

Graph Convolutional Networks Spectral & Spatial Techniques

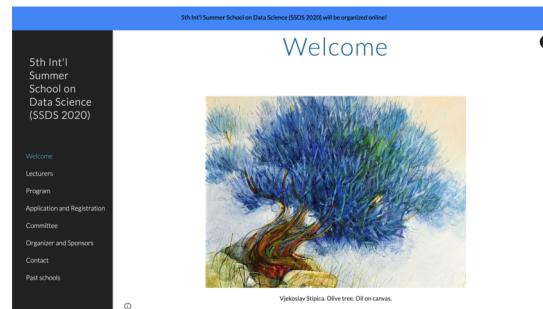
Xavier Bresson



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

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

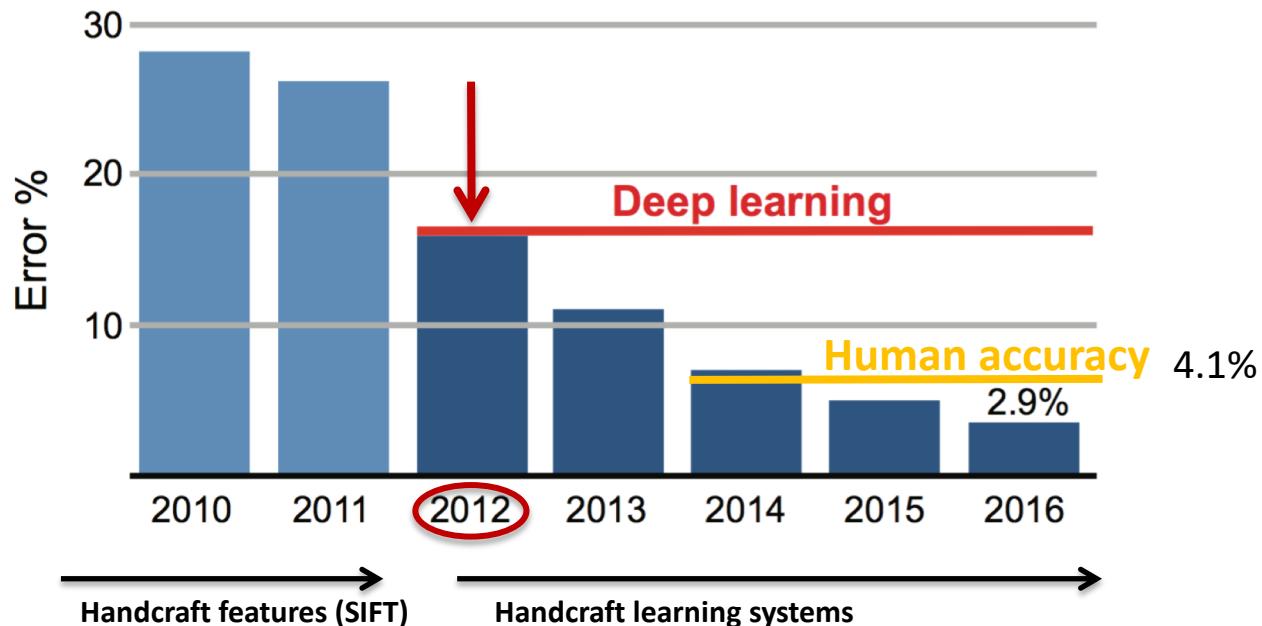
- Part 1: Traditional ConvNets
 - Architecture
 - Graph Domain
 - Convolution
- Part 2: Spectral Graph ConvNets
 - Spectral Convolution
 - Spectral GCNs
- Part 3: Spatial Graph ConvNets
 - Template Matching
 - Isotropic GCNs
 - Anisotropic GCNs
 - Lab on GatedGCNs
- Part 4: GNN Tasks
- Conclusion

ConvNets

- A breakthrough in Computer Vision :

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

IMAGENET



- Also in Speech and Natural Language Processing.



Oliver Schröbe [DE] Ich verstehe nicht, weshalb sich alle plötzlich darüber aufregen, dass wir Steuergelder nach Griechenland "verschwenden". Als wären Steuergelder bisher sinnvoll angelegt worden...

May 5 at 10:06am via TweetDeck · Comment · Like
· Translate

Oliver Schröbe [DE] I do not understand why all of a sudden get e about the fact that we are wasting taxpayers' money to Greece. Wh tax money would have been useful to create ...

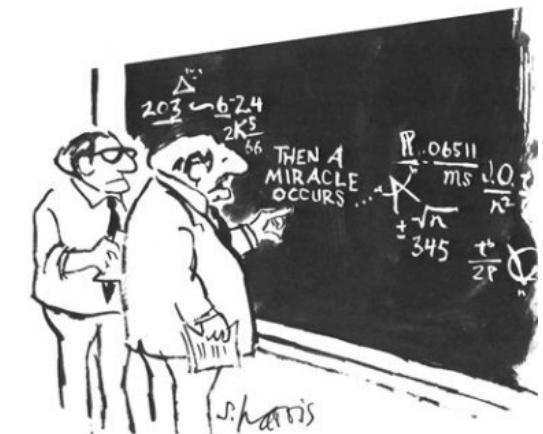
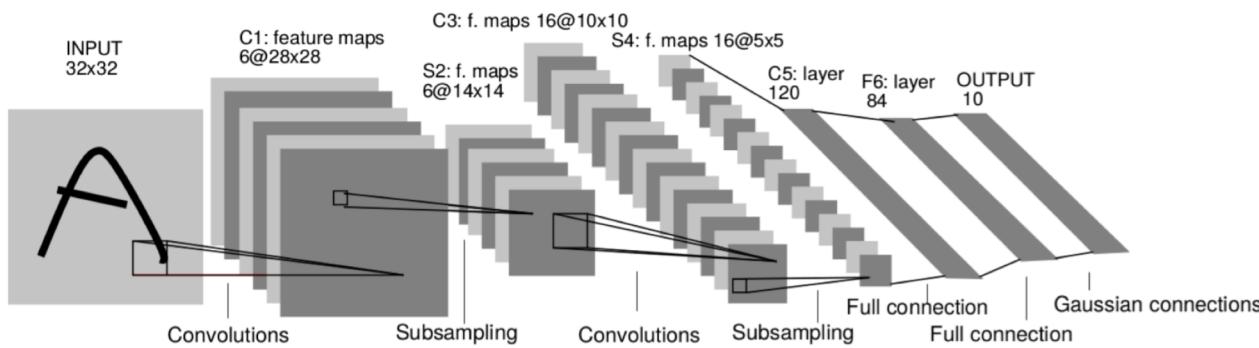
May 5 at 10:06am via TweetDeck · Comment · Like
· Untranslate

ConvNets

- ConvNets are **powerful architectures** to solve high-dimensional learning problems.
- Curse of dimensionality :
 $\text{dim(image)} = 1024 \times 1024 \approx 10^6$
For $N=10$ samples/dim $\Rightarrow 10^{1,000,000}$ points

TECHNOLOGY The New York Times SUBSCRIBE NOW

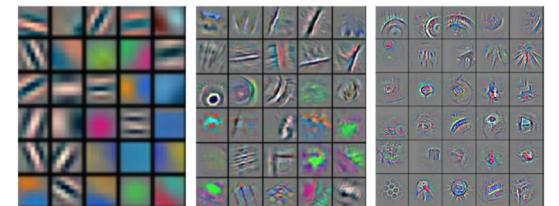
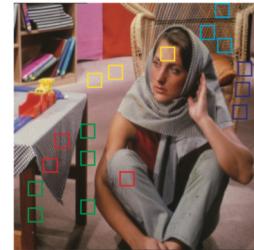
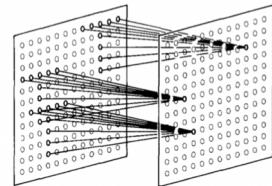
Turing Award Won by 3 Pioneers in Artificial Intelligence



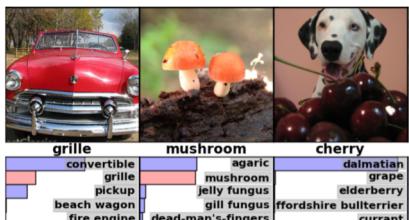
"I think you should be more explicit here in step two."

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

```
/* Duplicate lsm field information. The lsm_rule is opaque, so
 * we can't call lsm_rule->lsn() directly.
 * static inline int audit_dupe_lsm_field(struct audit_field *df,
 *                                     struct audit_field *sf)
{
    int ret = 0;
    /* If we have our own copy of lsm_str, /.
     * lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
     * if (!lsm_str)
     *     return -ENOMEM;
     * df->lsm_str = lsm_str;
     * df->refcnt += sf->refcnt;
     * df->rule_init(df->type, df->op, df->lsm_str);
     */
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (df->rule->INVALID)
        pr_warn("audit rule '%s' is invalid\n",
                df->rule->name);
    ret = 0;
}
return ret;
```

NLP



Speech



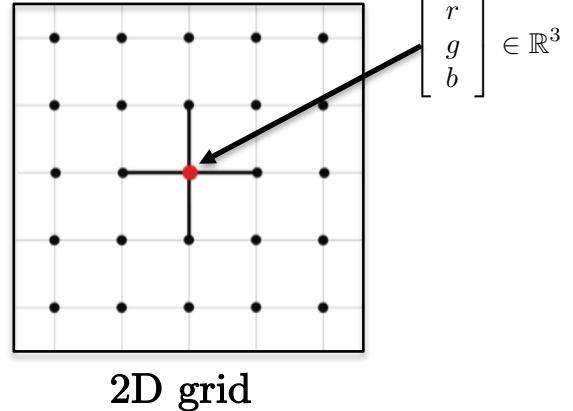
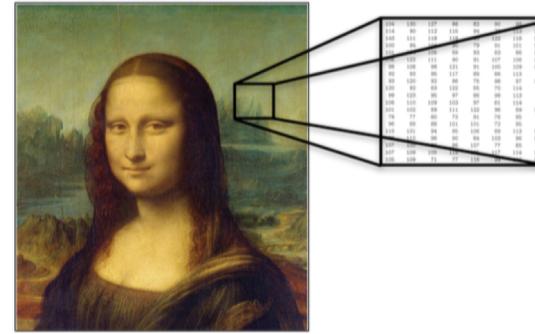
Game of Go

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

Data Domain

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



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

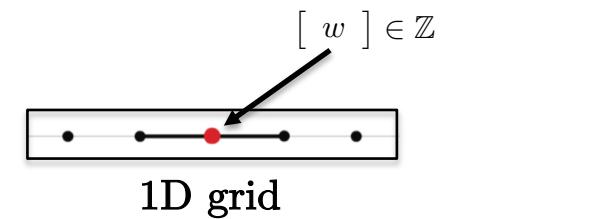
O ROMEO, ROMEO, WHEREFORE ART THOU ROMEO?
Although we use "wherefore," if at all, as a synonym for "why," Juliet uses the word in a more limited sense. By "wherefore" Juliet means "the reason why" if she had asked "why" she would have been named! She wouldn't be distinguishing the two major meanings of "why"—"from what cause" (in the past) and "for what purpose" (in the future). "Wherefore" clearly emphasizes the latter sense, which is why "why" and "wherefore" are equivalent.

"Wherefore" and its partner "wherefore" reflect the basic tending of English to use spatial ideas—"here"—to represent logical ideas, such as cause and effect.

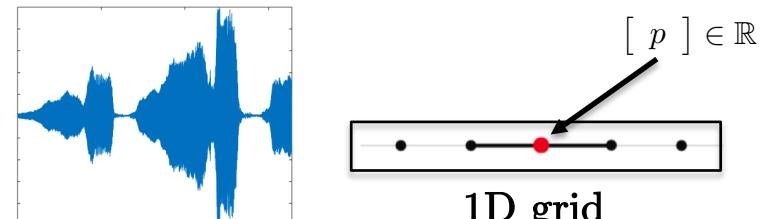
WHAT'S IN A NAME? THAT WHICH WE CALL A ROSE BY ANY OTHER WORD WOULD SMELL AS SWEET
There's such a thing as generic Shakespeare today, this is it. Both "sweet" and "smell" are instant Bard, although the latter is, as many forget, merely a paraphrase. From the romantic declaration to the crass advertisement, these phrases have served generations with complete flexibility.

"What's in a name?" is the less specific of the two phrases, and also the less common. Juliet here merely rehearses in a different form the point of "What's a Montague?" moving, like a good Renaissance oration, from the particular to the general. The two words, however, ought to be separable from the things they name. Romeo never does change his name, and it wouldn't have done much good anyway. Whether or not he's destined to end up as essentially a Capulet, their families will continue to act that way.

"That which we call a rose by any other word would smell as sweet" seems bloated to the modern ear. But we're accustomed to the paraphrase, which never occurred to the playwright or his audience. It's a little futile to second-guess Shakespeare now, but he did have to fill out a bit of space. In the original, Juliet's use of "word" instead of "name," we can perhaps be grateful; she already uses "name" six times in fifteen and a half lines.



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



Graph Domain



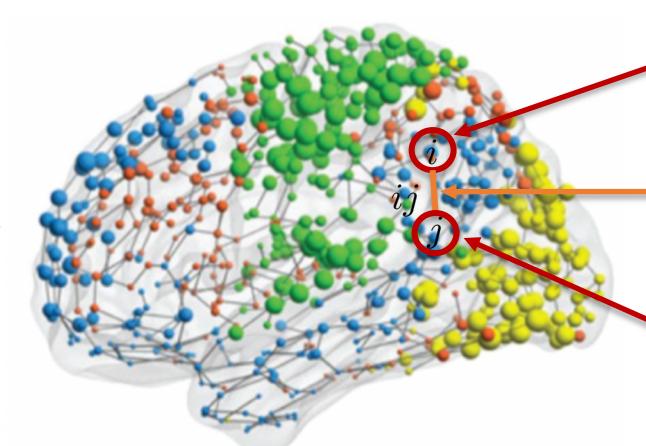
Social networks
(Advertisement/
recommendation)

Brain connectivity
(sMRI)

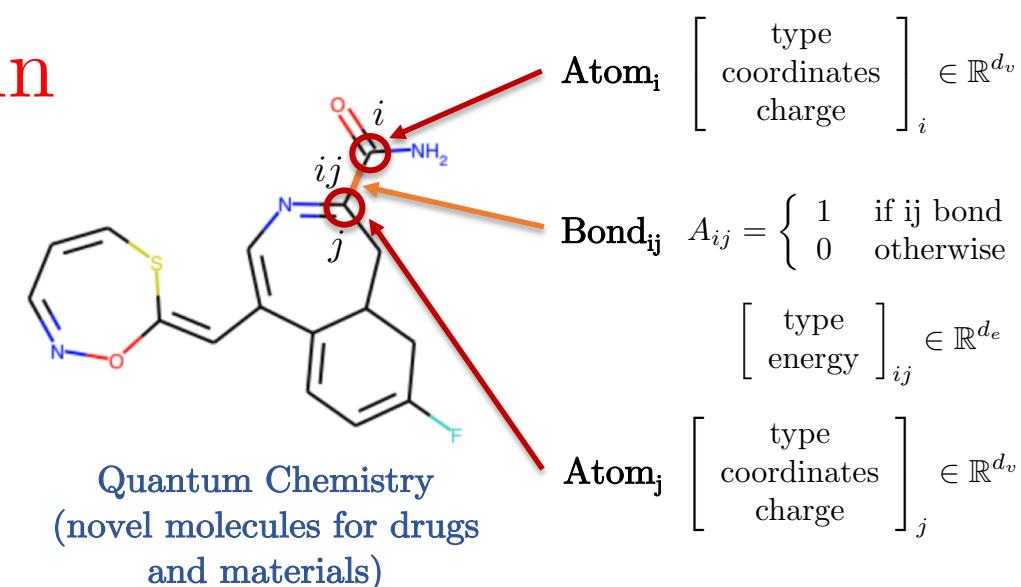
$$\text{User}_i \begin{bmatrix} \text{messages} \\ \text{images} \\ \text{videos} \end{bmatrix}_i \in \mathbb{R}^d$$

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

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



Brain analysis
(Neuroscience/neuro-diseases)



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

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

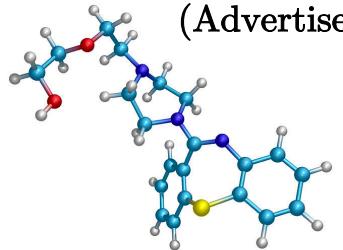
$$\text{ROI}_j \begin{bmatrix} a_1 \\ \vdots \\ a_T \end{bmatrix}_j \in \mathbb{R}^T$$

Functional
activations (fMRI)

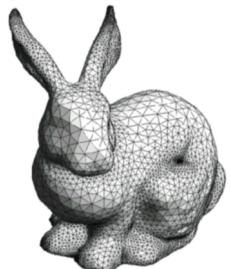
Graph Domain



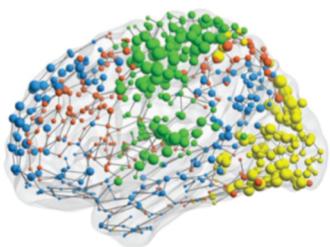
Social networks
(Advertisement)



Drug/Material
molecules
(Chemistry)



3D Meshes
(Computer Graphics)

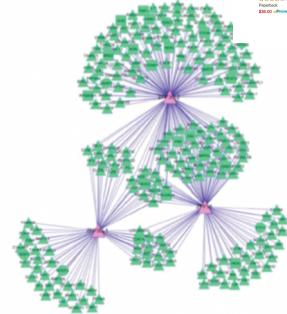


Brain
connectivity
(Neuroscience)

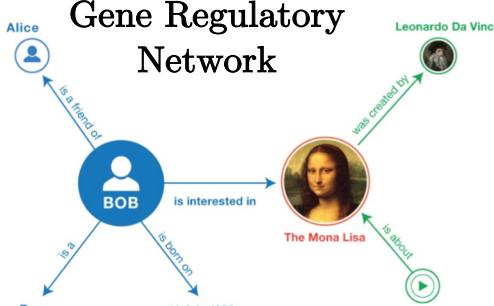


Transportation
networks

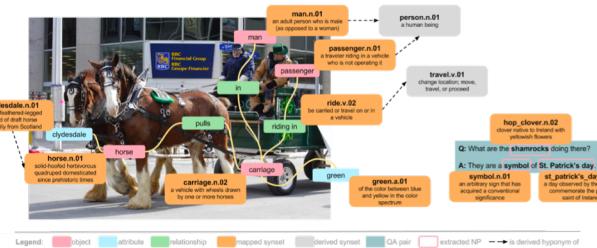
Words relationships
(NLP)



Gene Regulatory
Network



Knowledge graphs

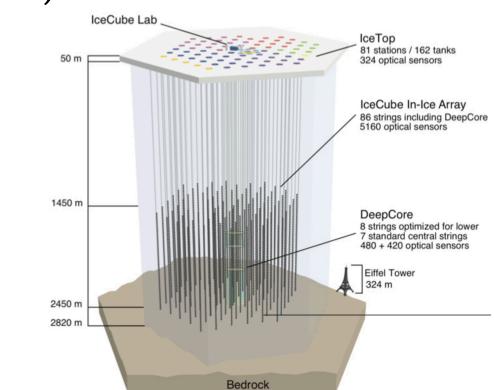


Scene understanding

=



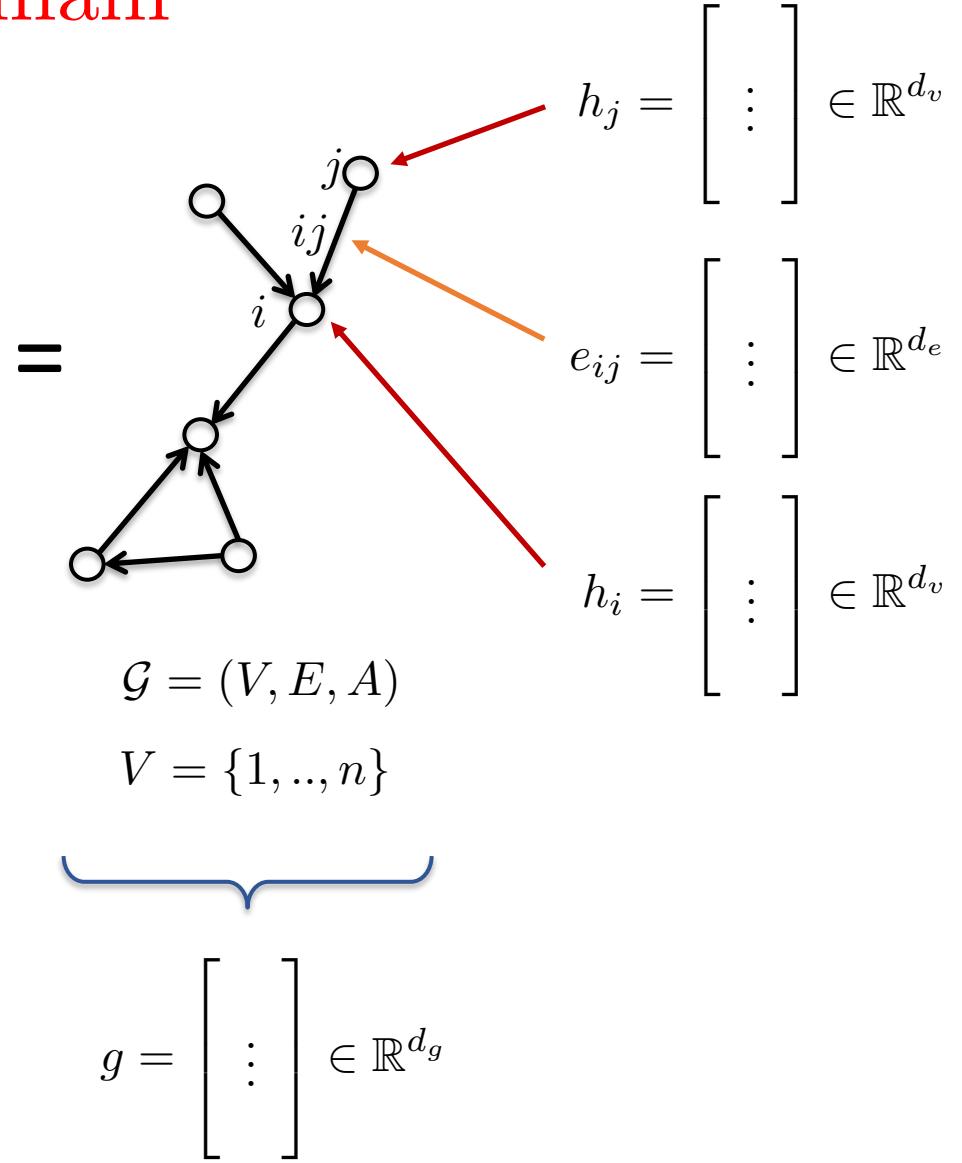
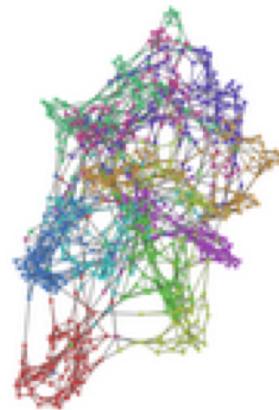
Graph



Neutrino
detection (High-
energy Physics)

Graph Domain

- Graphs G are defined by :
 - Vertices V
 - Edges E
 - Adjacency matrix A
- Graph features :
 - Node features : h_i, h_j (atom type)
 - Edge features : e_{ij} (bond type)
 - Graph features : g (molecule energy)



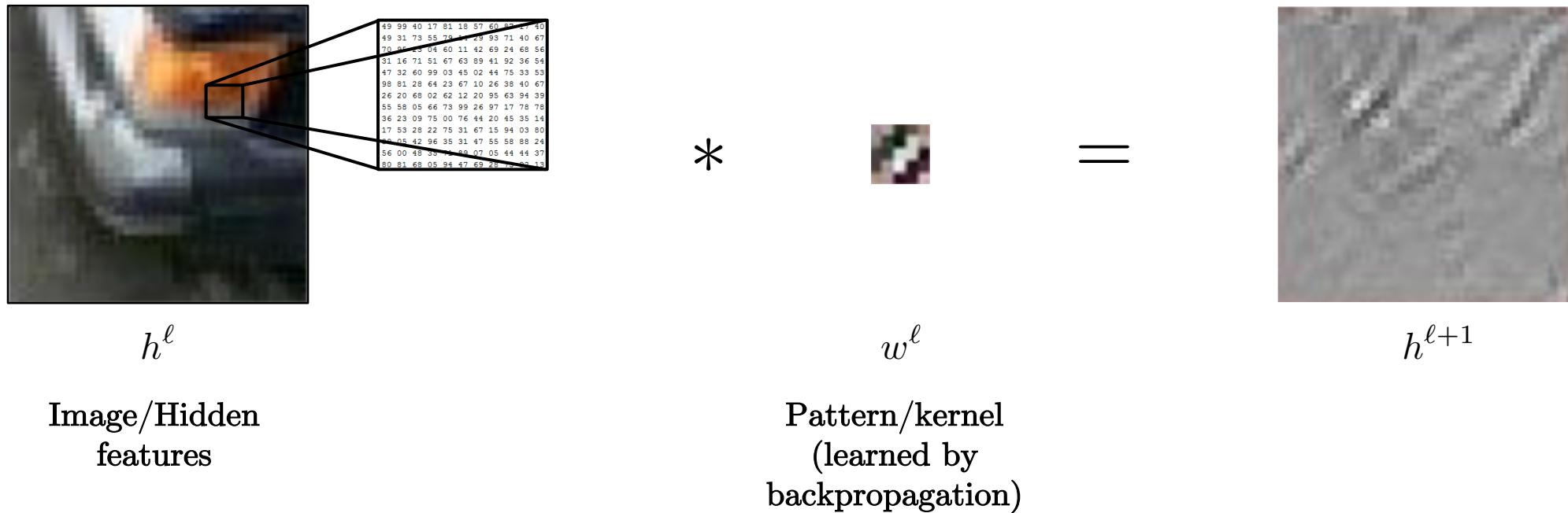
Outline

- **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

Convolution

- Convolutional layer (for grids) :

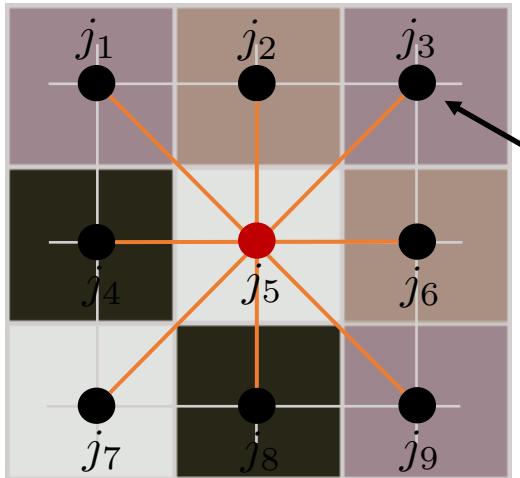
$$h^{\ell+1} = w^\ell * h^\ell$$
$$\begin{matrix} n_1 \times n_2 \times d \\ 3 \times 3 \times d \end{matrix} \quad \quad \quad \begin{matrix} n_1 \times n_2 \times d \end{matrix}$$



Convolution

- How to define convolution ?

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



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

$$w^l$$

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

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

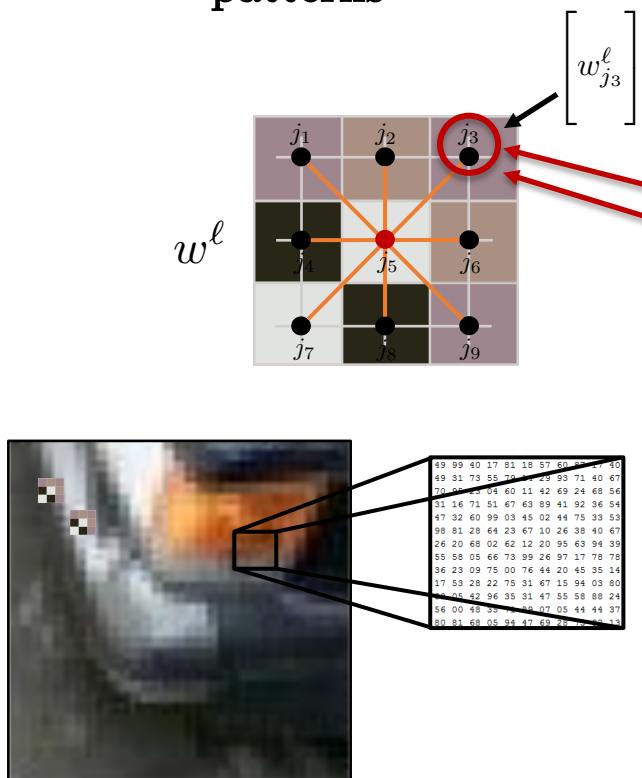
Template features at j_3

$$\begin{aligned}
 h_i^{\ell+1} &= w^l * h_i^\ell \\
 &= \sum_{\substack{j \in \Omega \\ j \in \mathcal{N}_i}} \langle w_j^\ell, \underbrace{h_{i-j}^\ell}_{h_{i+j}^\ell = h_{ij}^\ell} \rangle \\
 &= \sum_{j \in \mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle \\
 &= \sum_{j \in \mathcal{N}_i} \langle \begin{bmatrix} w_j^\ell \end{bmatrix}, \begin{bmatrix} h_{ij}^\ell \end{bmatrix} \rangle
 \end{aligned}$$

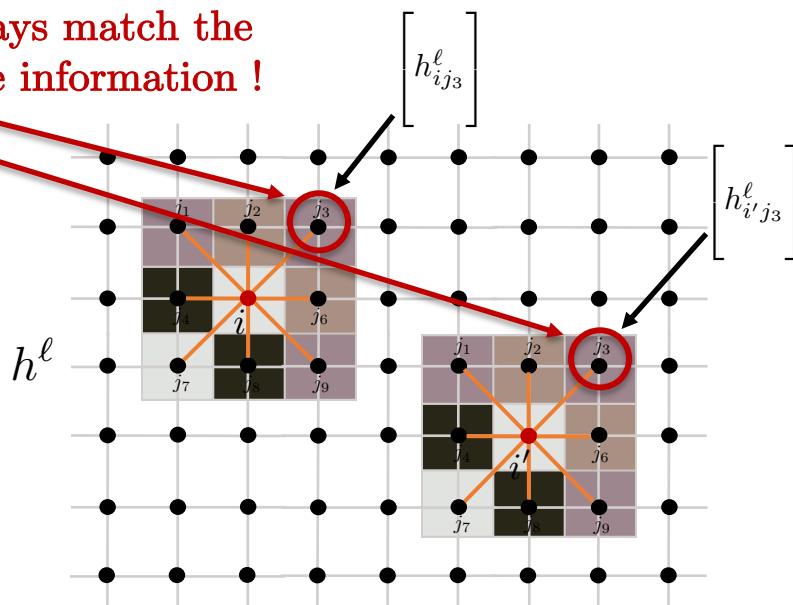
Convolution

- How to define convolution ?

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



Node indices j_3
always match the
same information !



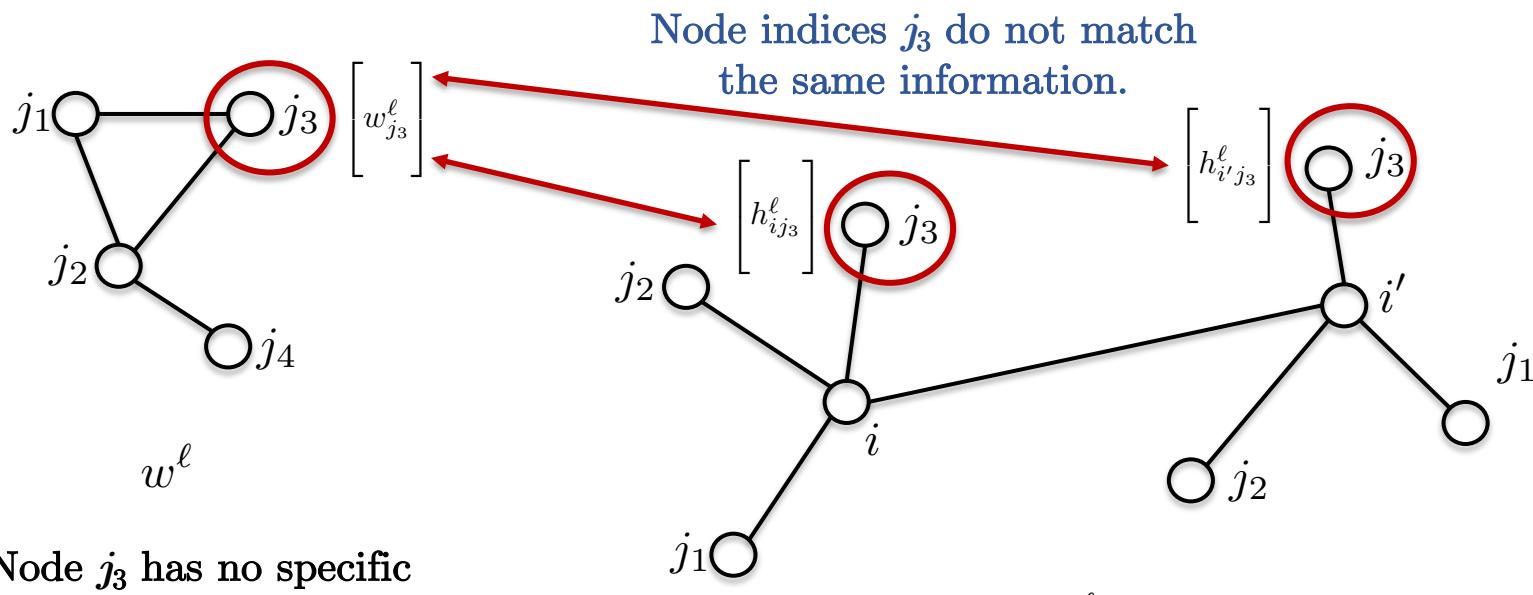
Same node ordering $\forall i$
 j_3 is always positioned at the
top-right corner of the template.

$$\begin{aligned}
 h_i^{\ell+1} &= w^\ell * h_i^\ell \\
 &= \sum_{j \in \mathcal{N}_i} \langle w_j^\ell, h_{ij}^\ell \rangle \\
 &= \sum_{j \in \mathcal{N}_i} \langle \begin{bmatrix} w_j^\ell \end{bmatrix}, \begin{bmatrix} h_{ij}^\ell \end{bmatrix} \rangle \\
 &\quad \langle \begin{bmatrix} w_{j_3}^\ell \end{bmatrix}, \begin{bmatrix} h_{ij_3}^\ell \end{bmatrix} \rangle \\
 &\quad \langle \begin{bmatrix} w_{j_3}^\ell \end{bmatrix}, \begin{bmatrix} h_{i'j_3}^\ell \end{bmatrix} \rangle
 \end{aligned}$$

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 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.

$$\langle \begin{bmatrix} w_{j_3}^\ell \end{bmatrix}, \begin{bmatrix} h_{ij_3}^\ell \end{bmatrix} \rangle$$

$$\langle \begin{bmatrix} w_{j_3}^\ell \end{bmatrix}, \begin{bmatrix} h_{i'j_3}^\ell \end{bmatrix} \rangle$$

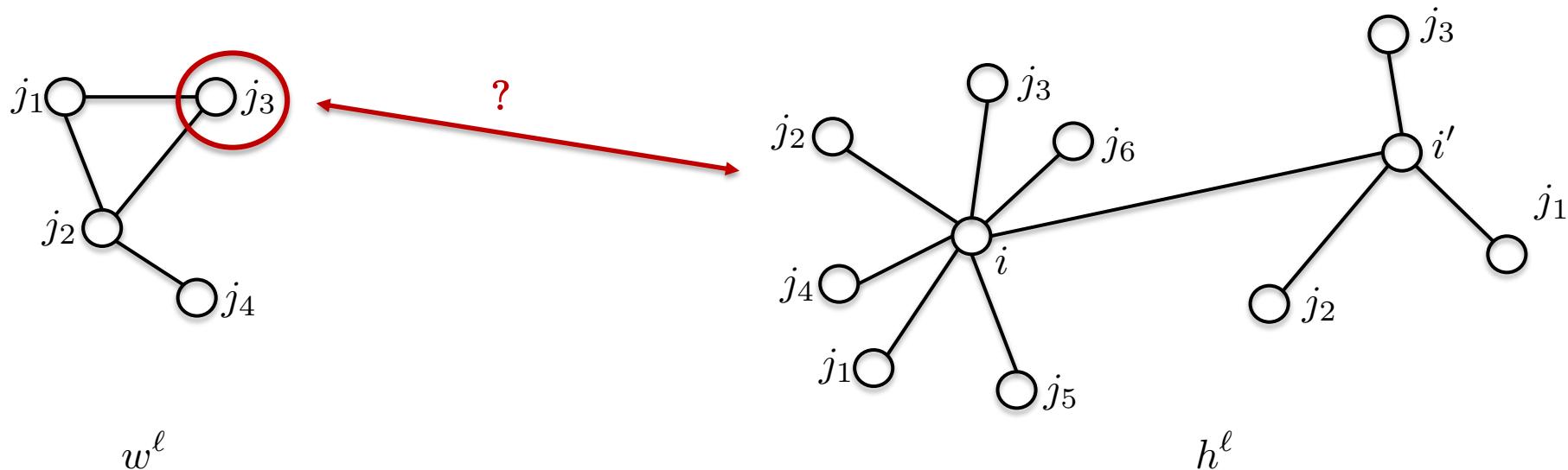
These matching scores have **no meaning** as they do not compare the same information.

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

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

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

- 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

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

Conference Board of the Mathematical Sciences
CBMS
Regional Conference Series in Mathematics
Number 92

Spectral Graph Theory
Fan R. K. Chung

Published for the
Conference Board of the Mathematical Sciences
American Mathematical Society
Providence, Rhode Island
with support from the
National Science Foundation



[1] FRK Chung, Spectral graph theory, 1997

Graph Laplacian

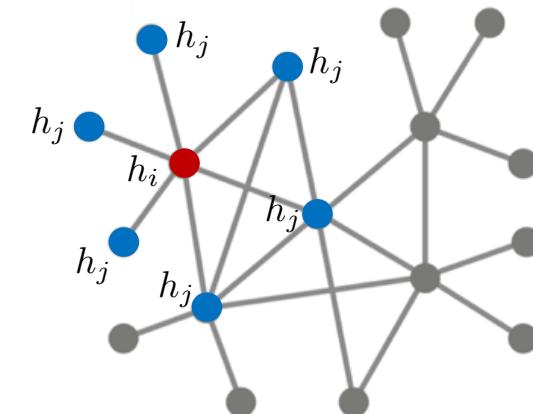
- Core operator in Spectral Graph Theory

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

where $D = \text{diag}(\sum_{j \neq i} A_{ij})$

- 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

$n \times n$

Lap Eigenvectors/
Fourier functions

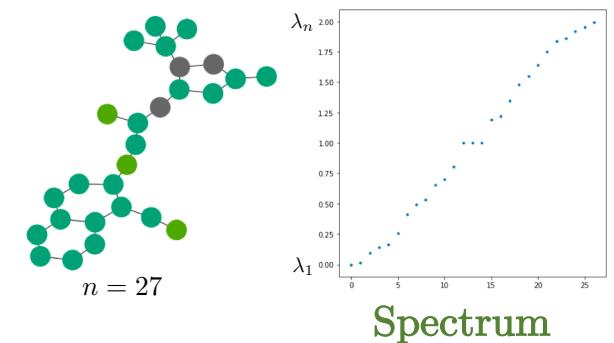
$$\Delta = \Phi^T \Lambda \Phi$$

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

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

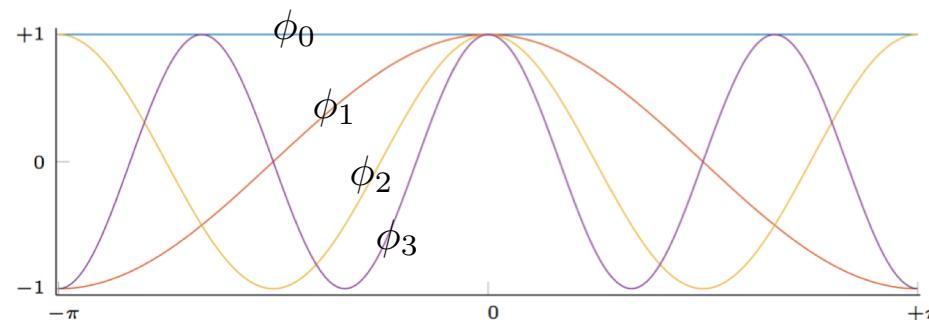
and $\Delta \phi_k = \lambda_k \phi_k$, $k = 1, \dots, n$

$n \times 1$



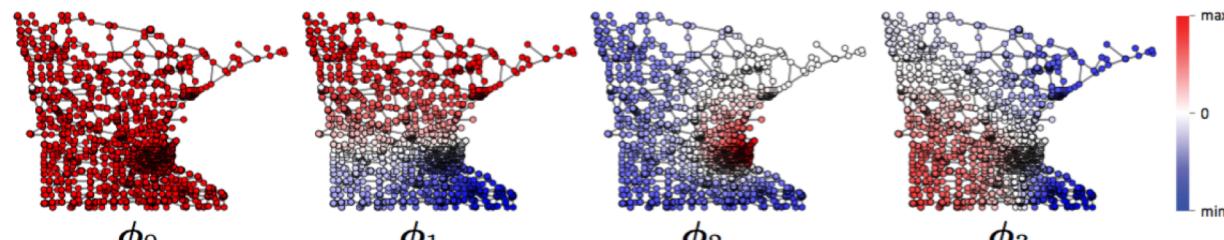
Fourier Functions

- 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] :

$$\begin{aligned}
 h &= \sum_{k=1}^n \underbrace{\langle \phi_k, h \rangle}_{\substack{\mathcal{F}(h)_k = \hat{h}_k = \phi_k^T h \\ \text{scalar}}} \phi_k \\
 &= \sum_{k=1}^n \hat{h}_k \phi_k \\
 &= \underbrace{\Phi \hat{h}}_{\mathcal{F}^{-1}(\hat{h})}
 \end{aligned}$$

Fourier transforms
are one line of code
(linear operations)



$$\begin{array}{l}
 \mathcal{F}(h) = \Phi^T h \\
 n \times 1 = \hat{h}
 \end{array}$$

Fourier Transform/
coefficients of Fourier Series

$$\begin{array}{l}
 \mathcal{F}^{-1}(\hat{h}) = \Phi \hat{h} \\
 n \times 1 = \Phi \Phi^T h = h \text{ as } \mathcal{F}^{-1} \circ \mathcal{F} = \Phi \Phi^T = I
 \end{array}$$

Inverse Fourier Transform

Orthonormal basis/
Invertible transformation

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

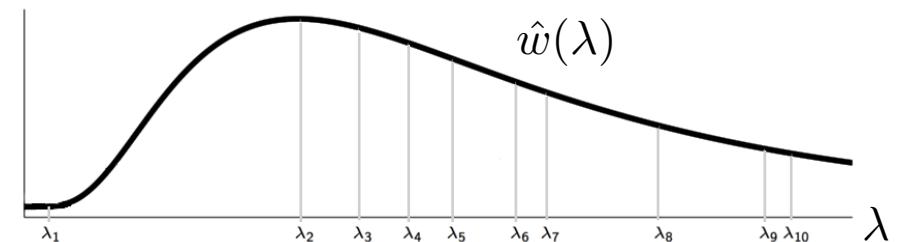
Convolution Theorem

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

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

$$\hat{w} = \begin{bmatrix} \hat{w}(\lambda_1) \\ \vdots \\ \hat{w}(\lambda_n) \end{bmatrix}$$

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



Expensive computation $O(n^2)$
No FFT

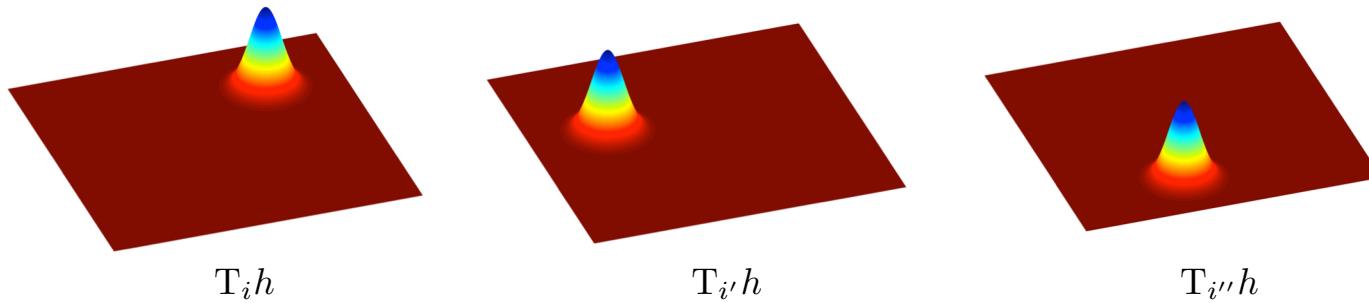
Spectral function/filter

No Shift Invariance for Graphs

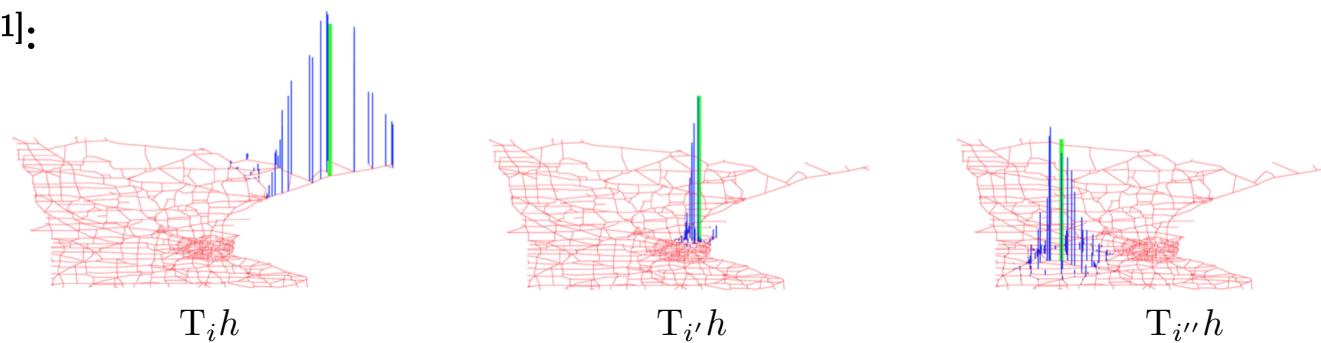
- A signal h on graph can be translated to vertex i as follows :

$$T_i h = \delta_i * h$$

- Grid/Euclidean domain :



- Graph domain^[1]:



[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]

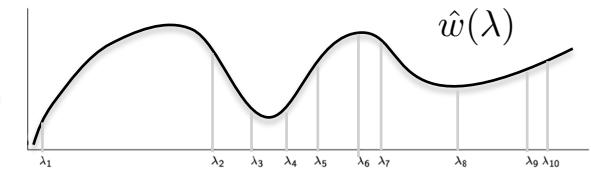
- Graph spectral convolutional layer :

$$\begin{aligned}
 h^{\ell+1} &= \eta(w^\ell * h^\ell) \\
 &= \eta(\hat{w}^\ell(\Delta)h^\ell) \\
 &= \eta(\Phi \hat{w}^\ell(\Lambda) \Phi^T h^\ell)
 \end{aligned}$$

Spatial filter



Spectral filter



- 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)

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

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

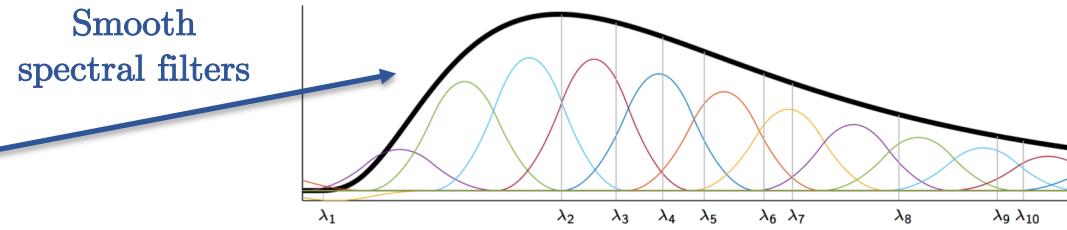
SplineGCNs^[1,2]

- Graph spectral convolutional layer :

$$\begin{aligned}
 h^{\ell+1} &= \eta(w^\ell * h^\ell) \\
 &= \eta(\Phi \hat{w}^\ell(\Lambda) \Phi^T h^\ell) \\
 \hat{w}^\ell(\Lambda) &= \text{diag}(B w^\ell) \\
 &\quad \xrightarrow{n \times n} K \times 1 \\
 &\quad \underbrace{\qquad\qquad\qquad}_{n \times K} \\
 &\quad \xrightarrow{n \times n} K \times 1
 \end{aligned}$$

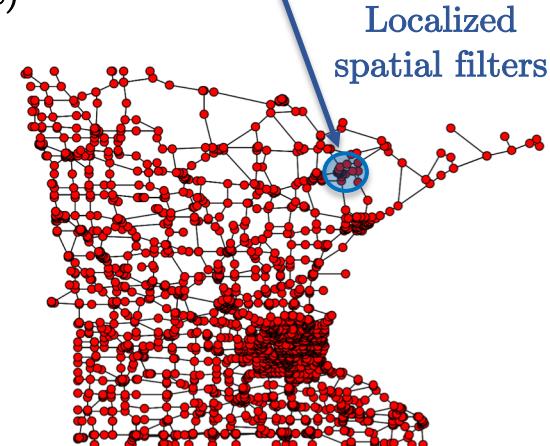
The K coefficients w^ℓ are learned by backpropagation

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



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

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



- 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 :
$$O(w * h) = O(\Phi \hat{w}^\ell(\Lambda) \Phi^T h) = O(n^2)$$

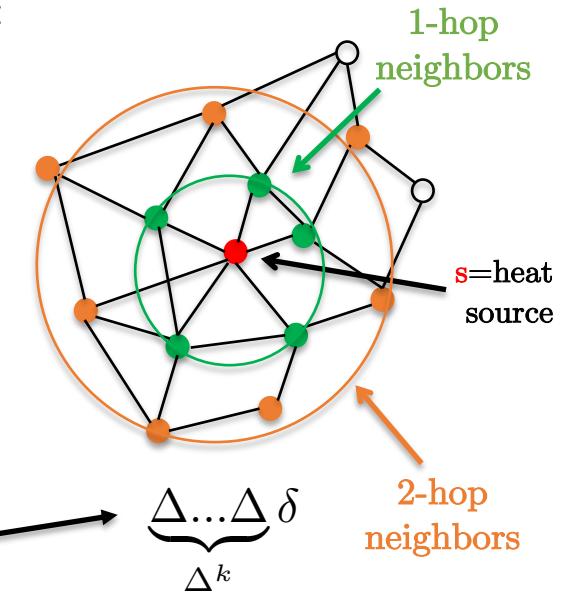
$n \times d$ $n \times n$ $n \times n$ $n \times d$
Full matrix

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

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

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

Coefficients w_k are learned by backpropagation.



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 :

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

**Recursive
equation**

- Sequence $\{X_k\}$ is generated by multiplying a matrix Δ and a vector X_{k-1} \Rightarrow Complexity is $O(E \cdot K) = O(n)$ 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 :

$$\begin{aligned}
 h^{\ell+1} &= \eta \left(\sum_{k=0}^{K-1} w_k^\ell \Delta^k h^\ell \right) \\
 &= \eta \left(\sum_{k=0}^{K-1} w_k^\ell X_k \right) \\
 &= \eta \left((w^\ell)^T \bar{X} \right) \\
 &\quad \underbrace{\hspace{1cm}}_{\text{reshape}}_{1 \times nd} \\
 &\quad \text{reshape} \\
 &\quad n \times d
 \end{aligned}$$

with $\bar{X} = \begin{bmatrix} - & \bar{X}_0 & - \\ \vdots & \ddots & \vdots \\ - & \bar{X}_{K-1} & - \end{bmatrix}$

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

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

- 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)

$$1, x, x^2, x^3, \dots \rightarrow \Delta^0, \Delta^1, \Delta^2, \Delta^3, \dots$$

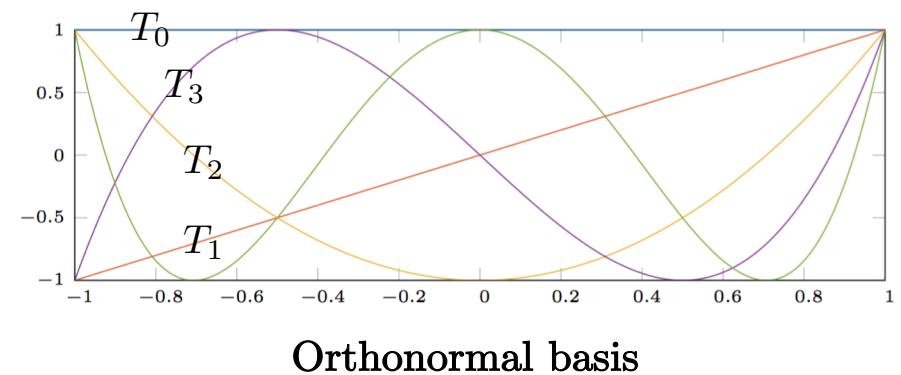
Chebyshev Polynomials

- Graph spectral convolution with Chebyshev polynomials :

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

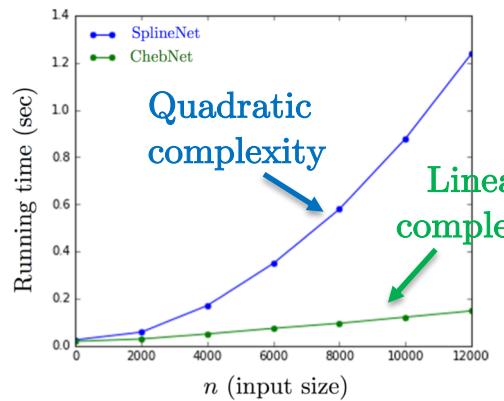
Recursive
equation

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

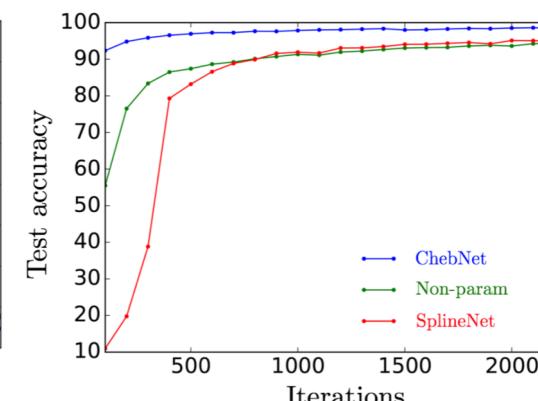
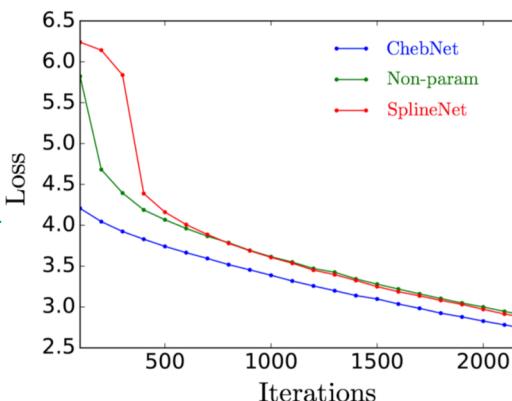


MNIST Numerical Experiment^[1]

Running time



Optimization



Accuracy

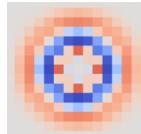
Model	Order	Accuracy
LeNet5	-	99.33%
SplineNet	25	97.75%
ChebNet	25	99.14%

- 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] :

$$h^{\ell+1} = \eta(\hat{w}(\Delta_1, \Delta_2) * h^\ell)$$

$$h^{\ell+1} = \eta\left(\sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} w_{k_1, k_2} T_{k_1}(\Delta_1) T_{k_2}(\Delta_2) h^\ell\right)$$

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

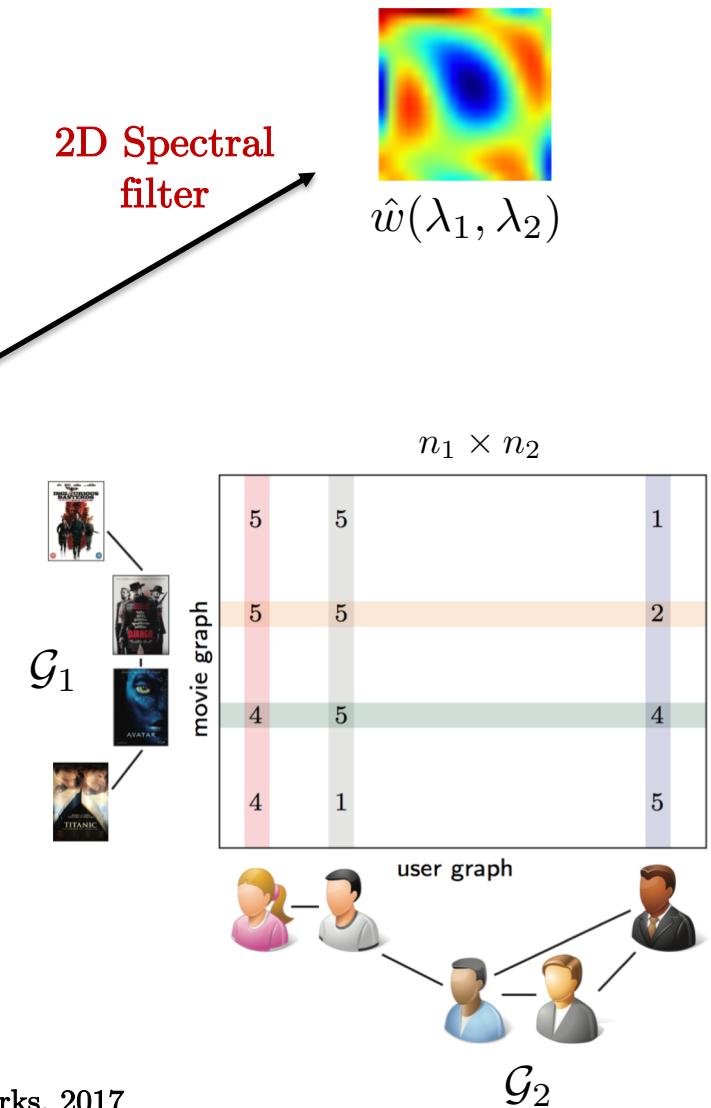
The K_1, K_2 coefficients w_{k_1, k_2} are learned by backpropagation.

- Recommendation task (Netflix)

- Matrix Completion with Multiple Graphs^[2] :

$$\min_{h \in \mathbb{R}^{n_1 \times n_2}} \|h\|_* + \mu \|\Omega \circ (h - r)\|_F^2$$

$$+ \mu_1 \underbrace{\text{tr}(h \Delta_1 h^\top)}_{\|h\|_{\mathcal{G}_1}^2} + \mu_2 \underbrace{\text{tr}(h^\top \Delta_2 h)}_{\|h\|_{\mathcal{G}_2}^2}$$



[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

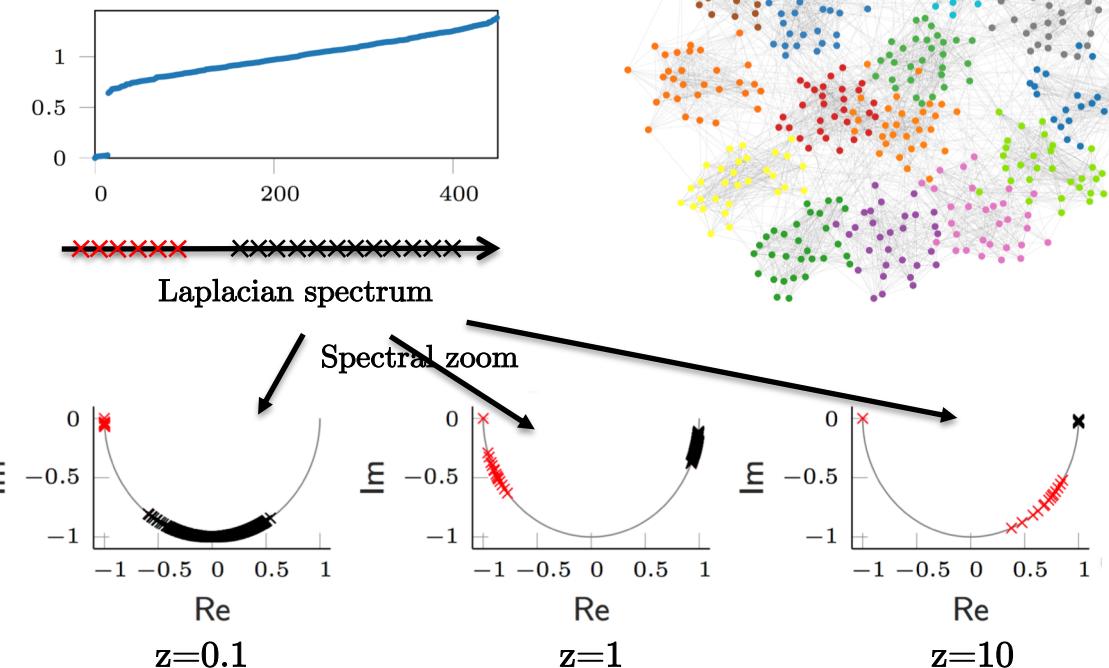
CayleyNets^[1]

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

Spectral zoom z focuses on frequency bands.

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

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

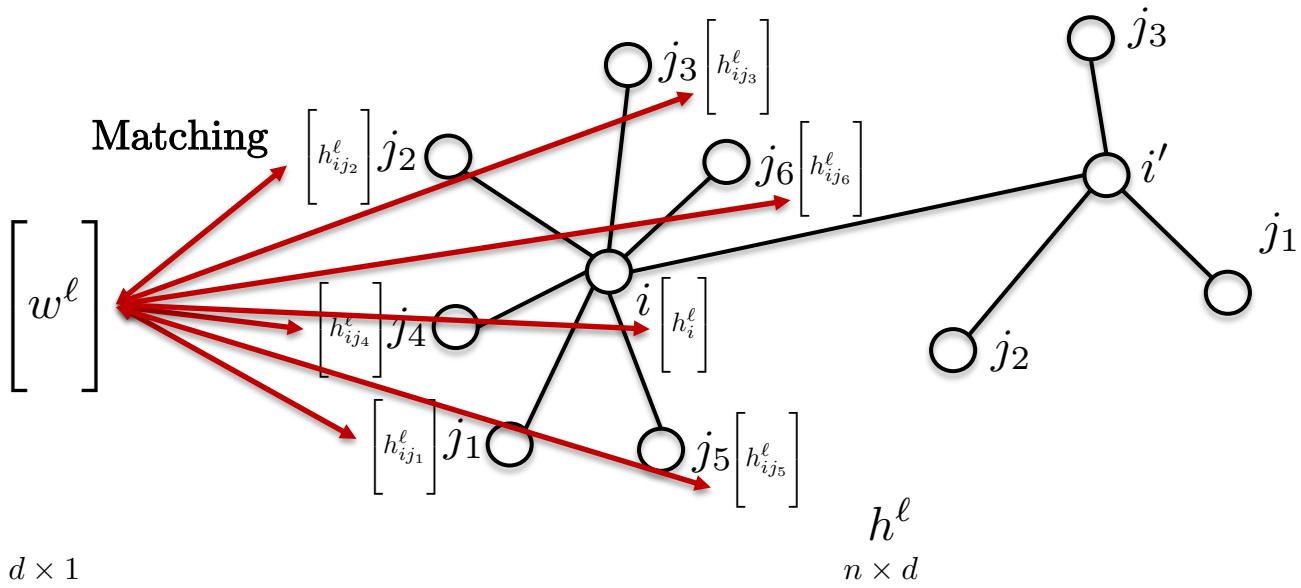
[1] R Levie, F Monti, X Bresson, MM Bronstein, CayleyNets: Graph convolutional neural networks with complex rational spectral filters, 2018

Outline

- 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

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 !



$$h_i^{\ell+1} = \eta \left(\underbrace{\sum_{j \in \mathcal{N}_i} \langle w^\ell, h_{ij}^\ell \rangle}_{\substack{\text{scalar} \\ \text{One feature}}} \right)$$

$$h_i^{\ell+1} = \eta \left(\sum_{j \in \mathcal{N}_i} W^\ell h_{ij}^\ell \right)$$

$d \times 1$
 $d \text{ features}$

$$h^{\ell+1} = \eta \left(\underbrace{A h^\ell W^\ell}_{\substack{d \times d \\ n \times d}} \right)$$

$n \times d$
 $n \times n$

Vectorial representation

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 GCNs^[1,2,3]

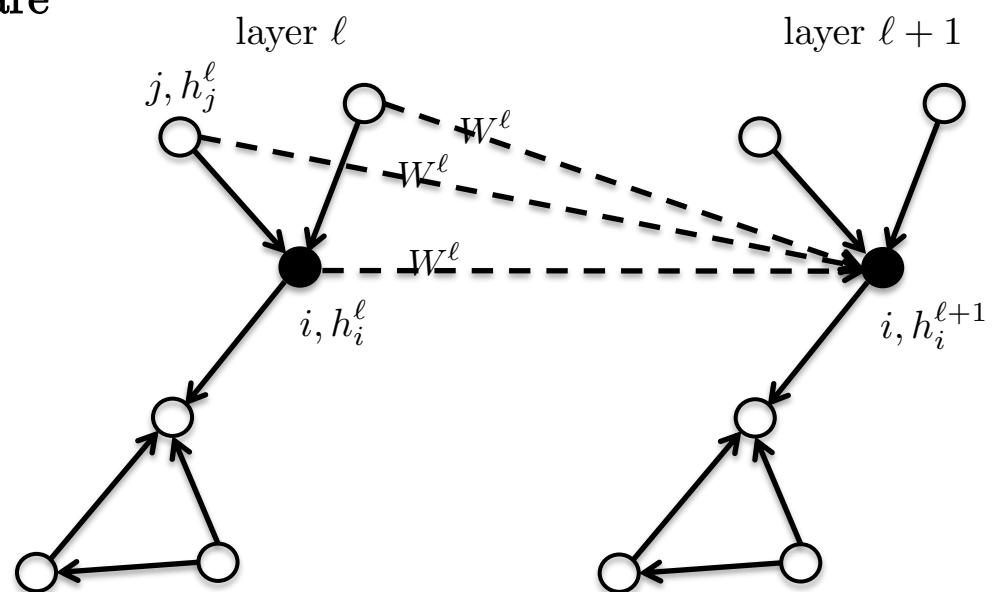
- Simplest formulation of spatial GCNs
 - Handle the absence of node ordering
 - Invariant by node re-parametrization
 - Deal with different neighborhood sizes
 - Local reception field by design (only neighbors are considered)
 - Weight sharing (convolution property)
 - Independent of graph size
 - Limited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

$$h^{\ell+1} = \eta(D^{-1} A h^\ell W^\ell)$$

$$h_i^{\ell+1} = \eta\left(\frac{1}{d_i} \underbrace{\sum_{j \in \mathcal{N}_i} A_{ij} W^\ell h_j^\ell}_{\text{scalar}}\right)$$

Mean

Matrix representation
Vectorial representation



[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

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

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

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

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

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

$$\begin{aligned} h^{\ell+1} &= \eta(w^\ell(T_0(\Delta) - T_1(\Delta))h^\ell) \\ &= \eta(w^\ell(I + D^{-1/2}AD^{-1/2})h^\ell) \quad \text{Operator with largest eigenvalue in } [0,2] \text{ may cause divergence.} \end{aligned}$$

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

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

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

with

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

[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]

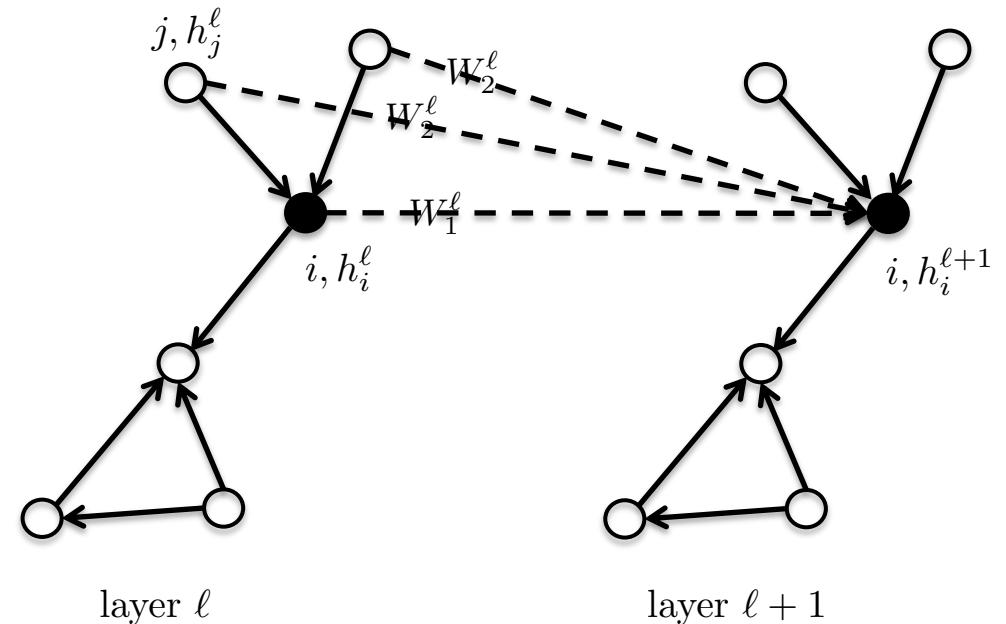
- Vanilla GCNs (supposing $A_{ij} = 1$) :
$$h_i^{\ell+1} = \eta\left(\frac{1}{d_i} \sum_{j \in \mathcal{N}_i} W^\ell h_j^\ell\right)$$
- GraphSage :
 - Differentiate template weights W^ℓ between neighbors h_j and central node h_i .
 - Isotropic GCNs

$$h_i^{\ell+1} = \eta\left(W_1^\ell h_i^\ell + \frac{1}{d_i} \underbrace{\sum_{j \in \mathcal{N}_i} W_2^\ell h_j^\ell}_{\text{Mean}_{j \in \mathcal{N}_i} W_2^\ell h_j^\ell} \right)$$

Or alternatively,

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

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



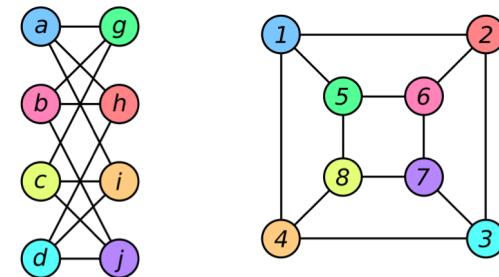
[1] W Hamilton, Z Ying, J Leskovec, Inductive representation learning on large graphs, 2017

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

$$\begin{aligned} h_i^{\ell+1} &= \text{ReLU}(W_2^\ell \text{ReLU}(\text{BN}(W_1^\ell \hat{h}_i^{\ell+1}))) \\ \hat{h}_i^{\ell+1} &= (1 + \epsilon) h_i^\ell + \sum_{j \in \mathcal{N}_i} h_j^\ell \end{aligned}$$

Batch
normalization



Example of two
isomorphic graphs

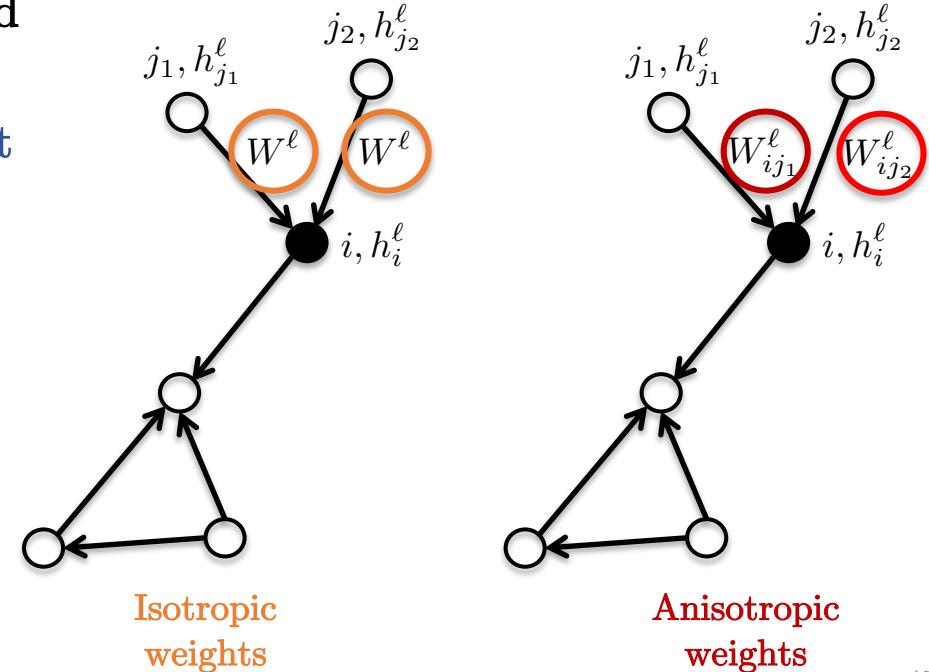
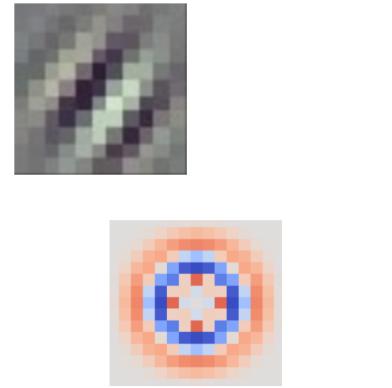
[1] K Xu, W Hu, J Leskovec, S Jegelka, How powerful are graph neural networks?, 2018

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

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.



[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

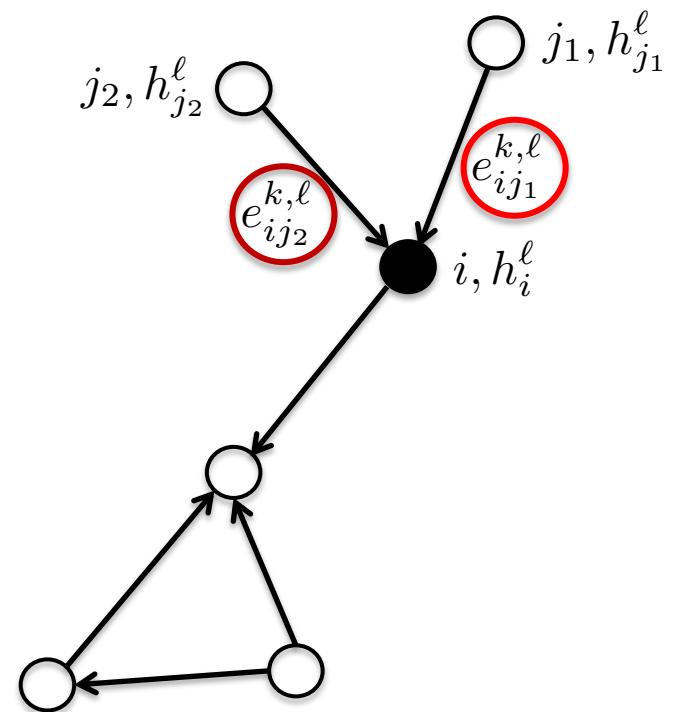
MoNets^[1]

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

$$h_i^{\ell+1} = \text{ReLU} \left(\sum_{k=1}^K \sum_{j \in \mathcal{N}_i} e_{ij}^{k,\ell} W_1^{k,\ell} h_j^\ell \right)$$

$$e_{ij}^{k,\ell} = \exp \left(-\frac{1}{2} (u_{ij}^\ell - \mu_k^\ell)^T (\Sigma_k^\ell)^{-1} (u_{ij}^\ell - \mu_k^\ell) \right)$$

$$u_{ij}^\ell = \text{Tanh} \left(A^\ell (\deg_i^{-1/2}, \deg_j^{-1/2}) + a^\ell \right)$$



[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

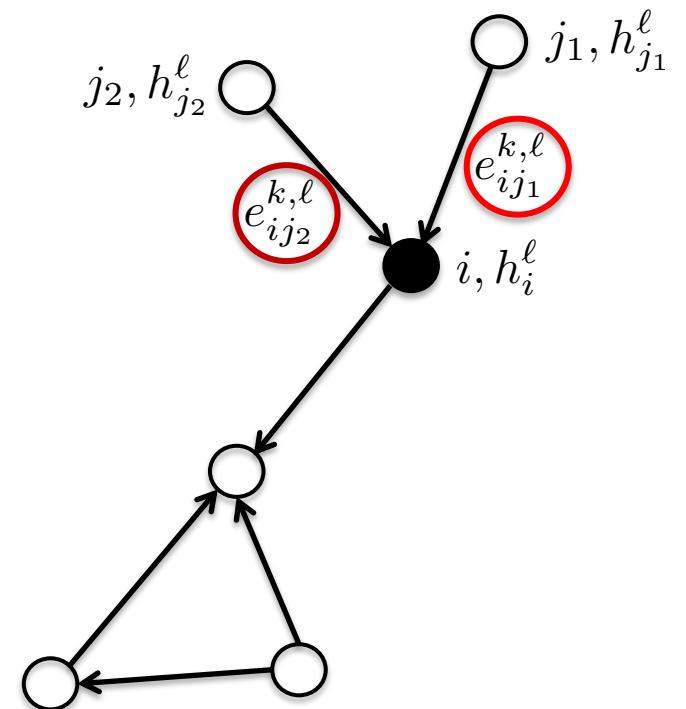
Graph Attention Networks^[1] (GAT)

- 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].

$$h_i^{\ell+1} = \text{Concat}_{k=1}^K \left(\text{ELU} \left(\sum_{j \in \mathcal{N}_i} e_{ij}^{k,\ell} W_1^{k,\ell} h_j^\ell \right) \right)$$

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

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



[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

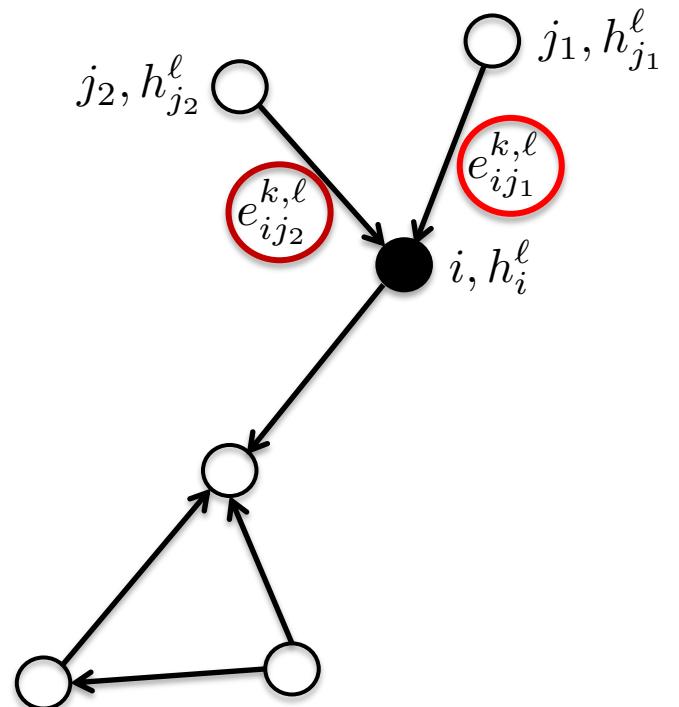
Gated Graph ConvNets^[1]

- 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.

$$h_i^{\ell+1} = h_i^\ell + \text{ReLU}\left(\text{BN}\left(W_1^\ell h_i^\ell + \sum_{j \in \mathcal{N}_i} e_{ij}^\ell \odot W_2^\ell h_j^\ell\right)\right)$$

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

$$\hat{e}_{ij}^\ell = \hat{e}_{ij}^{\ell-1} + \text{ReLU}\left(\text{BN}\left(V_1^\ell h_i^{\ell-1} + V_2^\ell h_j^{\ell-1} + V_3^\ell \hat{e}_{ij}^{\ell-1}\right)\right)$$



[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] :

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

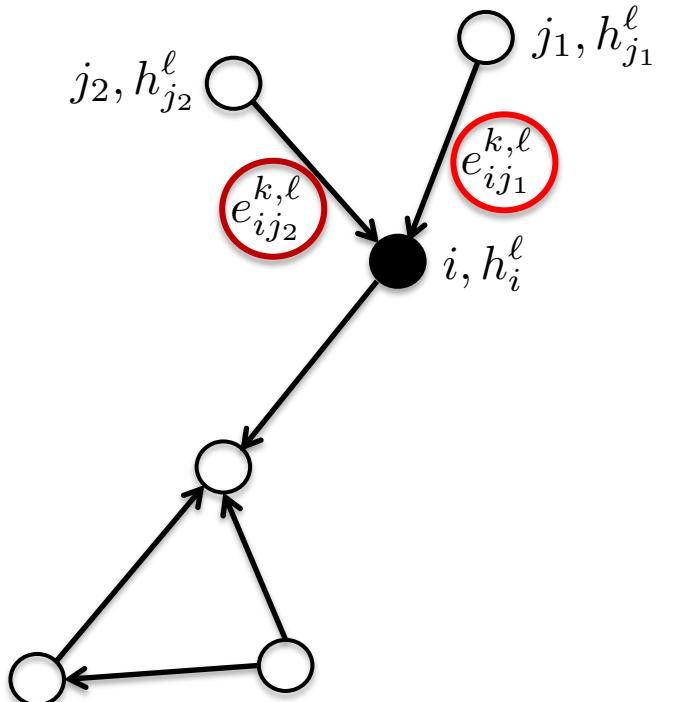
scalar

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

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

Query Key

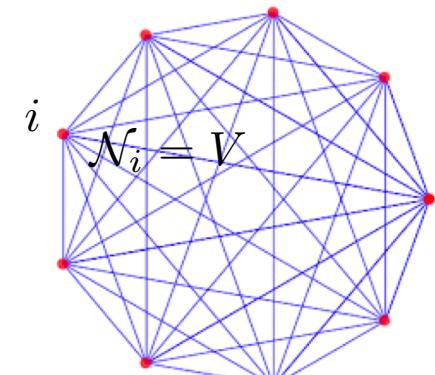
Attention mechanism in 1-hop neighborhood



[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

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

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

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

$$\mathcal{N}_i = V \implies$$

$$h^{\ell+1} = \text{Concat}_{k=1}^K \left(\underbrace{\text{Softmax}(Q^\ell K^{\ell T}) V^\ell}_{n \times \frac{d}{K}} \underbrace{W^\ell}_{\frac{d}{K} \times n} \right) \underbrace{W^\ell}_{n \times \frac{d}{K}}$$

$$Q^\ell = h^\ell \underbrace{W_Q^\ell}_{n \times \frac{d}{K}}$$

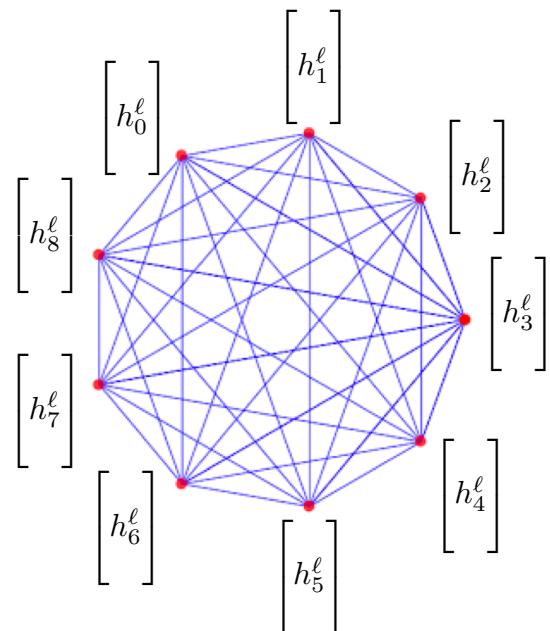
$$K^\ell = h^\ell \underbrace{W_K^\ell}_{\frac{d}{K} \times n}$$

$$V^\ell = h^\ell \underbrace{W_V^\ell}_{d \times \frac{d}{K}}$$

[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



$$\begin{bmatrix} h_0^\ell \\ h_1^\ell \\ h_2^\ell \\ h_3^\ell \\ h_4^\ell \\ h_5^\ell \\ h_6^\ell \\ h_7^\ell \\ h_8^\ell \end{bmatrix}$$

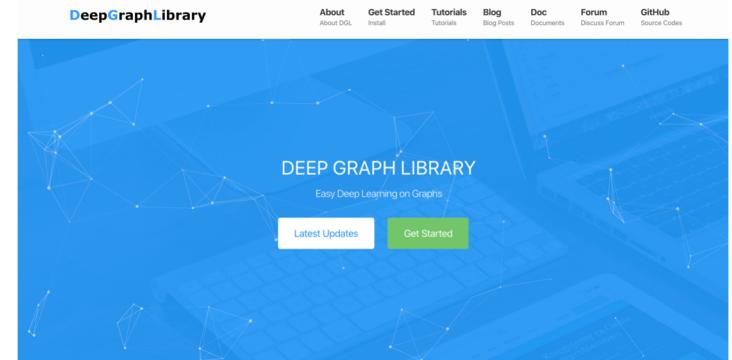
Sets of features

Outline

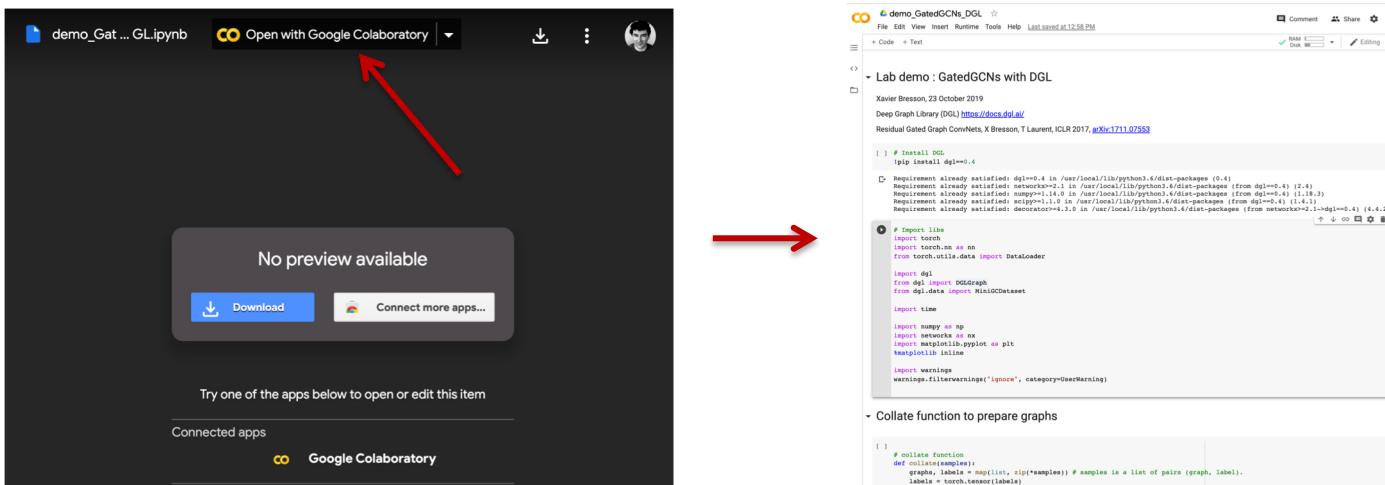
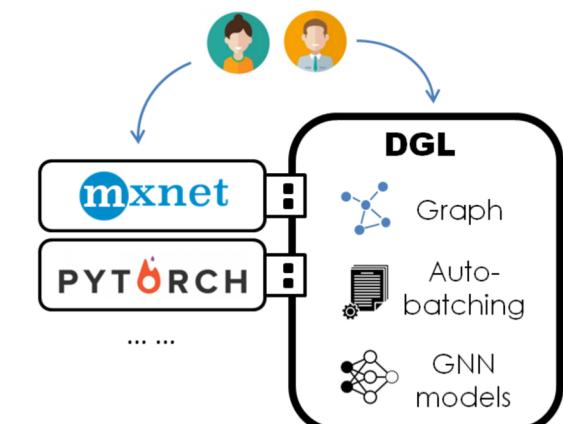
- Part 1: Traditional ConvNets
 - Architecture
 - Graph Domain
 - Convolution
- Part 2: Spectral Graph ConvNets
 - Spectral Convolution
 - Spectral GCNs
- Part 3: Spatial Graph ConvNets
 - Template Matching
 - Isotropic GCNs
 - Anisotropic GCNs
 - **Lab on GatedGCNs**
- Part 4: GNN Tasks
- Conclusion

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

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

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

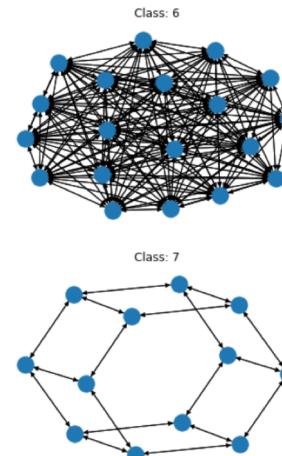
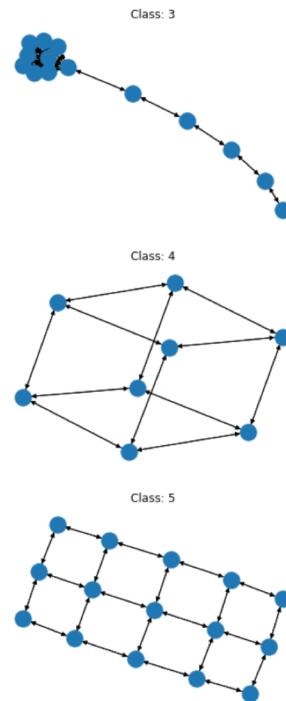
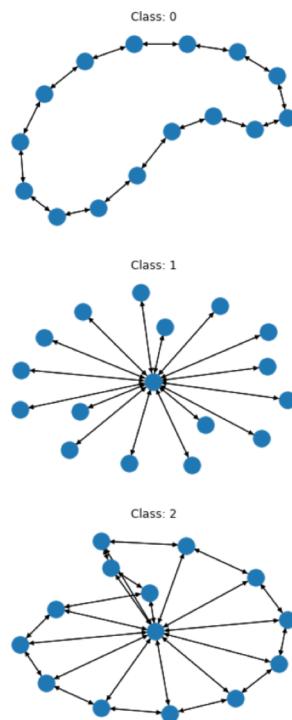
(DGLGraph(num_nodes=13, num_edges=39,
          ndata_schemes={'feat': Scheme(shape=(1,), dtype=torch.float32)},
          edata_schemes={'feat': Scheme(shape=(1,), dtype=torch.float32)}), 0)
```

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

Graph normalization
constants

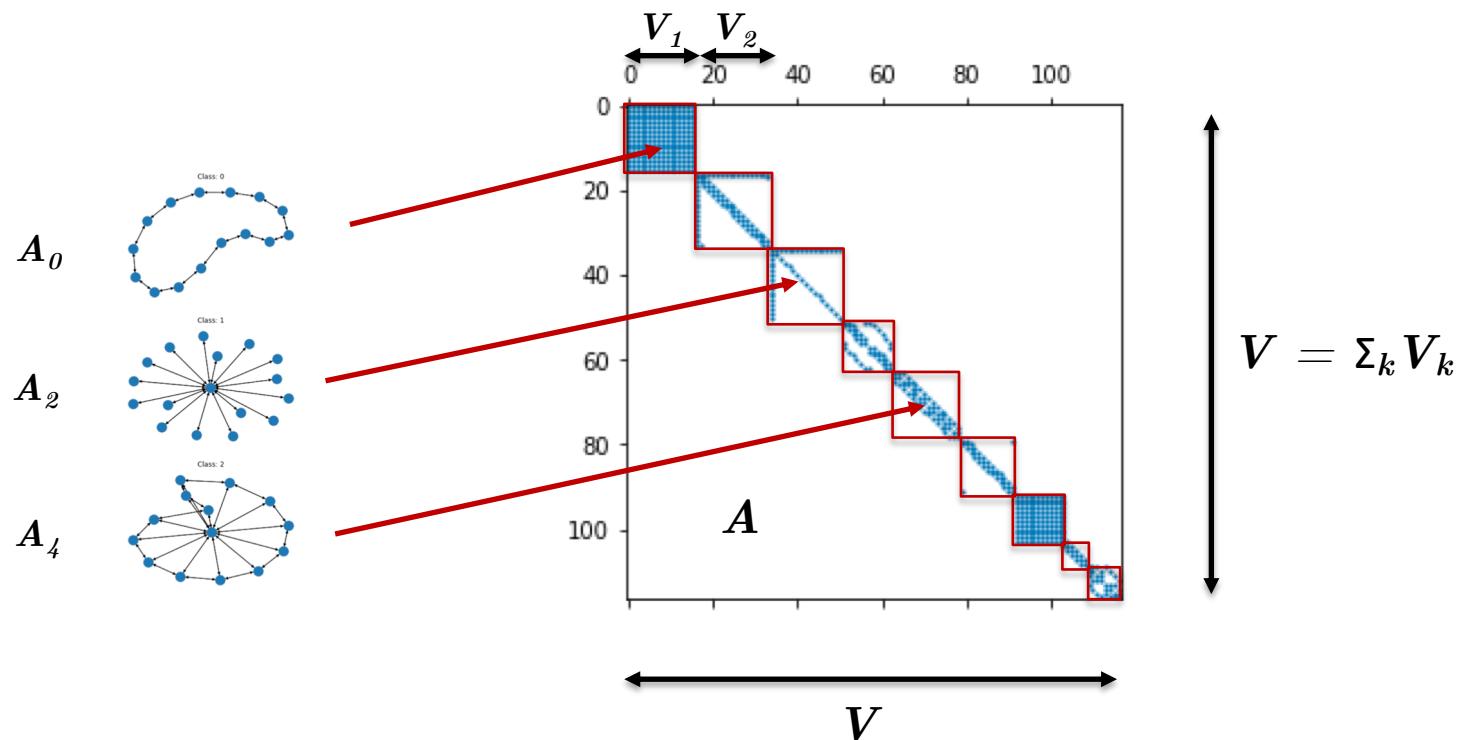
Lab on GatedGCNs

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



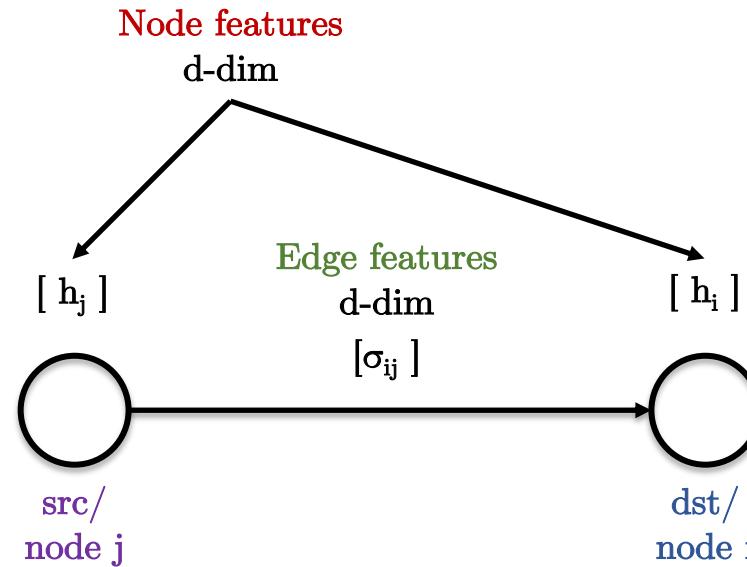
Lab on GatedGCNs

- Understanding DGL :
 - How to process K graphs of different sizes ?
Form a (big) sparse block diagonal matrix A with K adjacency matrices A_k .



Lab on GatedGCNs

- Basic structure of an **edge** in DGL :

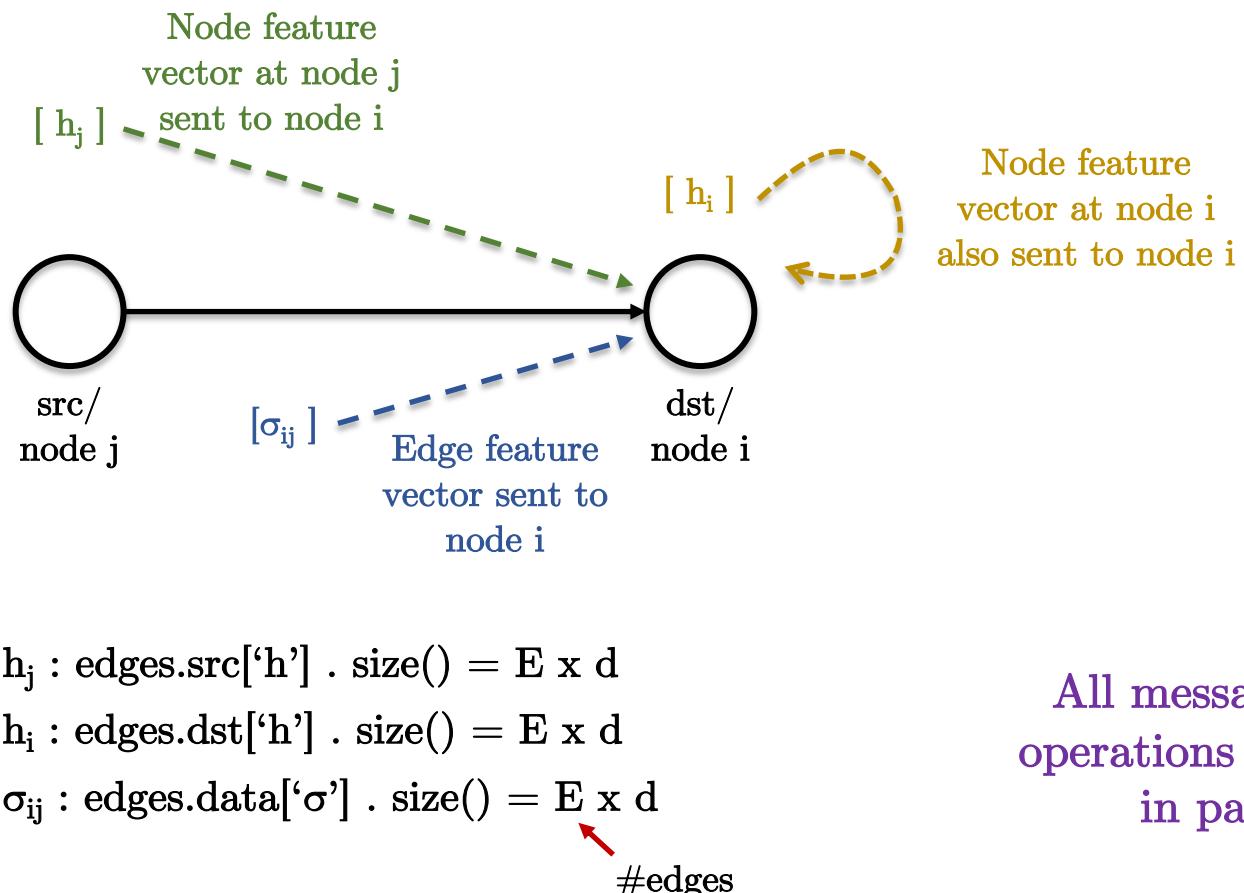


- Goal is to **compute efficiently** expressions of the **form** :

$$f_i = h_i + \sum_{j \rightarrow i} \sigma_{ij} \circ h_j$$

Lab on GatedGCNs

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



Lab on GatedGCNs

- Step 2 : Reduce function defined on nodes

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

Sum over neighbors

- GPU acceleration :

- DGL batches the nodes with the same number of neighbors.

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

#nodes in batch₁
#neighbors

$$\sigma_{ij} = \text{nodes.mailbox}[\sigma_{ij}] = \quad \text{same structure than } h_j$$

$$h_i = \text{nodes.data}[h] . \text{size}() = V \times d$$

Lab on GatedGCNs

GatedGCN layer :

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

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

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

```

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 = Ah_i + 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)
        gedata['e'] = e
        gedata['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

```
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

```
# 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
```

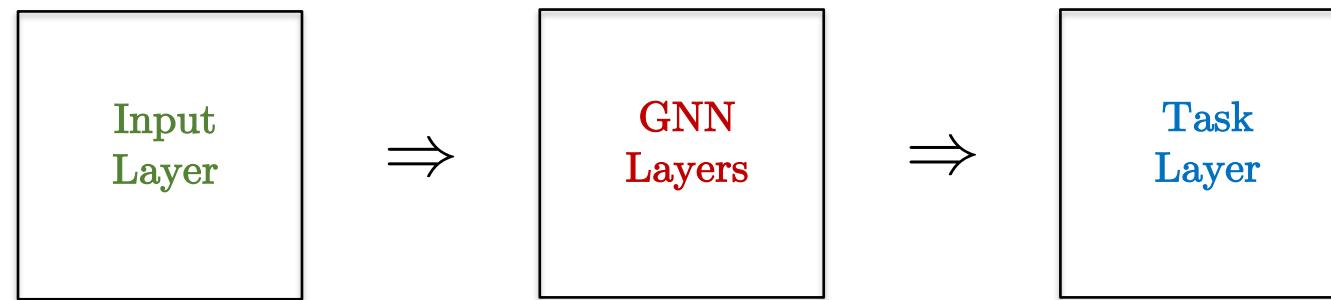
Main function

Outline

- 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

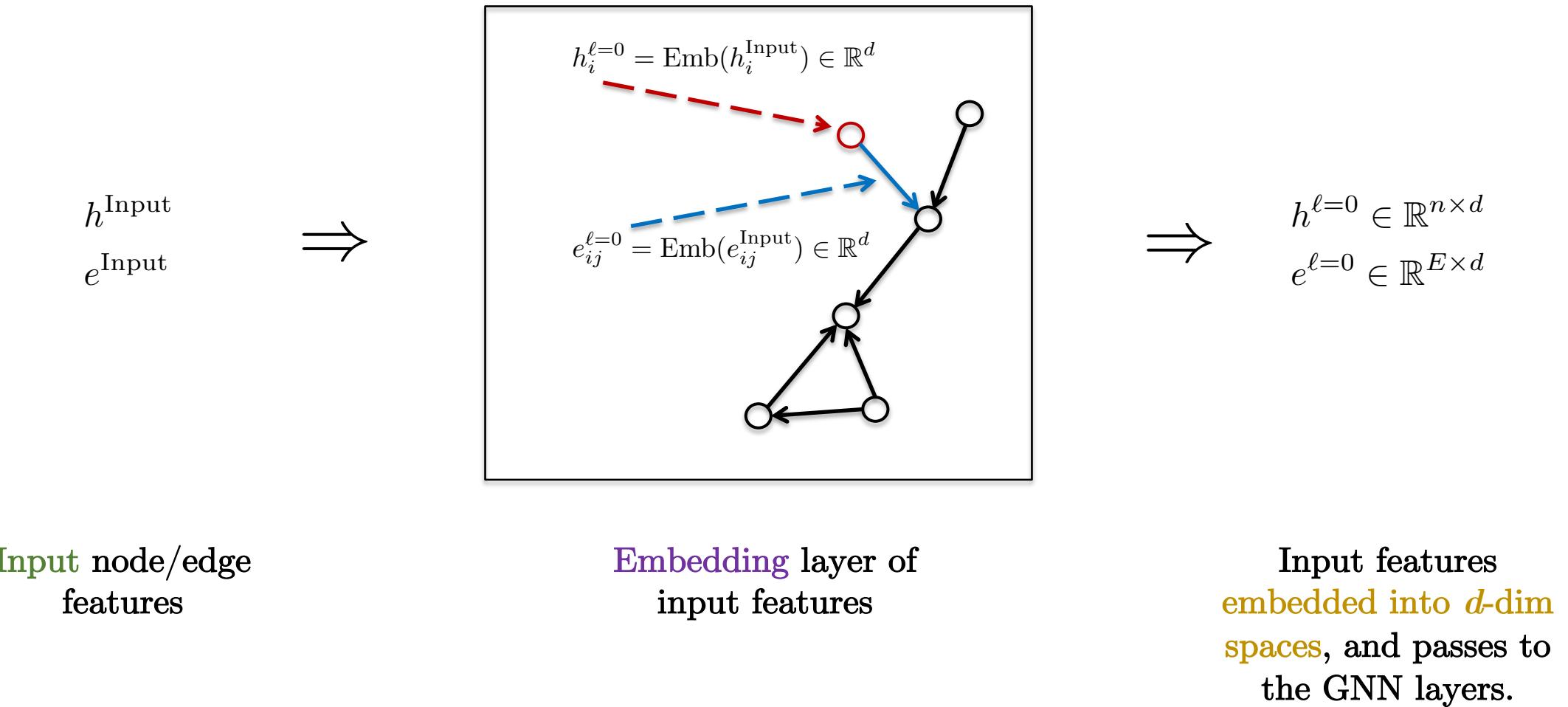
GNN Pipeline

- 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

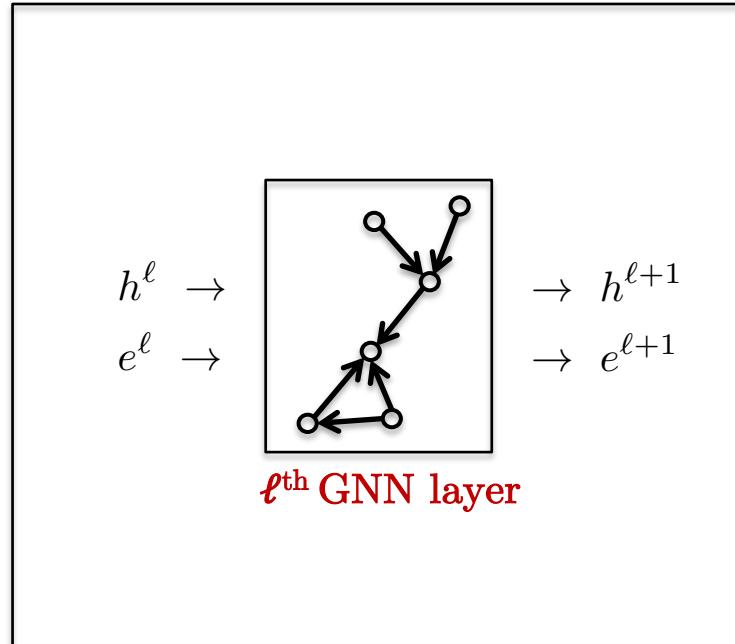
- Input layer :



GNN Pipeline

- GNN layers :

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



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

GNN layers applied
 L times

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

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

GNN Tasks

- Task-based layer

- Graph-level prediction :

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

- Node-level prediction :

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

- Edge-level prediction :

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

Outline

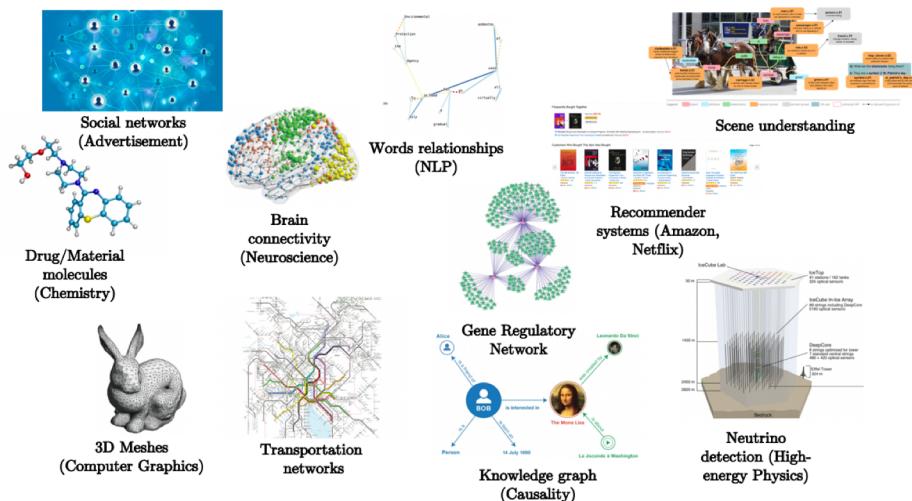
- Part 1: Traditional ConvNets
 - Architecture
 - Graph Domain
 - Convolution
- Part 2: Spectral Graph ConvNets
 - Spectral Convolution
 - Spectral GCNs
- Part 3: Spatial Graph ConvNets
 - Template Matching
 - Isotropic GCNs
 - Anisotropic GCNs
 - Lab on GatedGCNs
- Part 4: GNN Tasks
- Conclusion

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 :



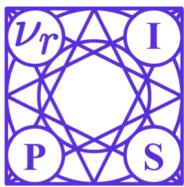
"Graphs are the most important discrete models in the world!" -
G. Strang (MIT)



Tutorials on Graph Deep Learning



SIAM Annual Meeting'18



NeurIPS'17
1,000-2,000 participants

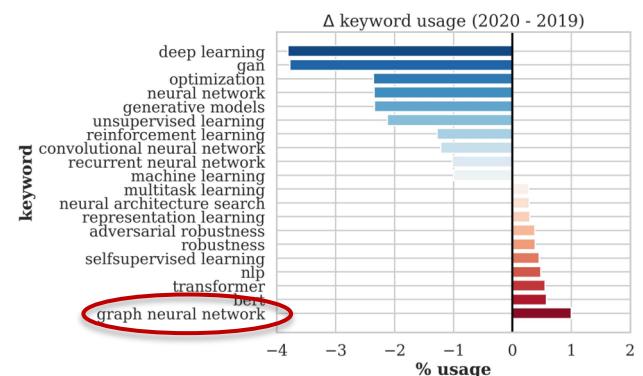
Organizers

- 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)



CVPR'17
500-1,000 participants

- 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)

The screenshot shows a workshop page titled 'Workshops' under 'Programs'. The specific workshop is 'New Deep Learning Techniques' held from February 5-9, 2018. The page includes tabs for 'OVERVIEW', 'SPEAKER LIST', 'SCHEDULE', and 'APPLY/REGISTER'. The 'OVERVIEW' section contains a brief description of deep learning's impact on various fields like computer vision, speech recognition, and natural language processing. It also mentions challenges related to irregular domains and heterogenous data. A small image of a colorful, abstract network or graph is shown on the right.

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)
Stéphane Mallat (École Normale Supérieure)
Federico Monti (Università della Svizzera Italiana)
Stanley Osher (University of California, Los Angeles (UCLA), Mathematics)
Elie 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)

The screenshot shows the IPAM-UCLA workshop website. At the top, there's a dark header with the word 'Workshops'. Below it, a banner for 'Workshop IV: Deep Geometric Learning of Big Data and Applications' is displayed, featuring a blue background with abstract geometric shapes like circles and lines. The main content area has a white background. It includes a navigation bar with tabs for 'OVERVIEW', 'SPEAKER LIST', 'SCHEDULE', and 'APPLY/REGISTER'. The 'OVERVIEW' tab is active. The page title is 'Workshop IV: Deep Geometric Learning of Big Data and Applications' and it is described as 'Part of the Long Program Geometry and Learning from Data in 3D and Beyond'. The date 'MAY 20 - 24, 2019' is listed. The 'OVERVIEW' section contains text about deep learning techniques and their applications in various fields. To the right of the text is a small image of a complex, colorful 3D point cloud or network visualization.

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>