1} w_k X_k, \;\; \text{with } X_k = \underset{n \times n}{\Delta} \underset{n \times d}{X_{k-1}} \text{ and } X_0 = \underset{n \times d}{h}
\end{aligned}
$$

Recursive equation

- Sequence $\{X_k\}$ is generated by multiplying a matrix Δ and a vector $X_{k\text{-}1}$ ⇒ Complexity is O(E.K)=O(n) for sparse (real-world) graphs.
- No eigen-decomposition of Laplacian (ϕ,Λ) was required.
  - The name spectral GCNs can be misguided as the Lap eigen-decomposition is not used (computations are done in the spatial domain, not the spectral domain).
- Graph convolutional layers are (sparse) linear operations, thus GPU friendly (but not yet optimized).

LapGCNs

- Implementation :

$$\underset{n\times d}{h^{\ell+1}} = \eta\Big(\sum_{k=0}^{K-1} w_k^\ell \underset{n\times n}{\Delta^k} \underset{n\times d}{h^\ell} \Big) = \eta\Big(\sum_{k=0}^{K-1} w_k^\ell \underset{n\times d}{X_k} \Big) = \eta\Big(\underbrace{(w^\ell)^T \underset{1\times nd}{\bar{X}}}_{\substack{\textbf{reshape} \\ n\times d}} \Big)$$

$$\text{with } \underset{K\times nd}{\bar{X}} = \begin{bmatrix} - & \bar{X}_0 & - \\ & \vdots & \\ - & \bar{X}_{K-1} & - \end{bmatrix}$$

$$\underset{1\times nd}{\bar{X}_k} = \text{reshape}(\underset{n\times d}{X_k})$$

$$\underset{K\times 1}{w^\ell} = \begin{bmatrix} w_0^\ell \\ \vdots \\ w_{K-1}^\ell \end{bmatrix}$$

- Filters are exactly localized in K-hop support
- O(*1*) parameters to learn per layer
- O(n) learning complexity
- Monomials basis are unstable under coefficients perturbation (hard to optimize)

$$1, x, x^2, x^3, \ldots \quad \rightarrow \quad \Delta^0, \Delta^1, \Delta^2, \Delta^3, \ldots$$

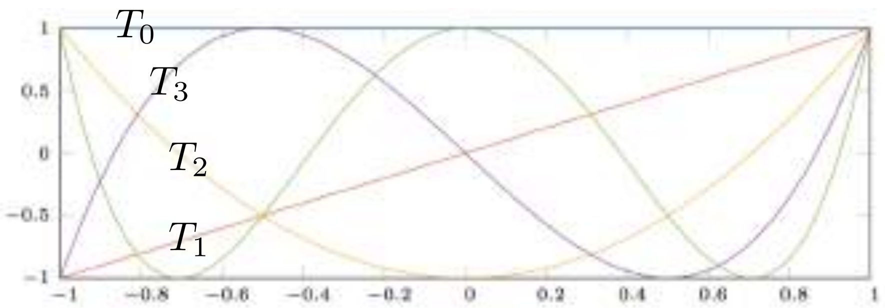
Chebyshev Polynomials

- Graph spectral convolution with Chebyshev polynomials :

$$
\begin{aligned}
w * h &= \hat{w}(\Delta)h \\
&= \sum_{k=0}^{K-1} w_k T_k(\Delta)h \\
&= \sum_{k=0}^{K-1} w_k X_k, \ \text{ with } X_k = 2\tilde{\Delta}X_{k-1} - X_{k-2}, X_0 = h, X_1 = \tilde{\Delta}h \text{ and } \tilde{\Delta} = 2\lambda_n^{-1}\Delta - \mathrm{I}
\end{aligned}
$$

Recursive equation

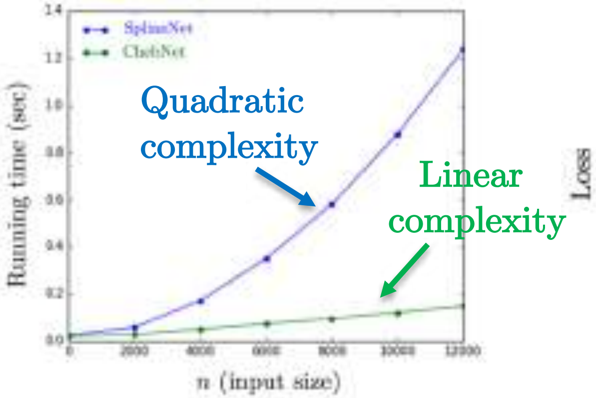
- Filters are exactly localized in K-hop support
- O(1) parameters to learn per layer
- O(n) learning complexity
- Stable under coefficients perturbation



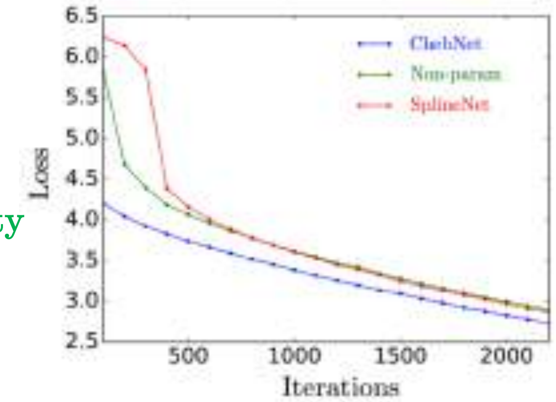
Orthonormal basis

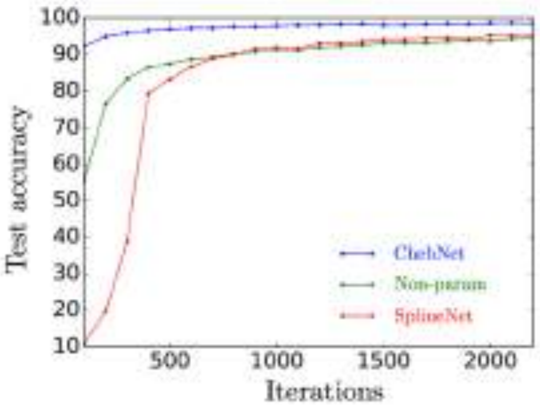
MNIST Numerical Experiment[1]

Running time



Optimization





Accuracy

Model	Order	Accuracy
LeNet5	-	99.33%
SplineNet	25	97.75%
ChebNet	25	99.14%

- ChebNets :
  - ConvNets for arbitrary graph domains
  - Same $O(n)$ learning complexity (but larger complexity constant)
  - Limitation : Isotropic model

- Isotropy vs anisotropy
  - Standard ConvNets produce anisotropic filters because Euclidean grids have directional structures (up, down, left, right).

  - Spectral ConvNets like ChebNets compute isotropic filters because there is no notion of directions on arbitrary graphs.

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

ChebNets for Multiple Graphs

- Multi-graph spectral convolution[1] :

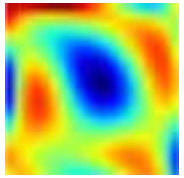
$$\underset{n_1 \times n_2 \times d}{h^{\ell+1}} = \eta(\ \hat{w}(\Delta_1, \Delta_2) * h^{\ell}\ )$$

$$h^{\ell+1} = \eta\Big(\sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \underset{K_1 \times K_2}{w_{k_1,k_2}} \underset{n_1 \times n_1}{T_{k_1}(\Delta_1)} \underset{n_2 \times n_2}{T_{k_2}(\Delta_2)} h^{\ell}\ \Big)$$

$$\underset{n_1 \times n_2}{\hat{w}(\lambda_1, \lambda_2)} = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} w_{k_1,k_2} T_{k_1}(\Lambda_1) T_{k_2}(\Lambda_2)$$

The $K_1.K_2$ coefficients $w_{k1,k2}$ are learned by backpropagation.

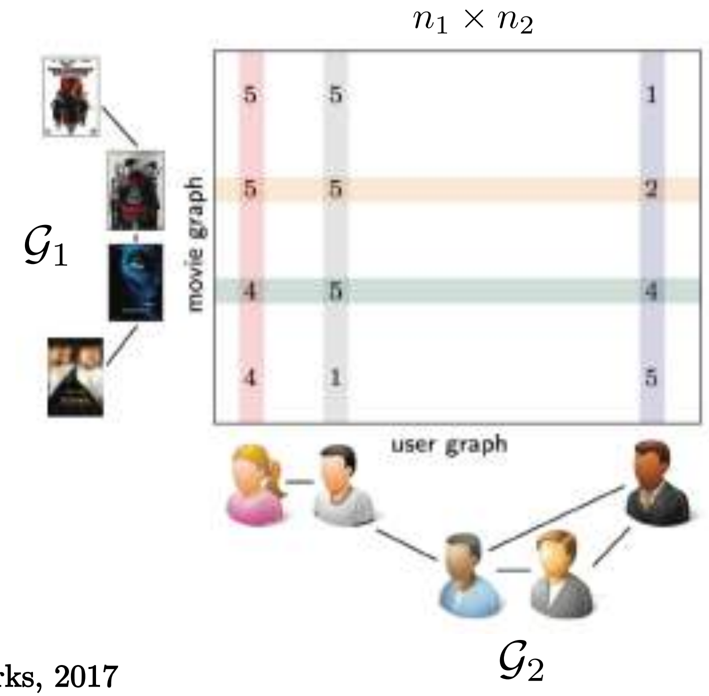
2D Spectral filter



$\hat{w}(\lambda_1, \lambda_2)$

- Recommendation task (Netflix)
  - Matrix Completion with Multiple Graphs[2] :

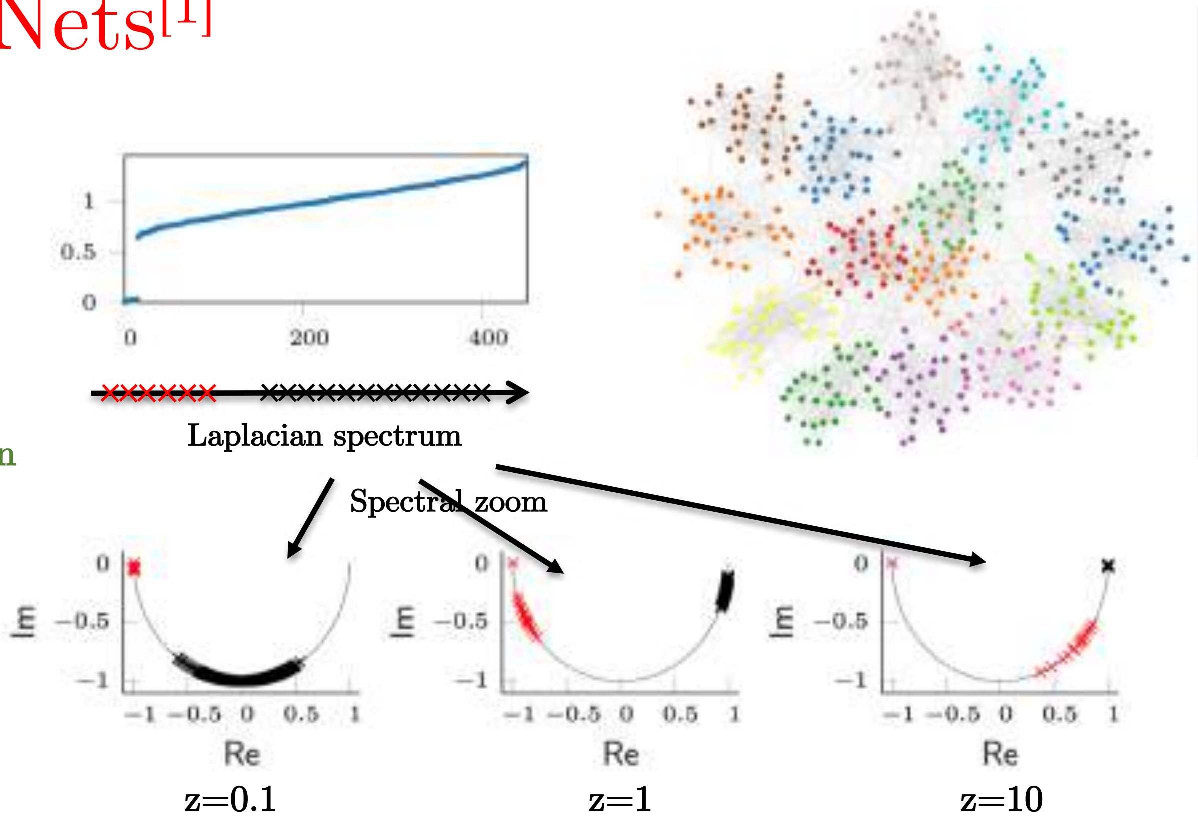
$$\min_{h \in \mathbb{R}^{n_1 \times n_2}} \quad \|h\|_* + \mu \|\Omega \circ (h - r)\|_F^2$$
$$+ \mu_1 \underbrace{\mathrm{tr}(h \Delta_1 h^\top)}_{\|h\|^2_{\mathcal{G}_1}} + \mu_2 \underbrace{\mathrm{tr}(h^\top \Delta_2 h)}_{\|h\|^2_{\mathcal{G}_2}}$$



[1] F Monti, M Bronstein, X Bresson, Geometric matrix completion with recurrent multi-graph neural networks, 2017
[2] V Kalofolias, X Bresson, M Bronstein, P Vandergheynst, Matrix completion on graphs, 2014

CayleyNets[1]

- ChebGCNs are unstable to produce filters with frequency bands of interest (graph communities).
- Cayley rationals can :



Spectral zoom z focuses on frequency bands.

$$\hat{w}(\Delta) = w_0 + 2\mathrm{Re}\{\sum_{k=0}^{K-1} w_k \frac{(z\Delta - i)^k}{(z\Delta + i)^k}\}$$

The K coefficients w_k are learned by backpropagation.

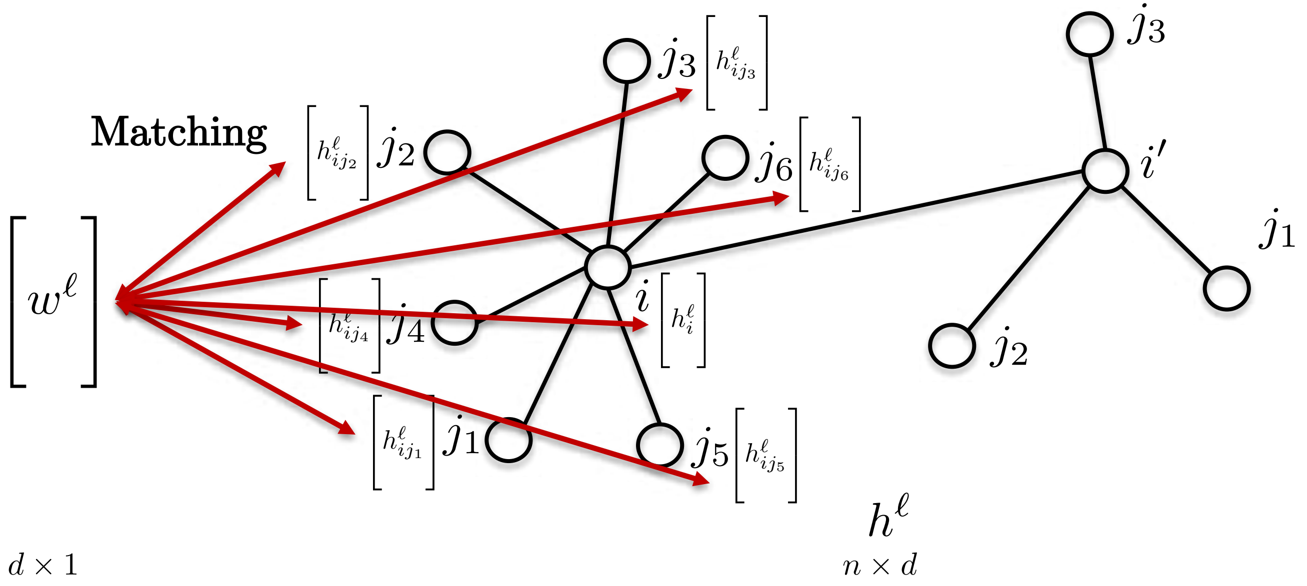
- CayleyNets
  - Same properties as ChebNets
  - Localized in frequency (with spectral zoom)
  - Richer class of filters for the same order K
  - Isotropic model

[1] R Levie, F Monti, X Bresson, MM Bronstein, CayleyNets: Graph convolutional neural networks with complex rational spectral filters, 2018

Outline

Template Matching

- How to define template matching for graphs ?
  - Main issue is the absence of node ordering/positioning.
    - Node indices are arbitrary and do not match the same information.
  - How to design template matching invariant to node re-parametrization ?
    - Simply use the same template features for all neighbors !



$$\underset{\text{scalar}}{h_i^{\ell+1}} = \eta\Big(\sum_{j\in\mathcal{N}_i} \underbrace{\langle \overset{d\times 1}{w^\ell}, \overset{d\times 1}{h_{ij}^\ell} \rangle}_{(h_{ij}^\ell)^T w^\ell} \Big)$$

One feature

$$\underset{d\times 1}{h_i^{\ell+1}} = \eta\Big(\sum_{j\in\mathcal{N}_i} \underset{d\times d}{W^\ell} \underset{d\times 1}{h_{ij}^\ell} \Big)$$

d features

$$\underset{n\times d}{h^{\ell+1}} = \eta\big(\underset{n\times n}{A} \underset{n\times d}{h^\ell} \underset{d\times d}{W^\ell} \big)$$

Vectorial representation

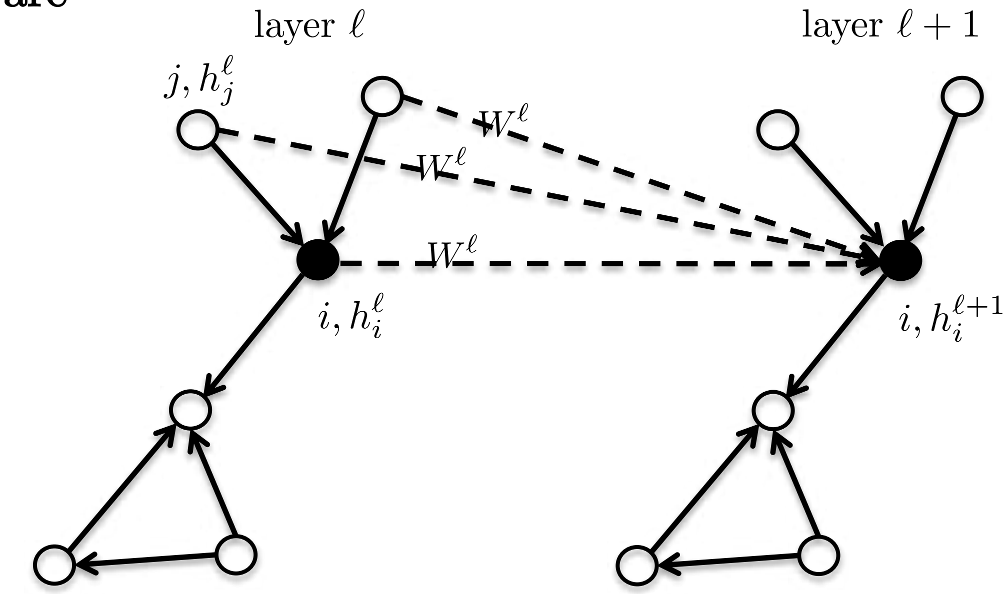
Outline

Vanilla GCNs[1,2,3]

- Simplest formulation of spatial GCNs
  - Handle the absence of node ordering
    - Invariant by node re-parametrization
  - Deal with different neighborhood sizes
  - Local reception field by design (only neighbors are considered)
  - Weight sharing (convolution property)
  - Independent of graph size
  - Limited to isotropic capability

$$\underset{n\times d}{h^{\ell+1}} = \eta\Big(\overset{n\times n}{D^{-1}} \underset{n\times n}{A} \overset{n\times d}{h^{\ell}} \underset{d\times d}{W^{\ell}} \Big)$$ **Matrix representation**

$$\underset{d\times 1}{h_i^{\ell+1}} = \eta\Big(\underbrace{\frac{1}{d_i} \sum_{j\in\mathcal{N}_i}}_{\text{Mean}} \underset{\text{scalar}}{A_{ij}} \underset{d\times d}{W^{\ell}} \underset{d\times 1}{h_j^{\ell}} \Big)$$ **Vectorial representation**



$$h_i^{\ell+1} = f_{\text{GCN}}(h_i^{\ell}, \{h_j^{\ell} : j \to i\})$$

[1] Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini, The Graph Neural Network Model, 2009
[2] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016
[3] S Sukhbaatar, A Szlam, R Fergus, Learning multiagent communication with backpropagation, 2016

ChebNets[1] and Vanilla GCNs[2,3]

- Vanilla GCNs is a simplification of ChebNets.
  - Truncated expansion of Chebyshev spectral convolution :

$$\begin{aligned} h^{\ell+1} &= \eta(\ w^\ell * h^\ell\ ) \\ &= \eta(\ \hat{w}^\ell(\Delta) h^\ell\ ) \\ &= \eta(\ \sum_{k=0}^{K-1} w_k^\ell T_k(\Delta) h^\ell\ ) \end{aligned}$$

Suppose $K=2,\ w_0^\ell = w^\ell,\ w_1^\ell = -w^\ell,\ \lambda_n = 2,$

$$\begin{aligned} h^{\ell+1} &= \eta(\ w^\ell (T_0(\Delta) - T_1(\Delta)) h^\ell\ ) \\ &= \eta(\ w^\ell (\mathrm{I} + D^{-1/2} A D^{-1/2}) h^\ell\ ) \end{aligned}$$

Operator with largest eigenvalue in [0,2] may cause divergence.

with

$$\begin{cases} T_0 = I \\ T_1 = \tilde{\Delta} = \frac{2}{\lambda_n}\Delta - \mathrm{I} \overset{\lambda_n=2}{=} \Delta - I \\ \Delta = \mathrm{I} - D^{-1/2} A D^{-1/2} \end{cases}$$

Add self-loop to graphs $\hat{A} = A + \mathrm{I}$

$$\begin{aligned} h^{\ell+1} &= \eta(\ w^\ell \hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} h^\ell\ ) \quad \text{(single feature)} \\ &= \eta(\ \hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} h^\ell W^\ell\ ) \quad (d \text{ features}) \end{aligned}$$

$$h_i^{\ell+1} = \eta(\ \frac{1}{\hat{d}_i^{-1/2}} \sum_{j \in \mathcal{N}_i} \frac{1}{\hat{d}_j^{-1/2}} \hat{A}_{ij} W^\ell h_j^\ell\ )$$

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016
[2] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016
[3] S Sukhbaatar, A Szlam, R Fergus, Learning multiagent communication with backpropagation, 2016

GraphSage[1]

- Vanilla GCNs (supposing $A_{ij}=1$) : $h_i^{\ell+1} = \eta\big(\frac{1}{d_i} \sum_{j\in\mathcal{N}_i} W^\ell h_j^\ell \big)$
- GraphSage :
  - Differentiate template weights W^l between neighbors h_j and central node h_i.
  - Isotropic GCNs

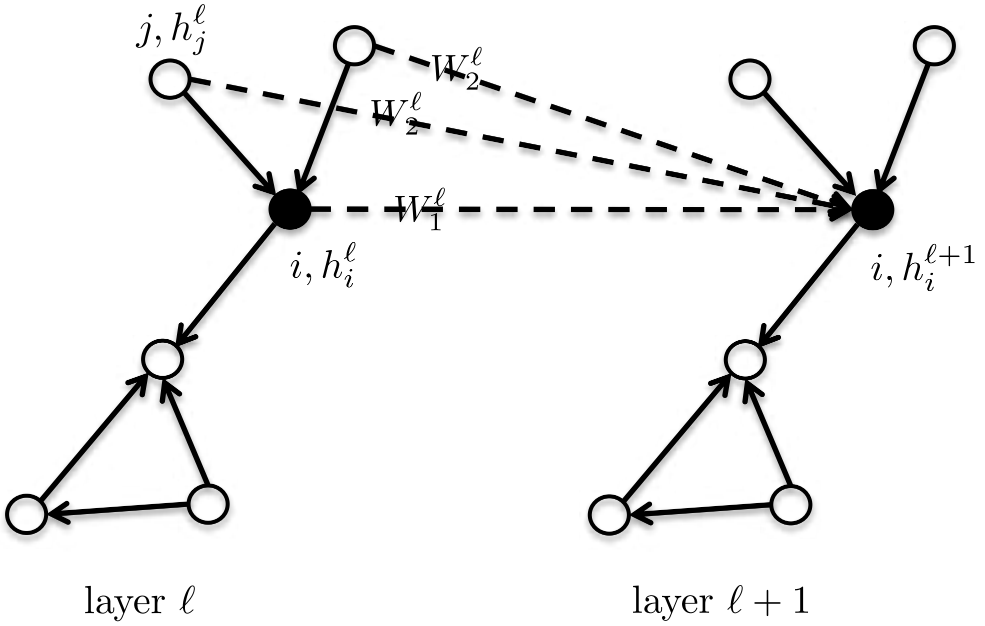
$$\underset{d\times 1}{h_i^{\ell+1}} = \eta\Big(\underset{d\times d}{W_1^\ell}\underset{d\times 1}{h_i^\ell} + \underbrace{\frac{1}{d_i}\sum_{j\in\mathcal{N}_i} \underset{d\times d}{W_2^\ell}\underset{d\times 1}{h_j^\ell}} \Big)$$

$\mathrm{Mean}_{j\in\mathcal{N}_i}\ W_2^\ell h_j^\ell$

Or alternatively,

$\mathrm{Max}_{j\in\mathcal{N}_i}\ W_2^\ell h_j^\ell$

$\mathrm{LSTM}^\ell(h_j^\ell)$



[1] W Hamilton, Z Ying, J Leskovec, Inductive representation learning on large graphs, 2017

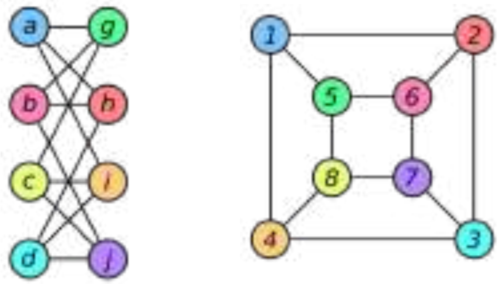
Graph Isomorphism Networks[1] (GIN)

- Architecture that can differentiate graphs that are not isomorphic.
  - Graph isomorphism is an equivalent relation for similar graph structures.

- Isotropic GCNs

$$\underset{d\times 1}{h_i^{\ell+1}} = \mathrm{ReLU}\Big(\underset{d\times d}{W_2^\ell}\ \mathrm{ReLU}\big(\overset{\text{Batch normalization}}{\mathrm{BN}}\big(\underset{d\times d}{W_1^\ell}\underset{d\times 1}{\hat{h}_i^{\ell+1}}\big)\big)\Big)$$

$$\underset{d\times 1}{\hat{h}_i^{\ell+1}} = (1+\epsilon)\, h_i^\ell + \sum_{j\in\mathcal{N}_i} h_j^\ell$$



Example of two isomorphic graphs

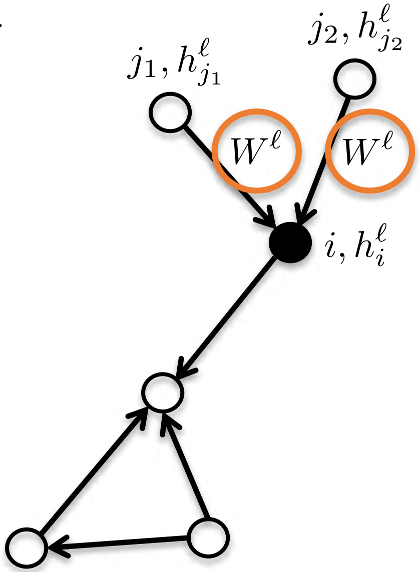
[1] K Xu, W Hu, J Leskovec, S Jegelka, How powerful are graph neural networks?, 2018

Outline

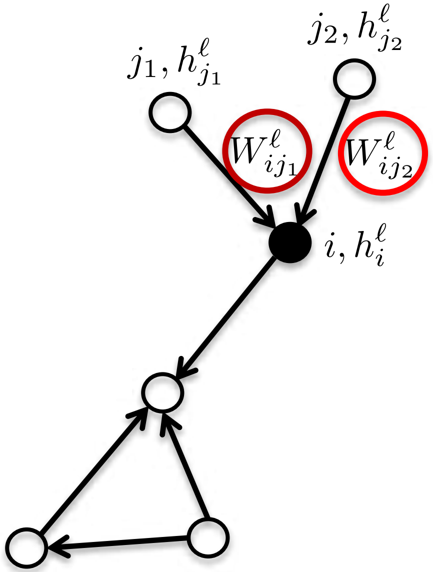
Anisotropic GCNs

- Reminder :
  - Standard ConvNets produce anisotropic filters because Euclidean grids have directional structures (up, down, left, right).
  - GCNs such as ChebNets, CayleyNets, Vanilla GCNs, GraphSage, GIN compute isotropic filters as there is no notion of directions on arbitrary graphs.

- How to get anisotropy back in GNNs ?
  - Natural edge features[1,2] if available (e.g. different bond connections between atoms).
  - We need an anisotropic mechanism that is independent of the node parametrization.
  - Edge degrees[3]/Edge gates[4]/Attention mechanism[5] : MoNets[3], GAT[5], GatedGCNs[4] can treat neighbors differently.



Isotropic weights



Anisotropic weights

[1] Gilmer, Schoenholz, Riley, Vinyals, Dahl, Neural message passing for quantum chemistry, 2017
[2] X Bresson, T Laurent, A Two-Step Graph Convolutional Decoder for Molecule Generation, 2019
[3] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model CNNs, 2016
[4] X Bresson, T Laurent, Residual gated graph convnets, 2017
[5] Velickovic, Cucurull, Casanova, Romero, Lio, Bengio, Graph Attention Networks, 2018

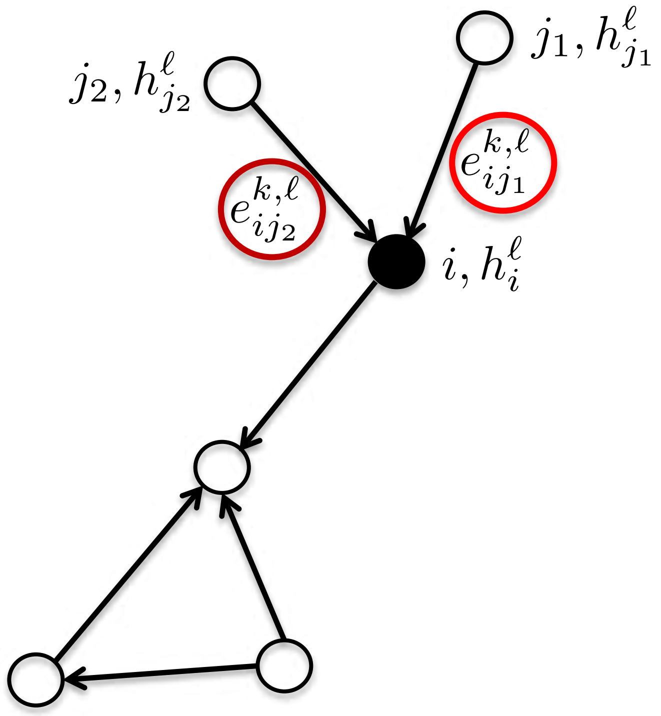
MoNets[1]

- MoNets[1] leverage the Bayesian Gaussian Mixture Model (GMM)[2].

$$\underset{d\times 1}{h_i^{\ell+1}} = \mathrm{ReLU}\Big(\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \underset{\text{scalar}}{e_{ij}^{k,\ell}}\ \underset{d\times d}{W_1^{k,\ell}}\ \underset{d\times 1}{h_j^{\ell}}\Big)$$

$$\underset{\text{scalar}}{e_{ij}^{k,\ell}} = \exp\Big(-\frac{1}{2}\underset{1\times 2}{(u_{ij}^{\ell}-\mu_k^{\ell})^T}\underset{2\times 2}{(\Sigma_k^{\ell})^{-1}}\underset{2\times 1}{(u_{ij}^{\ell}-\mu_k^{\ell})}\Big)$$

$$\underset{2\times 1}{u_{ij}^{\ell}} = \mathrm{Tanh}\Big(\underset{2\times 2}{A^{\ell}}\ \underset{2\times 1}{(\deg_i^{-1/2},\deg_j^{-1/2})} + \underset{2\times 1}{a^{\ell}}\Big)$$



[1] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model CNNs, 2016
[2] A. Dempster, NM Laird, D. Rubin, Maximum Likelihood from Incomplete Data Via the EM Algorithm, 1977

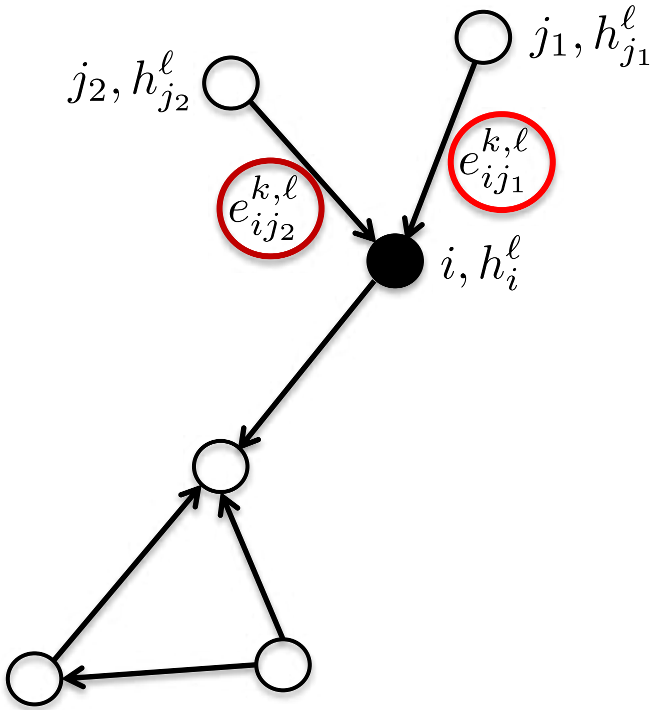
Graph Attention Networks[1] (GAT)

- GAT uses the attention mechanism[2] to introduce anisotropy in the neighborhood aggregation function.
- The network employs a multi-headed architecture to increase the learning capacity, similar to the Transformer[3].

$$\underset{d\times 1}{h_i^{\ell+1}} = \mathrm{Concat}_{k=1}^{K}\Big(\mathrm{ELU}\Big(\sum_{j\in\mathcal{N}_i} \underset{\text{scalar}}{e_{ij}^{k,\ell}}\ \underset{\frac{d}{K}\times d}{W_1^{k,\ell}}\ \underset{d\times 1}{h_j^{\ell}}\Big)\Big)$$

$$\underset{\text{scalar}}{e_{ij}^{k,\ell}} = \mathrm{Softmax}_{\mathcal{N}_i}(\hat{e}_{ij}^{k,\ell}) = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}$$

$$\hat{e}_{ij}^{k,\ell} = \mathrm{LeakyReLU}\Big(\underset{1\times\frac{2d}{K}}{W_2^{k,\ell}}\ \underbrace{\mathrm{Concat}\big(W_1^{k,\ell}h_i^{\ell},\ W_1^{k,\ell}h_j^{\ell}\big)}_{\frac{2d}{K}\times 1}\ \Big)$$



[1] Velickovic, Cucurull, Casanova, Romero, Lio, Bengio, Graph Attention Networks, 2018
[2] D Bahdanau, K Cho, Y Bengio, Neural machine translation by jointly learning to align and translate, 2014
[3] A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2017

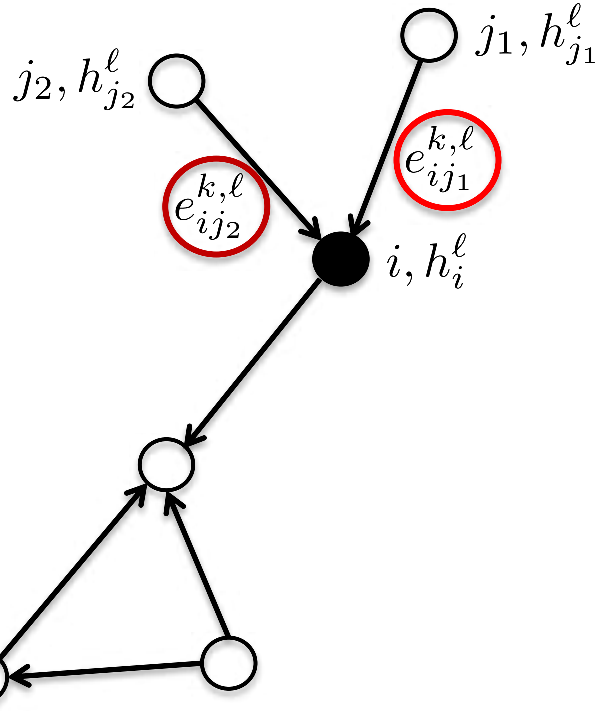
Gated Graph ConvNets[1]

- GatedGCNs use edge gates to design an anisotropic variant of GCNs.
  - Edge gates can be regarded as a soft attention process, related to the standard sparse attention mechanism[2].
  - Edge features are explicit (important for edge prediction tasks).
- Residual connections and batch normalization enhance learning speed and generalization.

$$\underset{d\times 1}{h_i^{\ell+1}} = h_i^{\ell} + \mathrm{ReLU}\Big(\mathrm{BN}\Big(\underset{d\times d}{W_1^{\ell}}\underset{d\times 1}{h_i^{\ell}} + \sum_{j\in\mathcal{N}_i} \underset{d\times 1}{e_{ij}^{\ell}} \odot \underset{d\times d}{W_2^{\ell}}\underset{d\times 1}{h_j^{\ell}}\Big)\Big)$$

$$\underset{d\times 1}{e_{ij}^{\ell}} = \frac{\sigma(\hat{e}_{ij}^{\ell})}{\sum_{j'\in\mathcal{N}_i}\sigma(\hat{e}_{ij'}^{\ell}) + \varepsilon}$$

$$\underset{d\times 1}{\hat{e}_{ij}^{\ell}} = \hat{e}_{ij}^{\ell-1} + \mathrm{ReLU}\Big(\mathrm{BN}\big(\underset{d\times d}{V_1^{\ell}} h_i^{\ell-1} + \underset{d\times d}{V_2^{\ell}} h_j^{\ell-1} + \underset{d\times d}{V_3^{\ell}} \hat{e}_{ij}^{\ell-1}\big)\Big)$$



[1] X Bresson, T Laurent, Residual gated graph convnets, 2017
[2] D Bahdanau, K Cho, Y Bengio, Neural machine translation by jointly learning to align and translate, 2014

Graph Transformers

- Graph version of Transformer[1] :

$$\underset{d\times 1}{h_i^{\ell+1}} = \underset{d\times d}{W^\ell}\ \text{Concat}_{k=1}^{K}\Big(\sum_{j\in\mathcal{N}_i} \overset{\text{scalar}}{e_{ij}^{k,\ell}} \underset{\frac{d}{K}\times d}{V^{k,\ell}}\ h_j^\ell \Big)$$

Value

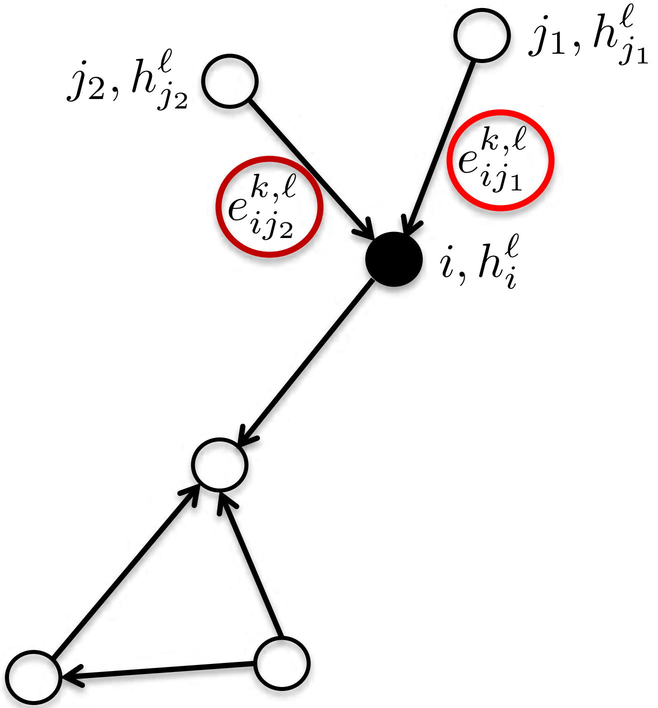
$$e_{ij}^{k,\ell} = \text{Softmax}_{\mathcal{N}_i}(\hat{e}_{ij}^{k,\ell}) = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}$$

$$\hat{e}_{ij}^{k,\ell} = (\underset{1\times d}{Q^{\ell,k}h_i^\ell})^T(\underset{d\times 1}{K^{\ell,k}h_j^\ell})$$

Query

Key

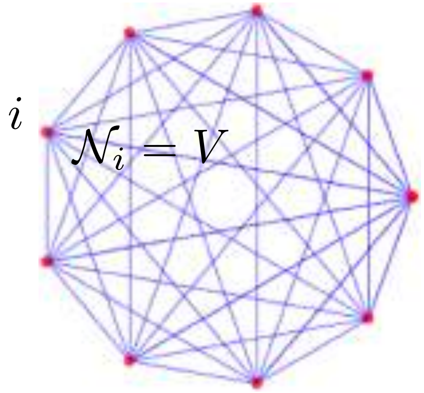
Attention mechanism in 1-hop neighborhood



[1] A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2017

Transformers[1]

- Transformers[1] is a special case of GCNs when the graph is fully connected.
- The neighborhood $\mathcal{N}_i$ is the whole graph.



Fully connected graph

$$h_i^{\ell+1} = W^\ell \ \mathrm{Concat}_{k=1}^{K}\Big(\sum_{j\in\mathcal{N}_i} e_{ij}^{k,\ell}\ V^{k,\ell}\ h_j^\ell \Big)$$

$$e_{ij}^{k,\ell} = \mathrm{Softmax}_{\mathcal{N}_i}(\hat{e}_{ij}^{k,\ell}) = \frac{\exp(\hat{e}_{ij}^{k,\ell})}{\sum_{j'\in\mathcal{N}_i}\exp(\hat{e}_{ij'}^{k,\ell})}$$

$$\hat{e}_{ij}^{k,\ell} = (Q^{\ell,k}h_i^\ell)^T(K^{\ell,k}h_j^\ell)$$

$$\mathcal{N}_i = V \quad \Rightarrow$$

$$\underset{n\times d}{h^{\ell+1}} = \mathrm{Concat}_{k=1}^{K}\Big(\underbrace{\mathrm{Softmax}(\underbrace{\underset{n\times\frac{d}{K}}{Q^\ell}\ \underset{\frac{d}{K}\times n}{{K^\ell}^T}}_{n\times n})\underset{n\times\frac{d}{K}}{V^\ell}}_{n\times\frac{d}{K}}\Big)\ \overset{d\times d}{W^\ell}$$

$$Q^\ell = h^\ell\ W_Q^\ell$$

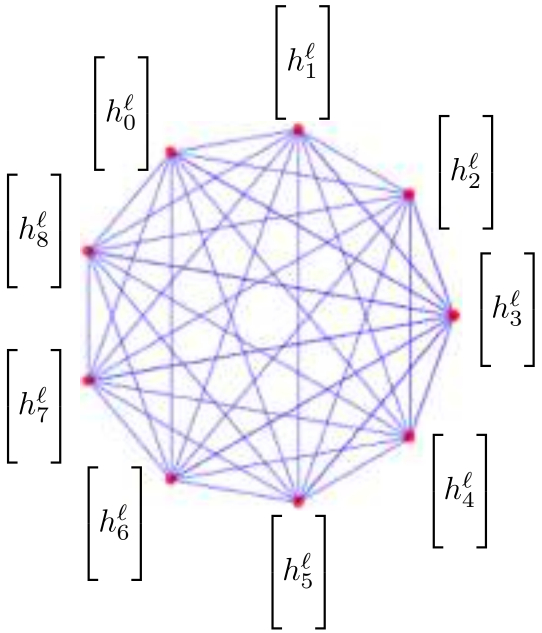
$$K^\ell = h^\ell\ W_K^\ell$$

$$\underset{n\times\frac{d}{K}}{V^\ell} = \underset{n\times d}{h^\ell}\ \underset{d\times\frac{d}{K}}{W_V^\ell}$$

[1] A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2017

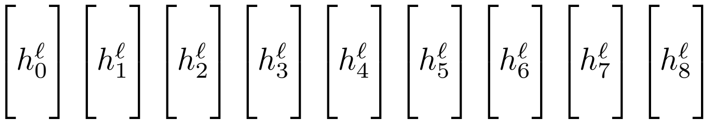
Transformers

- What does it mean to have a graph fully connected?
  - It becomes less useful to talk about graphs as each data point is connected to all other points. There is no particular graph structure that can be used.
  - It would be better to talk about sets rather than graphs in this case.
  - Transformers are Set Neural Networks.
    - They are today the best technique to analyze sets/bags of features.



Fully connected graph

$\Leftrightarrow$



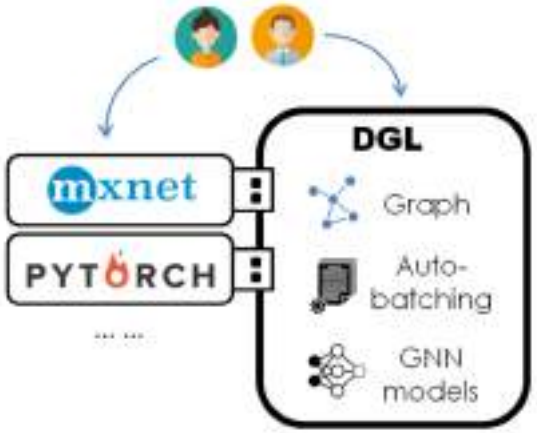
Sets of features

Outline

Lab on GatedGCNs

- GatedGCNs with DGL (Deep Graph Library), NYU-Shanghai, Prof. Zhang Zheng :
  - Website : https://www.dgl.ai
  - Documentation : https://docs.dgl.ai
  - Link to the lab (Google Collab, Gmail account required): https://drive.google.com/file/d/1WG5t6X12Z70JPtvA2-2PzdK3TMTQMsvm

DGL



Lab on GatedGCNs

DGL creates a batch of graphs

```python
# collate function
def collate(samples):
    graphs, labels = map(list, zip(*samples)) # samples is a list of pairs (graph, label).
    labels = torch.tensor(labels)
    tab_sizes_n = [ graphs[i].number_of_nodes() for i in range(len(graphs))] # graph sizes
    tab_snorm_n = [ torch.FloatTensor(size,1).fill_(1./float(size)) for size in tab_sizes_n ]
    snorm_n = torch.cat(tab_snorm_n).sqrt() # normalization constant for better optimization
    tab_sizes_e = [ graphs[i].number_of_edges() for i in range(len(graphs))] # nb of edges
    tab_snorm_e = [ torch.FloatTensor(size,1).fill_(1./float(size)) for size in tab_sizes_e ]
    snorm_e = torch.cat(tab_snorm_e).sqrt() # normalization constant for better optimization
    batched_graph = dgl.batch(graphs) # batch graphs
    return batched_graph, labels, snorm_n, snorm_e


# create artifical data feature (= in degree) for each node
def create_artificial_features(dataset):
    for (graph,_) in dataset:
        graph.ndata['feat'] = graph.in_degrees().view(-1, 1).float()
        graph.edata['feat'] = torch.ones(graph.number_of_edges(),1)
    return dataset

# use artifical graph dataset of DGL
trainset = MiniGCDataset(8, 10, 20)
trainset = create_artificial_features(trainset)
print(trainset[0])
```

```
(DGLGraph(num_nodes=13, num_edges=39,
         ndata_schemes={'feat': Scheme(shape=(1,), dtype=torch.float32)}
         edata_schemes={'feat': Scheme(shape=(1,), dtype=torch.float32)}), 0)
```

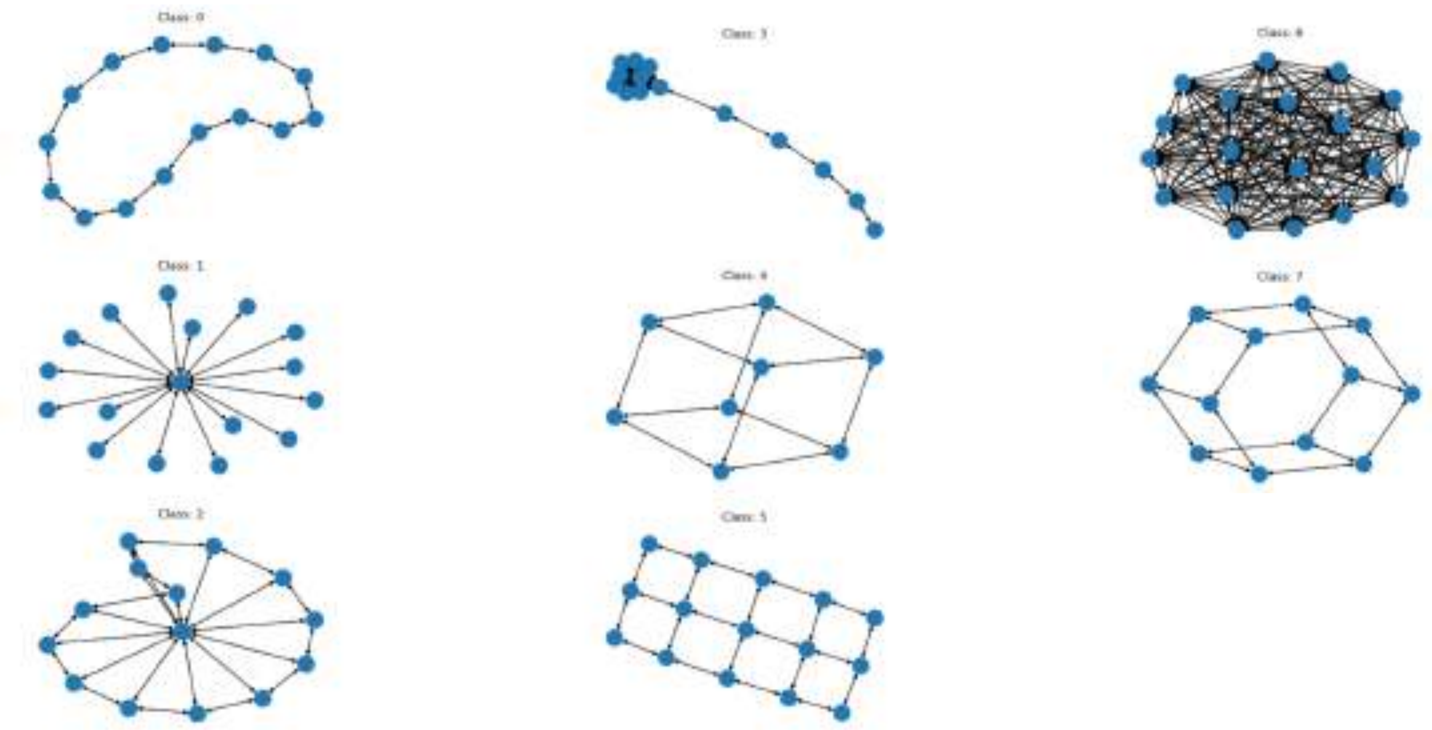
Create artificial node feature (input degree) and edge feature (value 1)

$$\frac{1}{\sqrt{V_k}}, \frac{1}{\sqrt{E_k}}$$

Graph normalization constants

Lab on GatedGCNs

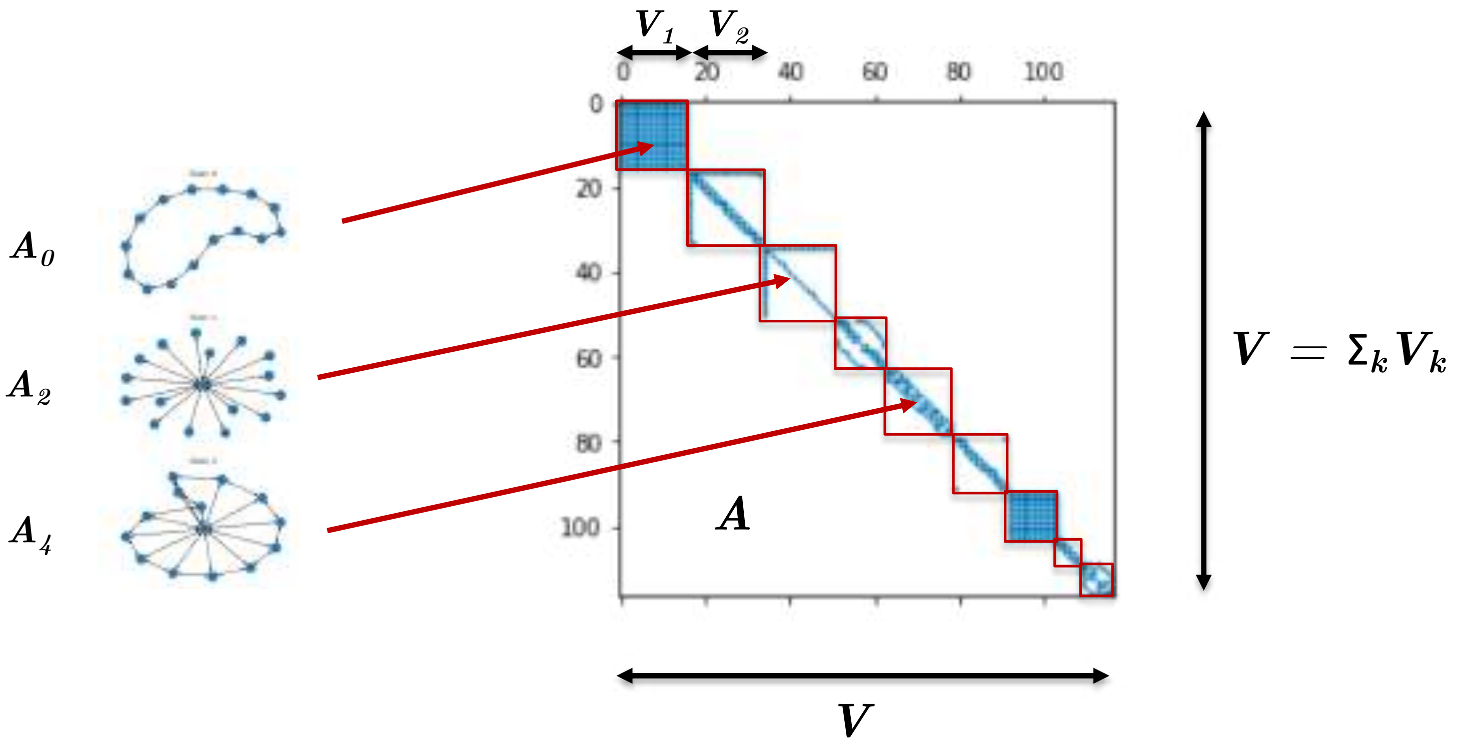
```python
# use artifical graph dataset of DGL
trainset = MiniGCDataset(8, 10, 20)
```



Lab on GatedGCNs

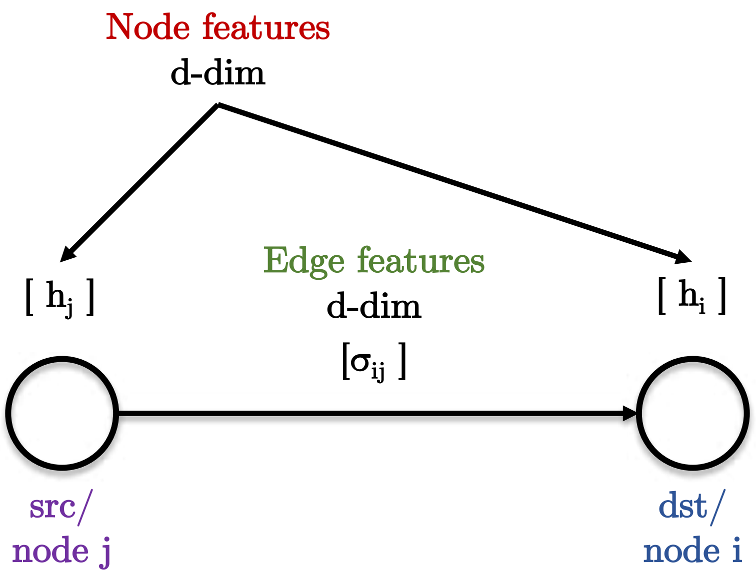
- Understanding DGL :
  - How to process K graphs of different sizes ?

    Form a (big) sparse block diagonal matrix $\boldsymbol{A}$ with K adjacency matrices $\boldsymbol{A}_k$.



Lab on GatedGCNs

- Basic structure of an edge in DGL :

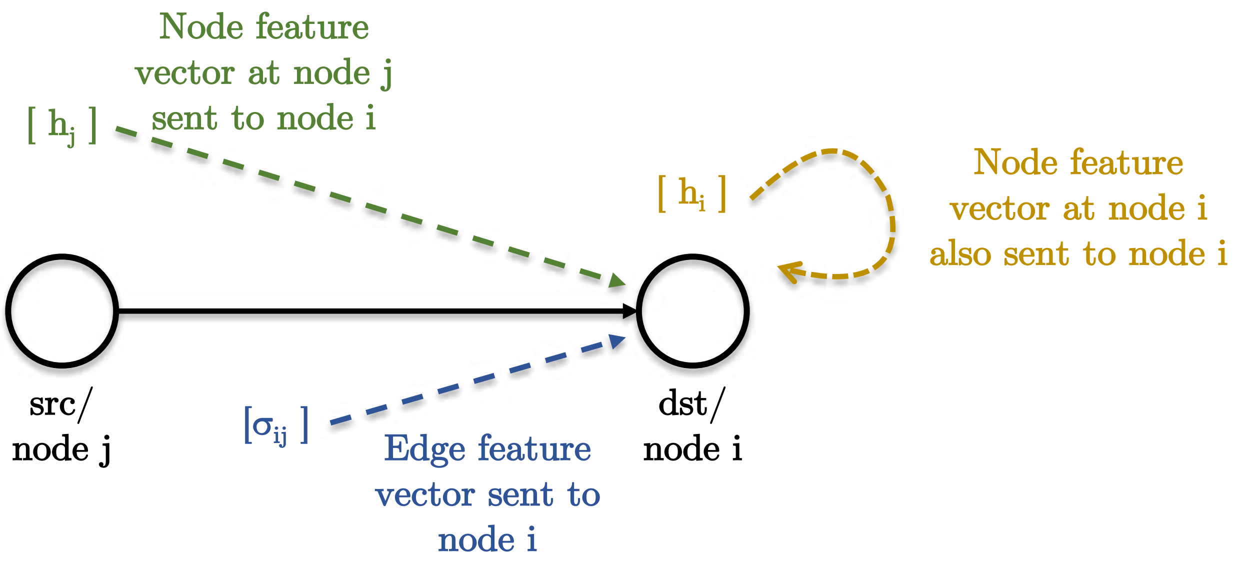


- Goal is to compute efficiently expressions of the form :

$$f_i = h_i + \Sigma_{j \to i}\, \sigma_{ij} \circ h_j$$

Lab on GatedGCNs

- Step 1 : Message passing function defined on edges
  - Node feature and edge feature are passed along all edges connecting a node.



h_j : edges.src['h'] . size() = E x d

h_i : edges.dst['h'] . size() = E x d

σ_{ij} : edges.data['σ'] . size() = E x d

#edges

All message passing operations can be done in parallel !

Lab on GatedGCNs

- Step 2 : Reduce function defined on nodes
  - Reduce functions of the form : $f_i = h_i + \Sigma_{j \to i}\ \sigma_{ij} \circ h_j$
  - Reduce function collects all messages passed in Step 1.
  - Code : $f = h_i +$ torch.sum $(h_j \times \sigma_{ij}$, dim=1) . size() = V x d

#nodes

Sum over neighbors

- GPU acceleration :
  - DGL batches the nodes with the same number of neighbors.

#nodes in batch$_1$

#neighbors

$$h_j = \text{nodes.mailbox['}h_j\text{']} = \begin{bmatrix} \text{batch}_1\text{.size()} = 11 \times 12 \times d \\ \vdots \\ \text{batch}_{34}\text{.size()} = 14 \times 9 \times d \end{bmatrix}$$

σ_{ij} = nodes.mailbox['σ_{ij}'] = same structure than h_j

h_i = nodes.data['h'] . size() = V x d

Lab on GatedGCNs

GatedGCN layer :

$$h_i^{\ell+1} = h_i^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(A^\ell h_i^\ell + \sum_{j\sim i} \eta(e_{ij}^\ell) \odot B^\ell h_j^\ell\Big)\Big)/\sqrt{V_k},$$

$$\eta(e_{ij}^\ell) = \frac{\sigma(e_{ij}^\ell)}{\sum_{j'\sim i} \sigma(e_{ij'}^\ell) + \varepsilon},$$

$$e_{ij}^{\ell+1} = e_{ij}^\ell + \mathrm{ReLU}\Big(\mathrm{BN}\Big(C^\ell e_{ij}^\ell + D^\ell h_i^{\ell+1} + E^\ell h_j^{\ell+1}\Big)\Big)/\sqrt{E_k}.$$

```python
class GatedGCN_layer(nn.Module):

    def __init__(self, input_dim, output_dim):
        super(GatedGCN_layer, self).__init__()
        self.A = nn.Linear(input_dim, output_dim, bias=True)
        self.B = nn.Linear(input_dim, output_dim, bias=True)
        self.C = nn.Linear(input_dim, output_dim, bias=True)
        self.D = nn.Linear(input_dim, output_dim, bias=True)
        self.E = nn.Linear(input_dim, output_dim, bias=True)
        self.bn_node_h = nn.BatchNorm1d(output_dim)
        self.bn_node_e = nn.BatchNorm1d(output_dim)

    def message_func(self, edges):
        Bh_j = edges.src['Bh']
        e_ij = edges.data['Ce'] +  edges.src['Dh'] + edges.dst['Eh'] # e_ij = Ce_ij + Dhi + Ehj
        edges.data['e'] = e_ij
        return {'Bh_j' : Bh_j, 'e_ij' : e_ij}

    def reduce_func(self, nodes):
        Ah_i = nodes.data['Ah']
        Bh_j = nodes.mailbox['Bh_j']
        e = nodes.mailbox['e_ij']
        sigma_ij = torch.sigmoid(e) # sigma_ij = sigmoid(e_ij)
        h = Ah_i + torch.sum( sigma_ij * Bh_j, dim=1 ) / torch.sum( sigma_ij, dim=1 )  # hi = Ahi + sum_j eta_ij * Bhj
        return {'h' : h}

    def forward(self, g, h, e, snorm_n, snorm_e):

        h_in = h # residual connection
        e_in = e # residual connection

        g.ndata['h']  = h
        g.ndata['Ah'] = self.A(h)
        g.ndata['Bh'] = self.B(h)
        g.ndata['Dh'] = self.D(h)
        g.ndata['Eh'] = self.E(h)
        g.edata['e']  = e
        g.edata['Ce'] = self.C(e)
        g.update_all(self.message_func,self.reduce_func)
        h = g.ndata['h'] # result of graph convolution
        e = g.edata['e'] # result of graph convolution

        h = h* snorm_n # normalize activation w.r.t. graph node size
        e = e* snorm_e # normalize activation w.r.t. graph edge size

        h = self.bn_node_h(h) # batch normalization
        e = self.bn_node_e(e) # batch normalization

        h = F.relu(h) # non-linear activation
        e = F.relu(e) # non-linear activation

        h = h_in + h # residual connection
        e = e_in + e # residual connection

        return h, e
```

Lab on GatedGCNs

MLP classifier layer

```
class MLP_layer(nn.Module):

    def __init__(self, input_dim, output_dim, L=2): # L = nb of hidden layers
        super(MLP_layer, self).__init__()
        list_FC_layers = [ nn.Linear( input_dim, input_dim, bias=True ) for l in range(L) ]
        list_FC_layers.append(nn.Linear( input_dim, output_dim , bias=True ))
        self.FC_layers = nn.ModuleList(list_FC_layers)
        self.L = L

    def forward(self, x):
        y = x
        for l in range(self.L):
            y = self.FC_layers[l](y)
            y = F.relu(y)
        y = self.FC_layers[self.L](y)
        return y
```

GatedGCN Network

Node input embedding

Edge input embedding

Run graphNN layers

Compute graph vectorial representation by a (simple) average of all node features with DGL.

Use MLP classifier

```
class GatedGCN_Net(nn.Module):

    def __init__(self, net_parameters):
        super(GatedGCN_Net, self).__init__()
        input_dim = net_parameters['input_dim']
        hidden_dim = net_parameters['hidden_dim']
        output_dim = net_parameters['output_dim']
        L = net_parameters['L']
        self.embedding_h = nn.Linear(input_dim, hidden_dim)
        self.embedding_e = nn.Linear(1, hidden_dim)
        self.GatedGCN_layers = nn.ModuleList([ GatedGCN_layer(hidden_dim, hidden_dim) for _ in range(L) ])
        self.MLP_layer = MLP_layer(hidden_dim, output_dim)

    def forward(self, g, h, e, snorm_n, snorm_e):

        # input embedding
        h = self.embedding_h(h)
        e = self.embedding_e(e)

        # graph convnet layers
        for GGCN_layer in self.GatedGCN_layers:
            h,e = GGCN_layer(g,h,e,snorm_n,snorm_e)

        # MLP classifier
        g.ndata['h'] = h
        y = dgl.mean_nodes(g,'h')
        y = self.MLP_layer(y)

        return y
```

Lab on GatedGCNs

```python
def train_one_epoch(net, data_loader):
    """
    train one epoch
    """
    net.train()
    epoch_loss = 0
    epoch_train_acc = 0
    nb_data = 0
    gpu_mem = 0
    for iter, (batch_graphs, batch_labels, batch_snorm_n, batch_snorm_e) in enumerate(data_loader):
        batch_x = batch_graphs.ndata['feat'].to(device)
        batch_e = batch_graphs.edata['feat'].to(device)
        batch_snorm_n = batch_snorm_n.to(device)
        batch_snorm_e = batch_snorm_e.to(device)
        batch_labels = batch_labels.to(device)
        batch_scores = net.forward(batch_graphs, batch_x, batch_e, batch_snorm_n, batch_snorm_e)
        gpu_mem = net.gpu_memory(gpu_mem)
        loss = net.loss(batch_scores, batch_labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_loss += loss.detach().item()
        epoch_train_acc += net.accuracy(batch_scores,batch_labels)
        nb_data += batch_labels.size(0)
    epoch_loss /= (iter + 1)
    epoch_train_acc /= nb_data
    return epoch_loss, epoch_train_acc, gpu_mem
```

Train function

```python
# datasets
train_loader = DataLoader(trainset, batch_size=50, shuffle=True, collate_fn=collate)
test_loader = DataLoader(testset, batch_size=50, shuffle=False, collate_fn=collate)
val_loader = DataLoader(valset, batch_size=50, shuffle=False, drop_last=False, collate_fn=collate)

# Create model
net_parameters = {}
net_parameters['input_dim'] = 1
net_parameters['hidden_dim'] = 100
net_parameters['output_dim'] = 8 # nb of classes
net_parameters['L'] = 4
net = GatedGCN_Net(net_parameters)
net = net.to(device)

optimizer = torch.optim.Adam(net.parameters(), lr=0.0005)

epoch_train_losses = []
epoch_test_losses = []
epoch_val_losses = []
epoch_train_accs = []
epoch_test_accs = []
epoch_val_accs = []
for epoch in range(50):

    start = time.time()
    epoch_train_loss, epoch_train_acc, gpu_mem = train_one_epoch(net, train_loader)
    epoch_test_loss, epoch_test_acc = evaluate_network(net, test_loader)
    epoch_val_loss, epoch_val_acc = evaluate_network(net, val_loader)

    print('Epoch {}, time {:.4f}, train_loss: {:.4f}, test_loss: {:.4f}, val_loss: {:.4f} \n
```

Main function

Outline

GNN Pipeline

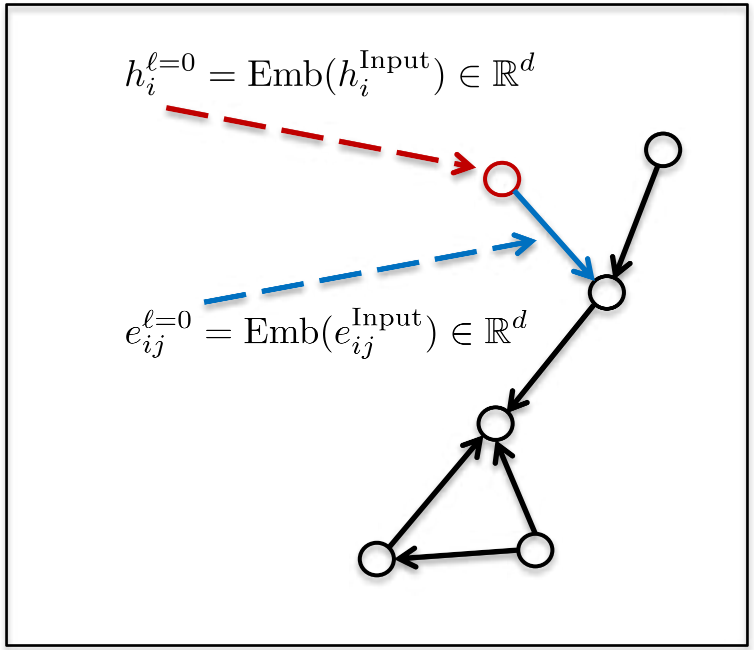
- Standard GNN pipeline :
  - Input layer : Linear embedding of input node/edge features.
  - GNN layers : Apply favorite GNN layer L times.
  - Task-based layer : Graph/node/edge prediction layer.

GNN Pipeline

- Input layer :

h^{Input}
e^{Input}

$\Rightarrow$



$\Rightarrow$

$h^{\ell=0} \in \mathbb{R}^{n \times d}$
$e^{\ell=0} \in \mathbb{R}^{E \times d}$

Input node/edge features

Embedding layer of input features

Input features embedded into d-dim spaces, and passes to the GNN layers.

GNN Pipeline

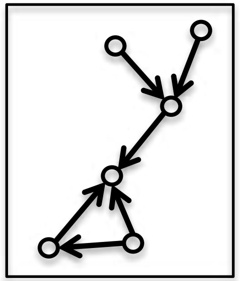
- GNN layers :

$$h^{\ell=0} \in \mathbb{R}^{n\times d}$$
$$e^{\ell=0} \in \mathbb{R}^{E\times d}$$

$\Rightarrow$

$h^{\ell} \rightarrow$ $\rightarrow h^{\ell+1}$

$e^{\ell} \rightarrow$ $\rightarrow e^{\ell+1}$



ℓ^{th} GNN layer

$\Rightarrow$

$$h^{\ell=L} \in \mathbb{R}^{n\times d}$$
$$e^{\ell=L} \in \mathbb{R}^{E\times d}$$

Input of GNNs indexed by $\ell=0$.

GNN layers applied L times

Output of GNNs indexed by $\ell=L$.

GNN Tasks

- Task-based layer
  - **Graph-level prediction :**

$$h^{\mathcal{G}} = \frac{1}{n}\sum_{i=0}^{n} h_i^{\ell=L} \in \mathbb{R}^d \quad \Rightarrow \quad \boxed{\textbf{MLP}} \quad \Rightarrow \quad \text{score} \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K, \ \ K>1 & \text{classification} \end{cases}$$

  - **Node-level prediction :**

$$h_i^{\ell=L} \in \mathbb{R}^d \quad \Rightarrow \quad \boxed{\textbf{MLP}} \quad \Rightarrow \quad \text{score}_i \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K, \ \ K>1 & \text{classification} \end{cases}$$

  - **Edge-level prediction :**

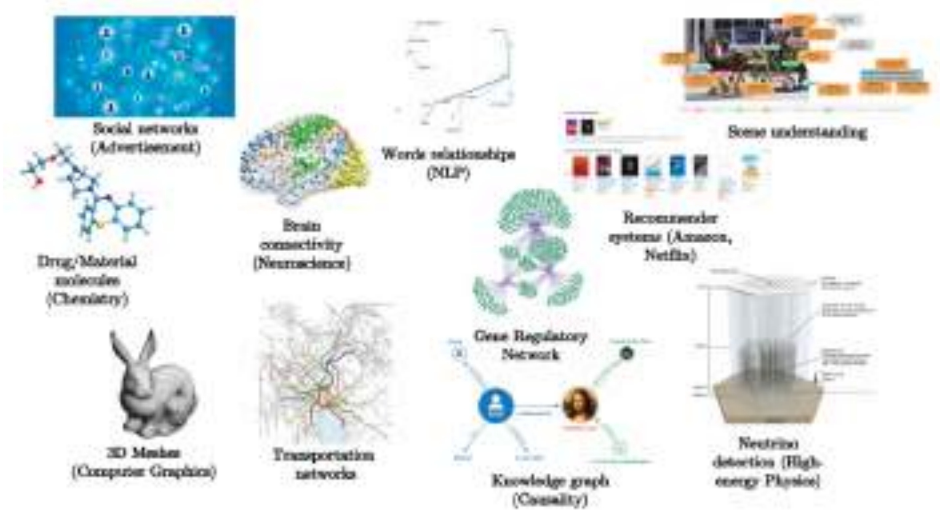
$$e_{ij}^{\text{link}} = \text{Concat}(h_i^{\ell=L}, h_j^{\ell=L}) \in \mathbb{R}^d \quad \Rightarrow \quad \boxed{\textbf{MLP}} \quad \Rightarrow \quad \text{score}_{ij} \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K, \ \ K>1 & \text{classification} \end{cases}$$

Outline

Conclusion

- Contributions :
  - Generalization of ConvNets to data on graphs
  - Re-design convolution operator on graphs
  - Linear complexity for sparse graphs
  - GPU implementation (not yet optimized for sparse matrix-matrix multiplications)
  - Universal learning capacity
  - Multiple and dynamic graphs

- Applications :



"Graphs are the most important discrete models in the world!" - G. Strang (MIT)

Tutorials on Graph Deep Learning

NeurIPS'17
1,000-2,000 participants

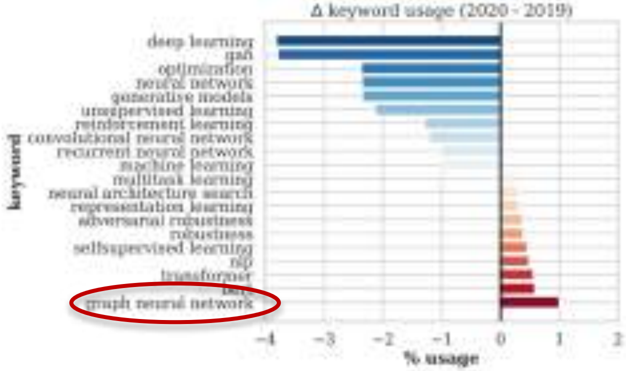
CVPR'17
500-1,000 participants

SIAM Annual Meeting'18

Organizers

- Y. LeCun (NYU, Chief AI Scientist at Facebook, 2019 Turing Award)
- M. Bronstein (Imperial, Head of Graph Deep Learning at Twitter)
- J. Bruna (NYU)
- A. Szlam (Facebook AI Research)
- X. Bresson (NTU, Singapore)

- Top AI/DL conferences organize workshops/tutorials on "Graph Neural Networks" :
  - NeurIPS'20, ICML'20, ICLR'19
  - CVPR'19, ICCV'19
  - Etc



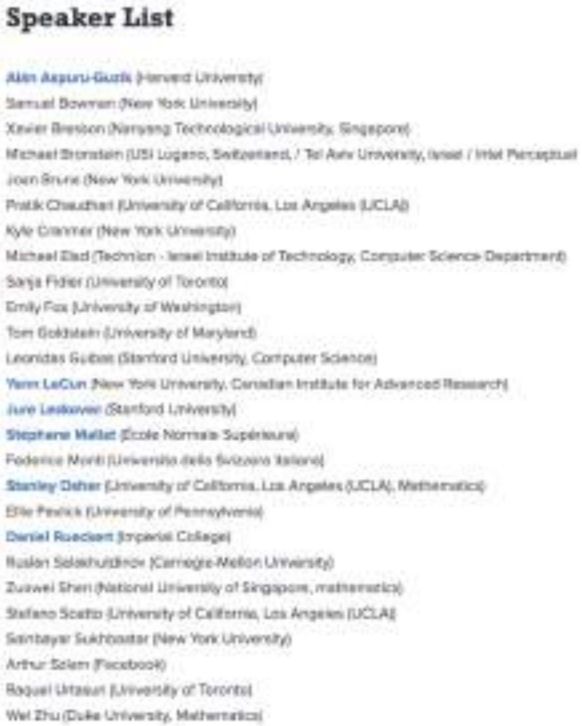
ICLR'20

2018 IPAM-UCLA Workshop

Organizers

- Y. LeCun (NYU, Chief AI Scientist at Facebook, 2019 Turing Award)
- M. Bronstein (Imperial, Head of Graph Deep Learning at Twitter)
- J. Bruna (NYU)
- A. Szlam (Facebook AI Research)
- S. Osher (UCLA, U.S. National Academy of Sciences)
- X. Bresson (NTU, Singapore)

Video talks : https://lnkd.in/fY2-9UU



Speaker List

Alán Aspuru-Guzik (Harvard University)
Samuel Bowman (New York University)
Xavier Bresson (Nanyang Technological University, Singapore)
Michael Bronstein (USI Lugano, Switzerland, / Tel Aviv University, Israel / Intel Perceptual
Joan Bruna (New York University)
Pratik Chaudhari (University of California, Los Angeles (UCLA))
Kyle Cranmer (New York University)
Michael Elad (Technion - Israel Institute of Technology, Computer Science Department)
Sanja Fidler (University of Toronto)
Emily Fox (University of Washington)
Tom Goldstein (University of Maryland)
Leonidas Guibas (Stanford University, Computer Science)
Yann LeCun (New York University, Canadian Institute for Advanced Research)
Jure Leskovec (Stanford University)
Stephane Mallat (Ecole Normale Supérieure)
Federico Monti (Universita della Svizzera Italiana)
Stanley Osher (University of California, Los Angeles (UCLA), Mathematics)
Ellie Pavlick (University of Pennsylvania)
Daniel Rueckert (Imperial College)
Ruslan Salakhutdinov (Carnegie-Mellon University)
Zuowei Shen (National University of Singapore, mathematics)
Stefano Soatto (University of California, Los Angeles (UCLA))
Sainbayar Sukhbaatar (New York University)
Arthur Szlam (Facebook)
Raquel Urtasun (University of Toronto)
Wei Zhu (Duke University, Mathematics)

2019 IPAM-UCLA Workshop

Organizers

- Y. LeCun (NYU, Chief AI Scientist at Facebook, 2019 Turing Award)
- Rene Vidal (Johns Hopkins, Director Data Science Institute)
- Rebecca Willett (Chicago U.)
- S. Osher (UCLA, U.S. National Academy of Sciences)
- X. Bresson (NTU, Singapore)

Video talks : https://bit.ly/2w8EtLV

Speaker List

Mikhail Belkin (Ohio State University)
Xavier Bresson (Nanyang Technological University, Singapore)
Soumith Chintala (Facebook AI Research)
Taco Cohen (Qualcomm AI Research)
Kostas Daniilidis (University of Pennsylvania)
Tom Goldstein (University of Maryland)
Bahram Jalali (University of California, Los Angeles (UCLA))
Thomas Kipf (Universiteit van Amsterdam)
Roy Lederman (Yale University, Applied Mathematics)
Jure Leskovec (Stanford University)
Federico Monti (Università della Svizzera italiana)
Mathias Niepert (NEC Laboratories Europe)
Stanley Osher (University of California, Los Angeles (UCLA), Mathematics)
Hamed Pirsiavash (University of Maryland Baltimore County)
Marc Pollefeys (ETH Zurich)
Srikumar Ramalingam (University of Utah)
Thiago Serra (Mitsubishi Electric Research Laboratories (Merl))
Jeremias Sulam (Johns Hopkins University)
Arthur Szlam (Facebook)
Jian Tang (HEC Montréal)
Luc Van Gool (ETH Zurich)
Rene Vidal (Johns Hopkins University)
Ersin Yumer (Uber ATG)
Hongyang Zhang (Carnegie Mellon University)

Thank you

Xavier Bresson

xbresson@ntu.edu.sg

http://www.ntu.edu.sg/home/xbresson
https://github.com/xbresson
https://twitter.com/xbresson
https://www.facebook.com/xavier.bresson.1
https://www.linkedin.com/in/xavier-bresson-738585b